Kurs OMSI
im WiSe 2014/15

Objektorientierte Simulation mit ODEMx

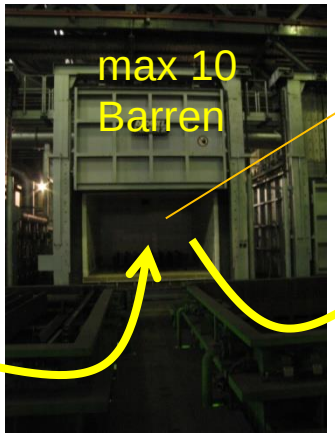
Prof. Dr. Joachim Fischer
Dr. Klaus Ahrens
Dr. Markus Scheidgen
Dipl.-Inf. Ingmar Eveslage

fischer|ahrens|eveslage@informatik.hu-berlin.de

Letzte Vorlesung

1. Systemsimulation – was ist das?
2. Ein Blick zurück in die Anfänge
3. Modelle und Originale
4. Modellierungssprachen, Simulationsumgebungen
5. Bespiele aus der aktuellen Forschung
6. Paradigma der objektorientierten Modellierung
7. Einordnung von UML
8. Klassifikation dynamischer Systeme
9. M&S eines Niedertemperaturofens

Beschickung eines Niedrigtemperaturofens (Wdh.)



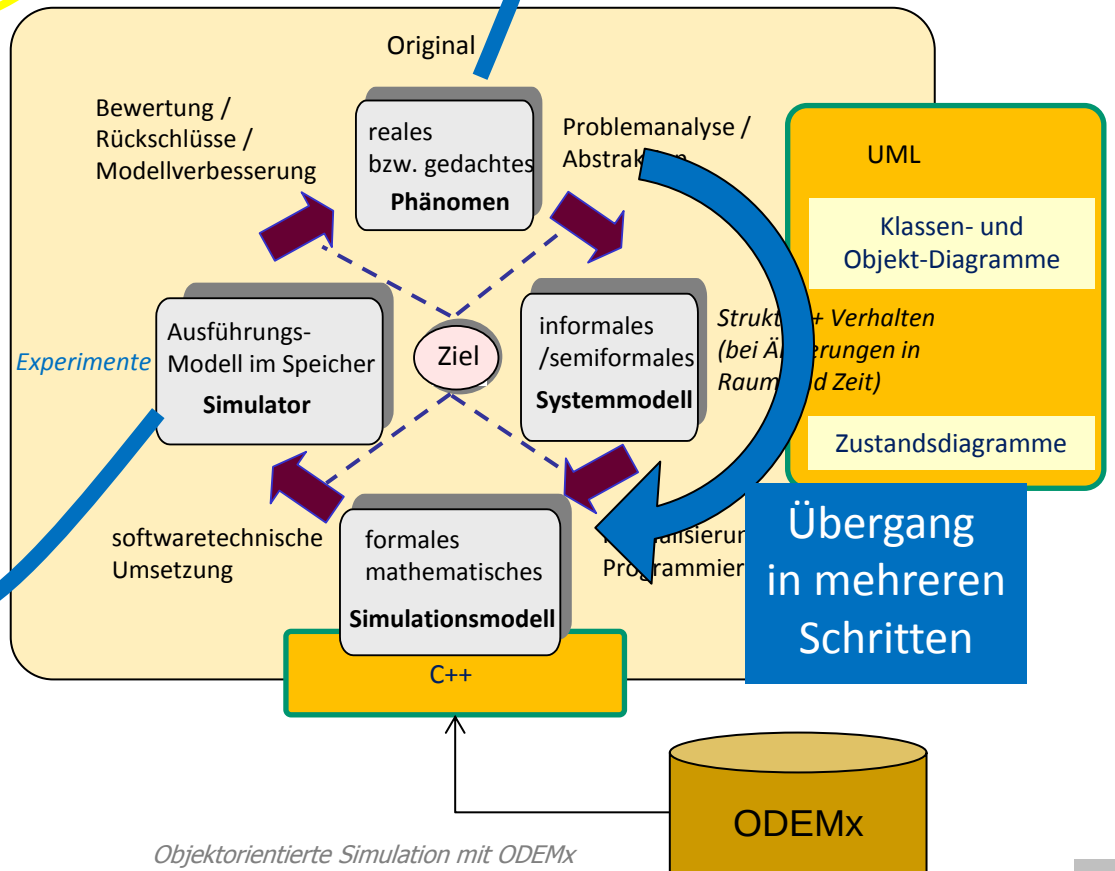
25°C

~200°C

max.
500°C

380°C

Ziel: Bestimmung der Frequenz ankommender Eisenbarren für optimalen Ofendurchsatz (bei begrenztem Puffer)

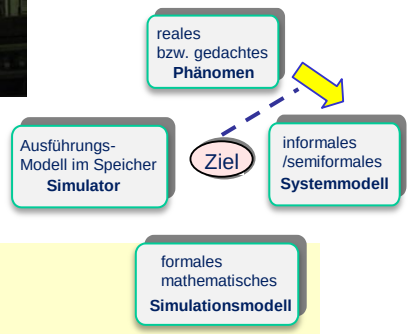


bei Animation der Abläufe

Übergang in mehreren Schritten

1. Schritt: Problemanalyse (Wdh.)

- allgemein -



mit informaler Darstellung des Phänomens als System

(aus systemtheoretischer Sicht)

bei Identifikation von

- Systemelementen und Systemumgebung
- Relationen (Wechselwirkungen) zwischen Systemelementen untereinander und zur Umgebung

zur Erbringung des bereits bestimmten

- Systemzwecks / Untersuchungsziels

Ziel: Bestimmung der Frequenz ankommender Eisenbarren für maximalen Ofendurchsatz (bei begrenztem Puffer)

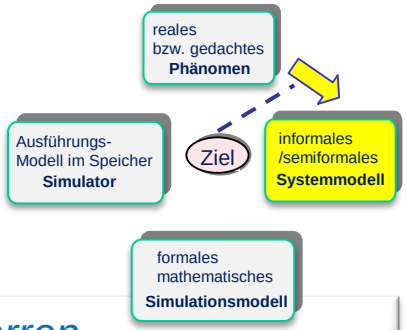
zu hoch:
- längere Wartezeiten der Barren
- Abkühlung,

➔ längere Verweilzeit im Ofen

zu niedrig:
- Im Ofen bleiben Plätze frei

2. Schritt: Problemanalyse (Wdh.)

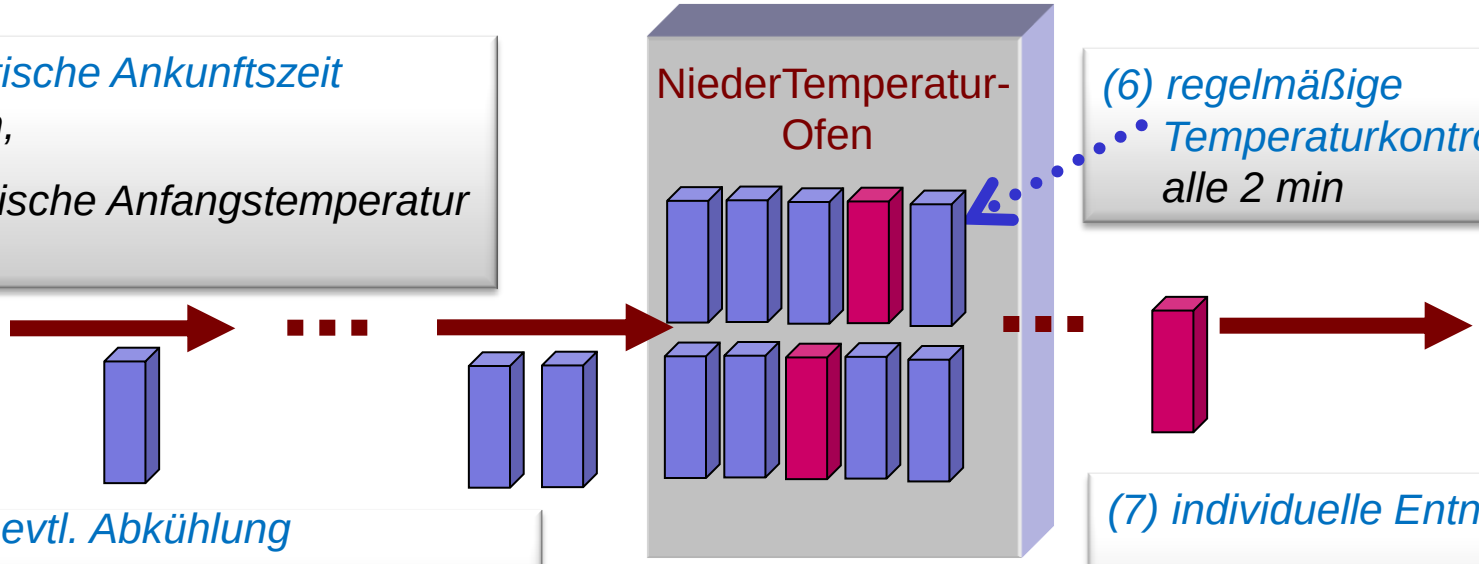
- Informale Darstellung -



(5) Änderung der Ofentemperatur
in Abhängigkeit der Belegung
(maximal 500°C)

(4) Erwärmung der Barren
auf Zieltemperatur von mind. 380 °C

(1) stochastische Ankunftszeit
2-10 min,
stochastische Anfangstemperatur
~200 °C



(6) regelmäßige Temperaturkontrolle
alle 2 min

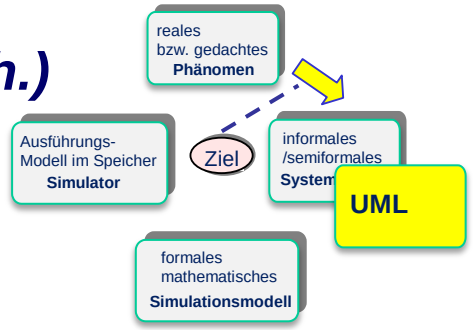
(2) evtl. Abkühlung
auf Umgebungstemperatur
25 °C möglich

(3) Beschickung
Ofen hat begrenzte Aufnahmekapazität
von 10 Barren

(7) individuelle Entnahme
und
(8) Nachfüllung

3. Schritt: partielle Modellformalisierung (Wdh.)

- UML-Notation -



informal beschriebenes Systemmodell

- **Struktur** (Objekt- und Klassendiagramm) und
- **Verhalten** als Ensemble asynchron kommunizierender Automaten

  Zustandsdiagramm

bei Identifikation

- verwendeter **Modellklassen**
- und von **Zustands-** und **Bewertungsgrößen**

bei (zunächst noch) verbaler **Verhaltensbeschreibung**

passive Klassen

Assoziationen

aktive Klassen

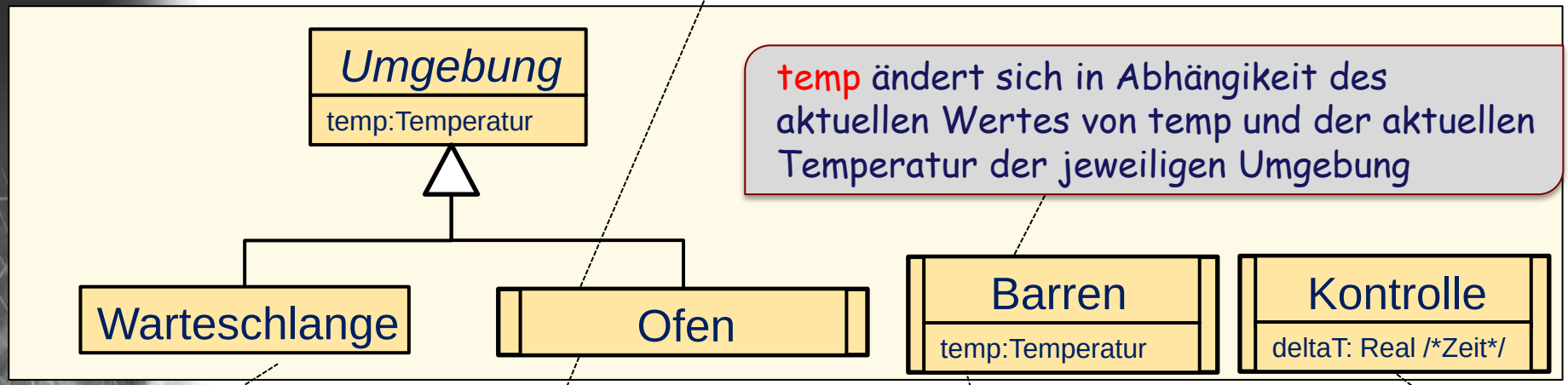
lifeCycle

zeitdiskrete Zustandsänderungen

zeitkontinuierliche Zustandsänderungen

Klassendiagramm (Wdh.)

temp ändert sich in Abhängikeit des aktuellen Wertes von temp und der aktuellen Temperaturen der zugeordneten Barren



temp ändert sich in Abhängikeit des aktuellen Wertes von temp und der aktuellen Temperatur der jeweiligen Umgebung

Lebenslauf ~
Temperaturänderung einer individuellen Warteschlange
(aber hier Annahme:
konstante Temperatur)

