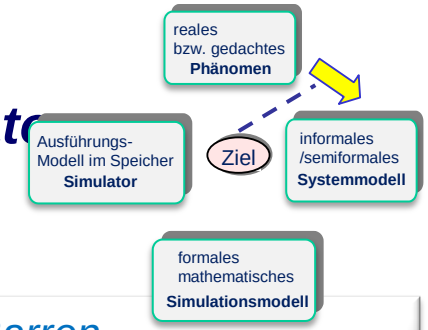
Kurs OMSI
im WiSe 2013/14

Objektorientierte Simulation mit ODEMx

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

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

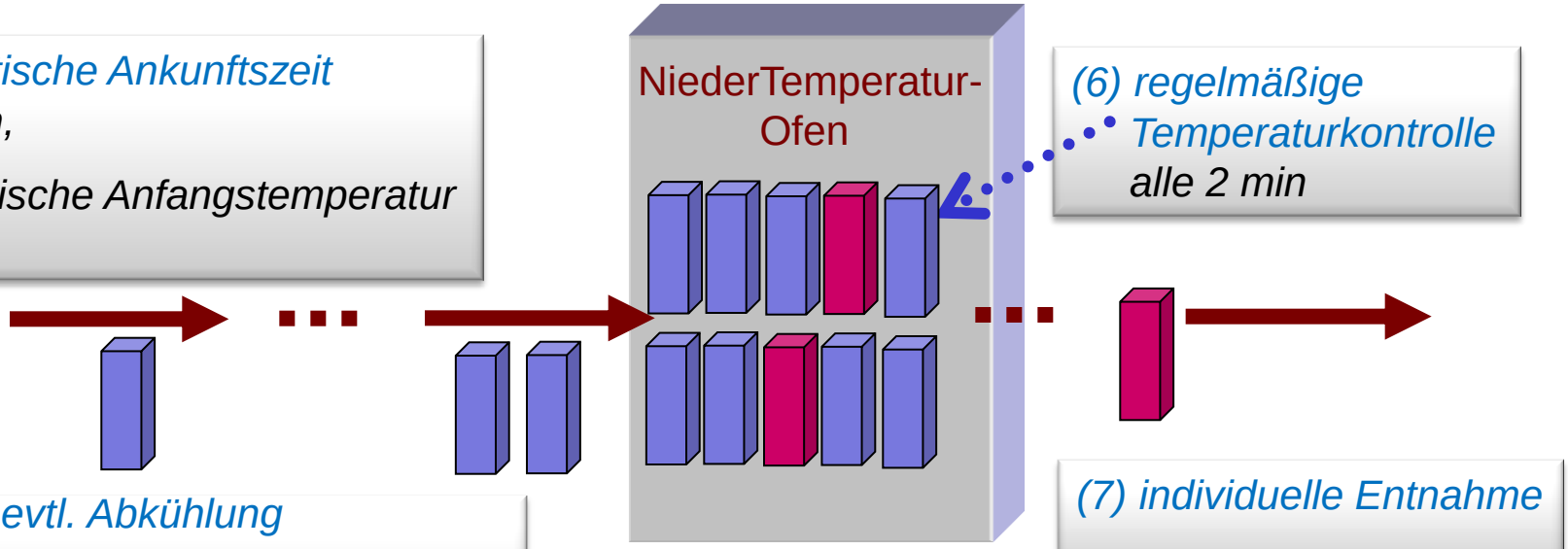
Beispiel für die Entwicklung eines Simulato



(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

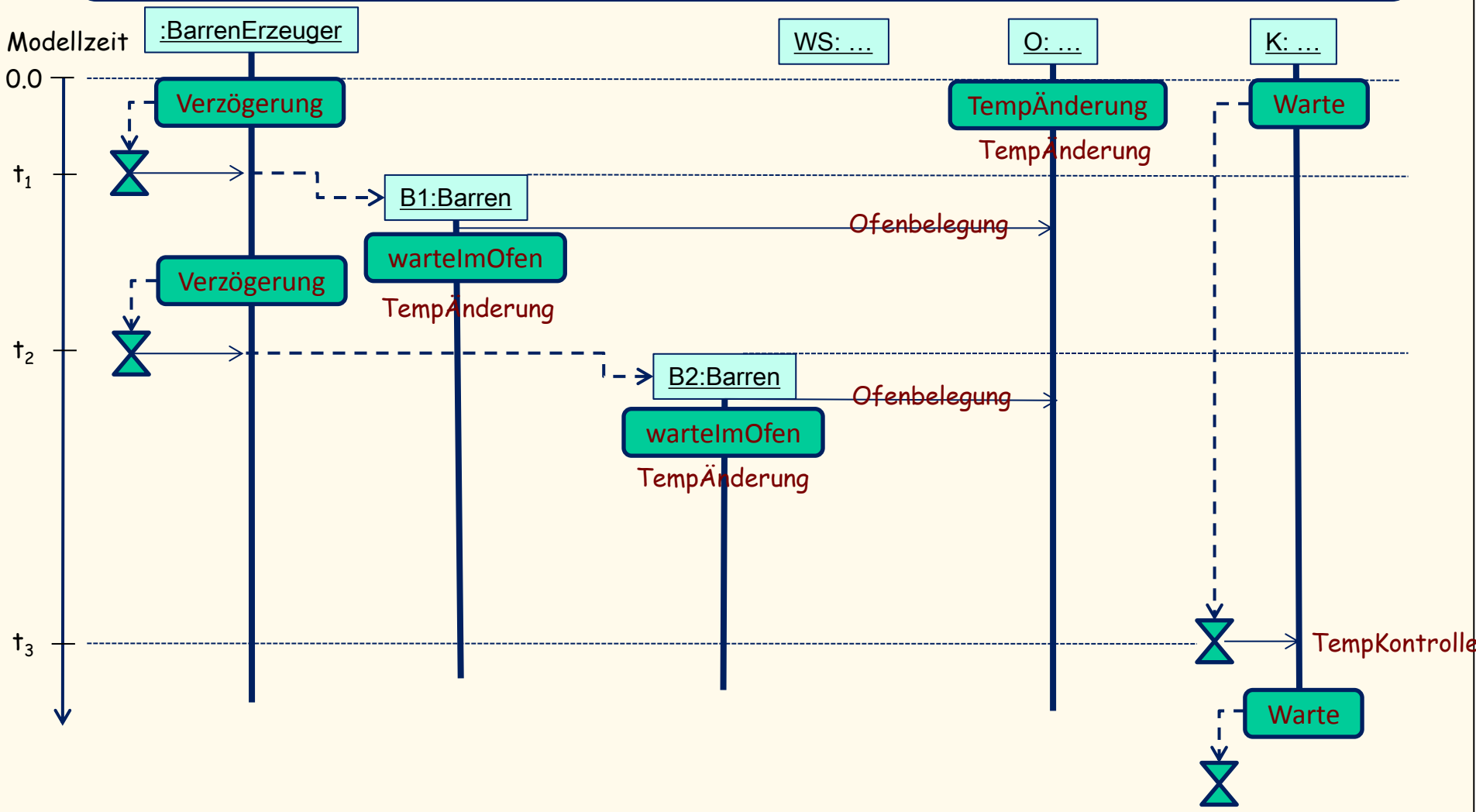
2. Prinzip der Next-Event-Simulation

1. Allg. Charakterisierung der Next-Event-Simulation
   - Ereignisse, Next-Event-Scheduler
   - Barren-Beispiel
   - Zusammenhang von ereignisbasierter und prozessbasierter Modellbeschreibung
   - Koroutinen: Basis der Implementierung

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

Laufzeitverwaltung *auf einer Ein-Prozessormaschine*

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



Basis für sequentielle Abwicklung paralleler Vorgänge *(Computersimulation)*

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

- führt nur 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)

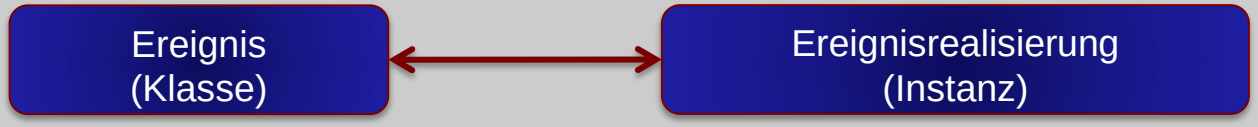
  oder

  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

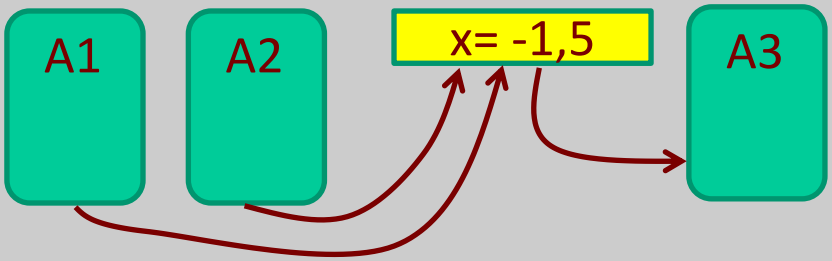
Bedienungssysteme sind klassische Vertreter von Systemen,
deren Verhalten sich durch sequentielle Ereignissimulationen beschreiben lassen

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)

