

33. Lebenszyklen einfacher Objekte: Aktionsdiagramme (Aktivitäten-, Zustandsdiagramme)

1

Prof. Dr. rer. nat. Uwe Aßmann
Institut für Software- und
Multimediatechnik
Lehrstuhl Softwaretechnologie
Fakultät für Informatik
TU Dresden
Version 15-1.1, 22.06.15

- 1) Aktivitätendiagramme in UML
 - 2) Zustandsdiagramme in UML
 - 3) Verhaltens-, Steuerungs-, und Protokollmaschinen
 - 4) Implementierung von Steuerungsmaschinen
 - 5) Einsatz im Test
- A1) Andere Notationen
A2) Impl. von Protokollmaschinen



Obligatorische Literatur

2

- ▶ Zuser 7.5.3
- ▶ Störrle Kap. 10 (Zustandsdiagramme), Kap. 11 (Aktivitätsdiagramme)
- ▶ ST für Einsteiger: Kap. 10

Überblick Teil III:

Objektorientierte Analyse (OOA)

3

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

Punktweise und querschneidende dynamische Verfeinerung

4

Punktweise funktionale Verfeinerung ist eine funktionale Verfeinerung eines Modellfragmentes (meist Objekt oder Methode), die *punktweise* geschieht, d.h. pro Modellfragment separat durchgeführt wird.

Kap. 33

- ▶ Ergebnis:
 - **Lebenszyklus** des Objekts oder komplexen Objektes
 - **Implementierung** einer Methode

Querschneidende funktionale Verfeinerung ist eine funktionale Verfeinerung *mehrerer* Modellfragmente gleichzeitig, die *querschneidend* geschieht

Kap. 34

- ▶ Damit kann man das Zusammenspiel mehrerer Objekte oder Methoden untersuchen, eine *Szenarienanalyse*, die quasi die Draufsicht auf ein Szenario ermittelt
 - Siehe Kapitel “35-Szenarienanalyse”



33.1. Aktivitätsdiagramme

5

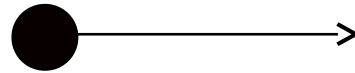
Aktionsdiagramme: Aktivitätsdiagramme (AD),
Statecharts (SC)

Start- und Endzustand

6

- ▶ Jedes Aktionsdiagramm (Statechart, Aktivitätendiagramm) sollte einen eindeutigen Startzustand haben. Der Startzustand ist ein "Pseudo-Zustand".

- ▶ **Notation:**



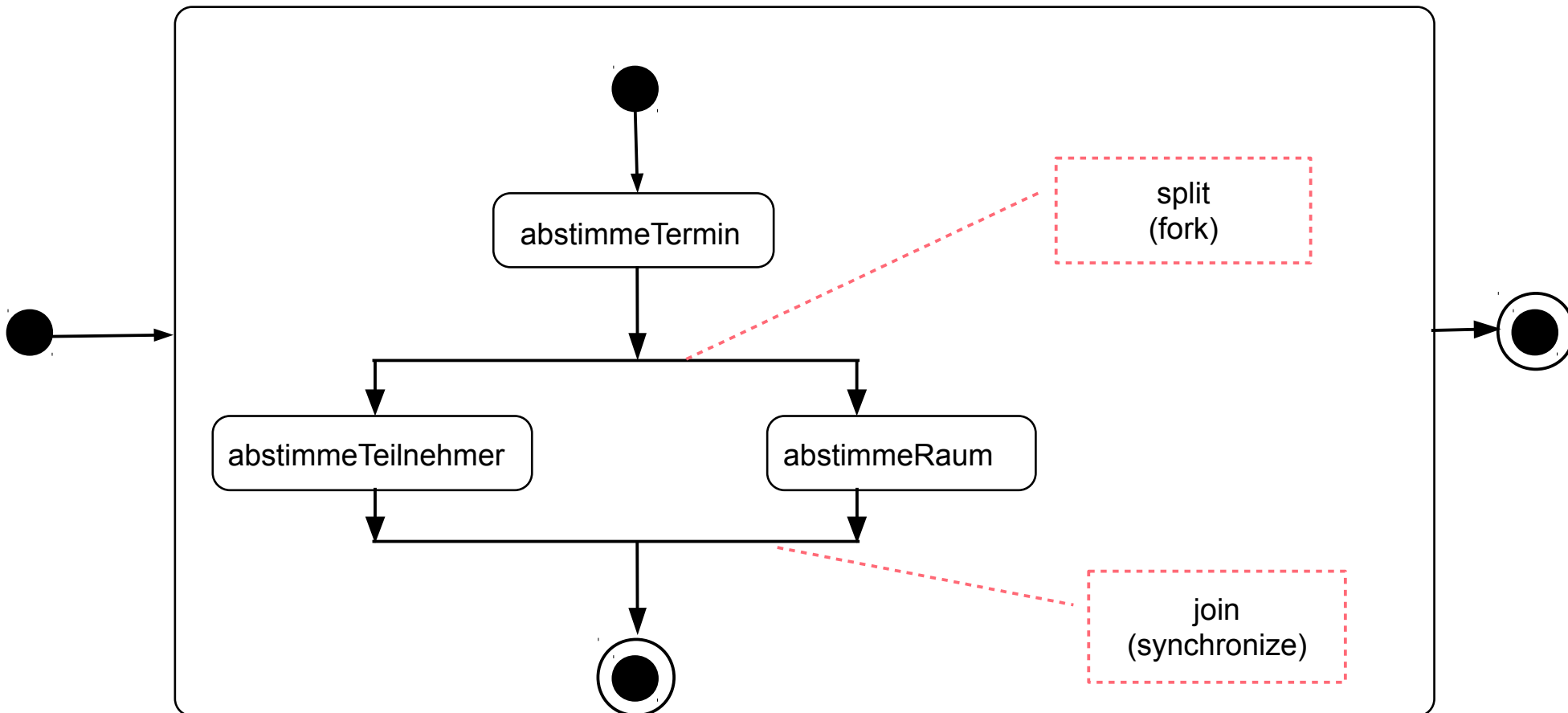
- ▶ Ein Aktionsdiagramm kann einen oder mehrere Endzustände haben.
- ▶ **Notation:** ("*bull's eye*")



Dynamische Modellierung (Verhaltensmodellierung)

7

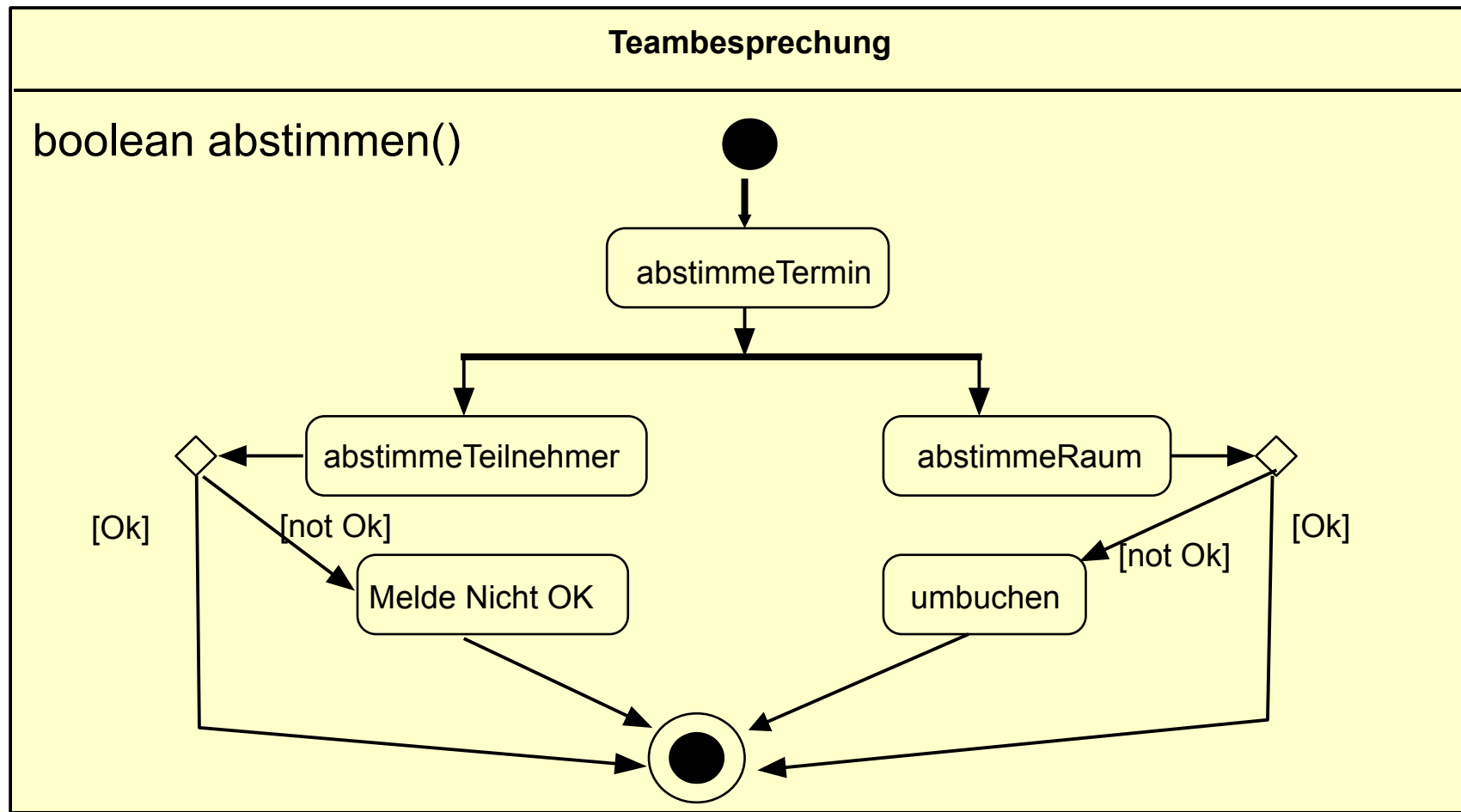
- ▶ Eine Signatur eines Objektes oder einer Methode muss **funktional verfeinert** werden
 - Das Verhalten (dynamische Semantik) muss spezifiziert werden (partiell oder vollständig)
 - Daher spricht man von *Verhaltensmodellierung* oder *dynamischer Modellierung*
 - und von *punktweiser Verfeinerung* einer Klassen- oder Methoden-Signatur
- ▶ Einfachste Form: Angabe von Aktivitäten, verknüpft mit Steuer- und Datenfluss
 - Geschachtelt in eine Oberaktivität



Aktivitätsdiagramm als Verhalten einer Methode (activity diagram)