Lebenslauf ~
Temperaturänderung eines individuellen Ofens

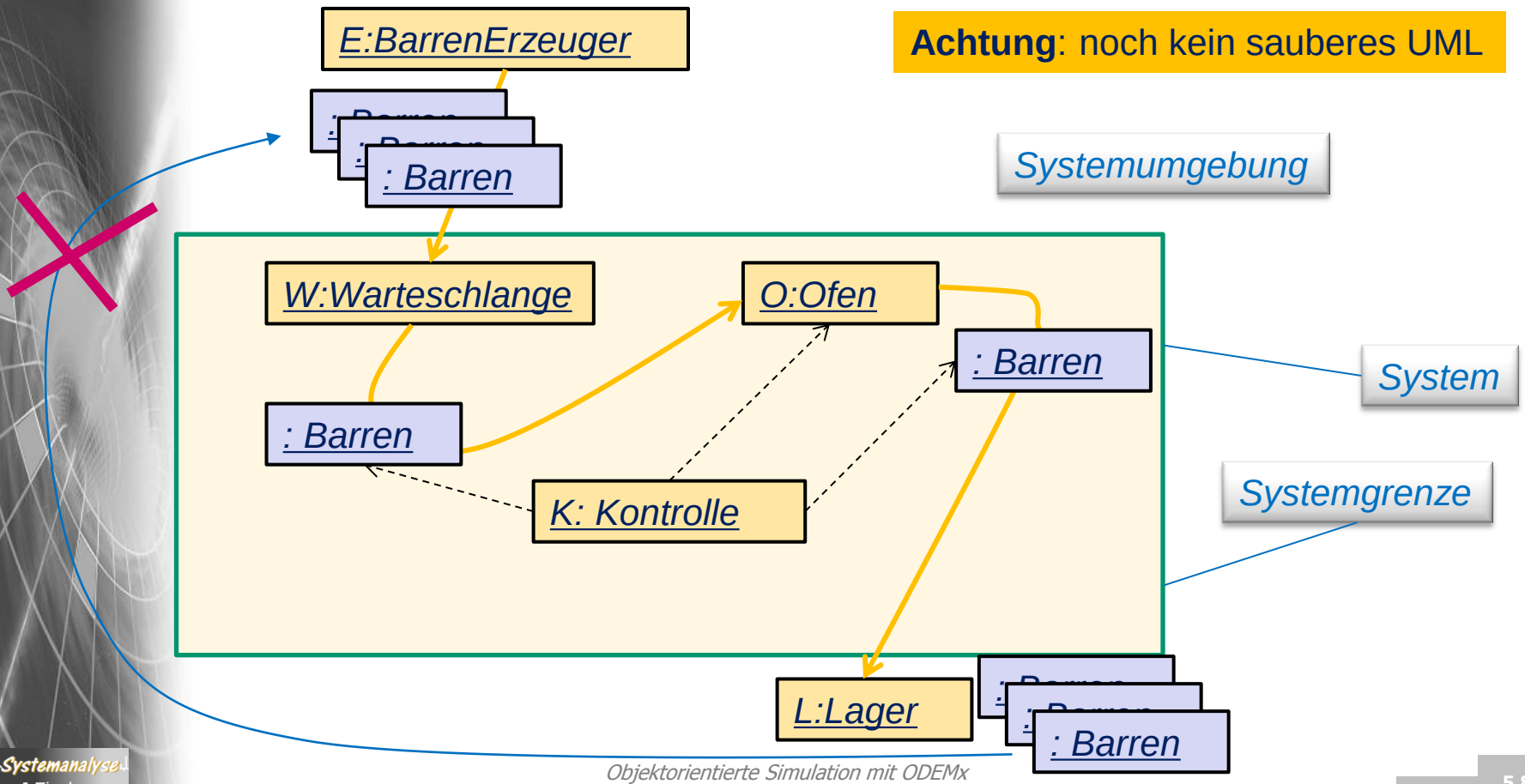
Lebenslauf ~
Temperaturänderung eines individuellen Barrens

Lebenslauf ~
regelmäßige Temperaturkontrolle und evtl. Bewegungen der Barren (Änderung der Zuordnung)
Annahme:
Bewegungen sind zeitlos und ressourcenlos

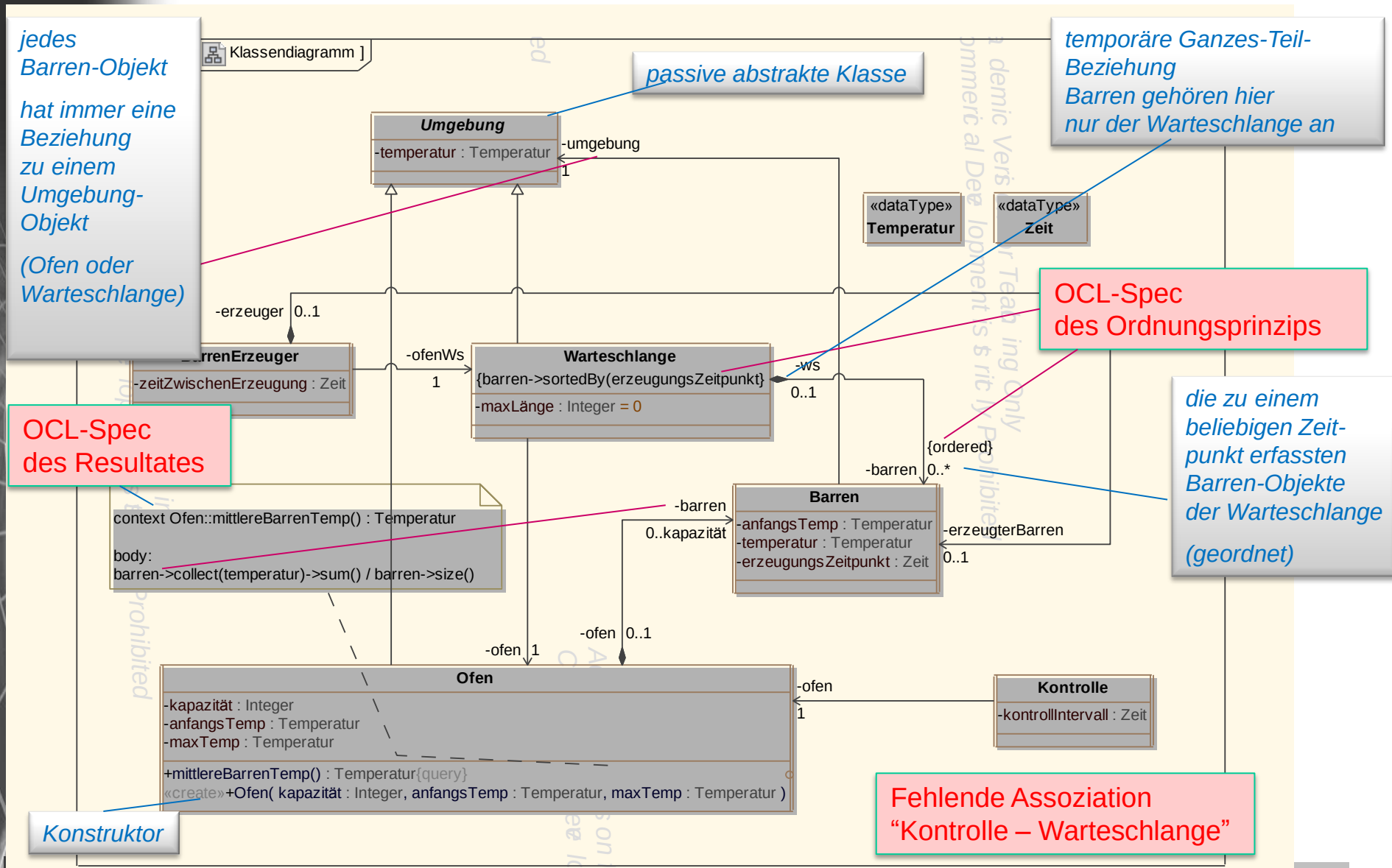
Barren-Objekte stehen mit
Objekten konkreter Umgebungs-Ableitungsklassen
in Wechselwirkung
die Zuordnung erfolgt dabei dynamisch

System (Umgebung, Grenze, Systemkomponenten)

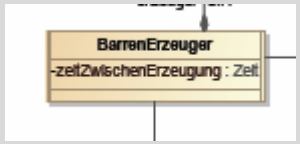
Was ist zu prüfen?
- können die Systemelemente in ihrer Wechselwirkung die Systemfunktionalität erbringen?
- Sind die Wechselwirkungen mit der Umgebung rückkopplungsfrei?



Klassendiagramm *(mit Assoziationen)*



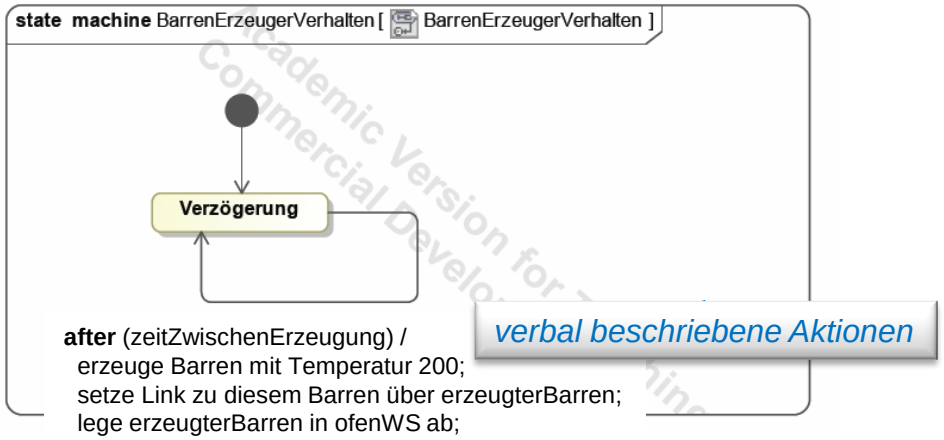
Verhaltensdiagramm (BarrenErzeuger, Kontrolle)



explizite Attribute

- erzeugterBarren: Barren
- ofenWS: Warteschlange
- ofen: Ofen

implizite Attribute



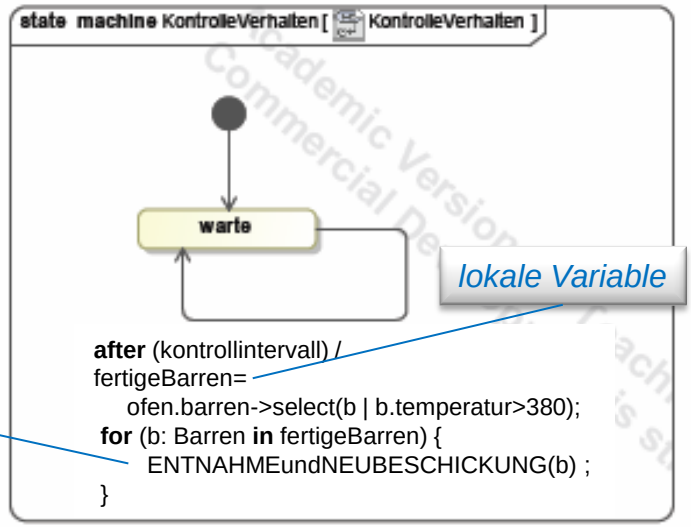
explizite Attribute

- ofen: Ofen
- ofenWS: Warteschlange

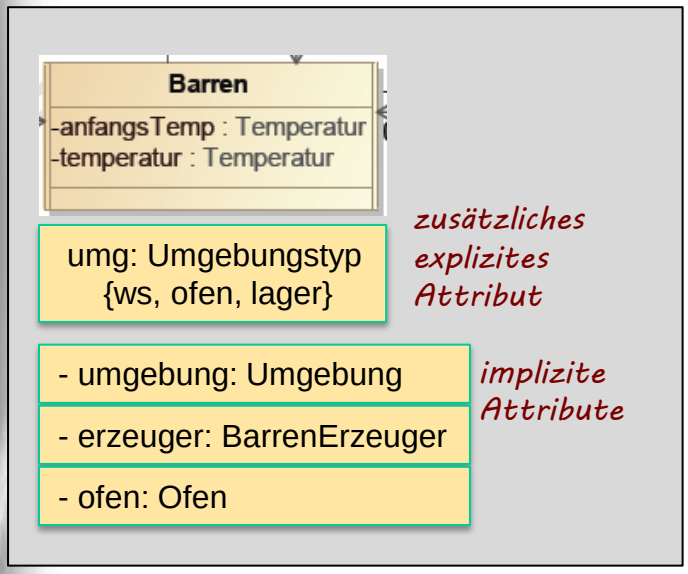
implizite Attribute

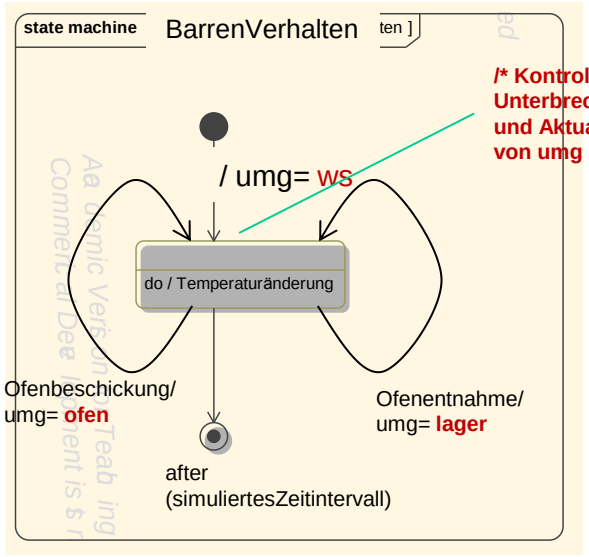
for-Schleife

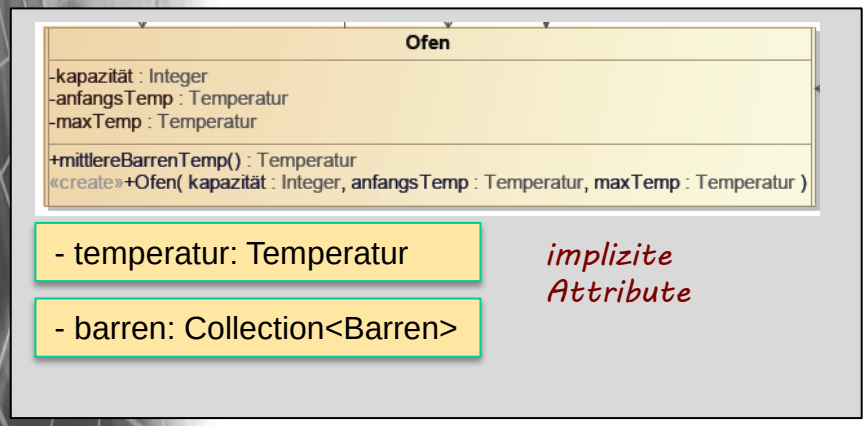
und Prozedur aus angenommener UML-Actionsprache

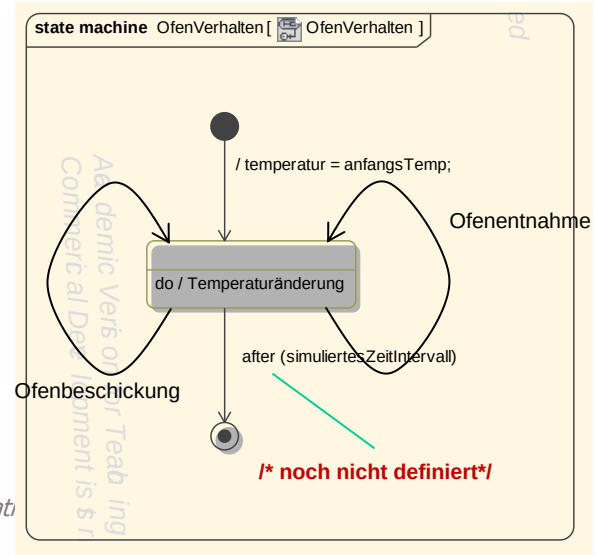


Verhaltensdiagramm (Barren, Ofen)

