Kurs OMSI
im WiSe 2011/12

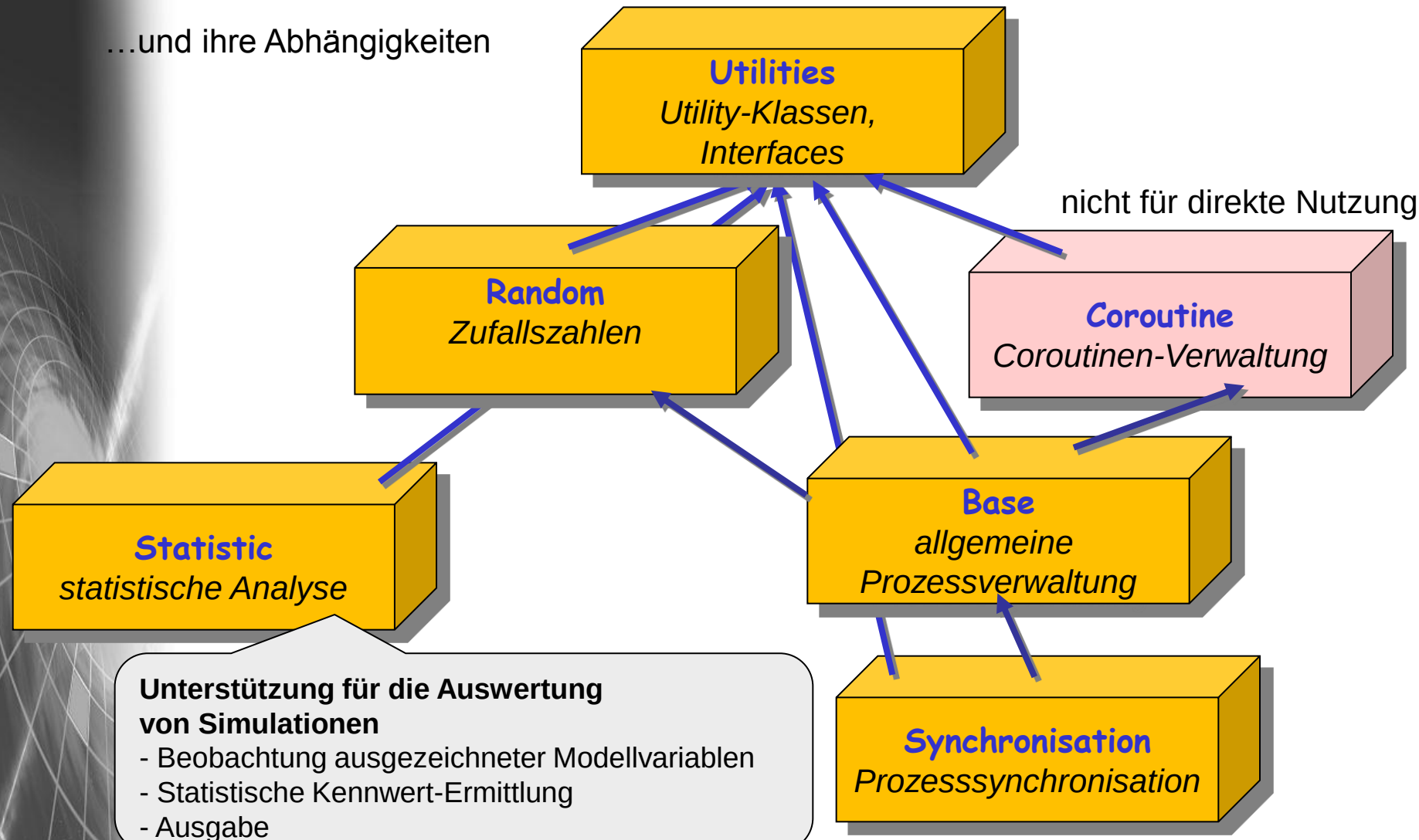
Objektorientierte Simulation mit ODEMx

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

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

Die ODEMx-Module

…und ihre Abhängigkeiten



Unterstützung für die Auswertung von Simulationen
- Beobachtung ausgezeichneter Modellvariablen
- Statistische Kennwert-Ermittlung
- Ausgabe

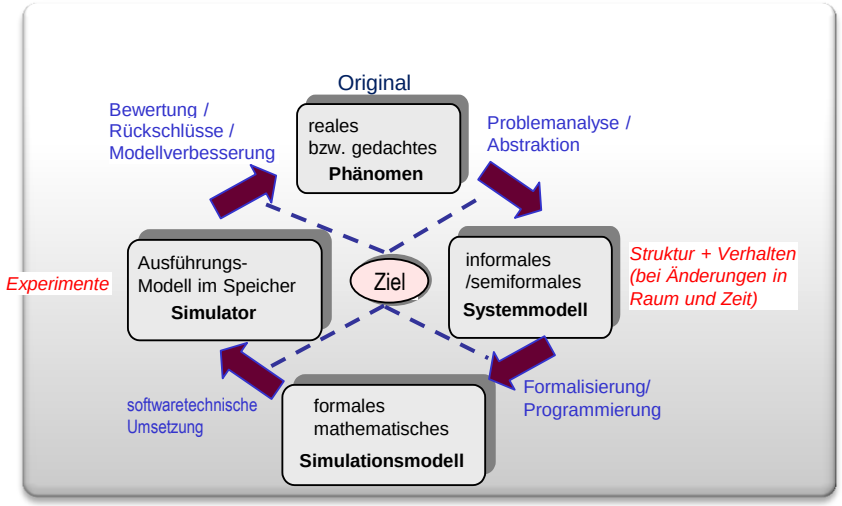
7. ODEMx-Modul Statistik:

Count, Tally, Accum, Histo, Regress

- Motivation und Konzept
- Allgemeine operationelle Schnittstelle (Tab)
- Zähler (Count, Sum)
- Ungewichtete Modelldaten (Tally, Histo)
- Gewichtete Modelldaten (Accum)
- Lineare Regression

