Kurs OMSI
im WiSe 2010/11

Objektorientierte Simulation mit ODEMx

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

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

2. Prinzip der Next-Event-Simulation

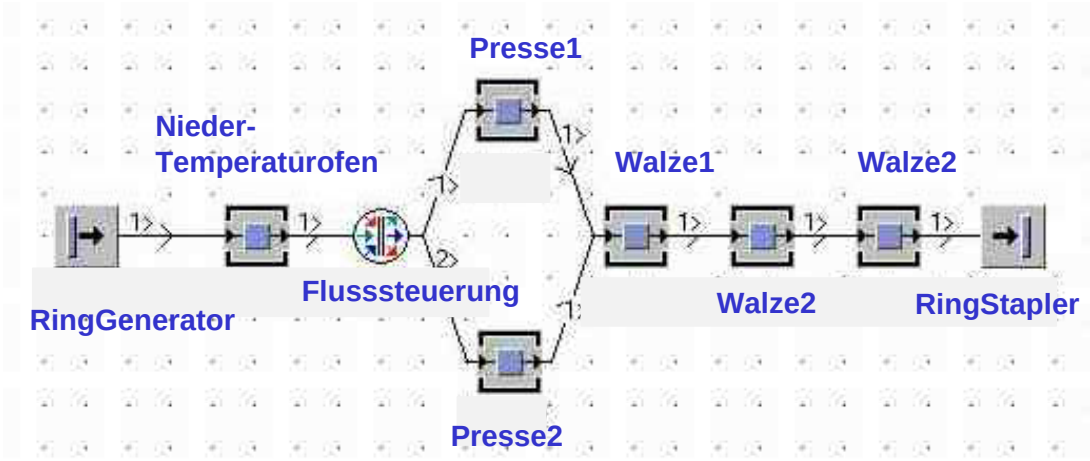
1. Charakterisierung der Next-Event-Simulation
2. Umsetzung des Prinzips in ODEMx
   - Aufbau von ODEMx
   - Simulationskontext
   - Simulationskontext (Barrenbeispiel)

Bedienungssysteme

Basis: aktive Objekte, passive Objekte
(als Variablen abstrakter Datenstrukturen)

Verhalten aktiver Objekte (Transaction): chronologische Sequenz von Ereignissen

Beispielhaft:

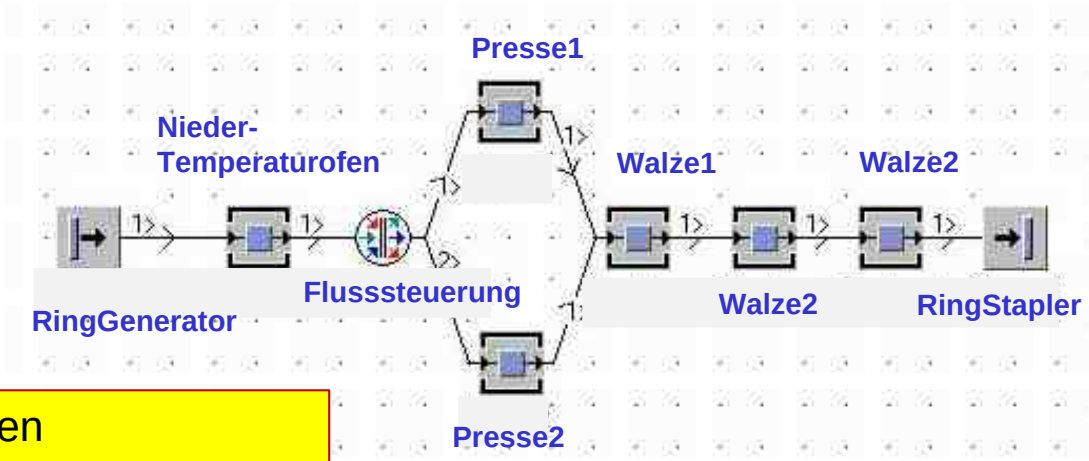


duale Problemlösungen: a) Ringe sind aktive Objekte, Stationen sind passiv
b) Ringe sind passive Objekte, Stationen sind aktiv
(Automaten mit In- und Output)

a) am häufigsten eingesetzte Technik beim Bau zeitdiskreter Simulatoren
gestattet Lebenslauf-Beschreibung für ein Werkstück
b) am häufigsten eingesetzte Technik beim Entwurf von Steuerungen der Maschinen

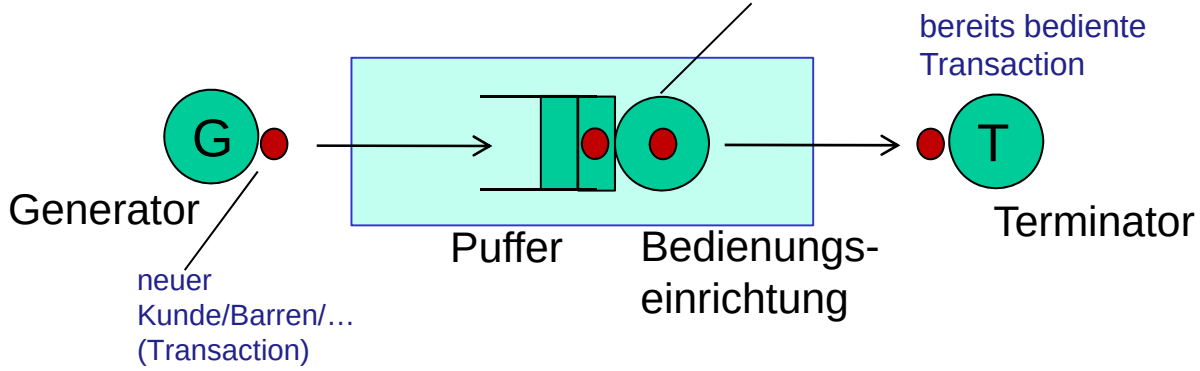
ODEMx: erlaubt beide Varianten

Verbrauch von Modellzeit



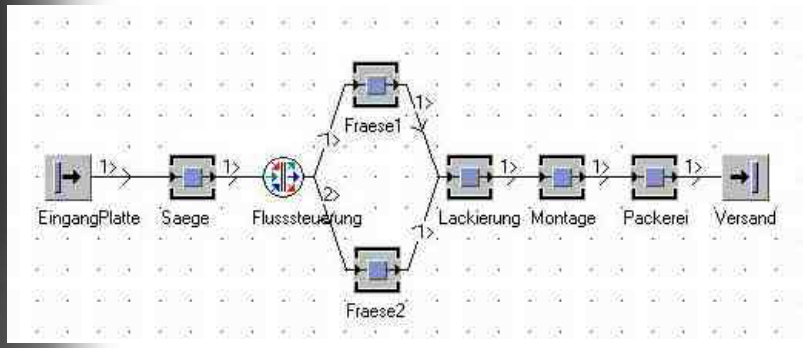
1. Zeit zwischen zwei aufeinanderfolgenden Transaction-Generierungen

bei belegter Einrichtung
kann es zur Warteschlangenbildung kommen



2. Verweildauer/Bedienungszeit einer Transaction in der Einrichtung (evtl. ist die Benutzung von Transporteinrichtungen zu berücksichtigen)

Ablaufschema der Next-Event-Simulation



Behandlung gleichzeitiger Ereignisse
- Ausführung in beliebiger sequentieller Form (z.B. FCFS, Prioritäten, ...)

