# Objektorientierte Modellierung

*Prof. Dr. Roland Dietrich*

## 5. Dynamische Modellierung mit UML

- Aktivitätsdiagramme
- Interaktionsdiagramme
  - Sequenzdiagramme
  - Kommunikationsdiagramme
- Zustandsautomaten

# Übersicht



1. Objektorientierte Softwareentwicklung ✓
2. Anforderungsanalyse mit UML ✓
   - Anwendungsfalldiagramme
3. Statische Modellierung mit UML ✓
   - Klassendiagramme
     Objekte und Klassen, Assoziationen, Vererbung
   - Paketdiagramme
4. Der Analyseprozess und Analysemuster ✓
5. **Dynamische Modellierung mit UML**
   - Aktivitätsdiagramme
   - Interaktionsdiagramme (Sequenz- und Kommunikationsdiagramme)
   - Zustandsautomaten
6. Entwurf mit UML
7. Implementierung in C++

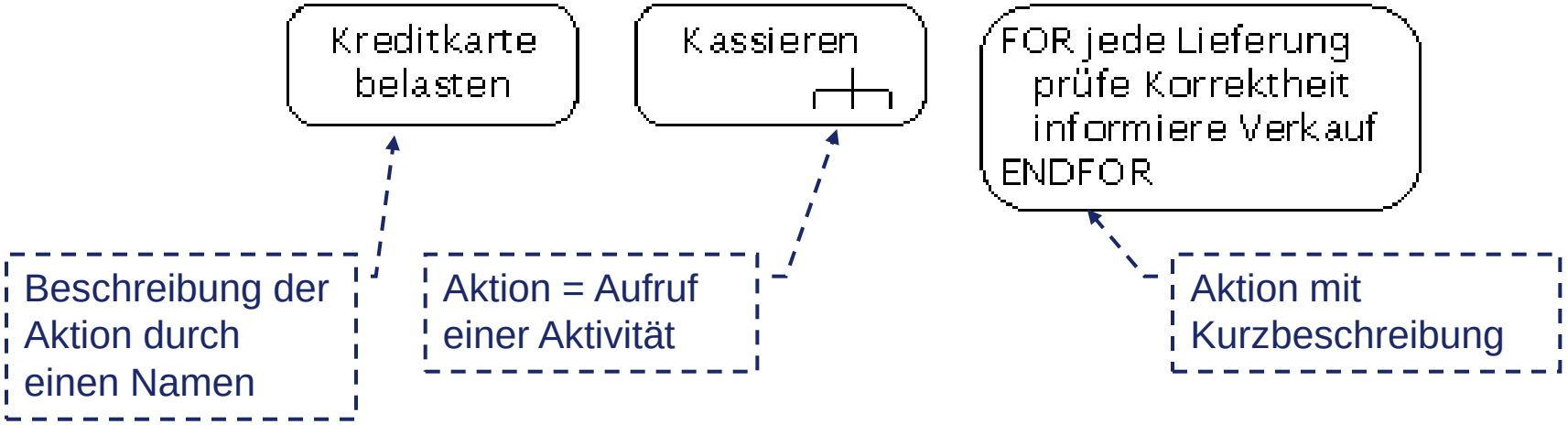
# Aktivitätsdiagramme



- **Definition**
  - Eine **Aktivität** *(activity)* beschreibt die Ausführung von Funktionalität
  - Sie wird durch mehrere **Knoten** (Aktionsknoten, Kontrollknoten, Objektknoten) modelliert, die durch **gerichtete Kanten** miteinander verbunden sind
  - Besitzt ein **Kontrollflussmodell**
    - Reihenfolge von Aktionen
  - Besitzt ein **Datenmodell**
    - Daten, die zwischen Aktionen ausgetauscht werden
  - Anwendung in der Analyse:
    - Beschreibung von Anwendungsfällen
    - Modellierung von Arbeitsabläufen („*workflows*“)
  - Anwendung im Entwurf
    - Beschreibung der Implementierung von Operationen (Algorithmen)

# Aktivitätsdiagramme

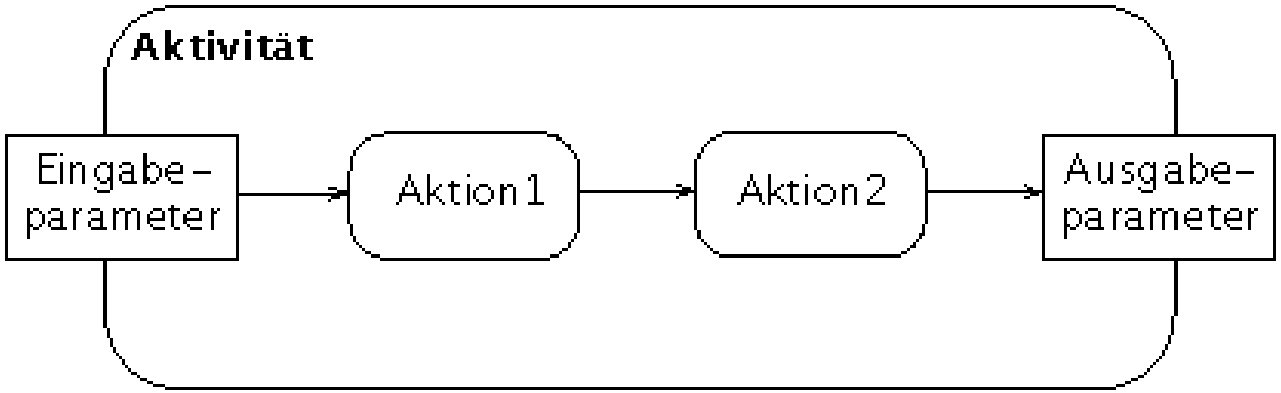
- **Aktionsknoten**
  - Kleinste ausführbare Einheit in einer Aktivität
  - Wird ausgeführt, wenn
    - Vorgängeraktion beendet ist
    - Daten zur Verfügung stehen oder
    - Ereignis auftritt
  - Notation:




Quelle: [Balzert 05], Abb. 2.9-1

# Aktivitätsdiagramme

- **Notation Aktivität**
  - Abgerundetes Rechteck mit Namen
  - Im Rechteck: Graph mit Knoten und gerichteten Kanten (Pfeile)
    - Beschreiben Kontroll- und Datenfluss
  - Auf dem Rand des Rechtecks: Ein-/Ausgabe-Parameter

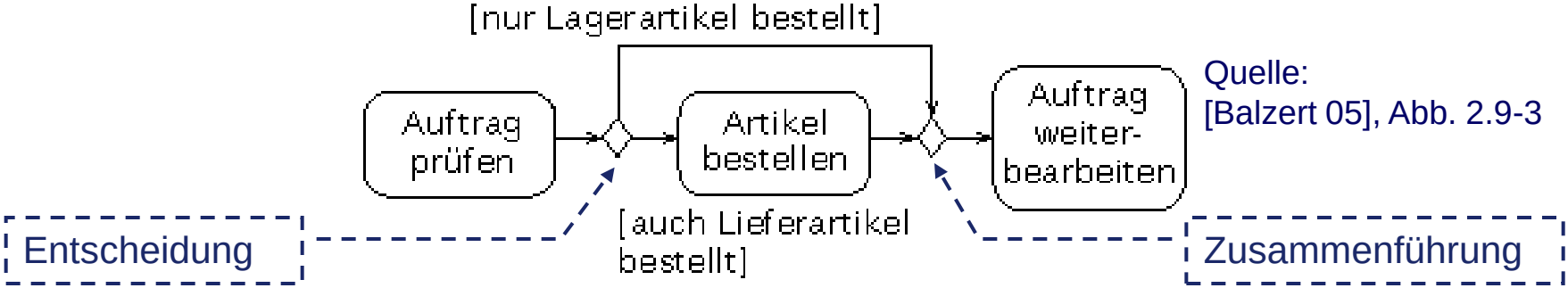



Quelle: [Balzert 05], Abb. 2.9-2

- **Aktivitätsdiagramm**
  - Enthält eine oder mehrere Aktivitäten
  - Häufig nur eine Aktivität
    - Die beschriebene Aktivität selbst wird dann oft nicht dargestellt (Aktivitätsdiagramm = Aktivität)

# Aktivitätsdiagramme

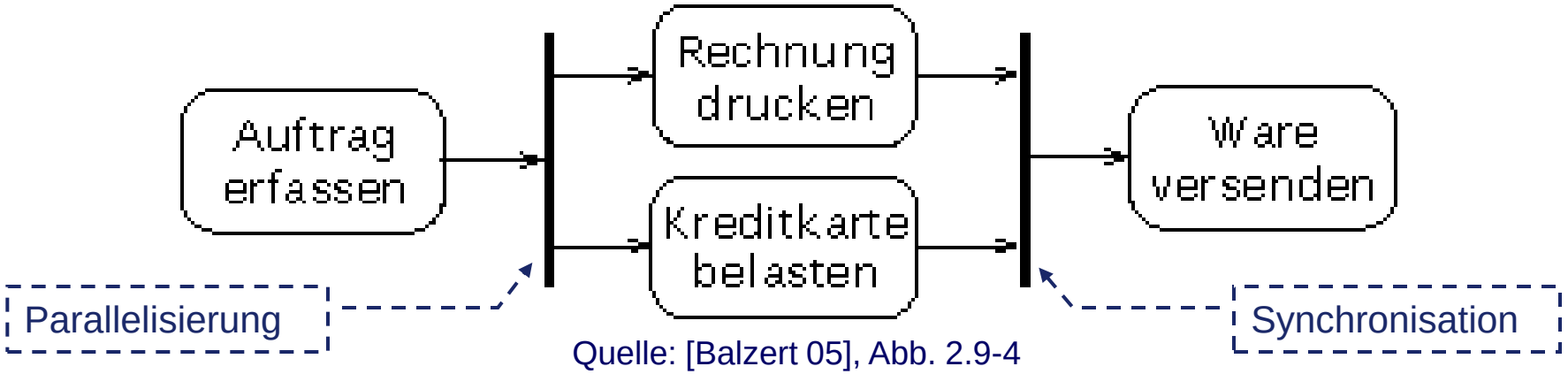
- **Kontrollknoten**
  - **Entscheidung** (Verzweigung, *decision node*)
    - Eine eingehende Kante → mehrere **alternative** ausgehende Kanten
    - Die ausgehenden Kanten sind mit Bedingungen […] markiert
      - Die Kante, deren Bedingung true ergibt, wird durchlaufen
      - Bei der Entscheidung darf nur eine (1!) Alternative zutreffen
      - Keine Beschriftung einer ausgehenden Kante bedeutet [true]
  - **Zusammenführung** (Verbindung, *merge node*)
    - Mehrere **alternative** eingehende Kanten → eine ausgehende Kante
    - Die ausgehende Kante wird durchlaufen, sobald die Zusammenführung erreicht wird.
  - Zusammenführung und Entscheidung können in einer Raute kombiniert werden




Quelle:
[Balzert 05], Abb. 2.9-3

# Aktivitätsdiagramme

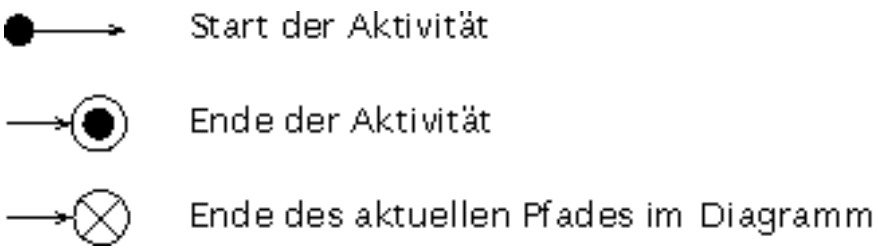
- Kontrollknoten
  - **Parallelisierung** *(splitting, fork node)*
    - eine eingehende Kante → mehrere **parallele** ausgehende Kanten
    - Die Reihenfolge, in der die ausgehenden Kanten abgearbeitet werden, spiel keine Rolle: **parallel** bzw. **nebenläufig** (*concurrent)*
  - **Synchronisation** *(join node)*
    - mehrere **parallele** eingehende Kanten → eine ausgehende Kante
    - Die ausgehende Kante wird durchlaufen, wenn alle eingehenden Pfade durchlaufen sind und an der Synchronisation „ankommen“.
  - Splitting und Synchronisation können in einem Balken kombiniert werden




Quelle: [Balzert 05], Abb. 2.9-4

# Aktivitätsdiagramme

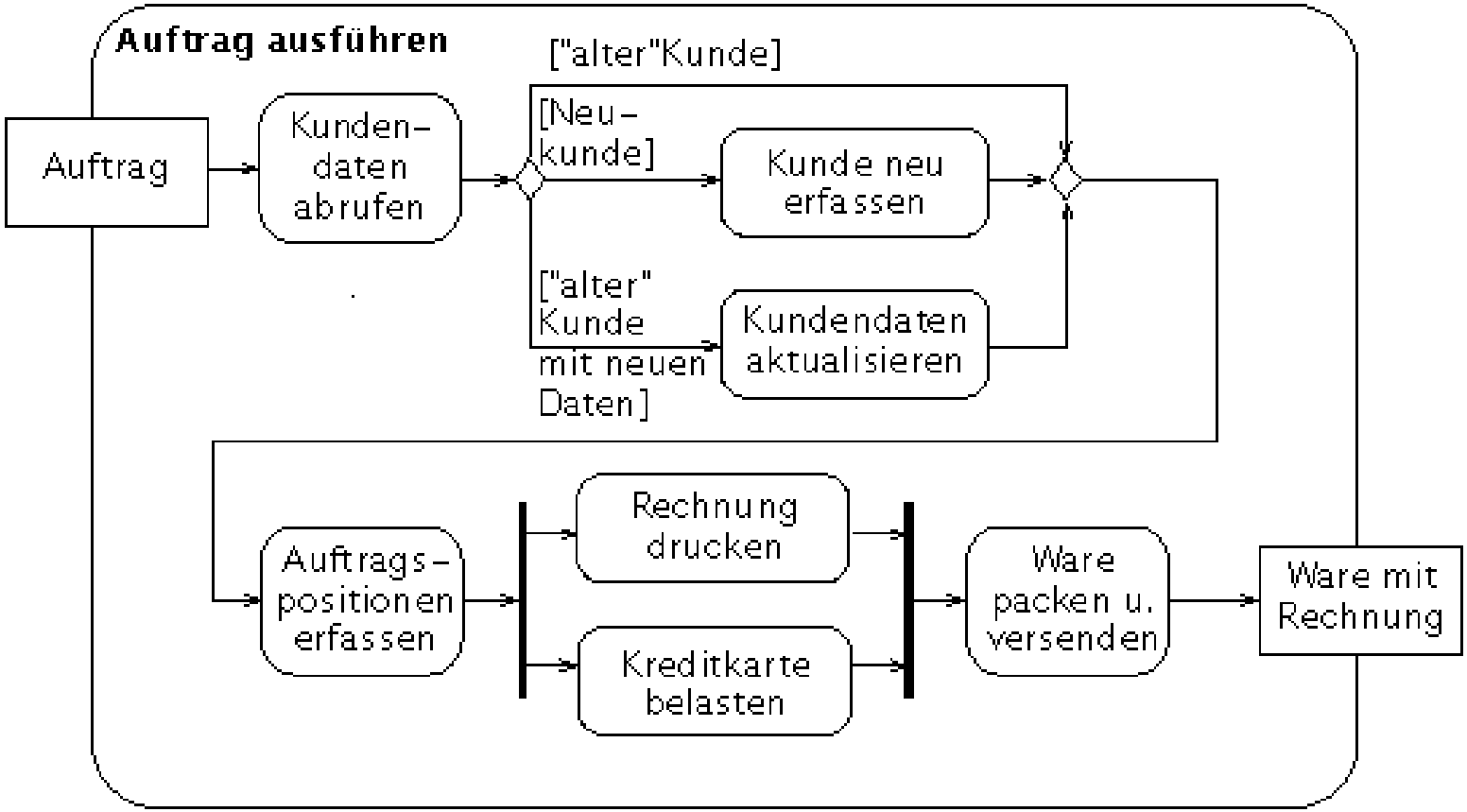
- Kotrollknoten
  - **Startknoten**
    - Startknoten markiert Aktion, die zuerst ausgeführt wird
    - Eine Aktivität kann mehrere Startknoten besitzen (startet dann parallel in mehreren Pfaden)
    - Startknoten kann entfallen (Aktivität startet dann, wenn Eingabedaten verfügbar sind)
  - **Endknoten**
    - Wird der Endknoten *(final node)* erreicht, dann werden sämtliche Aktionen der Aktivität sofort beendet
    - Wird der Endknoten für Kontrollflüsse erreicht *(final flow)*, dann endet nur der aktuelle Pfad




Quelle: [Balzert 05], Abb. 2.9-5

# Aktivitätsdiagramme

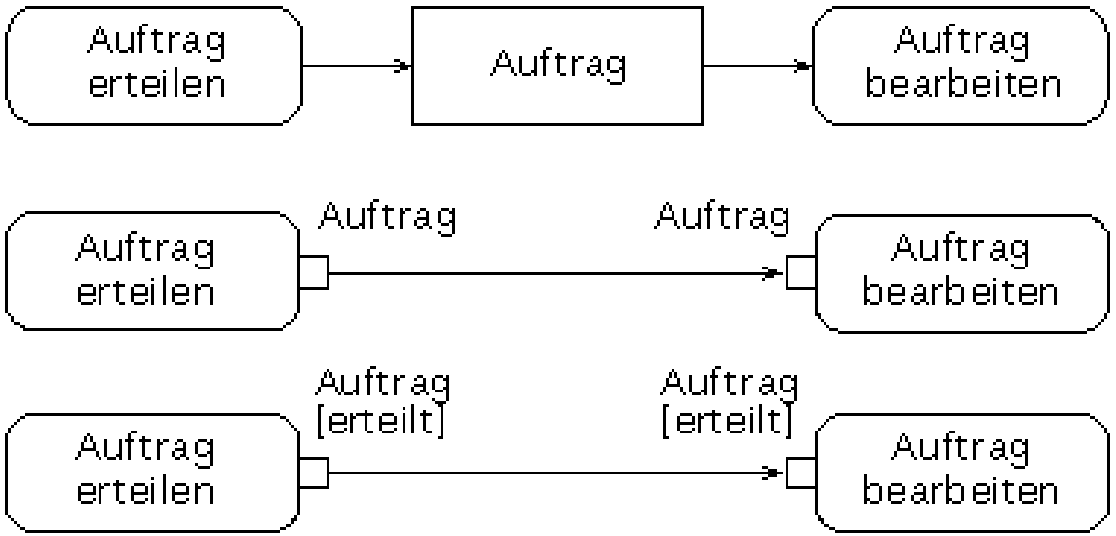
- **Beispiel**
  - Ausführen eines Auftrags (Warenhaus)




Quelle: [Balzert 05], Abb. 2.9-6

# Aktivitätsdiagramme

- **Objektknoten**
  - Beschreiben den Datenfluss („**Objektfluss**“) innerhalb einer Aktivität
  - Repräsentieren Instanzen eines bestimmten Typs
    - z.B. (primitiven) Datentypen, Klassen
  - Notation:
    - Rechteck, häufig mit dem Namen der Klasse/des Typs benannt
  - Alternativ kann die **Pin-Notation** verwendet werden
    - [*Bedingung*] beschreibt den Zustand des Objekts




Quelle: [Balzert 05], Abb. 2.9-7

Objekte der Klasse *Auftrag* „**fließen**“ von der Aktion *Auftrag erteilen* zur Aktion *Auftrag bearbeiten.*

Sie sind **Ausgabe** von *Auftrag erteilen* und **Eingabe** von *Auftrag bearbeiten*.

# Aktivitätsdiagramme

- Objektknoten
  - **Parameter**
    - Objektknoten auf der „Grenze“ einer Aktivität (vgl. S. 8)
      - Eingabeparameter: nur ausgehende Pfeile
      - Ausgabeparameter: nur eingehende Pfeile
  - **Datenspeicher**
    - Sonderform für persistente Daten
    - Notation: Objektknoten mit dem Schlüsselwort **«datastore»**
    - Eingehende Objekte werden permanent gespeichert
    - Alle gespeicherten Objekte stehen an ausgehenden Kanten zur Verfügung

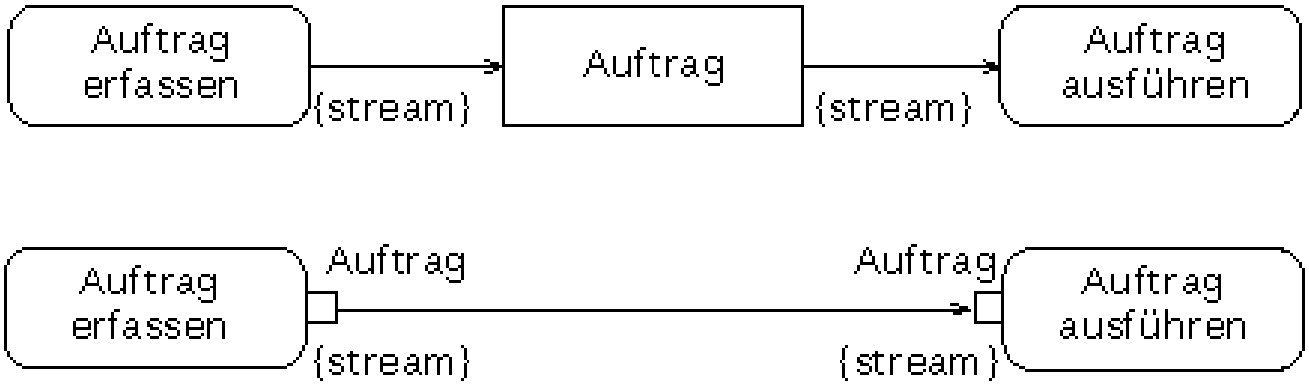

Quelle: [Balzert 05], Abb. 2.9-8

Die Aktion *Artikel ändern/erfassen* speichert seine Ausgaben in der Artikeldatenbank.

Alle Objekte der Artikeldatenbank stehen der Aktion *Artikelkatalog erstellen* als Eingabe zur Verfügung

# Aktivitätsdiagramme

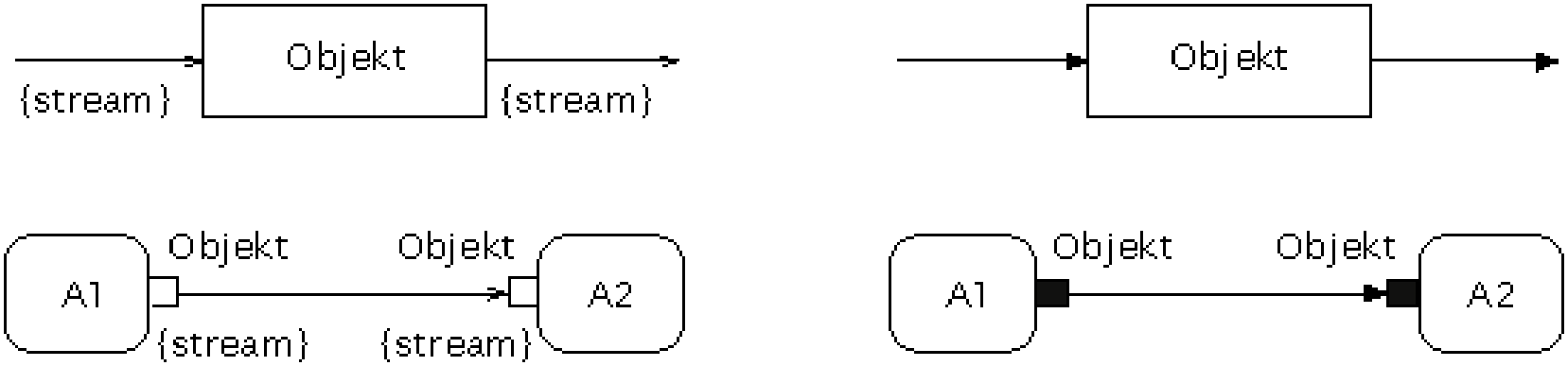
- Objektknoten
  - **Streaming-Modus**
    - Daten werden von einer Aktion kontinuierlich zur Verfügung gestellt und werden von einer Aktion kontinuierlich verbraucht
      - Aktion beginnt, sobald ein Objekt vorliegt
      - Verarbeitung weiterer Objekte, sobald sie vorliegen
    - Notation
      - **{stream}** am Eingabe bzw. Ausgabe-Pin, oder
      - **{stream}** an den Kanten des Objektknotens




Quelle: [Balzert 05], Abb. 2.9-9

# Aktivitätsdiagramme

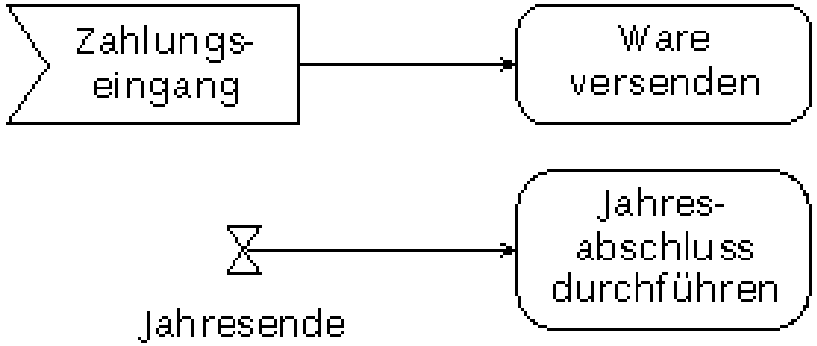
- Objektknoten
  - Streaming-Modus – alternative Notation
    - ausgefüllte Pfeilspitzen
    - ausgefüllte Ein-/Ausgabe-Pins




Quelle: [Balzert 05], Abb. 2.9-10

# Aktivitätsdiagramme

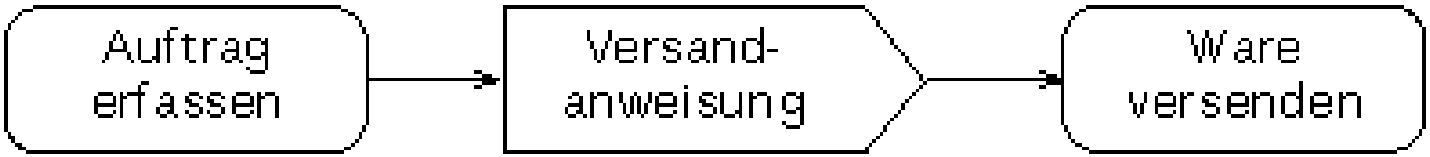
- **Ereignisse**
  - Ereignisse können innerhalb oder außerhalb des Systems auftreten
  - **Ereignisempfänger**
    - Aktion, die auf das Eintreten eines Ereignisses wartet *(accept event action)*
    - Darstellung durch konkaves Fünfeck
  - **Zeitereignisse**
    - *accept time event actions*
    - Werden durch Sanduhr symbolisiert




Quelle: [Balzert 05], Abb. 2.9-11

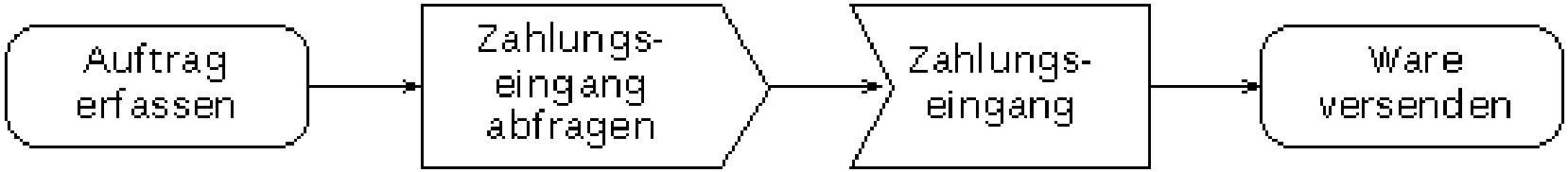
# Aktivitätsdiagramme

- Ereignisse
  - **Signalsender**
    - Aktion, die Signal an ein Zielobjekt sendet *(signal send action)*
    - Darstellung durch ein konvexes Fünfeck