8

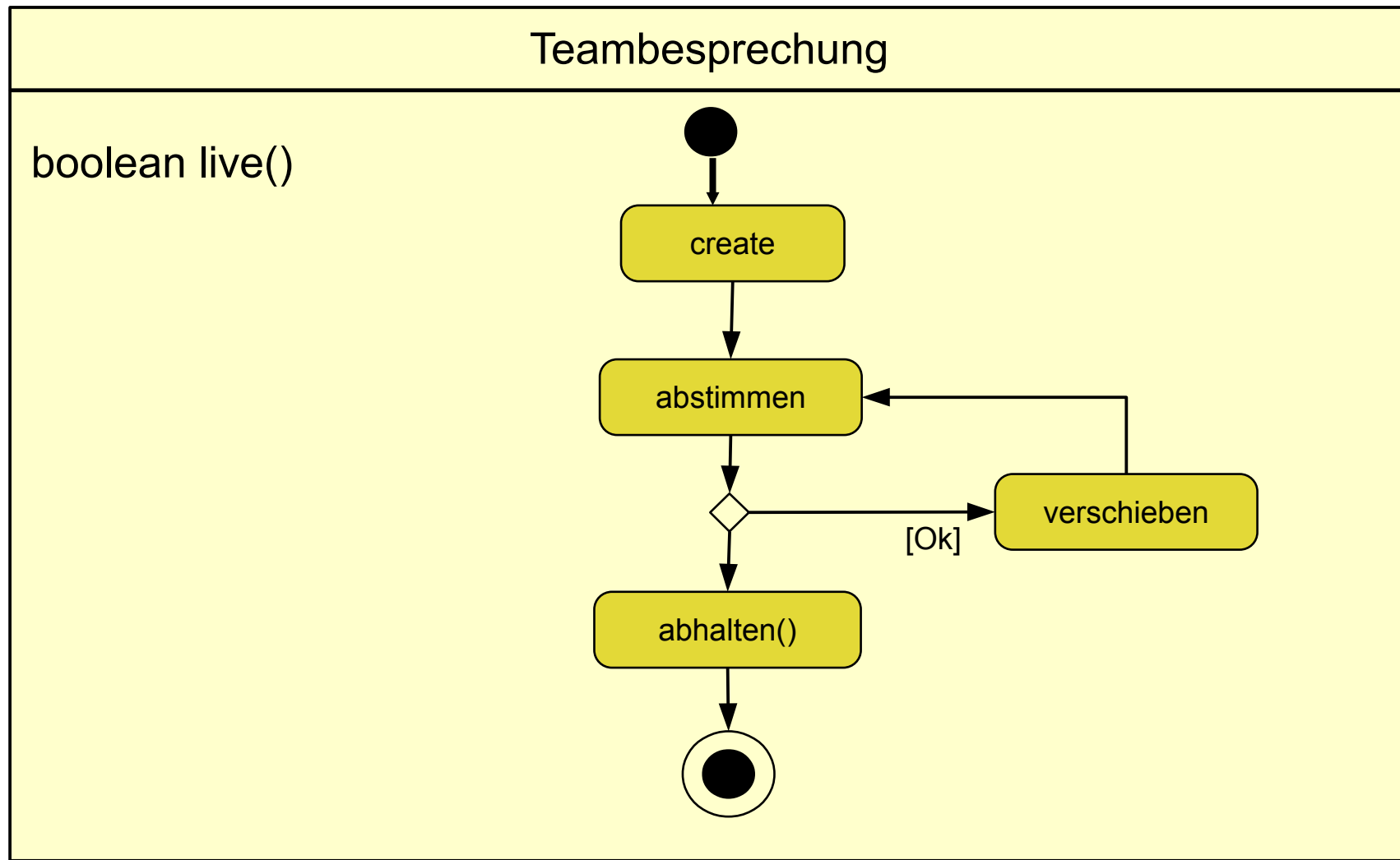
- ▶ Aktivitätsdiagramme können das Verhalten einer Methode beschreiben, dann werden sie in ein Abteil der Klasse notiert
- ▶ Aktivitäten, verbunden durch Datenfluß (Datenflußdiagramm, data-flow diagram)
 - Parallele Aktivitäten in parallelen Zweigen
 - Bedingungen (guards) bestimmen, ob über eine Kante Daten fließen (*bedingter* Datenfluß)



Aktivitätsdiagramm für Lebenszyklus eines Objekts

9

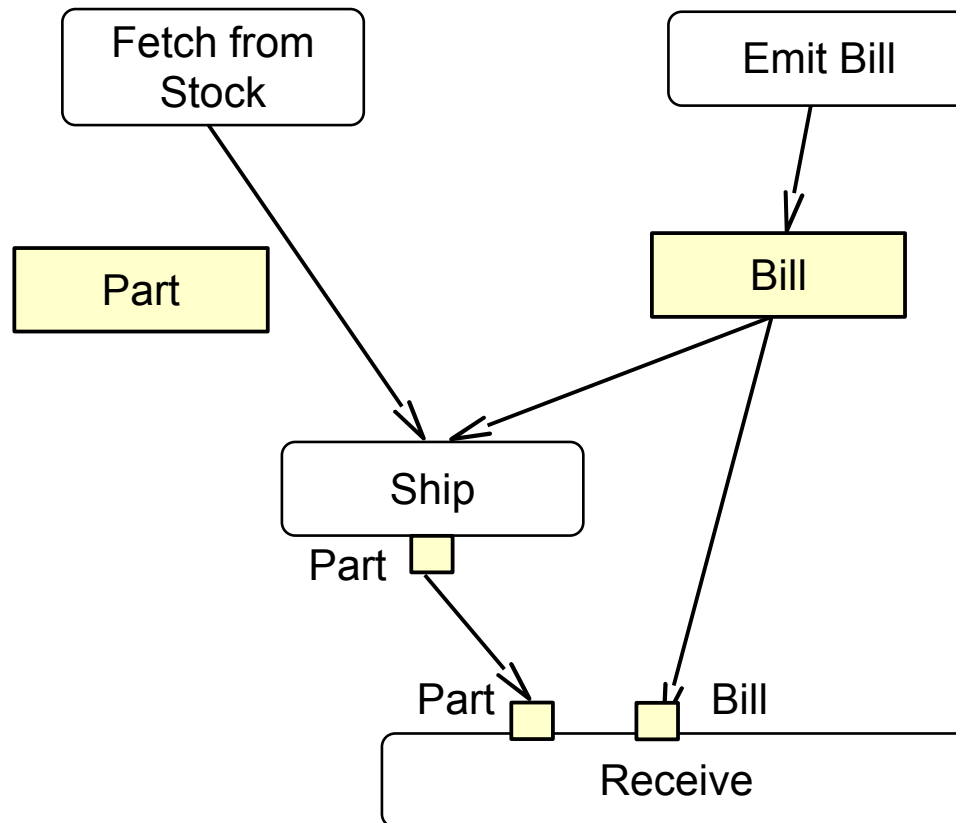
- ▶ Viele Objekte müssen in einer bestimmten Art und Weise aufgerufen werden, von ihrer Geburt bis zum Tod
- ▶ AD beschreiben den *Arbeitsfluss (Workflow)* der Methoden (hier: abstimmen() wird aufgerufen)



Verschiedene Notationen für Datenfluß

11

- ▶ Objekte, die zwischen Aktivitäten fließen, können verschieden notiert werden
- ▶ *Pins* sind benannte Parameter der Aktivitäten
 - Unterscheide von Ports von Komponenten!



33.2 UML-Zustandsdiagramme (Zustandsmaschinen, Statecharts)

12

Zustandsmaschinen gehören zu jUML, weil sie verlustfrei in Code überführt und zurücküberführt werden können (round-trip engineering)

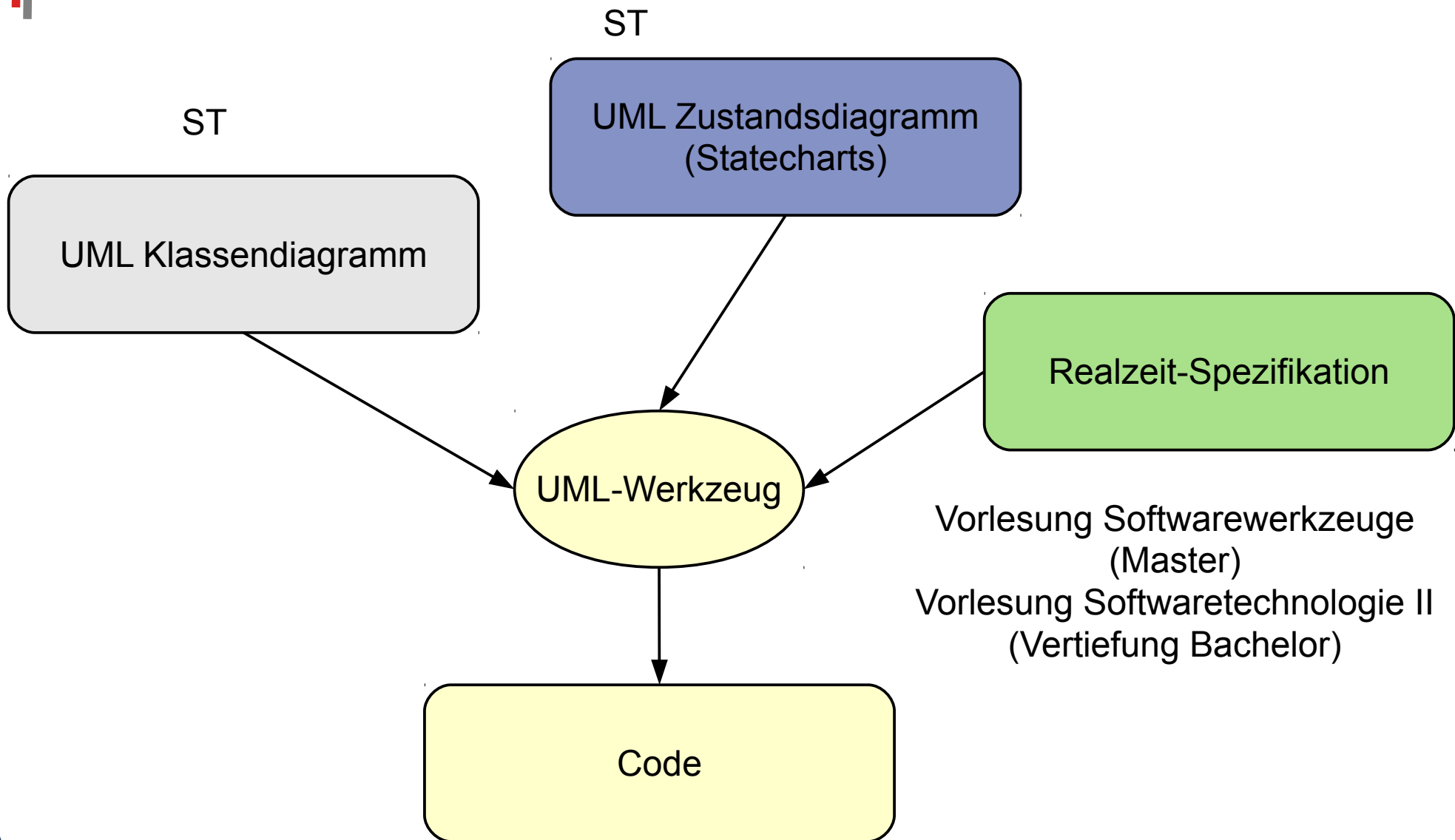
Zustandsbasierte dynamische Modellierung für sicherheitskritische Systeme

13

- ▶ Objekt-Verhalten und Szenarien können auch *zustandsbetont* analysiert werden
- ▶ Spezifikationen von Zustandsautomaten für Fly-by-wire und Drive-by-wire
- ▶ Methodik: Analyse und Entwurf mit UML-Statecharts

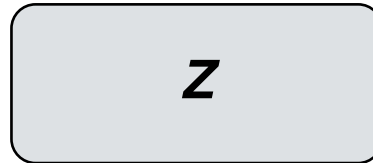


- Die Paderborner Railcabs arbeiten mit UML-Statecharts, die mit Realzeitattributen angereichert sind (real-time statecharts):



- ▶ **Definition:** Ein *Zustand* ist eine Eigenschaft eines Objektes oder Systems, die über einen begrenzten Zeitraum besteht.