Modellauswertung

Typischer Ablauf

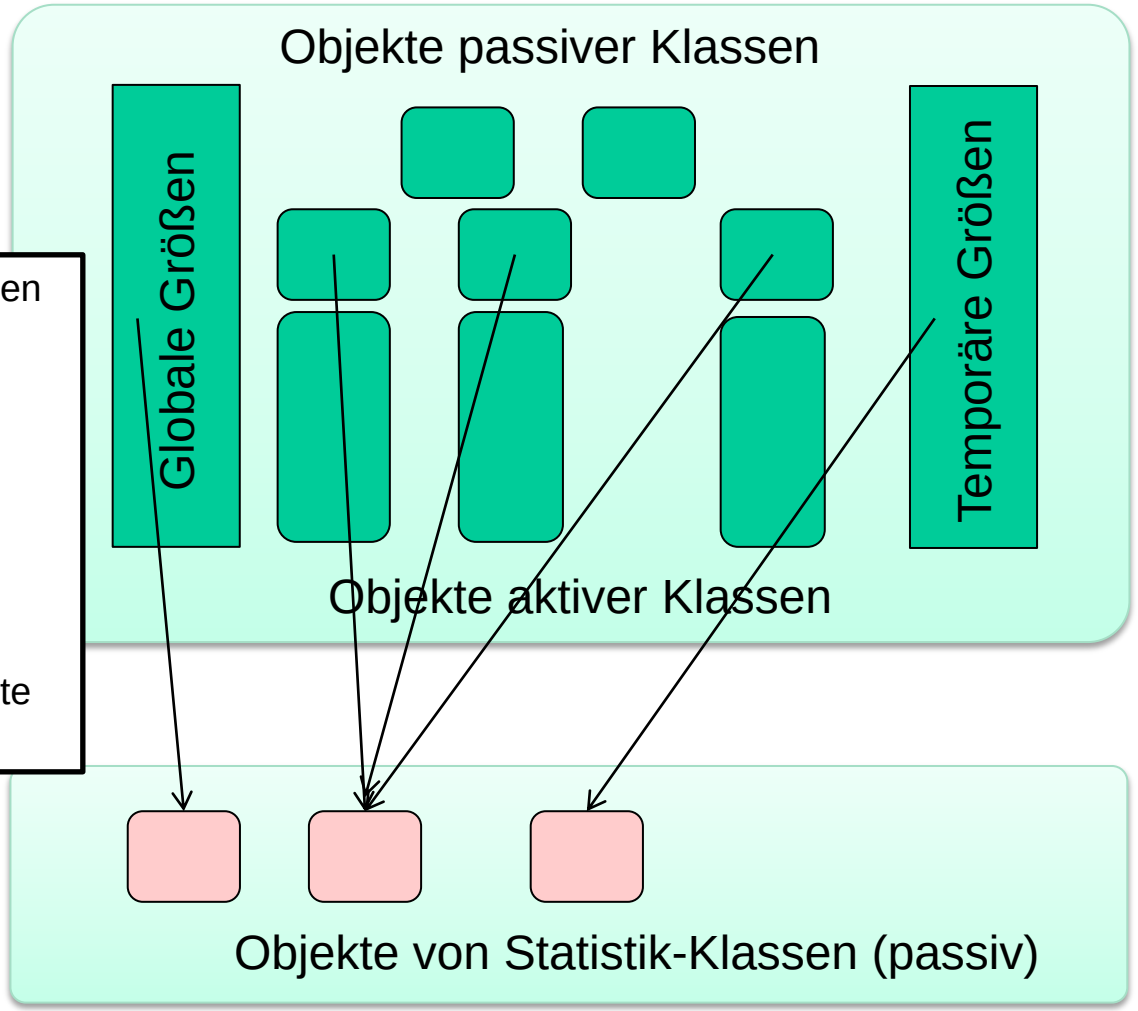


- Sammlung von Beobachtungsdaten je ausgezeichnete Modellvariable (Ergebnisgröße) in einem Simulationslauf
- Verdichtung der gewonnenen Rohdaten zu statistischen Kenngrößen (Mittelwert, Streuung/Standardabweichung) und deren Speicherung
- Durchführung weiterer Simulationsläufe bei Ermittlung und Speicherung der Kennwerte
- Berechnung von statistischen Parametern wie Mittelwert und Konfidenzintervall für die Ergebnisgrößen für alle Simulationsläufe

Erfassung von Beobachtungsdaten (Datenströme)