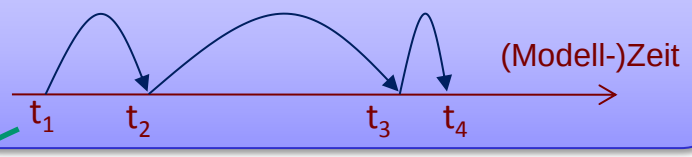
- Ereigniskalender
  a) Umgang mit Gleichzeitigkeit von Ereignissen

  b) Scheduling
  Bestimmung des nächsten Ereignisses im Kalender

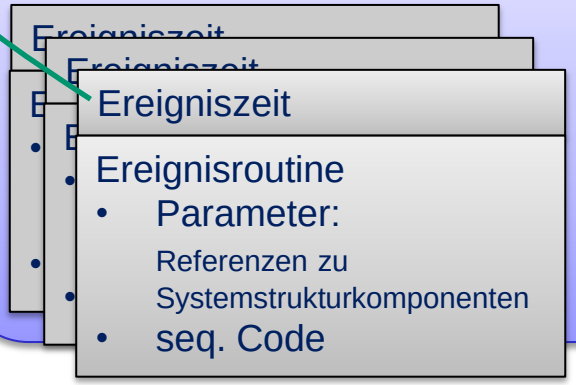
  c) Zeitfortschritt
  sobald zum aktuellen Zeitpunkt alle Ereignise ausgeführt sind, wird Uhr weitergestellt

  (aktuelle Modellzeit:= Ereigniszeit des nächsten Ereignisses)

Zustandsänderungen finden immer nur zu diskreten Zeitpunkten statt



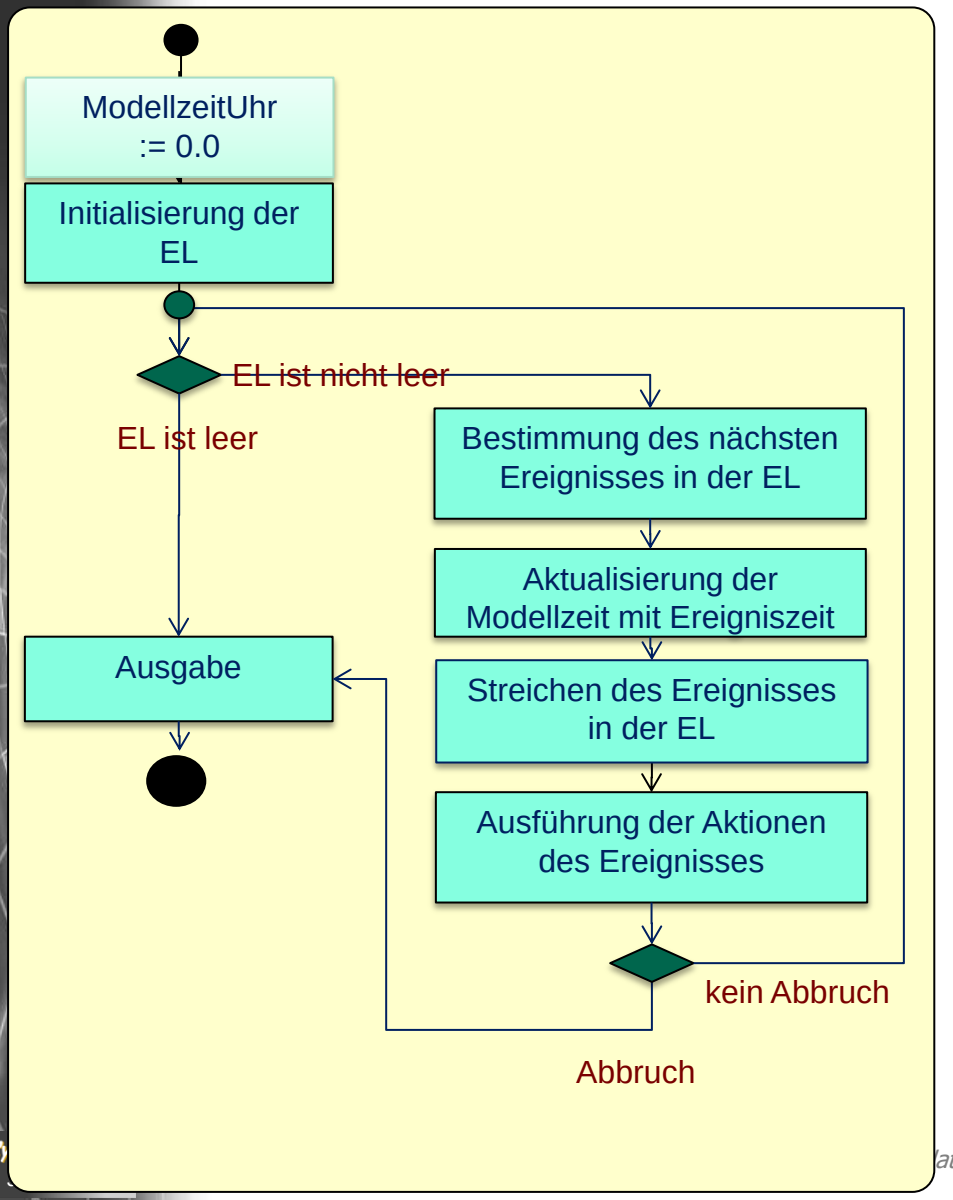
Ereignis-Chakteristik



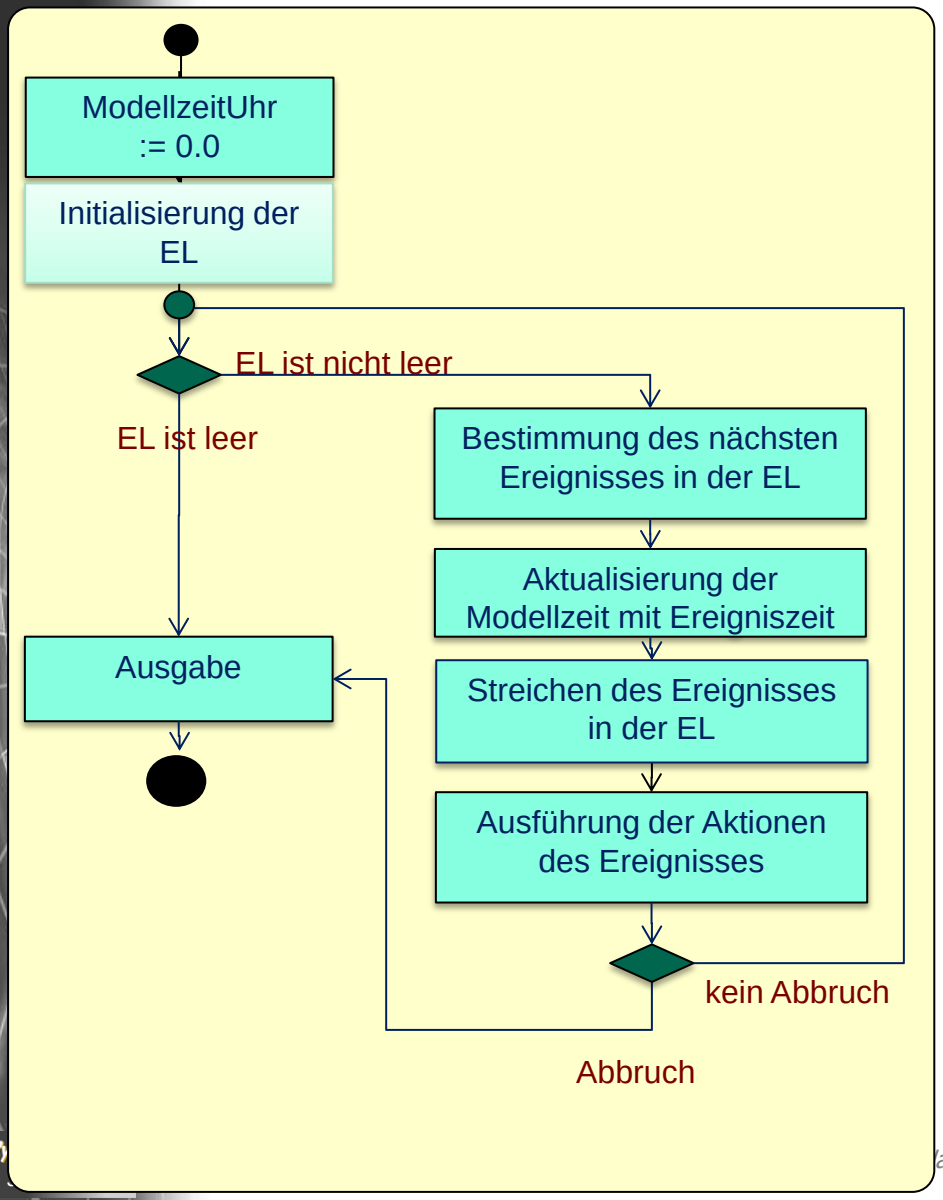
(Ereignis-) Scheduler

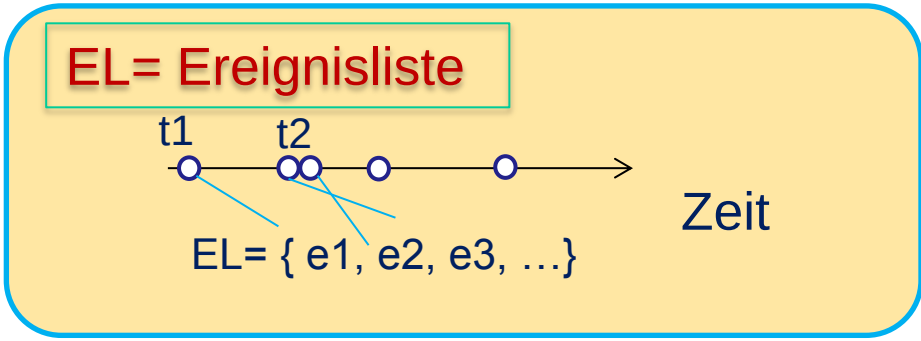
Aufgaben
- Zeitfortschritts-realisierung
- Realisierung des aktuellen Ereignisses