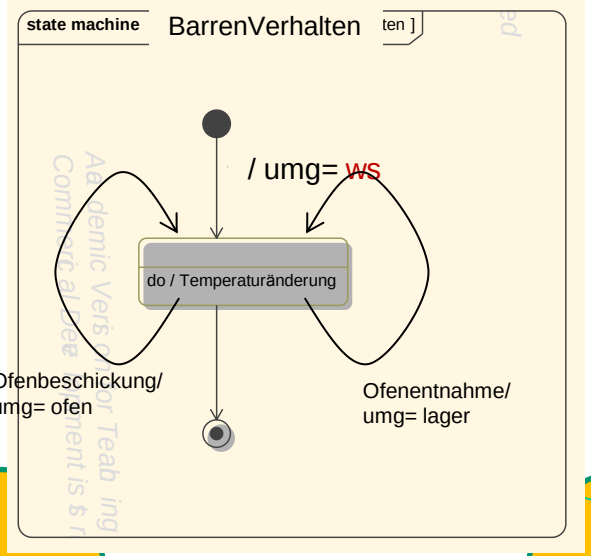








Barren: Temperaturänderung



umg= ws

x
[^{0}C]
x(t)
25
t [min]

Ofenbeschickung-Ereignis

Umgebungstemperatur
u(t)= 25

umg= ofen

x(t) [^{0}C]
400
25
t [min]

Erzeugungs-Ereignis

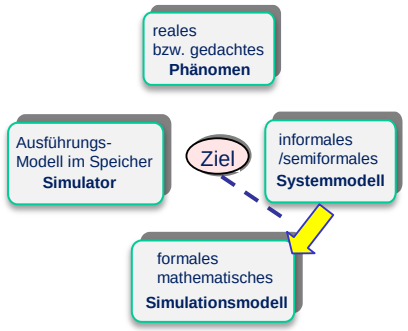
Ofenbeschickung-Ereignis

Ofenentnahme-Ereignis

~ für den betreffenden Barren

4. Schritt: Vervollständigung der Formalisierung

- Temperaturänderung als DGL -



negativer Wert für x'(t) → **Reduktion von** x(t)
Änderung von x(t):
- zunächst stark,
- dann schwach,

➔ asymptotische Annäherung an den Wert 25

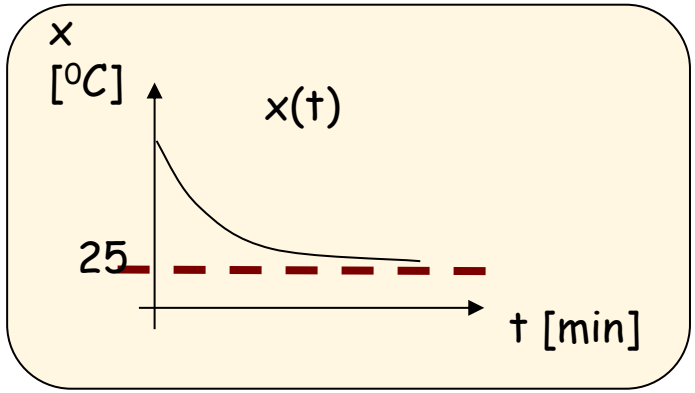
umg= ws

Barrentemperatur x(t)

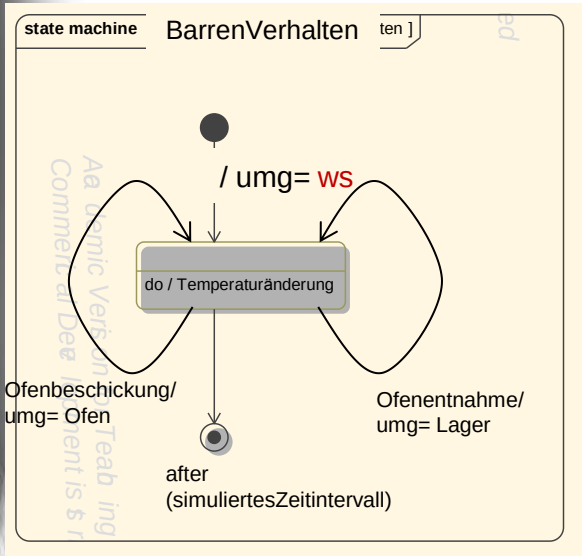
$x'(t) = \{ 25 - x(t) \} / 7$

25 - 200

Anfangstemperatur: ca. 200 ^{0}C



4. Schritt: Vervollständigung der Formalisierung
- Temperaturänderung als DGL -



umg= ofen

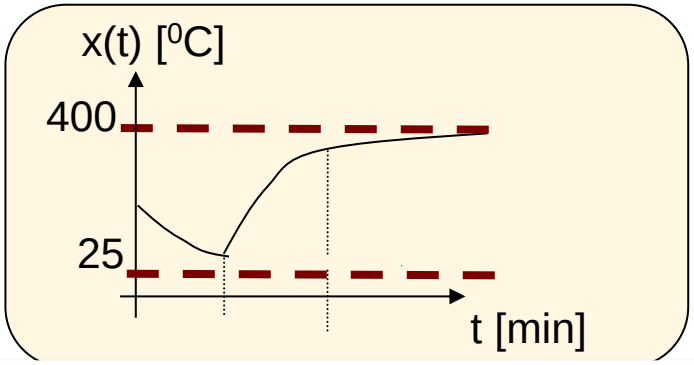
Positiver Wert für x'(t) → **Erhhöhung von** x(t)
Änderung von x(t):
- zunächst stark,
- dann schwach,

➔ asymptotische Annäherung an den Wert y

Barrentemperatur x(t)

$x'(t) = \{ y(t) - x(t) \} / 7$

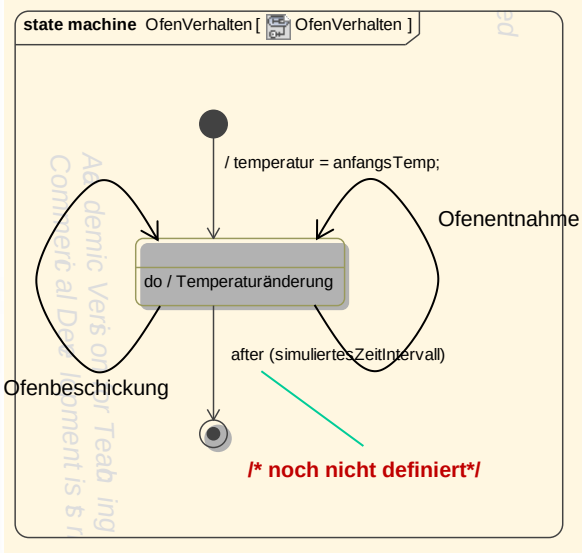
400 - 200



aber: Ofentemperatur y(t) ist nicht konstant, eine explizite Darstellung gibt es nicht, deshalb kann man die Zeit, wie lange ein beliebiger Barren im Ofen benötigt, nicht vorab berechnen.

4. Schritt: Vervollständigung der Formalisierung

- Temperaturänderung als DGL -



Energiezufuhr: max. 500°C

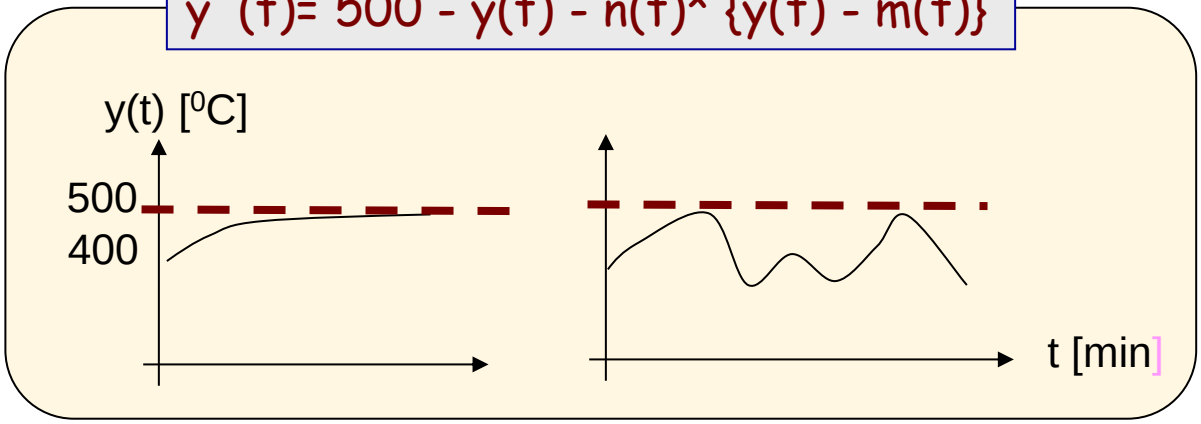
n(t): Anzahl von Barren im Ofen

m(t): mittlere Temperatur der Barren im Ofen

Anfangstemperatur: 400°C

Ofentemperatur y als y(t)

$$y'(t) = 500 - y(t) - n(t) * \{y(t) - m(t)\}$$



4. Schritt: Modellformalisierung (nun abstrakt vollzogen)
- Festlegung der Art der Zustandsänderung → Verhaltensklassen

von
- Struktur (Objekt- und Klassenstruktur) und
- Verhalten

bei Identifikation
- verwendeter Modellklassen
- und von Zustands- und Bewertungsgrößen

bei Bestimmung
- der Verhaltensfunktionen/-Gleichungen

passive Klassen

aktive Klassen

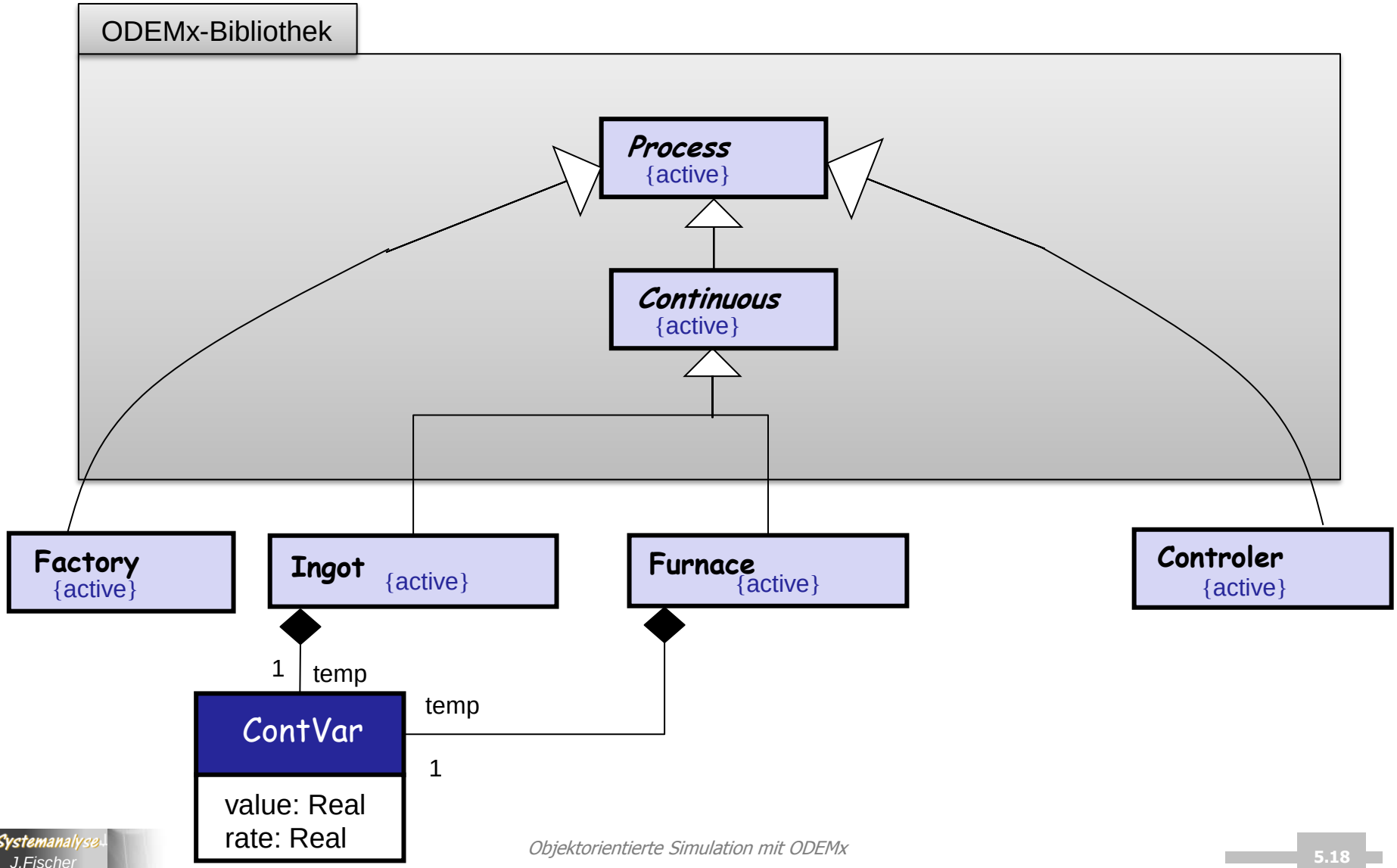
lifeCycle

zeitdiskrete Zustandsänderungen

zeitkontinuierliche Zustandsänderungen

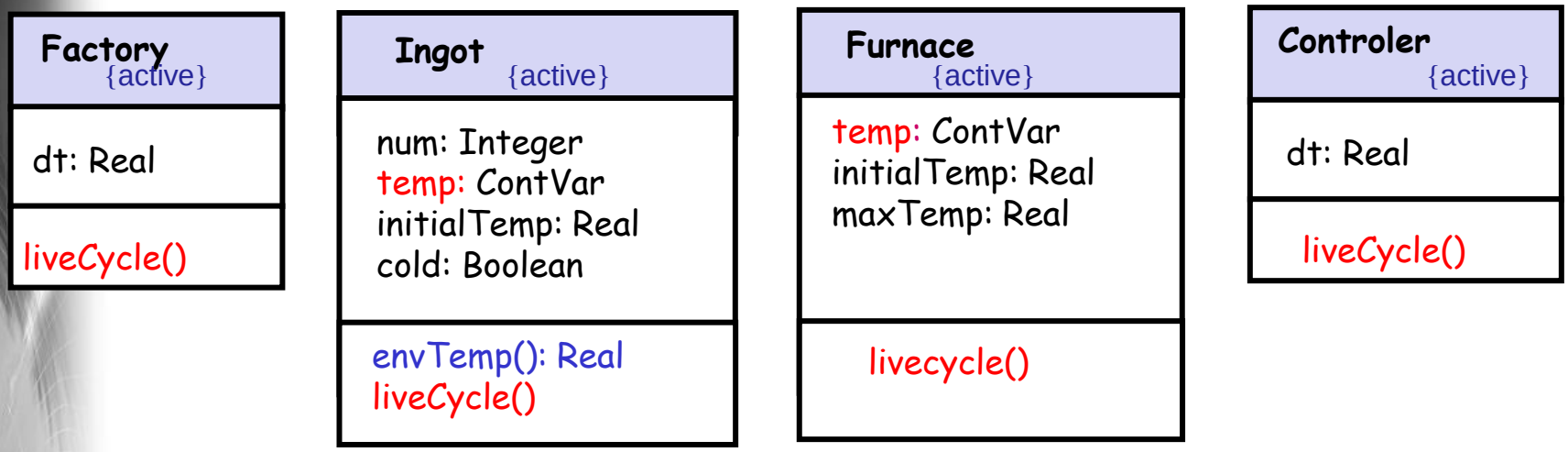
UML bietet Zustandsmaschine zur Beschreibung an, aber semantische Präzisierungen sind erforderlich