Kategorien von Ergebnisgrößen

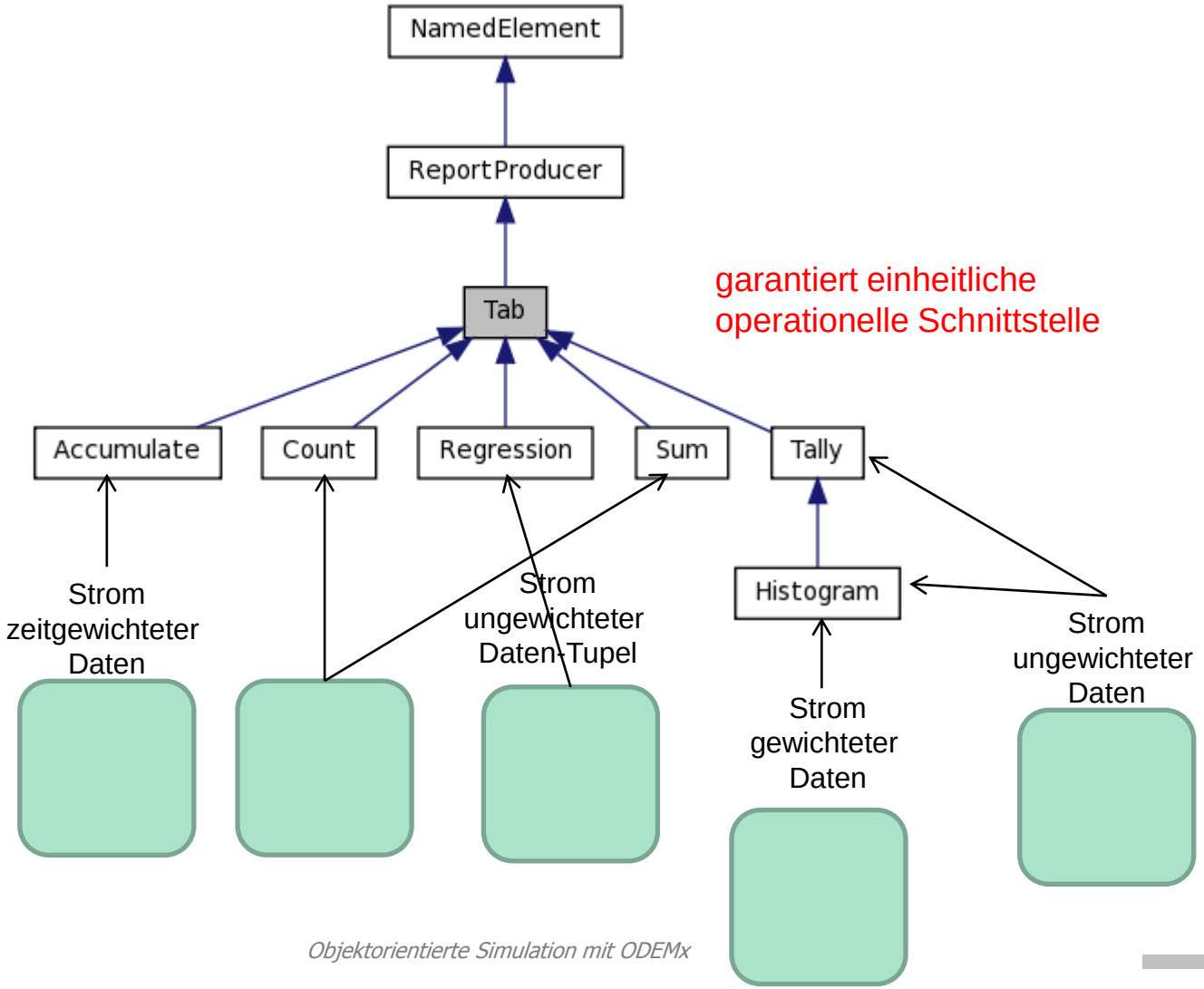
- ungewichtete Beobachtungsdaten
  - Zählungen, Summen
  - Werteverteilung
- zeitgewichtete Daten
- nutzerspezifisch gewichtete Daten



Modell-beschreibung

Modell-bewertung

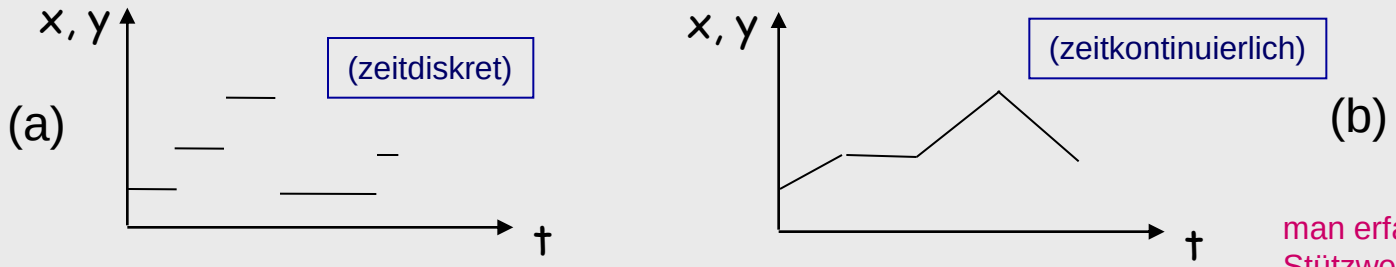
Tab – abstrakte Basisklasse *für alle Statistik-Klassen*



Profile zu beobachtender Modellgrößen

int- **oder** **double-** Modellgrößen x, y mögen sich im Laufe der Simulation ändern
(z.B. x Member der Klasse X, y Member der Klasse Y

unterstützte Arten von Werterfassungen (Beobachtungen)



man erfasst nur Stützwerte und nimmt lineare Übergänge an

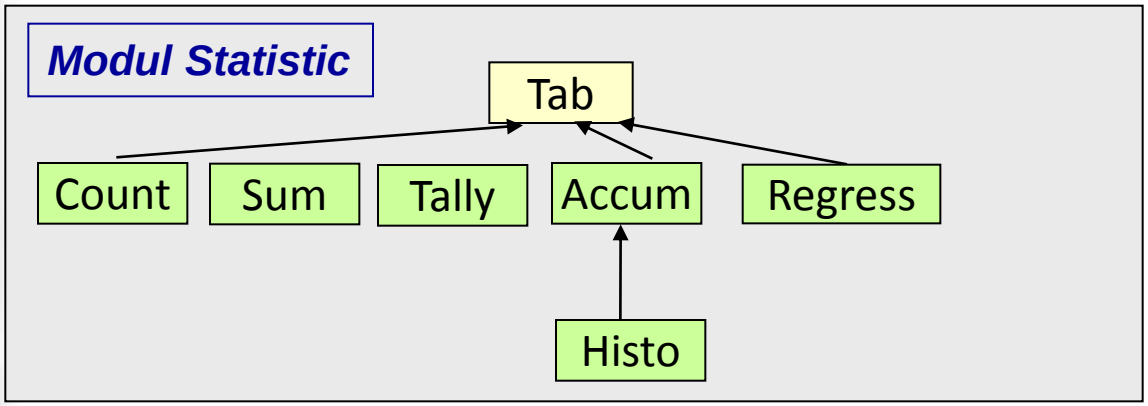
(1) Modellbeobachtungen: x1, x2, x3, … y1, …