- generiere initiale Systemstruktur
- initialisiere Systemzustand
- setze Modellzeit auf Null
- aktiviere Modellelemente

- **E:**= aktuelles Ereignis (erstes Element der Ereignisliste)
- setze Modellzeit auf Ereigniszeit von **E**

- entferne **E** in der Ereignisliste

- führe **E** über die zugeordneten Modelleinheitenaus

Ende der Simulation?

nein

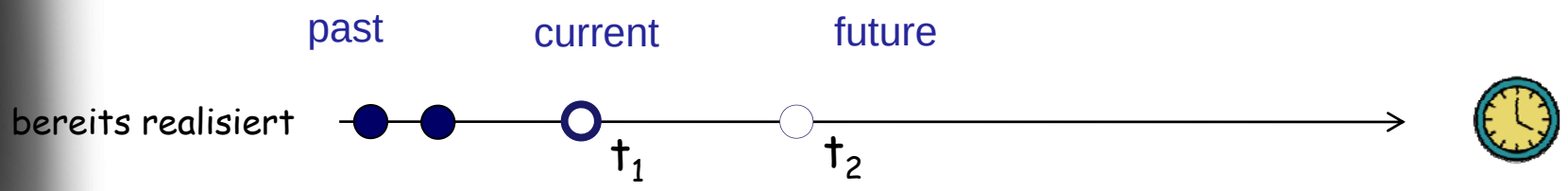
ja

Operationen der Ereignisliste
- Um- und Einsortierung
- Suche und Zugriff
- Streichen

von Ereignissen

Zustandstransformationen sind meist mit Ereignissen verbunden, die in die Ereignisliste aufgenommen werden

Ereignislistenverarbeitung (Beispiel)



Ereignisliste zum Zeitpunkt t_1

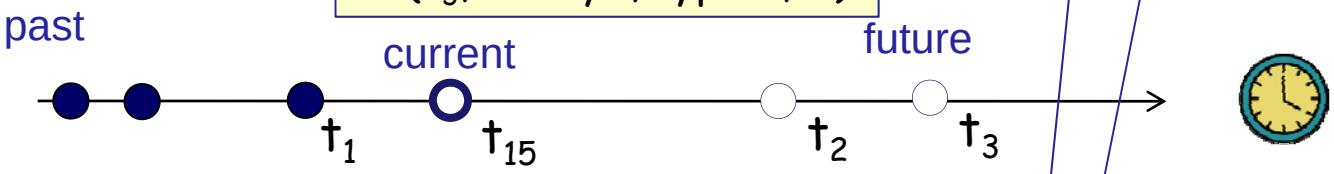
1: (t_1, entitiy1, type 2, ...)
2: (t_2, entity3, type 3, ...)

Ereignisaktionen

```
if (event type ==2) {
   generate a type 1 event for me at t15
   generate a type 2 event for entity 2 at t3
}
```

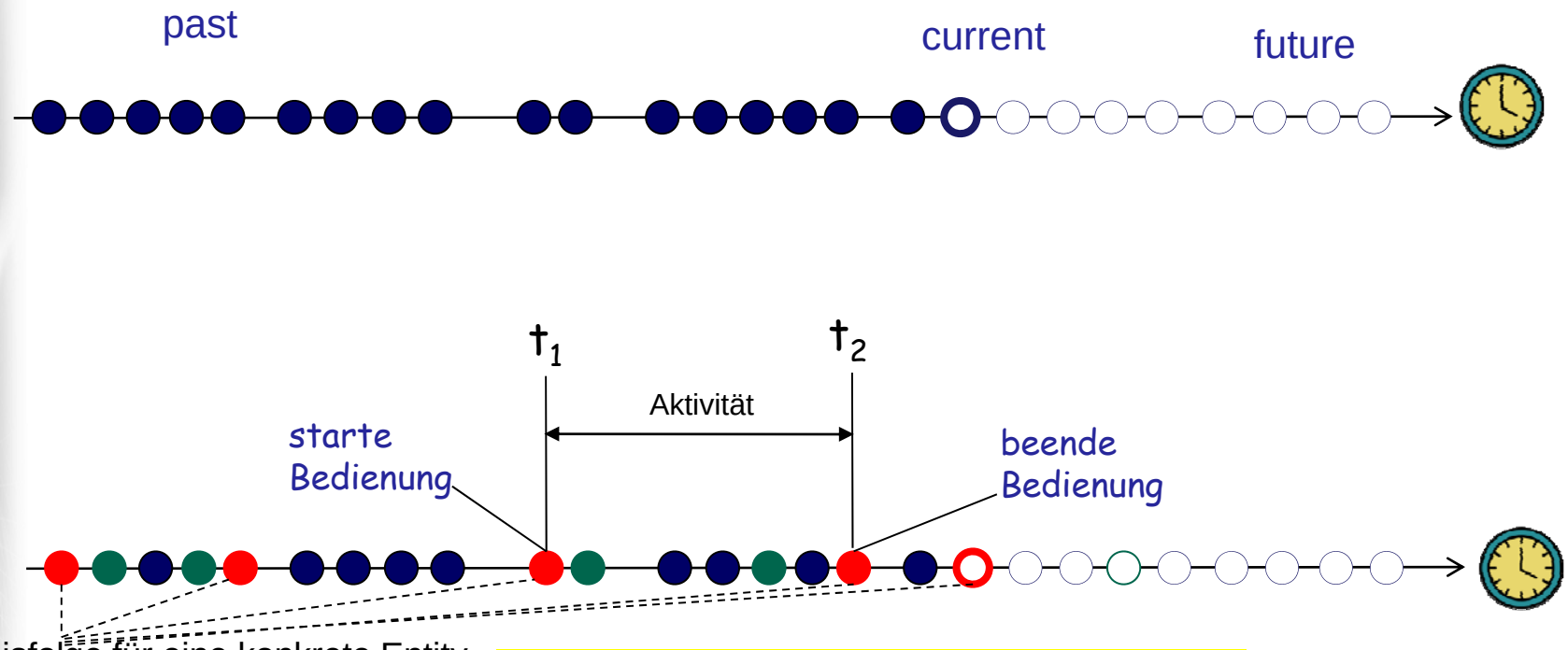
Ereignisliste zum Zeitpunkt t_{15}

1: (t_{15}, entitiy1, type 1, ...)
2: (t_2, entity3, type 3, ...)
3: (t_3, entity2, type 2, ...)



Achtung: Ereigniszeitpunkte können deterministisch oder stochastisch sein

Ereignis, Aktivität, Prozess: Zusammenhang



Ereignisfolge für eine konkrete Entity
(Objekt einer aktiven Klasse)

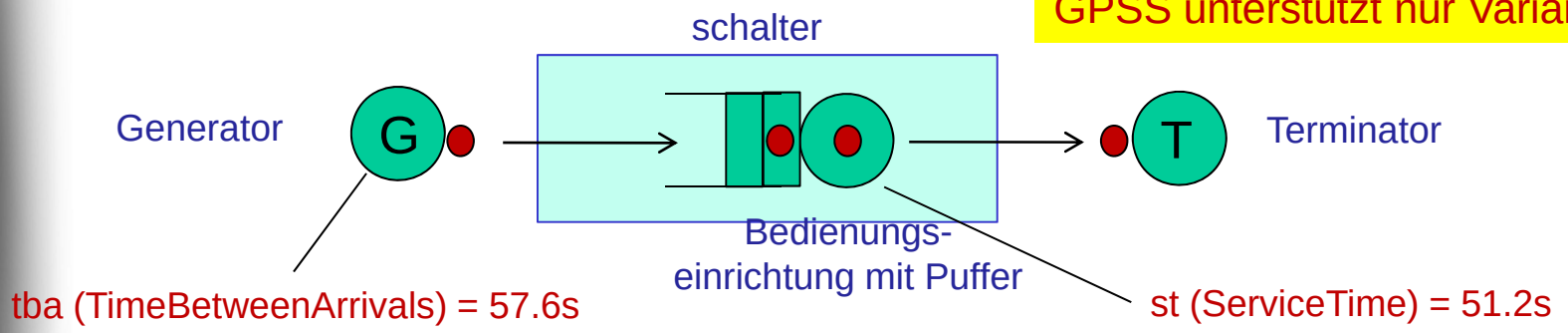
➔ Ereignisfolge als Prozessrealisierung

Ereignisfolge für eine andere konkrete Entity
(Objekt einer aktiven Klasse)

➔ Ereignisfolge als Prozessrealisierung

Elementares Bedienungssystem in GPSS

GPSS unterstützt nur Variante a)



GENERATE	tba, 1000
QUEUE	schalterQ
SEIZE	schalter
DEPART	schalterQ
ADVANCE	st
RELEASE	schalter
TERMINATE	1

neuer Kunde wird nach tba Zeiteinheiten erzeugt
dieser zeigt dann eigenständiges Verhalten

Kunde betritt zum aktuellen Zeitpunkt den Puffer schalterQ

Kunde belegt (falls möglich) zum aktuellen Zeitpunkt die Einrichtung schalter
sonst Blockierung (verlassender Kunde muss ihn wieder erwecken)

Kunde verlässt zum aktuellen Zeitpunkt den Puffer schalterQ

Kunde wird st Zeiteinheiten bedient

Kunde verlässt zum aktuellen Zeitpunkt die Einrichtung schalter und weckt den evtl. in schalterQ wartenden Nachfolgerkunden

