# *Modul OMSI-2*
## *im SoSe 2011*

## *Objektorientierte Simulation mit ODEMx*

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

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

- Homepage
  http://www.informatik.hu-berlin.de/sam/lehre

- Vorlesungen
  Mo: 9.15 Uhr RUD 25 IV.113
  Mi: 9.15 Uhr RUD 25 IV.113

- Praktikum *(Beginn 18.04.2011)*
  Mo: 11.00 Uhr RUD 25 IV.113

➔ Praktikumsschein, mündliche Prüfung

# Ziele von OMSI-1 und OMSI-2

C++ -Konzepte

ODEMx-Konzepte (Version 2.x)

Vorzüge

Grenzen

OO-Modellierung von Systemen
- zeitkontinuierlich (GDGL nur angedeutet)
- zeitdiskret (Prozesse, Ereignisse)

OMSI-1

GDGL

Monitor für Continuous

Zelluläre Automaten

Logging/RDB

Experiment Management

Protokoll-Simulation

ODEMx-Konzepte (Version 2.x)

Master/Slave für Continuous

ODEMx-Konzepte (Version 3.x)

OMSI-2

OO-Modellierung Ringwalzwerk

GPSS

SDL/RT

Vorzüge

Grenzen

*Prinzip-Beispiele*

Entwicklung einer ausführbaren DSL mit Laufzeitsystem in ODEMx zur Simulation von Workflows

aktuelle Forschung

Netzwerksimulatoren

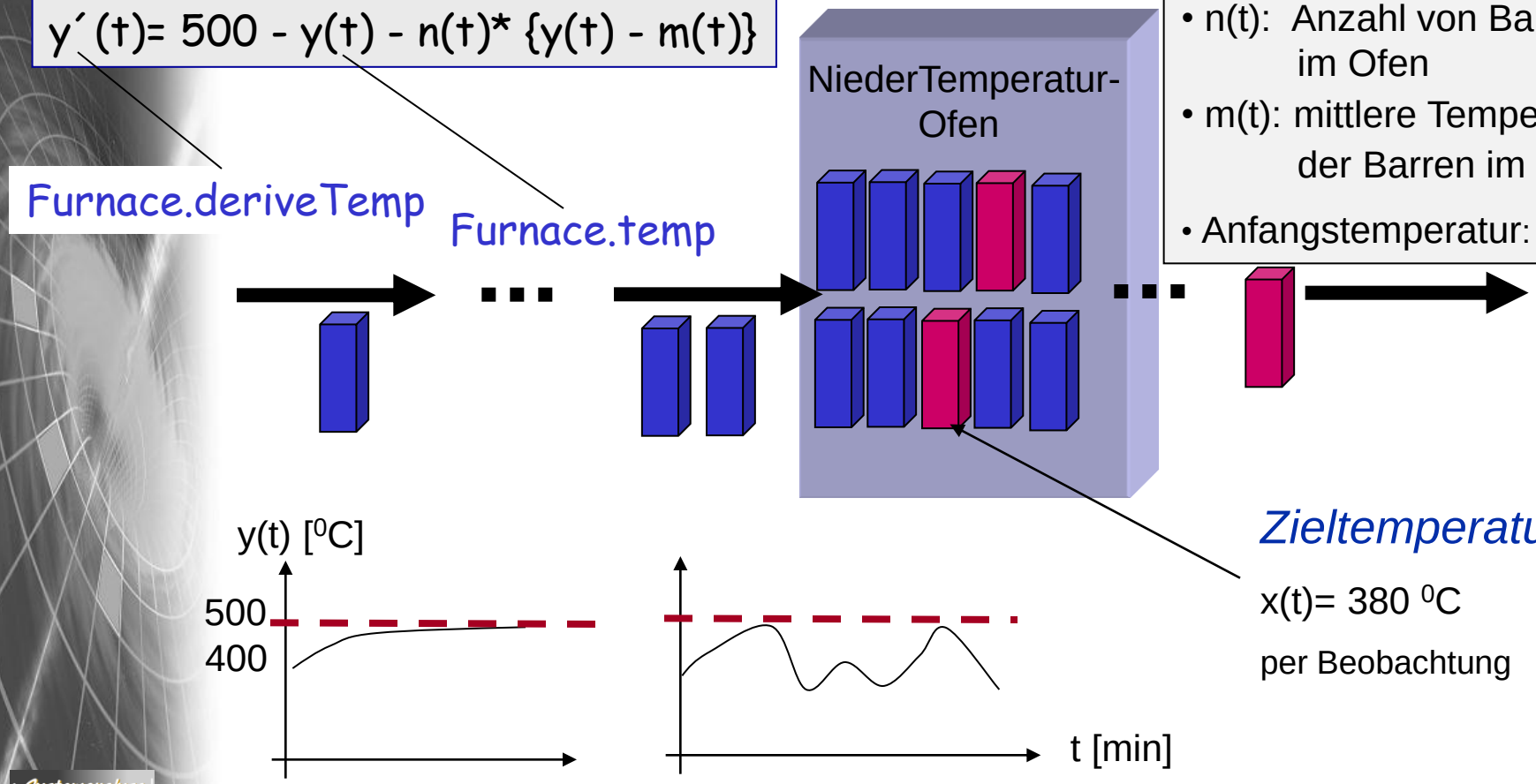
# Zur Erinnerung (OMSI-1)

Modellierung der Änderung der Ofentemperatur

Ofentemperatur y als y(t)

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

- Energiezufuhr: max. 500$^0$C
- n(t): Anzahl von Barren im Ofen
- m(t): mittlere Temperatur der Barren im Ofen
- Anfangstemperatur: 400$^0$C



# 1. Behandlung zeitkontinuierlicher Prozesse

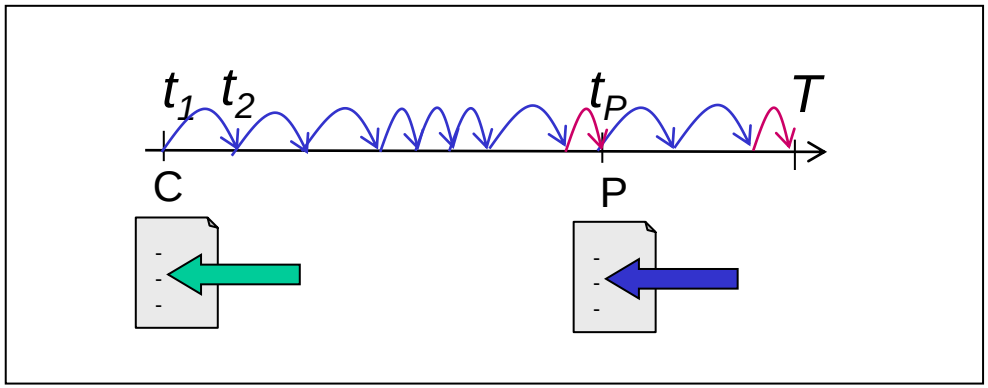
1. Scheduling von Process- und Continuous-Objekten
2. DGL, Richtungsfeld(Phasenraum), Trajektorie, Zustandsraum
3. Näherungsverfahren, Diskretisierungs/Verfahrensfehler
4. Runge-Kutta-Merson-Verfahren (ODEMx-Bibliothek)
5. Continuous-Implementierungskonzept
6. Ablauf am Beispiel (Barrentemperatur)
7. Synchronisationsprobleme und alternative Modellierungsmöglichkeiten in ODEMx