(2) Modellbeobachtungen: (x1, t1), (x2, t2), (x3, t3)

(3) Modellbeobachtungen: $(x_1, y_1)^{t1}$, $(x_2, y_2)^{t2}$, $(x_3, y_3)^{t1}$, …

Profil enthält Mittelwert, Standardabweichung, Minimum, Maximum

Anwendung der Statistic-Klassen



Annahme

obsX zeige auf Count-,..,Histo-Objekt

obsXY zeige auf Regress-Objekt

einheitliches Nutzungsschema

- pro Variable x (bzw. Variablenpaar (x,y)) vom Typ int oder double) ist
  - ein Beobachter-Objekt obsX bzw. obsXY zu konstruieren,
  - explizite Erfassung der x-Werte in obsX bzw. der (x,y)-Werte in obsXY
- zu diskreten Zeitpunkten wird x beobachtet: obsX->update(x)
  oder x und y: obsXY->update(x,y)
- zu gewissen Zeitpunkten können die Profile der Größen als Tabellen in Reports generiert werden: obsX->report(), …
- zu gewünschten Zeitpunkten können Einschwingphasen ausgeblendet werden: obsX->reset(), …

7. ODEMx-Modul Statistik:

Count, Tally, Accum, Histo, Regress

- Motivation und Konzept
- Allgemeine operationelle Schnittstelle (Tab)
- Zähler (Count, Sum)
- Ungewichtete Modelldaten (Tally, Histo)
- Gewichtete Modelldaten (Accum)
- Lineare Regression

Die abstrakte Klasse Tab

```
class Tab : public data::ReportProducer {
    public:
        Tab( base::Simulation& sim, const data::Label& label );
        virtual ~Tab();

        void update();
        virtual void reset( base::SimTime time );

        std::size_t  getUpdateCount() const;
        base::SimTime  getResetTime() const;

    protected:
        std::size_t  updateCount_;
        base::SimTime  resetTime_;
};
```

```
Tab::Tab( base::Simulation& sim,
          const data::Label& label )
    : ReportProducer( sim, label )
    , updateCount_( 0 )
    , resetTime_( sim.getTime() )
{ }
```

```
void Tab::update() {
        ++updateCount_;
}
```

erst Ableitungen stellen eigentliche update()- Funktionalität mit spezifischer Signatur bereit

```
void Tab::reset( base::SimTime time ) {
        resetTime_ = time;
        updateCount_ = 0;
}
```

7. ODEMx-Modul Statistik:

Count, Tally, Accum, Histo, Regress

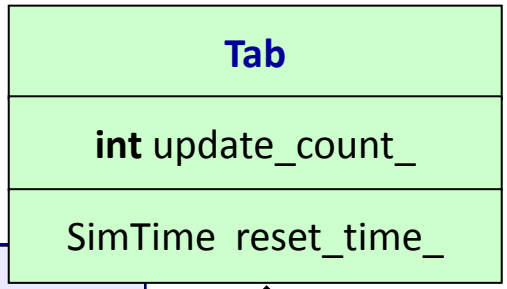
- Motivation und Konzept
- Allgemeine operationelle Schnittstelle (Tab)
- Zähler (Count, Sum)
- Ungewichtete Modelldaten (Tally, Histo)
- Gewichtete Modelldaten (Accum)
- Lineare Regression

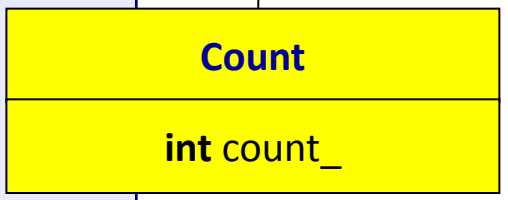
Profilbestimmung von Modellgrößen mit Count

- **Vor.:** zeitdiskrete Variable vom Typ **int**
- **Funktion**: Zähler
  Bestimmung des Profils zu einem beliebigen Zeitpunkt
- **Kennwertprofil**
  Stichprobenumfang (Anzahl von Änderungen), aktueller Wert
- **Beobachtung**

```
Simulation *sim;

Count *c= new Count(sim, "Zähler");
c->update (-3); // bei einer Reduktion von c um 3
```

Die Klasse Count





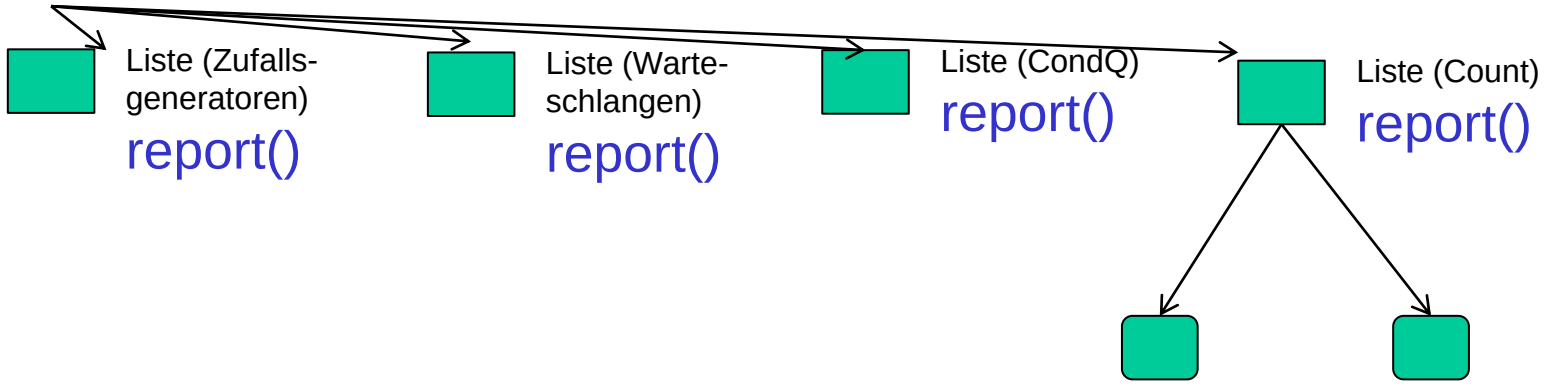

```
class Count : public Tab  {
public:
    Count( base::Simulation& sim,
           const data::Label& label );

    virtual ~Count();

    void update( int value = 1 );
    virtual void reset( base::SimTime time );
    int getValue() const;
    virtual void report( data::Report& report );

private:
    int count_;
};
```

```
void Count::update( int value ) {
        Tab::update();
        count_ += value;
}
```

