# *Kurs OMSI*
## *im WiSe 2013/14*

## *Objektorientierte Simulation mit ODEMx*

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

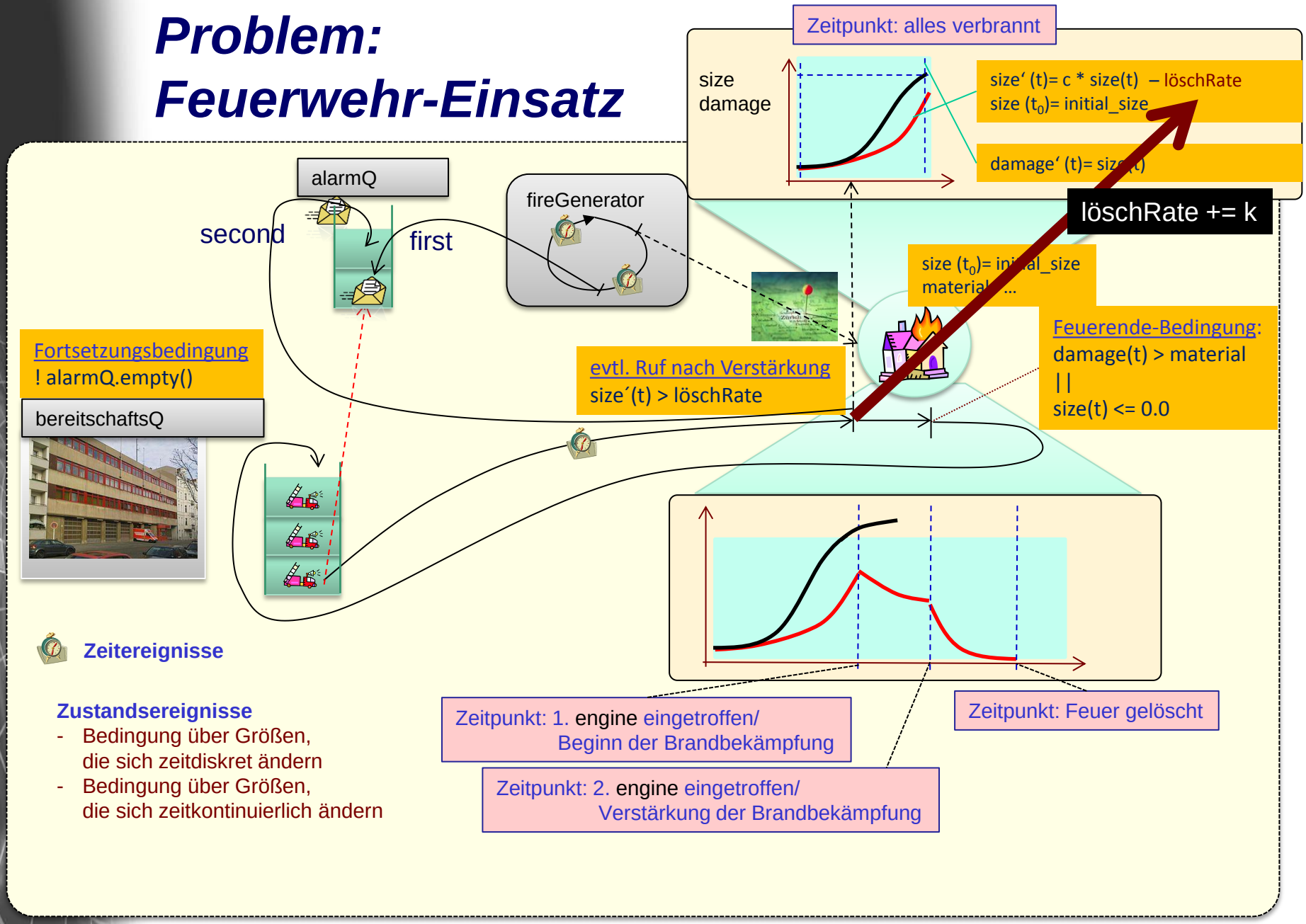
fischer|ahrens|eveslage@informatik.hu-berlin.de

# 8. Ausblick:

## *Behandlung zeitkontinuierlicher Zustandsänderungen*

- Beispiel: Feuerwehreinsatz
- Konzept für die zeitkontinuierliche Simulation
- Gleichgewicht, Stabilität, Chaotische Systeme