5. Schritt: Modellformalisierung
- Modell in einer ausführbaren Sprache
- in ODEMx



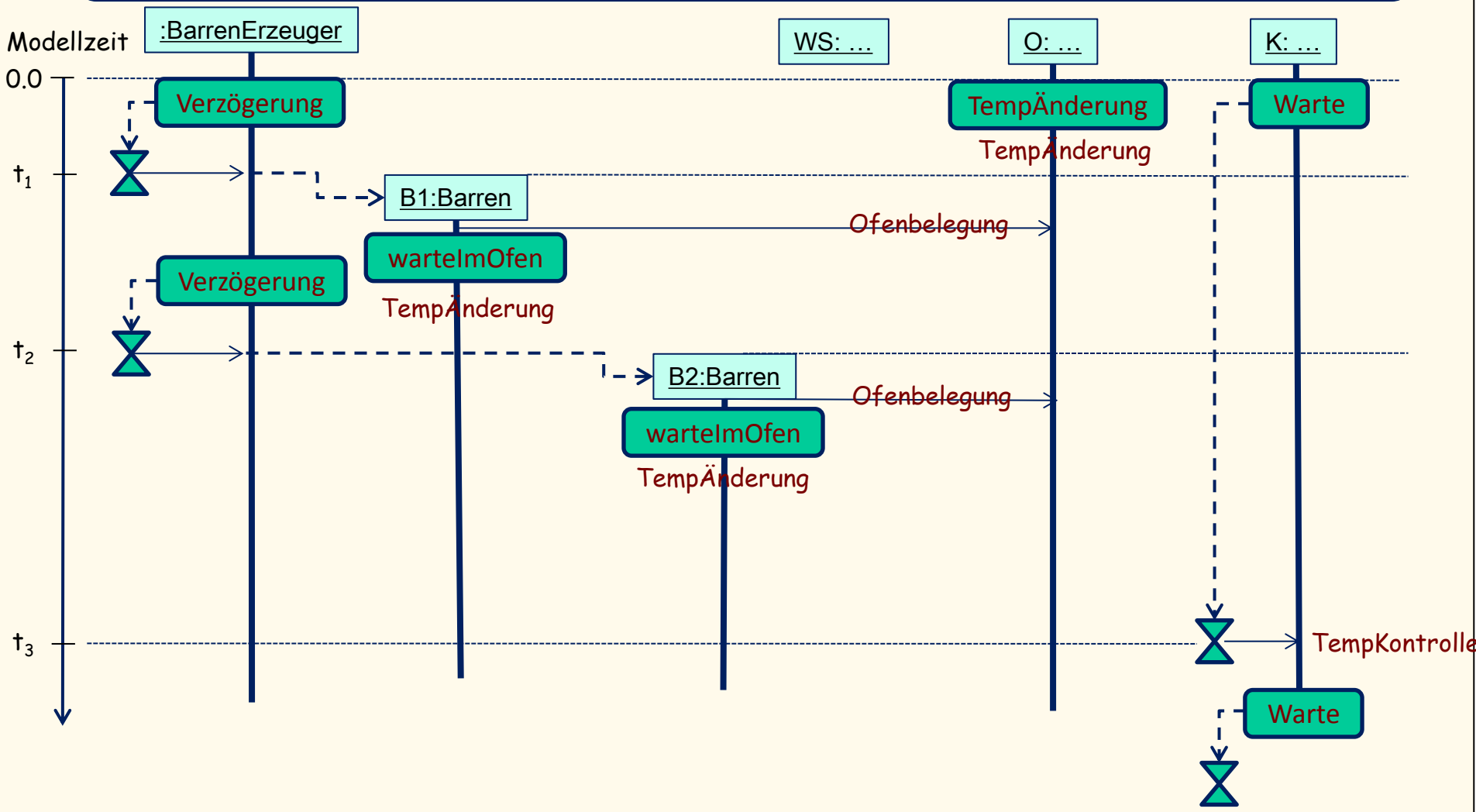
5. Schritt: Modellformalisierung

- Identifikation von Modellgrößen der Strukturelemente



Laufzeitverwaltung *auf einer Ein-Prozessormaschine*

Vor.: alle statischen Systemstrukturelemente sind als Objekte generiert:
Objekte aktiver Klassen führen ihre Starttransitionen zur Modellzeit 0.0 aus.



2. Prinzip der Next-Event-Simulation

1. Charakterisierung der Next-Event-Simulation
   - Ereignisse, Next-Event-Scheduler
   - Barren-Beispiel (ohne Verwendung aktiver Klassen)
   - Zusammenhang von ereignisbasierter und prozessbasierter Modellbeschreibung

2. Umsetzung des Prinzips in ODEMx
   - Aufbau von ODEMx (erster Blick)
   - Triviales Clock-Beispiel

Basis für sequentielle Abwicklung paralleler Vorgänge *(Computersimulation)*

Chronologische Sortierung der Ereignisse beginnend mit einem minimalen Zeitpunkt (0.0)

- führt zu einer festen **Reihenfolge**:
  bei Einsatz von Regeln zum Umgang mit gleichzeitigen Ereignissen

  Reihenfolge von Ereignissen zu einem Zeitpunkt ist

  1. beliebig, Änderung hat keinen Einfluss auf resultierenden Gesamtzustand des Systems
  2. Änderung hat einen Einfluss auf resultierenden Gesamtzustand des Systems
     - 2a. aber Reihenfolge ist bestimmt durch Kausalität (oder Priorität)
     - 2b. Reihenfolge ist nichtdeterminiert,

- **Regeln**

  ad 1): unkontrollierte Reihenfolgebestimmung

  ad 2a): explizite Sortierung der Ereignisse (oder Ereignisklassen)

  ad 2b): stochastische Auswahl (sichert einen Wechsel bei Wiederholung der Ereigniskonstellation)

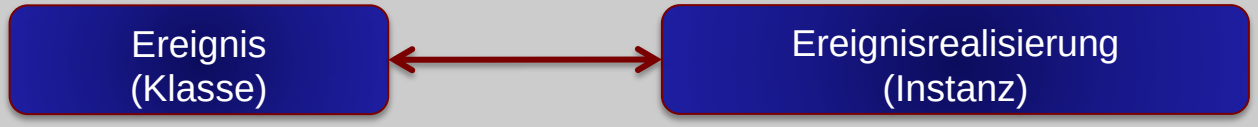
  oder

  Backtracking bei weiterer Verfolgung aller unterschiedlichen Systemzustände (Zustandsexplosion, scheidet bei großen Systemen aus)

Diskrete Ereignissimulation: Begriffe

Verhaltensnachbildung eines Systems per diskreter Ereignissimulation
ist das Resultat der Realisierung von Ereignissen in chronologischer Reihenfolge
bei Voranschreiten der Modellzeit und Auflösung von Gleichzeitigkeit

Ereignis: Menge von Aktionen,
die Systemveränderungen zu einem diskreten Ereigniszeitpunkt bewirken,
wenn es zu einer Ereignisrealisierung kommt



Sichere und unsichere Ereignisse → Bedingungen für das Eintreten von Ereignissen

- Zeitbedingungen (→ Zeitereignisse)
- Zustandsbedingungen (→ Zustandsereignisse)
- Wahrscheinlichkeit (→ stochastisches Ereignis)
  a) ob Ereignis zu einem Zeitpunkt überhaupt auftritt
  b) Zeitspanne bis zum Eintritt unterliegt einer Verteilungsfunktion

Sonderfall

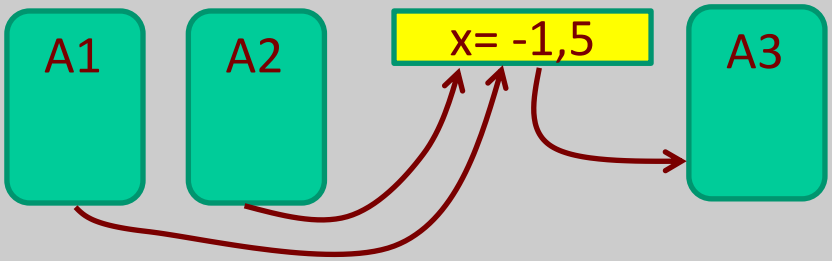
Bedienungssysteme sind klassische Vertreter von Systemen,
deren Verhalten sich durch sequentielle Ereignissimulationen beschreiben lassen

typische Ereignis-Klassen

- Ankunft von Barren, Kunden, … ,
- Beginn oder Ende einer Bedienung,
- Unterbrechung oder Fortsetzung einer Bedienung

Behandlung von Zustandsereignissen

Zentrales Problem



Ann.-1:
A3 verharrt in einem Zustand S1 mit Trigger für Zustandsereignis:
when x > 3.5 / op1
bei Übergang nach S2 (der Zeitpunkt ist nicht bekannt)

Ann.-2: - A1 und A2 haben zum aktuellen Zeitpunkt Trigger (Zeitereignisse), die eine Ausführung von Aktionen und einen Zustandswechsel zur Folge haben.
- Beide Automaten erhöhen dabei x jeweils um einen positiven Betrag.

Art der Problemlösung hat
großen Einfluss auf Modellierungskomfort und Laufzeiteffizienz
von Simulatoren

Ausprägungen der Next-Event-Methode

alternatives Verfahren:
äquidistante Änderung der Modellzeituhr bei Realisierung von Zustandsänderung (häufig bei zeitkontinuierlichen Simulationen)

klassische Methode der Simulationslaufzeitsteuerung:

NEXT-EVENT-Simulation
(nicht-äquidistante Sprünge der Modellzeituhr zu Zeitpunkten, an denen Zustandsänderungen des Systems wirksam werden.

→ Sprung von Ereignis zu Ereignis

verschiedene Next-Event-Realisierungsverfahren:

- Ereignis-basiert,
- Aktivitäts-basiert
- Prozess-basiert

Implementierung kommt ohne Koroutinen-Verwaltung aus (C-Structs, Funktionen)

Implementierung folgt OO-Paradigma (Simula),
benötigt
jedoch ein Koroutinen-Verwaltung (oder Thread-Verwaltung)

Konzepte der diskreten Ereignissimulation

- SystemStruktur (Zustand)
  eine Menge von Variablen
  (ausgezeichnete Menge von Modellbeschreibungsgrößen)
  beschreibt in ihrer Belegung den Systemzustand zum aktuellen Zeitpunkt

- Uhr
  (Modellzeit, dimensionslos, monoton wachsend)

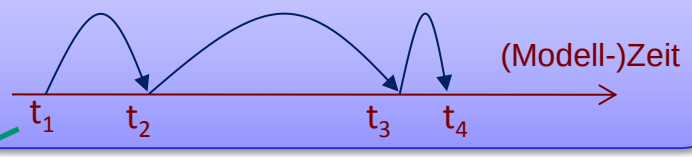
- Ereigniskalender
  a) Umgang mit Gleichzeitigkeit von Ereignissen

  b) Scheduling
  Bestimmung des nächsten Ereignisses im Kalender

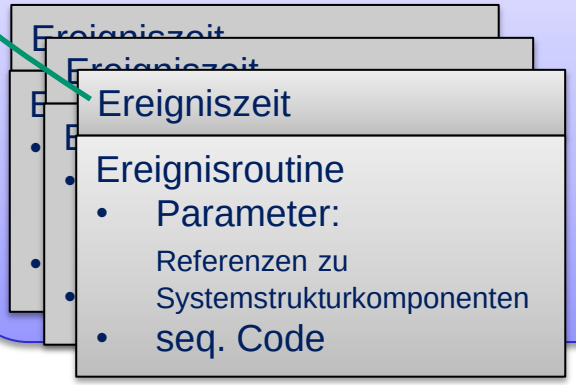
  c) Zeitfortschritt
  sobald zum aktuellen Zeitpunkt alle Ereignise ausgeführt sind, wird Uhr weitergestellt

  (aktuelle Modellzeit:= Ereigniszeit des nächsten Ereignisses)