Reduziere den GENERATE Startzähler (initial 1000) um eins

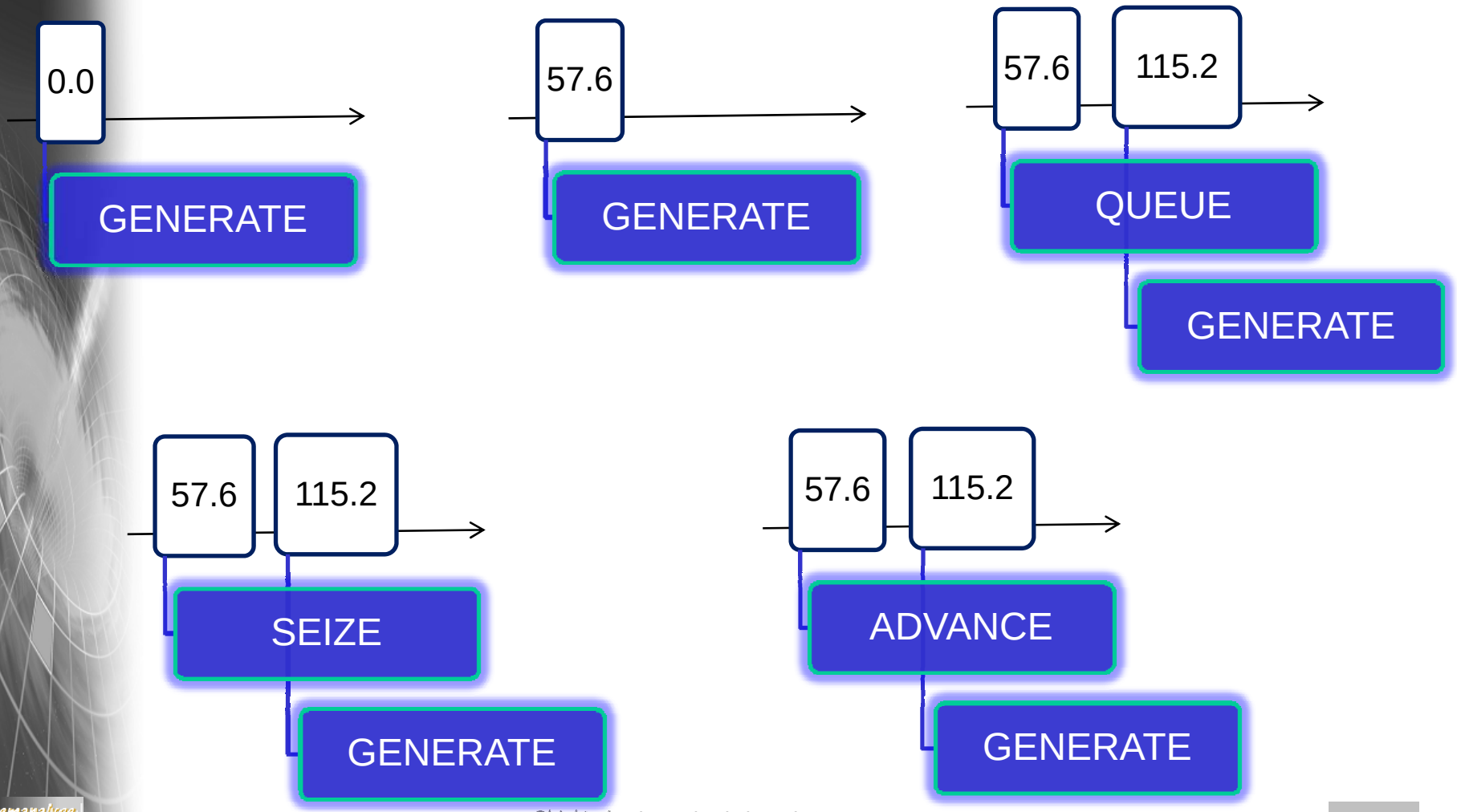
GPSS-Grundphilosophie

- konzpiert für diskrete Ereignissimulation
- Bewegung von Transaktionen (Kunden, Produkte, Aufträge), die schrittweise bedient/ verarbeitet werden
- Bearbeitung/Bedienung erfolgt in Stationen
- jede individuelle Transaction verändert in Abhängigkeit der aktuell belegten Einrichtung seine eigenen Attribute (Transactionzustand und die Attribute des Einrichtung und …)
- bei Verbrauch von Modellzeit

1. Ereignis = Folge sequentieller Operationen zu einem fixierten Modellzeitpunkt
2. Ereignisse werden in einem Terminkalender chronologisch sortiert (man benötigt Lösung für Gleichzeitigkeitsprobleme) und in dieser Reihenfolge ausgeführt
3. aktuelle Modellzeit ist gleich der Ereigniszeit des ersten Kalendereintrages
4. eine Ereignisoperation kann neben Zustandsänderungen neue Ereignisse generieren (aber nur mit Ereigniszeiten, die größer oder gleich der aktuellen Modellzeit sind)

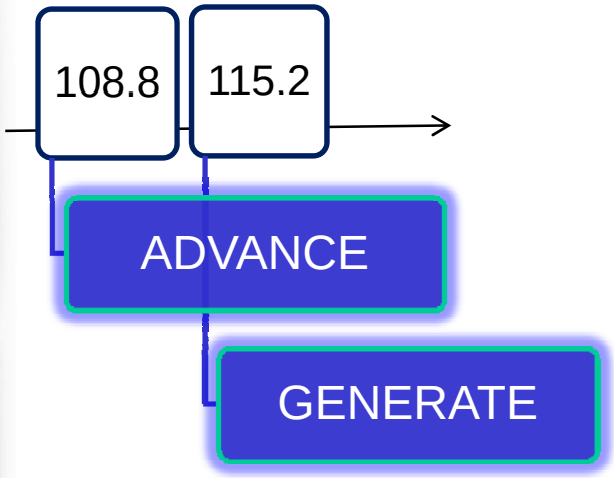
Ablauf als Ereignisfolge (1)

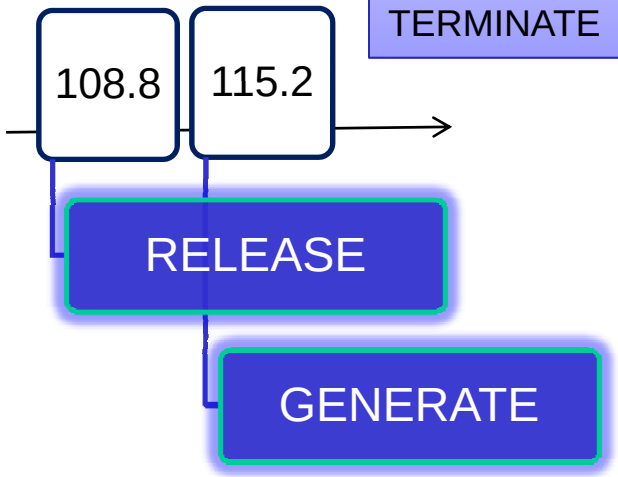
GENERATE	tba, 1000
QUEUE	schalterQ
SEIZE	schalter
DEPART	schalterQ
ADVANCE	st
RELEASE	schalter
TERMINATE	1

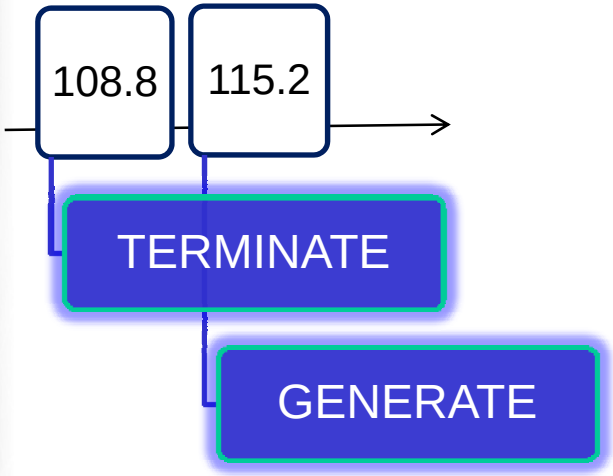


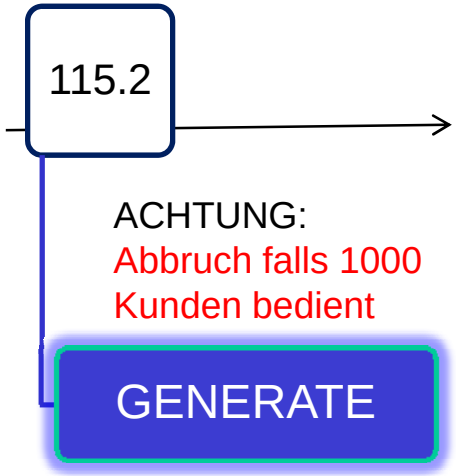
Ablauf als Ereignisfolge (2)