# Problem: Feuerwehr-Einsatz

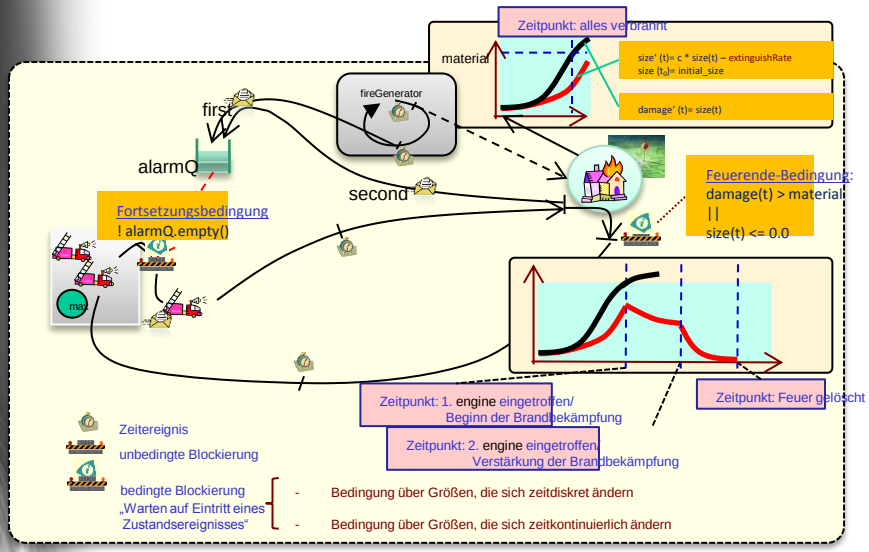



**Zeitereignisse**

**Zustandsereignisse**

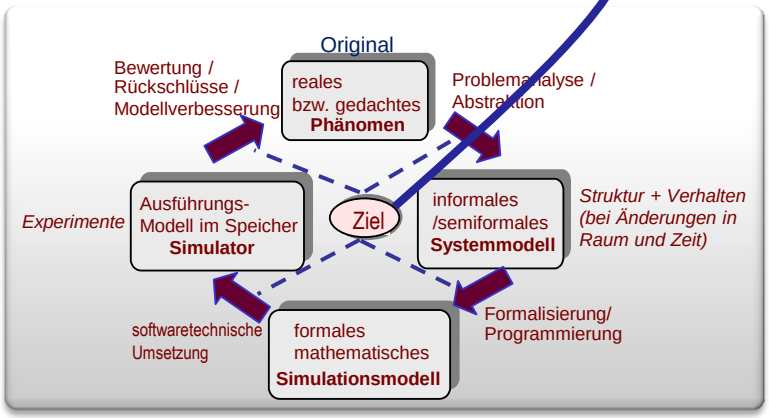
- Bedingung über Größen, die sich zeitdiskret ändern
- Bedingung über Größen, die sich zeitkontinuierlich ändern

# Informales → Formales System-Modell




mögliche Zielstellung ~ Untersuchung

- zur Anzahl und Ausstattung der Feuerwehrstationen
- zum Strategie-Vergleich
  (1 oder 2 Fahrzeuge als Ersteinsatz)
- Einsatz von WasserTanks in der Stadt