Trivialer Discrete-Event-Scheduler

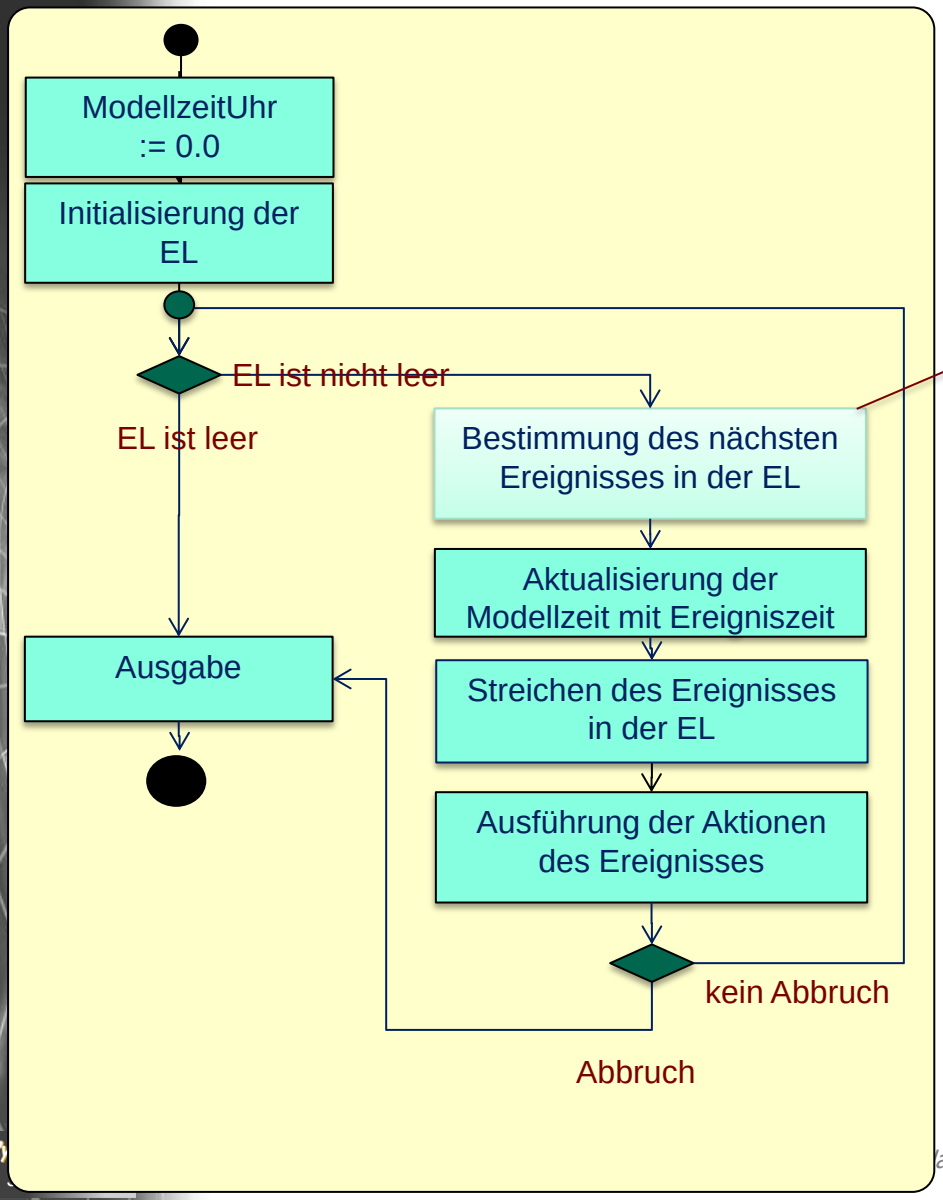


Trivialer Discrete-Event-Scheduler

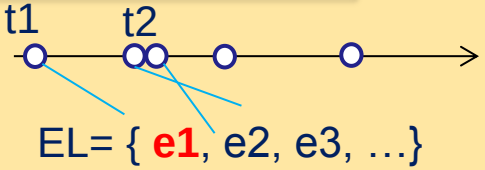




Trivialer Discrete-Event-Scheduler

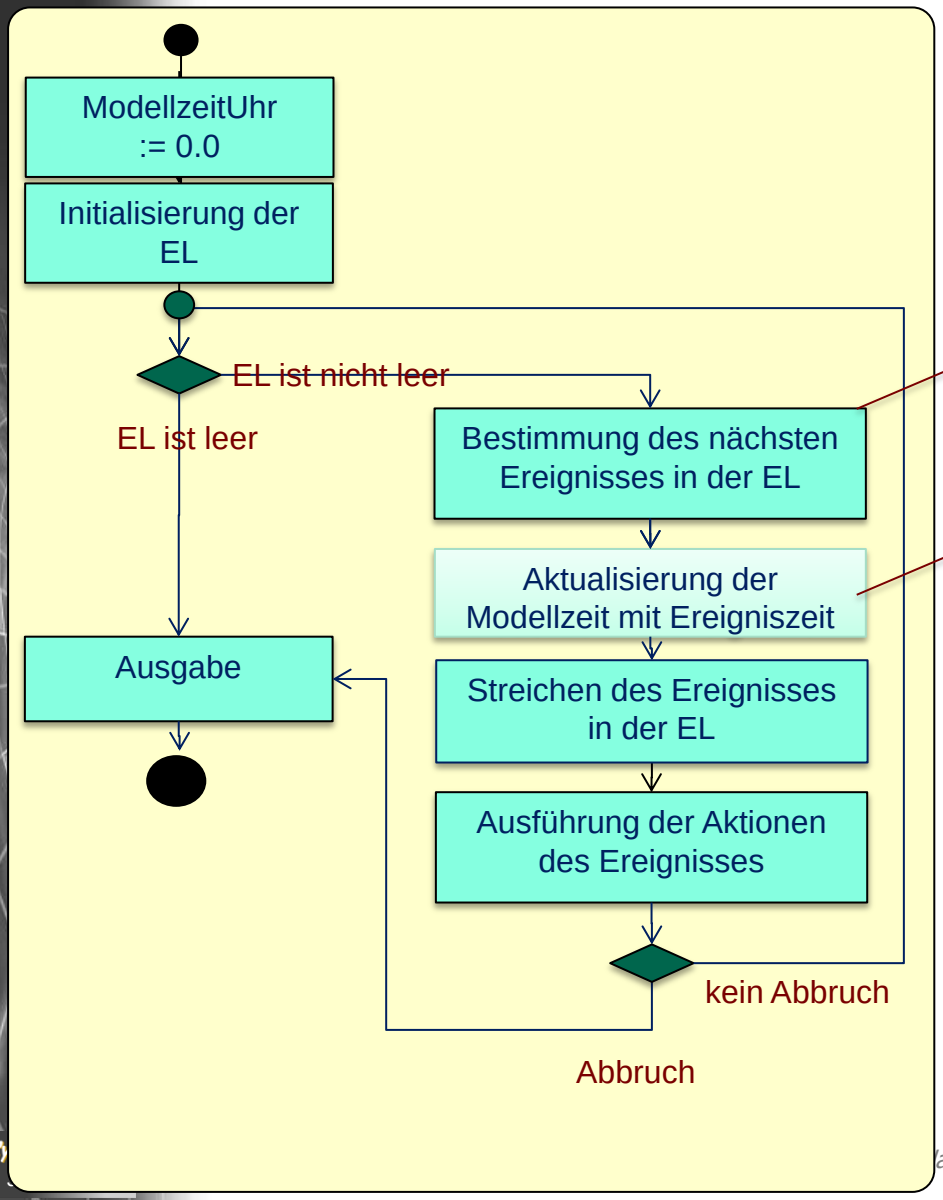


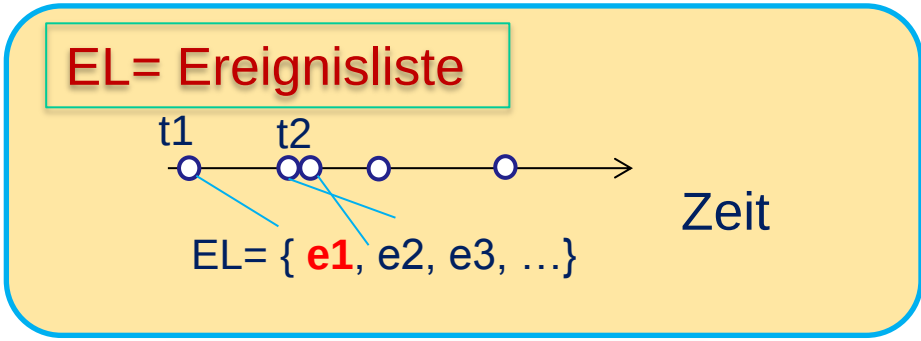
EL= Ereignisliste



EL ist chronologisch sortiert

Trivialer Discrete-Event-Scheduler

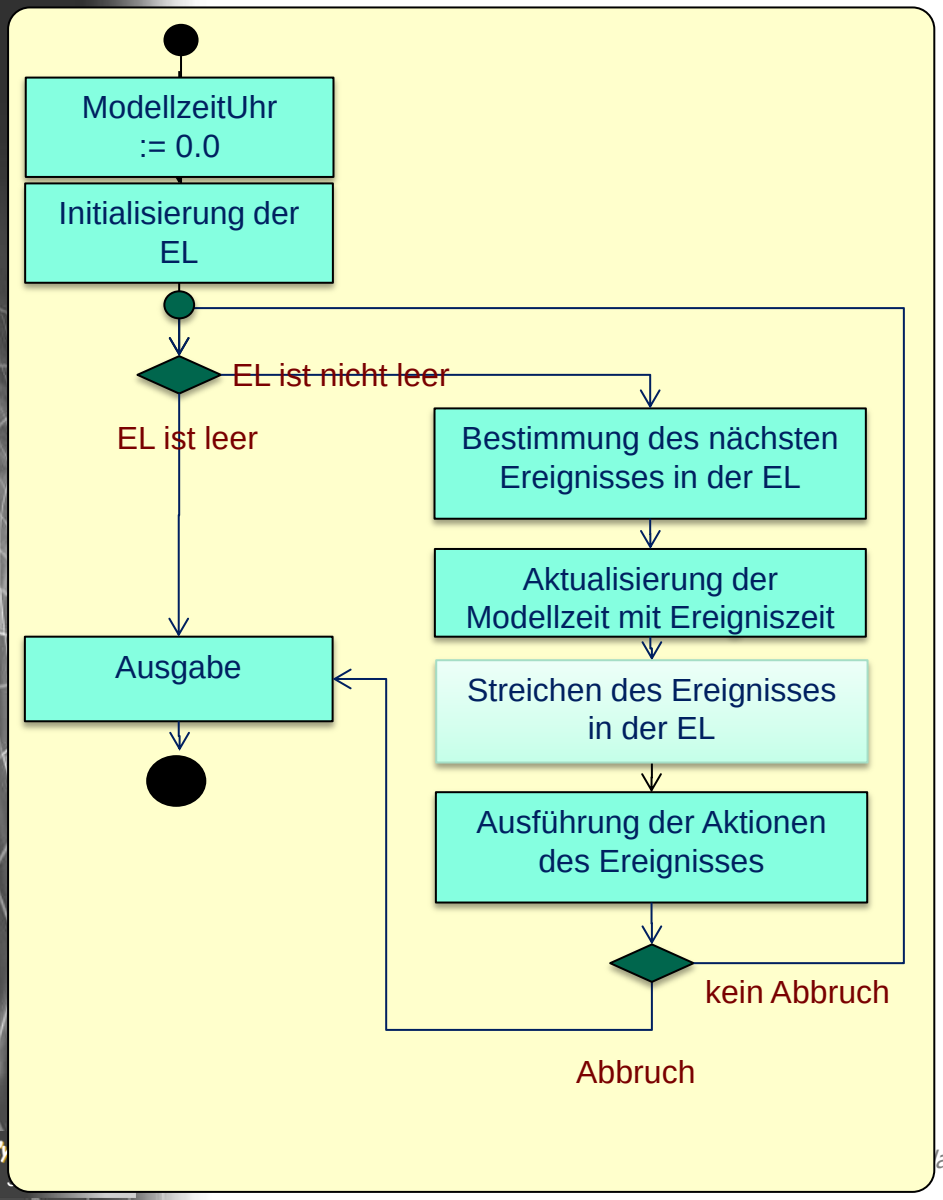


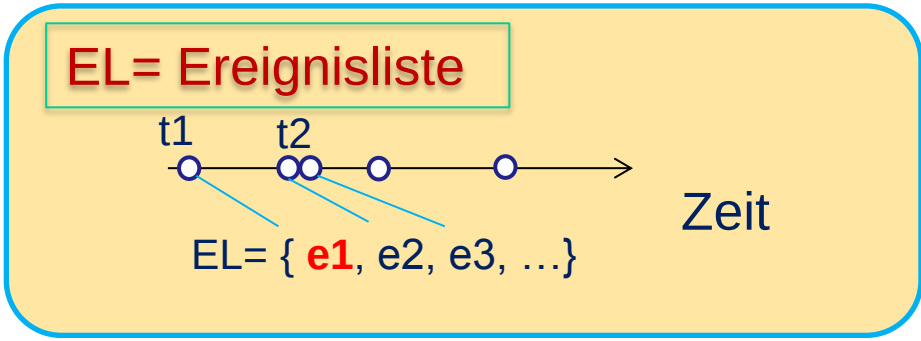