Zustandsänderungen finden immer nur zu diskreten Zeitpunkten statt



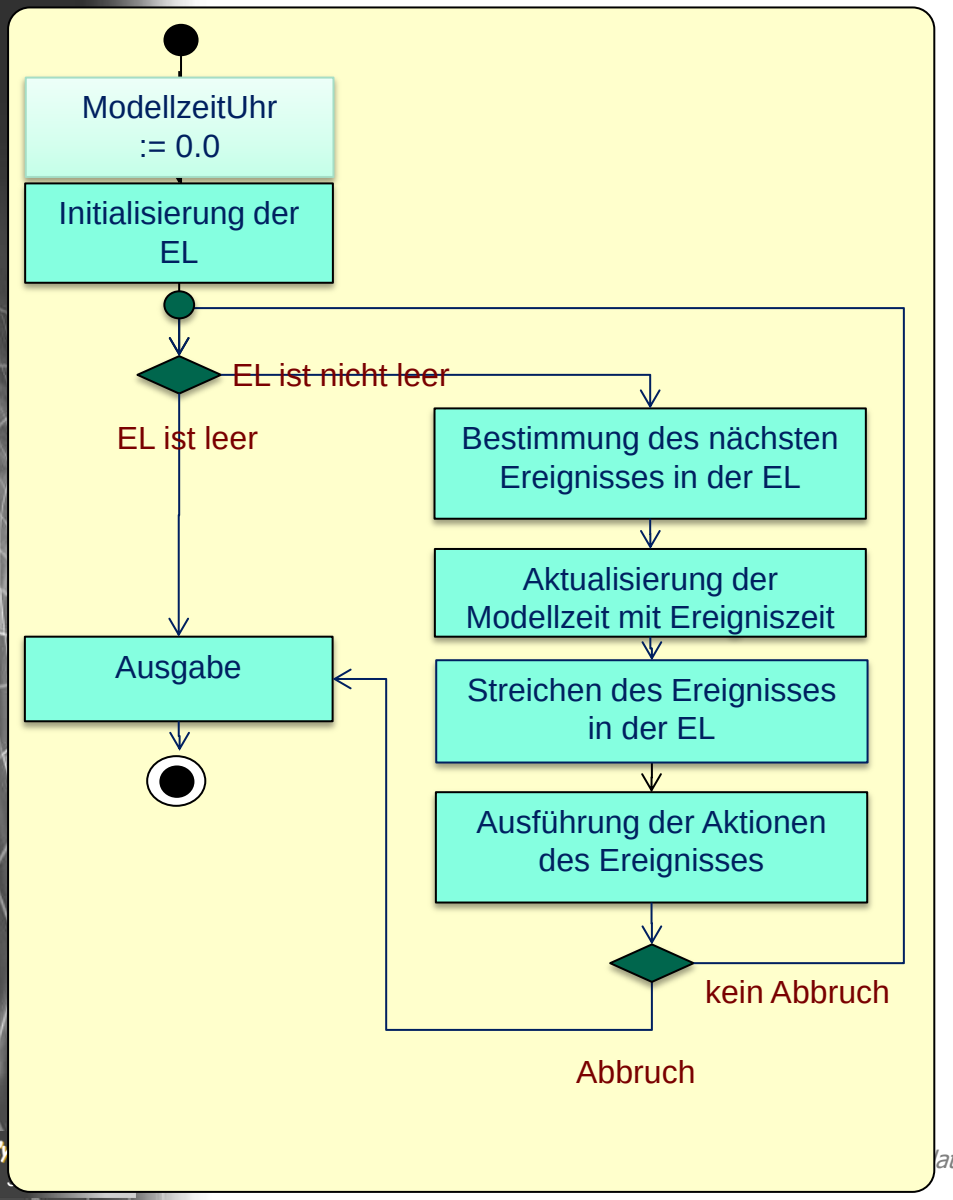
Ereignis-Chakteristik



(Ereignis-) Scheduler

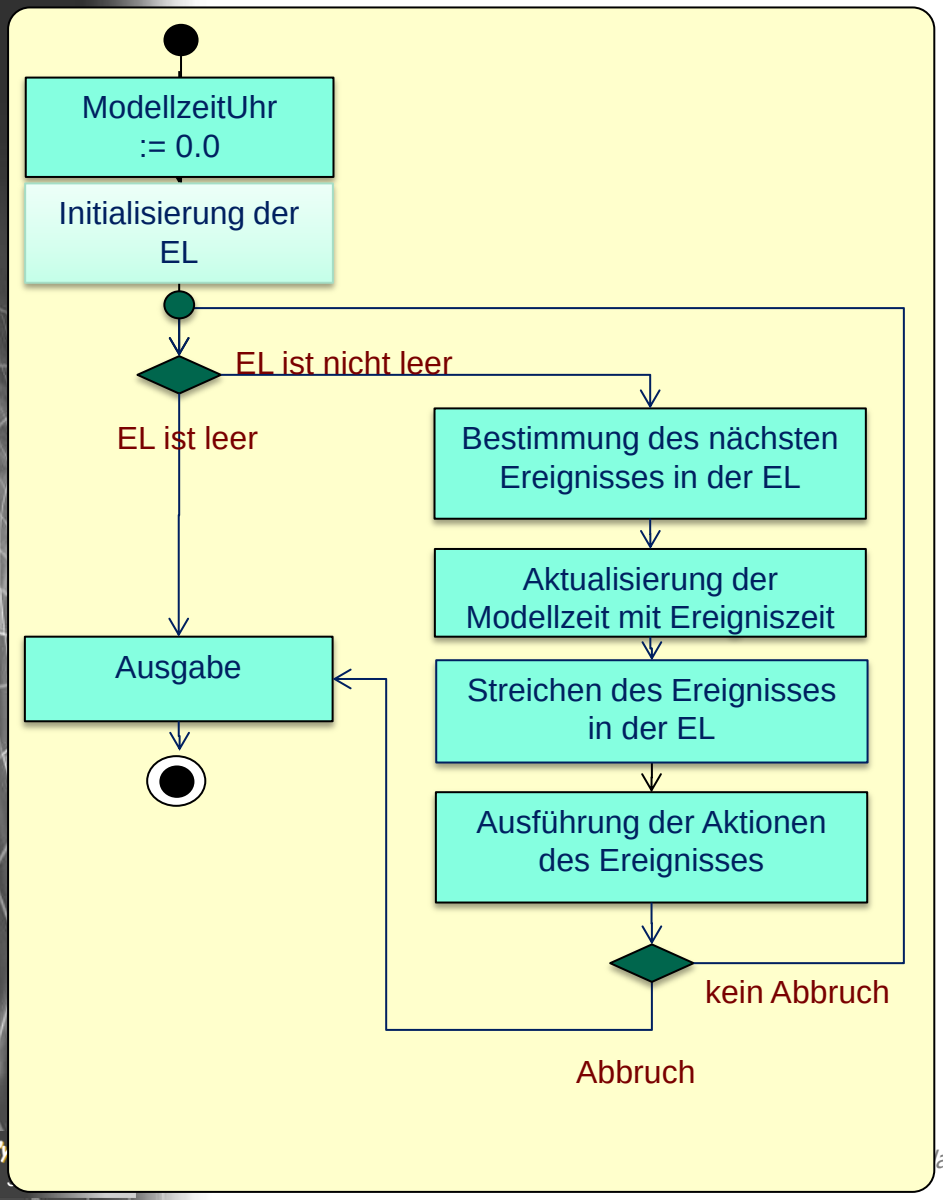
Aufgaben
- Zeitfortschritts-realisierung
- Realisierung des aktuellen Ereignisses

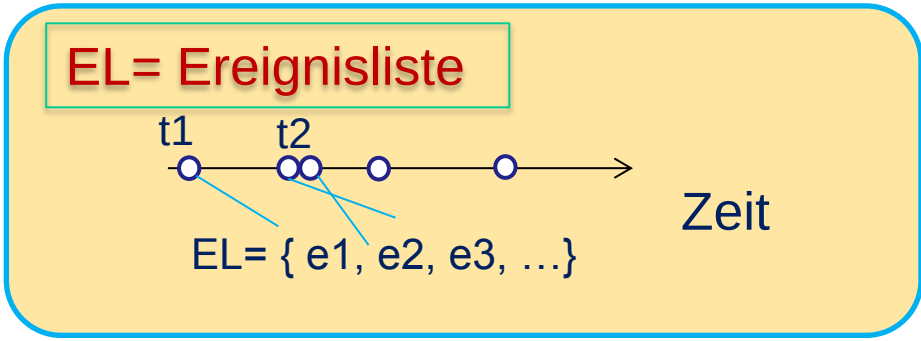
Trivialer Discrete-Event-Scheduler (Ablauf)