GENERATE	tba, 1000
QUEUE	schalterQ
SEIZE	schalter
DEPART	schalterQ
ADVANCE	st
RELEASE	schalter
TERMINATE	



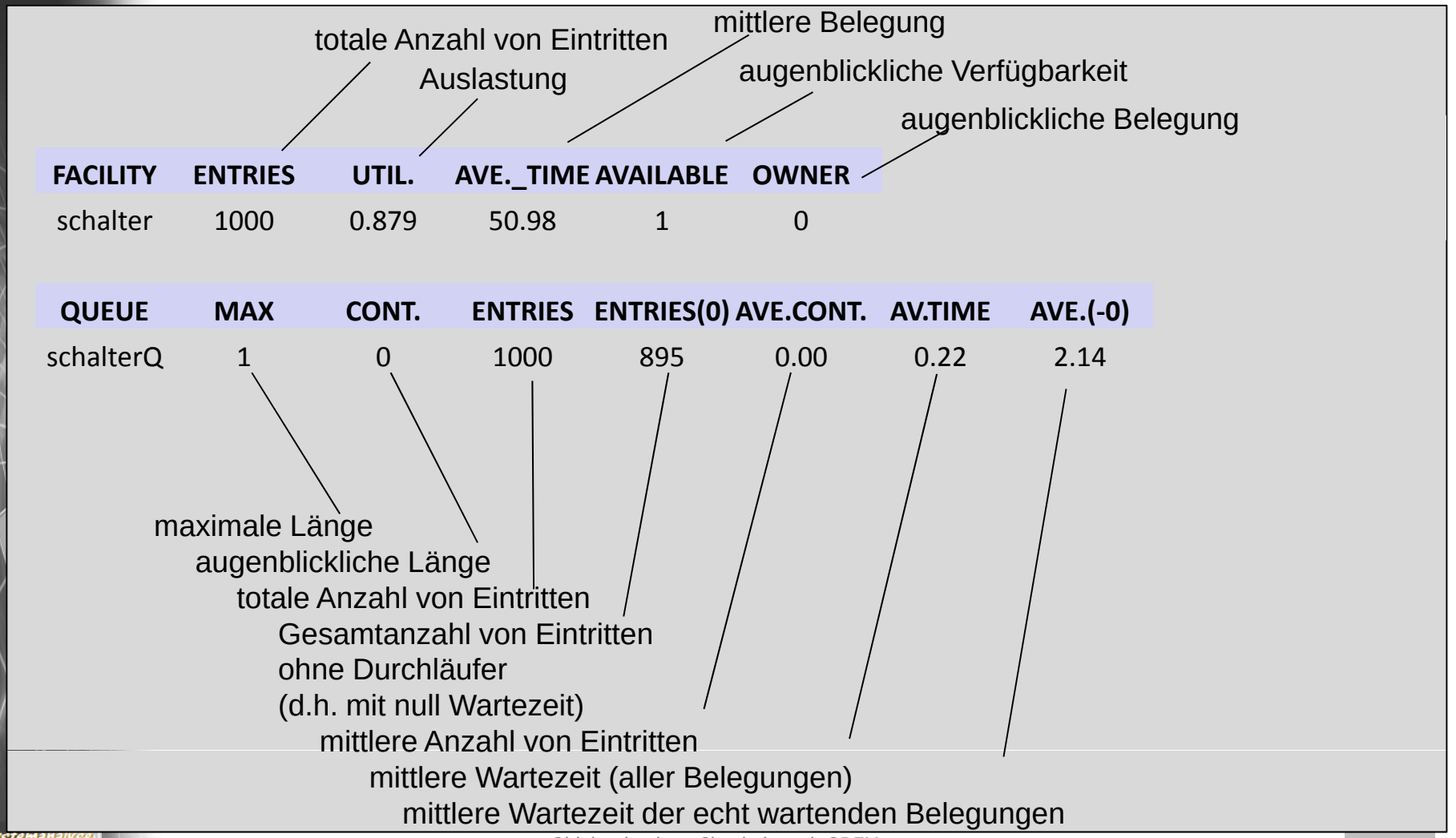






Typische Ausgabe eines NextEvent-Simulators

Statistik für alle Bedienungseinrichtungen:



2. Prinzip der Next-Event-Simulation

1. Charakterisierung der Next-Event-Simulation
2. Umsetzung des Prinzips in ODEMx
   - Aufbau von ODEMx
   - Simulationskontext
   - Simulationskontext (Barrenbeispiel)

Entwurfsmaxime für die ODEMx- Bibliothek

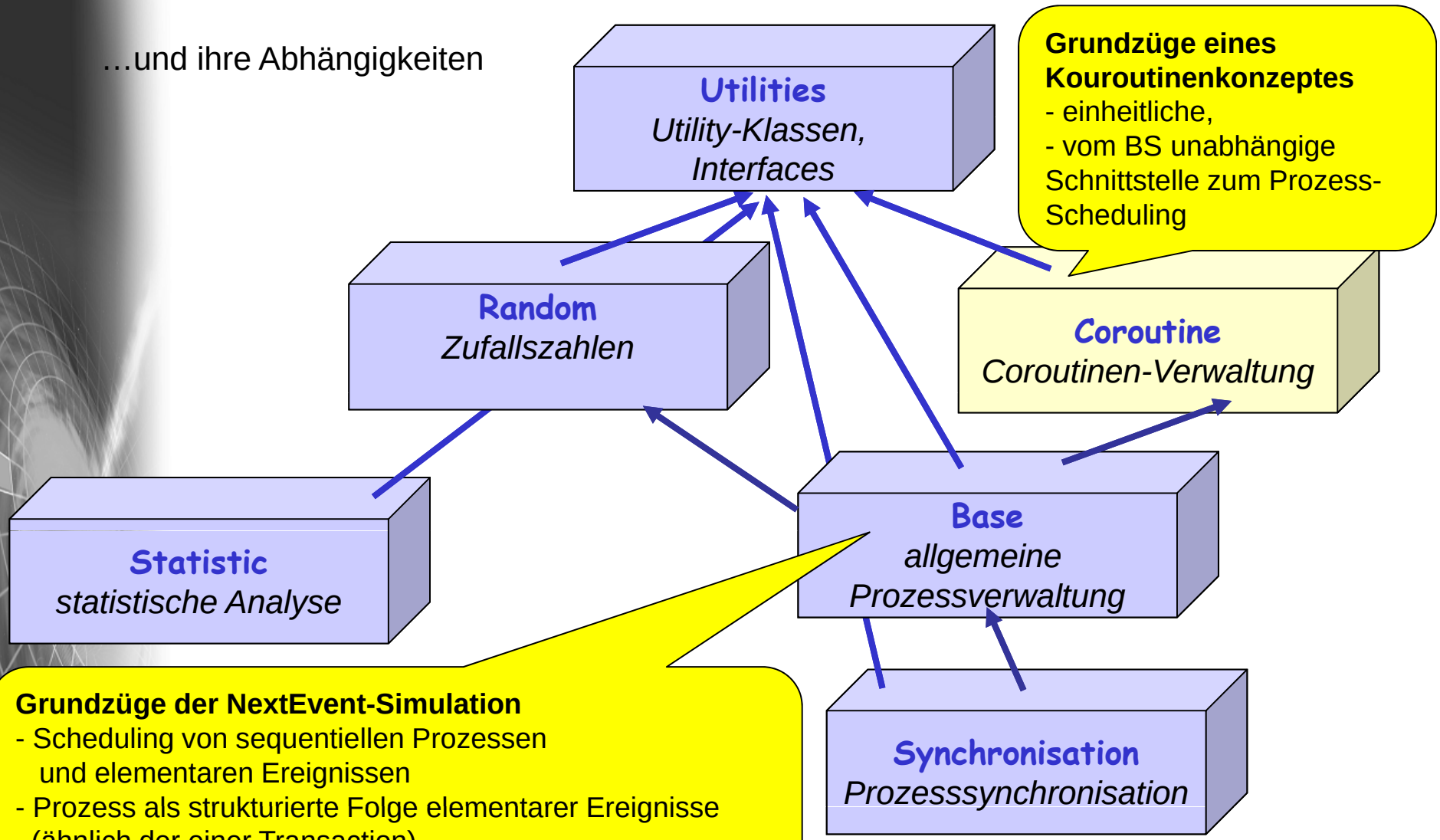
- Basis: ältere ODEM-Bibliothek
- leichte Benutzbarkeit:
  automatische Beobachtung, Sammlung von
  Modellwerten und deren statistische Auswertung
  vordefinierte Synchronisationsmechanismen
- ein C++ (Haupt-)Programm erlaubt parallele (unabhängige
  oder abhängige) System-Simulationen als Komponenten mit
  individueller Stack-Verwaltung,
  jede dieser Komponente verwaltet ein eigenständiges Ensemble pseudo-
  paralleler Prozesse,
  die in Abhängigkeit vom jeweiligen Modellzeitverbrauch
  alternierend ausgeführt werden (individuelle Prozessterminkalender)
- Standardfall:
  es gibt genau eine (Default-)Komponente, die (nur) ein Ensemble pseudo-
  paralleler Prozesse verwaltet (d.h. nur einen einzigen Prozessterminkalender)
- neue Version der ODEMx-Bibliothek erlaubt u.a. die laufzeiteffizientere
  Modellierung einzelner zeitdiskreter Ereignisse neben Prozessen