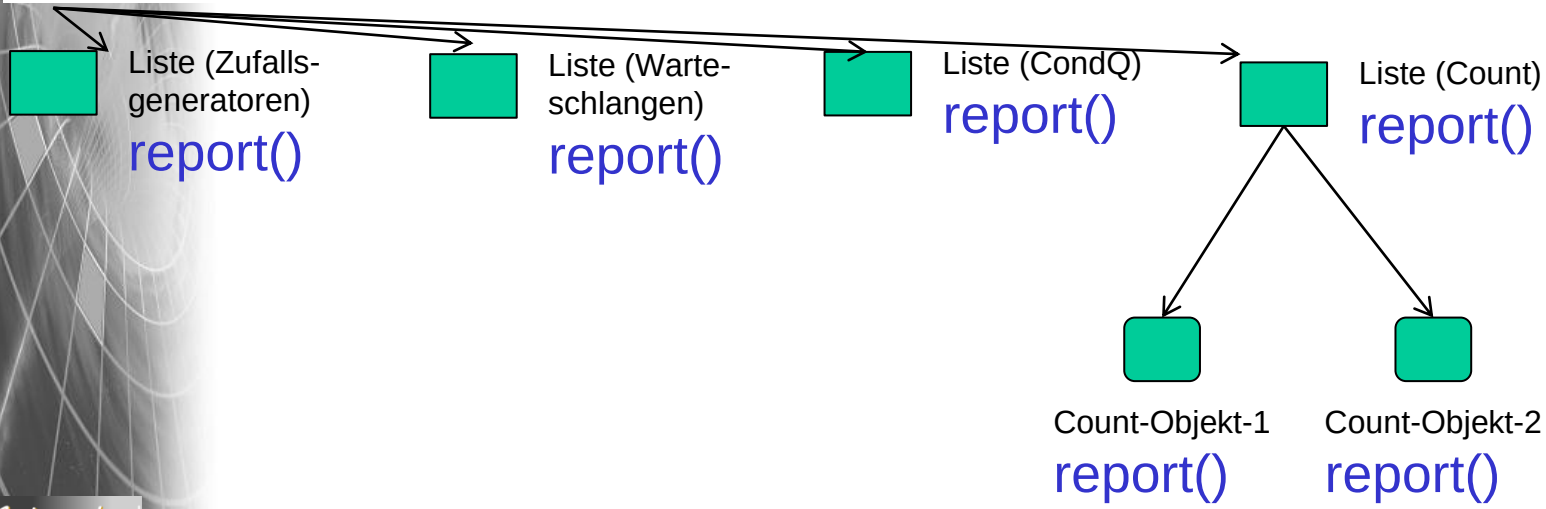
```
void Count::reset ( base::SimTime value ) {
        Tab::reset(time);
        count_ = 0;
}
```

Hierarchie von Report und Reset

report() // vom gesamten System



reset() // vom gesamten System



Beispiel: Count-Report

```
void Count::report( data::Report& report ) {
using namespace data;

        ReportTable::Definition def;
        def.addColumn( "Name", ReportTable::Column::STRING );
        def.addColumn( "Reset at", ReportTable::Column::SIMTIME );
        def.addColumn( "Uses", ReportTable::Column::INTEGER );
        def.addColumn( "Value", ReportTable::Column::INTEGER );

        ReportTable& table = report. getTable( "Count Statistics", def );

        table
        << getLabel()
        << getResetTime()
        << getUpdateCount()
        << getValue();
}
```

Member-Funktionen von Tab/Count

Profilbestimmung von Modellgrößen mit Sum

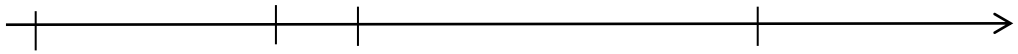
- **Vor.:** zeitdiskrete Variable vom Typ **double**

- **Funktion**
  Summenbildung
  Bestimmung des Profils zu einem beliebigen Zeitpunkt

- **Kennwertprofil**
  Stichprobenumfang (Anzahl von Änderungen), aktueller Wert

- **Beobachtung**

```
Simulation *sim;

Sum *s= new Sum(sim, "Menge");
s->update (15.3); // bei einer Erhöhung von s um 15.3
```



Aufruf von s->report()

SimTime

7. ODEMx-Modul Statistik:

Count, Tally, Accum, Histo, Regress

- Motivation und Konzept
- Allgemeine operationelle Schnittstelle (Tab)
- Zähler (Count, Sum)
- Ungewichtete Modelldaten (Tally, Histo)
- Gewichtete Modelldaten (Accum)
- Lineare Regression

Profilbestimmung von Modellgrößen mit Tally

- **Vor.:** zeitdiskrete Variable vom Typ **double**

- **Funktion**
  Erfassung von Werteänderungen der Variablen
  Bestimmung des Profils zu einem beliebigen Zeitpunkt

- **Tally-Kennwertprofil**
  unabhängig von der jeweiligen Dauer der Wertebelegungen
  (oder einem anderen nutzerspezifischen Gewicht)