OO- Modell
- Vorzüge
- Abstraktionsparadigmen

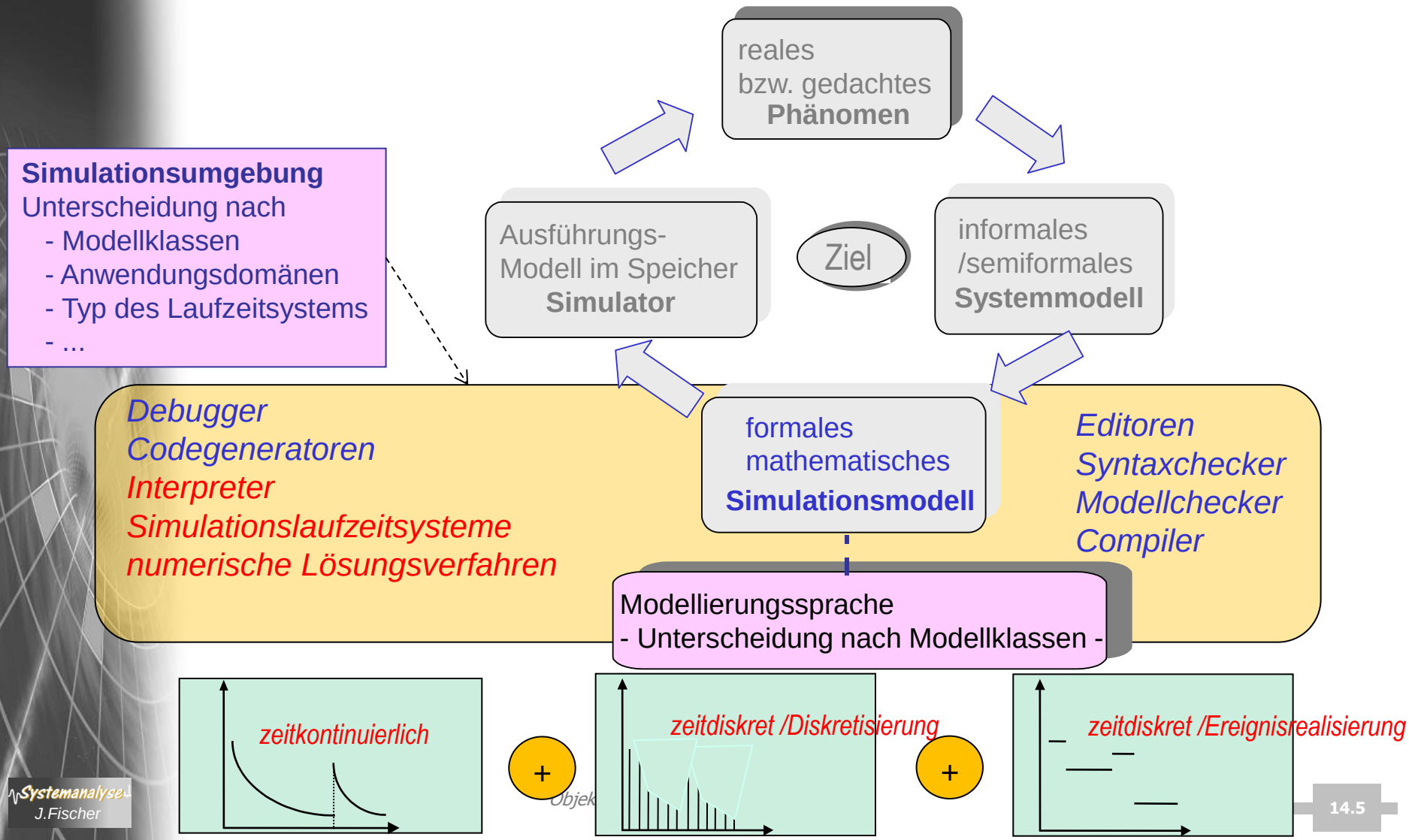
Struktur
Modellierung

Verhaltensmodellierung
Modellierung

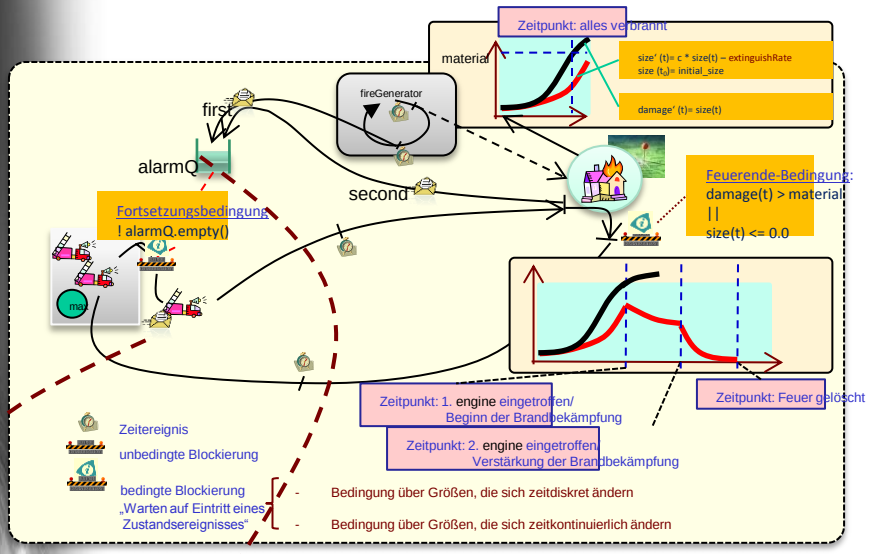
Math. Modellklasse
Verhaltensbeschreibung

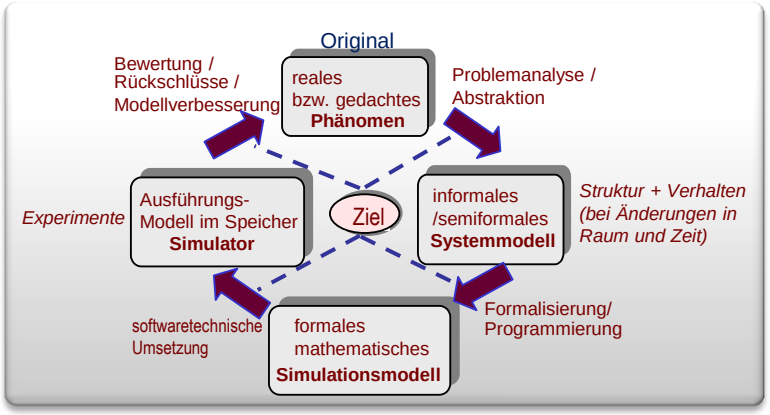
# Modellierungssprachen und Simulationsumgebung

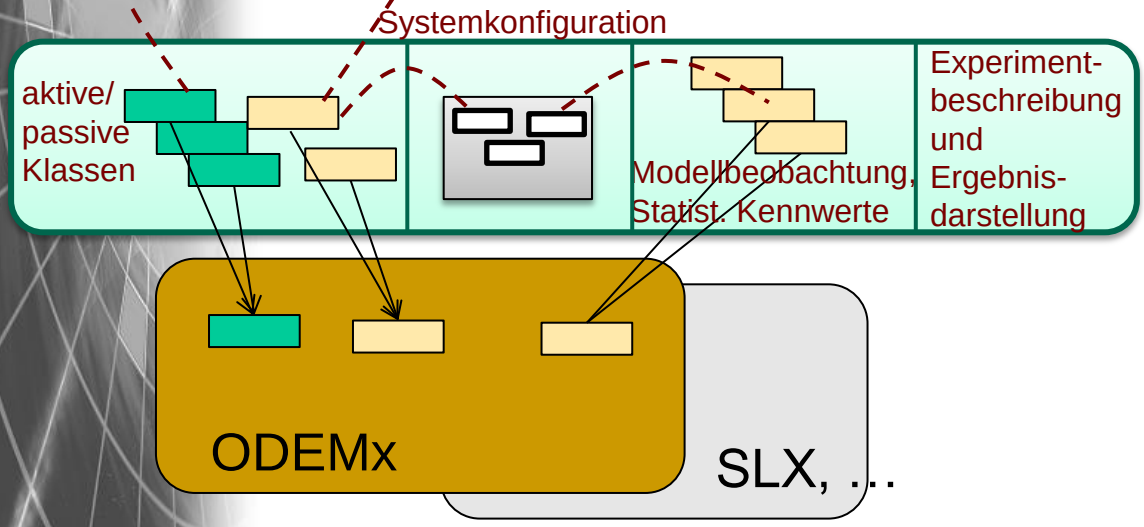
Zustandsänderungen kontinuierlich oder/und diskret in Raum und Zeit



# Benötigte Konzepte




**benötigte Konzepte**

- diskrete Prozesse, Zustandsereignisse
- Kontinuierliche Prozesse, Zustandsereignisse
- Port-Synchronisation (Alarm und Engine)
- CondQ-Synchronisation (Rückfahrt von Engine)
  (**aber**: wer gibt das Signal?)
- Zufallsgrößen
  - Hausbrand: (Zeitpunkt der Enstehung, Position, Anfangsgröße, Materialwert, Materialkoeffizient Zeitpunkt der Entdeckung)
  - Fahrtzeiten der Feuerwehr
- Tally/Count
- Graph/Tabelle

# 8. Ausblick:

***Behandlung zeitkontinuierlicher Zustandsänderungen***

- Beispiel: Feuerwehreinsatz
- Konzept für die zeitkontinuierliche Simulation
- Gleichgewicht, Stabilität, Chaotische Systeme

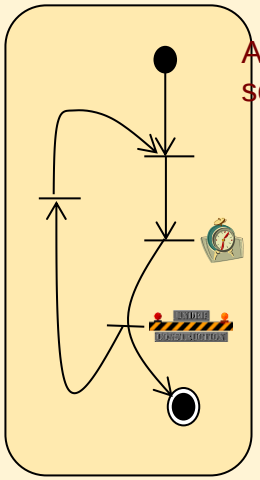
# Sequentieller Prozess

mit Zustandsgrößen, die sich zeitdiskret ändern

**generelles Problem**
Synchronisation nebenläufiger abhängiger Prozesse

*ODEMx-Lösung:*
*NextEventSimulation mit Process-Scheduling, WaitQ, CondQ, Bin, Res*