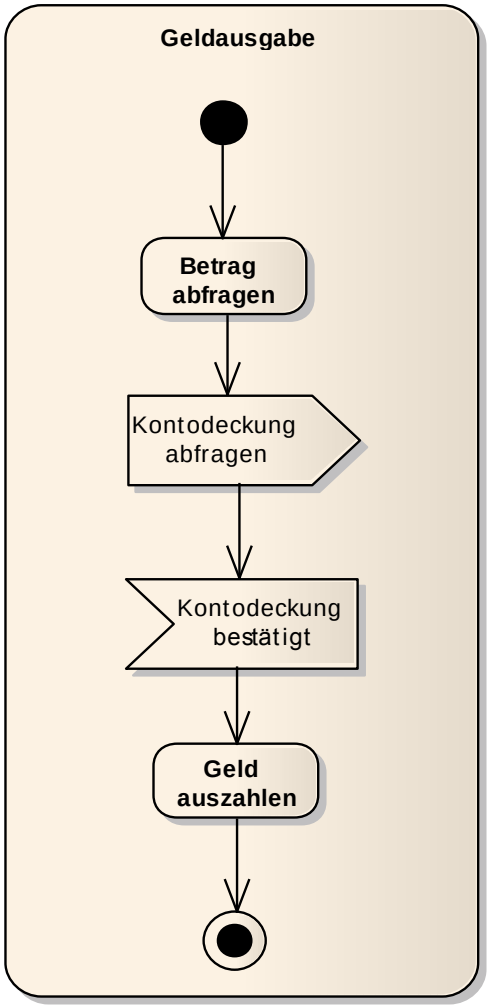

Quelle: [Balzert 05], Abb. 2.9-12

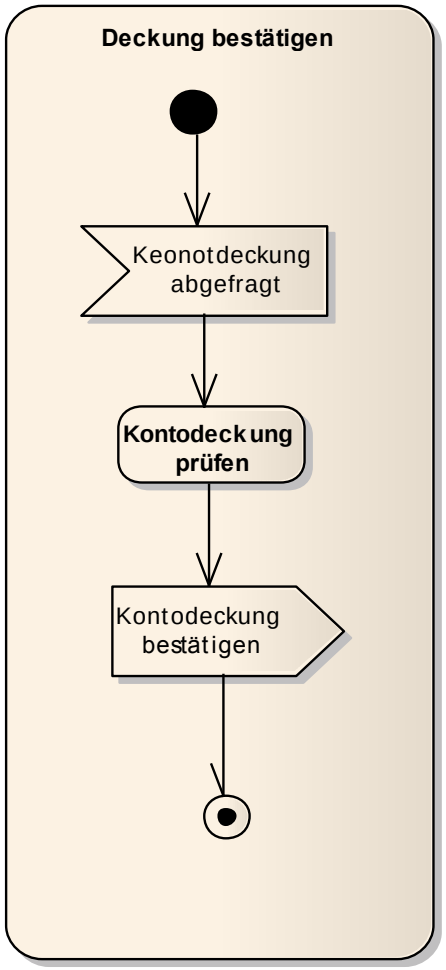
  - Häufig wird eine Kombination von Signalsender und Ereignisempfänger verwendet




Quelle: [Balzert 05], Abb. 2.9-13

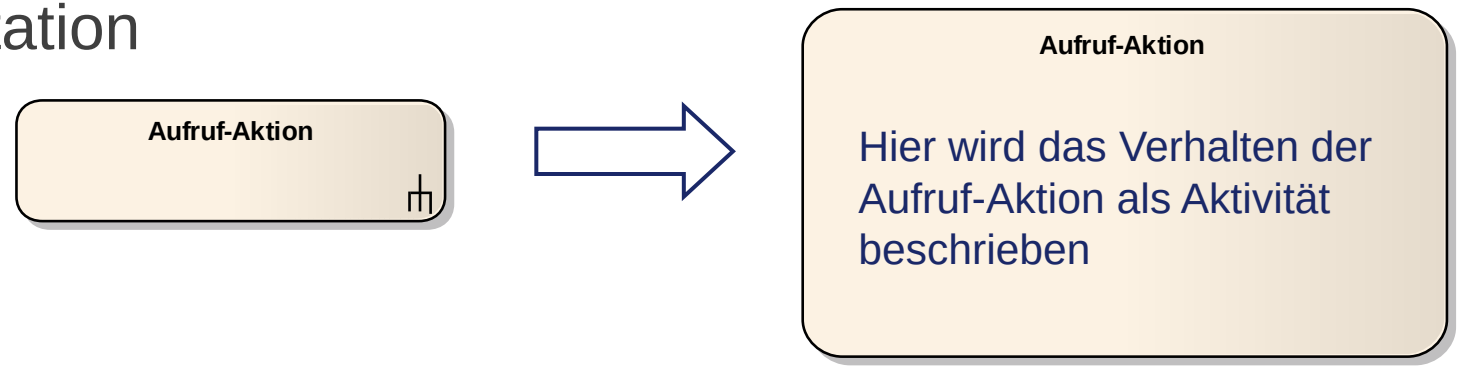
- **Beispiel**
  - Verwendung von Signalsendern und Ereignisempfängern



# Aktivitätsdiagramme

- **Schachtelung von Aktivitäten**
  - Durch Aktionen mit Aufruf-Verhalten (***Call Behaviour Action***)
  - Die Aktion wird separat durch eine Aktivität mit detailliertem Verhalten beschrieben
- Notation



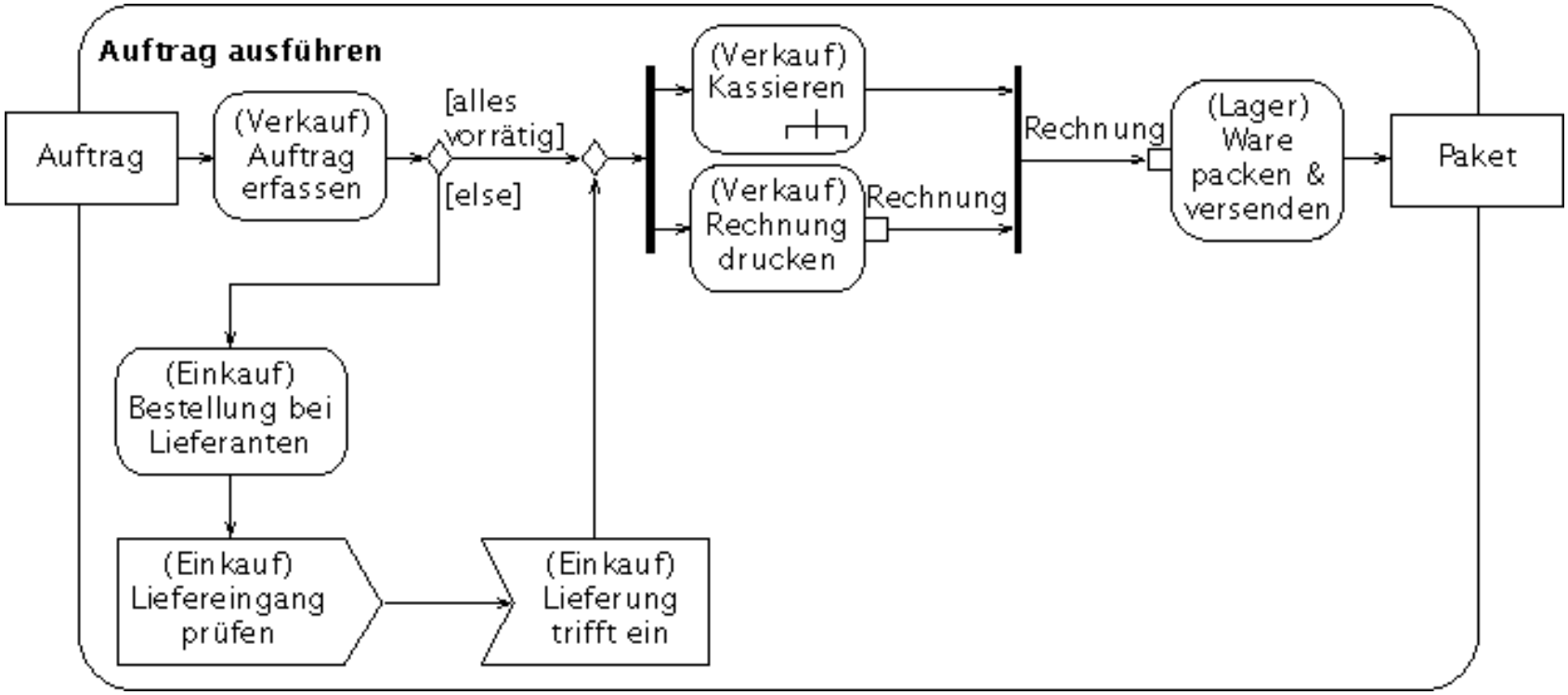

  - Notationsvarianten für den Namen beim Aktivitätsaufruf:

    *aufrufname: Aktivitätsname*

    falls mehrere Aufrufe derselben Aktivität unterschieden werden sollen
  - Ein-/Ausgabepins der Aufrufaktion müssen mit den Ein-/Ausgabe-parametern der aufgerufenen Aktivität korrespondieren

# Aktivitätsdiagramme

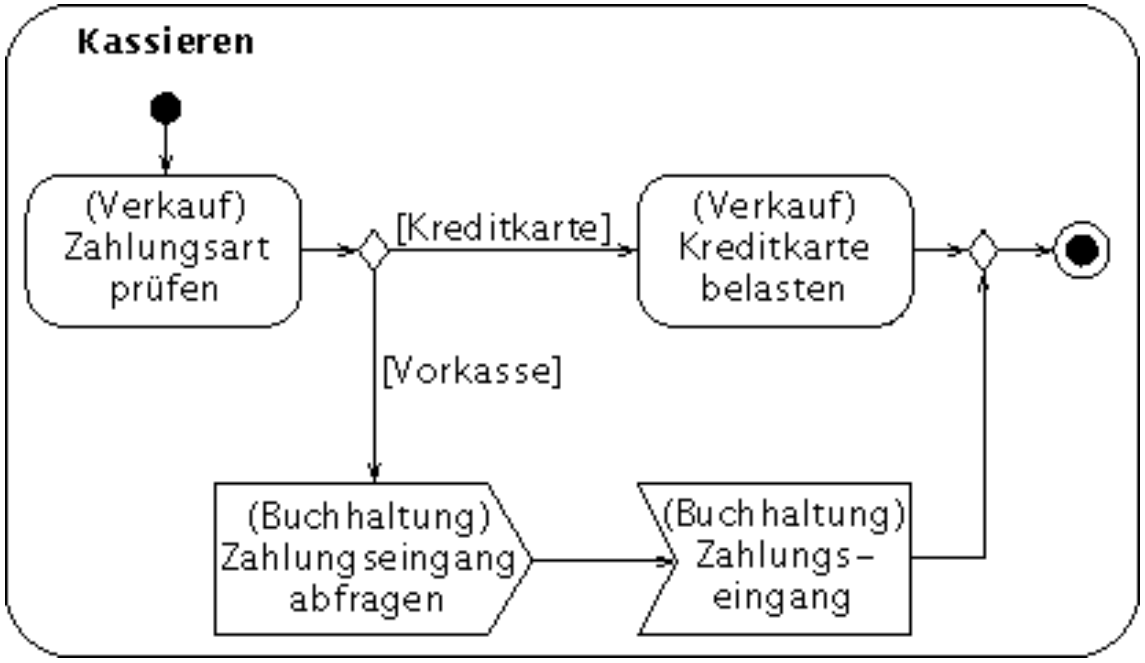
- **Beispiel**
  - Ausführung eines Auftrags mit Aktivitätsaufruf




Quelle: [Balzert 05], Abb. 2.9-17

# Aktivitätsdiagramme

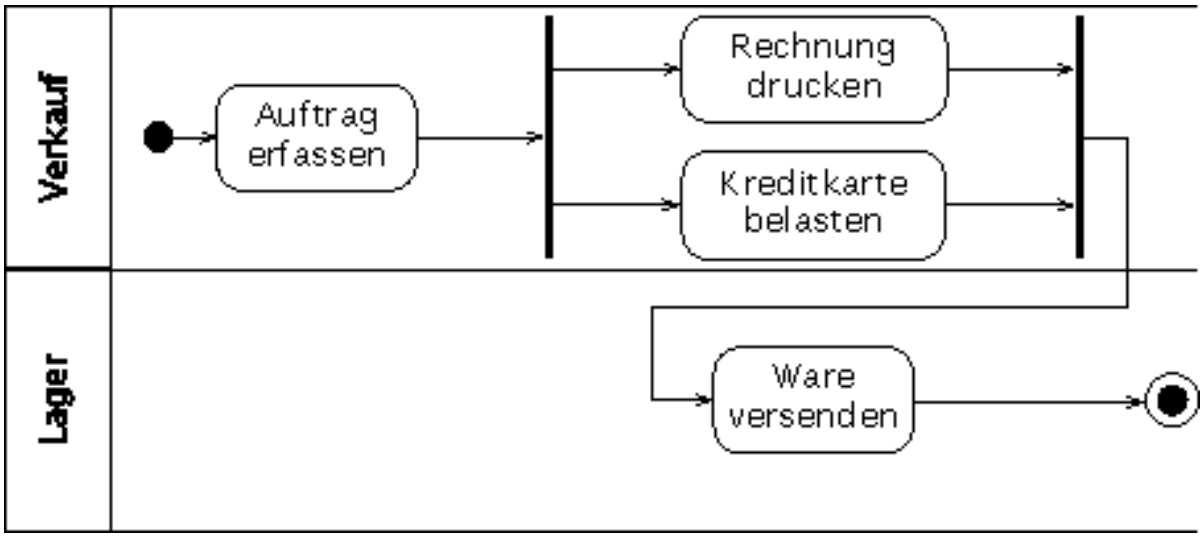
- Beispiel
  - Modellierung der aufgerufenen Aktivität




Quelle: [Balzert 05], Abb. 2.9-18

# Aktivitätsdiagramme

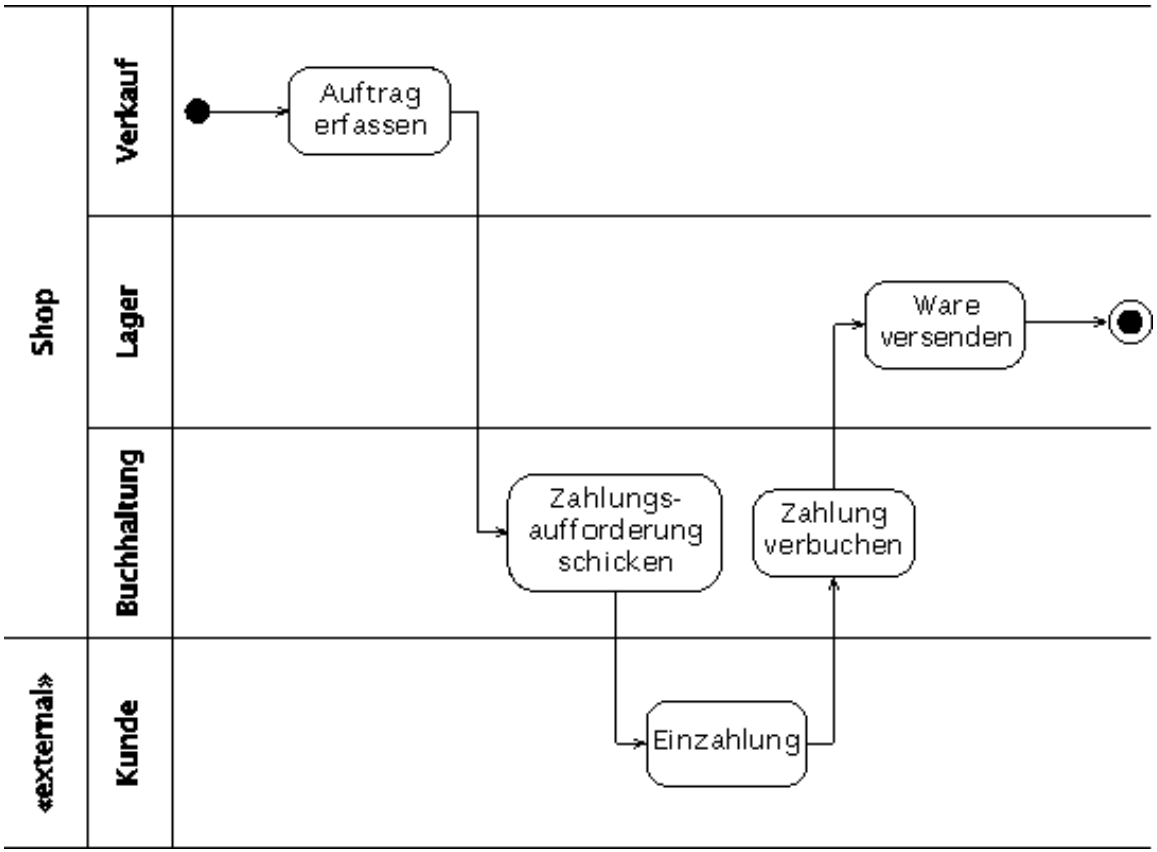
- **Aktivitätsbereiche**
  - **Aktivitätsbereiche** *(activity partitions)* fassen Aktionen zusammen, die bestimmte Gemeinsamkeiten besitzen
  - Werden auch als **Verantwortlichkeitsbereiche** oder **Partitionen** bezeichnet
  - Werden oft wie Schwimmbahnen *(swimlanes)* angeordnet




Quelle: [Balzert 05], Abb. 2.9-14

# Aktivitätsdiagramme

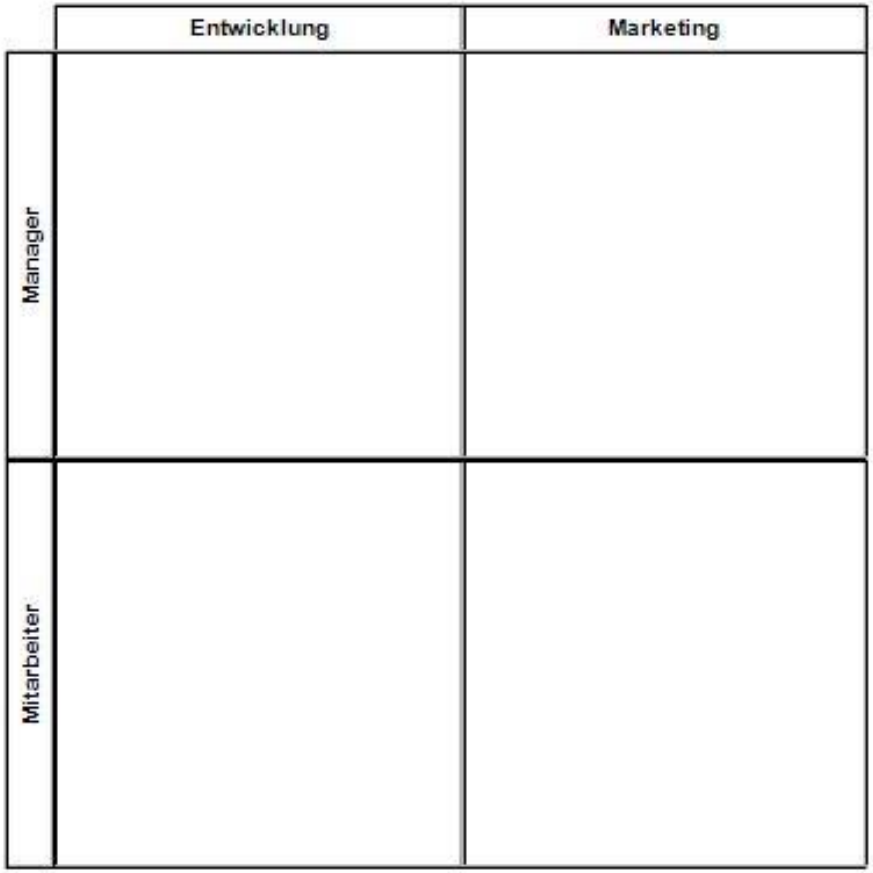
- Aktivitätsbereiche
  - Aktivitätsbereich kann **hierarchisch** strukturiert sein
  - Ein Aktivitätsbereich kann mit «external» beschriftet sein
    - Die darin beschriebenen Aktionen werden außerhalb des Systems durchgeführt




Quelle:
[Balzert 05], Abb. 2.9-15

# Aktivitätsdiagramme

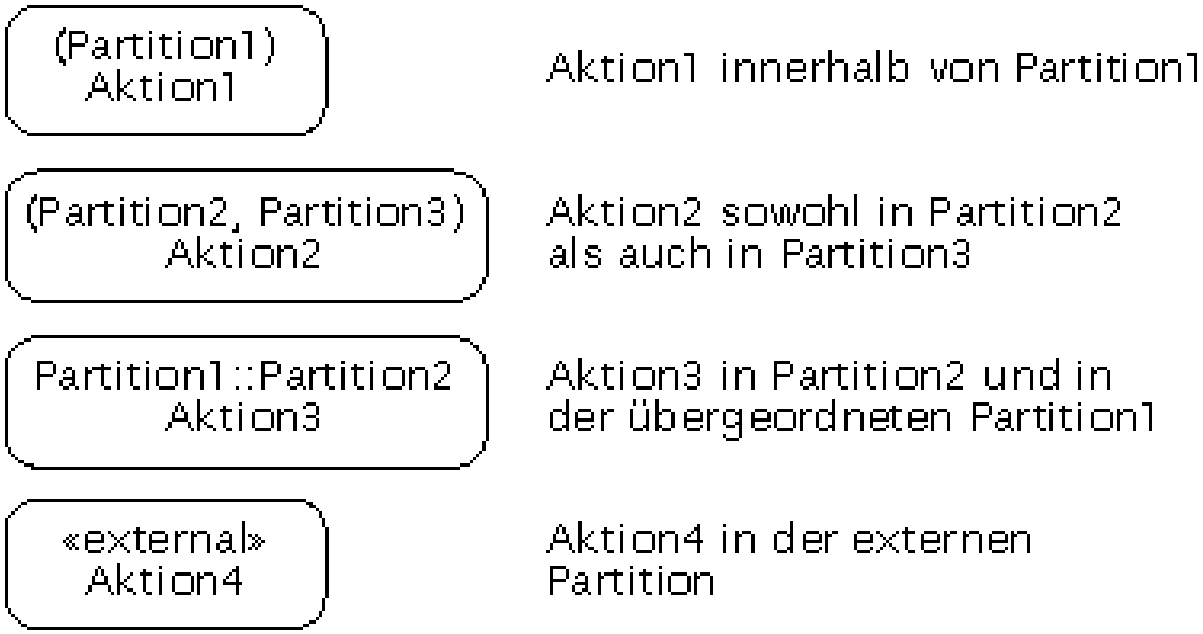
- Aktivitätsbereiche
  - Eine Aktivität kann auf unterschiedliche Arten in Aktivitätsbereiche eingeteilt sein (**mehrdimensionale** Aktivitätsbereiche)
    - Aktionen können gleichzeitig mehreren Aktivitätsbereichen zugeordnet sein
  - Notation:




Quelle: [Rupp et al. 07], Abb. 13.97

# Aktivitätsdiagramme

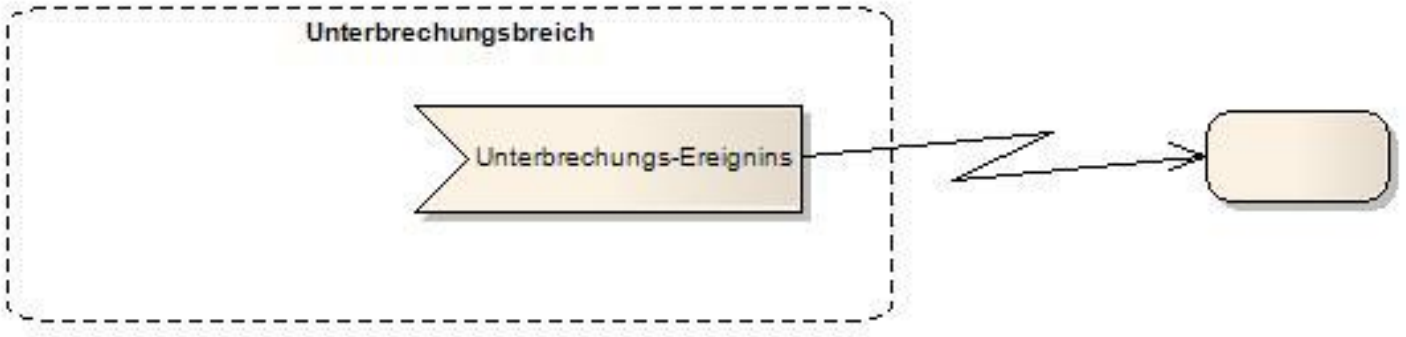
- Aktivitätsbereiche
  - Alternativ ist Text-Notation möglich
    - die Partition in runden Klammern über dem Aktionsnamen angegeben




Quelle: [Balzert 05], Abb. 2.9-16

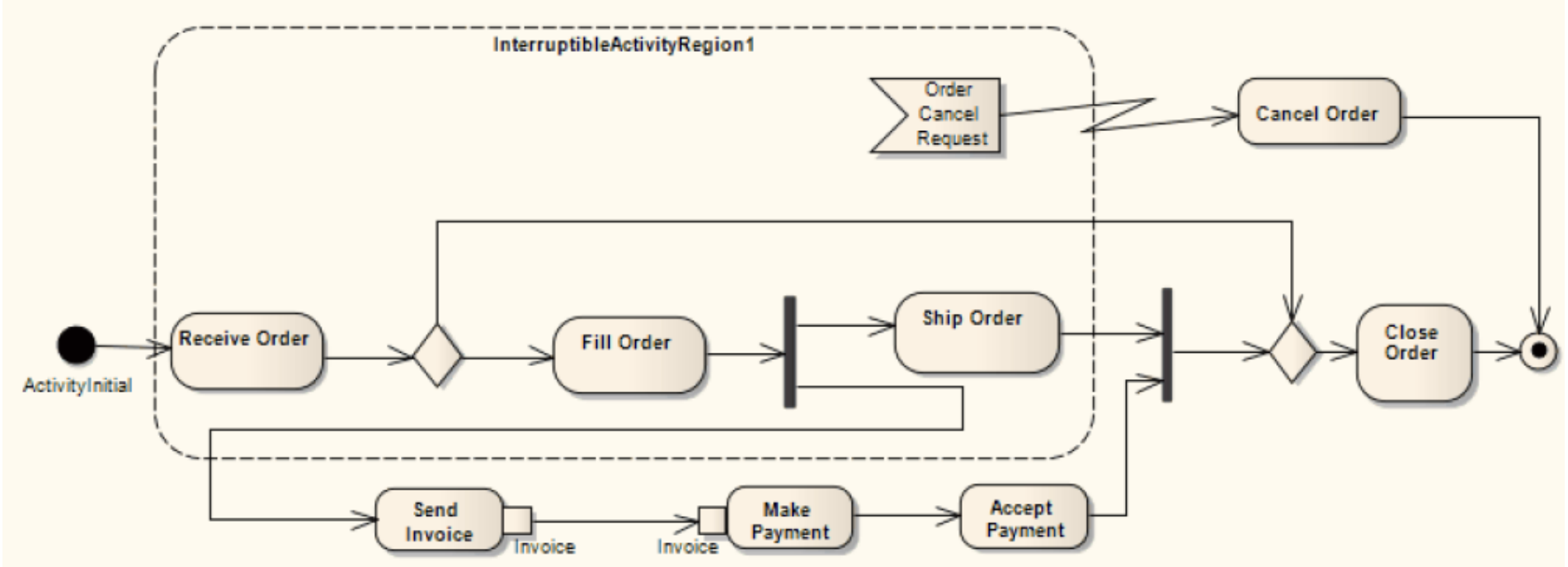
# Aktivitätsdiagramme

- **Unterbrechungsbereich**
  - Ein **Unterbrechungsbereich** fasst eine Gruppe von Aktivitätselemente (Knoten, Kanten) zusammen, deren Ablauf, durch ein **Unterbrechungsereignis** jederzeit unterbrochen werden kann.
  - Über eine **Unterbrechungskante** (*interrupt flow*) wird der Unterbrechungsbereich verlassen, sobald das Unterbrechungsereignis eintritt.
  - Dadurch werden alle im Unterbrechungsbereich laufenden Aktivitäten sofort beendet
- Notation



# Aktivitätsdiagramme

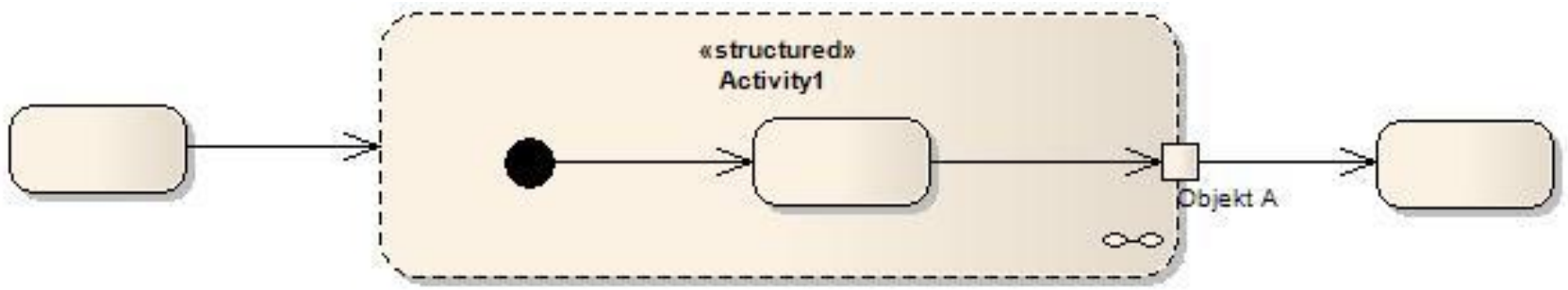
- **Beispiel**




*Quelle:* online-Hilfe des Enterprise Architect 7.5

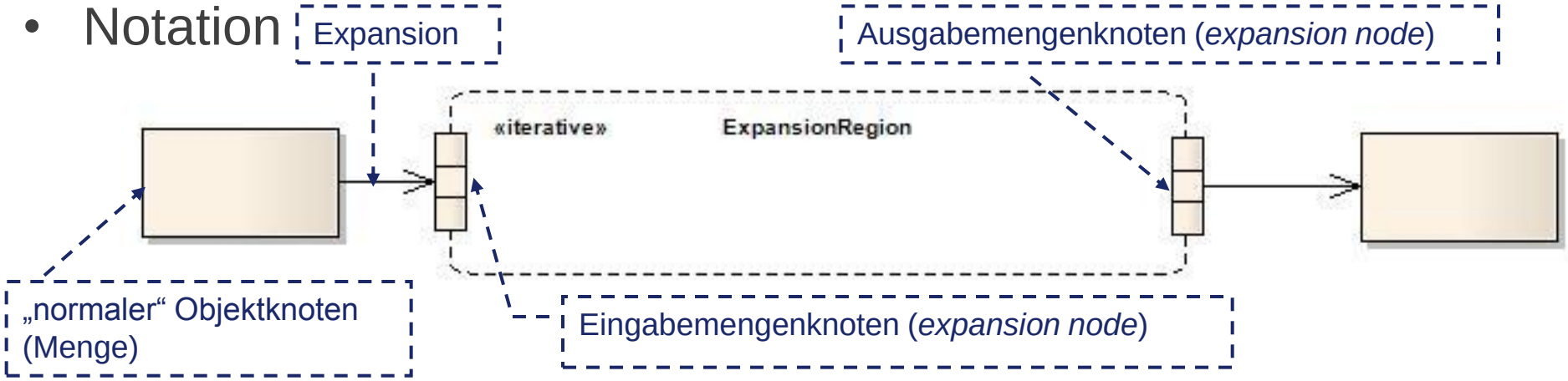
# Aktivitätsdiagramme

- **Strukturierter Knoten**
  - Ein **strukturierter Knoten** fasst Aktivitätselemente (Knoten, Kanten) in Gruppen zusammen
  - kann als ganzes in Abläufe eingebunden werde
  - kann auch Objektknoten (Pins) besitzen
  - Spezialfälle: Entscheidungsknoten, Schleifenknoten, Mengenverarbeitungsbereiche
- Notation:
  - Gestricheltes Rechteck mit abgerundeten Ecken
  - Verwendung des Stereotyps <<structured>>
    - Bei den Spezialfällen spezielle Stereotypen