- ▶ **Notation:**



- ▶ Ein (technisches) **System** besteht aus einem (komplexen) Objekt oder einem Netz von Objekten, das Aufgaben ausführt
 - eines komplexes Softwaresystem
 - ein Arbeitsprozess
 - ein Produkt eines Arbeitsprozesses
 - ein einzelnes Objekt
- Vorsicht! Zustände werden wie Aktivitäten mit ovalen Boxen notiert, gehören aber zu einem Zustandsdiagramm!

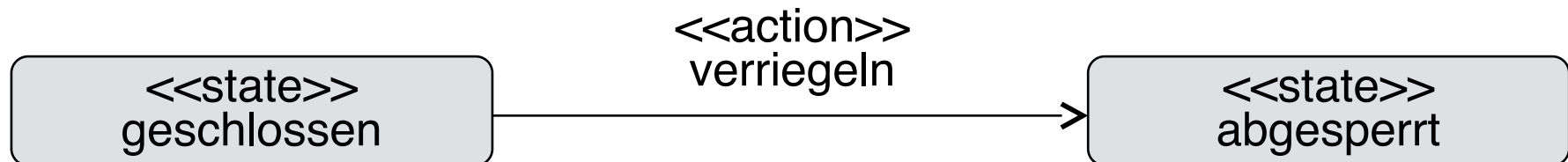
Endliche Automaten 1 (Akzeptoren)

16

- ▶ Theoretische Informatik, Automatentheorie:

Ein **endlicher Zustandsautomat (Akzeptor)** über einem Eingabealphabet A ist ein Tupel, bestehend aus:

- einer Menge S von Zuständen
- einer (partiellen) Übergangsfunktion $\text{trans} : S \times A \rightarrow S$
- einem Startzustand $s_0 \in S$
- einer Menge von Endzuständen $S_f \subseteq S$



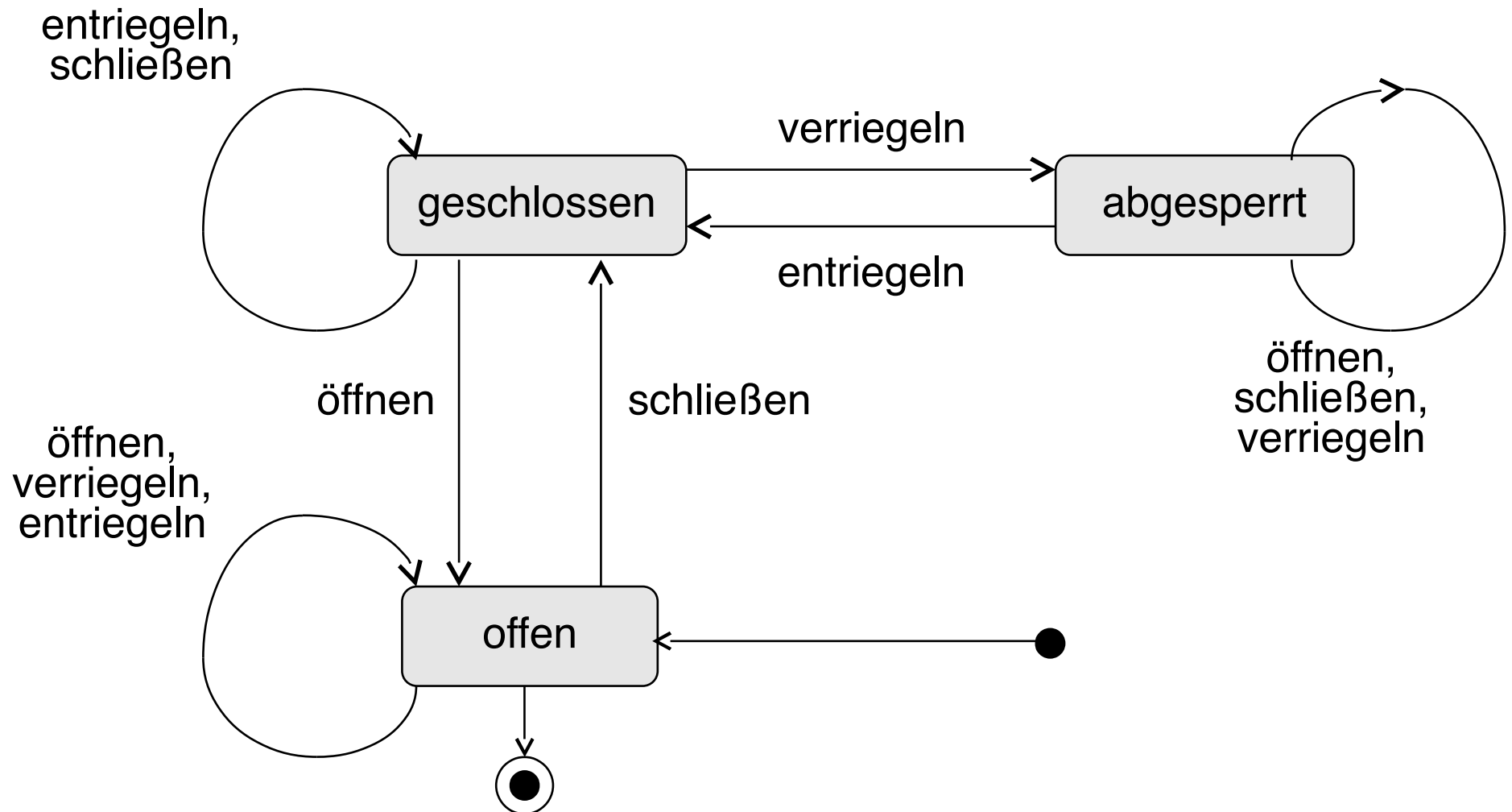
$\text{trans}(\text{geschlossen}, \text{verriegeln}) = \text{abgesperrt}$

Achtung! Notation von Zuständen ähnlich zur Notation von Aktivitäten!

Beispiel: Zustandsmodell einer Tür

17

- ▶ Der Tür-Akzeptor stellt einen Prüfer für mögliche Aktionsfolgen für eine Tür dar
- ▶ In UML heisst der Akzeptor **Protokoll(zustands)maschine**, denn er akzeptiert eine Folge von Ereignissen (Protokoll)



Zustandstabellen von Protokollmaschinen

18

Tür Ausgangs-/Endzustand	geschlossen	offen	abgesperrt
geschlossen	entriegeln, schließen	öffnen	verriegeln
offen	schließen	öffnen, verriegeln, entriegeln	-
abgesperrt	entriegeln	-	öffnen, schließen, verriegeln

- ▶ Tabellen bilden eine alternative Notation

Endliche Automaten mit Aktionen (Transduktoren)

19

- ▶ Ein **endlicher Zustandsübersetzer (Transduktor, endlicher Übersetzer)** übersetzt eine Folge von Zeichen aus einem Eingabealphabet A in eine Folge von Zeichen aus **einem Ausgabealphabet B** . Er ist ein Tupel, bestehend aus:
 - einer Menge S von Zuständen
 - einer (partiellen) Übergangsfunktion $\text{trans} : S \times A \rightarrow S$
 - einem Startzustand $s_0 \in S$
 - einer Menge von Endzuständen $S_f \subseteq S$

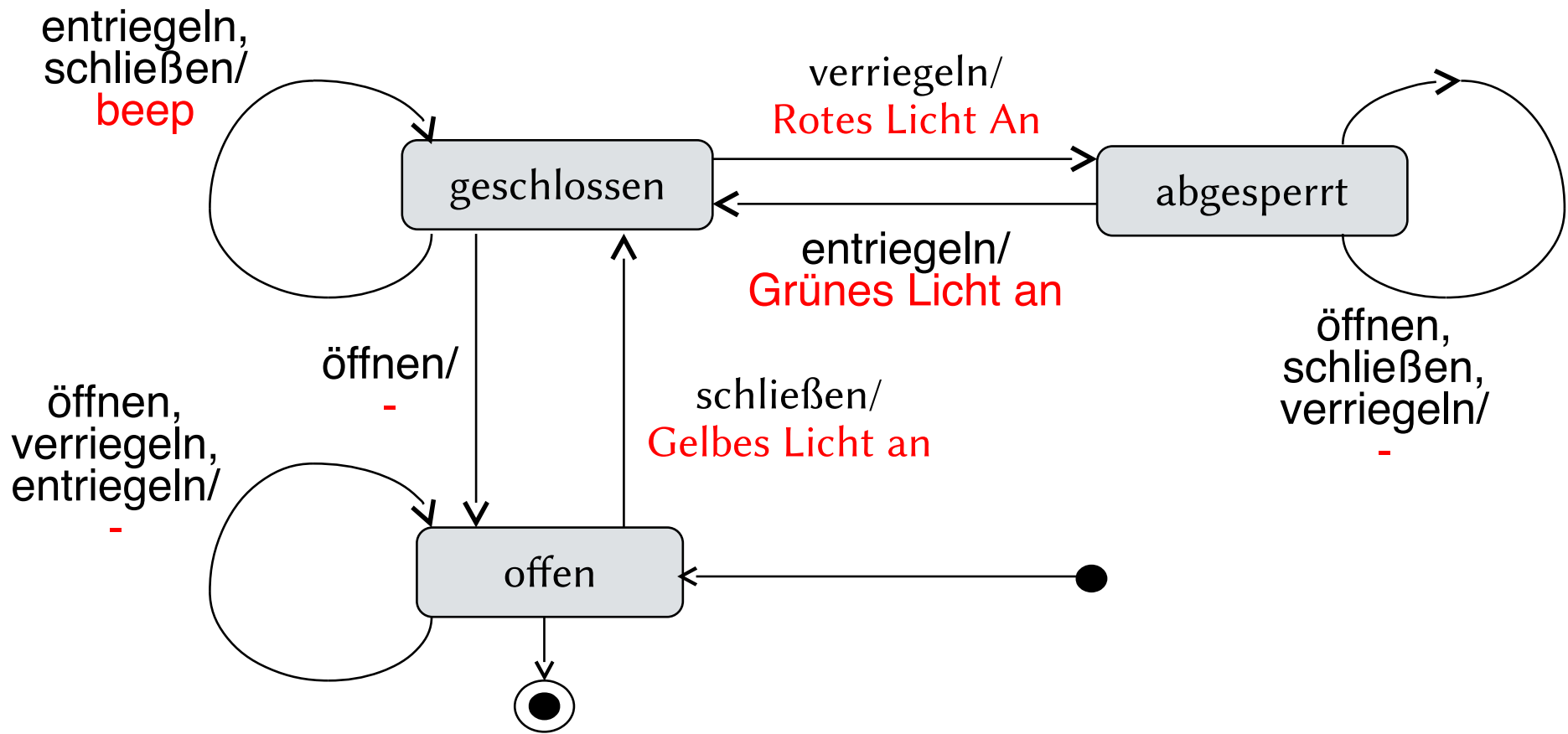


$\text{trans}(\text{geschlossen}, \text{verriegeln}) = (\text{abgesperrt}) / \text{rotes Licht einschalten}$