2. Prinzip der Next-Event-Simulation

1. Charakterisierung der Next-Event-Simulation
2. Umsetzung des Prinzips in ODEMx
   - Aufbau von ODEMx
   - Simulationskontext
   - Simulationskontext (Barrenbeispiel)

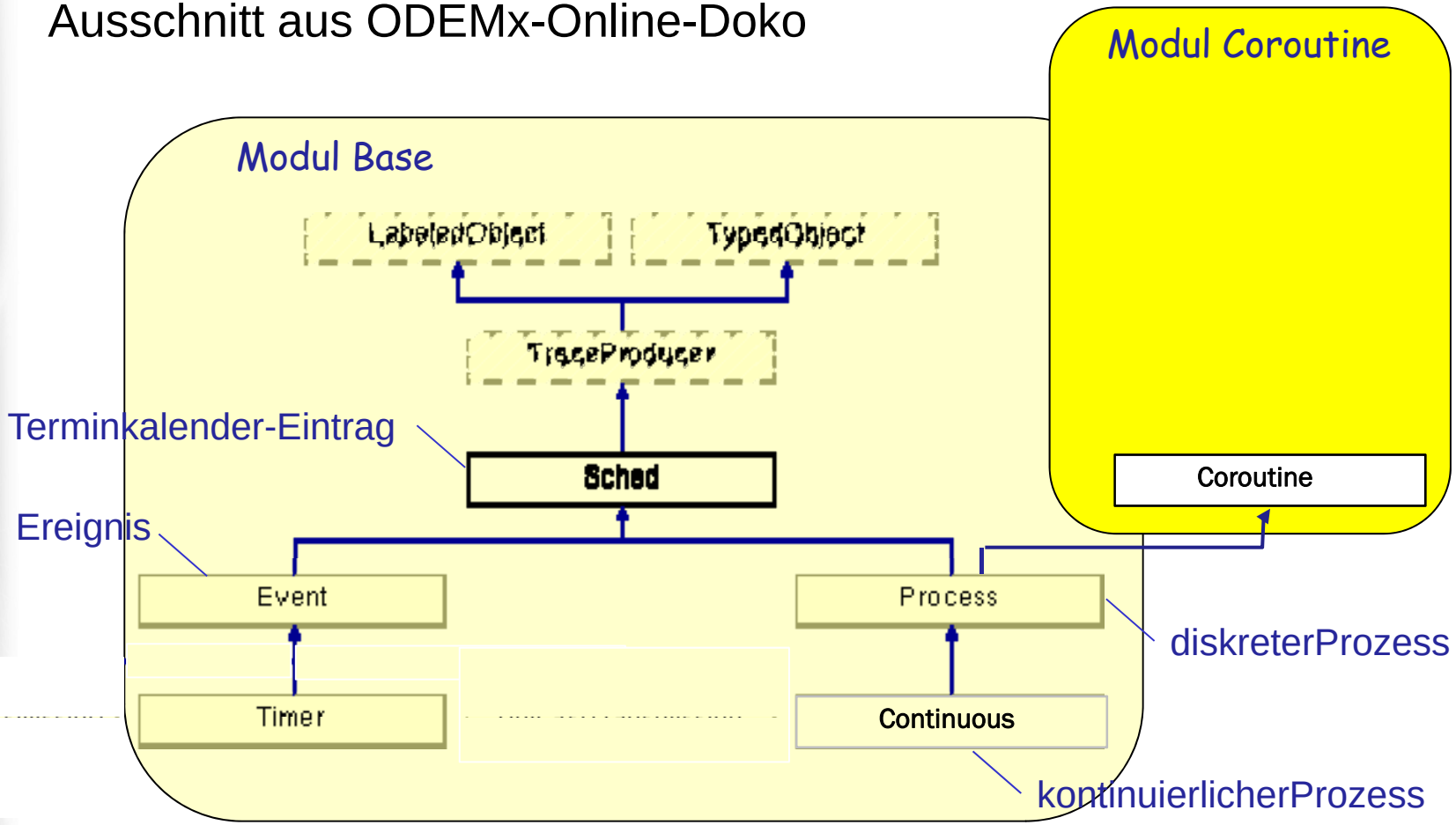
Die ODEMx-Module

…und ihre Abhängigkeiten



Klassenhierarchie

Ausschnitt aus ODEMx-Online-Doko



Timer als Ereignis
nicht als Prozess

Prozessverwaltung

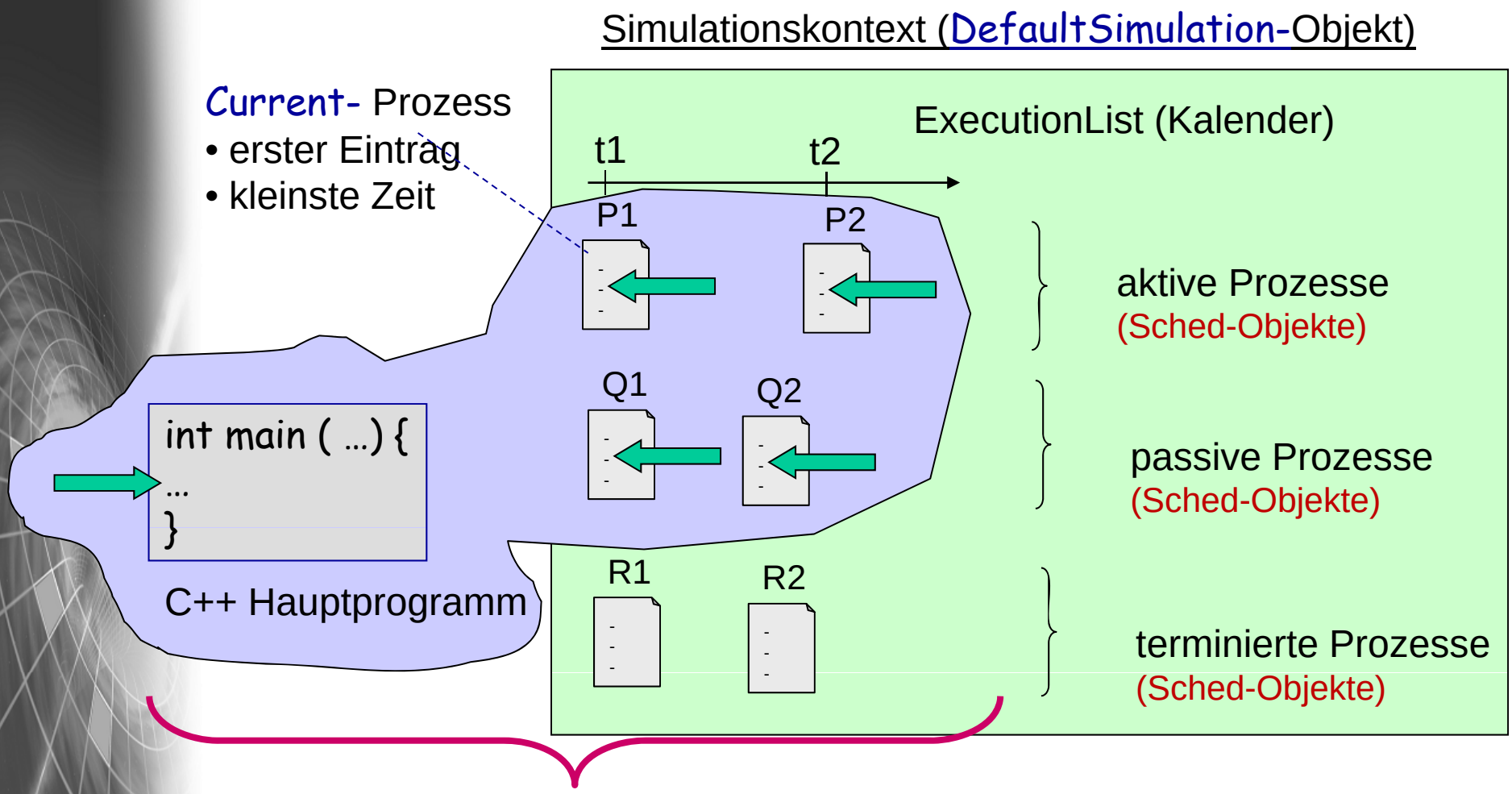
- realisiert über Simulation- Funktionalität
  - Verwaltung zustandsabhängiger Prozess- und Ereignislisten (z.B Terminkalender)
  - Erfassung sämtlicher Objekte (Zufallszahlengeneratoren, Synchronisationsobjekte, …)