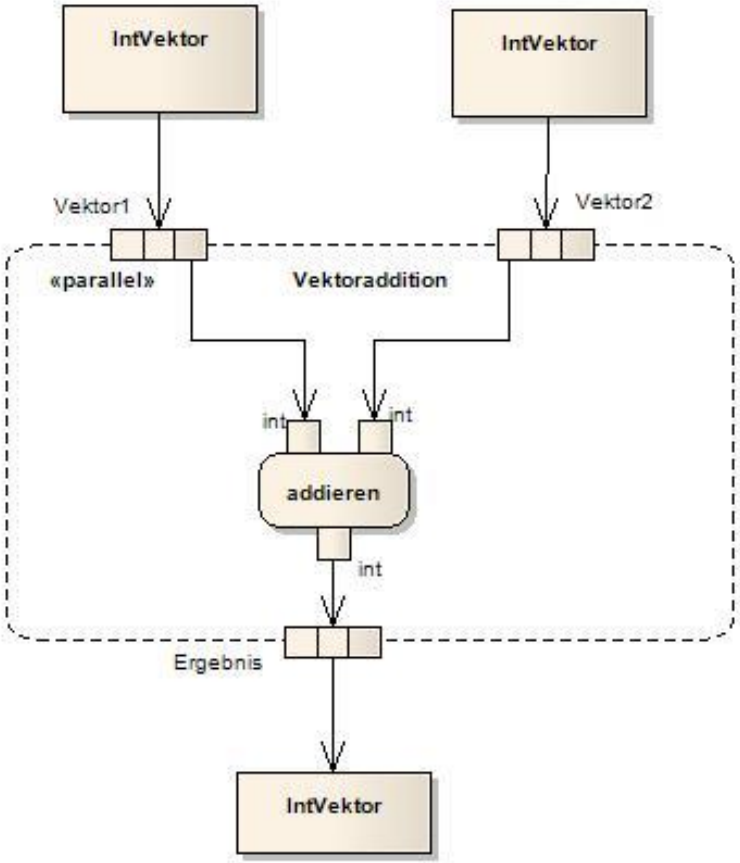
# Aktivitätsdiagramme

- **Mengenverarbeitungsbereich** (*expansion region*)
  - Ein Mengenverarbeitungsbereich fasst eine Gruppe von Aktivitätselementen zusammen, die wiederholt mit den Elementen einer Eingabemenge ausgeführt wird
  - hat als Eingabe eine oder mehrere Kollektionen von Werten
  - außerhalb des Mengenverarbeitungsbereichs wird die Kollektion als ganzes bearbeitet
  - innerhalb wird werden die einzelnen Elemente bearbeitet
    - Die Kollektion wird *expandiert*
  - die Bearbeitung kann *iterativ*, *parallel* oder *streaming* erfolgen
- Notation

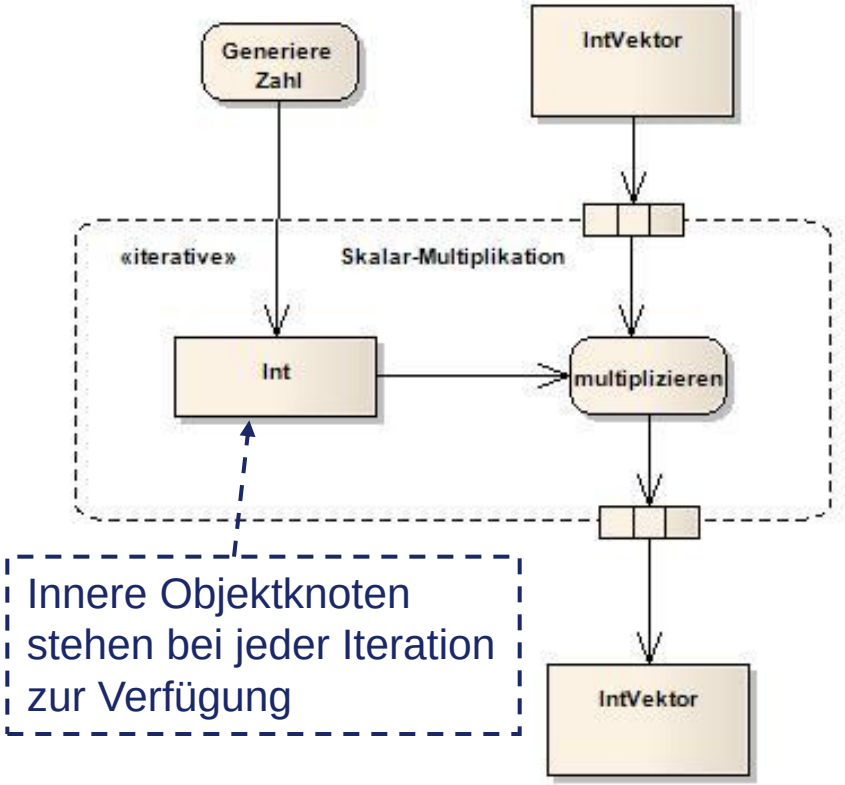


# Aktivitätsdiagramme

- Beispiel - Vektoroperationen



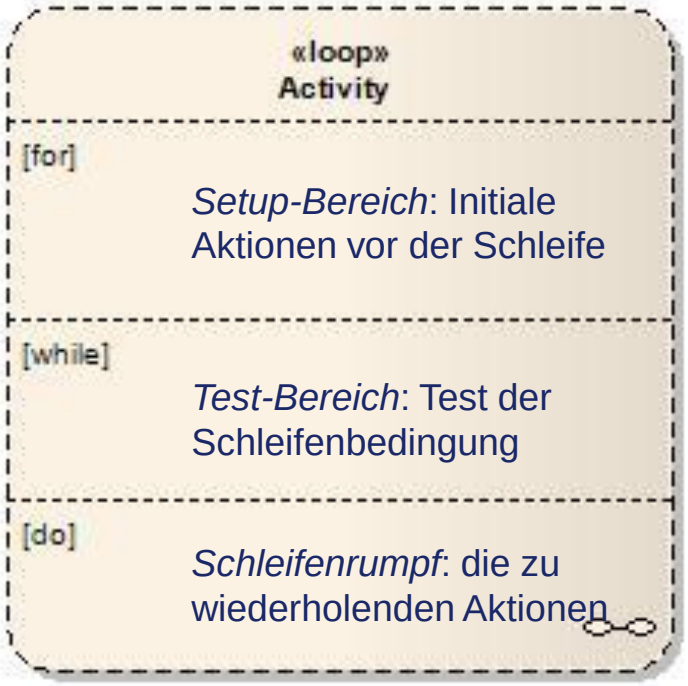

Vektoraddition mit paralleler Verarbeitung der Vektor-Komponenten



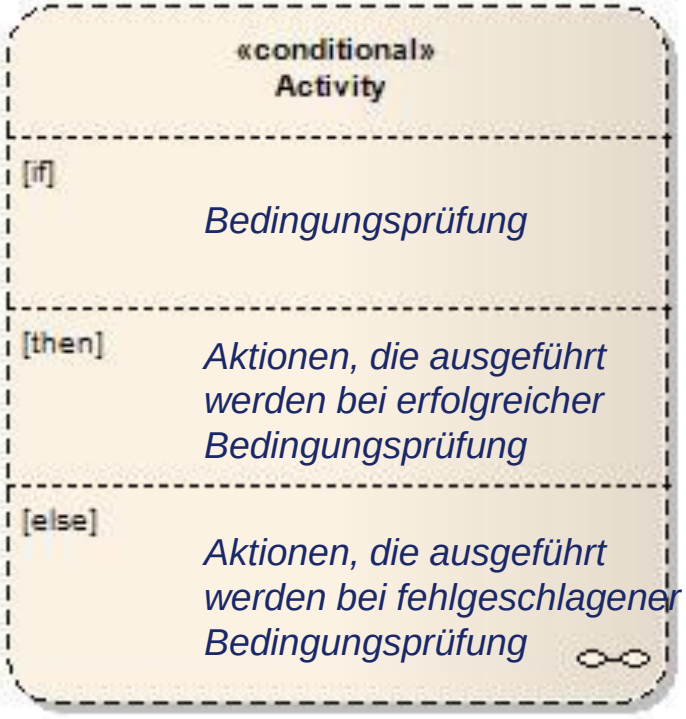

Multiplikation eines Vektors mit einer Zahl, iterative Verarbeitung der Vektor-Komponenten

# Aktivitätsdiagramme

- Notation **Schleifen-** und **Entscheidungsknoten**




**Schleifenknoten** (*loop node*)




**Entscheidungsknoten** (*decision node*). Auch Mehrfachentscheidung möglich (*if … then … elseif … then …. elseif …*)

# Aufgabe 1



- Ablauf eines Café-Besuchs
  - Beschreiben Sie den folgenden typischen Ablauf eines Besuchs in einem Café, bei dem ein Kaffee getrunken und eine Zeitung gelesen wird, als Aktivitätsdiagramm:
    - Der Besucher betritt das Kaffee . Falls noch ein Tisch frei ist, holt er sich als erstes eine Zeitung und setzt sich an einen freien Tisch. Falls kein Tisch mehr frei ist, verlässt er das Kaffee wieder.
    - Der Besucher liest die Zeitung. Parallel dazu wartet er auf die Bedienung und bestellt, sobald sie erscheint, einen Kaffee.
    - Die Bedienung nimmt die Bestellung auf, der Kaffee wird an der Theke zubereitet und anschließend serviert die Bedienung den Kaffee
    - Der Besucher trinkt seinen Kaffee (und liest weiter Zeitung).
    - Sobald er die Zeitung zu Ende gelesen und den Kaffee komplett getrunken hat, verlässt er das Café
  - Berücksichtigen Sie verschiedene Aktivitätsbereiche
  - Modellieren Sie die Aktion „Der Besucher trinkt seinen Kaffee“ als Aufrufaktion, die wie folgt abläuft:
    - Der Besucher nimmt die Tasse, trinkt Kaffee und stellt die Tasse wieder ab, solange bis die Tasse leer ist.

# Aufgabe 2



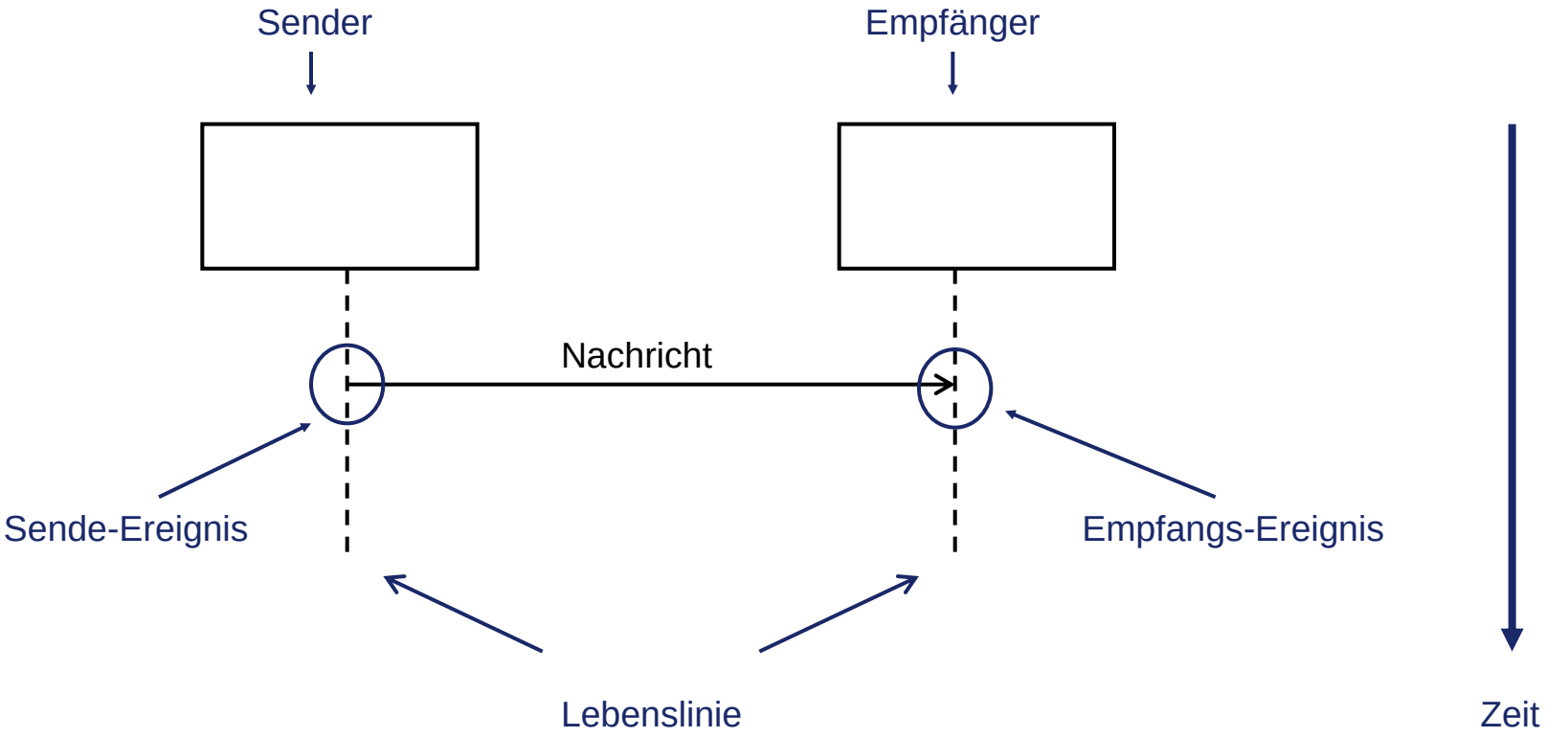
- Bearbeitung einer Bestellung
  - Beschreiben Sie den folgenden Ablauf der Bearbeitung einer Bestellung in einem Warehaus als Aktivitätsdiagramm
    - Sobald eine Bestellung eingegangen ist, wird jede Bestellposition wie folgt bearbeitet:
      - Falls die bestellte Ware vorrätig ist, wird sie im Lager bereitgestellt
      - Falls nicht, wird eine Bestellung der Ware beim Lieferanten veranlasst
      - Nach Lieferung der Ware wird sie im Lager bereitgestellt
    - Nachdem die Ware aller Bestellpositionen bereitgestellt ist, wird eine Rechnung erstellt und zusammen mit der Ware zu einem Paket verpackt.
    - Das Paket wird an den Kunden verschickt.
    - Sobald die Rechnung beglichen wurde, wird der Bestellvorgang beendet
    - Falls innerhalb von 4 Wochen kein Zahlungseingang erfolgt ist, wird eine Mahnung erstellt und an den Kunden geschickt.
    - Falls nach weiteren 4 Wochen immer noch kein Zahlungseingang erfolgt ist, wird der Vorgang an die Rechtsabteilung übergeben und abgeschlossen.
  - Berücksichtigen Sie in Ihrem Modell auch den Objektfluss!

# Interaktionsdiagramme

- **Interaktion**
  - Eine **Interaktion** beschreibt das Zusammenspiel von **Kommunikationspartnern**
  - Das „Zusammenspiel“ wird als Austausch von **Nachrichten** zwischen den Kommunikationspartnern beschrieben
    - Ein **Sender** schickt eine Nachricht zu einem **Empfänger**
    - Sender und Empfänger sind die beteiligten Kommunikationspartner
    - Senden einer Nachricht ist mit zwei Ereignissen verbunden: **Sende-Ereignis** beim Sender und **Empfangs-Ereignis** beim Empfänger
  - Beispiele für **Nachrichten**:
    - Aufruf einer Operation
    - Rückantwort als Ergebnis einer Operationsabarbeitung
    - Signal (z.B. ein Zeitereignis)
    - Logisches/analytisches Ereignis („Käufer unterschreibt Vertrag“)
    - Setzen einer Variablen mit einem Wert

# Interaktionsdiagramme

- Interaktion
  - Grundelemente einer Interaktion in Sequenzdiagramm-Form
    - Sender und Empfänger werden repräsentiert durch eine „Lebenslinie“

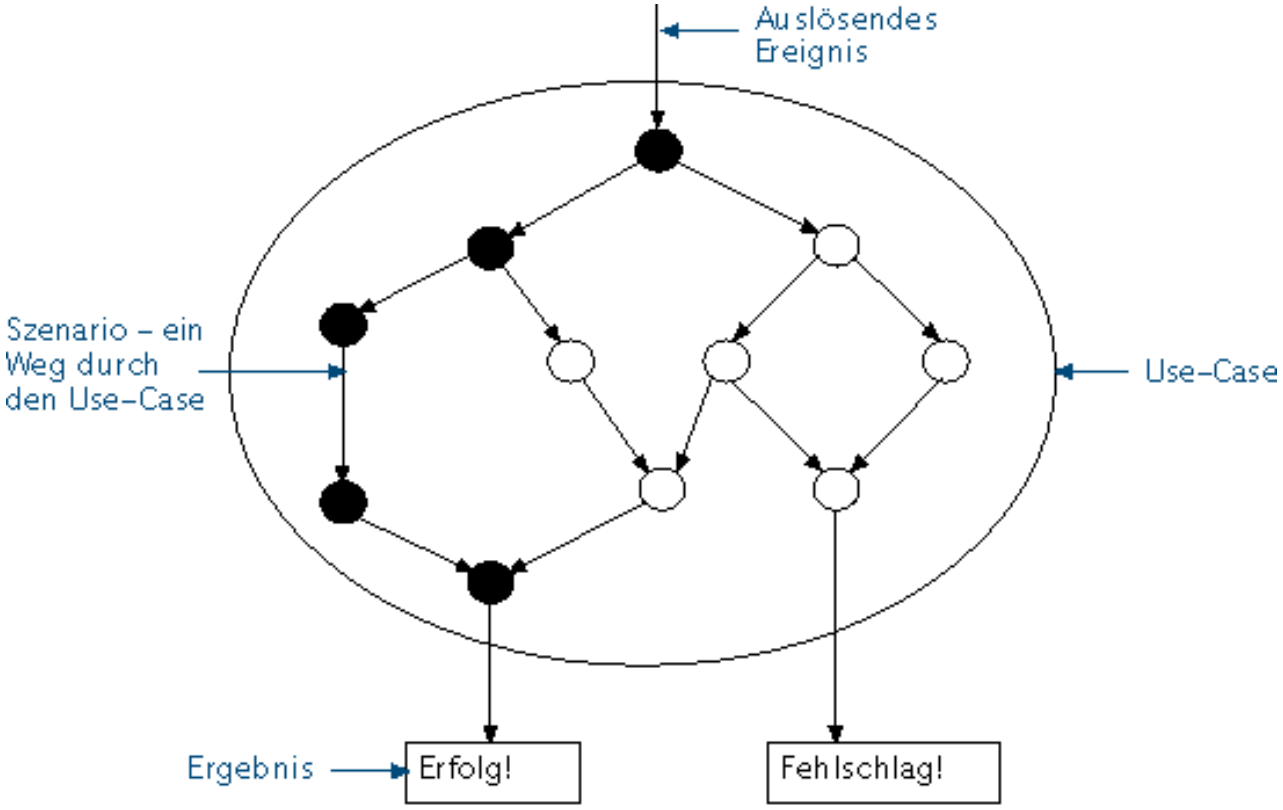


- **Szenario**
  - Ein **Szenario** *(scenario)* ist eine Sequenz von Verarbeitungsschritten, die unter bestimmten Bedingungen auszuführen ist
  - Anwendungen
    - Beschreibung von Anwendungsfällen
      - Ein Szenario kann einen beispielhaften Ablauf eines Anwendungsfalls beschreiben
      - Ein Anwendungsfall kann durch eine Kollektion von Szenarien dokumentiert werden
      - Jedes Szenario wird durch eine oder mehrere Bedingungen definiert, die zu einem speziellen Ablauf des Anwendungsfalls führen
    - Testszenarien
      - Beschreiben, wie sich das System bei einem Test verhalten soll
  - Dokumentation eines Szenarios:
    - **Name** des Anwendungsfalls
    - **Bedingungen**, die zu dieser Variation des Anwendungsfalls führen
    - **Ergebnis**

    Keine Abläufe beschreiben!

# Interaktionsdiagramme

- Anwendungsfall und Szenario




Quelle: [Balzert 05], Abb. 2.10-1

- **Szenario**
  - Beispiel: 3 Szenarios zum Anwendungsfall Auftrag ausführen
    - Auftrag für einen Neukunden bearbeiten und mindestens ein Artikel ist lieferbar
    - Auftrag bearbeiten, wenn der Kunde bereits existiert und kein Artikel lieferbar ist
    - Auftrag bearbeiten, wenn der Kunde bereits existiert, sich seine Daten geändert haben und mindestens ein Artikel lieferbar ist

# Interaktionsdiagramme



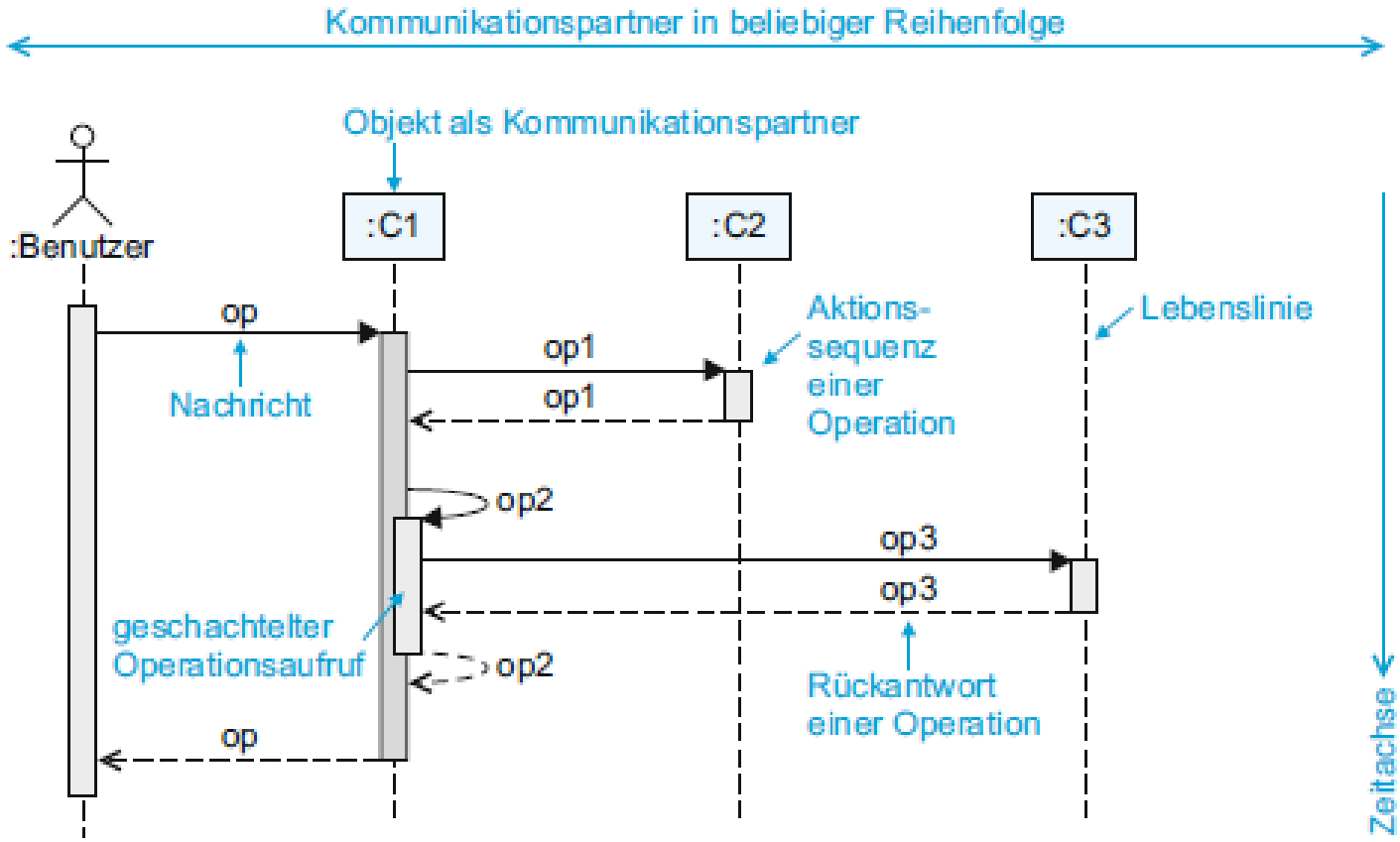
- Modellierung von Szenarien
  - Szenarien werden durch **Interaktionsdiagramme** *(interaction diagrams)* modelliert
  - UML bietet vier Arten von Diagrammen an
    - **Sequenzdiagramm** *(sequence diagram*)
      - Stark erweitert in UML2
    - **Kommunikationsdiagramm** *(communication diagram)*
      - UML 1.x: Kollaborationsdiagramm
    - **Timing-Diagramm** *(timing diagram)*
      - Neu in UML2
    - **Interaktionsübersichtsdiagramm** *(interaction overview diagram)*
      - Neu in UML2

- **Sequenzdiagramm**
  - Zeigt die Kommunikation zwischen mehreren **Kommunikationspartnern**
    - Kommunikationspartner: meist Objekte von Klassen oder Akteure
    - Kommunikation: Austausch von **Nachrichten**
      - Mit Kommunikationspartnern als Sender und Empfänger
  - Besitzt zwei Dimensionen
    - die Vertikale repräsentiert die Zeit
    - auf der Horizontalen werden die Kommunikationspartner eingetragen
  - Darstellung der Kommunikationspartner im Sequenzdiagramm
    - Rechteck mit Namen des Kommunikationspartners
      - bei Objekten: Objekt-Notation (ohne Unterstreichung des Namens)
      - bei Akteuren: Akteur-Symbol
    - vertikale gestrichelte Linie („Lebenslinie“)

- **Sequenzdiagramm**
  - **Nachricht** (Botschaft, *message*)
    - Aufforderung eines Senders (*client*) an einen Empfänger (*server, supplier*), eine Dienstleistung zu erbringen
    - Empfänger bearbeitet die Nachricht
      - → er führt eine Operation gleichen Namens aus
      - → Nachricht an ein Objekt senden ≈ Operation des Objekts aufrufen
    - Darstellung im Sequenzdiagramm
      - Pfeil von der Lebenslinie des Senders zur Lebenslinie des Empfängers
      - **Aktionssequenz:** verbreiterte Lebenslinie des Empfängers
      - beschreibt den Zeitraum, in dem eine empfangene Nachricht bearbeitet wird