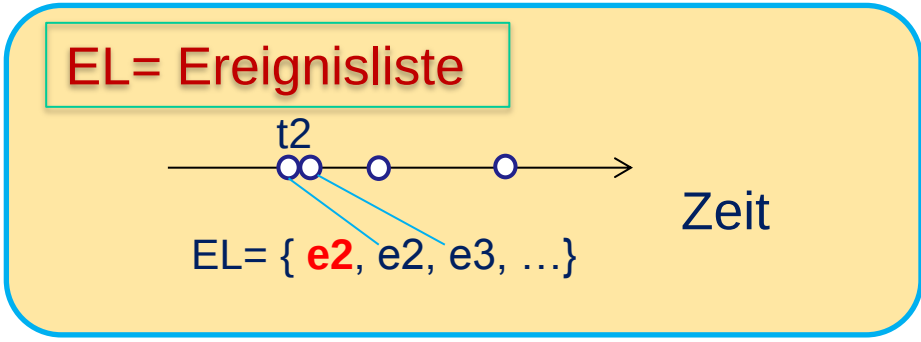
EL ist chronologisch sortiert

evtl. Zeitsprung (zeitdiskret)

Trivialer Discrete-Event-Scheduler

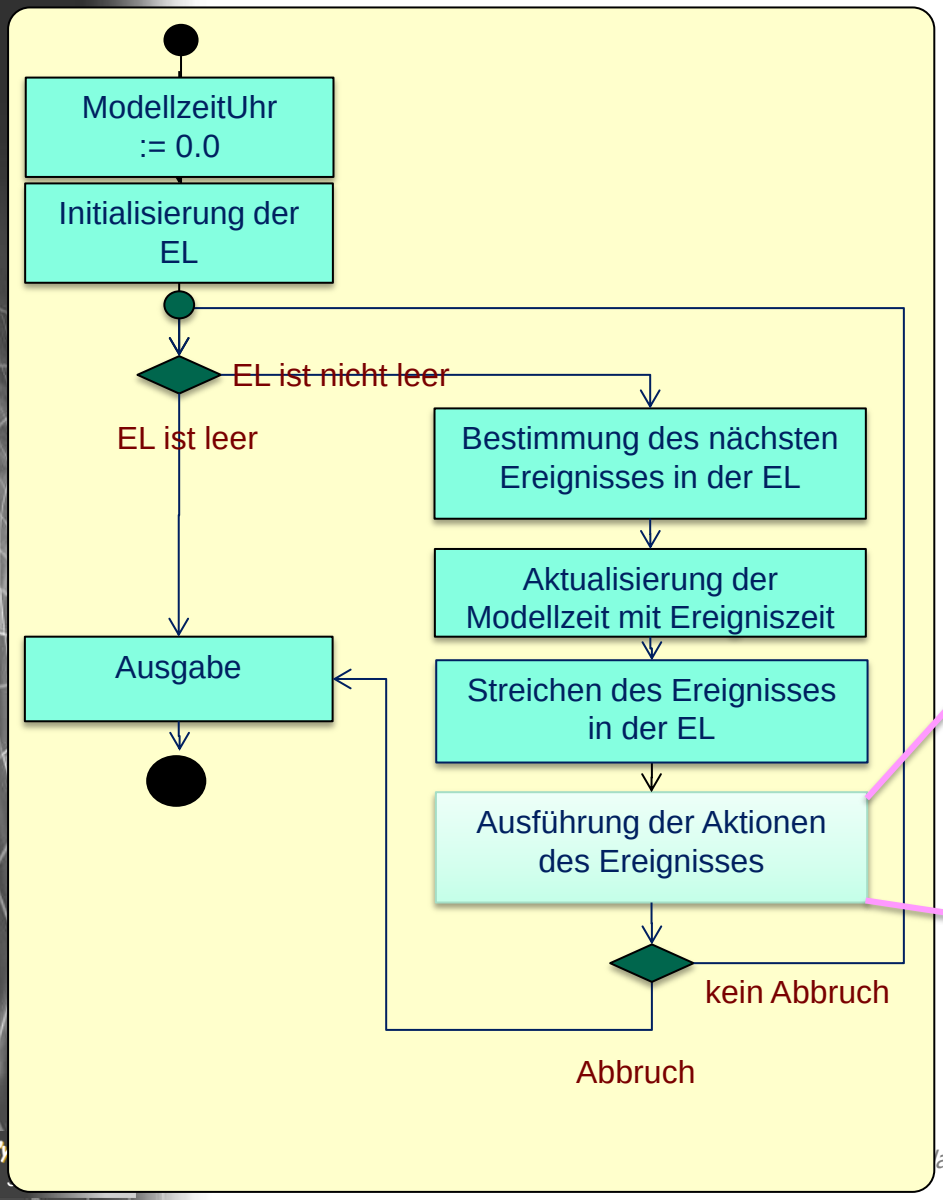


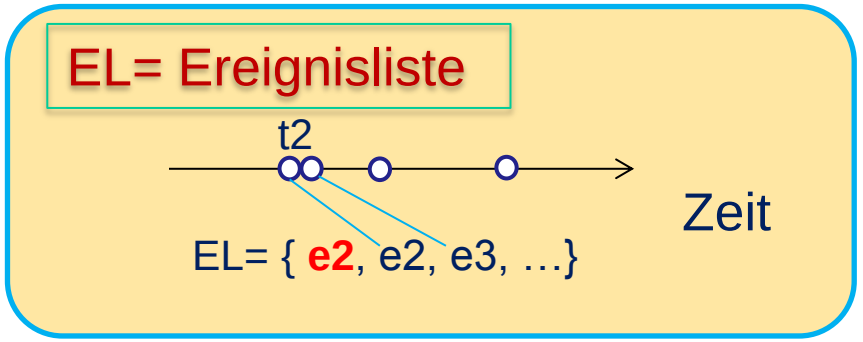




currentEvent=**e1**

Trivialer Discrete-Event-Scheduler





currentEvent=**e1**

Zustandsänderungen

Klassifizierung von Ereignisklassen, um partielle Zustandsänderungen strukturell zu unterstützen