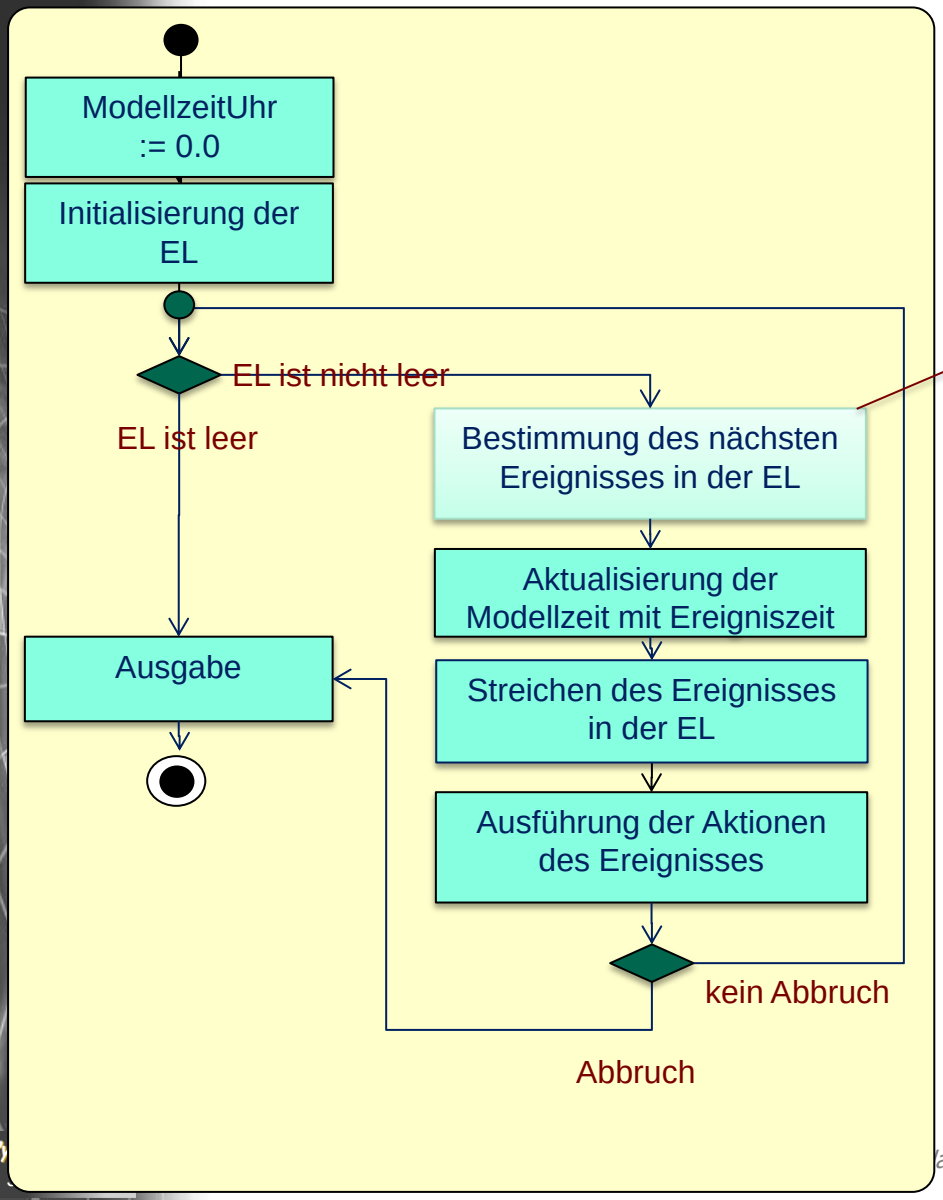
Einrichten der Verwaltungsstrukturen

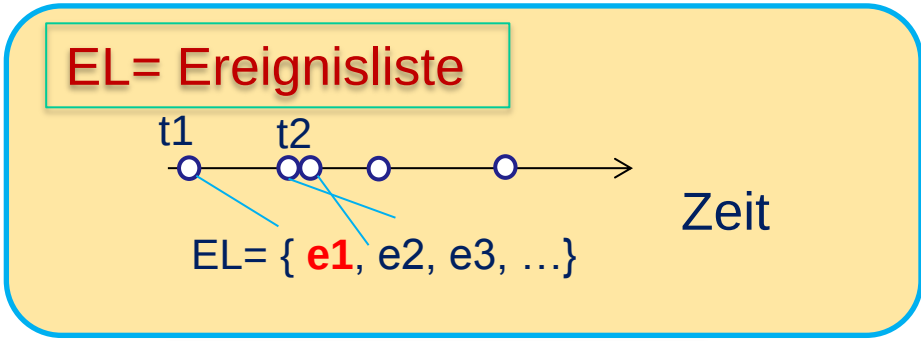
Trivialer Discrete-Event-Scheduler





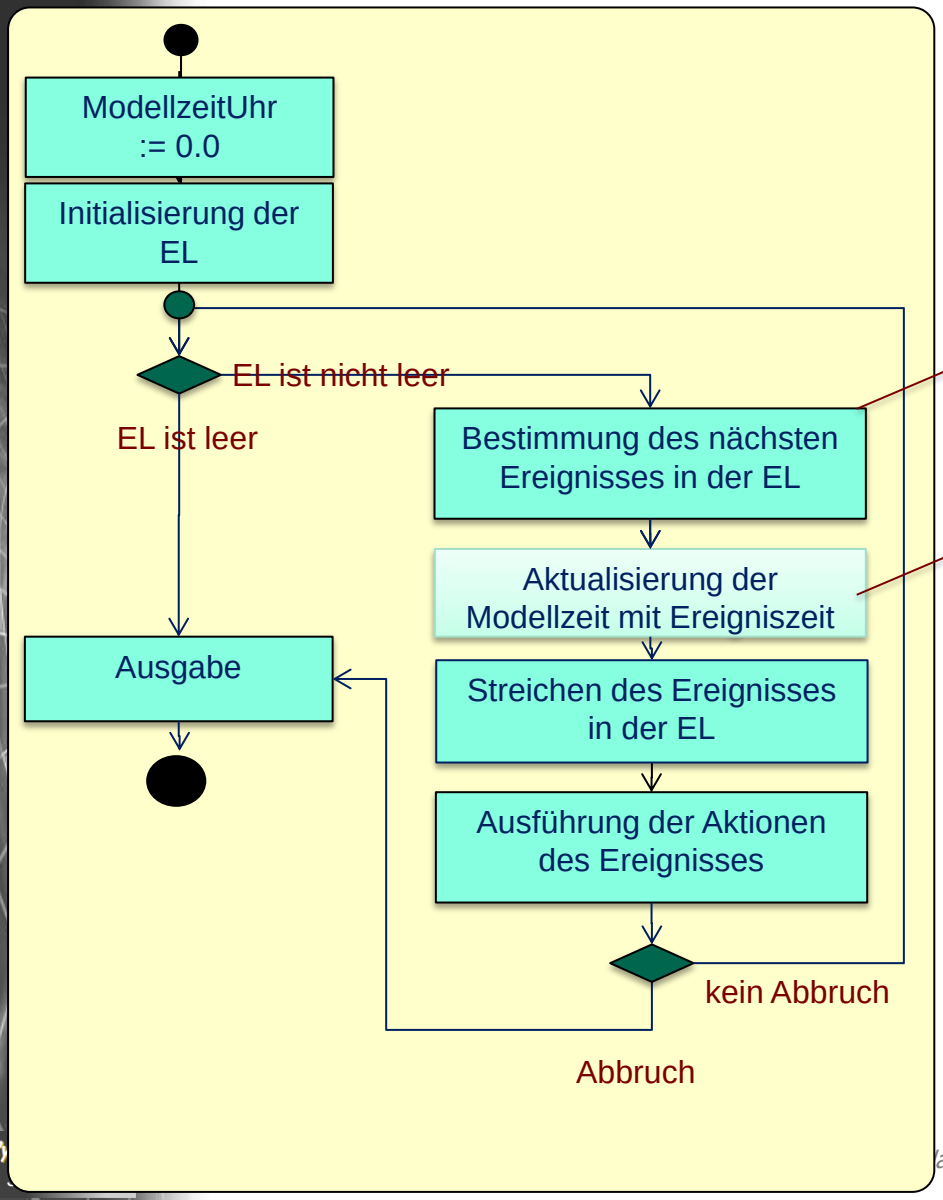
Trivialer Discrete-Event-Scheduler



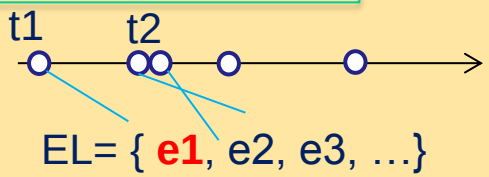


EL ist chronologisch sortiert

Trivialer Discrete-Event-Scheduler



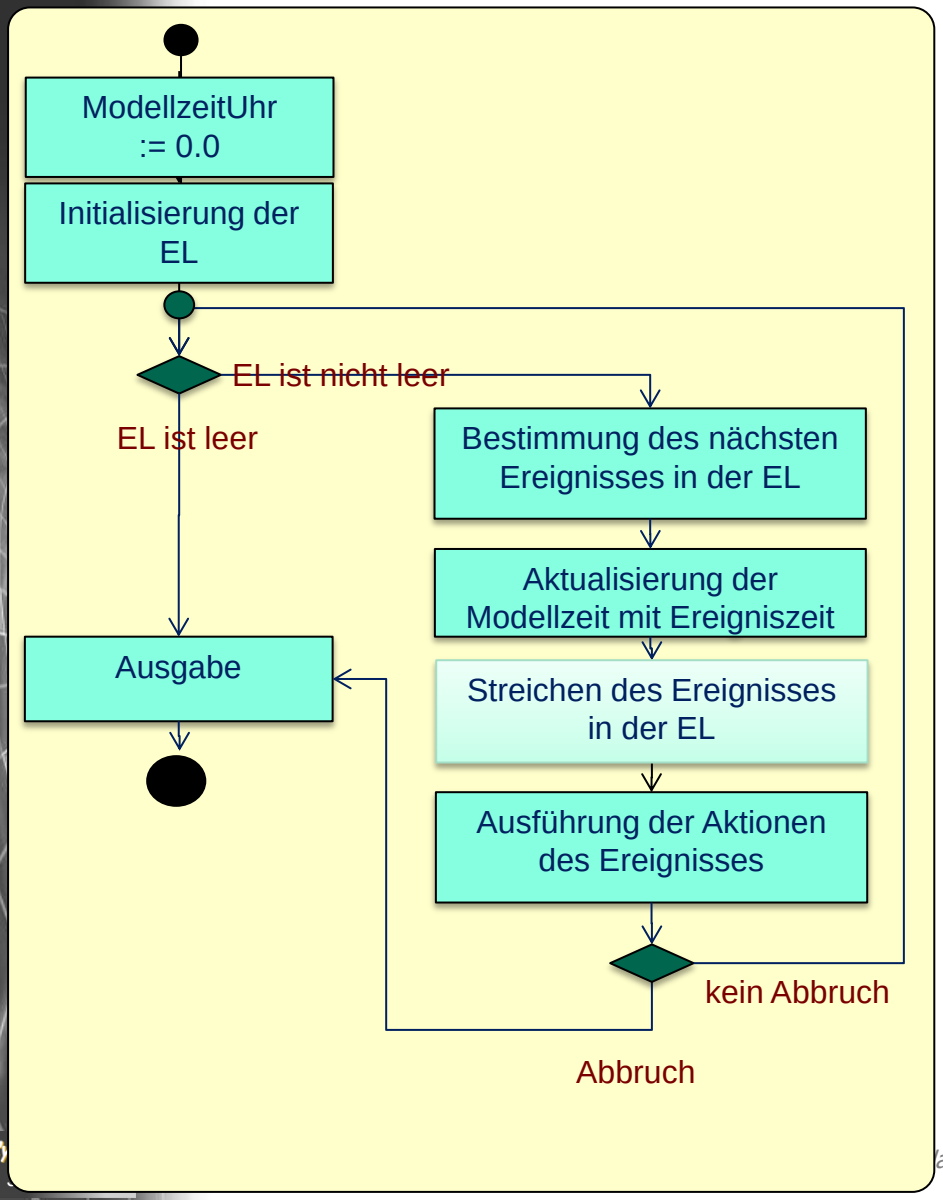
EL= Ereignisliste

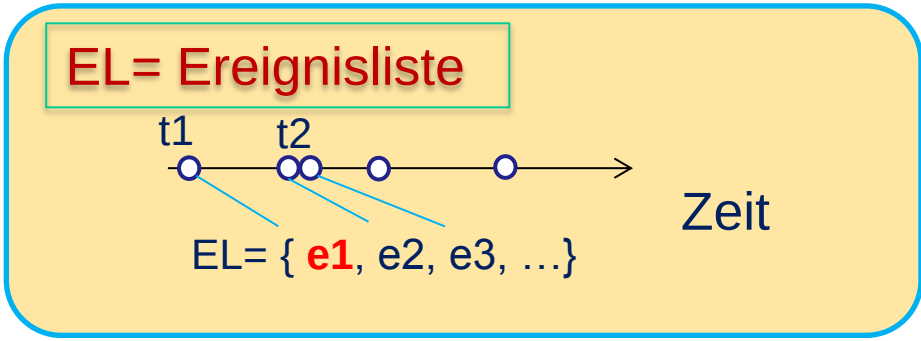


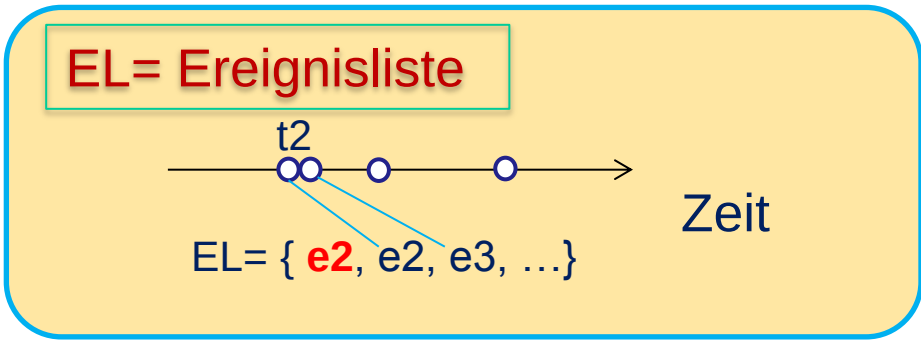
EL ist chronologisch sortiert

evtl. Zeitsprung (zeitdiskret)

Trivialer Discrete-Event-Scheduler

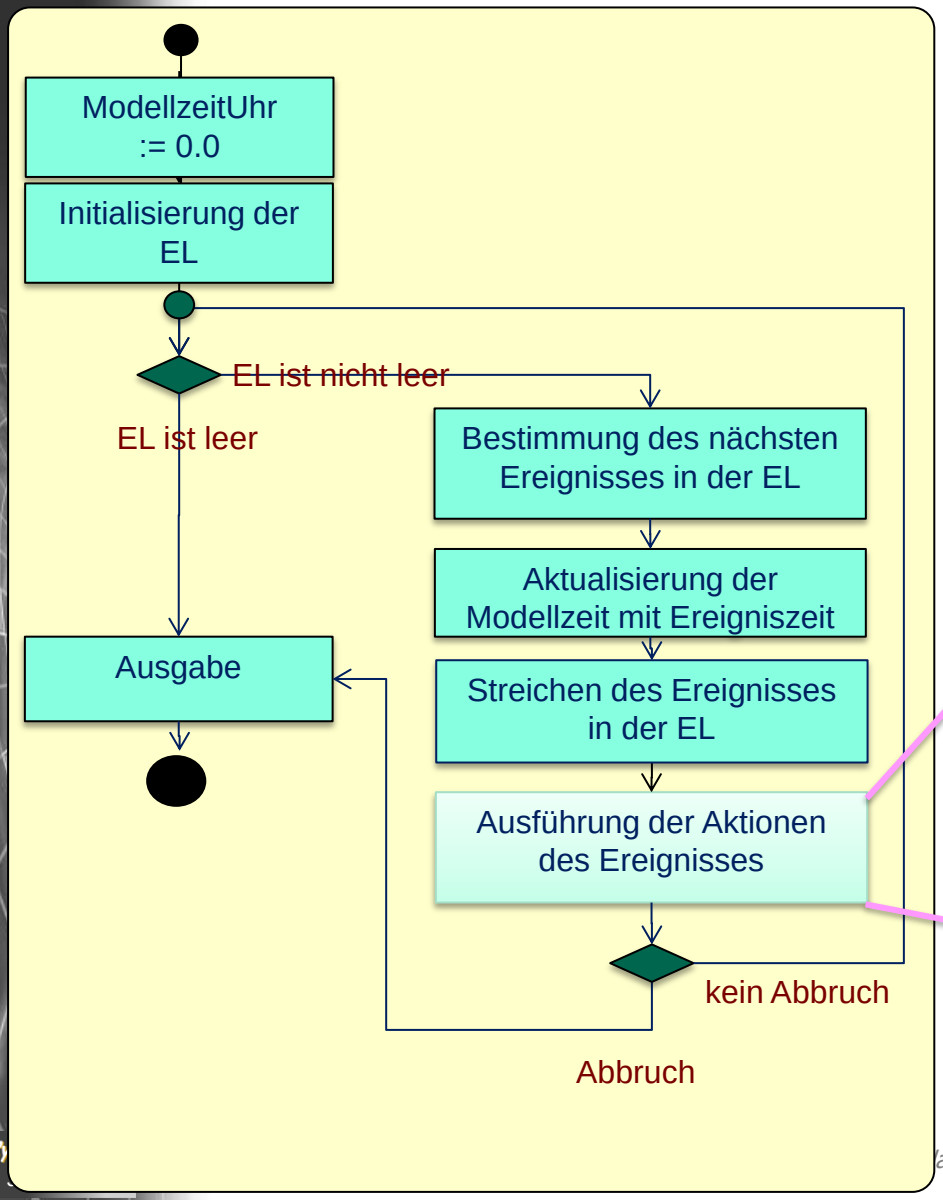




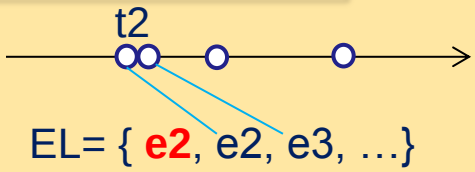


currentEvent=**e1**

Trivialer Discrete-Event-Scheduler



EL= Ereignisliste



currentEvent=**e1**

Zustandsänderungen

Klassifizierung von Ereignisklassen, um partielle Zustandsänderungen strukturell zu unterstützen

Planung neuer Ereignisse

Problem: Gleichzeitigkeit und

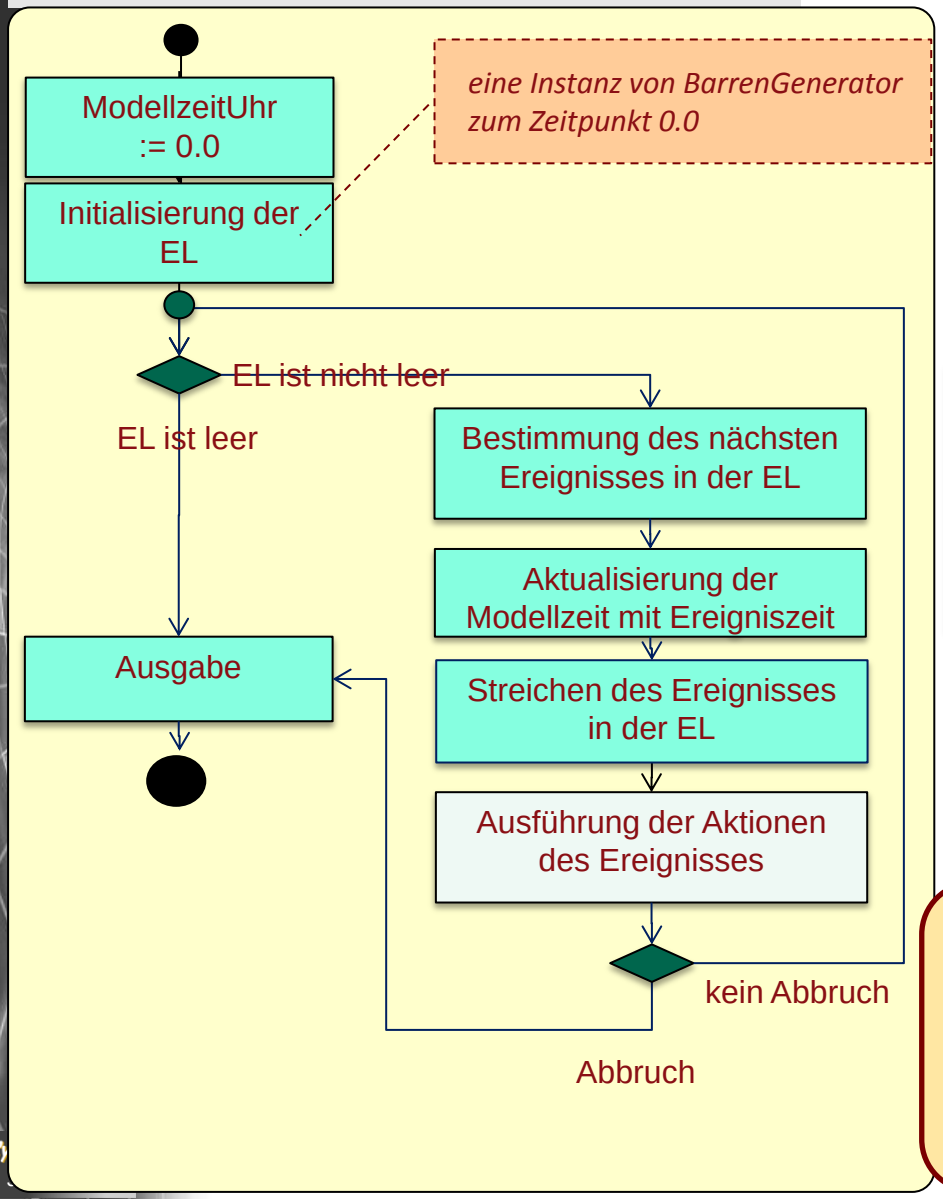
Zustandsereignisbehandlung

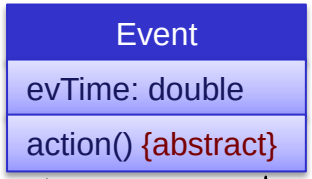
2. Prinzip der Next-Event-Simulation

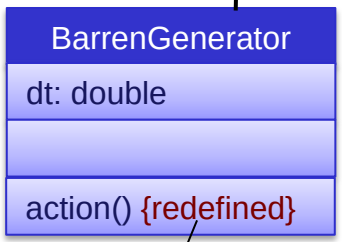
1. Charakterisierung der Next-Event-Simulation
   - Ereignisse, Next-Event-Scheduler
   - Barren-Beispiel (ohne Verwendung aktiver Klassen)
   - Zusammenhang von ereignisbasierter und prozessbasierter Modellbeschreibung