# Erste Vorstellung: Scheduling von Continuous-Objekten

$$t_0 \le t \le T$$

$$\vec{y}' = \vec{F}(\vec{y}, t)$$
$$\vec{y}(t_0) = \vec{y}_0 \quad \vec{y}_0 \in R^n$$

$$y' = f(y, t)$$
$$y(t_0) = y_0 \quad y_0 \in R^1$$




$t_P$: Steuerungswechsel in der Prozessausführung

## Continuous-Objekt

initiale Wertebelegung des State-Vektors

berechnet für jeden Diskretisierungsschritt $t_i$
- den Zustandsvektor $y(t_i)$ STATE mit Hilfe eines numer. Integrationsverfahrens (bei Berechnung aktueller Zustandsänderungen $y'(t_i)$: RATE) → *derivatives()*
- Umsortierung in der ExL um die Integrationsschrittweite (SW) nach jedem Integrationsschritt
- dynamische Variation der SW nach numerischen Kriterien
- dynamische Anpassung der SW an Ereigniszeitpunkte diskreter Prozesse $t_P$ und an das IntegrationsEnde $T$

# NextEvent-Prinzip des Simulators

bleibt auch für die Behandlung zeitkontinuierlicher Prozesse und damit im Mix mit zeitdiskreten Prozessen erhalten:

- Laden der initialen Prozesse in die ExL (mindestens einen)
- Starten des ersten Prozesses (kleinste Ereigniszeit, höchste Priorität)

  (Re-)Aktivierung seiner virtuellen Lebenslauf-Methode

  - Zustandsänderungen
    Zeitverbrauch (erneutes Scheduling mit evtl. Prozesswechsel)

  ODER

  - Blockierung (Entfernung aus der ExL, Prozesswechsel)

  ODER

  - Terminierung (Entfernung aus dem ExL, Prozesswechsel)

*diskret*

*kontinuierlich: integrate()*

**zyklisch**:
nächster Prozess
im Terminkalender
Next-Event-Prinzip

- Auswertung, Abbruch der Simulation

# 1. Behandlung zeitkontinuierlicher Prozesse

1. Scheduling von Process- und Continuous-Objekten
2. DGL, Richtungsfeld(Phasenraum), Trajektorie, Zustandsraum
3. Näherungsverfahren, Diskretisierungs/Verfahrensfehler
4. Runge-Kutta-Merson-Verfahren (ODEMx-Bibliothek)
5. Continuous-Implementierungskonzept
6. Ablauf am Beispiel (Barrentemperatur)
7. Synchronisationsprobleme und alternative Modellierungsmöglichkeiten in ODEMx

# Grundsätzliche Einteilung von Systemen und Modellen (Erinnerung)

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

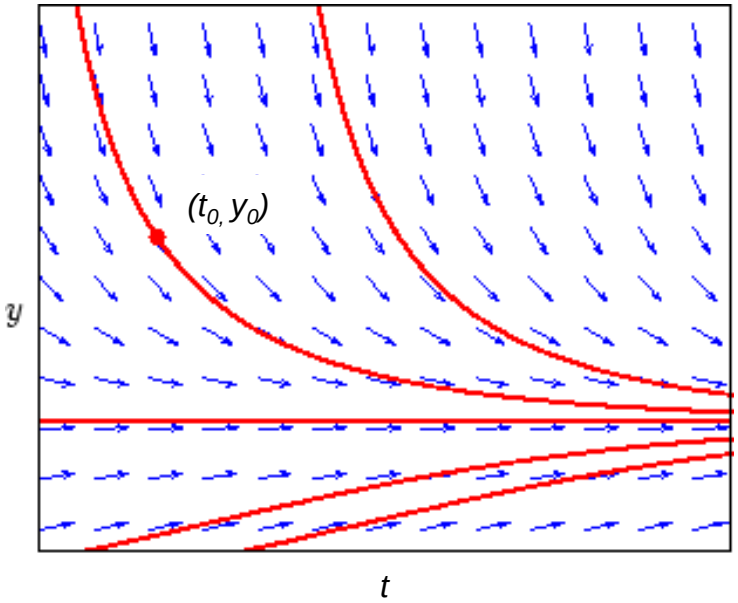
      modelliert als System
      von n Differentialgleichungen 1. Ordnung

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

kombinierte Systeme (Beispiel: Niedrigtemperaturofen)

# Lösung einer DGL

Das **Richtungsfeld** einer Differentialgleichung $\vec{y}' = \vec{F}(\vec{y}, t)$ ordnet jedem Punkt in der *(t, y)*-Ebene eine Tangente mit Steigung *y′* zu (wenn *y* 1-dimensonal ist).




Existenz und Eindeutigkeit einer AWA-Lösung ist bei stetiger Differenzierbarkeit der Funktion *F* gesichert.