# Sequenzdiagramme

- **Notation Sequenzdiagramm UML**



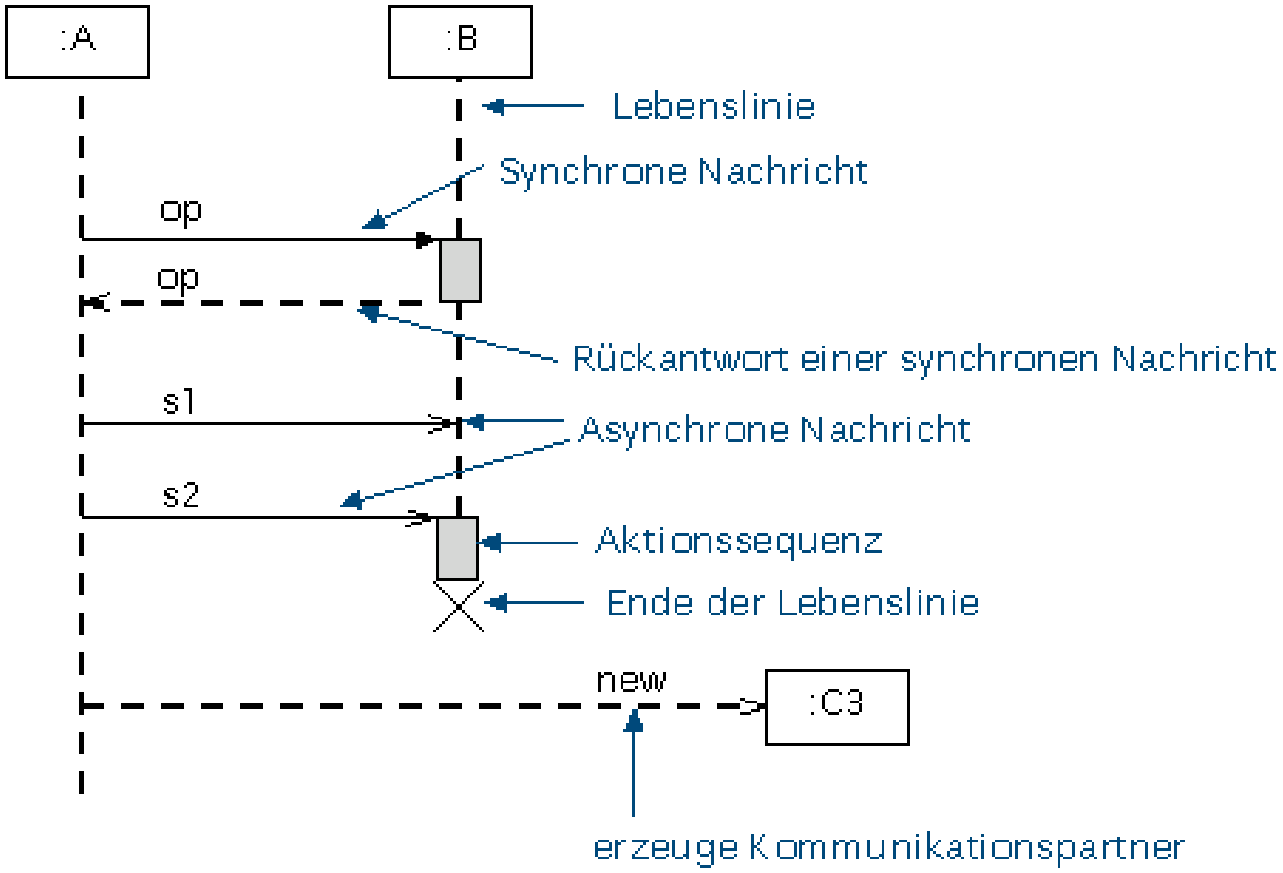

Quelle: [Balzert 09], Abb. 10.5-1

- **Nachrichten** im Sequenzdiagramm
  - **synchrone Nachrichten**
    - der Sender wartet, bis der Empfänger die Verarbeitung durchgeführt hat
    - der Empfänger schickt dem Sender daraufhin eine Antwortnachricht
      - teilt implizit das Ende der Verarbeitung mit
      - kann Antwortdaten enthalten
    - synchrone Nachrichten sind häufig Operationsaufrufe
    - können aber auch Signale sein
    - werden durch einen Pfeil mit gefüllter Spitze dargestellt
      - Rückantwort durch gestrichelte Linie mit offener Spitze
  - **asynchrone Nachrichten**
    - der Sender wartet nicht auf Fertigstellung der Verarbeitung durch den Empfänger
    - setzt stattdessen parallel dazu eigene Verarbeitung fort (nebenläufige Verarbeitung)
    - immer durch Signale realisiert
    - werden durch einen Pfeil mit offener Spitze dargestellt

- Nachrichten im Sequenzdiagramm
  - **Konstruktor-Nachrichten**
    - Erzeugen eines neuen Kommunikationspartners
    - Darstellung: gestrichelter Pfeil
    - Rechteck des neu erzeugten Kommunikationspartners wird auf der gleichen Höhe wie die Nachricht eingetragen
  - **Destruktor-Nachrichten**
    - Nach Bearbeiten der Nachricht hört das Empfänger-Objekt auf zu existieren
    - Lebenslinie endet
    - Darstellung durch ein großes X am Ende der Lebenslinie

# Sequenzdiagramme

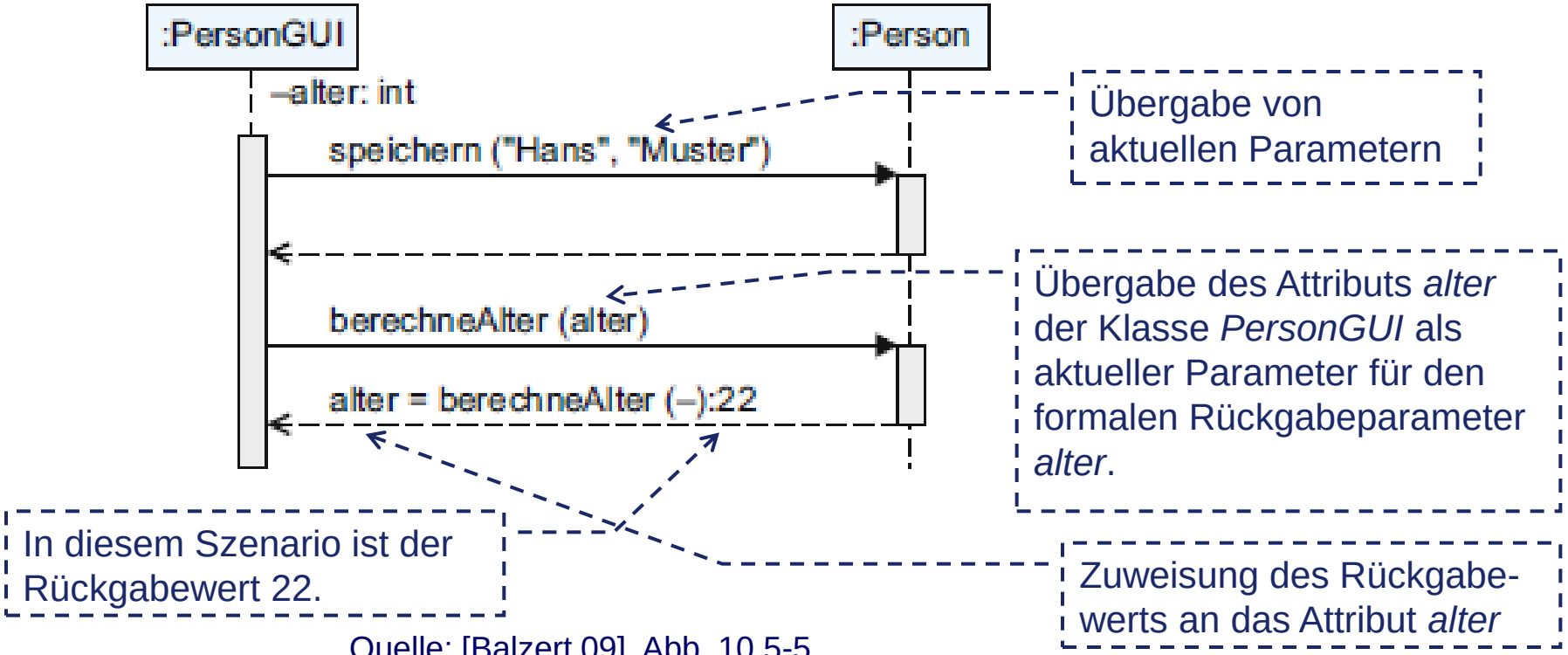
- Notation Nachrichten




Quelle: [Balzert 05], Abb. 6.10-2

# Sequenzdiagramme

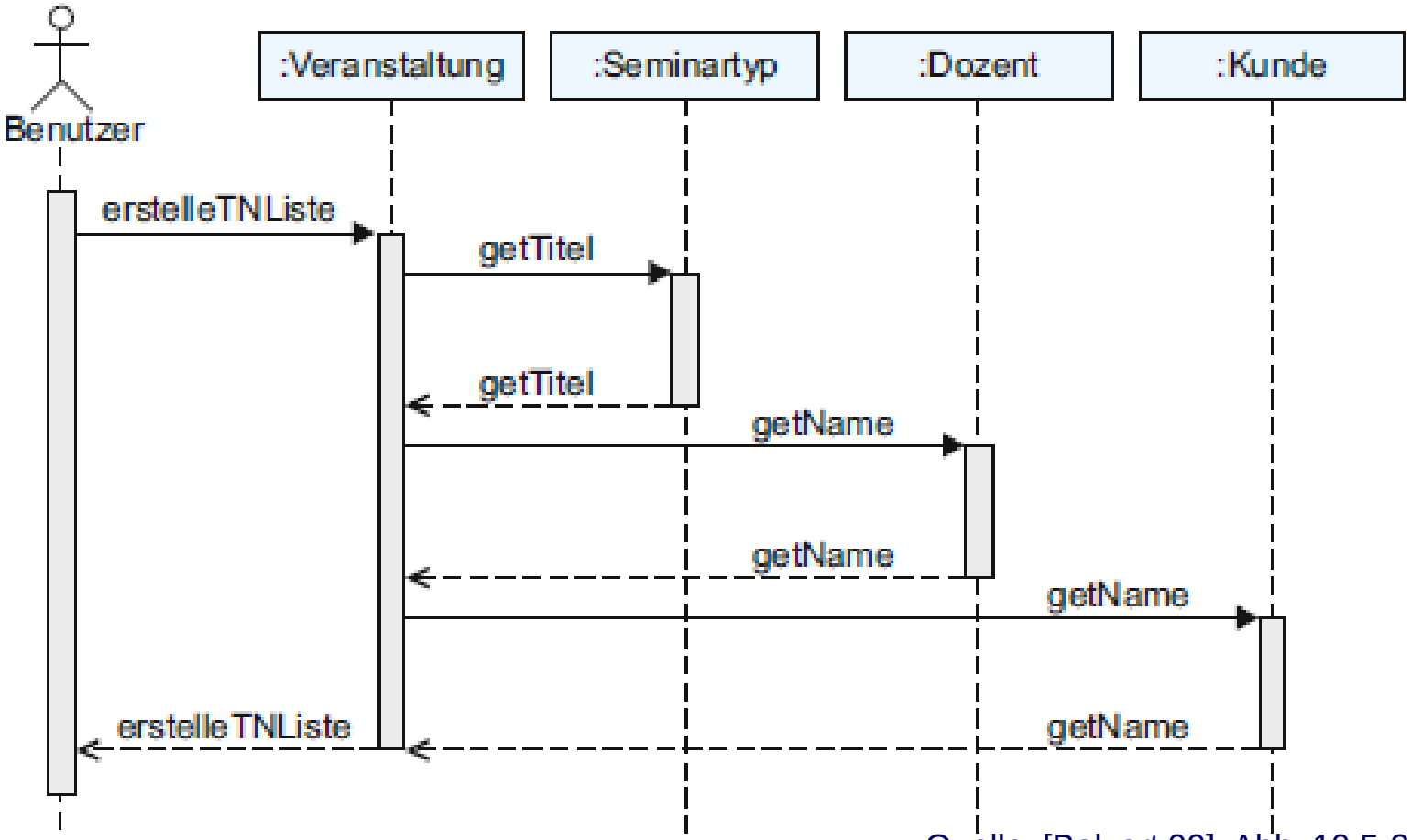
- Nachrichten mit Parameterübergabe
  - Operationen der Klasse Person
    - `speichern(in vorname:String, in nachname:String)`
    - `berechneAlter(return alter:int)`
      - ( ←→ `berechneAlter():int` )




Quelle: [Balzert 09], Abb. 10.5-5

# Sequenzdiagramme

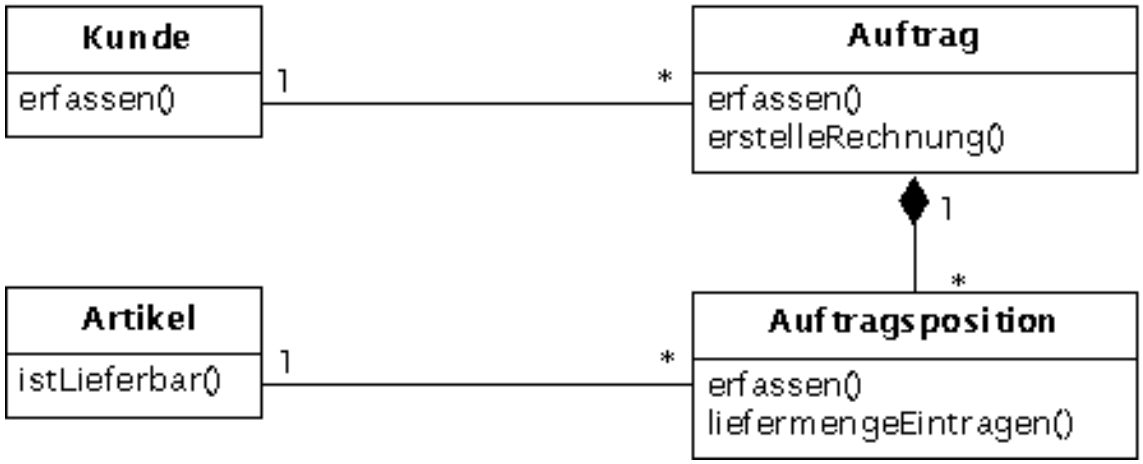
- Beispiel: Seminarorganisation
  - Erstellen einer Teilnehmerliste einer Seminarveranstaltung




Quelle: [Balzert 09], Abb. 10.5-2

# Sequenzdiagramme

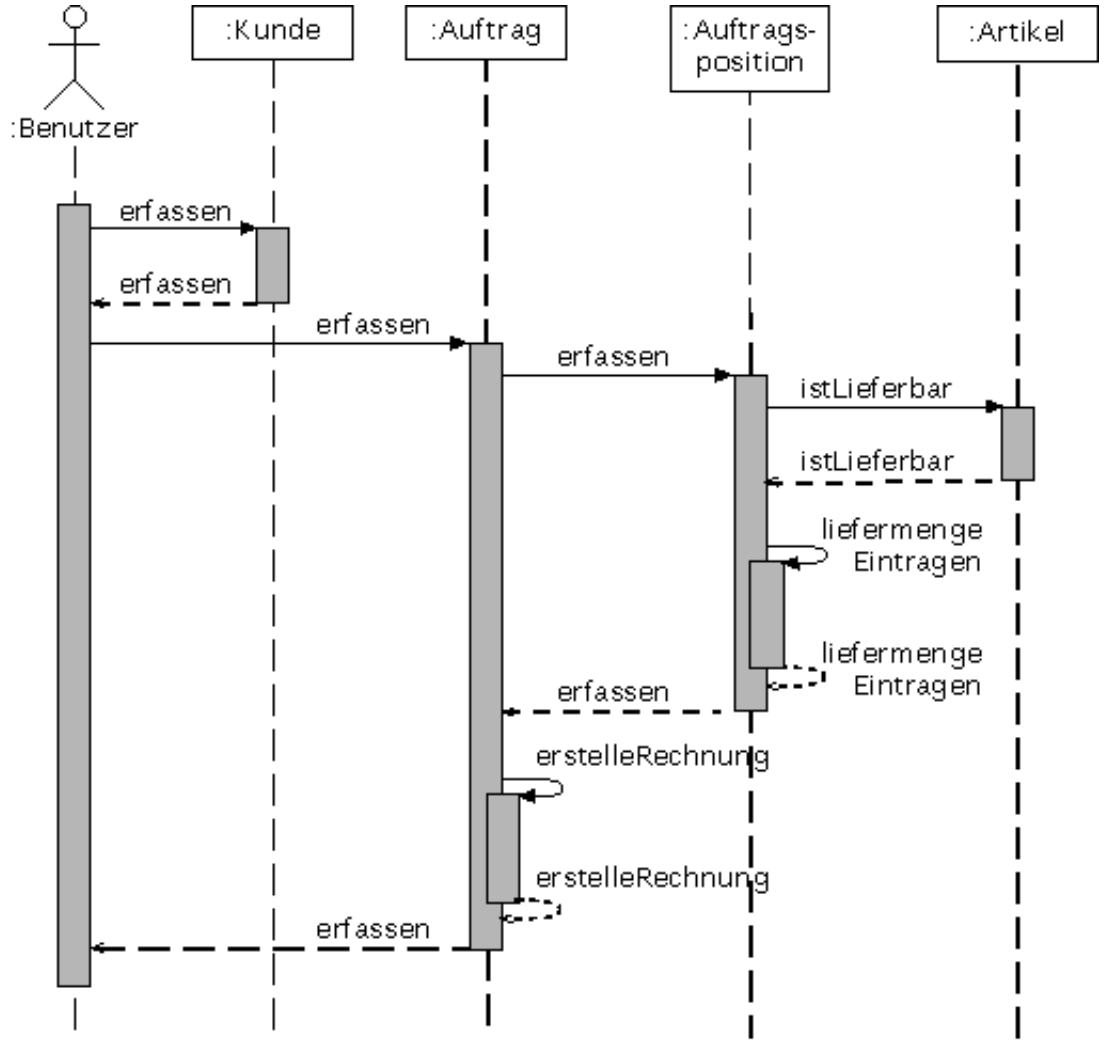
- *Beispiel:* Szenario für den AF „Auftrag ausführen“
  - Klassendiagramm




Quelle: [Balzert 05], Abb. 2.10-4

# Sequenzdiagramme

- *Beispiel:* Szenario für den AF „Auftrag ausführen“
  - Sequenzdiagramm

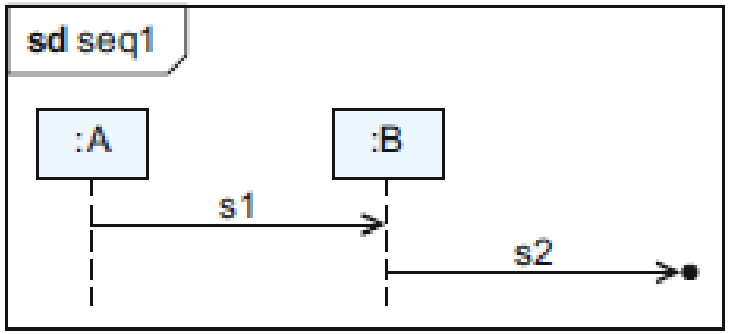


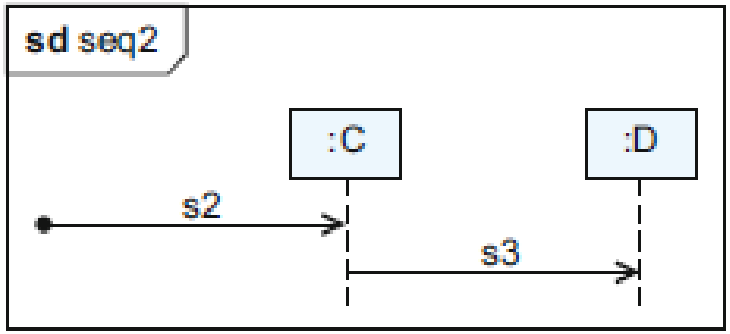

*Anmerkungen:*

- *Konstruktor/Destruktor-Nachrichten sind nicht dargestellt*
- *Ein vollständige Beschreibung des AF umfasst mehrere Sequenzdiagramme*

Quelle: [Balzert 05], Abb. 2.10-4

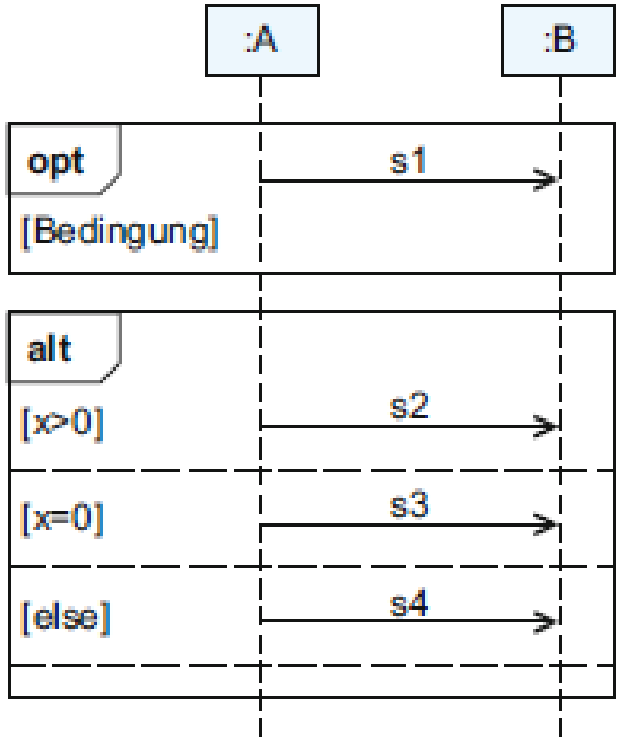
- **Verlorene und gefundene Nachrichten**
  - **Verlorene Nachricht**: Nachricht erreicht nicht ihr Ziel
    - Empfänger zum Modellierungszeitpunkt unbekannt
  - **Gefundene Nachricht**: Das Senden der Nachricht liegt außerhalb der Spezifikation
  - Darstellung: Pfeilspitze zeigt auf einen Punkt bzw. geht von einem Punkt aus



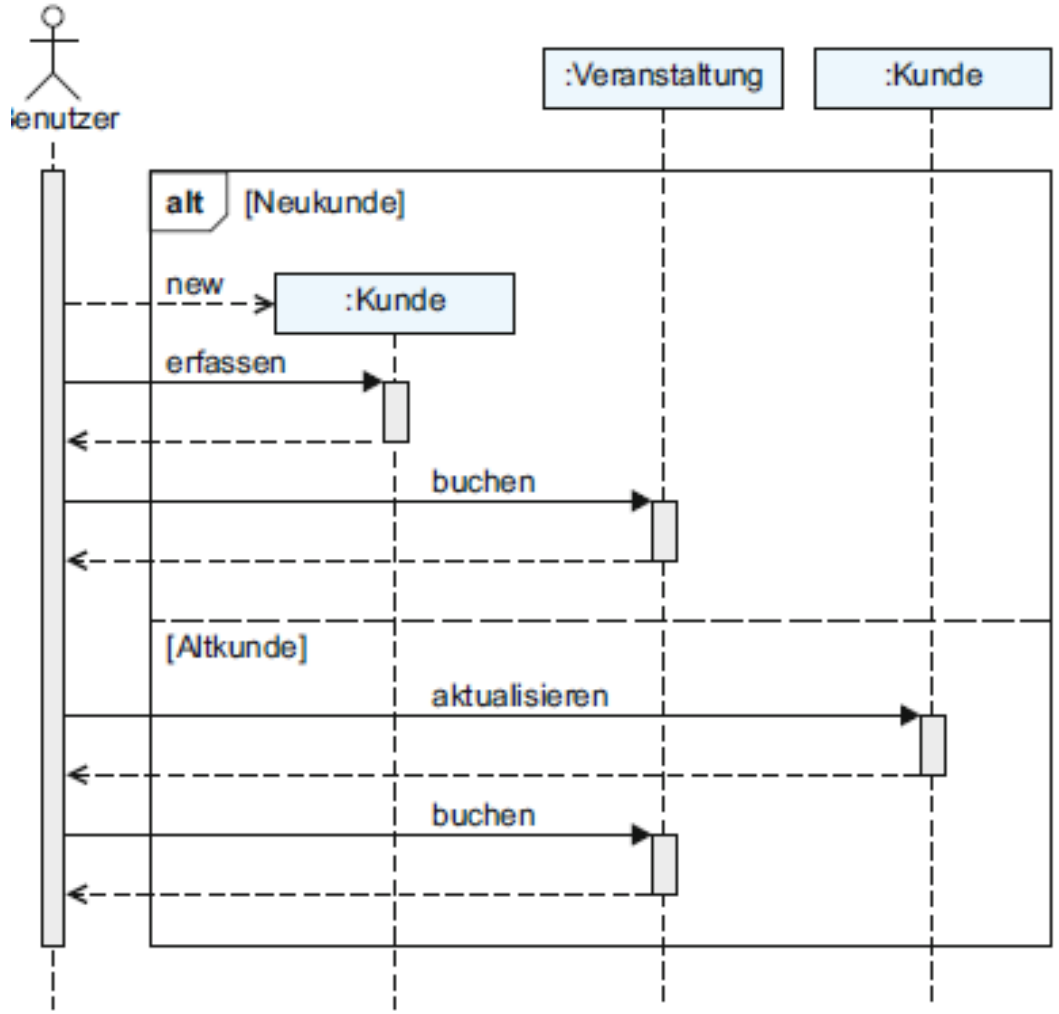

Quelle: [Balzert 09], Abb. 10.5-6

- **Kombinierte Fragmente**
  - Ermöglichen die Beschreibung von nicht-sequenziellen Abläufen
  - Darstellung: Rechteckiger Rahmen (Diagrammrahmen), in dem links oben der **Interaktionsoperator** eingetragen wird
  - Interaktionsoperatoren - Auswahloperatoren:
    - **opt:** optionaler Ablauf (if-then)
    - **alt:** alternative Abläufe (if-then-else, switch)



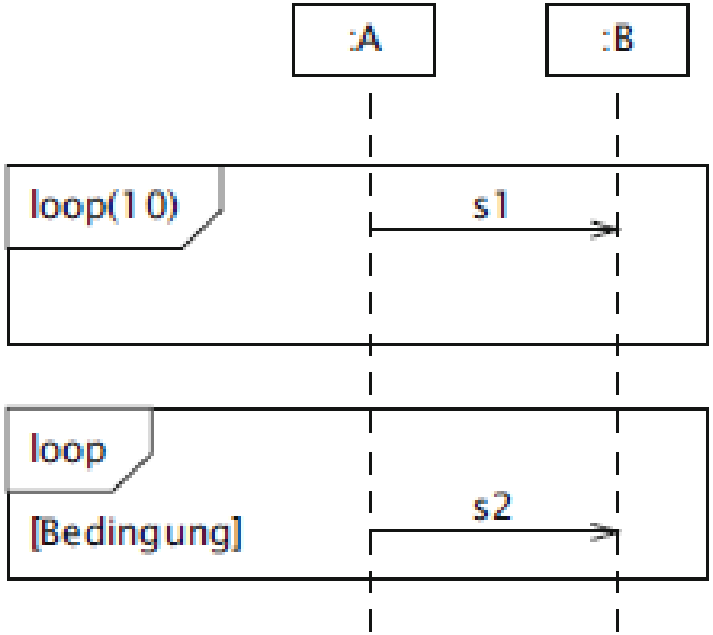

Quelle: [Balzert 09], Abb. 10.5-10

- Kombinierte Fragmente
  - Beispiel: Interaktionsoperator **alt**



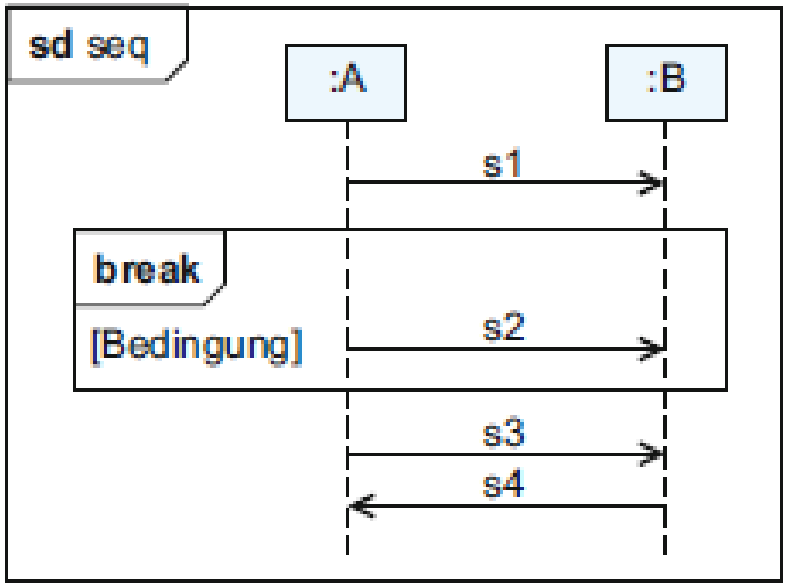

Quelle:
[Balzert 09], Abb. 10.5-9

- Kombinierte Fragmente
  - Interaktionsoperatoren - Schleifen:
    - **loop (min,max):** ganzzahlige Werte, die die Mindest- und Höchstzahl von Wiederholungen angeben
    - **loop (min,*):** min als Untergrenze, unendlich als Obergrenze
    - **loop (min):** Anzahl der Iterationen
    - **loop:** Schleife wird 0 bis unendlich Mal ausgeführt