Beispiel: Zustandsmodell einer Tür

20

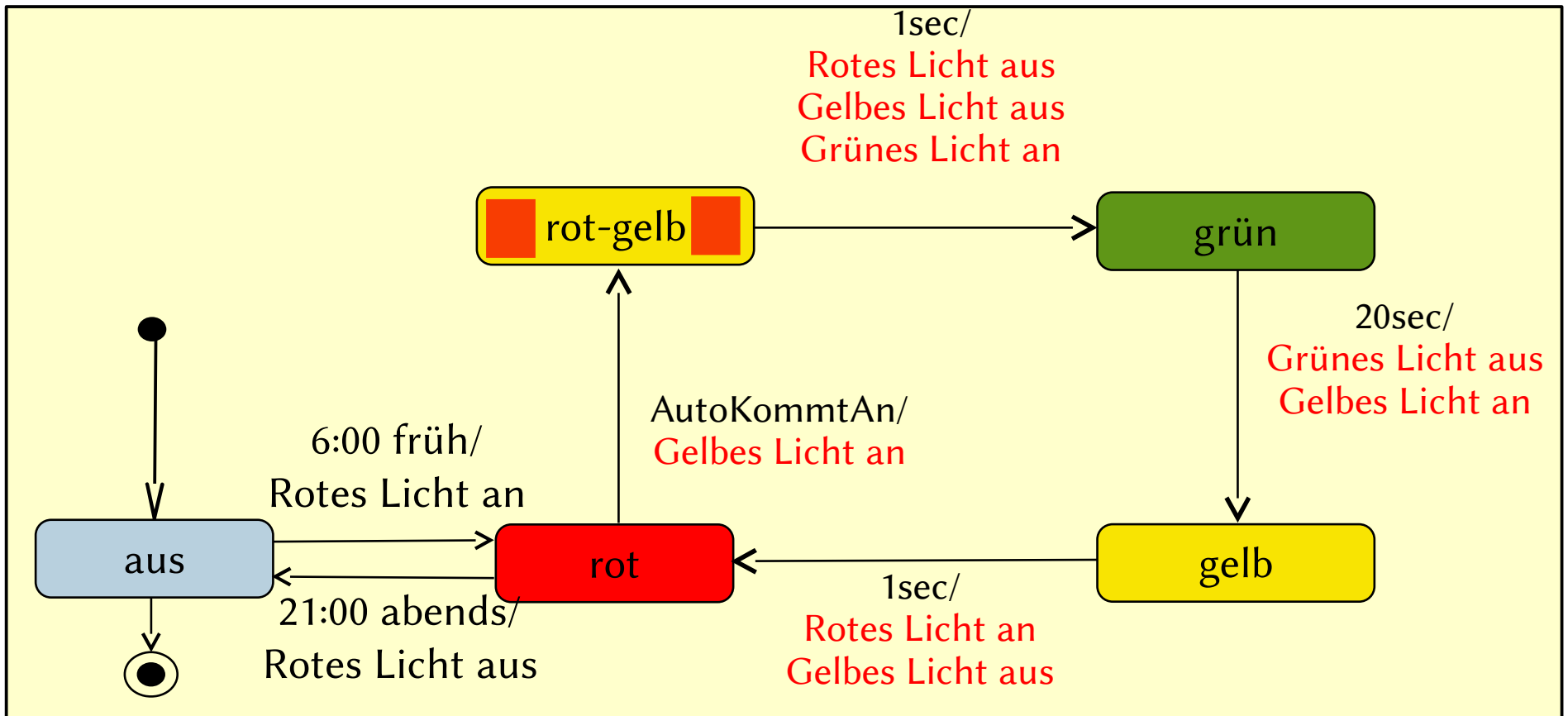
- ▶ Der Tür-Transduktor stellt zusätzlich zum Prüfer einen Steuerer (controller) für eine Tür-Zustandsmeldeampel dar
 - aus ihm kann ein Steuerungsalgorithmus für die Türampel abgeleitet werden
- ▶ In UML: **Zustandsmaschine (Verhaltensmaschine)**



Beispiel: Zustandsmodell einer bedarfsgesteuerten Ampel

21

- ▶ Welches Ereignis löst den Ampelzyklus aus?
- ▶ Welches Eingabealphabet hat der Transduktor (Ereignisse)?
- ▶ Welches Ausgabealphabet?
- ▶ Welche Sprachen (Mengen von Folgen von Ereignissen) übersetzt der Transduktor in einander?



Semantik eines Zustandsmodells

22

- ▶ Ein Zustandsmodell ist endlich, definiert aber einen unendlichen Zustandsraum
 - seine Semantik ist eine unendliche Menge von Pfaden über Zuständen und Aktionen (Aktionsfolgen):
 - Rollt man die Pfade des Zustandsmodells ab, entsteht ein unendlich tiefer Baum
- ▶ Die Semantik eines Zustandsmodells ist definiert als **Menge von Sequenzen (Aktionsfolgen)**:
 - in der Theoretischen Informatik:
 - Menge von "akzeptierten Wörtern" (Sprache über Grundalphabet von Ereignissen)
 - in der Softwaretechnik wird das interpretiert als:
 - Menge von zulässigen *Ereignisfolgen* (*Ereignissprache*)
 - Menge von zulässigen *Aufruffolgen* oder *Aktionen* (*Aufrufsprache*)
 - Menge von zulässigen *Pfaden in einem Graphen* (*Pfadsprache*)
- ▶ Wichtige Verallgemeinerung: "Automaten mit Ausgabe"
 - *Transduktor* (*Mealy-Automat*): Ausgabe bei Übergang
 - Softwaretechnik: *Aktion* bei Übergang
 - *Akzeptor* (*Moore-Automat*): Ausgabe bei Erreichen eines Zustands

Übung

23

- ▶ Schreiben Sie 4 zulässige Schaltfolgen der bedarfsgesteuerten Ampel aus
- ▶ Was ähnelt sich?

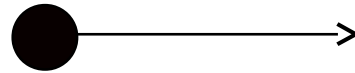
- ▶ Übung 2: Rollen Sie den Ampelautomaten ab in einen Zustandsübergangsbaum. Wann wiederholen sich die Ereignisfolgen?

Start- und Endzustand (wie bei AD)

24

- ▶ Jedes Zustandsdiagramm sollte einen eindeutigen Startzustand haben. Der Startzustand ist ein "Pseudo-Zustand".

- ▶ **Notation:**



- ▶ Ein Zustandsdiagramm kann einen oder mehrere Endzustände haben.
- ▶ **Notation:** ("*bull's eye*")



Bedingte Zustandsübergänge in Protokollmaschinen

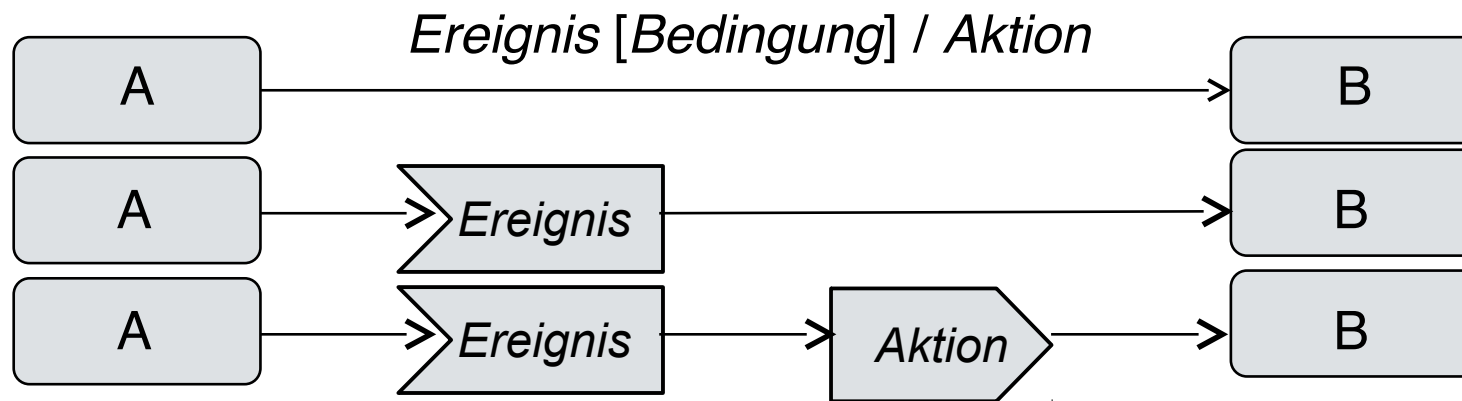
25



- ▶ **Definition** Eine **Bedingung** (*guard*) ist eine Boolesche Bedingung, die zusätzlich bei Auftreten des Ereignisses erfüllt sein muß, damit der beschriebene Übergang eintritt.
- ▶ **Notation:** Eine Bedingung kann folgende Informationen verwenden:
 - Parameterwerte des Ereignisses
 - Attributwerte und Assoziationsinstanzen (Links) der Objekte
 - ggf. Navigation über Links zu anderen Objekten
- ▶ **Beispiel:**



Aktionen bei Zustandsübergängen in Verhaltensmaschinen

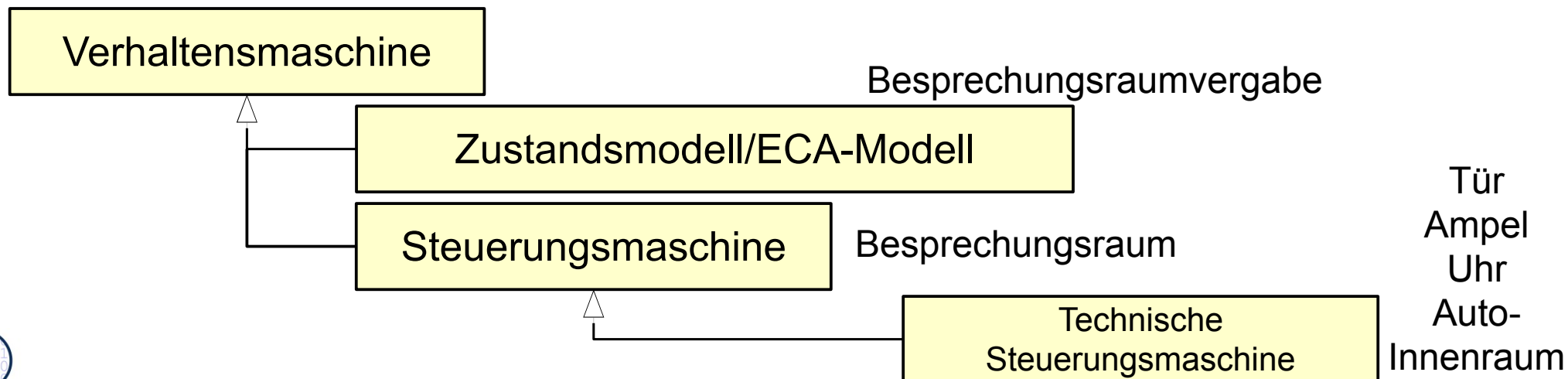


- ▶ **Definition** Eine *Aktion* ist die Beschreibung einer ausführbaren Anweisung. Dauer der Ausführung vernachlässigbar. Nicht unterbrechbar. Eine Aktion kann auch eine Folge von Einzelaktionen sein.
- ▶ In UML heissen Zustandsübergänge mit Aktionen **volle Zustandsübergänge**
- ▶ Typische Arten von Aktionen:
 - Lokale Änderung eines Attributwerts
 - Versenden einer Nachricht an ein anderes Objekt (bzw. eine Klasse)
 - Erzeugen oder Löschen eines Objekts
 - Rückgabe eines Ergebniswertes für eine früher empfangene Nachricht