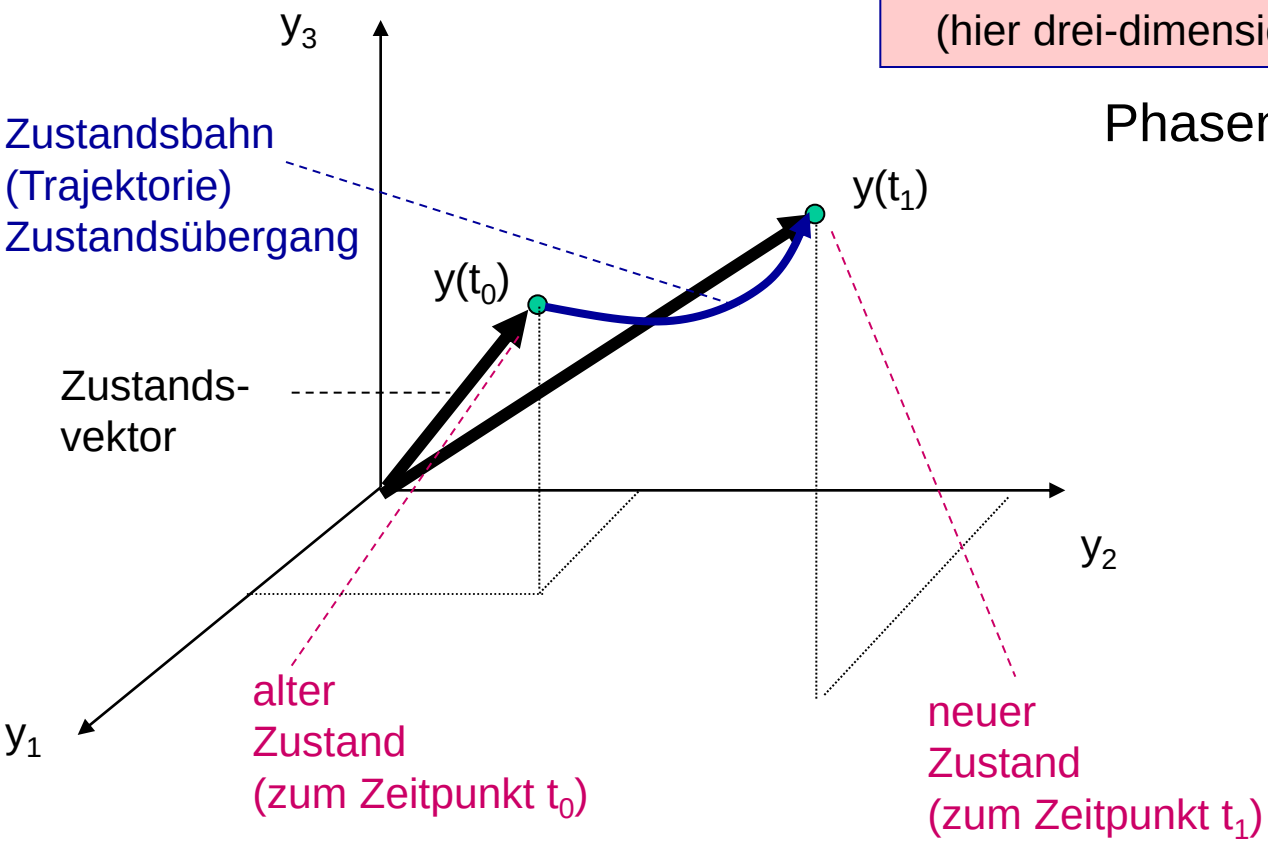
Die Aufgabe, eine Lösung *y(t)* für der Differentialgleichung mit $\vec{y}' = \vec{F}(\vec{y}, t)$ zu finden, heißt **Anfangswertaufgabe (AWA)**.

$$\vec{y}(t_0) = \vec{y}_0 \quad \vec{y}_0 \in R^n$$

# Zustandsraum, Zustandsbahnen

n Zustandsgrößen spannen n-dimensionalen Zustandsraum auf (hier drei-dimensional)

Phasenporträt



# 1. Behandlung zeitkontinuierlicher Prozesse

1. Scheduling von Process- und Continuous-Objekten
2. DGL, Richtungsfeld(Phasenraum), Trajektorie, Zustandsraum
3. Näherungsverfahren, Diskretisierungs/Verfahrensfehler
4. Runge-Kutta-Merson-Verfahren (ODEMx-Bibliothek)
5. Continuous-Implementierungskonzept
6. Ablauf am Beispiel (Barrentemperatur)
7. Synchronisationsprobleme und alternative Modellierungsmöglichkeiten in ODEMx

# Näherungsverfahren

betrachten der Einfachheit halber den 1-dimensionalen Fall

Ausgangspunkt: äquivalente Integralgleichung

$$y' = f(y,t)$$
$$y(t_0) = y_0$$

$$y(t_{k+1}) - y(t_k) = \int_{t_k}^{t_{k+1}} f(t, y(t))dt$$

$$y(t_{k+1}) = y(t_k) + \int_{t_k}^{t_{k+1}} f(t, y(t))dt$$

Einschrittverfahren

Näherungsverfahren unterschiedlicher Güte
(Fehlerordnung im Abgleich mit Taylor-Reihenentwicklung)

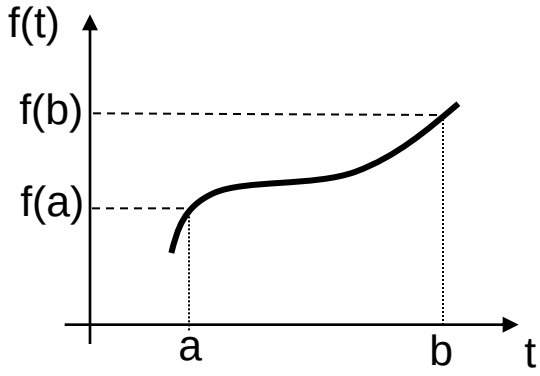
$$y' = f(y,t)$$

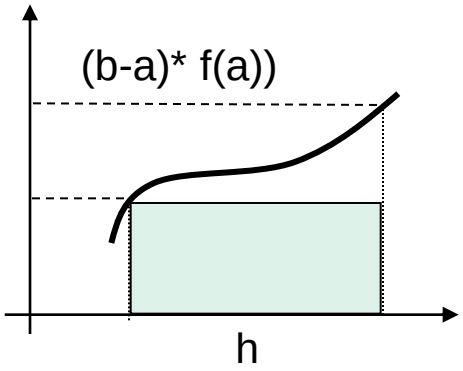
Prinzip:
- Intervall $[t_{k+1}, t_k]$ wird weiter unterteilt
  Ableitungsberechnung auch zu den Zwischenzeitpunkten innerhalb eines Integrationsschrittes
- Prädiktor- und Korrektorschritte mit gewichteten Anteilen der zu den Zwischenzeiten berechneten Ableitungen

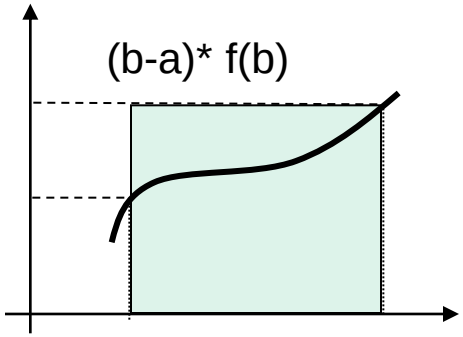
# Numerische Integrationsverfahren

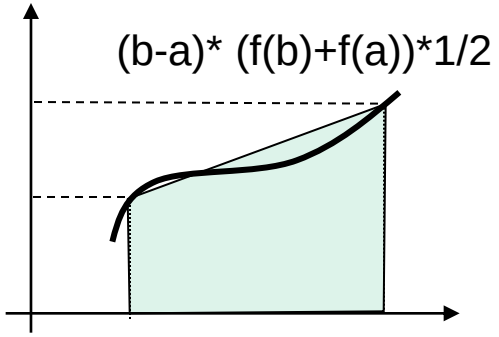
- Approximation des Integrals über $f$
  $a = t_0$, $b = T$