Planung neuer Ereignisse

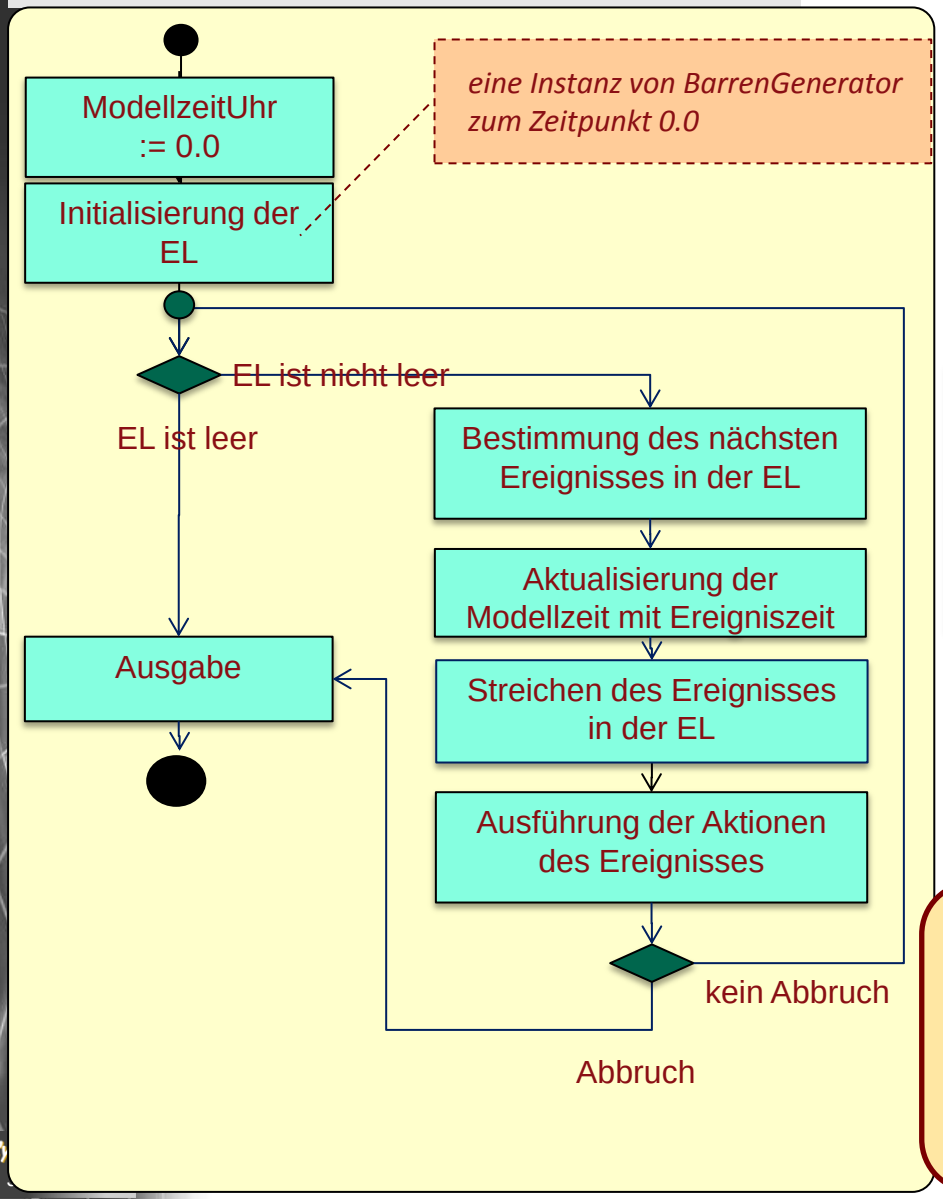
Problem: Gleichzeitigkeit/
Zustandsereignis

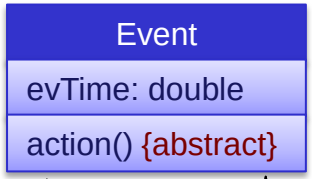
2. Prinzip der Next-Event-Simulation

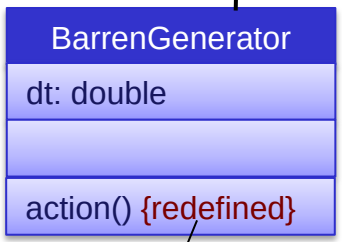
1. Charakterisierung der Next-Event-Simulation
   - Ereignisse, Next-Event-Scheduler
   - Barren-Beispiel
   - Zusammenhang von ereignisbasierter und prozessbasierter Modellbeschreibung

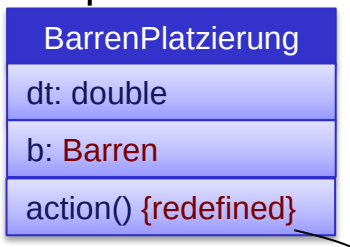
2. Umsetzung des Prinzips in ODEMx
   - Aufbau von ODEMx (erster Blick)
   - Triviales Clock-Beispiel

Erzeugung eines Ankunftsstroms von Barren durch eine zyklische Ereignisfolge



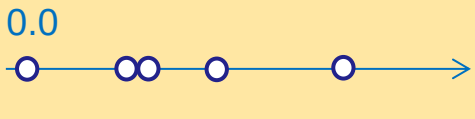






1. bestimme dt zufällig;
2. erzeuge b als neues Objekt von Barren;
3. plane EinsortierungsEreignis für b zum aktuellen Zeitpunkt in EL
4. evTime:= evtime+dt;
5. sortiere THIS-Event entspr. evtime in EL

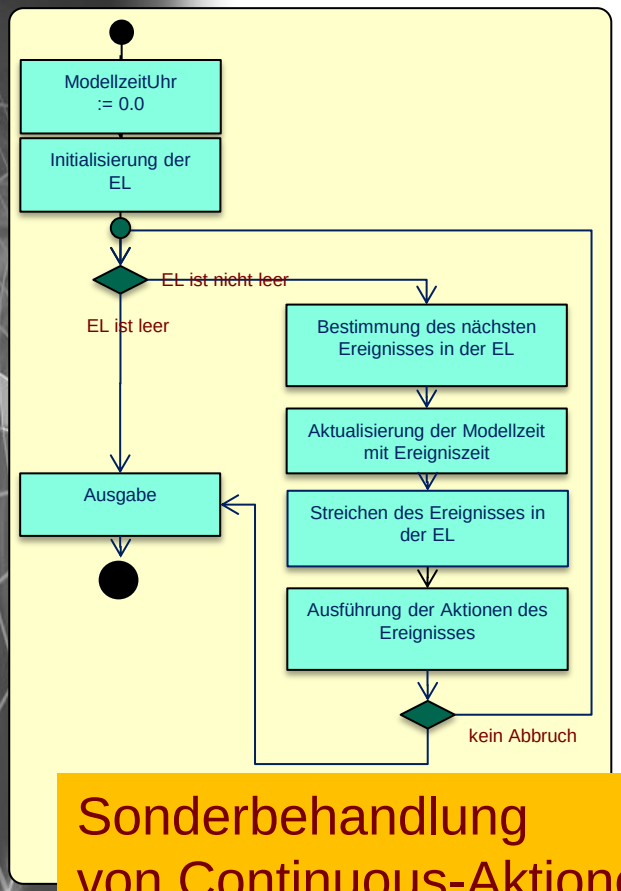
1. Ist Ofen vollbelegt: wird b in WS abgelegt
   b.umg wird 25;
   sonst: b wird im Ofen abgelegt
   b.umg wird ofen.temp
2. plane Temperaturänderung von b zum aktuellen Ereignis