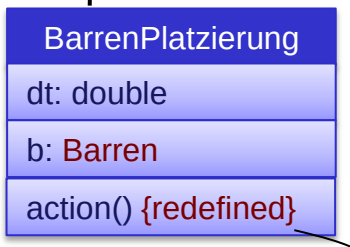
2. Umsetzung des Prinzips in ODEMx
   - Aufbau von ODEMx (erster Blick)
   - Triviales Clock-Beispiel

Erzeugung eines Ankunftsstroms von Barren durch eine zyklische Ereignisfolge



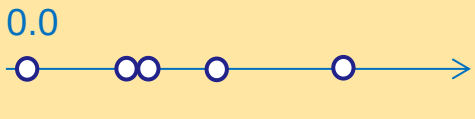






1. bestimme dt zufällig;
2. erzeuge b als neues Objekt von Barren;
3. plane PlatzierungsEreignis für b zum aktuellen Zeitpunkt in EL
4. evTime:= evtime+dt;
5. sortiere THIS-Event entspr. evtime in EL

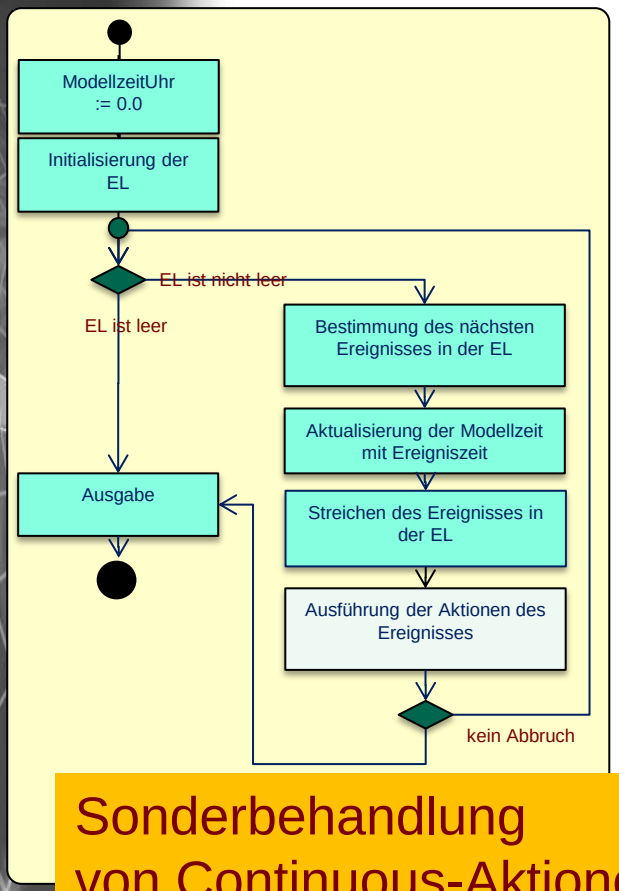
1. Ist Ofen vollbelegt: wird b in WS abgelegt
   b.umg wird 25;
   sonst: b wird im Ofen abgelegt
   b.umg wird ofen.temp
2. plane Temperaturänderung von b zum aktuellen Ereignis



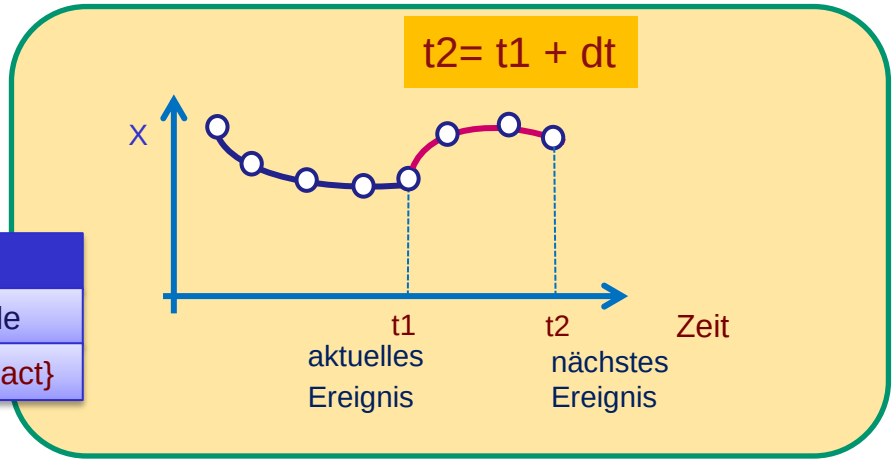
Modellzeitpunkte, zu denen Barren-Objekte ins „Leben gerufen“ werden

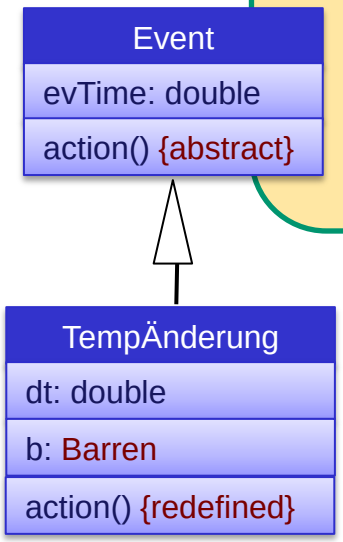
Trivialer Continues-Scheduler

- als Spezialisierung eines Discrete-Event-Schedulers



Sonderbehandlung von Continuous-Aktionen





Barrentemperatur temp (t)

temp´(t)= { umg(t) - temp(t) } / 7

1. bestimme dt verfahrensabhängig
2. bestimme b.temp zum Zeitpunkt evtime+dt
   aus bekannten Wert b.temp zum Zeipunkt evtime und der DGL von b für b.temp zum aktuellen Zeitpunkt evtime in EL mit numerischen Lösungsverfahren
3. evTime:= evtime+dt;
4. sortiere THIS-Event entspr. evtime in EL

2. Prinzip der Next-Event-Simulation

1. Charakterisierung der Next-Event-Simulation
   - Ereignisse, Next-Event-Scheduler
   - Barren-Beispiel
   - Zusammenhang von ereignisbasierter und prozessbasierter Modellbeschreibung

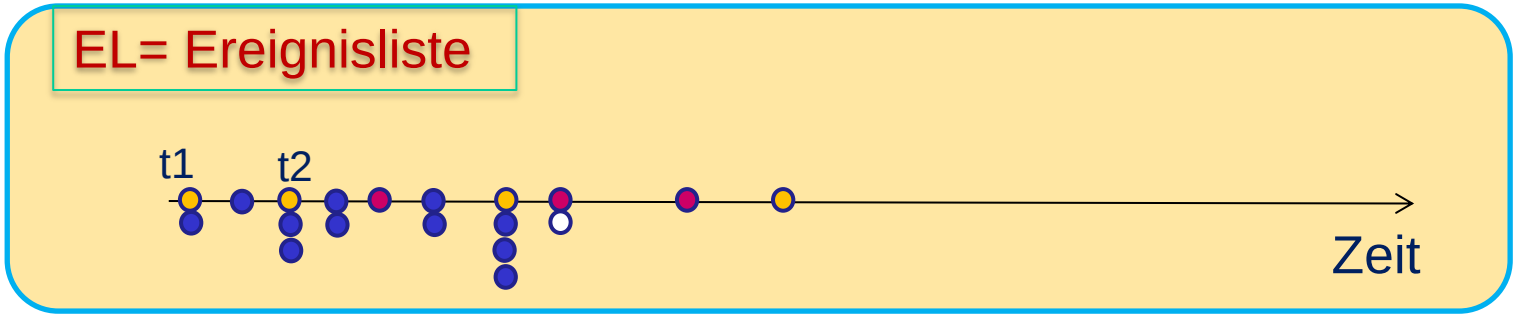
2. Umsetzung des Prinzips in ODEMx
   - Aufbau von ODEMx (erster Blick)
   - Triviales Clock-Beispiel

Zusammenhang

- BarrenErzeugnis-Ereignisse
- BarrenTempKontroll-Ereignisse
- BarrenTempÄnderungs-Ereignisse
- BarrenTerminierungs-Ereignisse

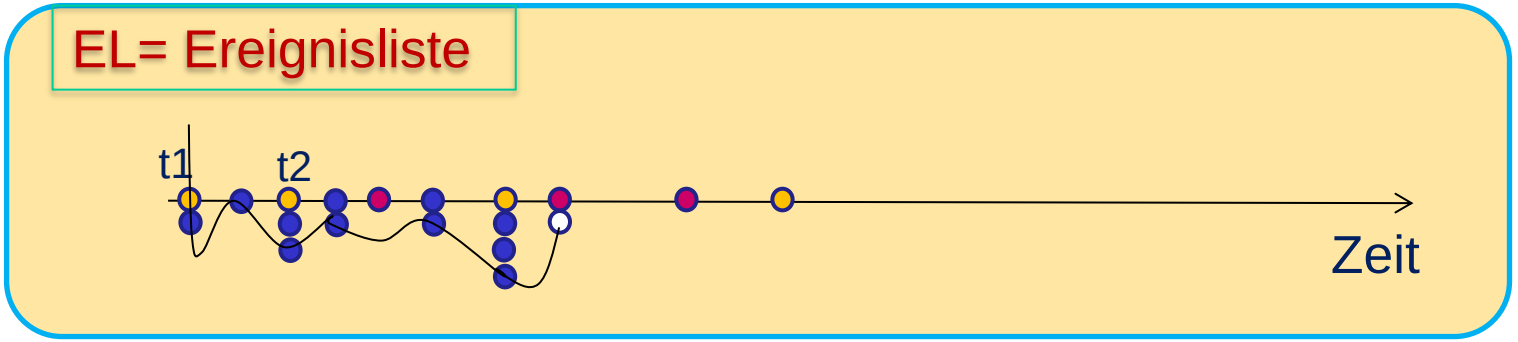
- Ereignis-orientiert:

  Ereignisse werden individuell modelliert



- Prozess-orientiert:

  modelliert werden komplette Ereignisketten



Prozessverlauf: spezielle Kette von Ereignissen je Barren-Objekt

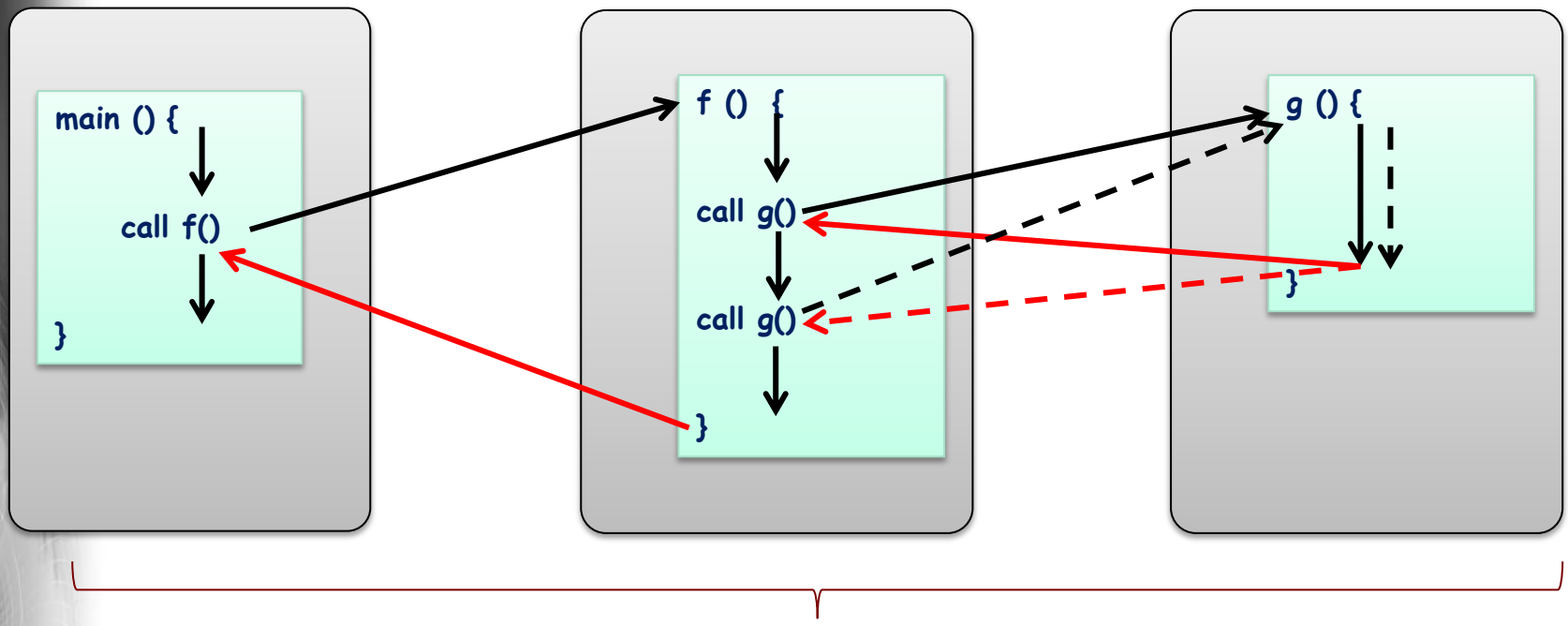
Nachteil der ereignisorientierten Modellierung eines Next-Event-Simulators

- Modellierung verlangt eine globale Sicht auf die Menge von Ereignissen („Vogel-Perspektive“)
- komplexe Modelle lassen sich nicht vernünftig strukturieren (lediglich Hierarchie von Ereignisklassen)

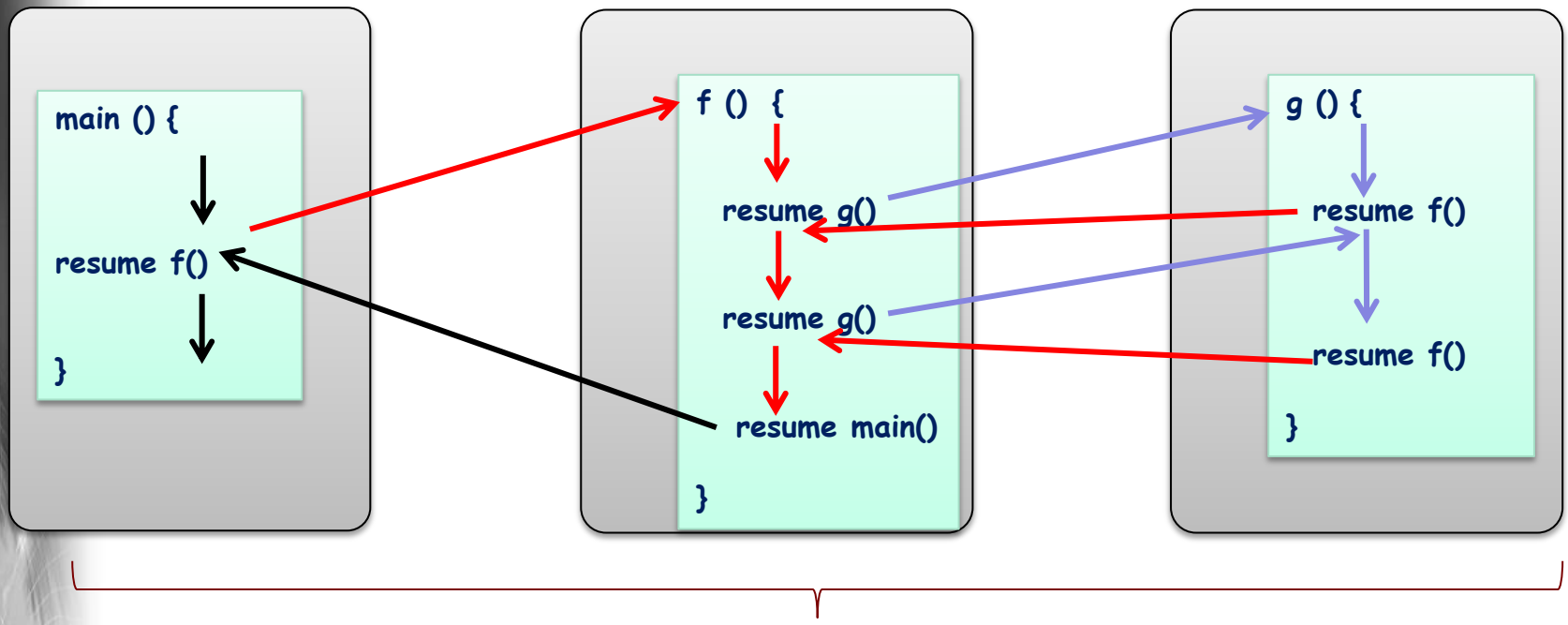
➔ Ausweg prozess-orientierte Modellierung („Frosch-Perspektive“ z.B. wie in ODEMx)

➔ jedoch wird **Koroutinenkonzept** für Implementierung benötigt

Routinen (Prozeduren/Funktionen)



Koroutinen



Zusätzlich muss auch der jeweilige Aufruf-Stack von resume wieder hergestellt werden. D.h. bei Absprung aus einer Koroutine muss dieser gespeichert werden.