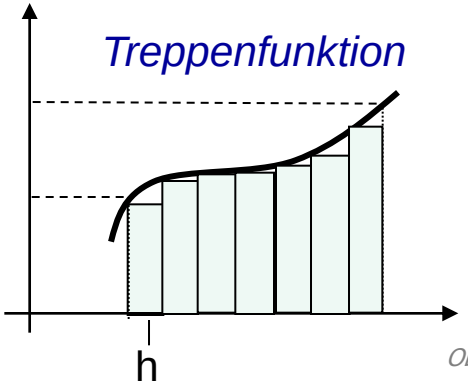
Wie groß ist der **Diskretisierungsfehler** ?
(auch Verfahrensfehler genannt)
Fehlerbetrachtungen, wenn h gegen Null geht

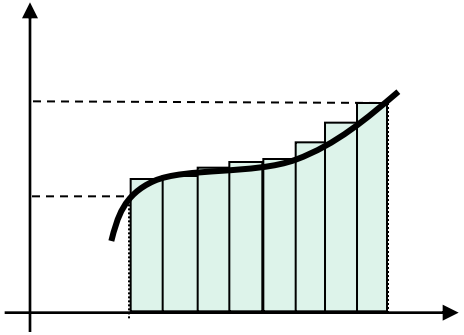


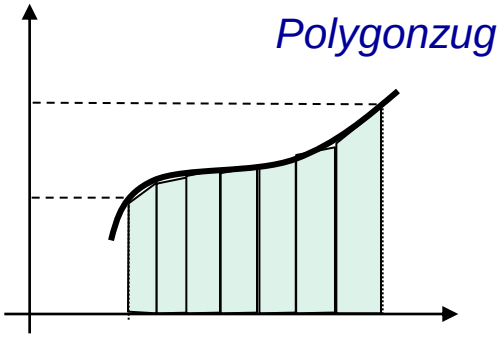



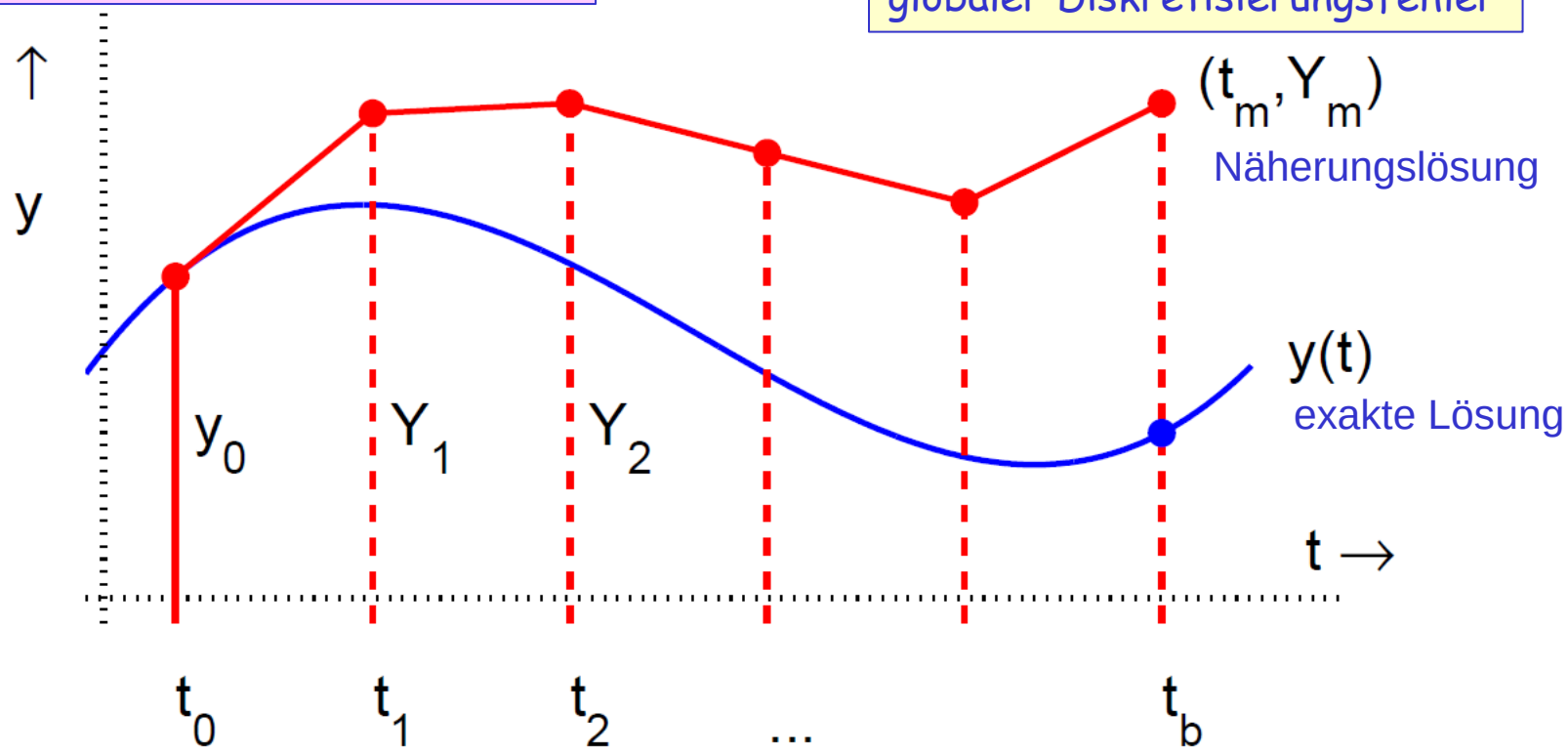


# Diskretisierungsfehler

Prinzipdarstellung

lokaler Diskretisierungsfehler
globaler Diskretisierungsfehler

betrachten jetzt y (nicht Funktion f)




$$y' = f(y,t) \qquad t_k = t_0 + kh \qquad (k = 1,2,\ldots)$$
$$y(t_0) = y_0 \qquad y_{k+1} = y_k + hf(y_k, t_k) \qquad (k = 1,2,\ldots)$$

# *Ordnung des Diskretisierungsfehlers (ein Gütekriterium des Verfahrens)*

Verfahren unterscheiden sich durch die Ordnung ihres Diskretisierungsfehlers

**Wie bestimmt man diese Ordnung?**

1. Differenzbildung von
   - Termen der exakten Funktion *f* (dargestellt als Taylor-Reihe) und
   - Termen der Approximation von *f*

   Dabei heben sich Terme kleiner Ordnungen gegenseitig auf

2. kleinste Ordnung der verbleibenden Terme in der Differenz bestimmt die **Ordnung des Fehlers**

weiteres Gütekriterium: **globale Fehler (schwieriger zu bestimmen)**

**FRAGE:**

wie unterscheiden sich exakter Verlauf und approximierter Verlauf nach einer Reihe von Diskretisierungsschritten (Fortpflanzung des lokalen Fehlers) ?