Quelle: [Balzert 09], Abb. 10.5-11

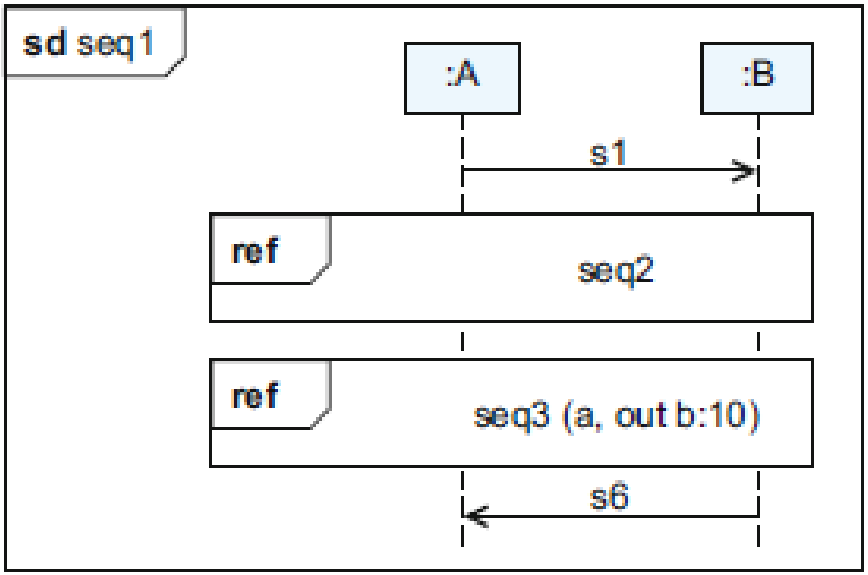
- Kombinierte Fragmente
  - Interaktionsoperatoren - Abbruch
    - **break**: Interaktion bei Ausnahmebehandlungen (exception, goto)
      - Falls bei Erreichen des break-Fragments die [Bedingung] erfüllt ist, wird
        - » Die im Fragment beschriebene Interaktion durchgeführt („Ausnahmebehandlung“)
        - » die umschließende Interaktion abgebrochen

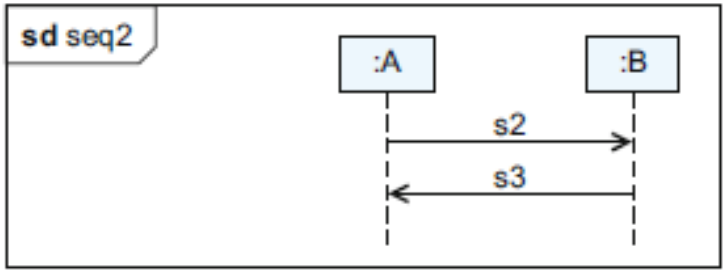


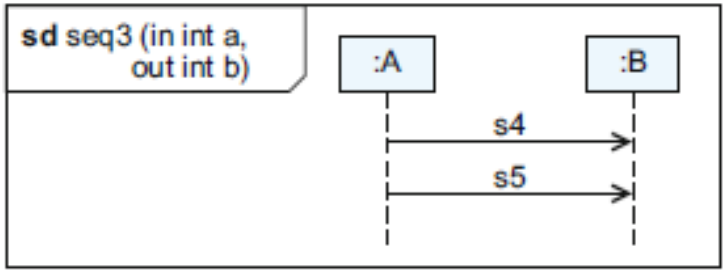

Quelle: [Balzert 09], Abb. 10.5-12

# Sequenzdiagramme

- Interaktionsreferenz
  - Referenz auf ein anderes Sequenzdiagramm oder anderes Interaktionsdiagramm
    - Häufig wiederkehrende Abläufe können durch ein eigenständiges Diagramm beschrieben und an beliebigen Stellen eingefügt werden
    - Parameter können übergeben werden



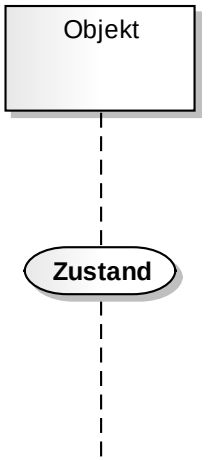



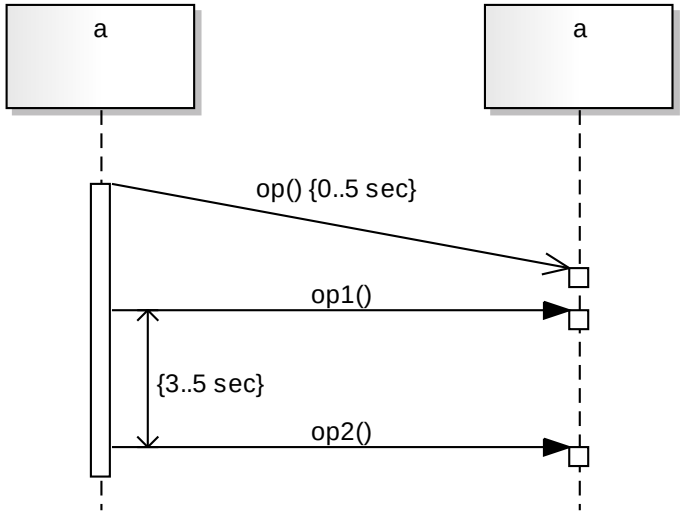

Quelle: [Balzert 09], Abb. 10.5-13

# Sequenzdiagramme

- **Zustandsinvarianten**
  - Auf der Lebenslinie eines Kommunikationspartners kann der Zustand, indem sich das Objekt befindet, symbolisch oder in { … } angetragen werden.
    - Das Objekt muss in dem Zustand sein, sobald alle Interaktionen beendet sind, die zeitlich vorher ablaufen
    - Alle zeitlich nachfolgenden Interaktionen müssen das Objekt in dem Zustand lassen (bis eine neuer Zustand erreicht wird)

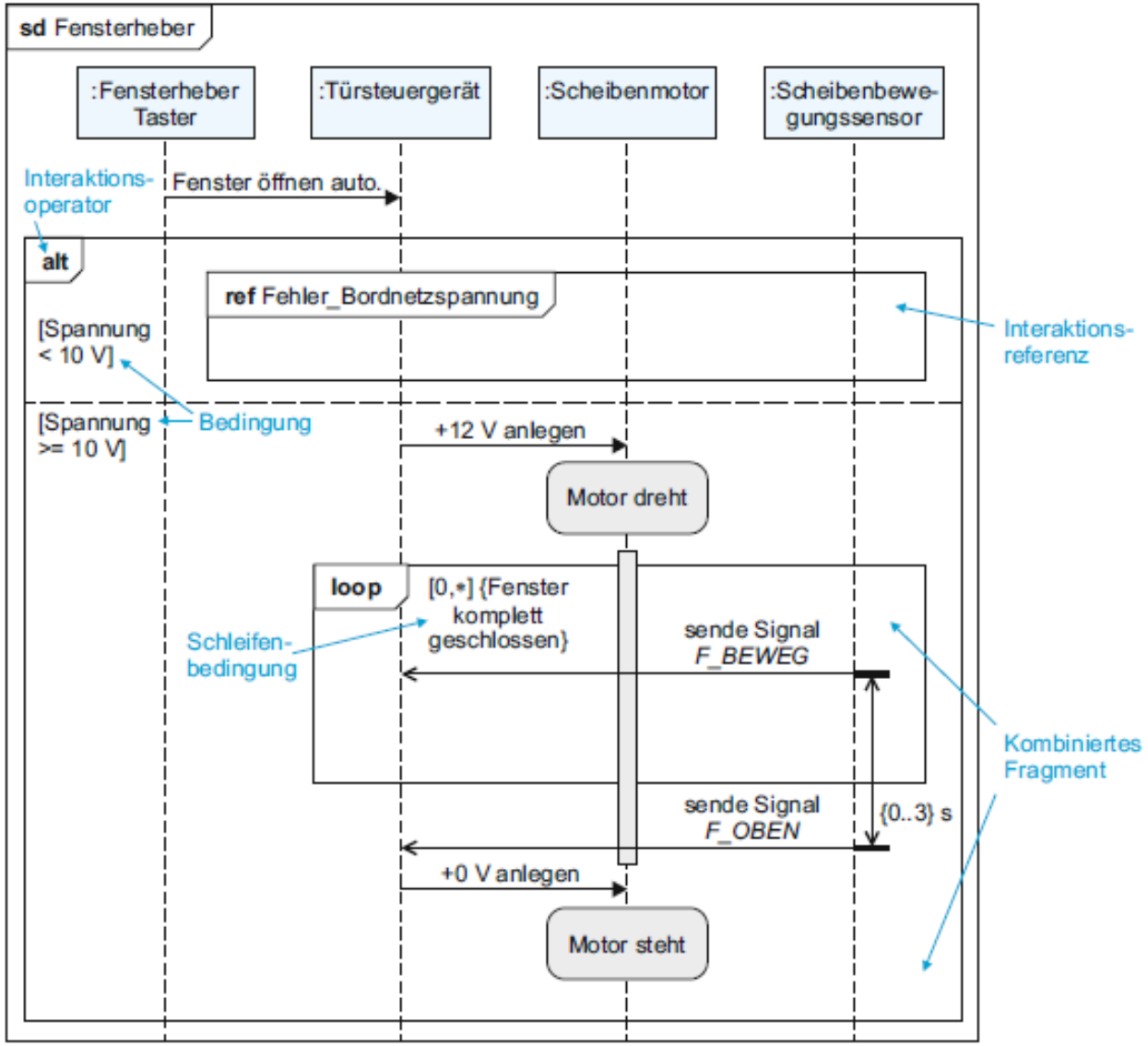



- **Zeitdauer**
  - Die Zeitdauer zwischen zwei Interaktionen kann explizit angegeben werden
  - Die Zeitdauer einer Interaktion kann angegeben werden
    - Maximale Dauer zwischen Sende-Ereignis und Empfangsereignis



# Sequenzdiagramme

- Beispiel: Türsteuergerät
  - Vollständiges Sequenzdiagramm für den Anwendungsfall *Fenster schließen automatisch*
  - Umfasst alle möglichen Szenarien



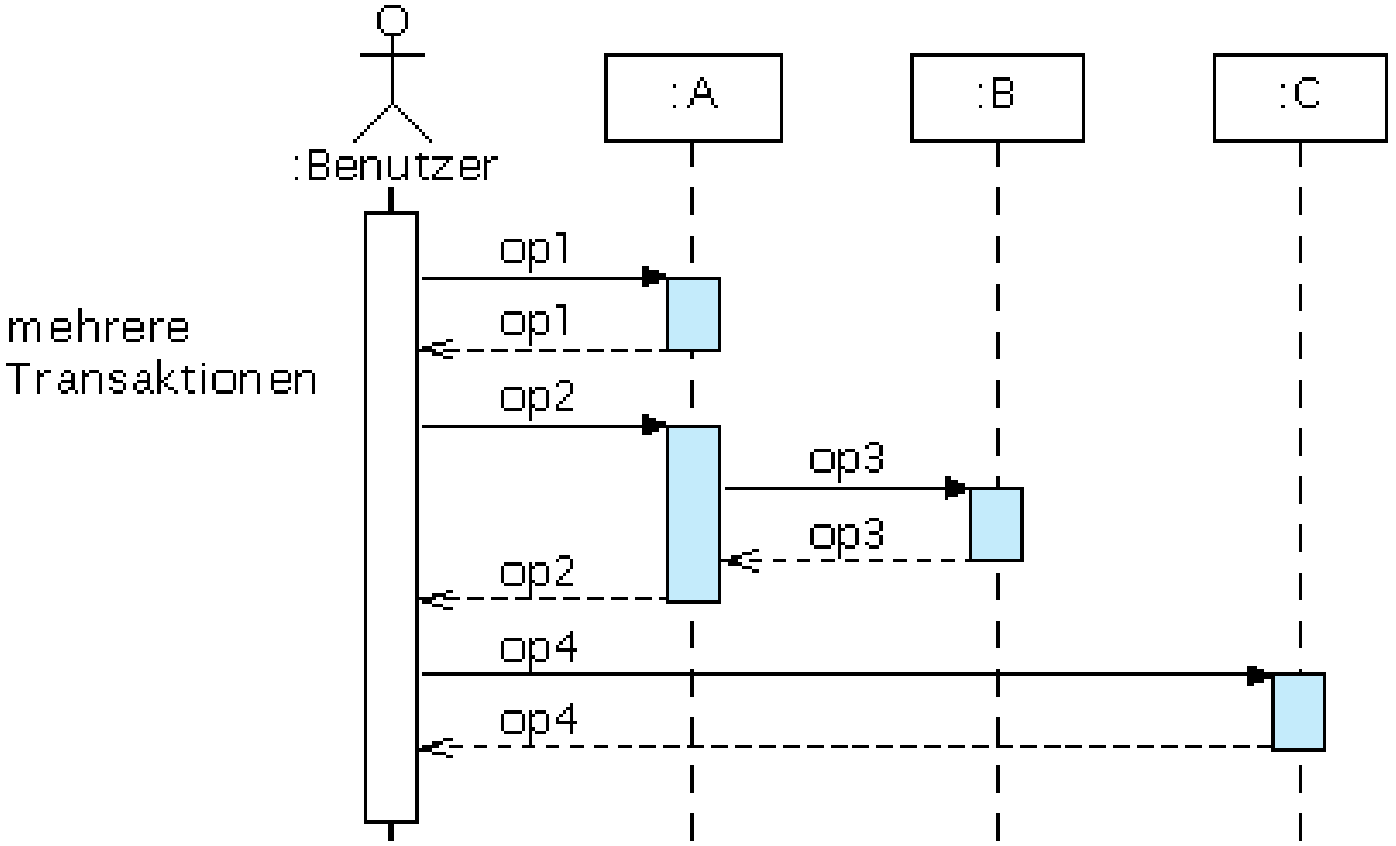

Quelle: [Balzert 09], Abb. 10.5-15

# Sequenzdiagramme

- Anmerkungen
  - Mit Sequenzdiagrammen lassen sich komplizierte Abläufe beschreiben
  - Lesbarkeit und Verständlichkeit ist fragwürdig
    - Häufig sind Pseudocode oder Aktivitätsdiagramme besser verständlich
  - Typische Anwendung von Sequenzdiagrammen:
    - Beschreibung mehrerer einzelner Szenarien für einen Anwendungsfall
    - Jedes Szenario beschreibt einen möglichen Ablauf (ohne Fragmente und Referenzen)
    - Die Bedingungen, unter denen ein Szenario abläuft kann bei den Nachrichten in […] angegeben werden
  - Teil-Ziel der Interaktionsmodellierung
    - Finden der erforderlichen Operationen in Klassen
- Beispiel:
  - Sequenzdiagramme für „Türme von Hanoi“

# Sequenzdiagramme

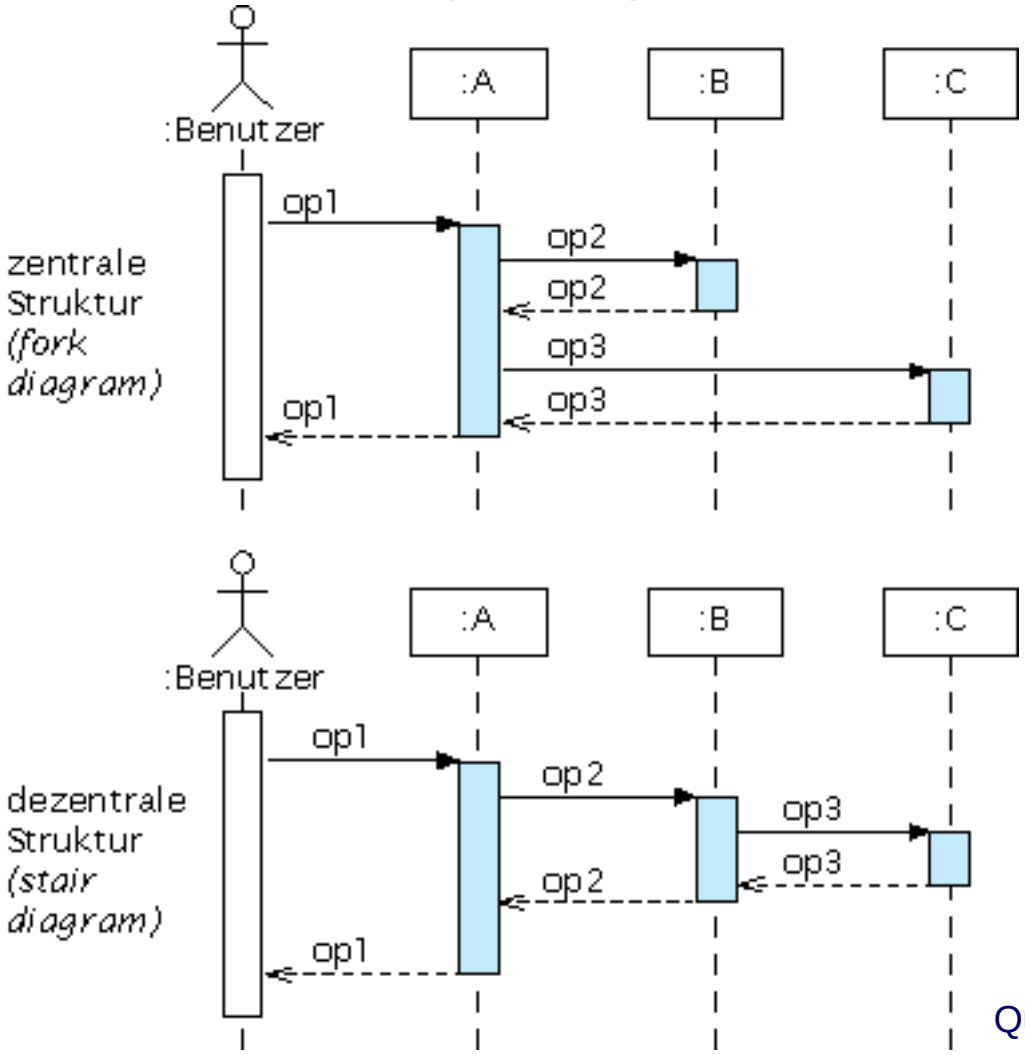
- Struktur von Sequenzdiagrammen
  - **Eine** oder **mehrere** Transaktionen (Interaktionen mit dem Aktor)




Quelle: [Balzert 05], Abb. 4.9-4

# Sequenzdiagramme

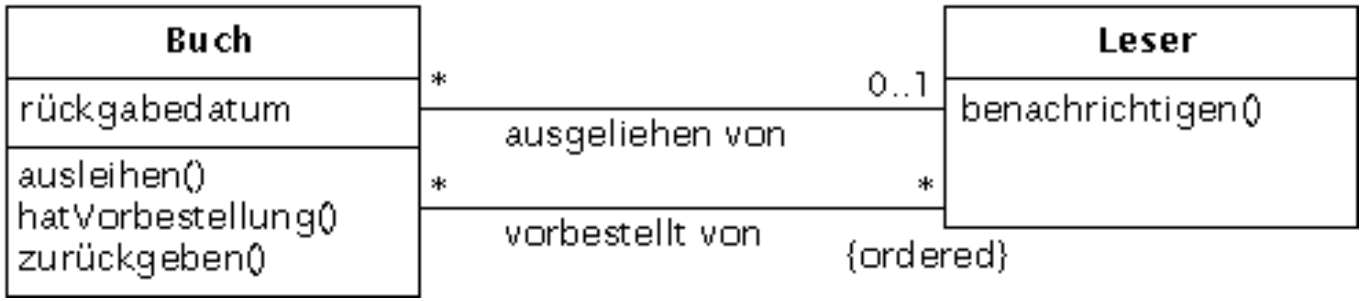
- Struktur von Sequenzdiagrammen
  - **Fork** oder **Stair-** Diagramm (eine Transaktion)




Quelle: [Balzert 05], Abb. 4.9-5

# Sequenzdiagramme

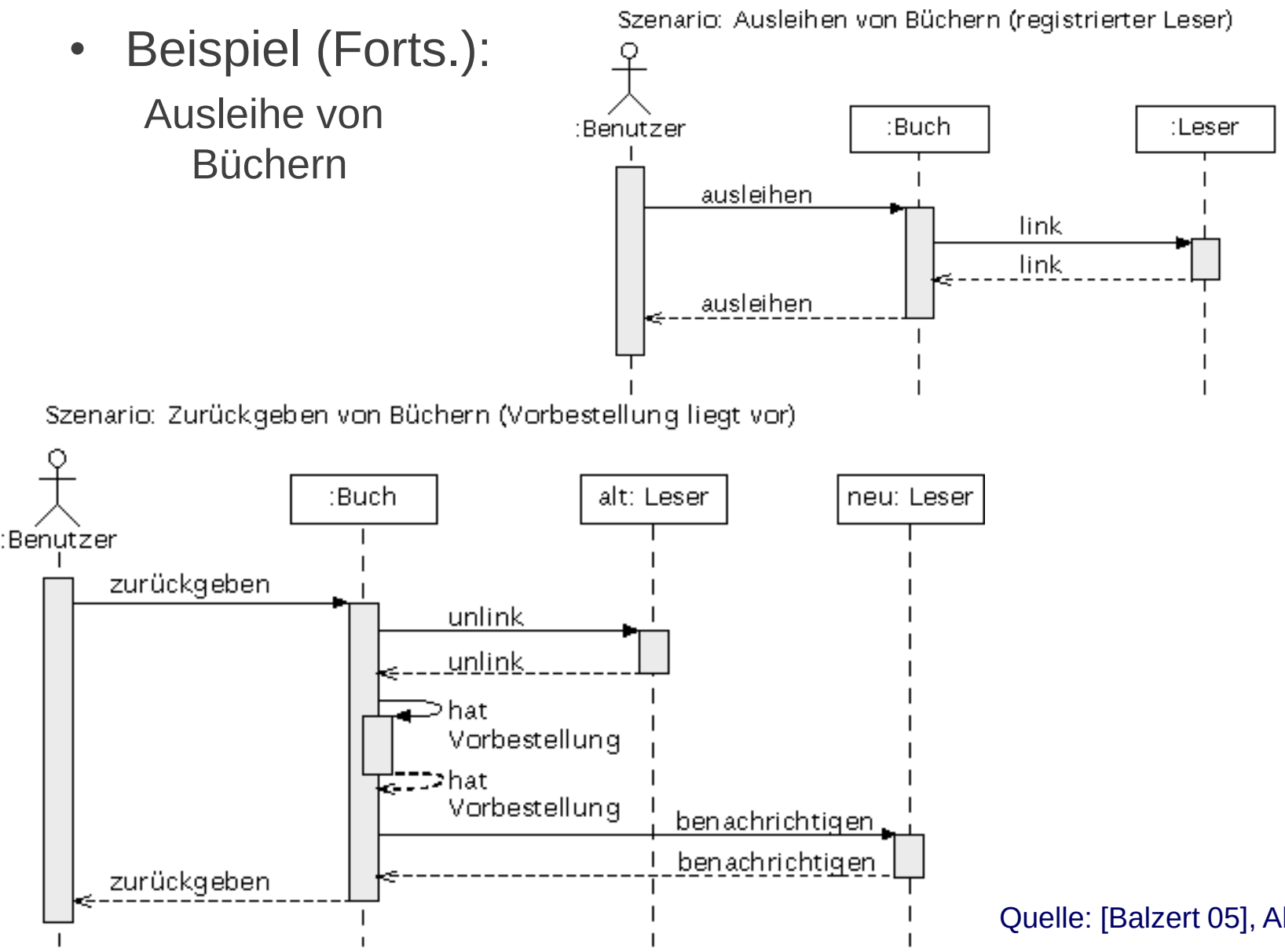
- Qualitätskriterien
  - Können alle Sender-Objekte ihre Empfänger-Objekte erreichen?
    - Assoziation zwischen Klassen (permanente Verbindung)
    - Nur Temporäre Verbindung erforderlich, wenn …
      - sich Sender und Empfänger nicht permanent kennen müssen
      - Kommunikation nur selten stattfindet
      - die Objektidentität des Empfängers dynamisch ermittelt werden kann
  - Konsistenz von Sequenz- und Klassendiagramm
    - Sind alle Klassen der Kommunikationspartner auch im Klassendiagramm enthalten?
    - Sind alle Operationen außer Verwaltungsoperationen im Klassendiagramm eingetragen?
  - ***Beispiel:*** Ausleihen von Büchern



# Sequenzdiagramme

- Beispiel (Forts.):

  Ausleihe von Büchern



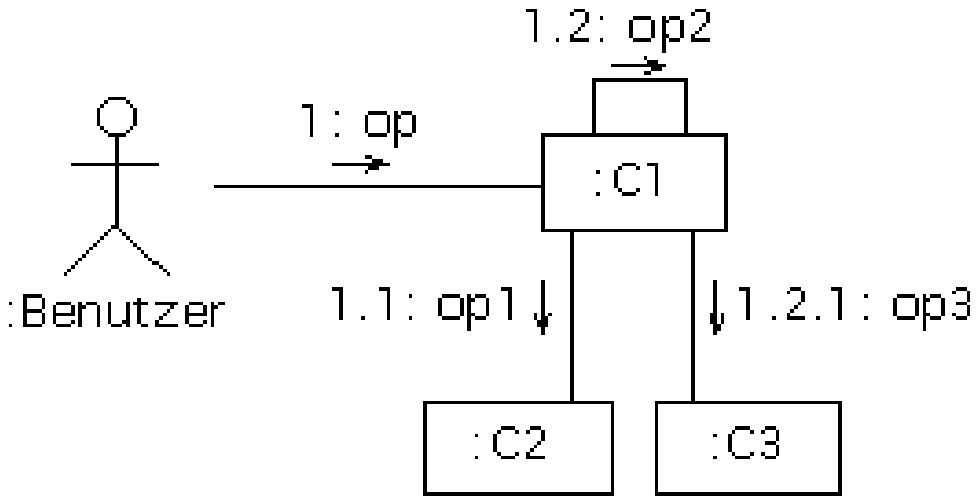

Quelle: [Balzert 05], Abb. 4.9-6

# Kommunikationsdiagramme

- **Kommunikationsdiagramm**
  - Ist gut geeignet, um das grundsätzliche Zusammenspiel mehrerer Kommunikationspartner zu zeigen
  - Kommunikationspartner ohne Lebenslinie, frei angeordnet
  - Zusätzlich Objektverbindungen *(links)* zwischen den Kommunikationspartnern
    - d.h. die Partner können sich gegenseitig Nachrichten schicken
  - Nachrichten
    - an Objektverbindungen zwischen Sender und Empfänger antragen
    - Richtung der Nachricht wird durch Pfeil gekennzeichnet
    - Zeitliche und logische Abfolge der Nachrichten durch Nummerierung kennzeichnen
    - Zusätzlich Schleifen und Bedingungen an Nachrichten:
      - `[Bedingung] nachricht`
        - » Sende Nachricht, falls Bedingung erfüllt
      - `*[Bedingung] nachricht`
        - » Sende Nachricht, solange Bedingung erfüllt

# Kommunikationsdiagramme

- Notation UML

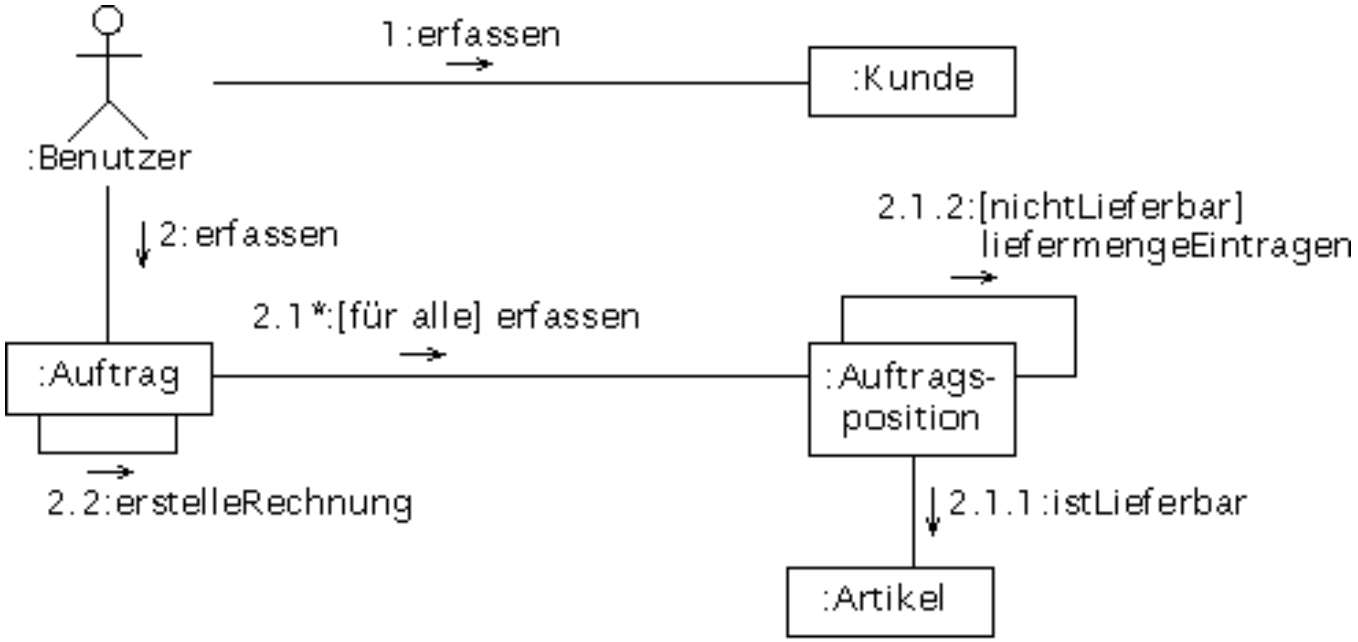



Quelle: [Balzert 05], Abb. 2.10-5

– Diese Kommunikationsdiagramm zeigt denselben Ablauf wie das Sequenzdiagramm auf S. 39
– Der Diagrammrahmen von Kommunikationsdiagrammen hat wie Sequenzdiagramme das Kürzel **sd**!