Attribute



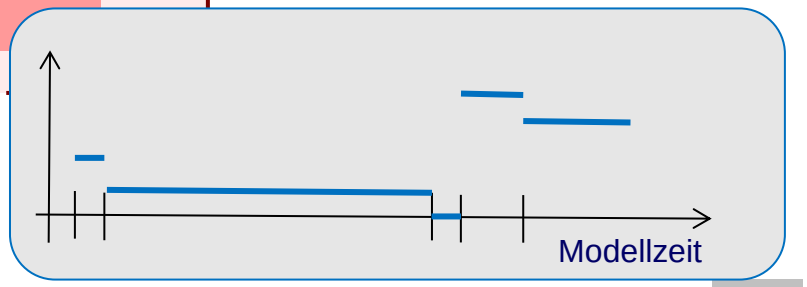
Aktionsfolge sequentiell

*ändert Zustand des Systems, d.h.*
*eigene*
*fremde*
*Attribute*
*erzeugt/vernichtet/aktiviert Objekte*
*mit*
*- eventl. Zeitverbrauch,*
*- zustandsbedingten Blockierungen,*
*- unbedingten Blockierungen*

Umgebung (Prozesse, Objekte passiver Klassen globale Größen)

Attribute

Attribute




zeitdiskrete Änderung einer einzelnen Zustandsgröße

# Sequentieller Prozess

mit Zustandsgrößen, die sich zeitkontinuierlich ändern

Differential-Gleichung als virtuelle Fkt.

elementare Attribute: nur **double**
meist in Strukturen/Feldern verpackt

*ODEMx-Lösung: Kontinuierliche Prozesse als Ableitung diskreter Prozesse*

Attribute

*ändert Zustand*
- *eigene*
- *evtl. auch fremde Attribute (zeitkontinuierlicher Prozesse)*

zeitliche Verzögerung um Schrittweite **h**

$10^{-5}$

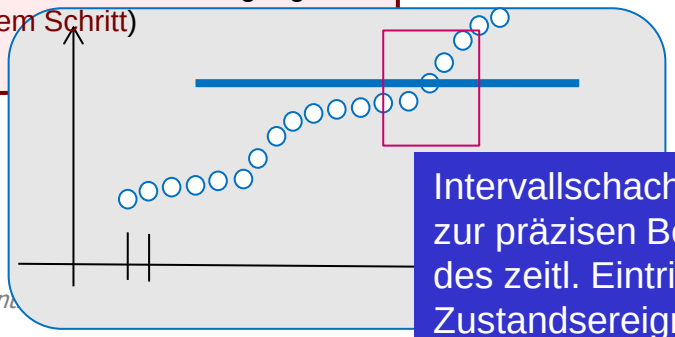
**Integrationsverfahren**
als Ein-Schritt-Verfahren

sequentielle Aktionen,
die einen Integrationsschritt
mit Schrittweite **h** ausführen

condQ.Signal()

Attribute

**Abbruch/Unterbrechung**
- nach Erreichen einer bestimmten Zeit
- nach Eintritt einer Zustandsbedingung
(Test nach jedem Schritt)



zeitdiskrete Approximation
der Änderung einer einzelnen
zeitkontinuierlichen Zustandsgröße

Intervallschachtelung zur präzisen Bestimmung des zeitl. Eintritts des Zustandsereignisses

# Grundsätzliche Einteilung von Systemen und Modellen

- Einteilung von (Teil-) Systemen
  - **zeitdiskrete** Prozesse
    - kennen wir bereits
  - **zeitkontinuierliche** Prozesse
    - nach höchster Ableitung
    - nach Anzahl der Zustandsgrößen
    - System n-ter Ordnung hat n Zustandsgrößen

kombinierte Systeme
**Beispiele:**
- Niedrigtemperaturofen,
- Feuerwehr,
- Tanker-Tank

modelliert als System
von n Differentialgleichungen 1. Ordnung

(math. äquivalent zu einer Differentialgleichung n-ter Ordnung)

size‘ (t)= c * size(t)
damage‘ (t)= size(t)

Hausbrand ~ kontinuierlicher Prozess
mit zwei Zustandsgrößen

- Gewöhnliche DGL
- als Anfangswertaufgabe
- (Teil-)system 1.Ordnung der Dimension 2
- Lineare DGL

size (0.0)= initial size
damage (0.0)= initial_damage

# Integrationsverfahren

Differential-Gleichung als virtuelle Fkt.

Betrachten hier sehr einfaches Verfahren (Euler-Heun-Verfahren)

size‘ (t)= c * size(t) – extinguishRate

damage‘ (t)= size(t)

- Vektor x(t) z.Zeitpunkt $t= t_0$ gegeben

  hier:

  size $(t_0)$ = initial_size

  damage $(t_0)$ = initial_size

- Berechnung der Änderungen zum Zeitpunkt t

  hier x‘(t) berechnet sich entsprechend der beschreibenden DGLs

  Vor.: man kennt x zu diesem Zeitpunkt t

  (Anfangswert muss gegeben sein)