# Abschätzung des lokalen Diskretisierungsfehlers

$$y(t_{k+1}) = y(t_k) + \int_{t_k}^{t_{k+1}} f(t, y(t))dt$$

Berechnung mit Verfahren der Fehlerordnung N+1

$y(t_{k+1})^{N+1}$

Berechnung mit Verfahren der Fehlerordnung N

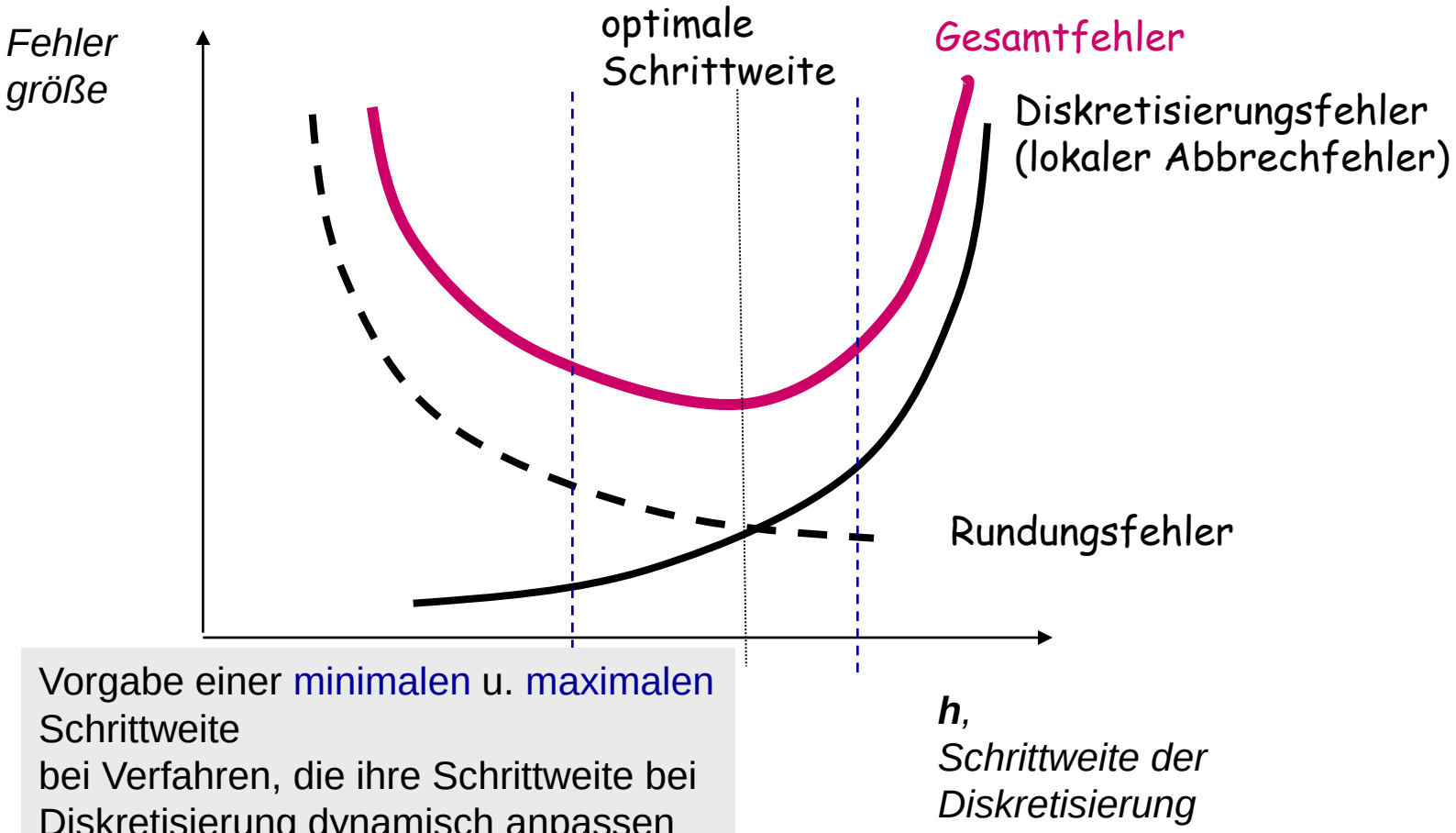
$y(t_{k+1})^{N}$

Abschätzung als:
- Differenz der Lösungen (nach Verfahren unterschiedlicher Ordnung) zum jeweils aktuellen diskretisierten Zeitpunkt

Prinzip einer üblichen Integrationsschrittweitensteuerung:
- berechneter Diskretisierungsfehler zu groß: Wiederholung des letzten Integrationsschrittes mit halbierter Schrittweite
- berechneter Diskretisierungsfehler sehr klein: Verdopplung der Schrittweiten beim nächsten Schritt