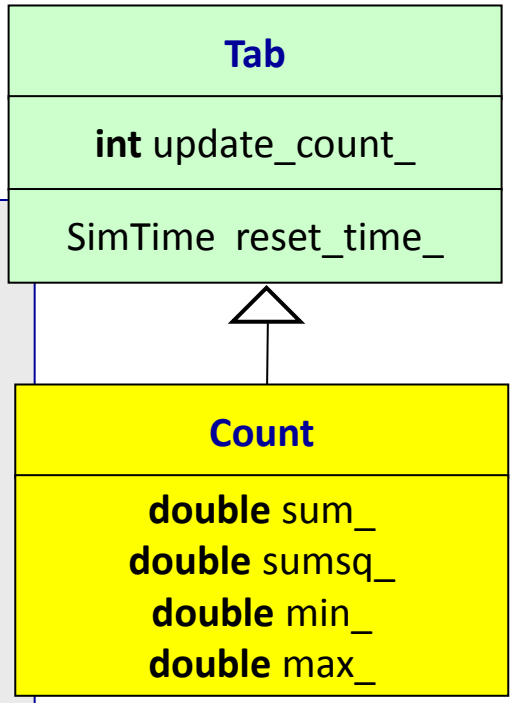
  Stichprobenumfang, Min, Max, Mittelwert, Standardabweichung

- **Beobachtung**

```
double x;
Simulation *sim;

Tally *t= new Tally(sim, "Wartezeiten");

t->update (x);   // x gemessene Wartezeit eines Autos auf eine Fähre
                 // Aufruf für jedes Auto nach Beendigung des Wartens
...
t->report();
```

Die Klasse Tally

```
class Tally : public Tab {
    public:
        Tally (base::Simulation* s, data::Label title="");
        virtual ~Tally();

        virtual void update (double v);
        virtual void reset(base::SimTime time);

        unsigned int getSize()
            {return getUpdateCount();}
        double getMin() {return min_;}
        double getMax() {return max_;}
        double getMean()
            {return getUpdateCount() ?
                    sum/ getUpdateCount() : 0.0;}
        double getDivergence();
        virtual void report (data::Report& r);

    protected:
        double sum_,
               sumsq_,
               min_,
               max_;
};
```



Die Klasse Tally

Wieviel Speicherplätze werden zur Profilbestimmung benötigt ?

- Anzahl der Beobachtungen
- Summe der bisher beobachteten Werte
- Summe der Quadrate der bisher beobachteten Werte
- Minimum
- Maximum

```
class Tally : public Tab {
    public:
        Tally (base::Simulation* s, data::
        virtual ~Tally();

        virtual void update (double v);
        virtual void reset(base::SimTime time);

        unsigned int getSize()
            {return getUpdateCount();}
        double getMin() {return min_;}
        double getMax() {return n
        double getMean()
            {return getUpdateCoun
                    sum/ getUpdat
        double getDivergence();
        virtual void report (data::

    protected:
        double sum_,
               sumsq_,
               min_,
               max_;
};
```

Berechnung erfolgt erst zur Report-Zeit

Mittelwert m

$$m = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Streuung/Varianz s^2

Standardabweichung s

$$s^2 = \frac{1}{n-1} \left(\sum_{i=1}^{n} x_i^2 - \frac{1}{n} \left(\sum_{i=1}^{n} x_i \right)^2 \right)$$

Profil von Modellgrößen mit Histo

- **Vor.:** zeitdiskrete Variable vom Typ **double**

- **Funktion**
  Erfassung von Werteänderungen der Variablen
  Bestimmung des Profils zu einem beliebigen Zeitpunkt

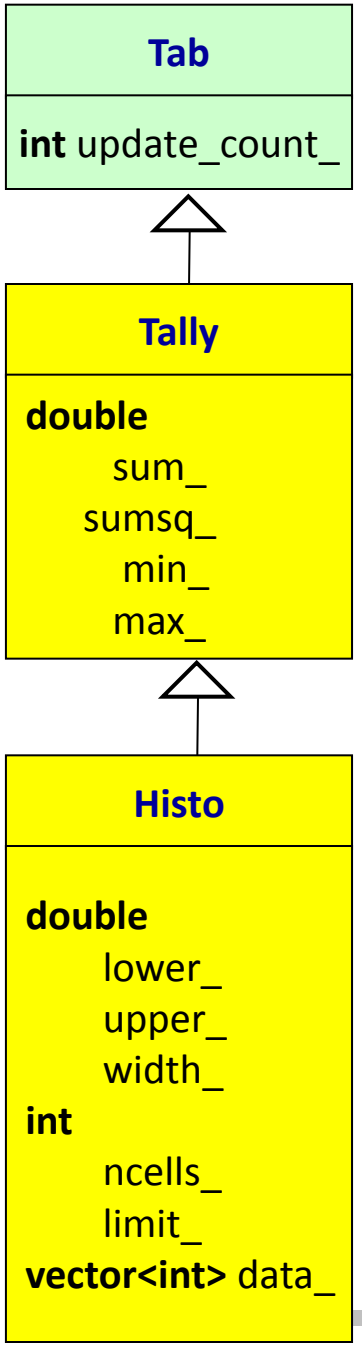
- **Kennwertprofil** (wie bei Tally)
  unabhängig von der jeweiligen Dauer einer Wertebelegung
  mit zusätzlicher Erfassung in vorgegebene Werteklassen

- **Beobachtung**

```
double x;
Simulation *sim;

Histo *h= new Histo(sim, "Wartezeiten", 1.0, 300.0, 25);
h->update (x); // bei jeder Änderung von x
```

Die Klasse Histo

```
class Histo : public Tally {
public:
        Histo(Simulation* s, Label title,
                double low, double up, int n);
        virtual ~Histo();
        virtual void update(double v);
        virtual void reset(SimTime time);

        const Tally* getTally() ;
        const std::vector<int>& getData() ;

        int maximumelem();
        virtual void report(Report& r);

protected:
        double lower_, upper_;
        int ncells_;
        std::vector<int> data_;
        int limit_;
        double width_;
};
```