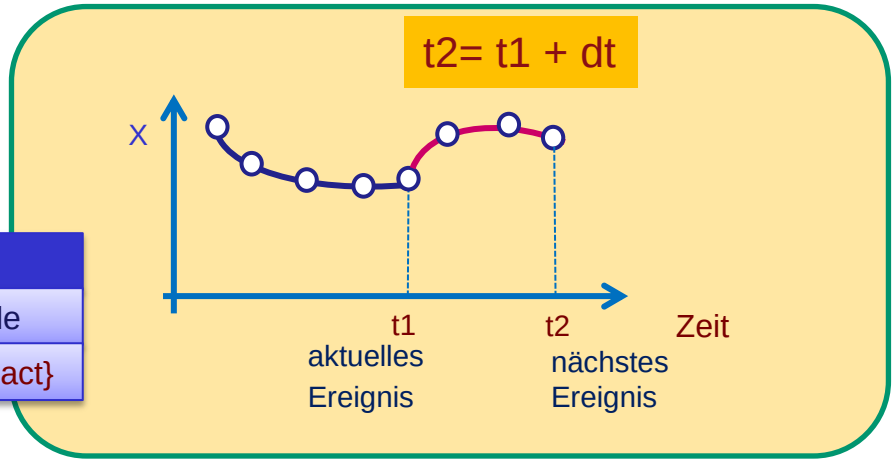
Modellzeitpunkte, zu denen Barren-Objekte ins „Leben gerufen“ werden

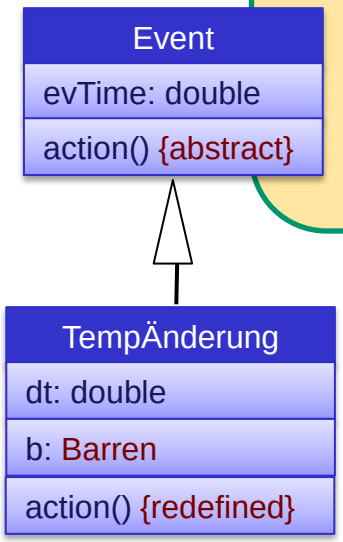
Trivialer Continues-Scheduler

- 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

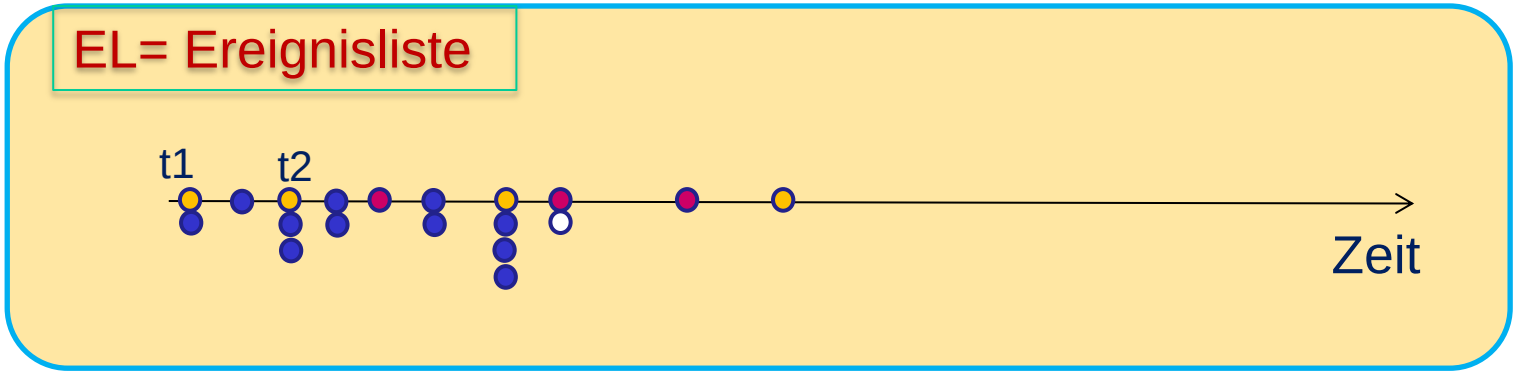
1. Charakterisierung der Next-Event-Simulation
   - Ereignisse, Next-Event-Scheduler
   - Barren-Beispiel
   - Zusammenhang von ereignisbasierter und prozessbasierter Modellbeschreibung

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

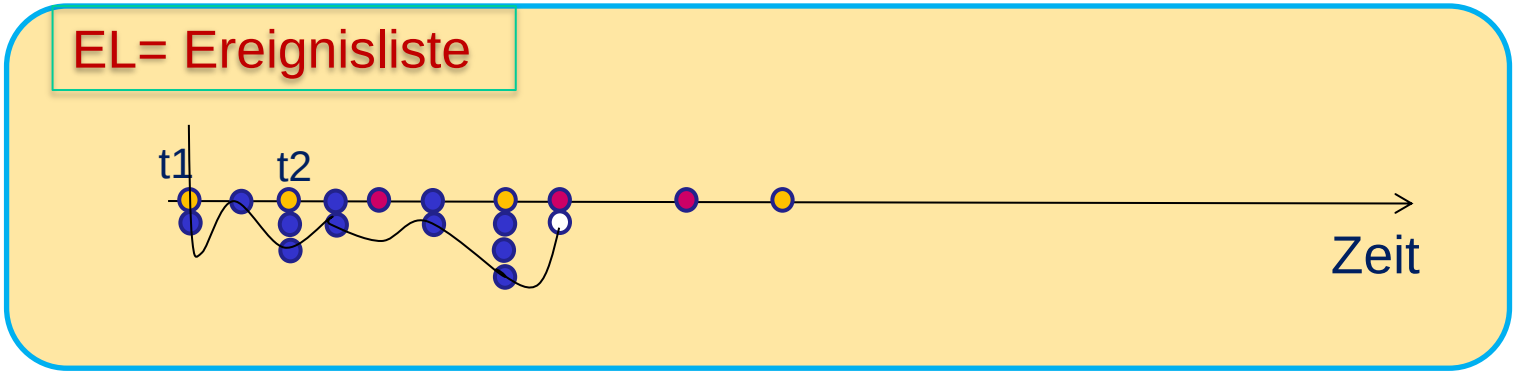
Zusammenhang

- 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)

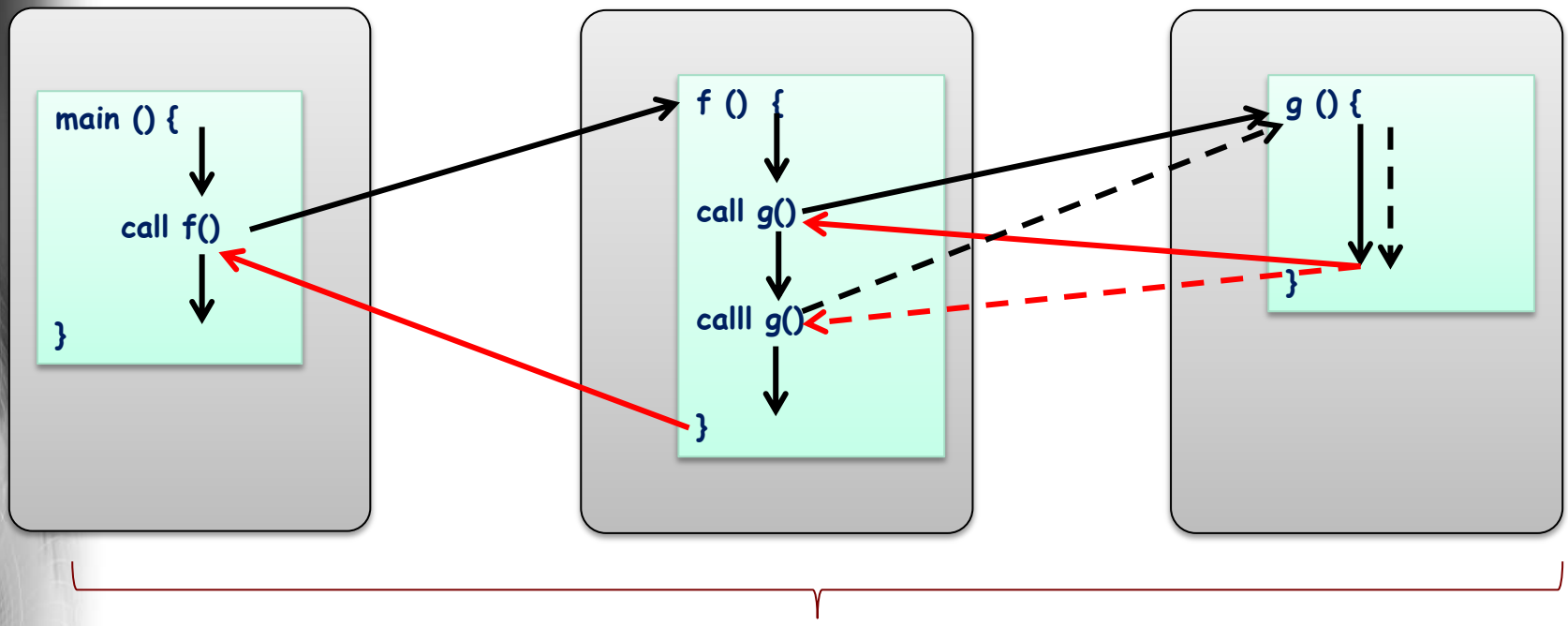
➔ Jedoch wird Koroutinenkonzept benötigt