- **Prädiktor-Schritt:**

  Berechnung der neuen Werte zum Zeitpunkt t+h (Schrittweite h)

  x(t+h)= x(t) + h* x‘(t) // x´(t)= (size‘, damage‘) (t) ➔ r1

  Berechnung der Ableitung (Änderungen) zum Zeitpunkt t+h

  nach Anwendung der DGLs

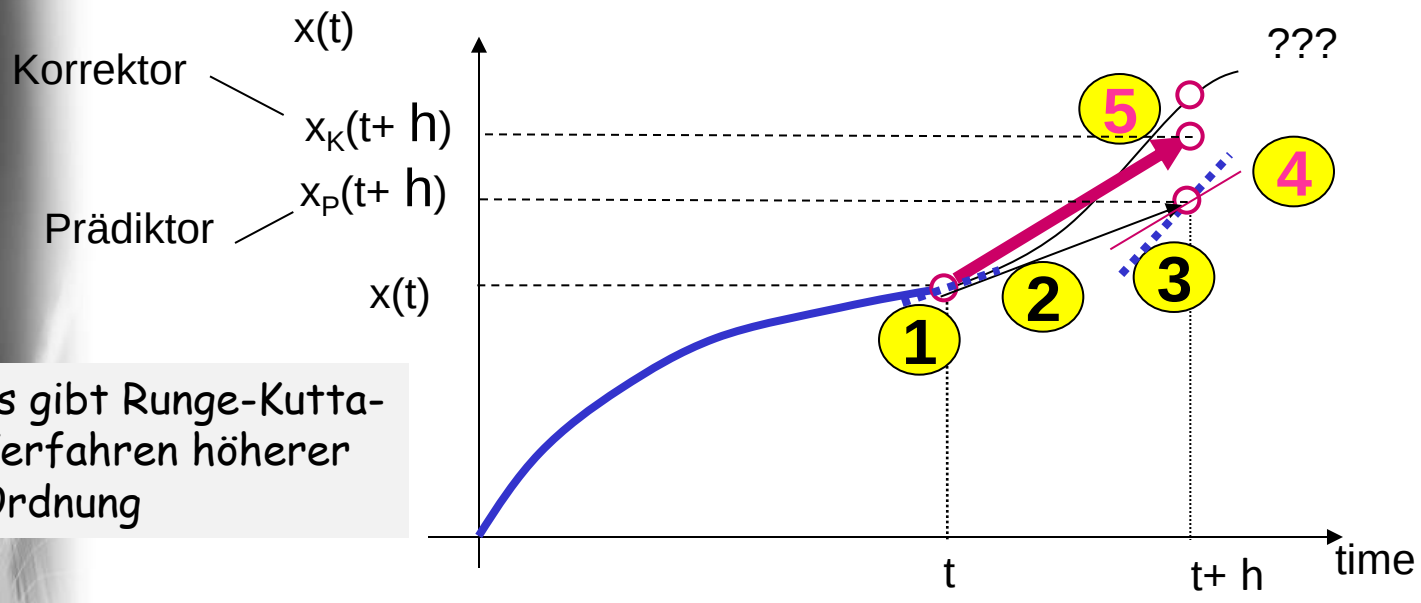
  // x´(t)= (size‘, damage‘) (t+h) ➔ r2

- **Korrektor-Schritt:**

  abermalige Berechnung der neuen Werte zum Zeitpunkt t+h

  x(t+h)= x(t) + h/2* (r1 + r2)