Spezielle Verhaltensmaschinen (Transduktoren):

28

- ▶ Ein **Zustandsmodell (Ereignis/Bedingungs/Aktionsmodell, event/condition/action model, ECA model)** ist eine Verhaltensmaschine, die keinem Objekt (keiner Klasse) zugeordnet ist
- ▶ Eine **Steuerungsmaschine** ist eine spezielle Verhaltensmaschine, die einem Objekt zugeordnet ist und das Verhalten eines Objekts beschreibt
 - Sie beschreibt dann einen vollständigen *Objektlebenszyklus* (white-box object life cycle)
- ▶ Eine **technische Steuerungsmaschine** beschreibt das Verhalten eines technischen Gerätes
 - Aus Steuerungsmaschinen kann die Implementierung der Steuerungssoftware des Objekts bzw. des Geräts abgeleitet werden (wichtig für eingebettete Systeme)



33.3 Unterschied von Verhaltens-, Steuer und Protokollmaschinen

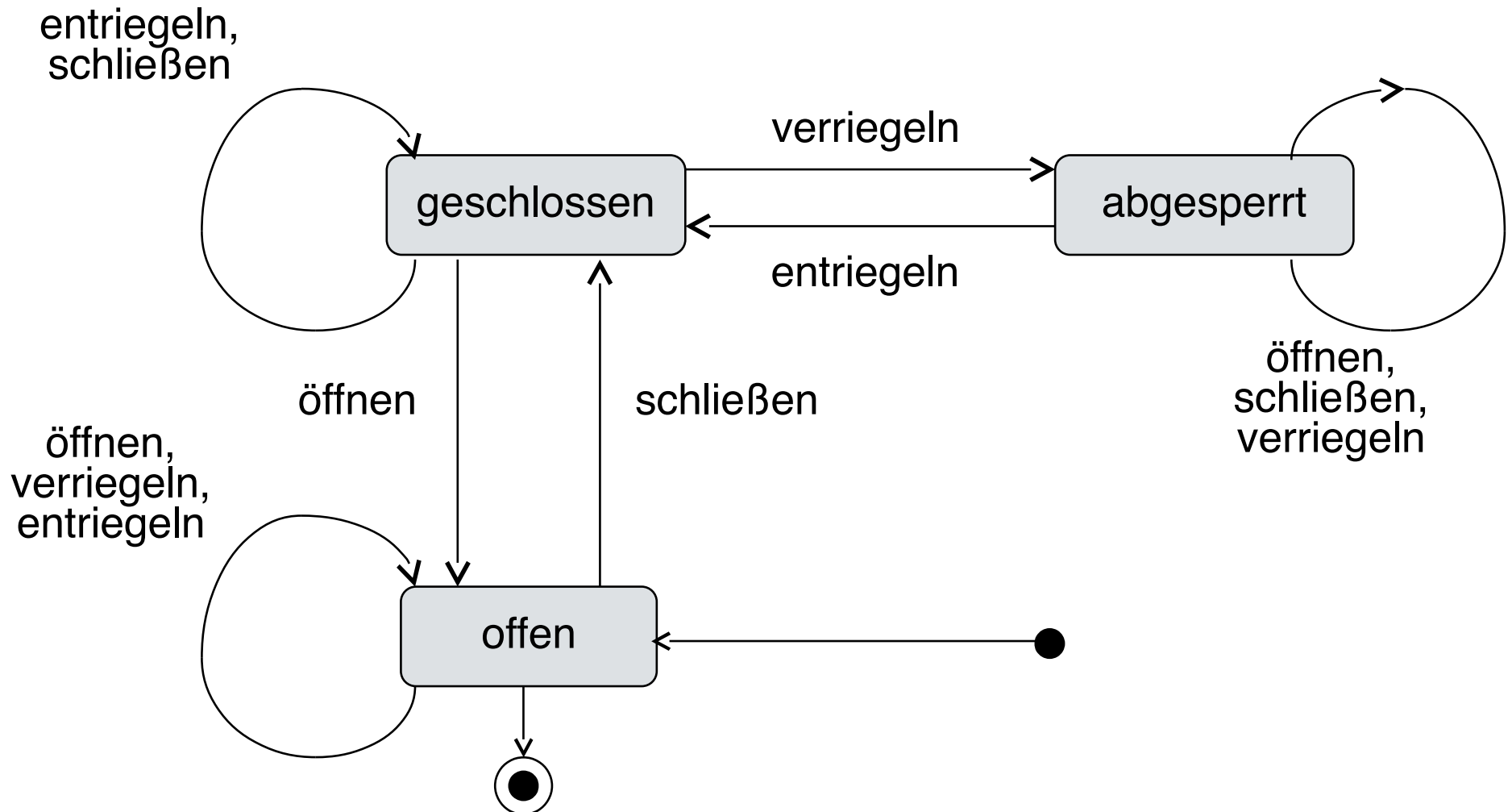
29



Beispiel: Protokollmaschine kontrolliert die Benutzung einer Tür

30

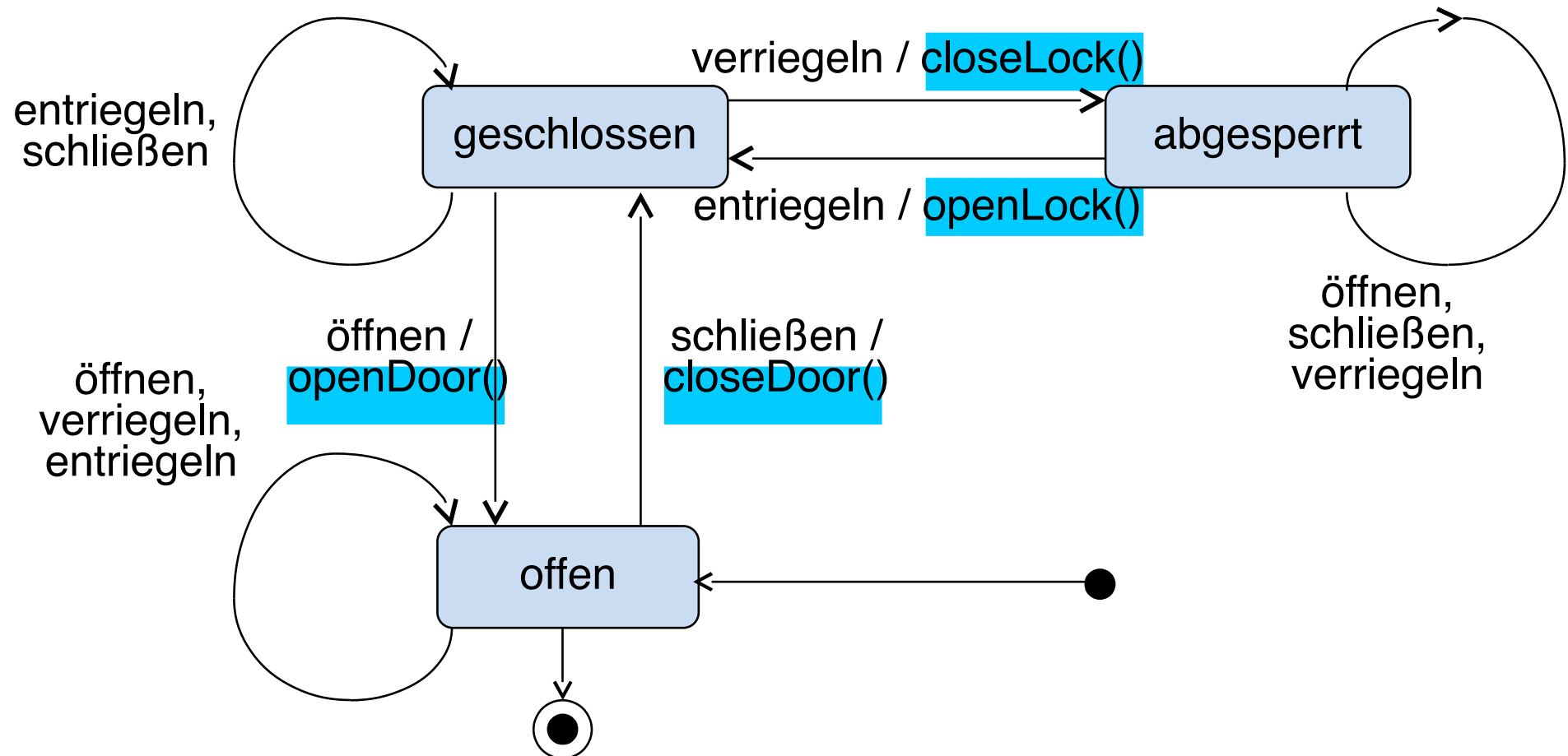
- ▶ Eine Protokollmaschine *kontrolliert*, ob ein Benutzer eine Zustandsmaschine richtig bedient,
 - d.h. ob die Benutzungsreihenfolge einer Zustandsmaschine folgt (akzeptierend, beobachtend, prüfend).