2. Prinzip der Next-Event-Simulation

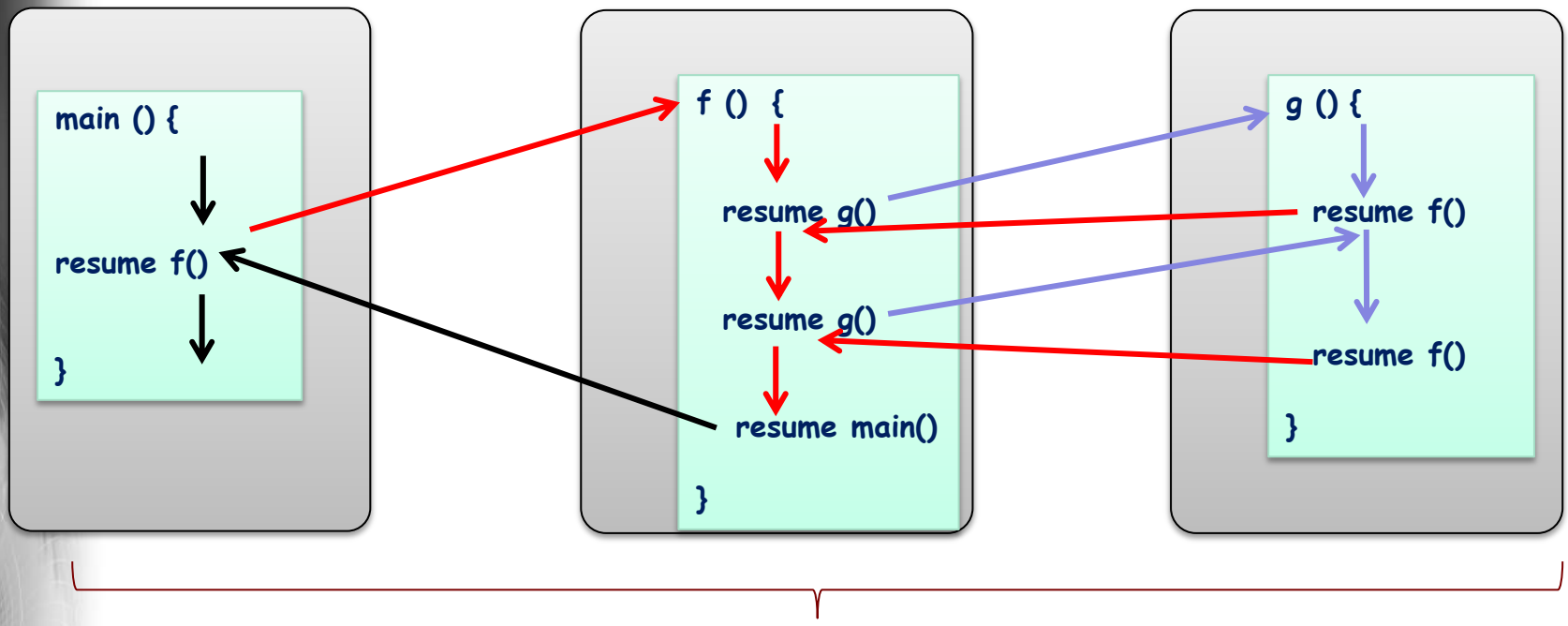
1. Charakterisierung der Next-Event-Simulation
   - Ereignisse, Next-Event-Scheduler
   - Barren-Beispiel
   - Zusammenhang von ereignisbasierter und prozessbasierter Modellbeschreibung

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

Routinen (Prozeduren/Funktionen)



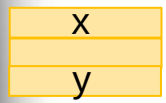
Koroutinen



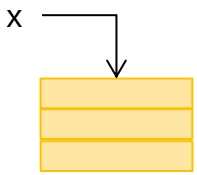
Programm-Code je Koroutinen-Körper
(Befehle) im Speicher

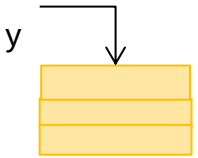
Koroutinen als Member-Funktion

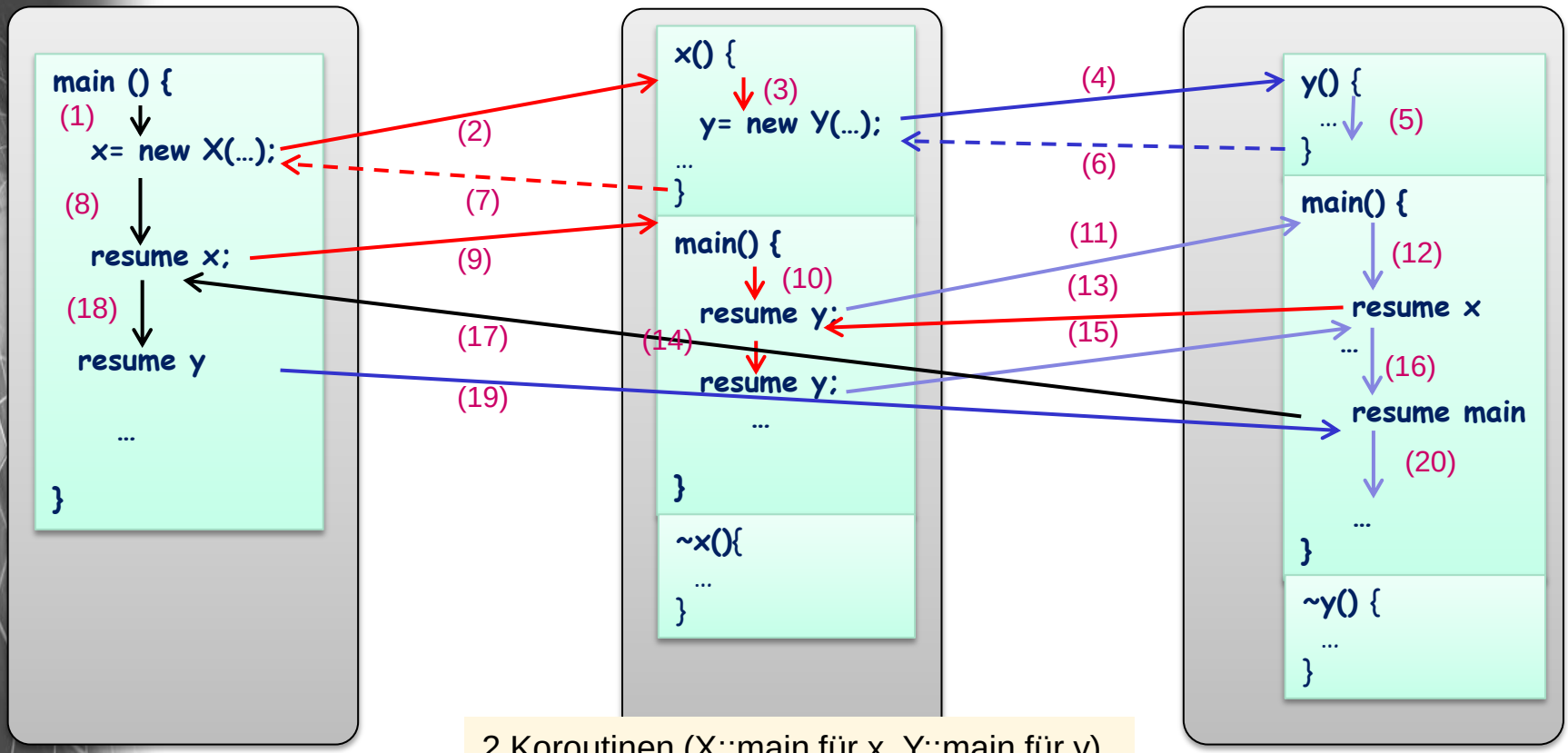
class X {…}
class Y {…}



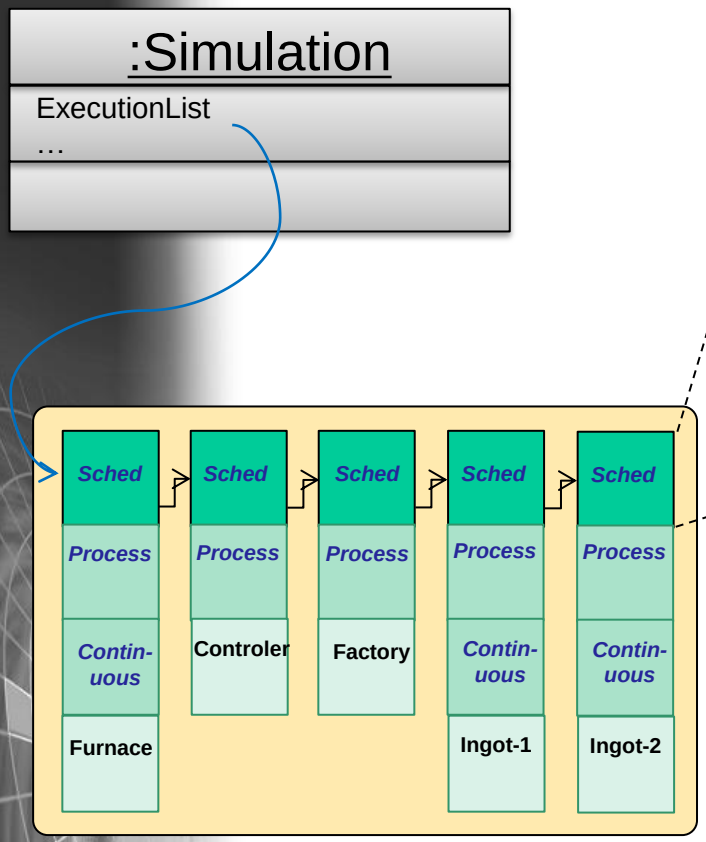
globale
Daten (Zeiger)







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



Sched

#execute()=0
+getExecutionTime() const =0
+getPriority() const =0
+getSchedType() const
+getTime() const
+isScheduled() const
+Sched (Simulation &sim, …)
+setExecutionTime(SimTime time)=0
+setPriority(Priority newPriority)=0
+~Sched ()

abstrakte passive Klasse

Abteilung der **Attribute**
leer: nicht genannt/vorhanden

Abteilung der **Operationen**

Sichtbarkeit: +,-,#,~

Konstruktor sorgt für Zuordnung zu einem Simulationskontext

Process

- ProcessState processState_;
- Priority priority_;
- SimTime executionTime_;