# Runge-Kutta- Integrationsverfahren (2. Ordnung)

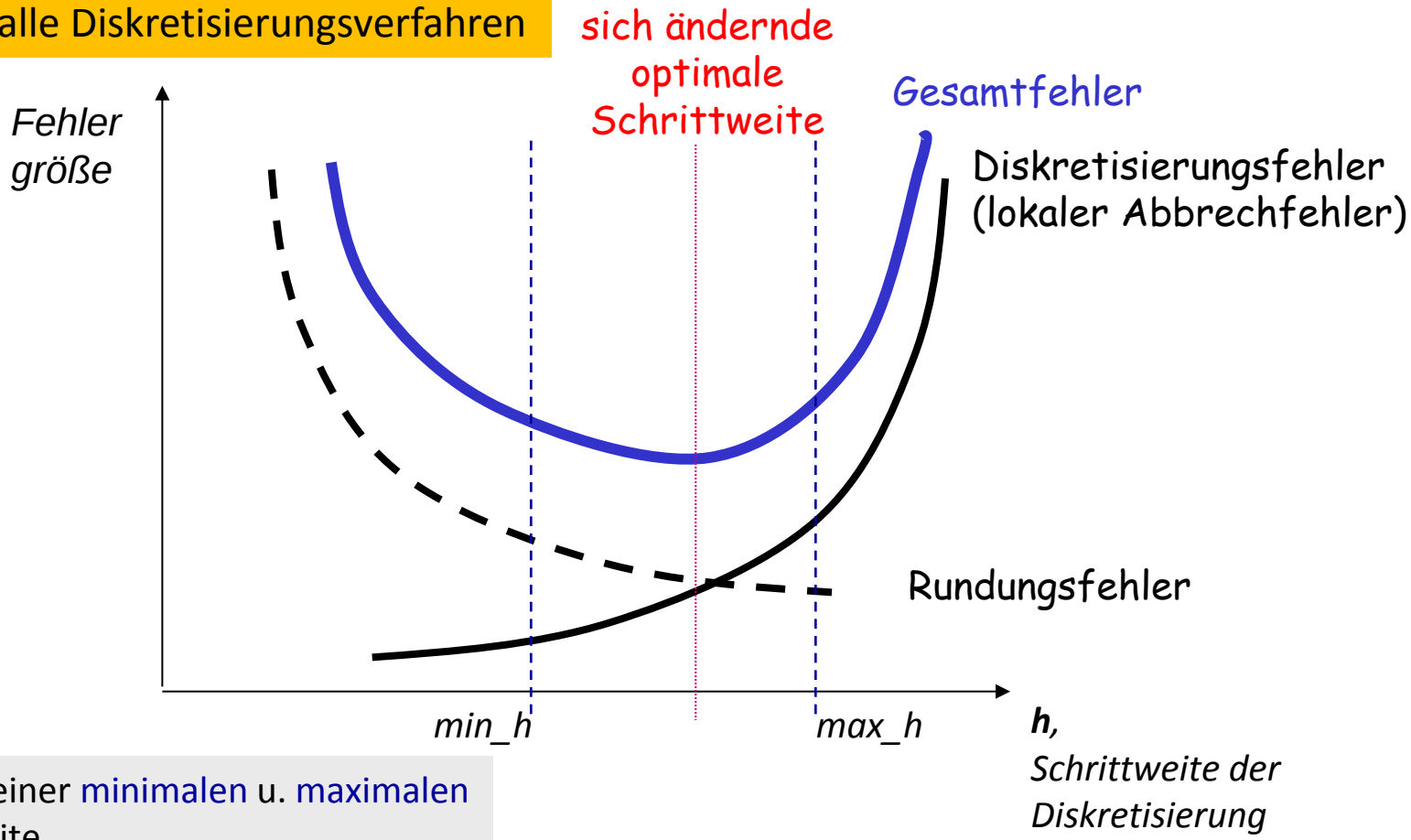



es gibt Runge-Kutta-Verfahren höherer Ordnung

1. berechne x'(t), d.h. f(x(t), t)
2. berechne x(t+h) nach Euler-Vorwärts mit x'(t)
3. berechne x'(t+h), d.h. f(x(t+h))
4. **Prädiktorschritt**: bilde den Mittelwert von x'(t) und x'(t+h)
5. **Korrektorschritt**: wiederhole Berechnung von x(t+h) nach Euler-Vorwärts, diesmal aber mit dem Mittelwert der beiden Ableitungen

# Fehlerüberlagerung

... gilt für alle Diskretisierungsverfahren




Vorgabe einer minimalen u. maximalen Schrittweite
bei Verfahren, die ihre Schrittweite bei Diskretisierung dynamisch anpassen

# Zeitkontinuierliche *Zustandsänderungen in ODEMx*

Einführung einer aktiven Klasse Continuous

- verwaltet „kontinuierliche“ **Variablen** anderer Prozesse (Referenzen auf Attribute)

  hier: die Variablen size, damage
  eines jeden Hausbrand-Objektes

  - verwaltet „beschreibende“ **DGLs**
    hier: die Berechnungsvorschriften für jede Brandentwicklung
    (als Zeiger auf eine Member-Funktion)

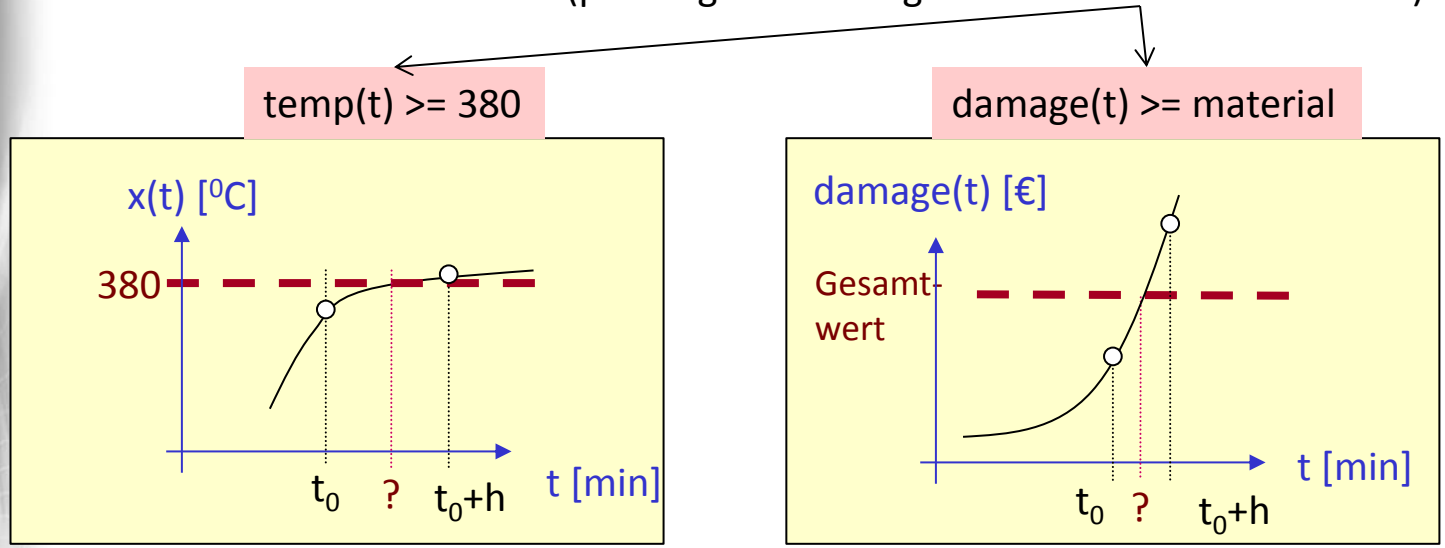
  - numerisches **Integrationsverfahren**,
    das zu einem Zeitpunkt t mit
    einer vorgegebenen Schrittweite h aus den aktuellen Werten der kontinuierlichen Variablen (unter Nutzung der DGLs)

    ihre Werte zum Zeitpunkt t+h ermittelt und
    sich mit holdFor(h) verzögert.

- … bei Bewältigung einer Reihe von Grundproblemen
  - numerische Genaugikeit/ Berechnungsgeschwindigkeit
    ( ~numerische Schrittweite und von Wahl des Integrationsverfahrens
    (dynamisch änderbar)
  - Synchronisation mit anderen zeitdiskreten und zeitkontinuierlichen Prozessen
  - Genauigkeit bei der Bestimmung von Zustandsereignissen
  - dynamische Änderung der Verhaltensbeschreibung

# Zustandsereignisse zeitkontinuierlicher Abläufe

- **Überwachung**
  jedes vollzogenen (numerisch akzeptierten) Integrationsschrittes,
  bei Überprüfung einer zugeordneten **Zustandsbedingung**
  (per Zeiger einer zugeordneten Memberfunktion)




- Bei Eintritt der Bedingung wird versucht, den Zeitpunkt des Eintritts der Bedingung genauer zu bestimmen:
  - **Interpolation** (nur begrenzt anwendbar)
  - **Intervallschachtelung** bei Halbierung der Integrationsschrittweite
    und Abbruch der numerischen Integration,
    sobald man eine untere vorgegeb Schranke für die Schrittweite (**min_h**) erreicht hat