# Kommunikationsdiagramme

- Beispiel



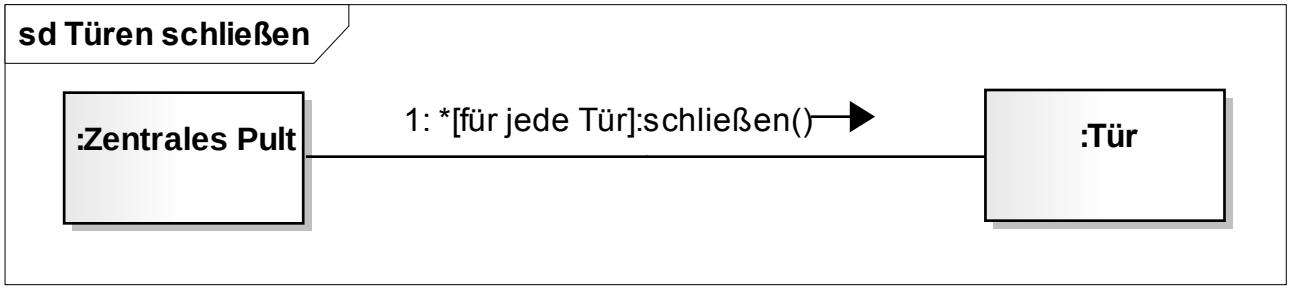

Quelle: [Balzert 05], Abb. 2.10-6

– Diese Kommunikationsdiagramm zeigt denselben Ablauf wie das Sequenzdiagramm auf S. 46

- **Bedingte Nachrichten**
  - Nachrichten können durch eine Bedingung eingeschränkt werden

    *[Bedingung] Nachricht*
  - Die Nachricht wird nur gesendet, wenn die Bedingung erfüllt ist
- **Iterative Nachrichten**
  - Nachrichten können beliebig oft wiederholt gesendet werden
  - Die Anzahl der Wiederholungen kann durch eine Schleifenbedingung definiert werden

    ***** *[Bedingung] Nachricht*
  - Beispiel (Zugtürensystem):
    - Die Türen eines Zuges werden über ein „zentrales Pult“ geschlossen, indem and jede der Türen, die Botschaft „schließen“ gesendet wird



# Kommunikationsdiagramme



- Anmerkungen
  - Kommunikationsdiagramme sind äquivalent mit einer Teilmenge der Sequenzdiagramme
    - In UML 1.x waren sie gleichwertig!
  - Kriterien für die Auswahl von Kommunikationsdiagrammen zur Modellierung:
    - Komplexe Struktur mit vielen Teilen liegt vor und das Zusammenwirken der Teile soll auf einfache Weise dargestellt werden
    - Die Interaktion ist einfach. Zeitliche Abhängigkeiten, Kontrollelemente (Schleifen, Verzweigungen, Nebenläufigkeit) sind weniger wichtig
    - Das Hauptgewicht liegt auf einem grundlegenden Verständnis, weniger auf Details

# Aufgabe 3



- Beschreiben Sie folgendes Szenario als Sequenz- und als Kommunikationsdiagramm.
  - Ein neuer Kunde eröffnet bei einer Bank ein Sparkonto. Zuerst werden die Daten dieses Kunden erfasst. Bei der Kontoeröffnung muss der Kunde gleich eine Einzahlung vornehmen, d.h. es findet die erste Kontobewegung für dieses Konto statt.
- Erstellen Sie auch ein Klassendiagramm und Ordnen Sie die Operationen des Interaktionsdiagramms den Klassen zu.

# Zustandsautomaten



- Konzepte zur Verhaltensmodellierung
  - Beschreibung des Verhalten von Objekten als Folge von durchlaufenen **Zuständen** (*states*) und
  - **Ereignisse** (*events*) und **Bedingungen** (*guards*) lösen **Zustandsübergänge** (*state transitions*) aus
  - **Aktivitäten** (Operationen), die ausgeführt werden
    - bei Zustandsübergängen (***effect***)
    - bei Eintritt in einen Zustand (***entry behavior***)
    - beim Austritt aus einem Zustand (***exit behavior***)
    - während eines Zustands (***do activities***)
  - Zustände können **geschachtelt** werden (*composite states*)
    - Ein Zustand kann **Unterzustände** enthalten
    - Die Unterzustände können in **Regionen** (*regions*) eingeteilt werden
    - Unterzustände können Übergänge von den Oberzuständen „erben“
- Diagrammkopf

  ***stm*** *Diagramm-Name*

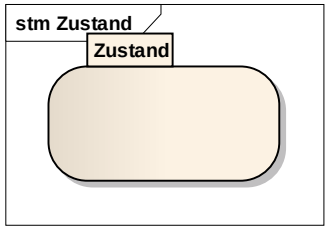
# Zustandsautomaten

- **Zustand**
  - Ein Zustand modelliert eine Situation, in der eine eventuell implizite, invariante Bedingung gilt
- Eigenschaften eines Zustands
  - wird durch einen **Namen** beschrieben
    - Eindeutig innerhalb eines Diagramms
    - Beliebig viele „anonyme“ Zustände ohne Namen möglich
      - diese sind alle implizit verschieden
  - kann Verhalten beinhalten
    - **Eintritts-Aktivität** (*entry behavior*)
      - Wird ausgeführt, wenn der Zustand eingenommen wird
    - **Austritts-Aktivität** (*exit behavior*)
      - Wird ausgeführt, wenn der Zustand verlassen wird
    - **do-Aktivität** (*do-activity)*
      - Wird ausgeführt während des Zustands
        - » startet bei Eintritt in den Zustand
        - » endet bei Verlassen des Zustands oder terminiert selbstständig

# Zustandsautomaten

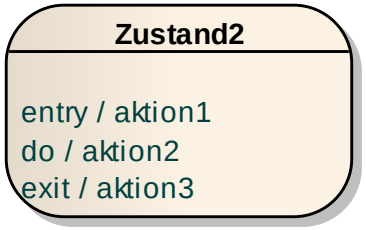
- **Notation Zustand**


Zustand mit Namen




Der Name kann auch in einem „Aktenreiter“ (Register-Tab) am Zustand angebracht werden.

Der Diagrammrahmen enthält das Schlüsselwort ***stm*** oder ***sm***




Zustand mit eigenem Compartment für den Namen und die Aktivitäten

Die UML definiert je **eine** optionale *entry*-, *do*,- und *exit*-Aktivität.

O.B.d.A. ermöglichen die Werkzeuge meist mehrere davon:

→ können zu je einer zusammengefasst werden;

→ Ausführungssemantik: sequentiell „von oben nach unten.

# Zustandsautomaten

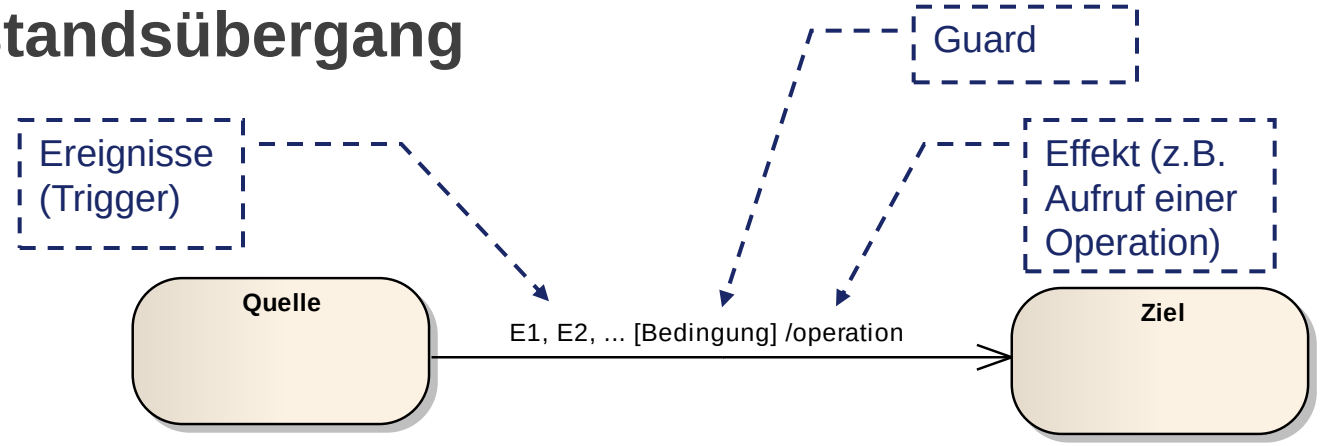


- Definition
  - Ein **Zustandsautomat** *(finite state machine)* besteht aus Zuständen und Zustandsübergängen
    - Ein **Zustand** ist eine Zeitspanne, in der ein Objekt auf ein Ereignis wartet
    - Durch **Ereignisse** (*events*) werden **Zustandsübergänge** (*transitions*) ausgelöst
      - Eine **Zustandsübergang** (Transition) besitzt theoretisch keine Dauer und kann nicht unterbrochen werden
- Anwendung
  - Beschreibung des Verhaltens von Modellelementen
    - z.B. Ablauf von Operationen, Ablauf von Anwendungsfällen
    - Dynamisches Verhalten von Objekten einer Klasse („**Lebenszyklen**“)

- **Zustandsübergang**
  - Gerichtete Beziehung zwischen einer Quelle und einem Ziel
    - Quelle und Ziel sind Zustände
  - So genannte **Trigger** lösen die Zustandsübergänge aus (eine Transition „**feuert**“)
    - Ein Trigger spezifiziert ein **Ereignis** (*event*), das die Ausführung von mit dem Übergang verbundenen Verhalten nach sich ziehen kann
    - Die Ereignisse sind häufig Resultat von Aktionen, die entweder innerhalb des Systems oder auch außerhalb des Systems ablaufen
  - Neben den Triggern kann ein optionaler **Guard** (Bedingung, Einschränkung) den Zustandsübergang steuern
    - Die Transition feuert nur, wenn der Guard erfüllt ist
  - Ein optionales Verhalten (***effect***) kann mit einem Zustandsübergang verbunden sein
    - Der Effekt wird ausgeführt, wenn der Übergang feuert

# Zustandsautomaten

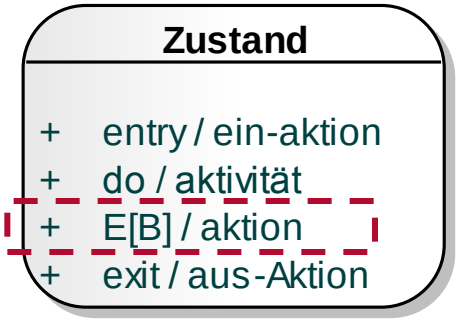
- **Notation Zustandsübergang**



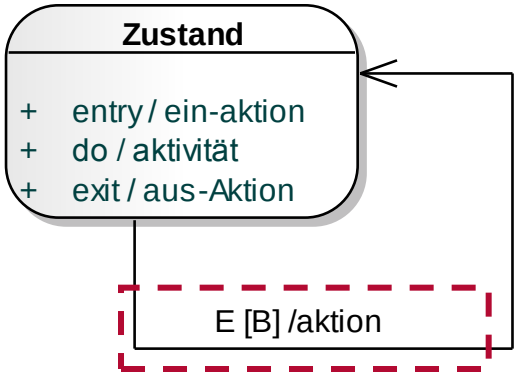

- Ablauf
  - Der Übergang „feuert“, wenn
    - Im Zustand *Quelle* eines der Ereignisse *E1, E2, …* eingetreten ist
    - Und die *Bedingung* erfüllt ist
  - Wenn der Übergang feuert, ist der nächste Zustand *Ziel*
    - „während des Übergangs“ wird *operation* ausgeführt
  - Anmerkungen
    - Ein Trigger und das zugehörige Ereignis werden meist zusammengefasst
      - Wir sprechen i.A. nur von Ereignissen
    - Meist nur ein Ereignis/Trigger
    - Mit einem Zustandsübergang wird keine signifikante Zeitdauer assoziiert
      → Die *operation* sollte keine signifikante Zeit beanspruchen!

# Zustandsautomaten

- **Interne Transitionen**
  - Können alle Merkmale einer Transition haben
    - Trigger
    - Guard
    - Effekt
  - Bewirken keine Zustandsveränderung
    - Falls die Transition feuert, wird nur der Effekt ausgeführt
    - Exit-/entry-/do-Aktivitäten des Zustands werden nicht noch mal ausgeführt




- **Selbst-Transitionen**
  - Quelle und Ziel der Transition sind identisch
  - Der Zustand wird „kurz“ verlassen und gleich wieder eingenommen
    - Exit-/entry-/do-Aktivitäten werden erneut ausgeführt

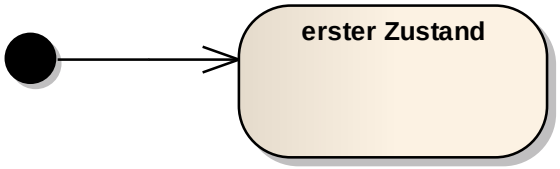


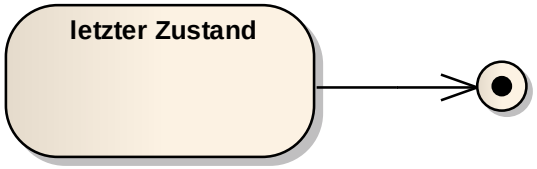
# Zustandsautomaten

- **Pseudozustände**
  - „Transiente“ Knoten
    - kein „Aufenthalt“ im Knoten vorgesehen
  - sind mit „echten“ Zuständen verbunden
  - d.h. eine Transition über mehrere Pseudoknoten hinweg darf keine signifikante Zeitdauer beanspruchen
  - ausgehende Transitionen dürfen keine Trigger haben
    - Ausnahme: Initialzustand
  - Arten von Pseudozuständen
    - Initialzustand
    - Endzustand
    - Auswahl (*choice*)
    - Kreuzung (*junction*)
    - Gabelung (*fork*)
    - Vereinigung (*join*)
    - u.a.m.

# Zustandsautomaten

- Pseudozustände
  - Start- und Endzustand




Der Ablauf beginnt im Zustand ***erster Zustand***

Übergang kann auch mit Trigger/Guard/Effekt versehen sein.

Wenn im Zustand ***letzter Zustand*** der Übergang feuert, ist der Ablauf beendet.

Übergang kann mit Trigger/Guard/Effekt versehen sein

Genau genommen kein Pseudozustand, sondern Spezialfall von Zustand (siehe UML-Spezifikation)

# Zustandsautomaten

- Verarbeitung in einem Zustand
  - ***entry*-Aktivität**
    - wird ausgeführt unmittelbar nach Eintreten in den Zustand
  - ***do*-Aktivität**
    - beginnt, wenn Objekt Zustand einnimmt und endet, wenn es ihn verlässt
    - oder: beginnt, wenn Objekt Zustand einnimmt und terminiert selbstständig
  - ***exit*-Aktivität**
    - wird ausgeführt beim Verlassen des Zustands



Quelle: [Balzert 05], Abb. 2.11-3

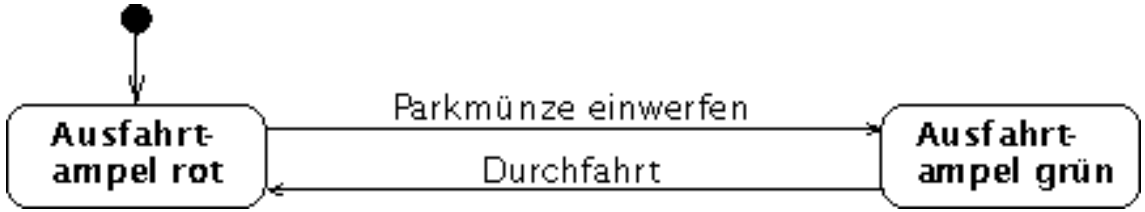
*Anmerkung:*

Normalerweise ist eine sequentialisierter Ablauf erforderlich:

entry-Aktivität → do-Aktivität → exit-Aktivität

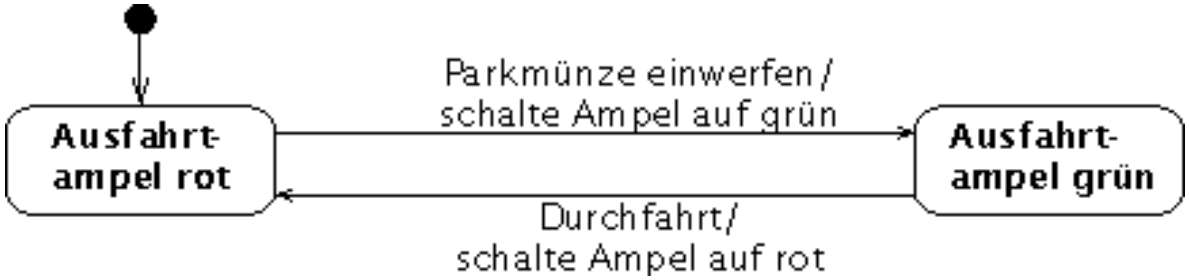
# Zustandsautomaten

- Beispiel: Verhalten der Ausfahrt-Ampel im Parkhaus
  - Einfacher Zustandsautomat:



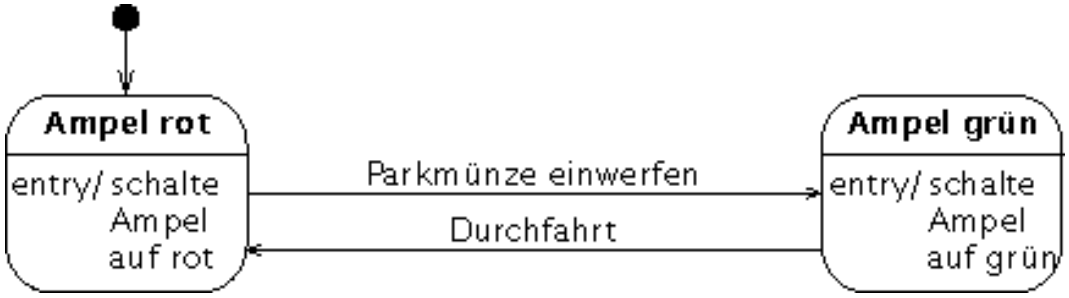

Quelle: [Balzert 05], Abb. 2.11-1

  - Mit Aktionen an Übergängen:




Quelle: [Balzert 05], Abb. 2.11-2

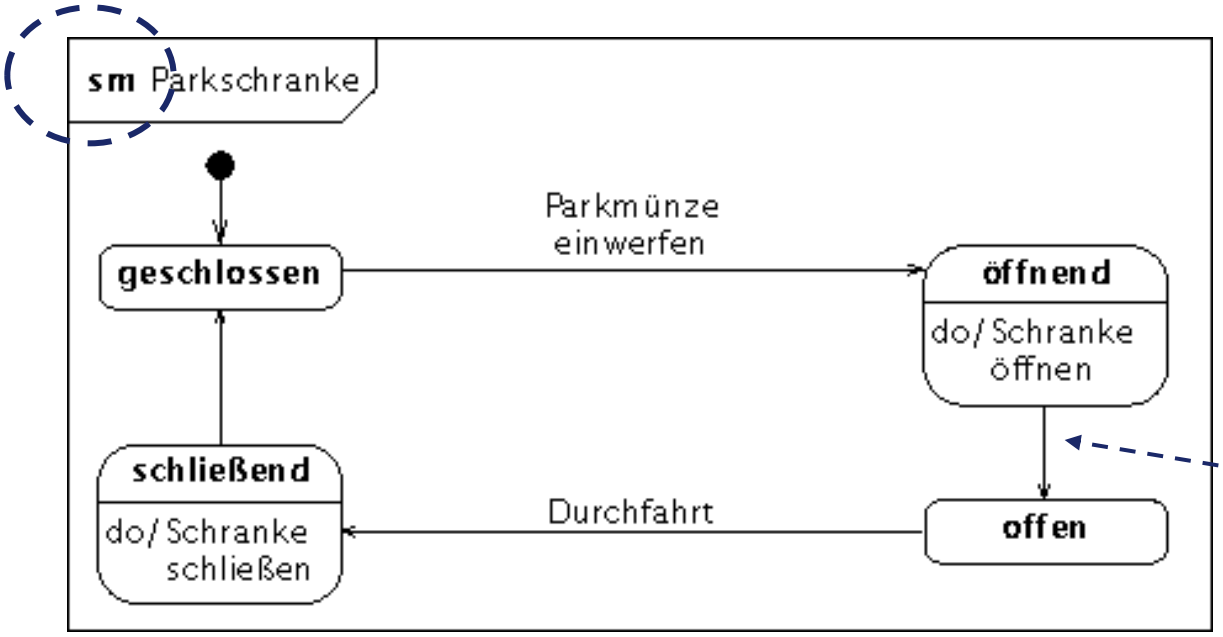
  - Mit entry- und exit-Aktivitäten




Quelle: [Balzert 05], Abb. 2.11-4

# Zustandsautomaten

- Beispiel: Ausfahrt-Schranke im Parkhaus [Balzert]
  - Zustandsautomat mit do-Aktivitäten und **Diagrammrahmen**




**Automatischer Übergang**:

Übergang ohne Trigger und Guard: feuert, wenn Aktivität des Ausgangszustands beendet ist.

*sm = state machine*

Quelle: [Balzert 05], Abb. 2.11-5

- **Guards**
  - Vollständigkeit und Eindeutigkeit
    - Bei mehreren ausgehenden Kanten eines Zustands mit Guards muss genau ein Guard erfüllt sein.
  - Seiteneffektfreiheit
    - Guards dürfen keine „Seiteneffekte“ haben, insbesondere keine Effekte auf Werte, die die Gültigkeit von anderen Guards beeinflussen.
  - Zustandsabfragen
    - In Guards können Bedingungen der Form

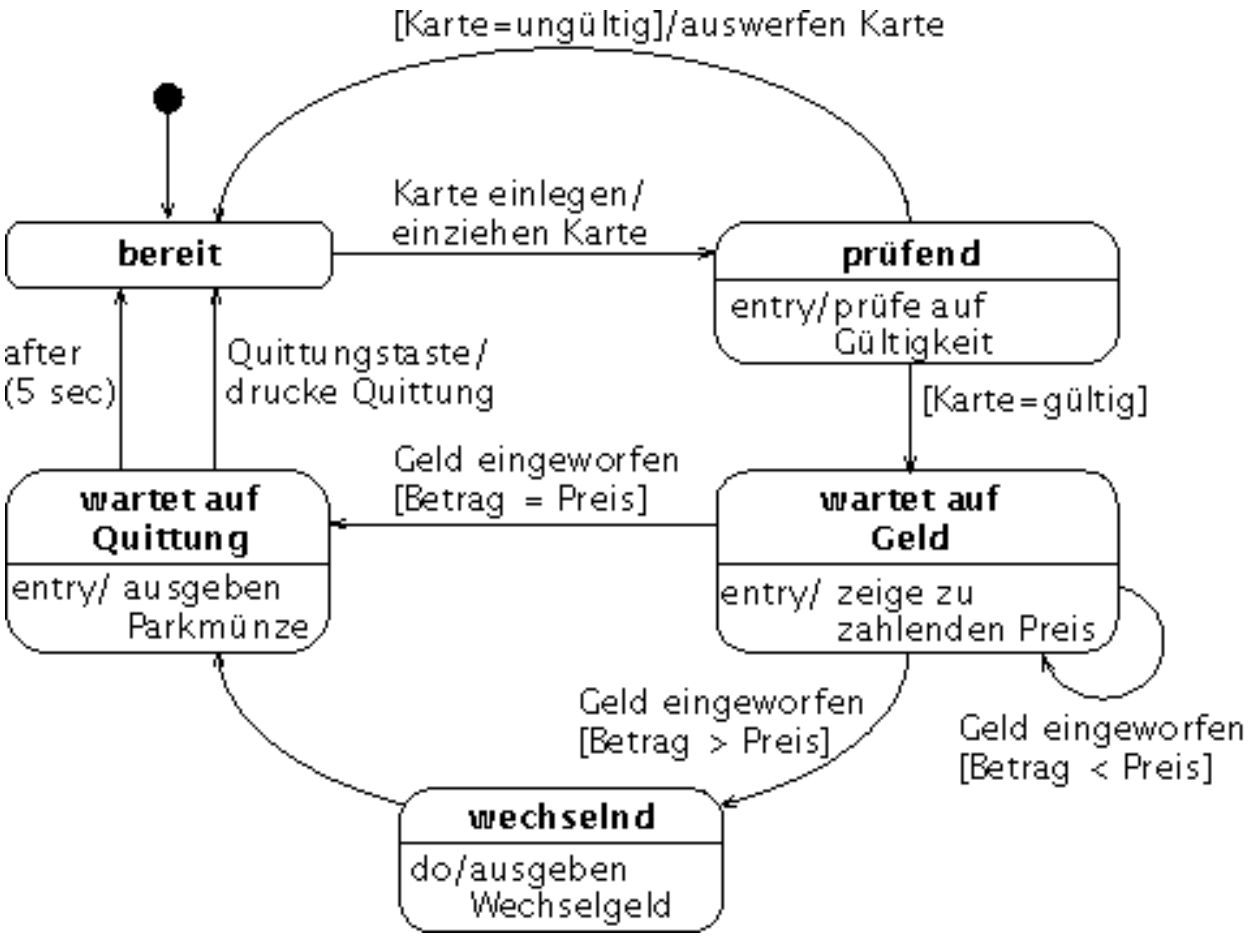
      ***in*** *(zustand)*

      ***not in*** (*zustand*)

      verwendet werden.
    - Sie sind erfüllt, wenn sich der Automat (nicht) im Zustand *zustand* befindet
    - Werden meist angewendet auf "Unterzustände" in zusammengesetzten Zuständen (s.u.)