Beispiel: Steuerungsmaschine für eine Tür einer Behindertentoilette

31

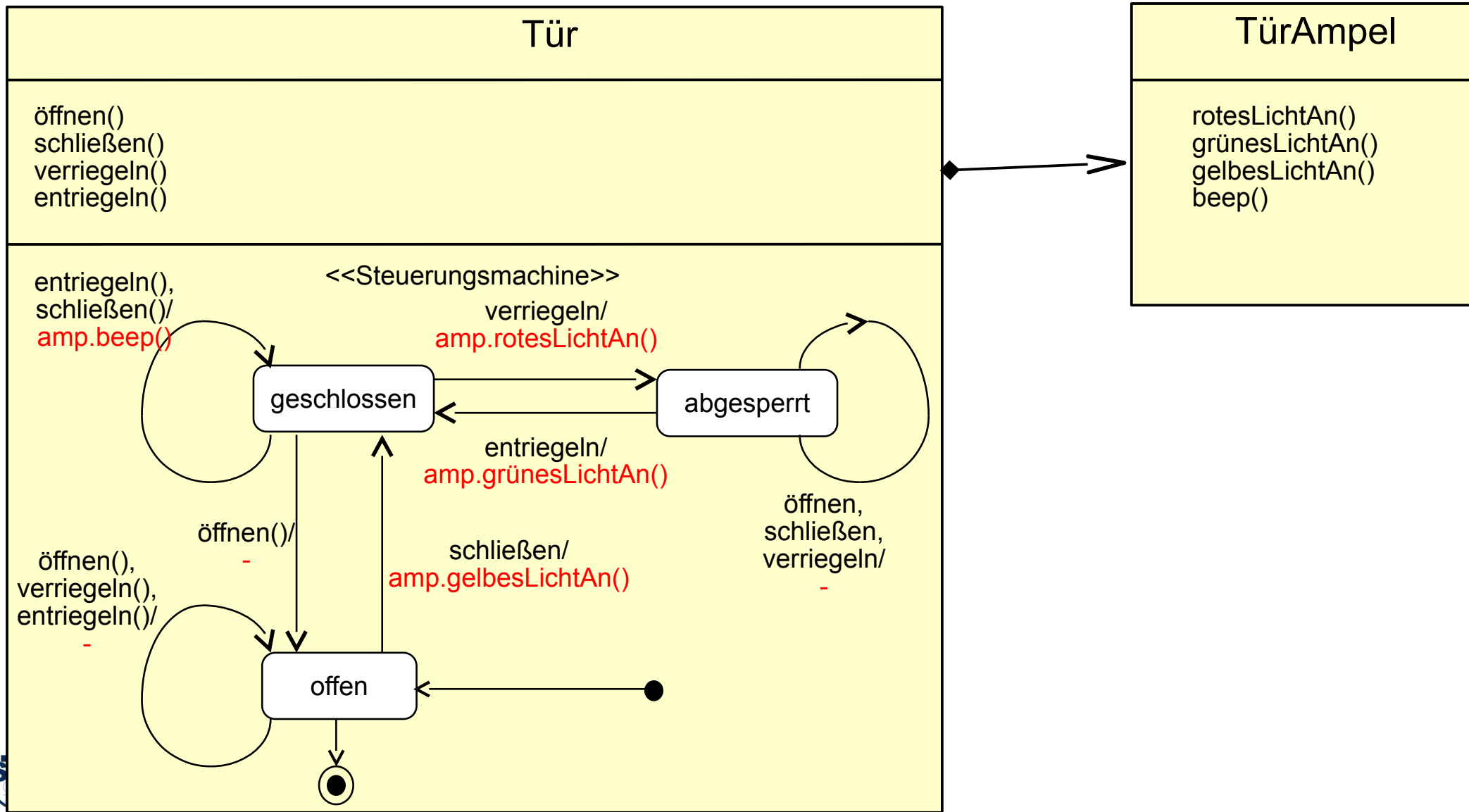
- ▶ Eine Steuerungsmaschine steuert zusätzlich weitere Klassen an
- ▶ Hier: die Türsteuerung empfängt die Signale des Türbenutzers und steuert Servo-Motoren an
 - Achtung: das ist bereits die zweite Steuerungsmaschine zur Protokollmaschine des Türprotokoll-Prüfers!



Objektlebenszyklus von innen und aussen

32

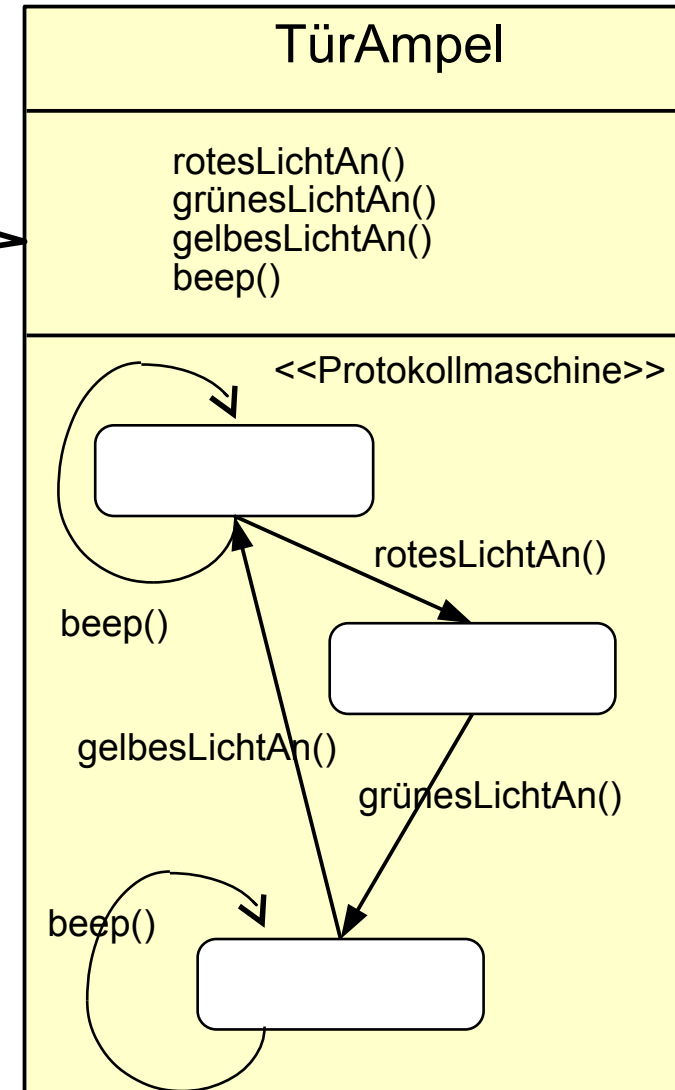
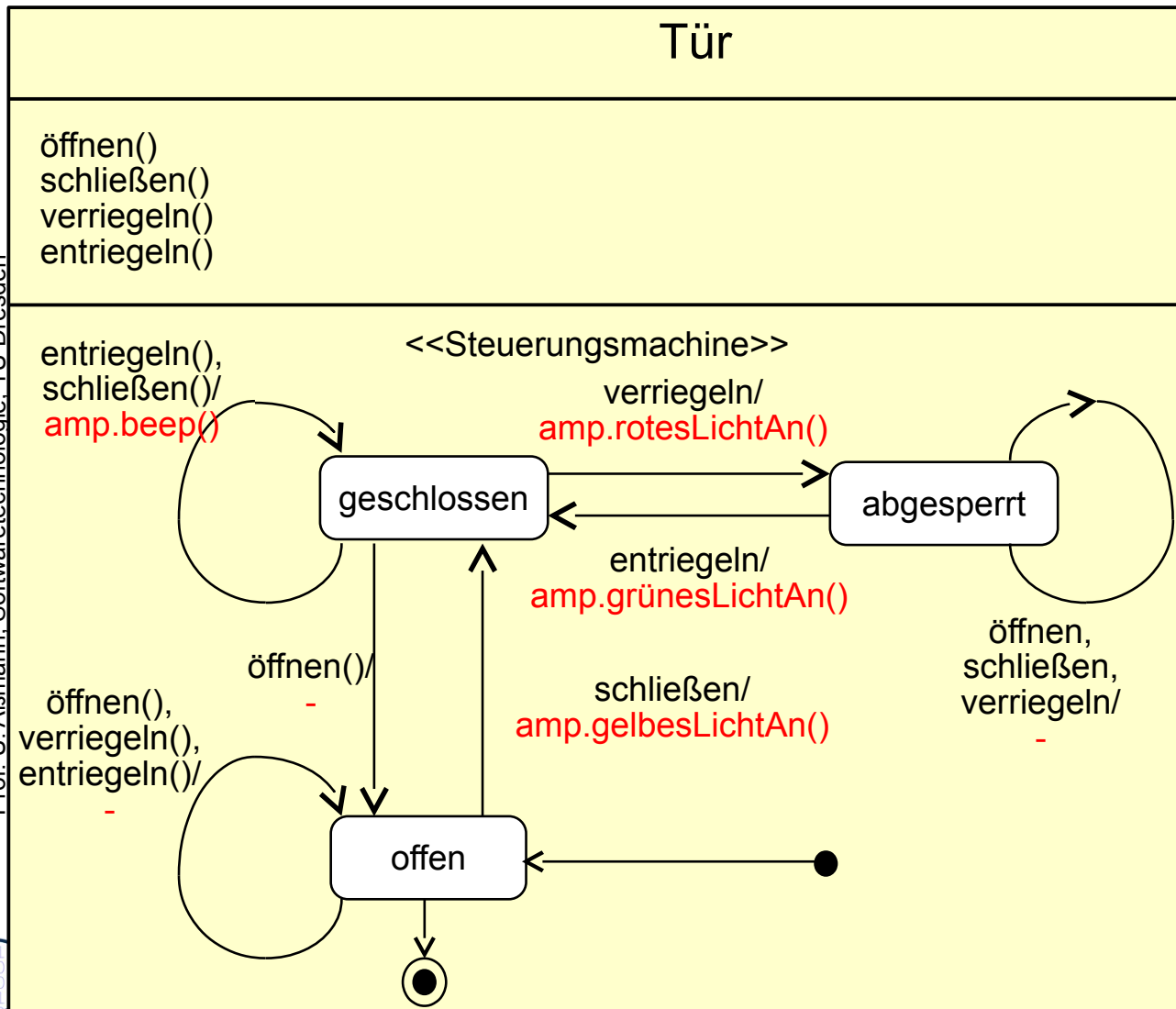
- ▶ Eine Steuerungsmaschine kann im Compartment einer Klasse erscheinen; sie beschreibt einen *whitebox*-Objektlebenszyklus



Objektlebenszyklus von nur von aussen

33

- ▶ Auch eine Protokollmaschine kann im Compartment einer Klasse erscheinen. Dann beschreibt sie einen *blackbox*-Objektlebenszyklus, d.h. die beobachtbare Sicht von aussen, das *Protokoll* der Schnittstelle oder Klasse



▶ Verhaltens- (Steuerungs-)maschinen

- steuern
- müssen das Wissen über das gesteuerte System *vollständig* repräsentieren, ansonsten gerät das System ausser Kontrolle
- geben mit ihren Aktionen eine Implementierung der Steuerungssoftware des technischen Systems an
- können verschiedene Dinge steuern:
 - sich selbst (reine Steuerungsmaschine)
 - andere Klassen
 - ein Subsystem von Klassen

▶ Protokollmaschinen

- kontrollieren, prüfen
- können ein *partielles* Wissen über das geprüfte System kontrollieren (der Rest des Verhaltens wird *nicht* abgeprüft)
- Beschreiben eine Sicht von aussen auf das System
- Beschreiben das Aufruf- oder Ereignisprotokoll des Systems

33.4 Implementierung von Steuerungsmaschinen

35



Implementierung von Steuerungsmaschinen mit Implementierungsmuster *IntegerState*

36

- ▶ Entwurfsmuster *IntegerState*
 - Zustand wird als Integer-Variable repräsentiert, Bereich [1..n]
 - Alle Ereignisse werden zu “Reaktions”-Methoden, die von aussen aufgerufen werden
 - Externe Ereignisse werden mit “Reaktions-Methoden” modelliert
 - Interne Ereignisse werden den Implementierungen der Methoden zugeordnet
- ▶ Reaktionsmethoden schalten den Zustand fort, indem sie Fallanalyse betreiben
 - In jeder Methode wird eine Fallunterscheidung über den Zustand durchgeführt
 - Jeder Fall beschreibt also ein Paar (Ereignis, Zustand)
 - Der Rumpf des Falles beschreibt
 - den Zustandsübergang (Wechsel des Zustands)
 - die auszulösende Aktion

```
class Tuer {
```