# 8. Ausblick:

***Behandlung zeitkontinuierlicher Zustandsänderungen***

- Beispiel: Feuerwehreinsatz
- Konzept für die zeitkontinuierliche Simulation
- Gleichgewicht, Stabilität, Chaotische Systeme

# Gleichgewichtspunkte

- (oder stationärer Punkte, Fixpunkte) im Zustandsraum
  sind natürliche Ruhepunkte eines Systems
  ihre Kenntnis ist für die Beurteilung des Systemverhaltens wichtig

- an diesen Stellen verschwinden die Ableitungen der Zustandsgrößen nach der Zeit $\frac{dx}{dt} \rightarrow 0$

- nichtlineare Systeme können mehrere Gleichgewichtspunkte besitzen, welcher angenommen wird, hängt häufig von Anfangswerten der Zustandsgrößen ab

- besonders interessant ist die jeweilige Bewegung des Systems (Zustandsbahnen) in der Umgebung seiner Gleichgewichtspunkte

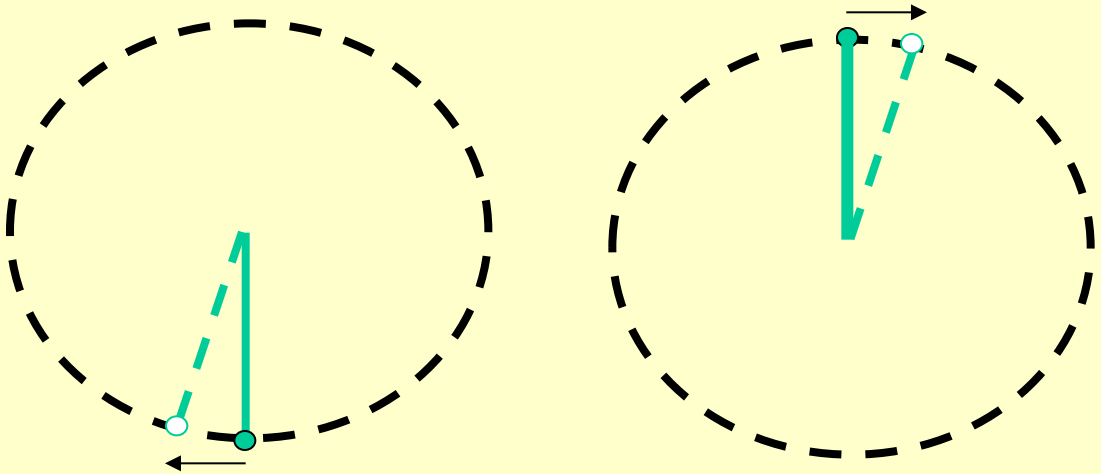
Stabilität von Gleichgewichtspunkten

# Stabile und instabile Gleichgewichtspunkte

**Entscheidungskriterium :** Welches Verhalten ergibt sich bei kleinster Bewegung aus dem Gleichgewichtspunkt ?

**stabil**: Rückkehr in den Gleichgewichtspunkt
**instabil**: Verlassen des Gleichgewichtspunktes **(meist Übergang in anderen Gleichgewichtspunkt)**

## Pendel-Beispiel



unterer GP ist stabil, der obere dagegen instabil

Es gibt mathematische Verfahren zur Bestimmung der Qualität von Gleichgewichtspunkten

# Ziele der Systemuntersuchung

**interessantes Charakteristikum des Systemverhaltens**
stationäre / eingeschwungene Zustände

**Bewertung des Systemverhaltens:**
Stabilität stationärer / eingeschwungener Zustände
Identifikation von Grenzbereichen

**Bewertung des Systemverhaltens:**
lineare – nichtlineare (chaotische und nichtchaotische) Systeme

**Typisches Untersuchungsziel**:
Sensitivität von Parametern auf das Systemverhalten
(**z.B.** stellt man häufig chaotisches Verhalten nur bei bestimmten Parameterkonstellationen fest)

Analytische Verfahren  Experimentelle Verfahren

# Chaotische Systeme

- bei den meisten realen Systemen bleiben benachbarte Zustandsbahnen im Laufe der Zeit nahe beieinander.

➔ Systeme sind deshalb vorhersagbar

- aus den Anfangswerten lässt sich die zukünftige Entwicklung ermitteln
- Entwicklung ist gegenüber kleinen Messfehlern der Anfangswerte nicht sehr empfindlich

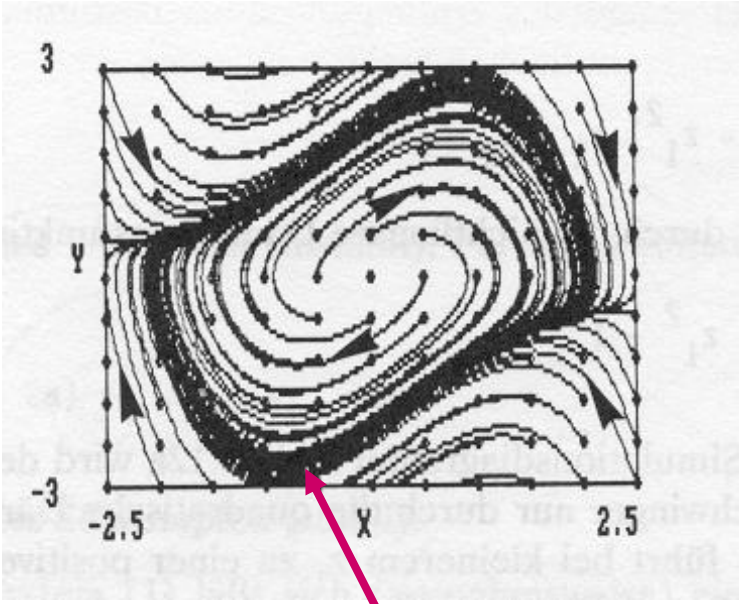
- Verhaltensbereich chaotischer Systeme zerfällt zwar in Regionen, in denen sich der Systemzustand nach einer gewissen Zeit befinden muss, ohne dass jedoch sein exakter Ort mit Sicherheit vorhergesagt werden kann
  - benachbarte Zustandsbahnen streben exponentiell auseinander, verlassen aber ihre Region nicht
  - Endzustand liegt irgendwo auf dem Regionen-Rand