# Zustandsautomaten

- Beispiel: Bezahlen einer Parkgebühr am Automaten




Quelle: [Balzert 05], Abb. 2.11-7

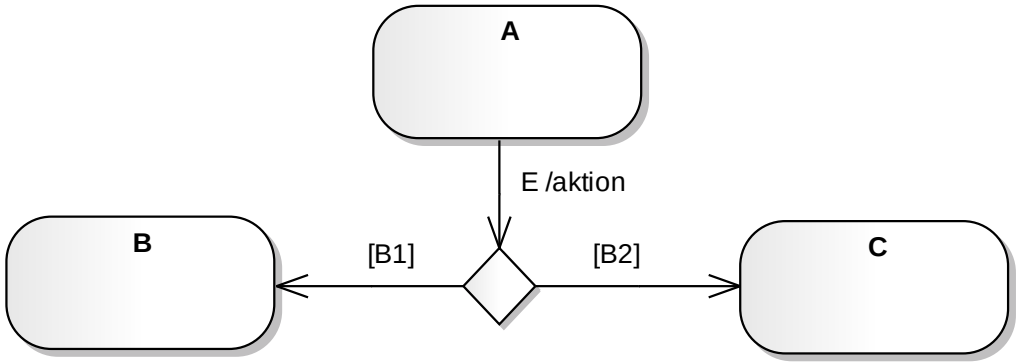
- **Ereignisarten**
  - Aufrufereignis (*CallEvent*)
    - Wird durch den Aufruf einer Operation ausgelöst
      - z.B. Operationen der Klasse, dessen Verhalten beschrieben wird.
    - Es können Parameter (Ereignisattribute) übermittelt werden
      - Entsprechen einer Zuweisung der Ereignisattribute an Attribute des Objekts, dessen Verhalten beschrieben wird
      - Können in nachfolgenden Guards und Aktivitäten wieder verwendet werden
    - Notation: ***name(Parameter1, Parameter2,…)***
  - Signal (*SignalEvent*)
    - Ein Signal wird empfangen
    - Parameter und Notation wie *CallEvent*
  - Bedingung (*ChangeEvent*)
    - Ein Boole‘scher Wert wird *true*
    - Alternative zu Gurads
    - Notation: ***when(Boolescher_Ausdruck)***

# Zustandsautomaten

- Ereignisarten
  - Zeitereignis (*TimeEvent*)
    - Relatives Zeitereignis:
      - tritt nach einer in einem Zustand verstrichenen Zeit ein
      - Notation: ***after(Zeitraum)***
    - Absolutes Zeitereignis:
      - tritt zu einem bestimmten Zeitpunkt ein
      - Notation: ***at(Zeitpunkt)***
  - *AnyReceiveEvent*
    - Ein Ereignis wird empfangen, das keinem anderen Trigger an ausgehenden Kanten entspricht
    - Notation: ***all***
  - Implizites Ereignis („automatischer Übergang“)
    - Übergang ohne Trigger
    - Transition wird ausgeführt, wenn die mit dem Zustand verbundene Verarbeitung beendet ist (vgl. S. 77)

# Zustandsautomaten

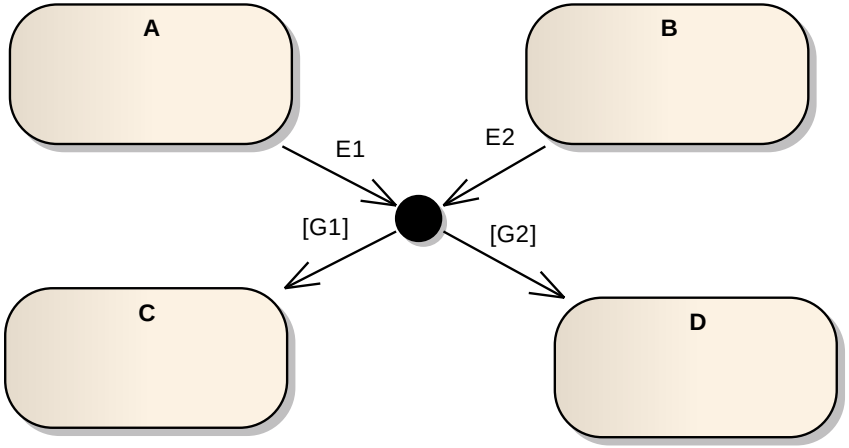
- Pseudozustände
  - Entscheidung (***choice***)
    - Mindestens ein eingehender und ein ausgehender Übergang
    - Ausgehende Kanten dürfen keine Trigger haben.
    - Ausgehende Kanten können Guards haben, deren Auswertung von Effekten an den eingehenden Übergängen abhängt
      - „Dynamische Bedingte Verzweigung“




- Übergang vom Entscheidungsknoten nach B, falls B1 erfüllt.
- Übergang von Entscheidungsknoten nach C, falls B2 erfüllt.
- B1 und B2 müssen sich gegenseitig ausschließen und entweder B1 oder B2 muss gelten.
- Die Auswertung von B1 und B2 kann abhängen vom Effekt *aktion*

# Zustandsautomaten

- Pseudozustände
  - Kreuzung (***junction***)
    - Mindestens ein eingehender und ein ausgehender Übergang
    - Ausgehende Kanten dürfen keine Trigger haben!
    - Ausgehende Kanten können Guards haben, deren Auswertung **nicht** von Effekten an den eingehenden Übergängen abhängen darf
      - „Statische Bedingte Verzweigung“



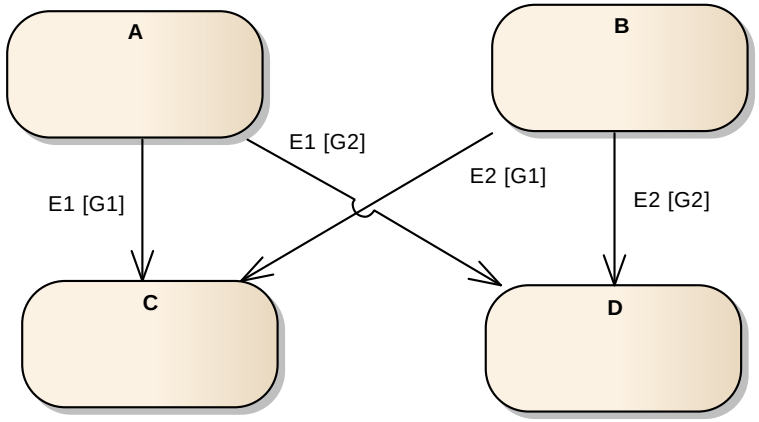




Diagramm mit Kreuzung

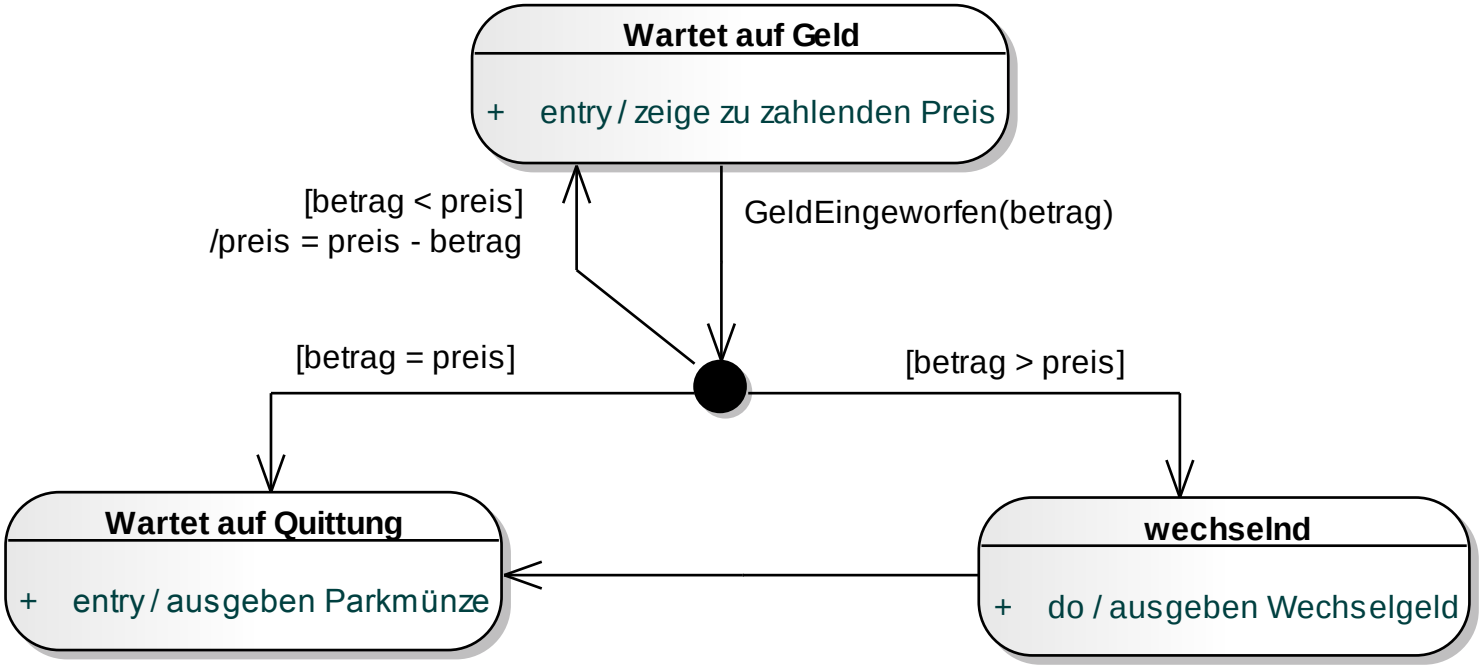
- Die Auswertung von G1 und G2 muss bei Verlassen der Zustände A bzw. B möglich sein

Äquivalentes Diagramm ohne Kreuzung:

- Kombination von Transitions-Pfaden, z.B. unterschiedliche Trigger mit unterschiedlichen Guards

# Zustandsautomaten

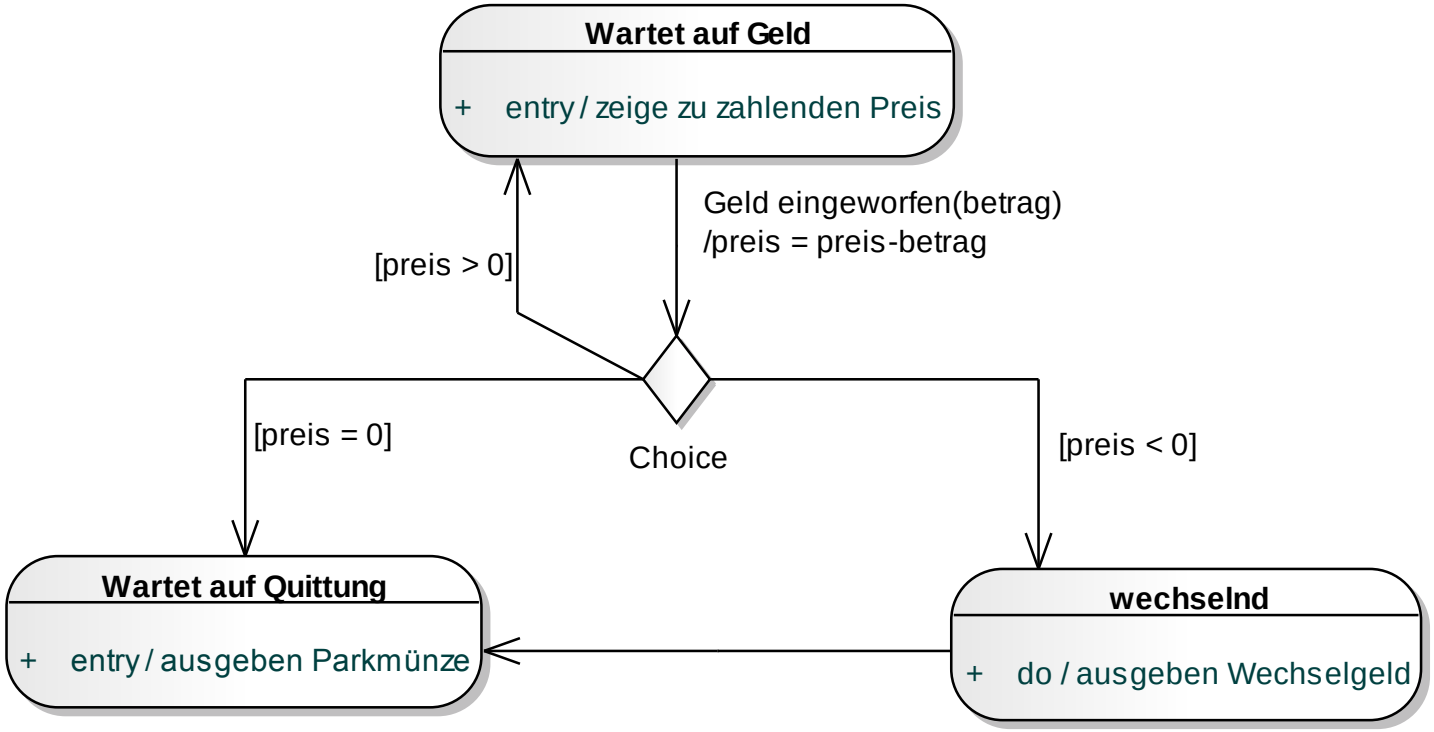
- Beispiel: Bezahlen einer Parkgebühr am Automaten (vgl. S. 79)
  - Mit Kreuzung




- Bei verlassen des Zustands *Wartet auf Geld* durch das Ereignis *Geld eingeworfen(betrag)* steht das Ergebnis die Auswertung der Guards bereits fest.
- Das Diagramm kann in ein äquivalentes Diagramm ohne die Kreuzung unter Verwendung der selben Zuständen, Trigger, Effekte und Guards umgewandelt werden

# Zustandsautomaten

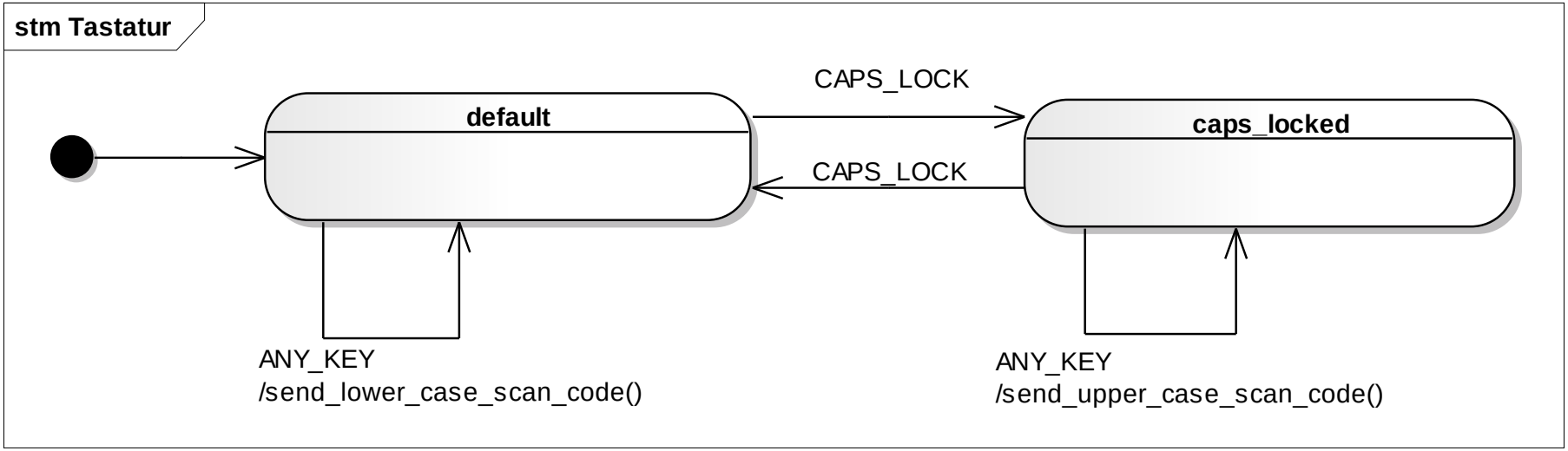
- Beispiel: Bezahlen einer Parkgebühr am Automaten (vgl. S. 79)
  - Mit Entscheidung




- Bei verlassen des Zustands *Wartet auf Geld* durch das Ereignis *Geld eingeworfen* steht das Ergebnis die Auswertung der Guards noch nicht fest (erst nach Ausführung des Effekts *preis = preis - betrag*.
- Das Diagramm kann nicht in ein äquivalentes Diagramm ohne die Entscheidung unter Verwendung der selben Zuständen, Trigger, Effekte und Guards umgewandelt werden

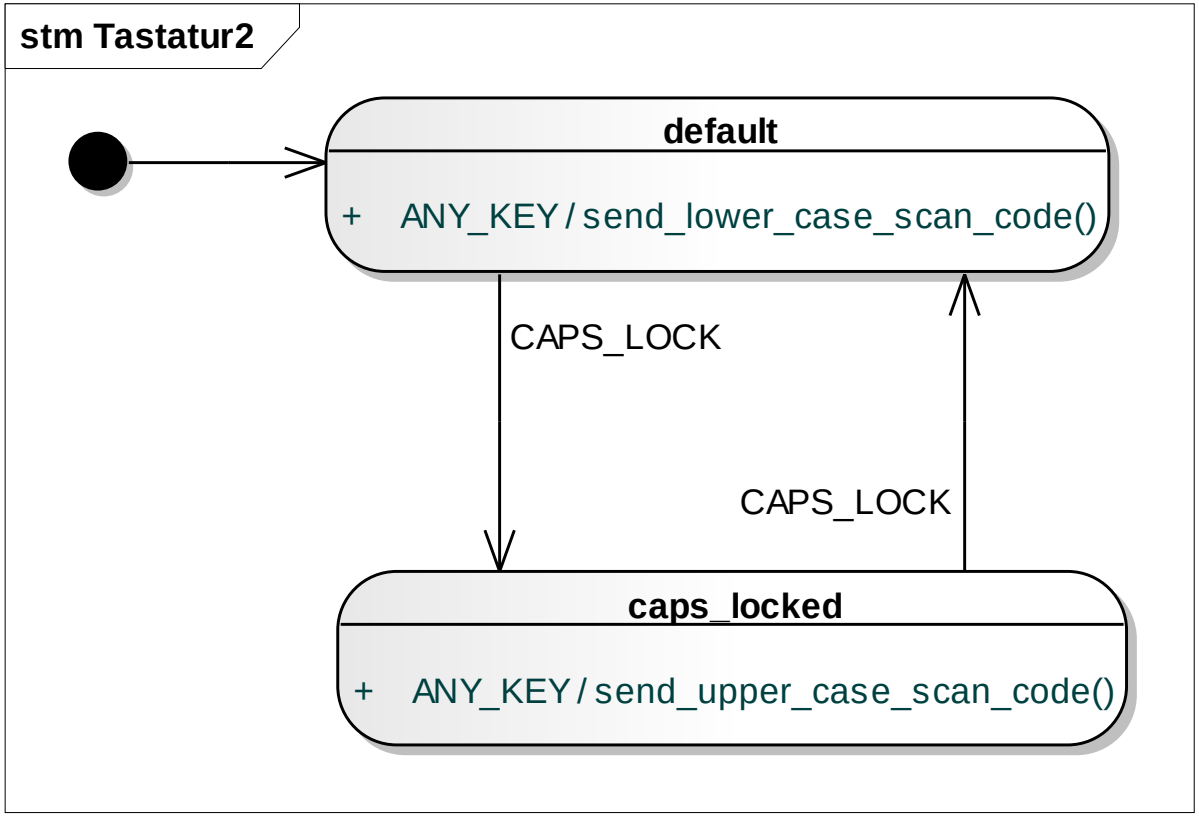
# Zustandsautomaten

- Beispiel: Verhalten einer Computer-Tastatur [Samek 09]
  - Stark Vereinfacht:
    - „Normale“ Zeichentasten
    - Eine Steuerungstaste: *CapsLock* („Feststelltaste“)
    - Ereignisse:
      - CAPS_LOCK: die CapsLock-Taste wurde gedrückt
      - ANY_KEY: eine normale Taste wurde gedrückt



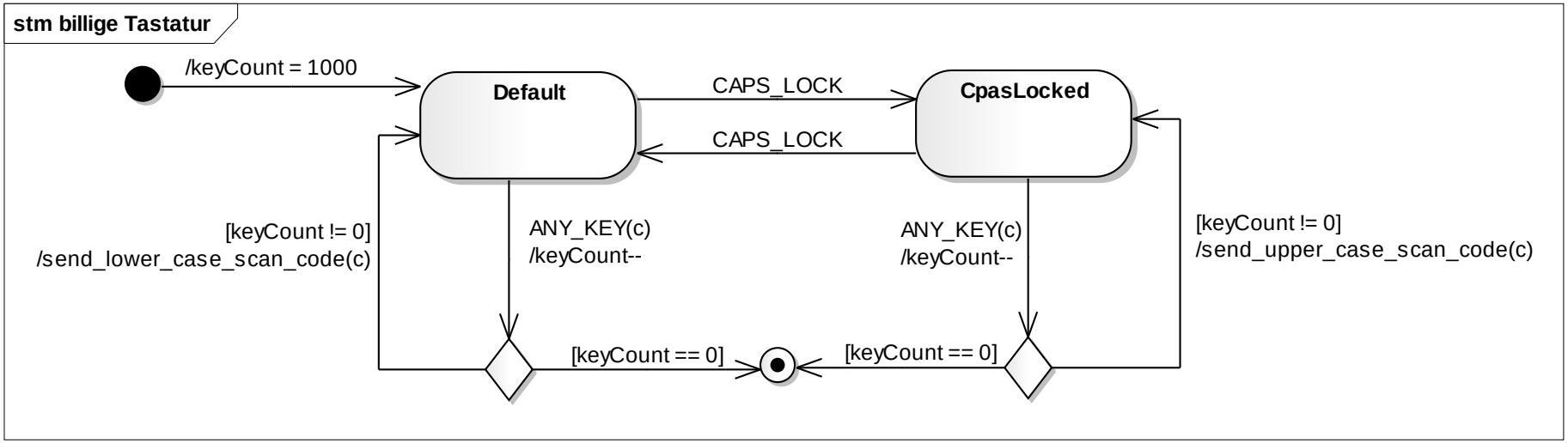
# Zustandsautomaten

- Beispiel: Verhalten einer Computer-Tastatur [Samek 09]
  - Mit internen Transitionen:



# Zustandsautomaten

- Beispiel: Eine billige Computer-Tastatur [Samek 09]
  - Maximal 1000 Tastenanschläge sind möglich
  - Verwenden von „globalen Variablen“ in Guards und Effekten
    - z.B. Variablen/Attribute des Objekts, dessen Verhalten beschrieben wird
    - in [Samek 09] „**erweiterte Zustandsautomaten**“ genannt
    - Vorsicht: nicht zuviel globale Variablen und Guards!
      → Spagetti-Code!
  - Verwenden von Ereignisattributen und Weitergabe an Aktivitäten



# Zustandsautomaten

- Anmerkungen
  - Ausführungsmodell: *Run To Completion (RTC)*
    - Ein Ereignis wir immer erst vollständig verarbeitet, bevor ein anderes Ereignis verarbeitet werden kann (Ereignisverarbeitung kann nicht unterbrochen werden)
  - „erweiterte Zustandsautomaten“ [Samek 09]
    - **Qualitative Zustände**: die Zustände
    - **Quantitative Zustände**: die Werte von (globalen) Variablen bzw. Objektattributen (vgl. S. 85, S. 88)
    - Zustandsübergänge hängen von qualitativen Zuständen (dem jeweils aktuellen) und quantitativen (in Guards) ab
    - Man kann qualitative durch quantitative ausdrücken und umgekehrt
    - Design-Aufgabe: geschicktes wählen von quantitativen und qualitativen Zuständen
  - Klassische Zustandsautomaten beinhalten die bisher behandelten Konzepte

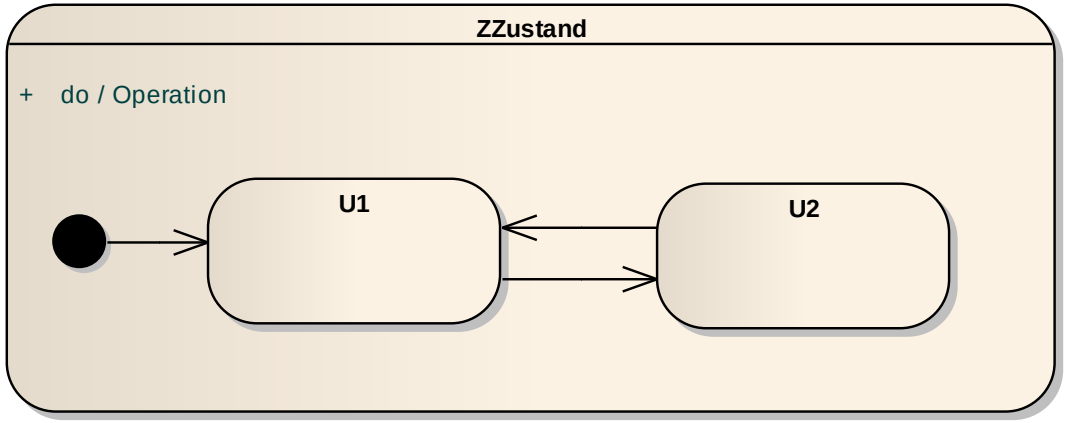
# Zustandsautomaten



- Erweiterung der klassischen Zustandsautomaten
  - Zustandsdiagramme nach [Harel 87] (***Statecharts***) erweitern die klassischen Zustandsautomaten um
    - Hierarchische Strukturierung
    - Parallelität
    - Kommunikation
  - Vorteile
    - Modellierung komplexer Systeme in unterschiedlichen Detaillierungsgraden
    - Modularer Aufbau von Verhaltensmodellen
    - Wiederverwendung von Verhaltensmodellen in verschiedenen Kontexten
  - Anwendungsmöglichkeiten lt. [Harel 87]
    - Spezifikation und Entwurf diskreter ereignisgesteuerter Systeme, z.B.
      - mehr-Rechner Echtzeitsysteme
      - Kommunikationsprotokolle
      - Digitale Steuereinheiten

# Zustandsautomaten

- **Zusammengesetzte Zustände**
  - Ein **zusammengesetzter Zustand** enthält eine **Region** oder mehrere **orthogonale Regionen**.
  - Jede Region enthält eine Menge von Zuständen und Transitionen
  - Ein Zustand, der zu einer Region eines Zusammengesetzten Zustands gehört, heißt **Unterzustand** des Zusammengesetzten Zustands
  - Notation
    - Zusammengesetzter Zustand mit **einer** Region

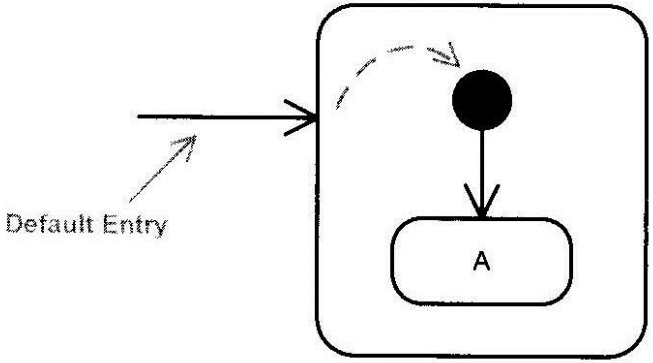


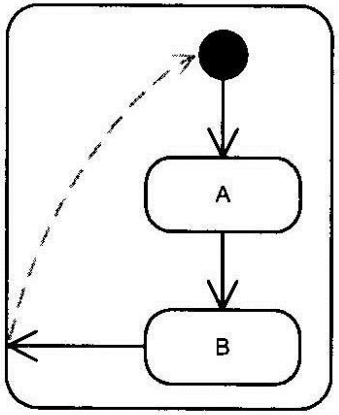

Ein System im Zustand ***ZZustand*** ist **entweder** im Unterzustand ***U1*** **oder** im Unterzustand ***U2***.

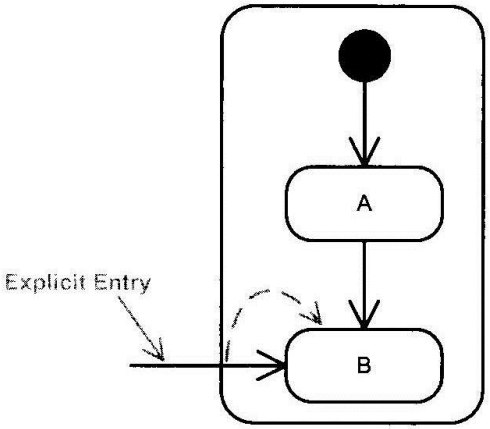
Bei Eintreten in den Zustand ***ZZustand*** wird als erstes der Unterzustand ***U1*** eingenommen.

# Zustandsautomaten

- Ablauf: Betreten eines zusammengesetzten Zustands
  (Quelle: [Rupp et al. 07], Abb. 14.47 – 14.49)




***Default Entry (1):***

Transition von außen in den Zusammengesetzten Zustand

→ Übergang in den Start-Unterzustand

***Default Entry (2):***

Transition von einem Unterzustand in den Zusammengesetzten Zustand

→ Übergang in den Start-Unterzustand

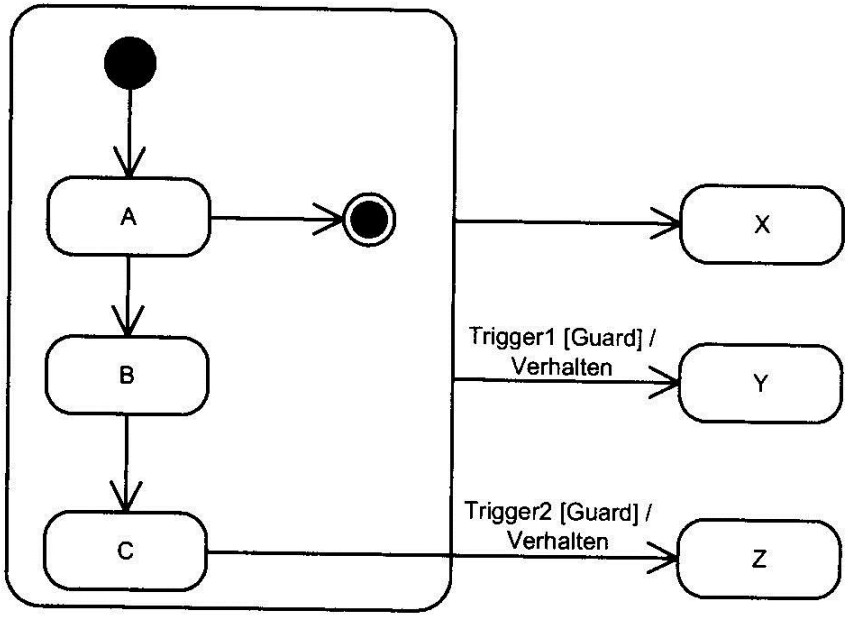
***Explicit Entry:***

Transition von außen in einem Unterzustand

→ Übergang in diesen Unterzustand

# Zustandsautomaten

- Ablauf: Verlassen eines zusammengesetzten Zustands
  (Quelle: [Rupp et al. 07], Abb. 50)




***Erreichen des Endzustands:***

→ Der automatische (triggerlose) Übergang feuert und es erfolgt ein Übergang nach X

***Trigger am Übergang vom z. Zustand:***

→ Jeder Unterzustand „erbt“ diesen Übergang

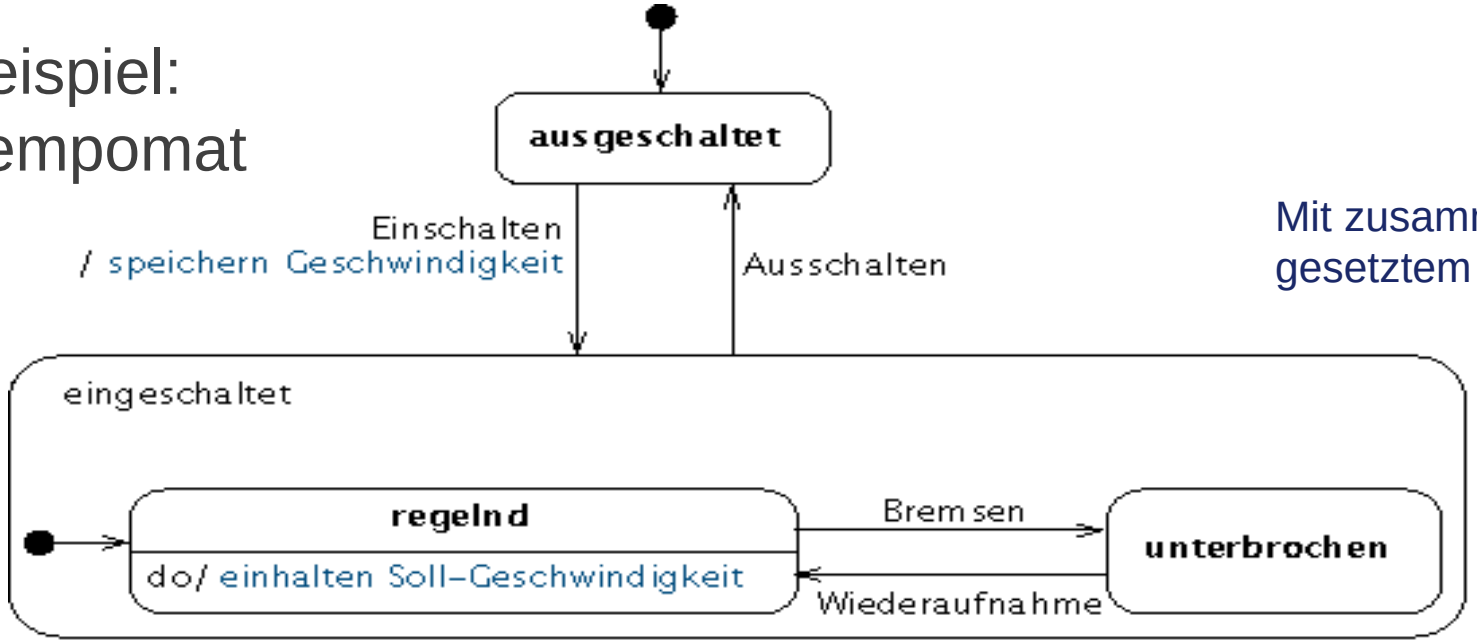
→ Egal in welchem Unterzustand sich das System befindet, es erfolgt ein Übergang nach Y, falls der Übergang feuert.