  ➔ stellt **Kontextinformation** für die Simulation eines Systems/Teilsystems bereit

einfache Prozess- und Ereignisverwaltung
durch Nutzung eines
DefaultSimulation-Objektes

hierarchische Prozess- und Ereignisverwaltung
parallele Prozessverwaltung von Teilsystemen
durch Nutzung von Objekten von
nutzereigenen Simulation-Ableitungen
(ein Objekt pro Ableitung)

Grundidee einer einfachen Prozessverwaltung

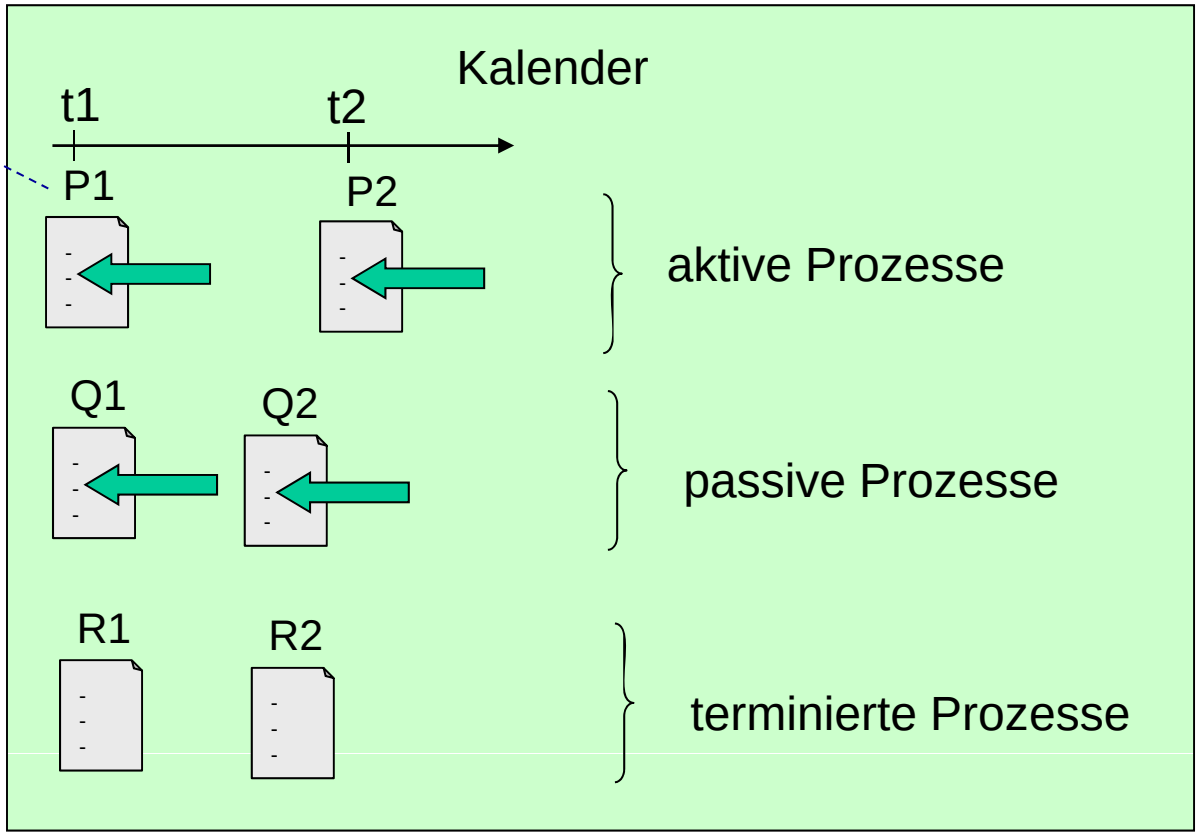


Hauptprogramm und Prozesse eines Simulationskontextes bilden ein Coroutinensystem

Prozessverwaltung in ODEMx

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

simulation context (DefaultSimulation-Objekt)