➔ chaotische Systeme sind deshalb nicht vorhersagbar

# Attraktoren als Grenzlinie

- teilen den Zustandsraum in Regionen mit unterschiedlichem Verhalten
- **weiteres Beispiel**: van der Pol Oszillator

Verlauf mit a= -1.0




$$z'_1 = z_2$$
$$z'_2 = a\, z_1 + (1 - z_1^2) z_2$$

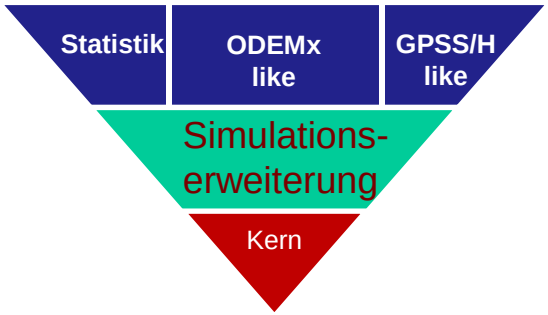
$$z'_1 = z_2$$
$$z'_2 = (1 - z_1^2) z_2 - z_1$$

# 8. Ausblick:

## *Alternative Simulationssprachen (SLX, GPSS, SDL, … )*

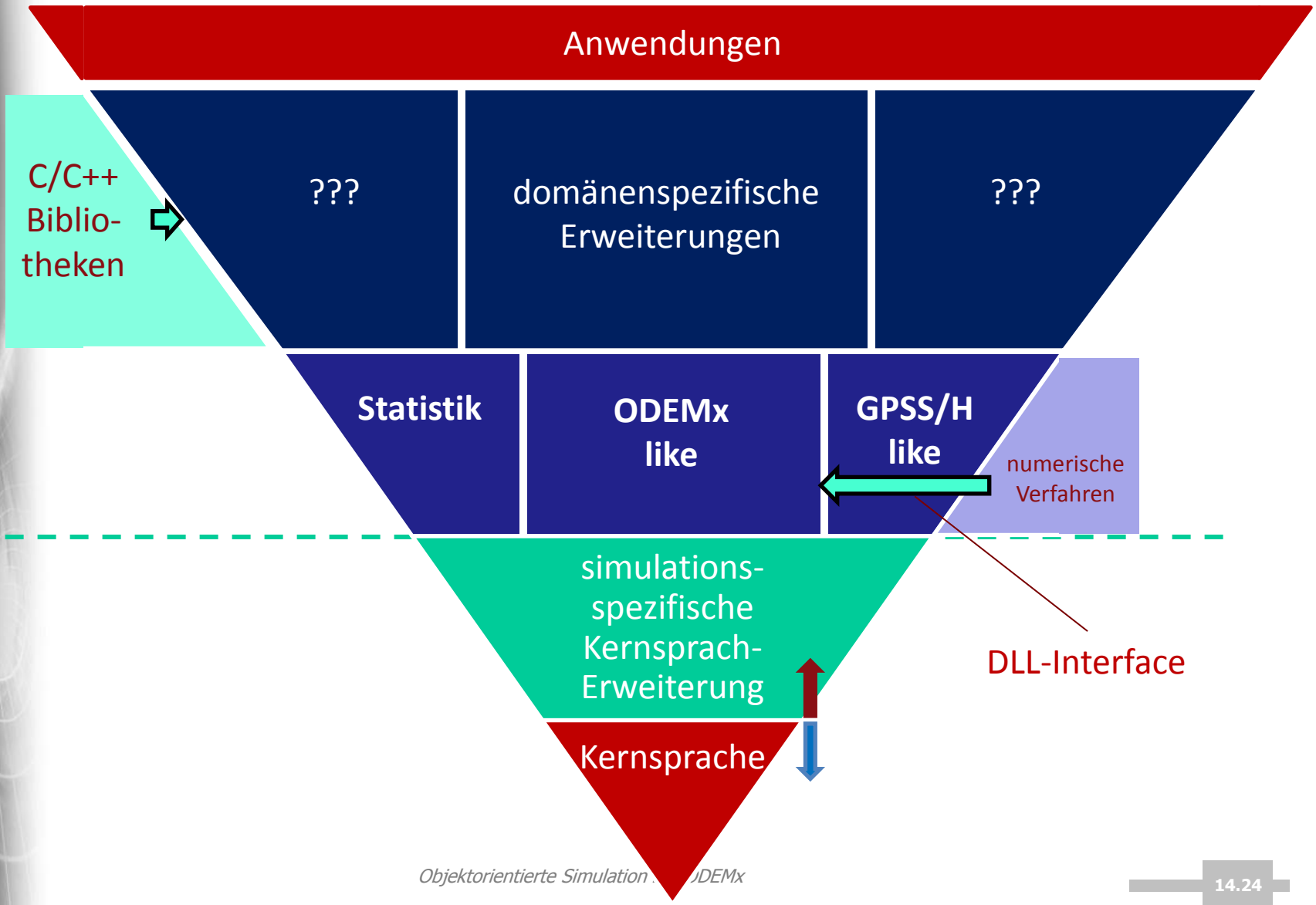
**Hintergrund**: laufende Projekte

- SLX- Deutsche Bahn AG
- SLX- Workflows, SLX-Logistik
- Verkehrsdetektion
- ODEMx Verbesserung
- DSL-Sprachentwicklung




Wir suchen Interessenten …
(mit Programmiererfahrung JAVA, C, C++)

# Die SLX-Modul-Pyramide



# SLX-Entwicklungsumgebung