#virtual int main() = 0;
+setExecutionTime(SimTime time);
#void execute();
+void holdFor (SimTime *t*)

abstrakte aktive Klasse

Grundzustände:
CREATED, CURRENT,
RUNABLE, IDLE, TERMINATED

Priorität:
Gleichzeitigkeitskonfliktbehebung bei der
Sequentialisierung von Ereignissen

Ereigniszeit:
Basis der Sortierung des Terminkalenders (ExecutionList)
ModelTime-Typ ist variabel

Abstrakter Lebenslauf: main
Redefinition von execution
(Fortsetzung im Lebenslauf)

Zeitverbrauch (Terminkalender)

Konkrete aktive Klasse

Konkreter Lebenslauf

Factory

-SimTime dt
-Ingot* ing

```
#int main() {
      holdFor (dt);
      ing= new Ingot (…);
}
```

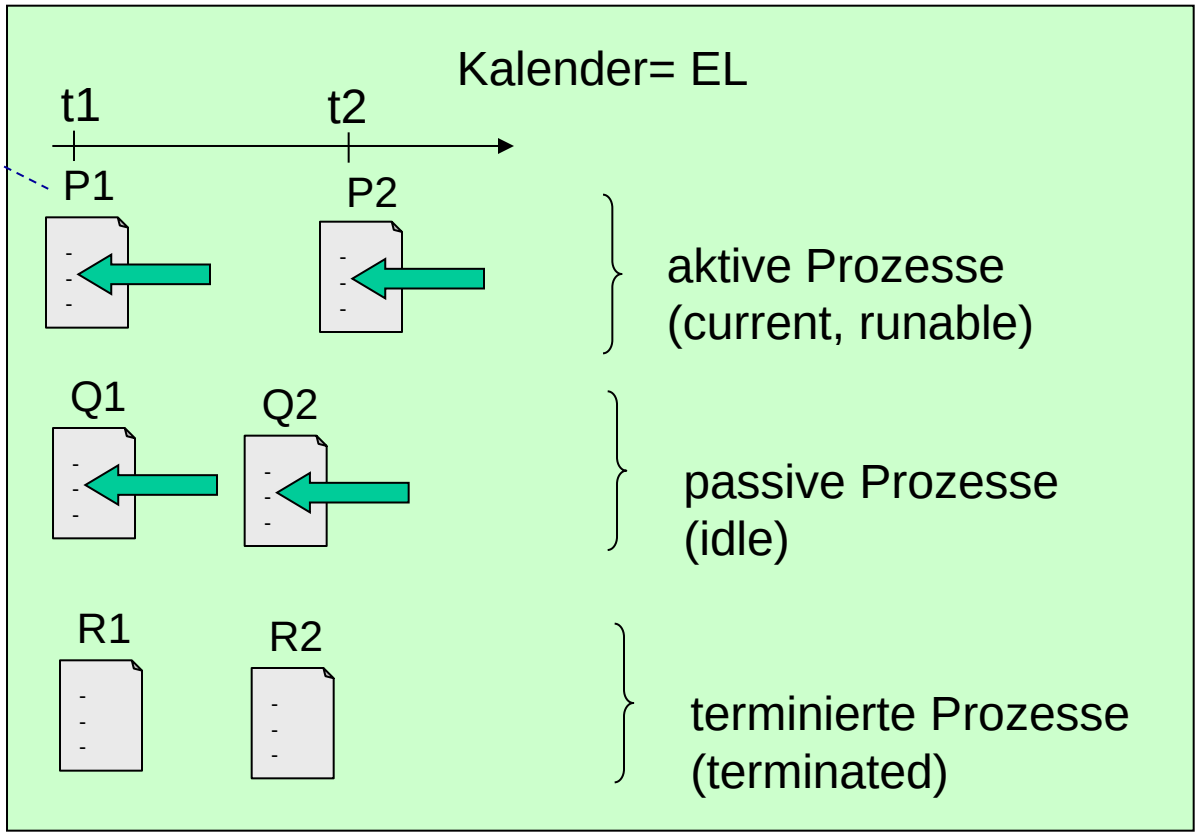
Prozessverwaltung in ODEMx

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

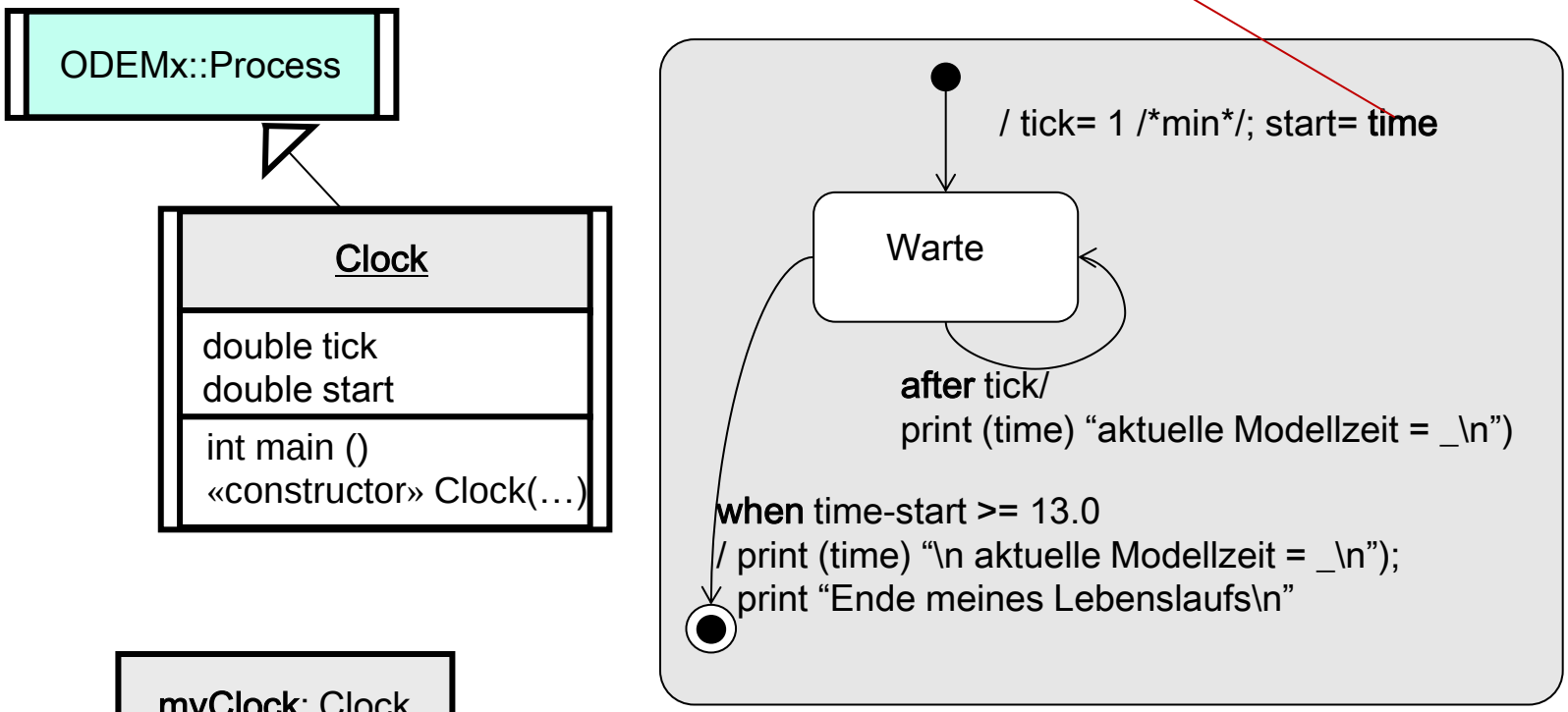
2. Prinzip der Next-Event-Simulation

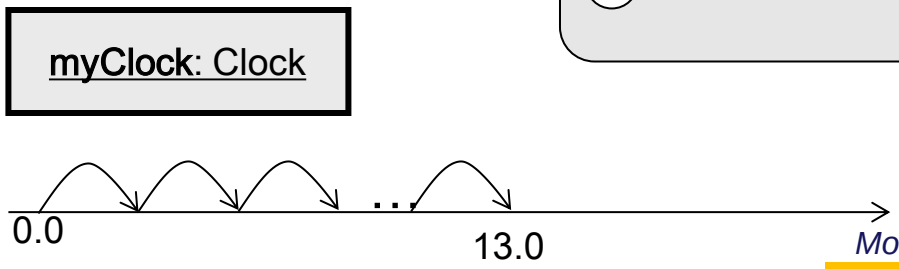
1. Charakterisierung der Next-Event-Simulation
   - Ereignisse, Next-Event-Scheduler
   - Barren-Beispiel
   - Zusammenhang von ereignisbasierter und prozessbasierter Modellbeschreibung

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

Nachbildung einer Uhr

aktuelle Modellzeit
zum Generierungs-/Startzeitpunkt
des jeweiligen Clock-Objektes





Ausgabe eines Punkt-Zeichens (als Tick)
zu ganzzahligen Modellzeitpunkten
als Prozess (der Ereignisfolge erzeugt)

System-Erweiterung:
zwei Uhren werden nicht synchron gestartet: Ausgabe einer überlagerten farblich markierten Punktfolge