7. ODEMx-Modul Statistik:

Count, Tally, Accum, Histo, Regress

- Motivation und Konzept
- Allgemeine operationelle Schnittstelle (Tab)
- Zähler (Count, Sum)
- Ungewichtete Modelldaten (Tally, Histo)
- Gewichtete Modelldaten (Accum)
- Lineare Regression

Profilbestimmung von Modellgrößen mit Accum

- **Vor.:** Variable vom Typ double (zeitdiskret/zeitkontinuierlich)

- **Funktion**
  Erfassung von Werteänderungen der Variablen
  Bestimmung des Profils zu einem beliebigen Zeitpunkt

- **Accum-Kennwertprofil**
  abhängig von der jeweiligen Dauer der Wertebelegungen (spezielles Gewicht)
  Stichprobenumfang, Min, Max, Mittelwert, Standardabweichung

- **Beobachtung**

```
double x /*zeitdiskret*/ ,  y /*zeitkontinuierlich*/;
Simulation *sim;

Accum *a1= new ACCUM(sim, "Warteschlangenlänge");
a1->update (x); // bei jeder Änderung von x

 Accum *a2= new ACCUM(sim, "Tankinhalt");
a2->integrate (y); // Annahme eines linearen Übergangs von y zwischen zwei
                   // Beobachtungen
```

Die Klasse Accum

```
class Accum : public Tab {
    public:
        Accum (Simulation* s, Label title="");
        virtual ~Accum();

        virtual void update (double v);
        virtual void integrate (double v);

        virtual void reset (SimTime time);
        unsigned int getSize() const ;
        double getMin() const ;
        double getMax() const ;
        double getMean() const;
        double getDivergence() const;

        virtual void report(Report& r);

    protected:
        double      sumt_,
                    sumsqt_,
                    min_,
                    max_,
                    lasttime_,
                    lastv_;
};
```

Die Klasse Accum

```
class Accum : public Tab {
    public:
        Accum (Simulation* s, Label title="");
        virtual ~Accum();

        virtual void update (double v);
        virtual void integrate (double v);

        virtual void reset (SimTime time);
        unsigned int getSize() const ;
        double getMin() const ;
        double getMax() const ;
        double getMean() const;
        double getDivergence() const;

        virtual void report(Report& r);

    protected:
        double      sumt_,
                    sumsqt_,
                    min_,
                    max_,
                    lasttime_,
                    lastv_;
};
```

```
void Accum::update (double v) {
        double now, span;
    //Zeitintegral einer Treppenfunktion
        Tab::update();
        now    = env->getTime();
        span   = now – lasttime_;
        lasttime_= now;

        if (getUpdateCount()>1) {
            sumt_   += lastv_ * span;
            sumsqt_ += lastv_ * lastv_ * span;
        }
        lastv_   = v;

        if (getUpdateCount() == 1) min_ = max_ = v;
        else if (v < min_) min_ = v;
        else if (v > max_) max_ = v;
}
```

7. ODEMx-Modul Statistik:

Count, Tally, Accum, Histo, Regress

- Motivation und Konzept
- Allgemeine operationelle Schnittstelle (Tab)
- Zähler (Count, Sum)
- Ungewichtete Modelldaten (Tally, Histo)
- Gewichtete Modelldaten (Accum)
- Lineare Regression

Profil von Modellgrößen mit Regress

- **Vor.:** Paar zeitdiskreter Variablen vom Typ **double**

- **Funktion**
  Erfassung von Werteänderungen des Variablenpaares (x, y)
  Bestimmung einer angenommenen linearen Abhängigkeit für ein Beobachtungsintervall

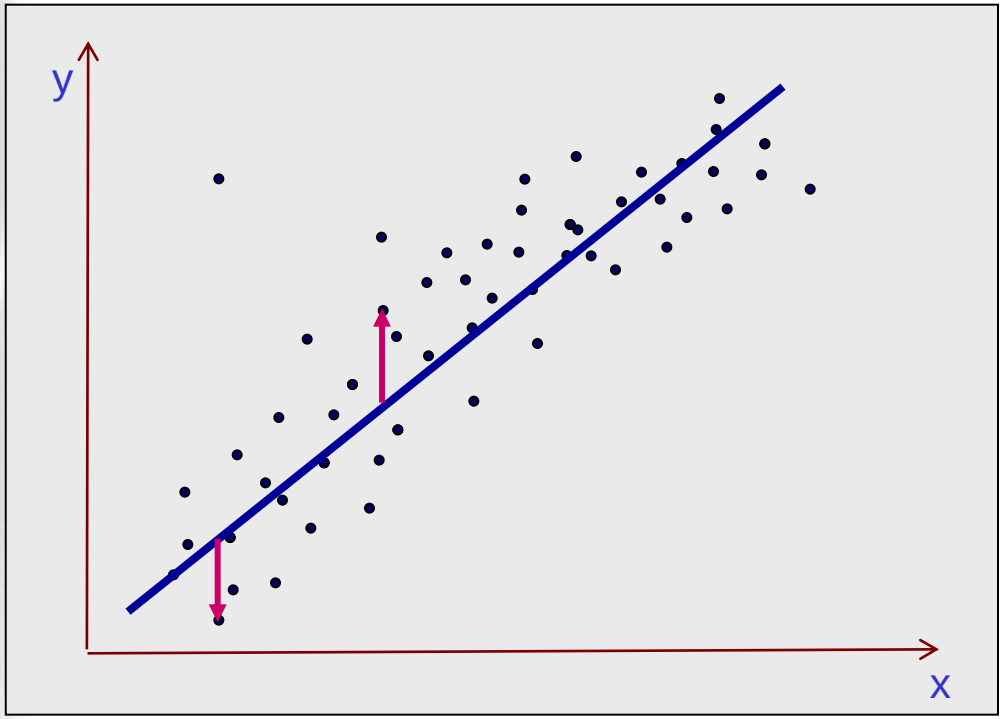
- **Kennwertprofil**
  unabhängig von der jeweiligen Dauer einer Wertebelegung
  Parameterschätzung des linearen Zusammenhangs:
  - Erwartungswert, Standardabweichung für
    Schätzungen von m und b (bei Annahme von y= mx + b),
  - Regressionskoeffizient