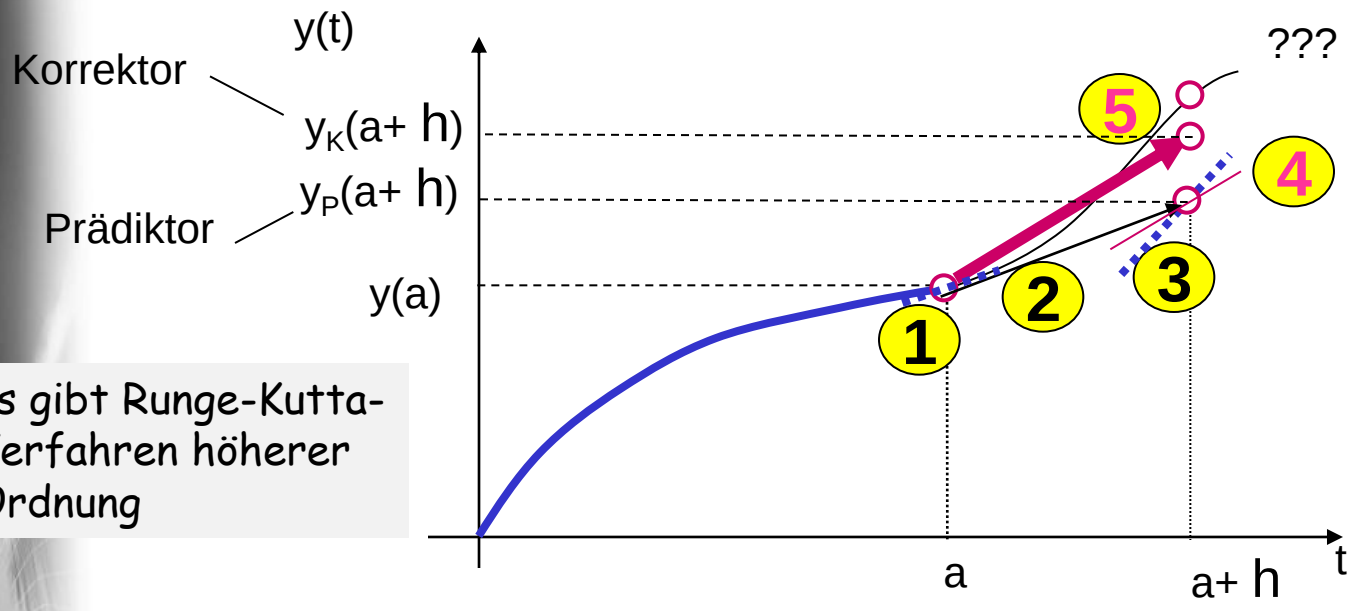
# Fehlerüberlagerung

... gilt für alle Diskretisierungsverfahren




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

# Runge-Kutta- Integrationsverfahren (2. Ordnung)




es gibt Runge-Kutta-Verfahren höherer Ordnung

1. berechne y'(a), d.h. f(y(a), a)
2. berechne y(a+h) nach Euler-Vorwärts mit y'(a)
3. berechne y'(a+h), d.h. f(y(a+h))
4. Prädiktorschritt: bilde den Mittelwert von y'(a) und y'(a+h)
5. Korrektorschritt: wiederhole Berechnung von y(a+h) nach Euler-Vorwärts, diesmal aber mit dem Mittelwert der beiden Ableitungen

# 1. Behandlung zeitkontinuierlicher Prozesse

1. Scheduling von Process- und Continuous-Objekten
2. DGL, Richtungsfeld(Phasenraum), Trajektorie, Zustandsraum
3. Näherungsverfahren, Diskretisierungs/Verfahrensfehler
4. Runge-Kutta-Merson-Verfahren (ODEMx-Bibliothek)
5. Continuous-Implementierungskonzept
6. Ablauf am Beispiel (Barrentemperatur)
7. Synchronisationsprobleme und alternative Modellierungsmöglichkeiten in ODEMx

# Runge-Kutta-Merson (ODEMx-Verfahren)

berechnet
$y(t_{k+1})$ aus $y(t_k)$ in 5 Zwischenschritten:

$$y' = f(y,t)$$
$$y(t_0) = y_0$$

$$t_0 \le t \ge T$$

$$k_1 = hf(t_0, y_0)$$
$$k_2 = hf(t_0 + \frac{1}{3}h, y_0 + \frac{1}{3}k_1),$$
$$k_3 = hf(t_0 + \frac{1}{3}h, y_0 + \frac{1}{6}k_1 + \frac{1}{6}k_2),$$
$$k_4 = hf(t_0 + \frac{1}{2}h, y_0 + \frac{1}{8}k_1 + \frac{3}{8}k_2),$$
$$k_5 = hf(t_0 + h, y_0 + \frac{1}{2}k_1 - \frac{3}{2}k_3 + 2k_4),$$
$$y^1(t_0 + h) = y_0 + \frac{1}{2}k_1 - \frac{3}{2}k_3 + 2k_4),$$
$$y^2(t_0 + h) = y_0 + \frac{1}{6}k_1 - \frac{2}{3}k_4 + \frac{1}{6}k_5),$$

lokaler Diskretisierungsfehler

$$R = 0.2\left|y^1 - y^2\right|$$

Norm des Differenzvektors

falls R > ε, dann h:= h/2
falls R <= ε, dann h:= h*2

ε vorgegebene Schranke

Verfahren der Ordnung 4

Verfahren der Ordnung 5

# *Runge-Kutta-Merson (ODEMx-Verfahren)*

## Charakteristik

- implementiert als virtuelle Methode takeAStep (double h)
  d.h.: vorab-installiertes Verfahren, das ersetzt werden kann
- Ein-Schritt-Verfahren
- Verfahren 4.Ordnung

SW=Schrittweite

- automatische SW-Korrektur (erfolgt über integrate!)
  bei Vorgabe
  (a) des maximalen Diskretisierungsfehlers $\varepsilon$
  (b) und SW-Grenzen (minh, maxh)
- Fehlerabbruch,
  falls Diskretisierungsfehler auch bei minmaler SW auftritt

# 1. Behandlung zeitkontinuierlicher Prozesse

1. Scheduling von Process- und Continuous-Objekten
2. DGL, Richtungsfeld(Phasenraum), Trajektorie, Zustandsraum
3. Näherungsverfahren, Diskretisierungs/Verfahrensfehler
4. Runge-Kutta-Merson-Verfahren (ODEMx-Bibliothek)
5. Continuous-Implementierungskonzept
6. Ablauf am Beispiel (Barrentemperatur)
7. Synchronisationsprobleme und alternative Modellierungsmöglichkeiten in ODEMx

# Continuous: die zentralen Funktionen

Member-Funktionen:

- **virtual int** main() = 0;

*ruft*

- **int** integrate (SimTime timeEvent, Condition stateEvent=0);

realisiert zeitkontinuierliche Verhaltensphase (mehrere möglich):

Ausführung einer Folge von Integrationsschritten

nläufig stattfindenden Abläufen

*ruft sukzessive RKM-Verfahren mit aktualisierter SW*

- **virtual void** takeAStep (**double** h);
- **virtual void** derivatives (**double** t) = 0;

*ruft für jeden Zwischenzeitpunkt (5-mal) s. 10.25*

- **virtual double** errorNorm();