***Trigger am Übergang von einem Unterzustand:***

→ Falls sich das System im Unterzustand C befindet und der Übergang feuert, ist das System im Zustand Z.

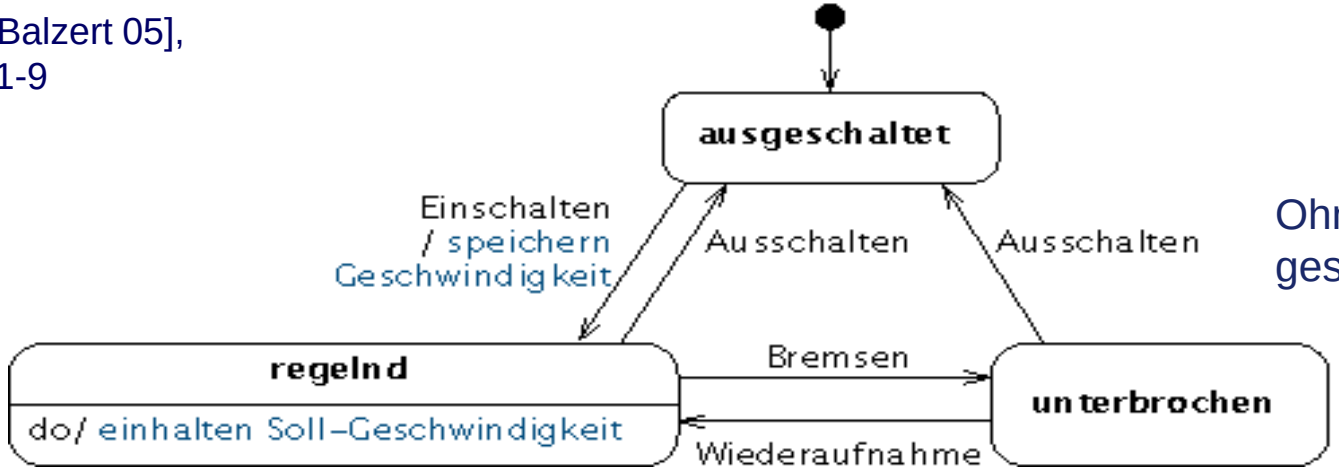
# Zustandsautomaten

- Beispiel: Tempomat




Mit zusammengesetztem Zustand

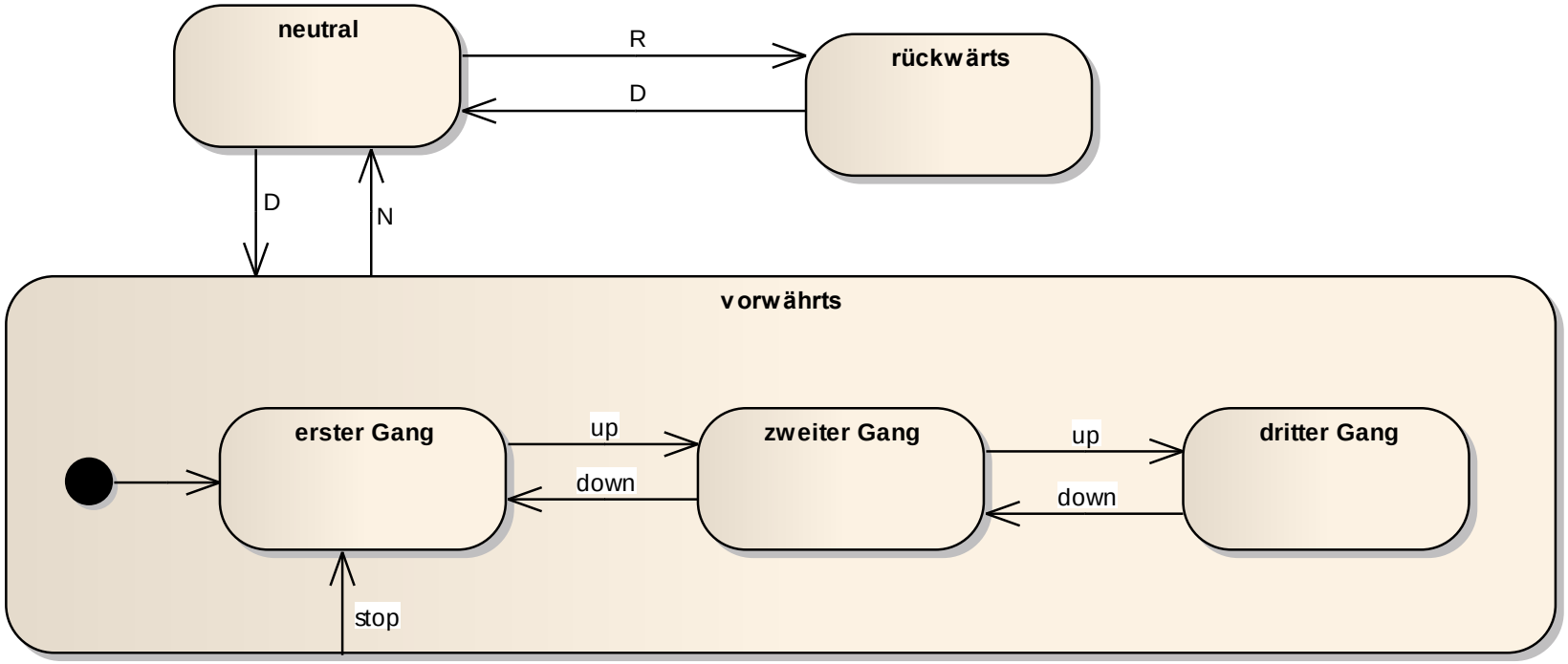
Quelle: [Balzert 05], Abb. 2.11-9




Ohne zusammengesetzten Zustand

# Zustandsautomaten

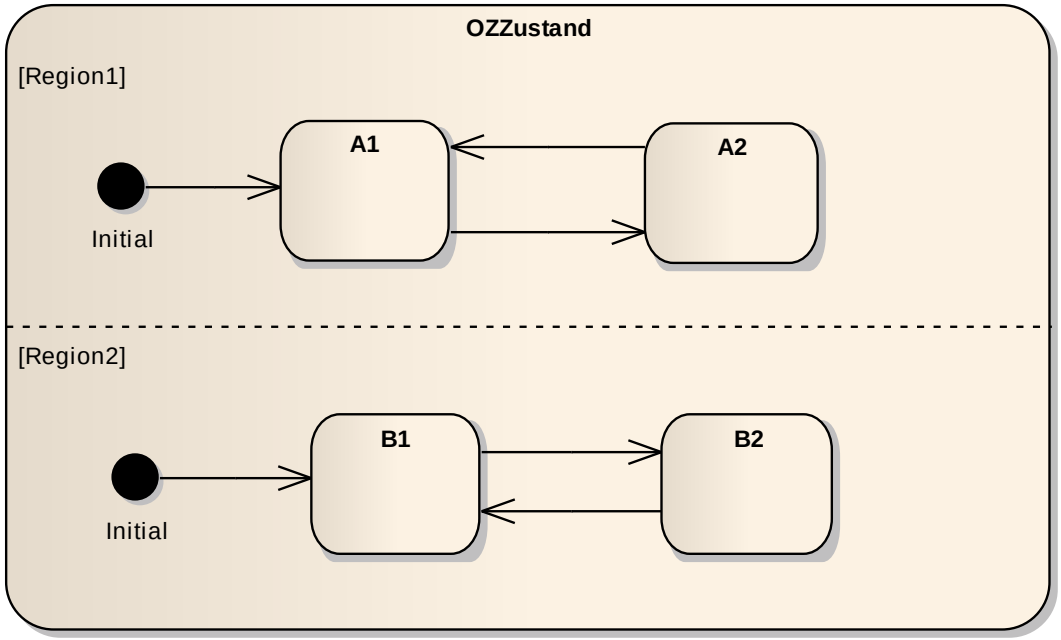
- Beispiel: automatisches Getriebe




***Ereignisse:***

- Stellen des Schalthebels: N (neutral), R (rückwärts), D (vorwärts fahren)
- Impulse des Getriebes: up (hochschalten), down (runterschalten)
- Fahr-Aktionen: stop (bremsen bis zum Stillstand)

- **Orthogonale zusammengesetzte Zustände**
  - Enthalten mehrere Regionen
    - Bei Betreten eines orthogonalen zusammengesetzten Zustands laufen alle Regionen unabhängig voneinander („nebenläufig“, concurrent) ab.
  - Notation:

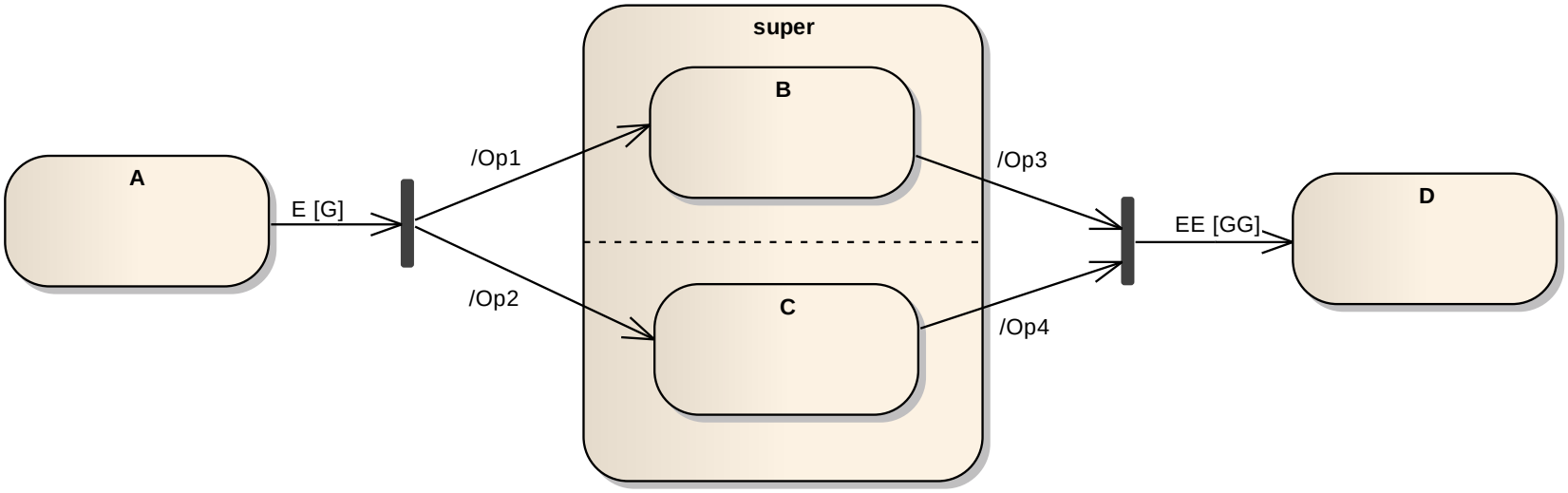



Bei Eintreten in den Zustand ***OZZustand*** laufen die Automaten der beiden Regionen unabhängig voneinander („nebenläufig“) ab.

Ein System im Zustand ***OZZustand*** ist einem Zustand aus der ***Region1*** **und** in einem Zustand aus der ***Region2.***

# Zustandsautomaten

- Pseudozustände
  - Gabelung und Vereinigung (***fork*** *and* ***join***)




**Gabelung (*fork*)**:

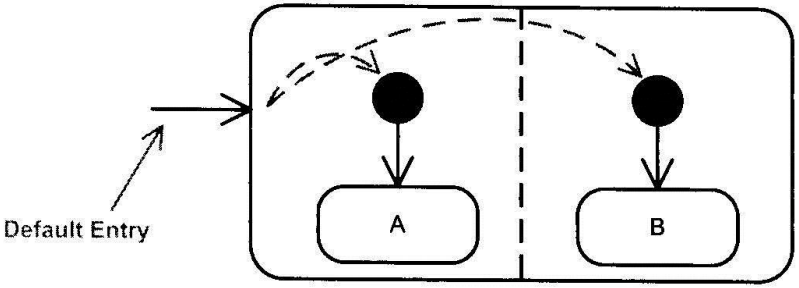
- Verzweigung in zwei parallele Abläufe
- Ausgehende Kanten dürfen weder Trigger noch Guards haben!
- Ausgehende Kanten müssen in orthogonalen Regionen enden

**Vereinigung (*join*)**:

- Zusammenführung zweier paralleler Abläufe
- Eingehende Kanten dürfen weder Trigger noch Guards haben!
- Eingehende Kanten müssen aus zwei orthogonalen Regionen kommen.

# Zustandsautomaten

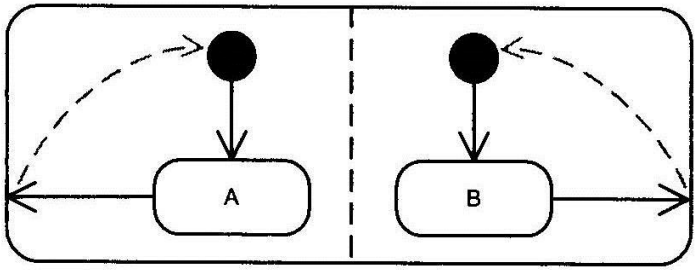
- Ablauf: Betreten eines orthogonalen zusammengesetzten Zustands (Quelle: [Rupp et al. 07], Abb. 14.60 – 14.61)




***Default Entry (1):***

Transition von außen in den orthogonalen Zusammengesetzten Zustand

→ Übergang in die Startzustände in jeder Region

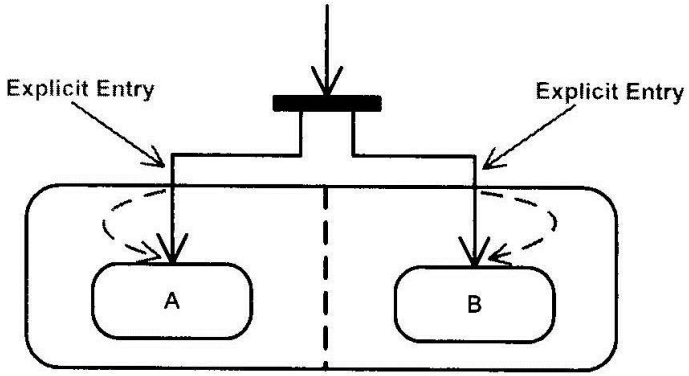



***Default Entry (2):***

Transition von einem Unterzustand in den Zusammengesetzten Zustand

→ Übergang in die Start-Unterzustände

→ Eine bereits beendete Region „wartet“ auf die noch arbeitende.



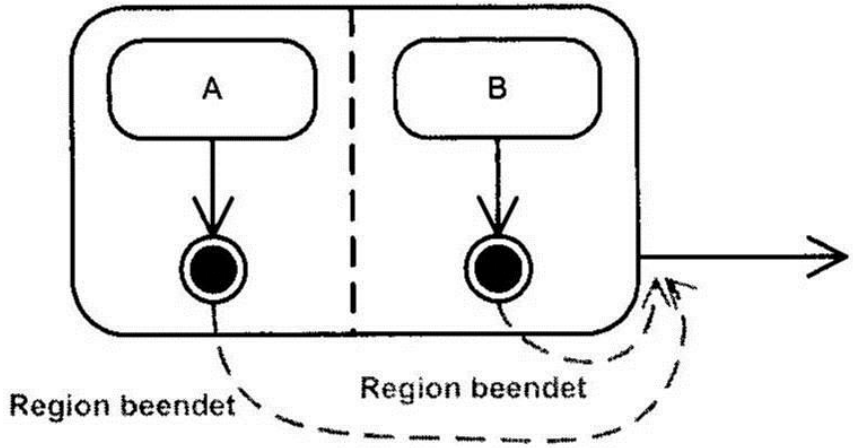

***Explicit Entry:***

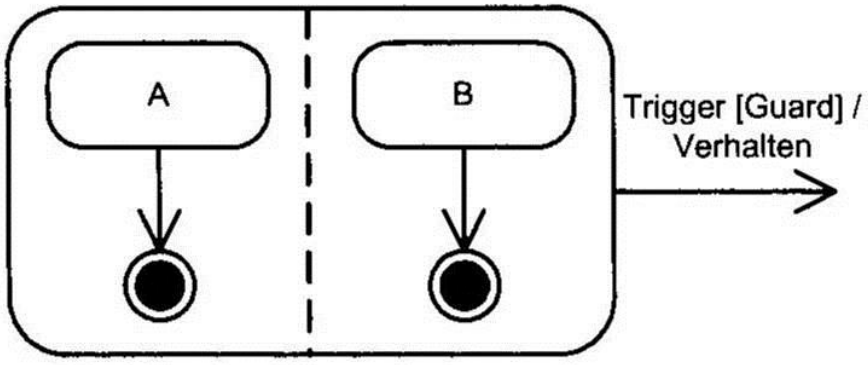
Parallele Transition von außen in einen Unterzustand in jeder Region

→ Übergang in die jeweiligen Unterzustände

# Zustandsautomaten

- Ablauf: Verlassen eines orthogonalen zusammengesetzten Zustands (Quelle: [Rupp et al. 07], Abb. 14.63 + 14.64)




***Erreichen des Endzustands:***

Jede Region hat den Endzustand erreicht

→ Wenn jede Region den Endzustand erreicht hat, feuert der automatische (triggerlose) Übergang

→ Regionen, die den Endzustand früher erreichen, „warten“, bis alle Regionen den Endzustand erreicht haben.

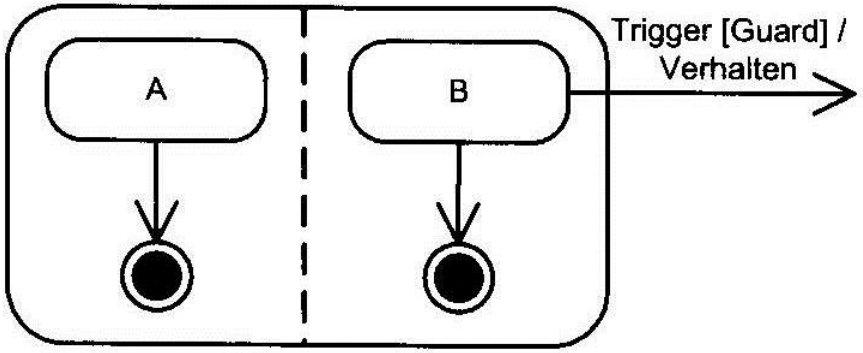
***Trigger am Übergang vom o. z. Zustand:***

→ Jeder Unterzustand „erbt“ diesen Übergang

→ Wenn der Übergang feuert, werden alle Regionen beendet (egal in welchem Unterzustand).

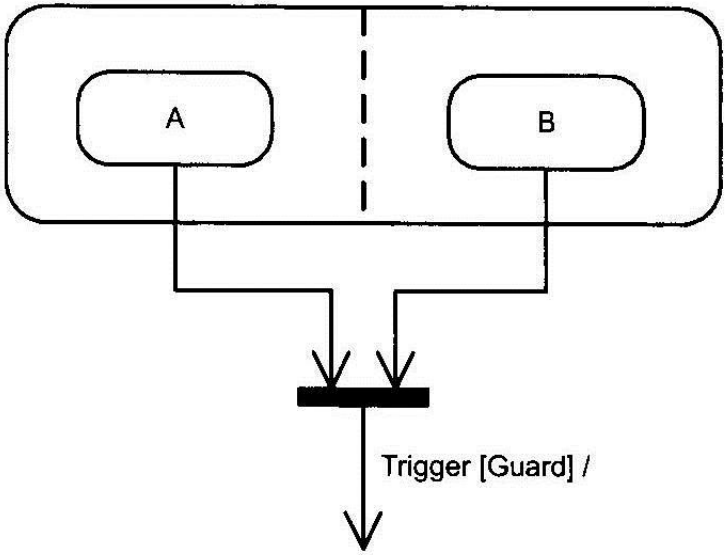
# Zustandsautomaten

- Ablauf: Verlassen eines orthogonalen zusammengesetzten Zustands (Quelle: [Rupp et al. 07], Abb. 14.65 + 14.66)




***Trigger an einem Unterzustand:***

→ Wenn im Zustand B der Übergang feuert, werden alle Regionen beendet.




***Verlassen des o. Zustands durch Vereinigung:***

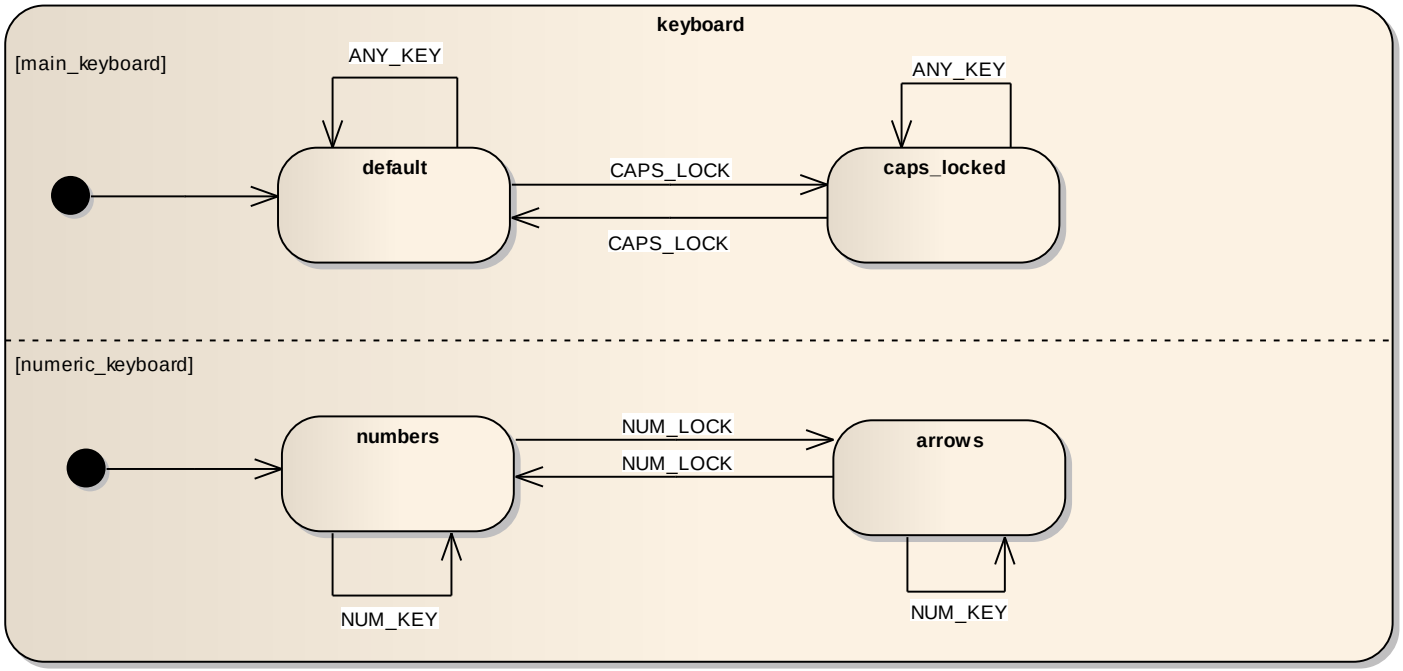
→Wenn die Zustände A und B erreicht sind und der Trigger ausgelöst wird, werden alle Regionen beendet und der Übergang feuert

→Die Regionen „warten“ aufeinander

*Bachte:*
an den von A und B ausgehenden Transitionen sind keine Trigger erlaubt!

# Zustandsautomaten

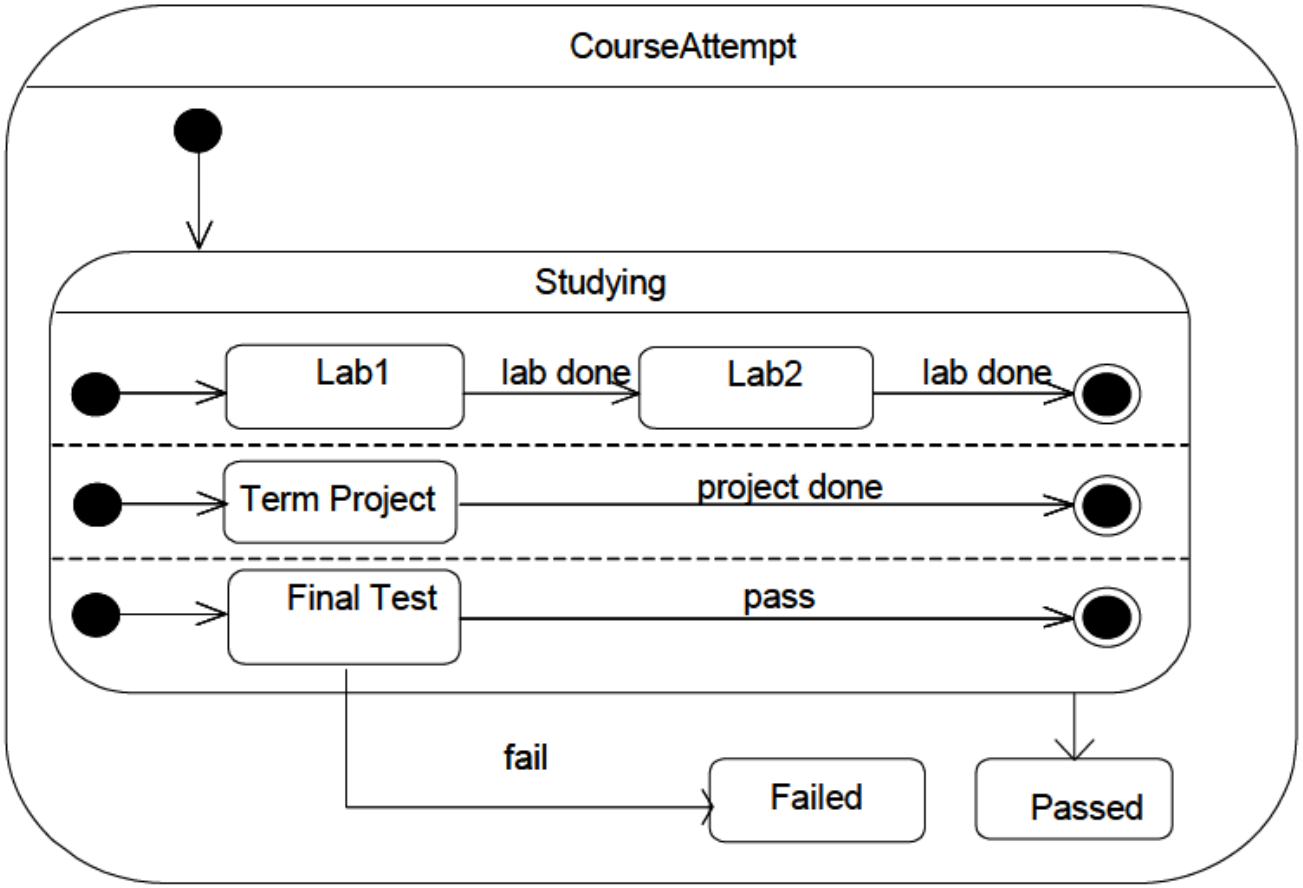
- Beispiel: Tastatur mit Nummerntastenfeld [Samek]
  - Zwei orthogonale Regionen
    - Haupttastenfeld und Nummerntastenfeld
  - Beide Regionen laufen parallel ab
    - Auf Ereignisse ANY_KEY und CAPS_LOCK reagiert nur das Haupttastenfeld
    - Auf Ereignisse NUM_KEY und NUM_LOCK reagiert nur das Nummerntastenfeld



# Zustandsautomaten

- Beispiel: der Versuch, eine Lehrveranstaltung zu bestehen

  (Quelle: [UML], Abb. 15.35)



# Literatur




- [Samek 09] Miro Samek: Practical UML Statecharts in C/C++. Elsevier, 2009, Kapitel 2
- [Harel 87] David Harel: Statecharts: A Visual Formalism for Complex Systems.
  Science of Computer Programming, 8, 1987, pp. 231-274,
  www.wisdom.weizmann.ac.il/~dharel/SCANNED.PAPERS/Statecharts.pdf