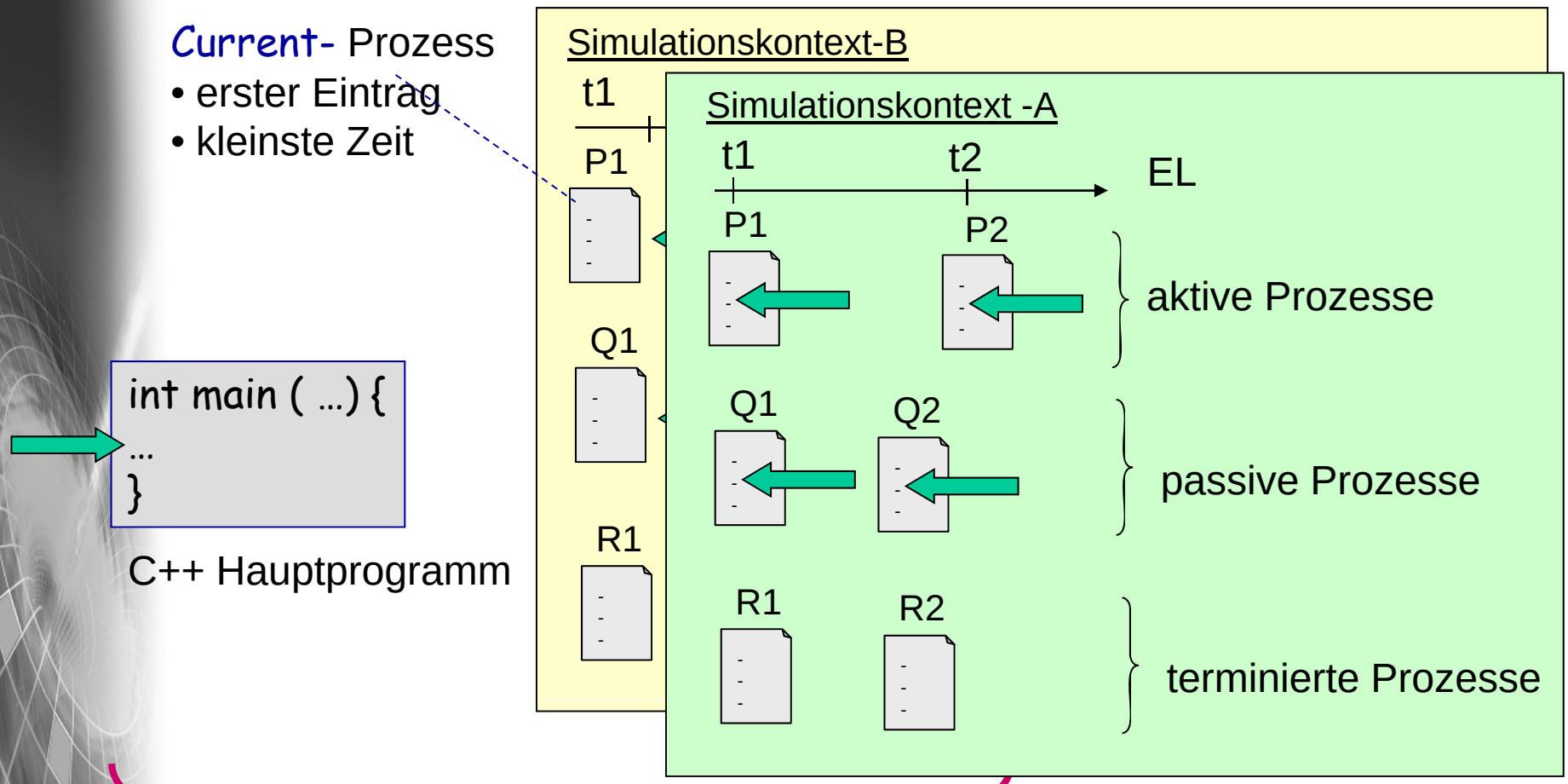

```
int main ( ...) {
...
}
```

C++ main program

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

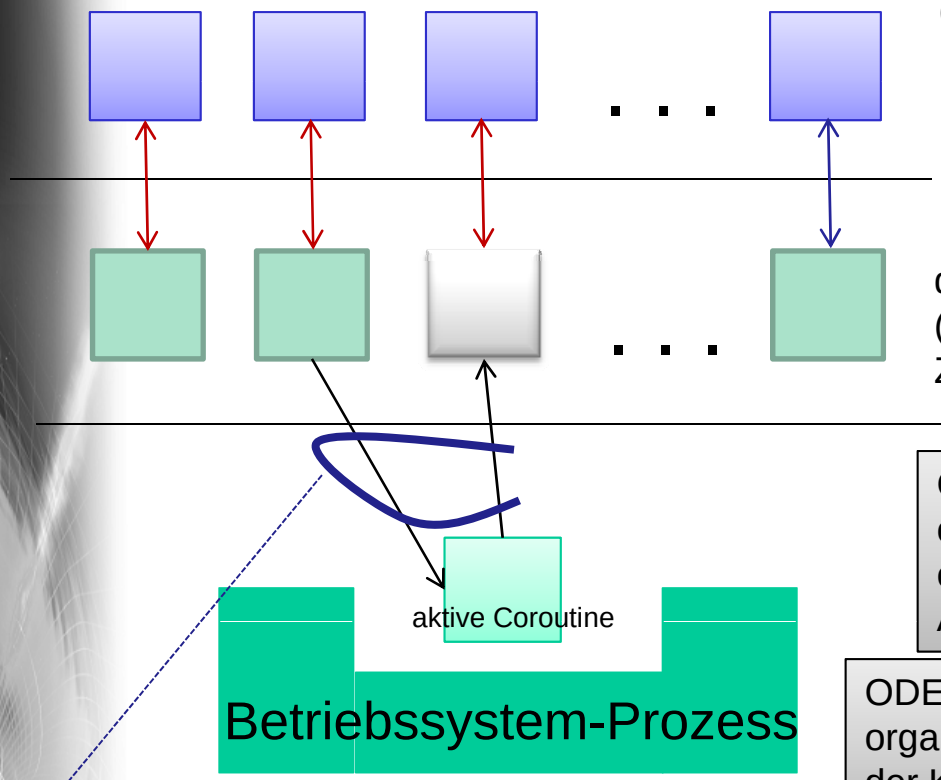
Grundidee einer hierarchischen Prozessverwaltung

Current- Prozess
- erster Eintrag
- kleinste Zeit



Hauptprogramm und Prozesse aller Simulationskontexte bilden ein hierarchisches Coroutinensystem

Universelle ODEMx- Urvariante



dynamisches Ensemble von realen/gedachten Prozessen (zeitskret/zeitkontinuierlich) (Original)

Struktur-Äquivalenz (1:1)

dynamisches Ensemble von ODEMx-Prozessen (Simulationsmodell)
Zustand jedes Prozesses auf der Halde

ODEMx-Laufzeitsystem
organisiert Scheduling der ODEMx-Prozesse entsprechend ihrem Modellzeitverbrauch für Änderung ihrer Zustandsgrößen

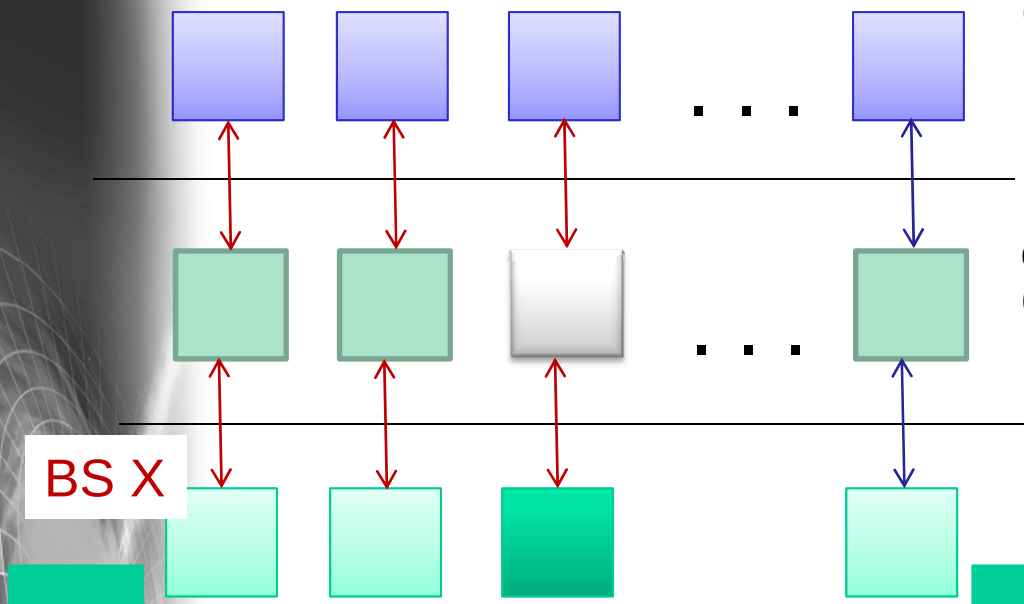
ODEMx-Laufzeitsystem
organisiert Speicherung und Restaurierung der Koroutinen-Zustände (Laufzeitkeller, Register)

rechenzeitaufwändiger

… im Vergleich zu prozeduralen Next-Event-Verfahren

aber hoher Grad an auf natürliche Weise erreichbarer Strukturäquivalenz

Laufzeitverbesserte Varianten



dynamisches Ensemble von realen/gedachten Prozessen (zeitdiskret/zeitkontinuierlich) (Original)

Struktur-Äquivalenz (1:1)

dynamisches Ensemble von ODEMx-Prozessen (Simulationsmodell)

Struktur-Äquivalenz (1:1)

BS X

Betriebssystem-Prozess

ODEMx-Laufzeitsystem
organisiert Scheduling der ODEMx-Prozess- entsprechend ihrem Modellzeitverbrauch für Änderung ihrer Zustandsgrößen