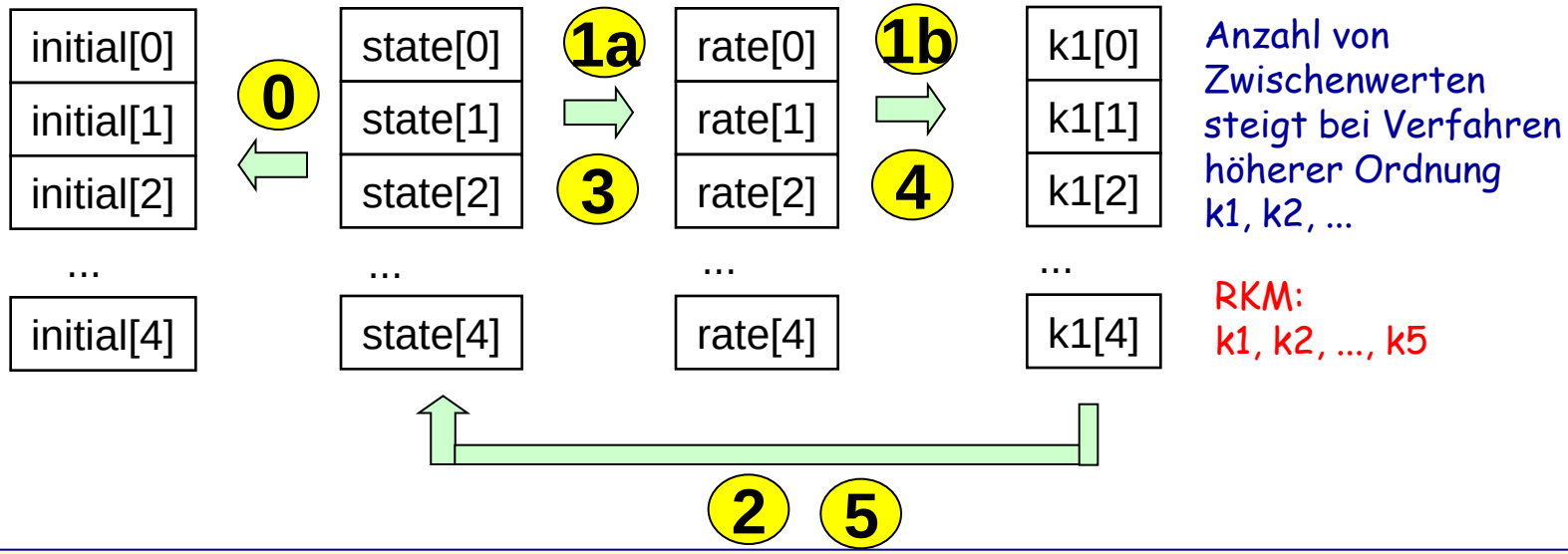
Maximumnorm, Euklidische Norm

$$R = 0.2\left|y^1 - y^2\right|$$

*nach jedem takeAStep-Ruf*

# Continuous-Implementationsschema

- Datenstruktur:
  Vektoren der entsprechenden Dimension des Zustandsraumes

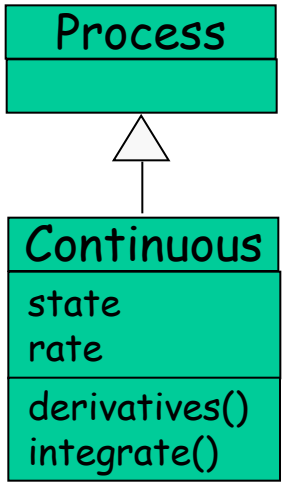



0. retten der Anfangswerte im initial-Vektor
1a. rate-Vektor speichert die aktuellen Ableitungswerte f(state[], t)
1b. retten der aktuellen Ableitungen (rate) im k1-Vektor
2. Berechnung des neuen state-Vektors (mit k1) nach Euler-Vorwärts
3. rate-Vektor speichert die aktuellen Ableitungswerte f (state[], t+h)
4. Mittelwert von rate und k1 mit Speicherung in k1
5. Wiederholung Euler-Vorwärts mit diesem Mittelwert der Ableitungen: state[]
6. ...

# 1. Behandlung zeitkontinuierlicher Prozesse

1. Scheduling von Process- und Continuous-Objekten
2. DGL, Richtungsfeld(Phasenraum), Trajektorie, Zustandsraum
3. Näherungsverfahren, Diskretisierungs/Verfahrensfehler
4. Runge-Kutta-Merson-Verfahren (ODEMx-Bibliothek)
5. Continuous-Implementierungskonzept
6. Ablauf am Beispiel (Barrentemperatur)
7. Synchronisationsprobleme und alternative Modellierungsmöglichkeiten in ODEMx

# Ablauf am Beispiel: Barrentemperatur




jeder kontinuierliche Prozess (Continuous) verfügt über

1. Zustandsattribut state (n-dim. Vector) ~ Temperatur
2. erste Ableitung des Zustandsattributs rate (n-dim. Vector)

   Werte zum aktuellen Zeitpunkt

3. *Pure Virtual Function* dervatives (double t) zur Codierung der Funktion *f* zur Berechnung von rate

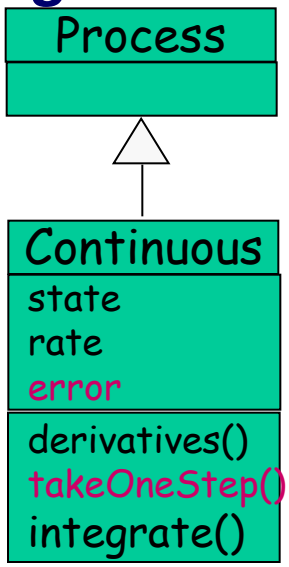
Temperatur x eines Barrens als x(t)

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

```
class Ingot : public Continuous {
        ...
        Ingot ();
        void derivatives (double);
        int main();
}
```

```
void Ingot::dervatives (double t) {
        rate[0] = { u() – state[0] } / 7
}
```

# Zur Umsetzung der numerischen Berechnung

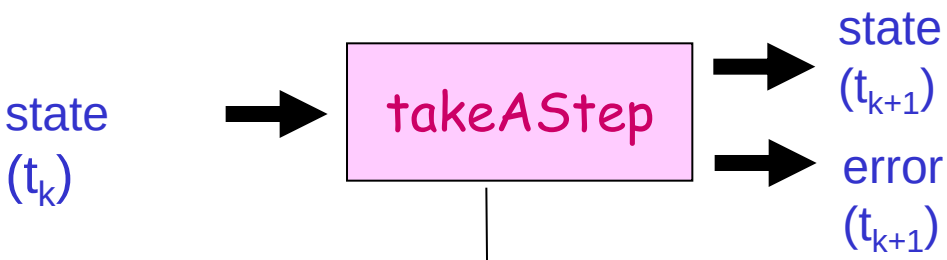



...