Modifier "final" bei Attributen: unveränderlich

```
    // Konstante zur Zustandskodierung
    private static final int Z_offen = 0;
    private static final int Z_geschlossen = 1;
    private static final int Z_abgesperrt = 2;

    // Zustandsvariable
    private int zustand = Z_offen;

    // Reaktionsmethode oeffnen (Reaktion auf Ereignis)
    public void oeffnen() {
        // Fallanalyse über Zustand
        switch (zustand) {
            case Z_offen:
                break;
            case Z_geschlossen:
                zustand = Z_offen;
                System.out.println("Klack");
                break;
            case Z_abgesperrt:
                break;
        }
    }
}
```

IntegerState

Beispiel: Code zur Steuerung einer Tür (2)

```
public void schliessen() {
    // Fallanalyse
    switch (zustand) {
        case Z_offen:
            zustand = Z_geschlossen;
            System.out.println("Klick");
            break;
        case Z_geschlossen:
            break;
        case Z_abgesperrt:
            break;
    }
}

public void verriegeln() {
    switch (zustand) {
        case Z_offen:
            break;
        case Z_geschlossen:
            zustand = Z_abgesperrt;
            System.out.println("Knirsch");
            break;
        case Z_abgesperrt:
            break;
    }
}
```

38

IntegerState

Beispiel: Code zur Steuerung einer Tür (3)

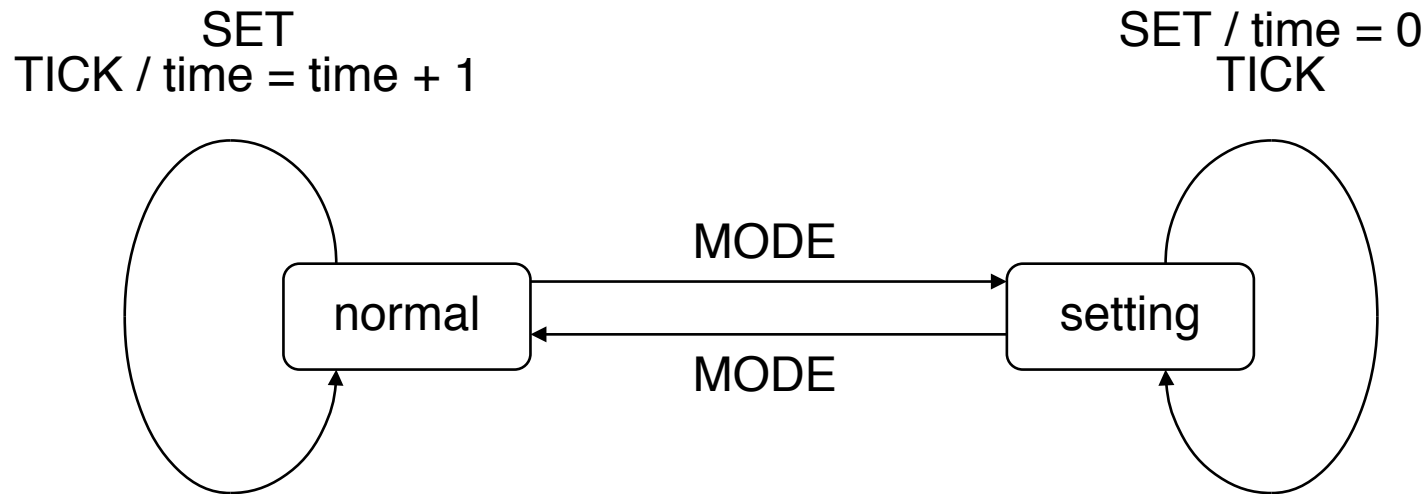
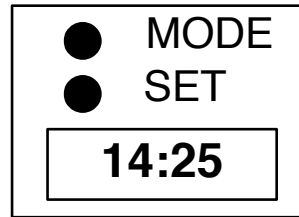
```
public void entriegeln() {
    switch (zustand) {
        case Z_offen:
            break;
        case Z_geschlossen:
            break;
        case Z_abgesperrt:
            zustand = Z_geschlossen;
            System.out.println("Knirsch");
            break;
    }
}
```

```
// Client-Klasse
class TuerBediener {
    public static void main(String[] args) {
        Tuer t1 = new Tuer();
        t1.oeffnen();
        t1.schliessen();
        t1.verriegeln();
        t1.entriegeln();
        t1.oeffnen();
        t1.schliessen();
    }
}
```

Aufgabe: Steuerungsmaschine realisieren

40

- ▶ Beispiel: Betriebsmodi einer Taschenuhr (stark vereinfacht)



Implementierung mit IntegerState

41

```
class Clock {  
  
    private int time = 0;  
    private static final int NORMAL = 0;  
    private static final int SETTING = 1;  
  
    private int state = NORMAL;  
  
    public void set () {  
        switch (mode) {  
            case NORMAL: {  
                time = time+1;  
                break;  
            };  
            case SETTING: {  
                time = 0;  
                setChanged();  
                break;  
            };  
        };  
    }  
  
    ...// analog tick(), mode()  
}
```

Übung

42

- ▶ Implementieren Sie den Ampelautomaten mit dem Implementierungsmuster IntegerState.
- ▶ Wieviele Reaktionsmethoden brauchen Sie?



33.5 Kooperierende Zustandsmaschinen

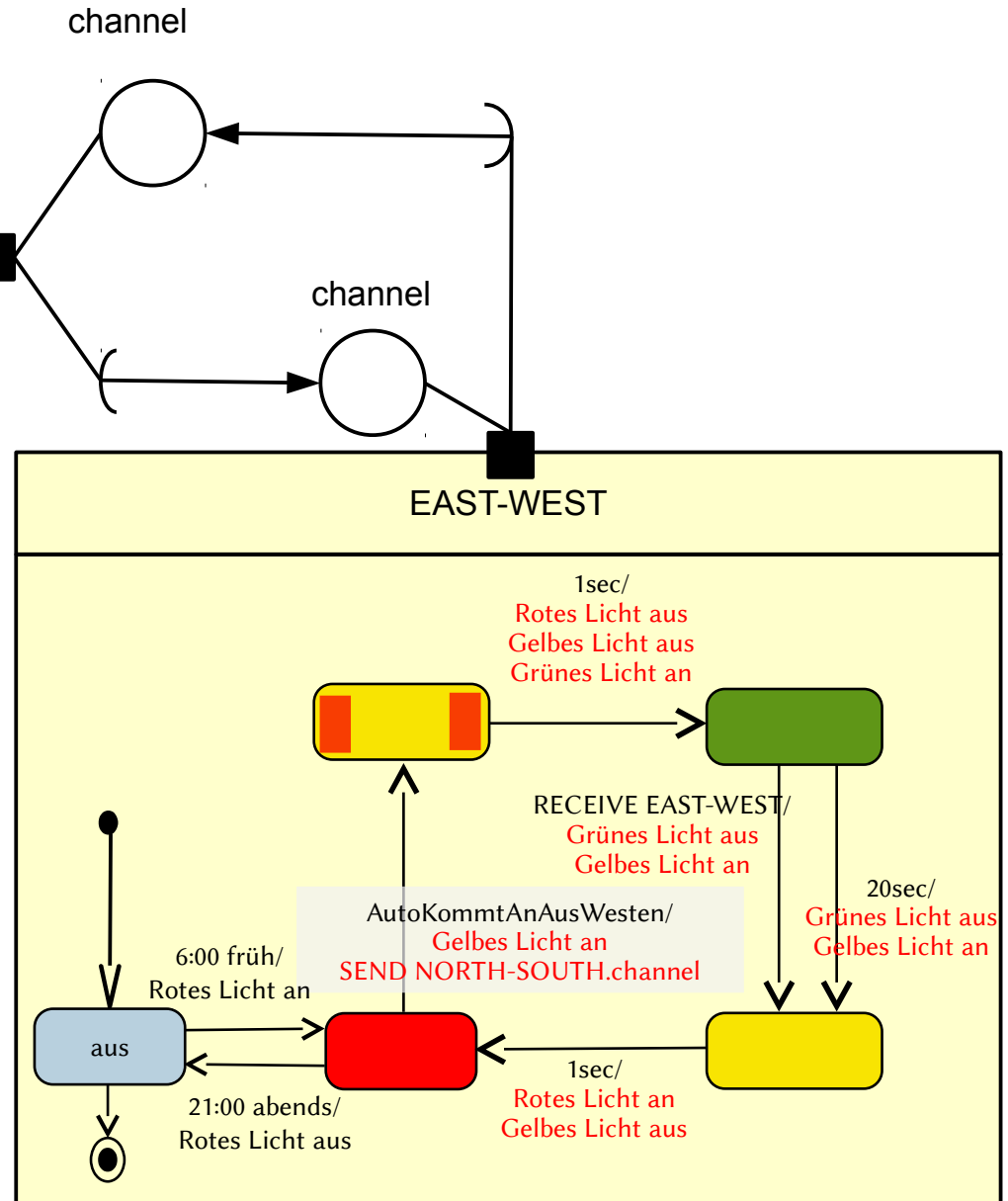
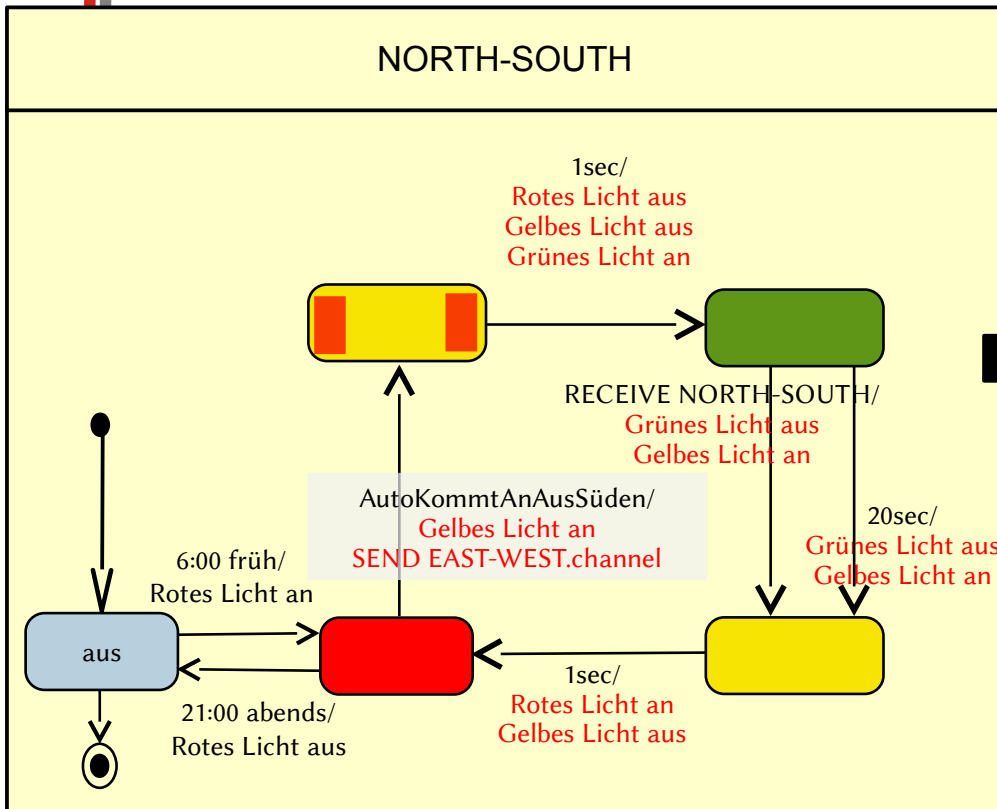
43