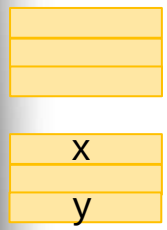
2. Prinzip der Next-Event-Simulation

1. Charakterisierung der Next-Event-Simulation
   - Ereignisse, Next-Event-Scheduler
   - Barren-Beispiel
   - Zusammenhang von ereignisbasierter und prozessbasierter Modellbeschreibung

2. Umsetzung des Prinzips in ODEMx
   - Aufbau von ODEMx (erster Blick)
   - Triviales Clock-Beispiel

Koroutinen als Member-Funktion

```
class X {…}
class Y {…}
```



globale Daten

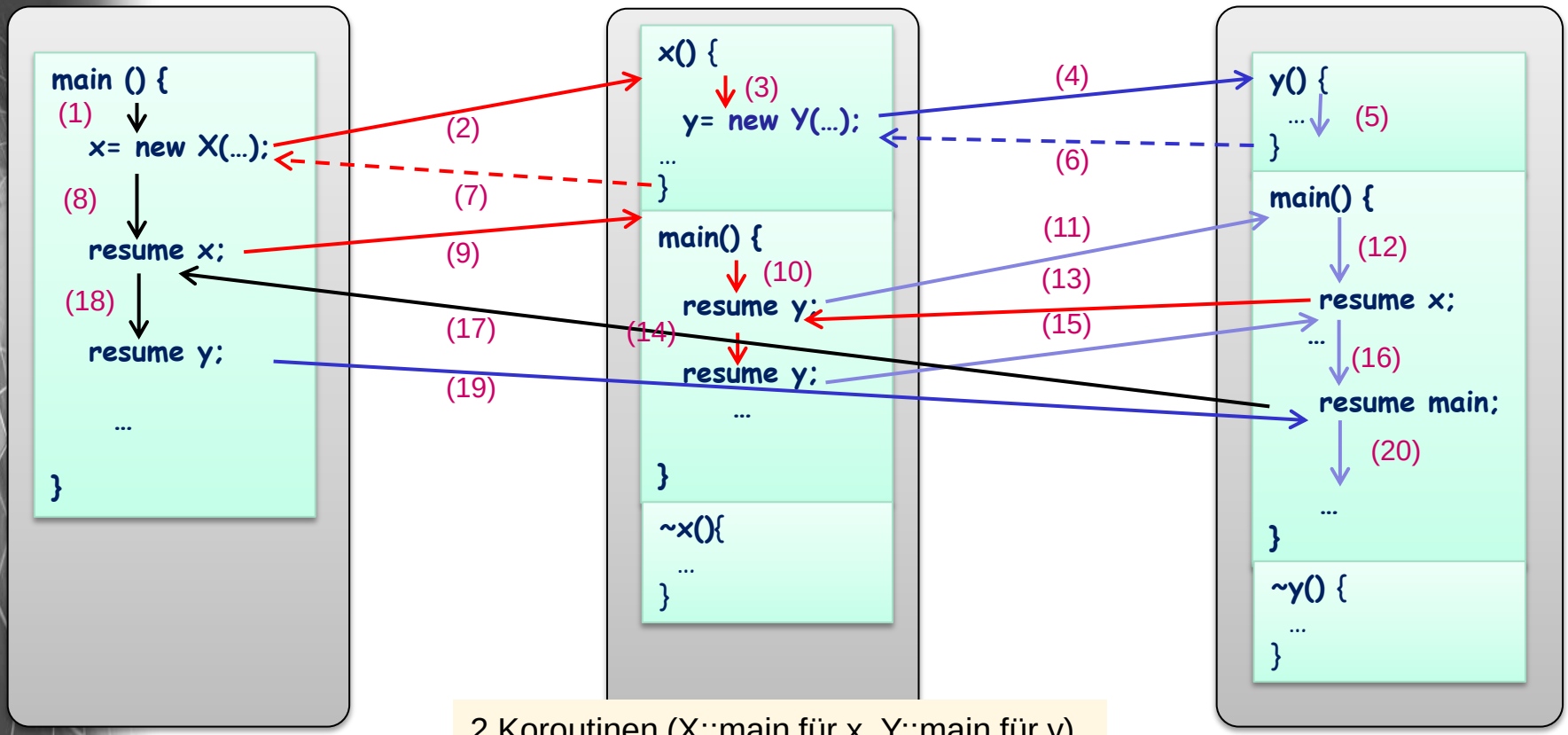
lokale main-Daten

x

lokale x-main-Daten

y

lokale y-main-Daten



2 Koroutinen (X::main für x, Y::main für y) plus ausgezeichnete Koroutine ::main

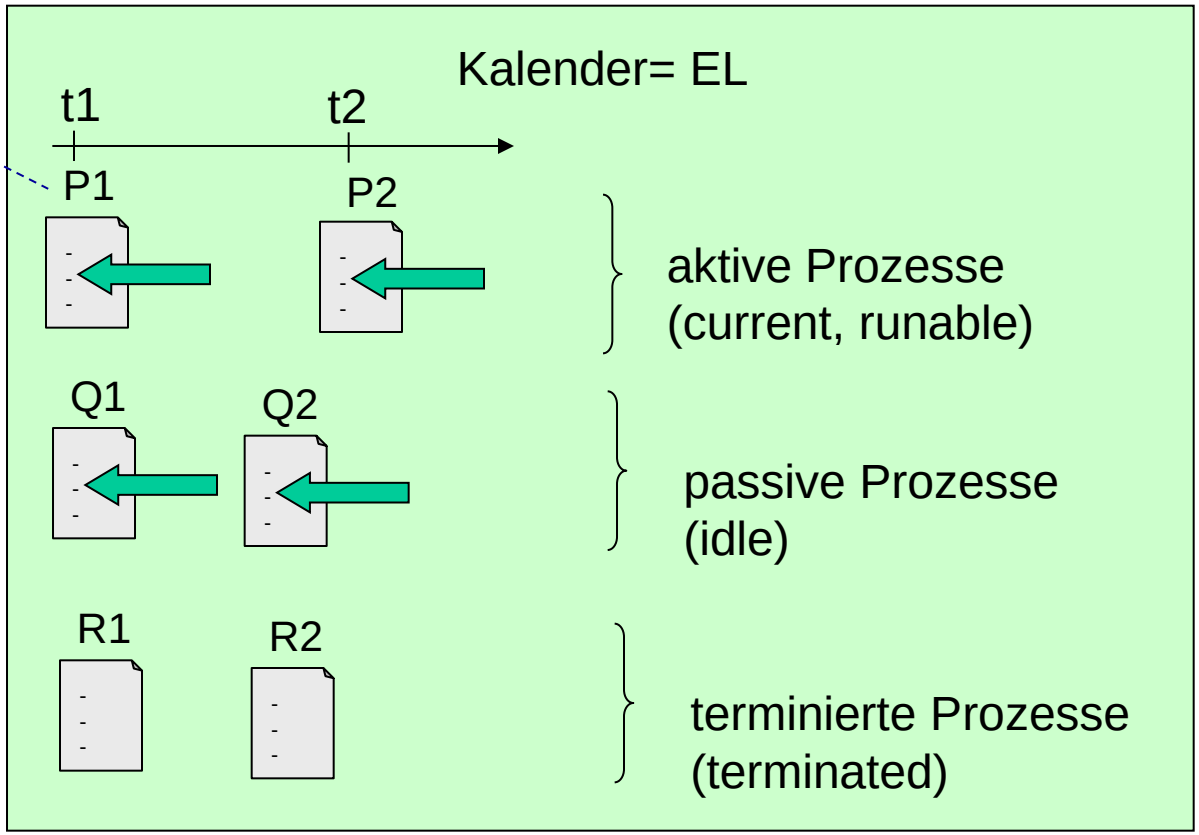
Elementare Prozessverwaltung in ODEMx

Klasse Process
verfügt
über eine virtuelle
Memberfunktion main
(Lebenslauf des Prozesses)

```
int main ( ...) {
...
}
```

C++ main program

simulation context (DefaultSimulation-Objekt)



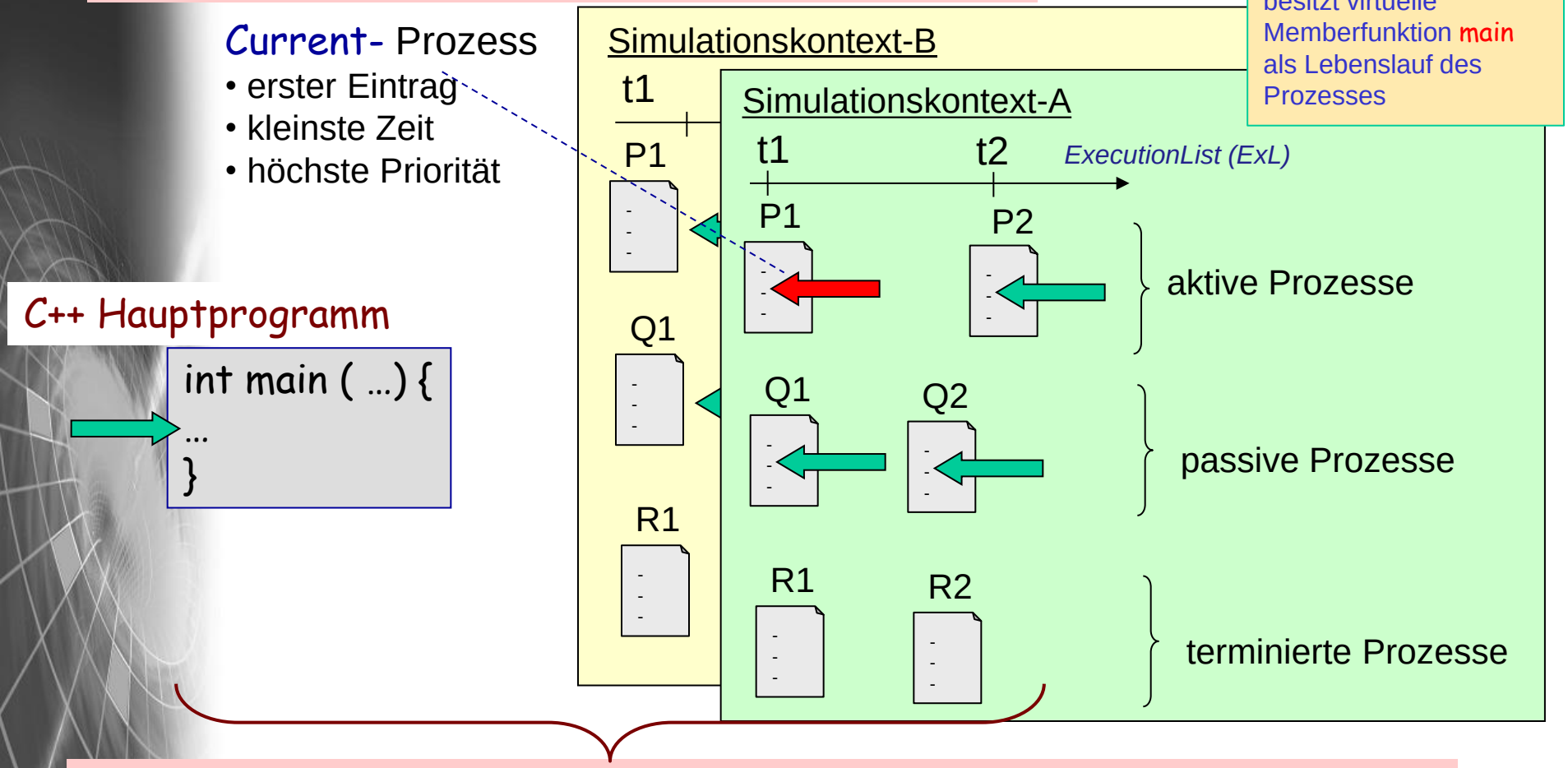
Ensemble von main()-Funktionen aller existenten Prozess-Objekte wird zusammen mit eigentlichem C++ Hauptprogramm als Ensemble von Coroutinen quasiparallel ausgeführt

Grundidee einer hierarchischen Prozessverwaltung

*Zu einem Zeitpunkt kann immer **nur ein Kontext** und in dem **nur ein Prozess** aktiv sein (current)*

Klasse Process
besitzt virtuelle Memberfunktion main als Lebenslauf des Prozesses

Current- Prozess
- erster Eintrag
- kleinste Zeit
- höchste Priorität



C++ Hauptprogramm

```
int main ( ...) {
...
}
```

*Hauptprogramm (main-Fkt) und Prozesse (lokale main-Fkt) aller Simulationskontexte bilden ein **hierarchisches Koroutinensystem** auf einer Ein-Prozessor-Maschine*