4. virtuelle Integrationsmethode: hier RKM

   void takeOneStep(double h)

   führt einen **einzelnen** Integrationsschritt aus




bei Berechnung von rate
zu verschiedenen Zwischenzeitpunkten

```
void dervatives (double t) {
        rate[0] = { u() – state[0] } / 7
}
```

*zusätzliches Feld der Dim. n*

durch Aufruf von derivatives
und Verarbeitung der zwischenzeitlichen
rate-Werte

# Zur Umsetzung der numerischen Berechnung

…

5. Methode zur globalen Steuerung der numerischen Integration:

`int integrate (SimTime timeEvent, Condition stateEvent=0)`

- startet die numerische Integration zum aktuellen Zeitpunkt ausgehend vom initialen `state`-Wert
- benutzt eine vorgegebene Schrittweite `h`: Bereichsangabe ($10^{-4}$ .. $10^{-2}$)
- benutzt eine vorgegebene Fehlerschranke `errorLimit`: Wert aus ($10^{-5}$ .. $10^{-3}$)
- benutzt einer vorgegebene virtuelle Methode einer Vektornorm `double errorNorm()` (Euklidische Norm, Maximum-Norm)

`initialState` *zusätzliches Feld der Dim. n*

➔ d.h. führt einen Integrationschritt mit SW `h` aus

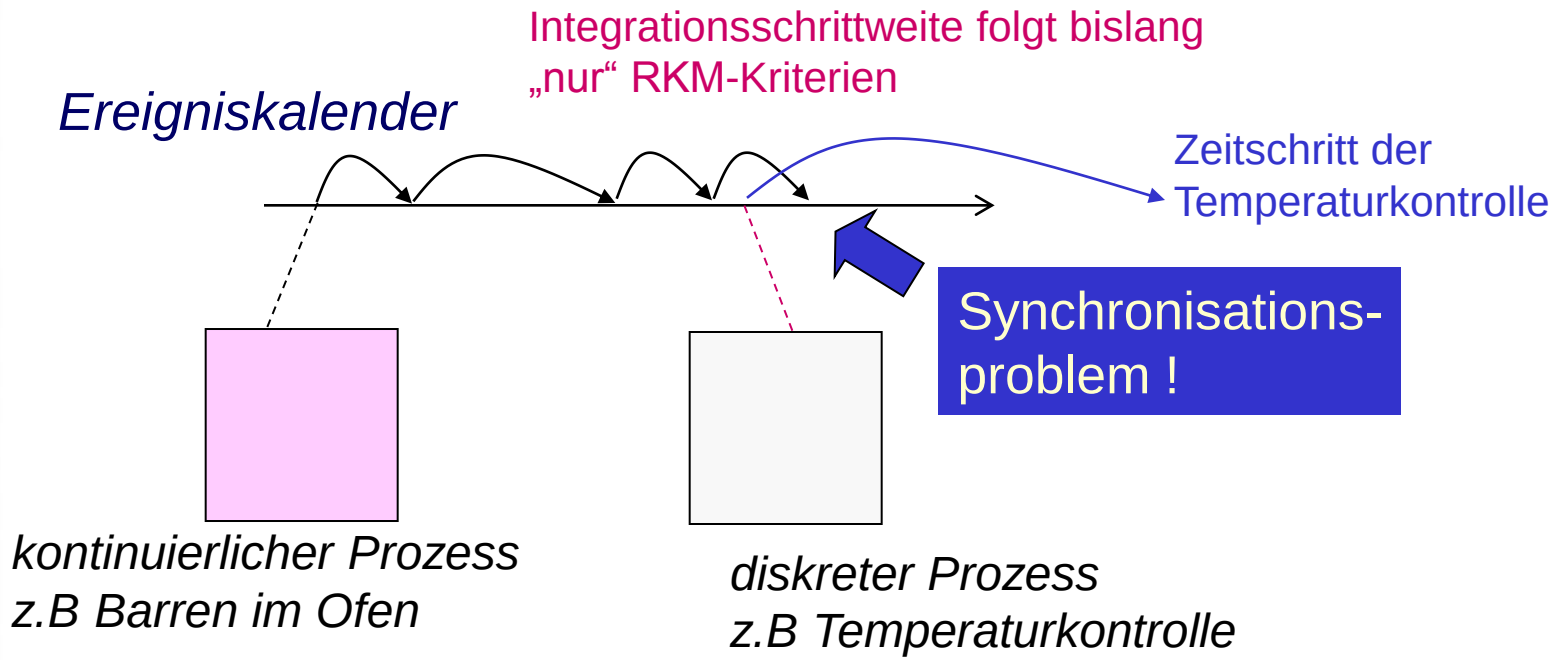
- berechnet die Norm vom `error`-Vektor → double-Wert
- vergleicht diesen mit vorgeg. errorLimit
  **Fall wesentlich kleiner**: nächster Integrationsschritt mit doppelter Schrittweite h
- **Fall größer**: Wiederholung des Integrationschrittes mit halbierter Schrittweite (vorher: `state= initialState`)
- **Fall sonst:** nächster Integrationsschritt mit unveränderter Schrittweite h

# Zur Umsetzung der numerischen Berechnung

…

6. zusätzliche Nebenbedingungen zur Steuerung der numerischen Schrittweite:

```
int integrate (SimTime timeEvent, Condition stateEvent=0)
```



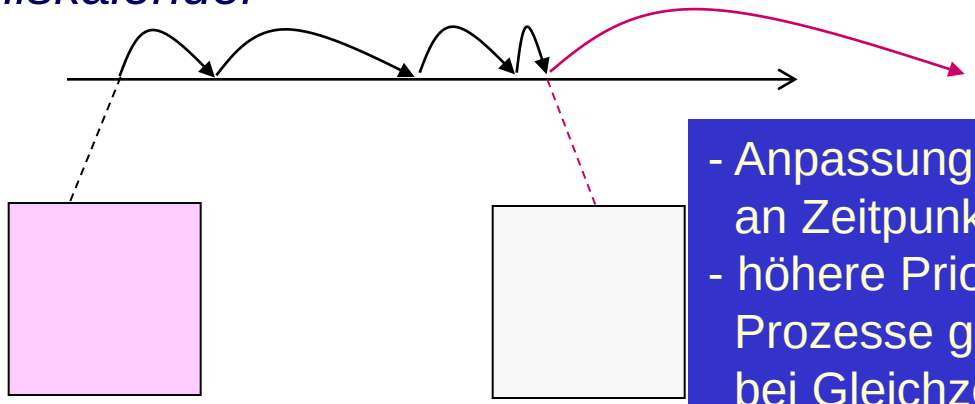
# Zur Umsetzung der numerischen Berechnung

…

6. zusätzliche Nebenbedingungen zur Steuerung der numerischen Schrittweite:

```
int integrate (SimTime timeEvent, Condition stateEvent=0)
```

*Ereigniskalender*



Problemlösung

- Anpassung der Schrittweite an Zeitpunkte diskreter Prozesse
- höhere Priorität diskreter Prozesse gegenüber kont. bei Gleichzeitigkeit

*kontinuierlicher Prozess*
*z.B Barren im Ofen*

*diskreter Prozess*
*z.B Temperaturkontrolle*