Kooperierende Zustandsmaschinen

44

- ▶ Eine besonders einfache Art von Objektnetz bilden solche, in denen alle Objekte Zustandsmaschinen bilden, die kooperieren
 - sich Nachrichten senden (Ereignisse)
 - auf Ereignisse in Nachbarobjekten mit eigenen Reaktionsmethoden reagieren

Bsp.: Kopplung zweier Ampeln an einer Kreuzung durch Ereignis-Kanäle

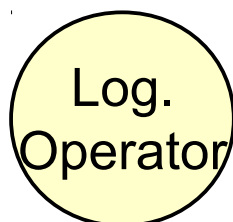
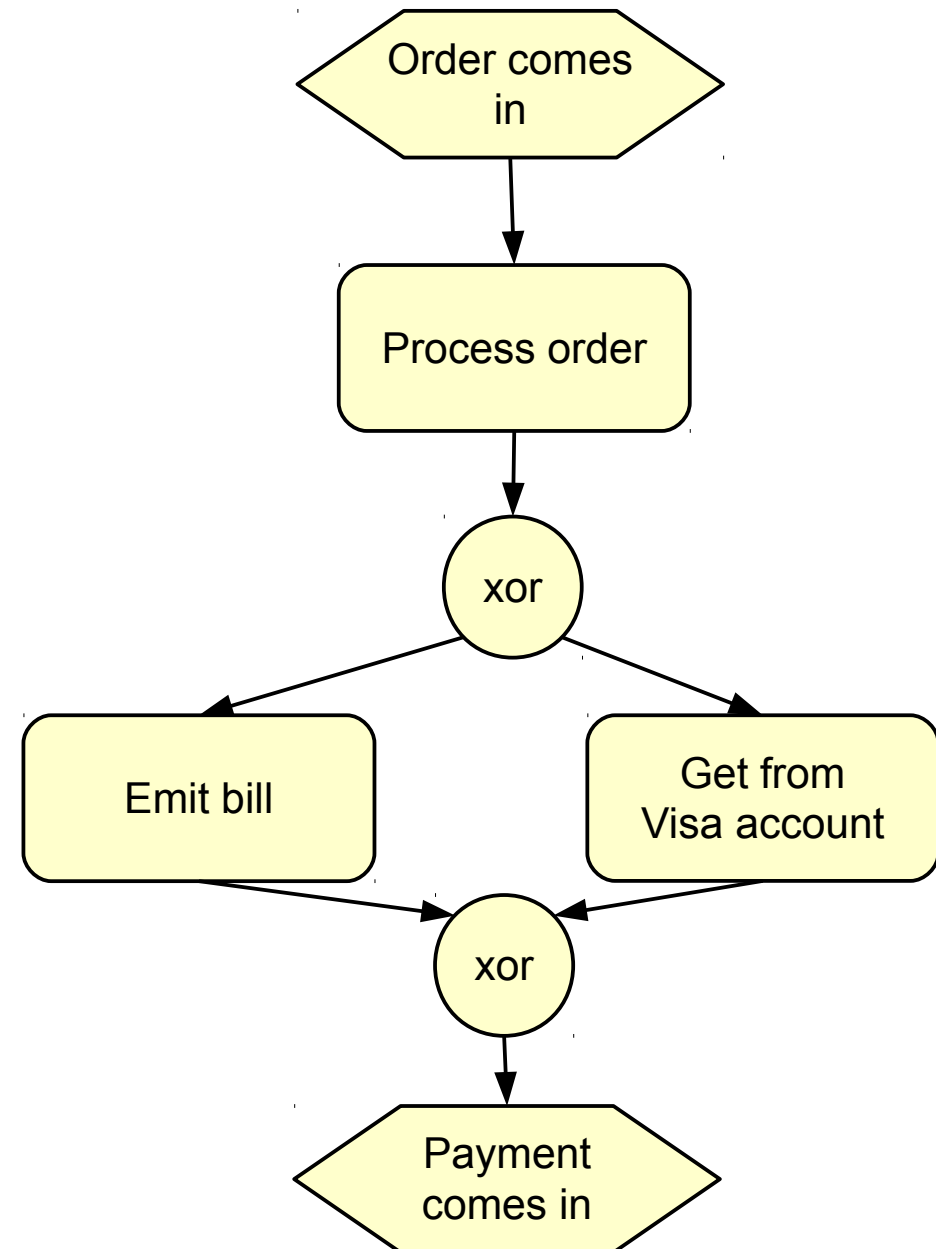


- ▶ Many slides courtesy to © Prof. Dr. Heinrich Hussmann, 2003. Used by permission.
- ▶ Typische Steuerungsmaschinen
 - Stellverhalten von Uhren
 - Autotüren und -heckklappen
 - Geldautomaten
 - Bahnkarten-Verkaufsautomat
 - Fahrstühle [Jazayeri]
- ▶ Typische Zustandsmaschinen für Abläufe:
 - Hausbau
 - Projekte
 - Immatrikulation eines Studenten

33.A.1 Andere Notationen für Aktivitätsdiagramme

49

- ▶ Ähnlich, wird in der industriellen Praxis oft benutzt: Ereignisgesteuerte Prozessketten (EPK), Sprache des ARIS-Toolkits für Prozessmodellierung von SAP-Systemen





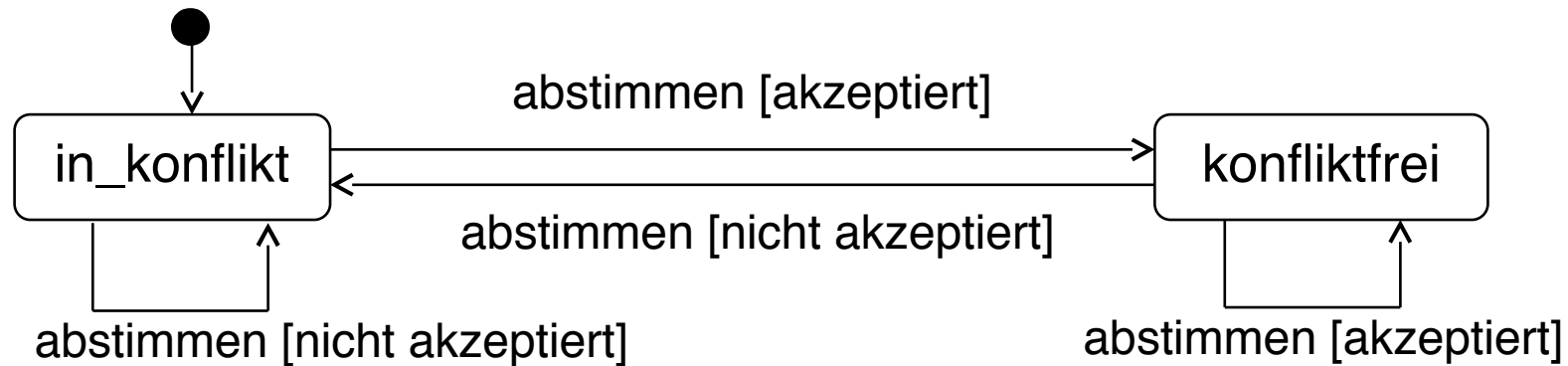
33.A.2 Implementierung von Protokollmaschinen

50

Beispiel: Protokollmaschine

51

- ▶ Folgende Protokollmaschine definiert die zulässigen Aufrufreihenfolgen der Klasse `Terminverschiebung`:



("akzeptiert" = von allen Teilnehmern akzeptiert)

- Begriff "Protokoll":
 - Kommunikationstechnologie
 - Regelwerk für Nachrichtenaustausch
- Protokollmaschinen in der Softwarespezifikation:
 - **zusätzliche** abstrakte Sicht auf komplexen Code (partielles Wissen)
 - Vertragsprüfer zur Einhaltung von Aufrufreihenfolgen

```
public Teambesprechung
    (String titel, Hour beginn, int dauer,
     Teammitglied[] teilnehmer) {
    int zustand = Z_nicht_abgestimmt;
    super(titel, beginn, dauer);
    this.teilnahme = teilnehmer;
    if (! abstimmen(beginn, dauer)) {
        System.out.println("Termin bitte verschieben!");
        zustand = Z_in_konflikt;
    }
    else {
        for (int i=0; i<teilnahme.length; i++)
            teilnahme[i].teilnahmeSetzen(this);
        zustand = Z_konfliktfrei;
    }
}
```

Explizites
Zustandsattribut

- Analog zu IntegerState, aber keine Aktionen
- Ablauflogik kann den Zustandswert benutzen (muß aber nicht!)

Implementierungsmuster Protokollmaschine

Implicit Tracing State

53

- ▶ Information über Zustand jederzeit berechenbar - hier aus den Werten der Assoziationen und den Datumsangaben
- ▶ Zustandsinformation gibt zusätzliches Modell, nicht direkt im Code wiederzufinden

```
public Teambesprechung
    (String titel, Hour beginn, int dauer,
     Teammitglied[] teilnehmer) {
    super(titel, beginn, dauer);
    this.teilnahme = teilnehmer;
    if (! abstimmen(beginn, dauer)) {
        System.out.println("Termin bitte verschieben!");
    }
    else {
        for (int i=0; i<teilnahme.length; i++)
            teilnahme[i].teilnahmeSetzen(this);
    }
}
```

Zustandsw
echsel

Zustand
unklar

Zustand
in_konflikt

Zustand
konfliktfrei

- ▶ Anwendungsgebiet: Prüfen von Aufrufreihenfolgen
- ▶ Codegenerierung von Implementierungen aus Zustandsmodell:
 - Implementierungsmuster ImplicitTracingState, ExplicitTracingState, State (aber ohne Aktionen)
 - Nur zur Ableitung von Prüfcode! Zustandsmodell liefert Information für Teilaspekte des Codes (zulässige Reihenfolgen), keine vollständige Implementierung
- ▶ Praktische Aspekte:
 - In der Analyse zur Darstellung von Geschäftsprozessen und -regeln
 - komplexen Lebenszyklen für Geschäftsobjekte (Modellierung mit Sichten, die jeweils durch eine Protokollmaschine beschrieben werden)
 - Nützlich für den Darstellung von Klassen mit komplexen Regeln für die Aufrufreihenfolge
 - Hilfreich zur Ableitung von Status-Informationen für Benutzungs-Schnittstellen
 - Hilfreich zum Definieren sinnvoller Testfälle für Klassen