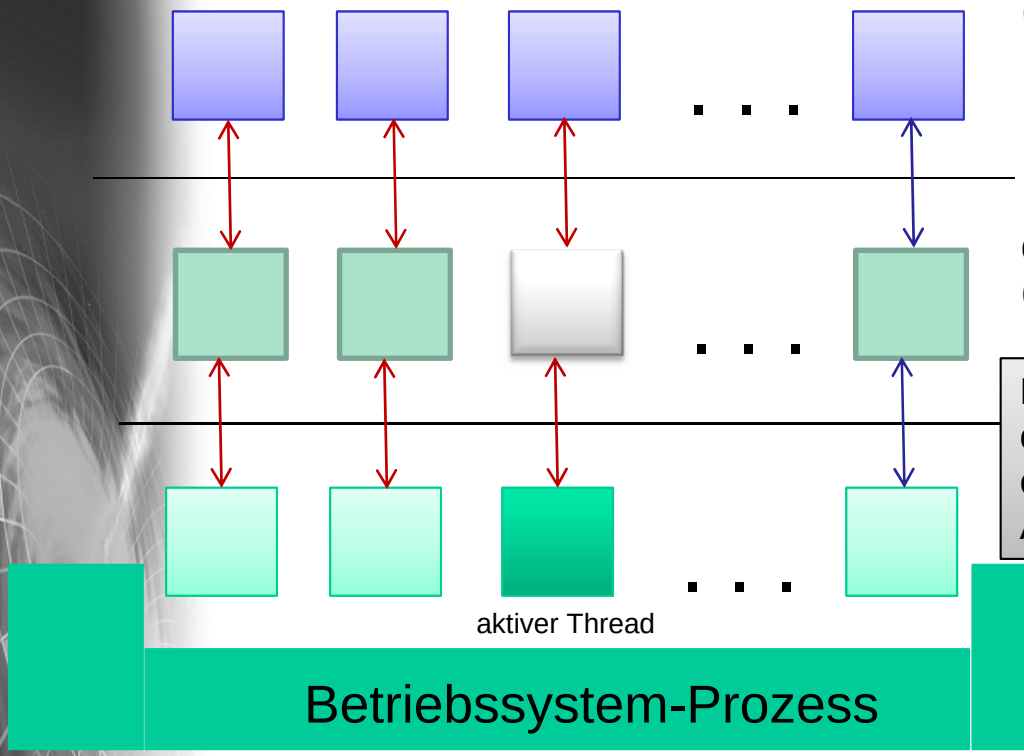
X= Fibres, MS Windows

X= uContext, Linux

ODEMx-Laufzeitsystem
organisiert Umsetzung von Zeigern zu Koroutinen-Zustände (Laufzeitkeller) und speichert und restauriert lediglich Register

Java-basierte Variante (DESMO-J)

DESMO-J = ODEMx – {zeitkontinuierl., Protokoll-Konzepte}



dynamisches Ensemble von realen/gedachten Prozessen (zeitdiskret/zeitkontinuierlich) (Original)

Struktur-Äquivalenz (1:1)

dynamisches Ensemble von DESMO-J-Prozessen (Simulationsmodell)

DESMO-J-Laufzeitsystem
organisiert Scheduling der DESMO-J-Prozesse entsprechend ihrem Modellzeitverbrauch für Änderung ihrer Zustandsgrößen

aber laufzeitschlechter als ODEMx-Urvariante (schwergewichtigere Lösung)

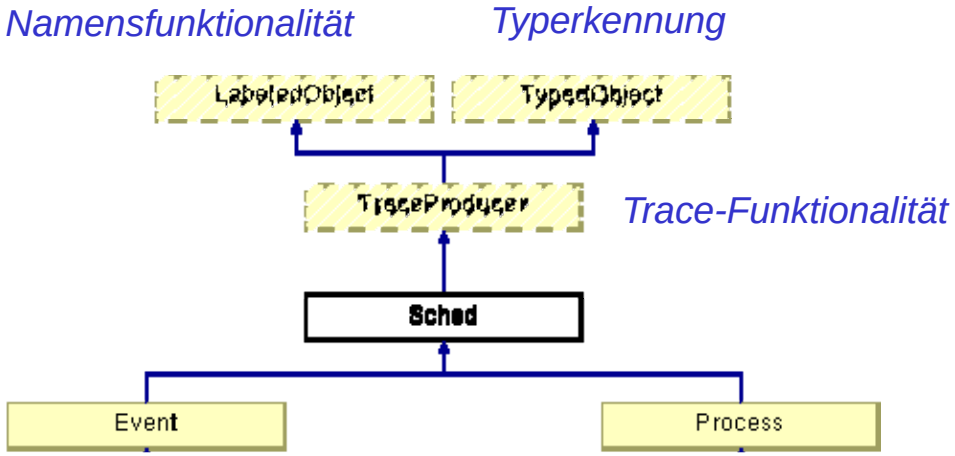
Klasse Sched, Event, Process

Sched

- abstrakte Klasse
- Objekte werden im Kalender in chronologischer Reihenfolge erfasst

Simulationslauf

- ist die Ausführung (execute) von Sched-Objekten
- in Abhängigkeit von
  - der jeweiligen Kalenderkonstellation und
  - der Typen der Sched-Objekte



dabei können neben Zustandsänderungen auch
- Eintragungen,
-Verschiebungen und Streichungen von Sched-Objekten vorgenommen

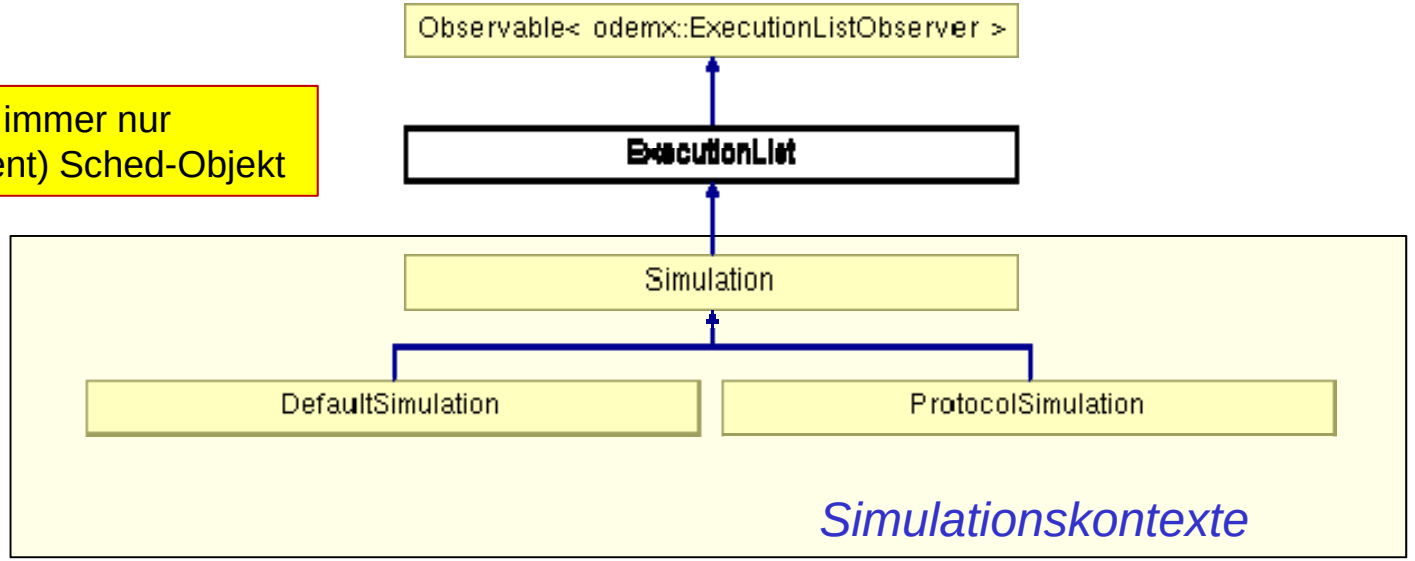
```
virtual SimTime getExecutionTime () const =0          // Get model time.
virtual SimTime setExecutionTime (SimTime time)=0     // Set model time.
virtual Priority getPriority () const =0              // Get priority.
virtual Priority setPriority (Priority newPriority)=0 // Set new priority.
bool isScheduled () const                             // Check if Sched object is in schedule.
SchedType getSchedType () const                       // Determine the Sched object's type.
virtual void execute ()=0                             // Execution of Sched object.
```

Die Klasse ExecutionList *(Ereignisliste, Kalender)*

```
Sched * getNextSched ()    // top most Sched in ExecutionList
bool isEmpty ()            // check if ExecutionList is empty
virtual SimTime getTime () // const =0  get model time
```

Vergangenheit wird nicht konserviert

ausgeführt wird immer nur das **erste** (current) Sched-Objekt



jeder Simulationskontext (Objekt von Simulation bzw. Ableitung) verfügt über eine eigene ExecutionList-Funktionalität

Scheduling-Prinzip: A Sched object is scheduled
- at a given time
  considering its priority and a FIFO- or LIFO- strategy,
  or
  in relation to an already scheduled object.

Modellzeit

Wert vom Typ SimTime

getTime() liefert stets die aktuelle Modellzeit (größer gleich Null)

ODEMx-Bibliothek in zwei Varianten:

- diskrete Zeit: long
- kontinuierliche Zeit: double

ODEMx-Bibliothek mit unterschiedlichen Ausbaustufen:

- zeitdiskrete Zustandsänderungen
- zeitdiskrete und zeitkontinuierliche Zustandsänderungen