- **Beobachtung**

```
double x, y;
Simulation *sim;
Regress *r= new regress ("Abh“, sim);
r->update (x,y);  //bei jeder Änderung von x oder y
```

Lineare Regressionsanalyse

Punkteschwarm (als Ergebnis einer mit Messfehlern behafteten Beobachtung)



Koeffizienten für $y = mx + b$
so bestimmen, dass die Summe der einzelnen quadrierten Abstände minimal wird

Annahme:
bei festem (aber beliebigen) x ist y normalverteilt !!!

Koeffizienten
eines angenommenen linearen Zusammenhangs müssen geschätzt werden

Korrelation

$$r = \frac{\sum_{i=1}^{n} (x_i - x^M)(y_i - y^M)}{(n-1)\, s_x\, s_y}$$

Regressionskoeffizient
$-1 \leq r \leq 1$

Maß für Zusammenhang von x und y

Regressions- oder Korrellationskoeffizient

- **Wert: +1 (bzw. −1)**
  → vollständig positiver (bzw. negativer) linearer Zusammenhang zwischen den betrachteten Merkmalen

- **Wert: 0**
  → Merkmale hängen überhaupt nicht linear voneinander ab.

  Allerdings können x und y in *nicht-linearer* Weise voneinander abhängen.

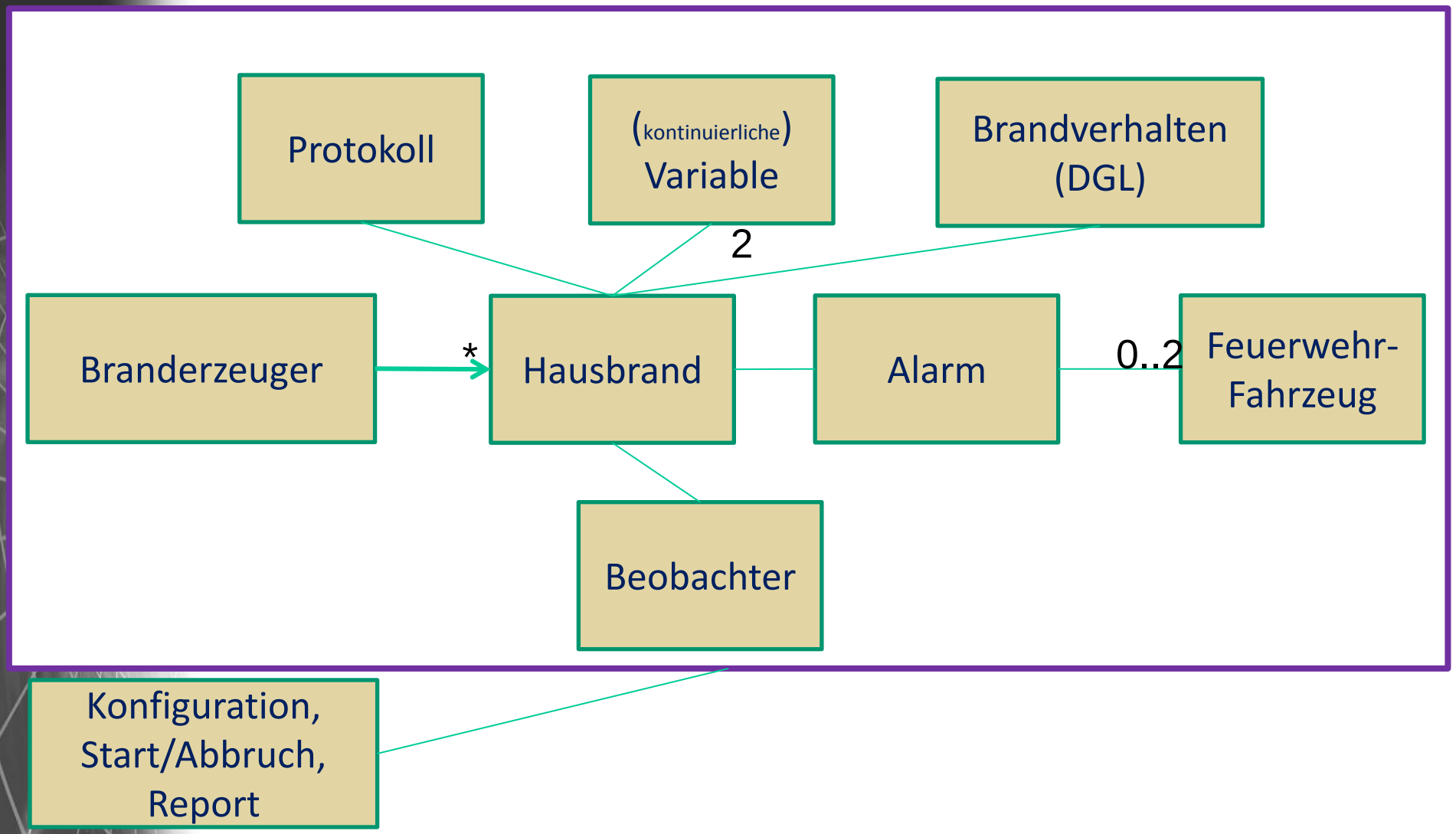
  Achtung: der Korrelationskoeffizient ist damit kein geeignetes Maß für die stochastische Abhängigkeit von Merkmalen an sich

8. Ausblick:

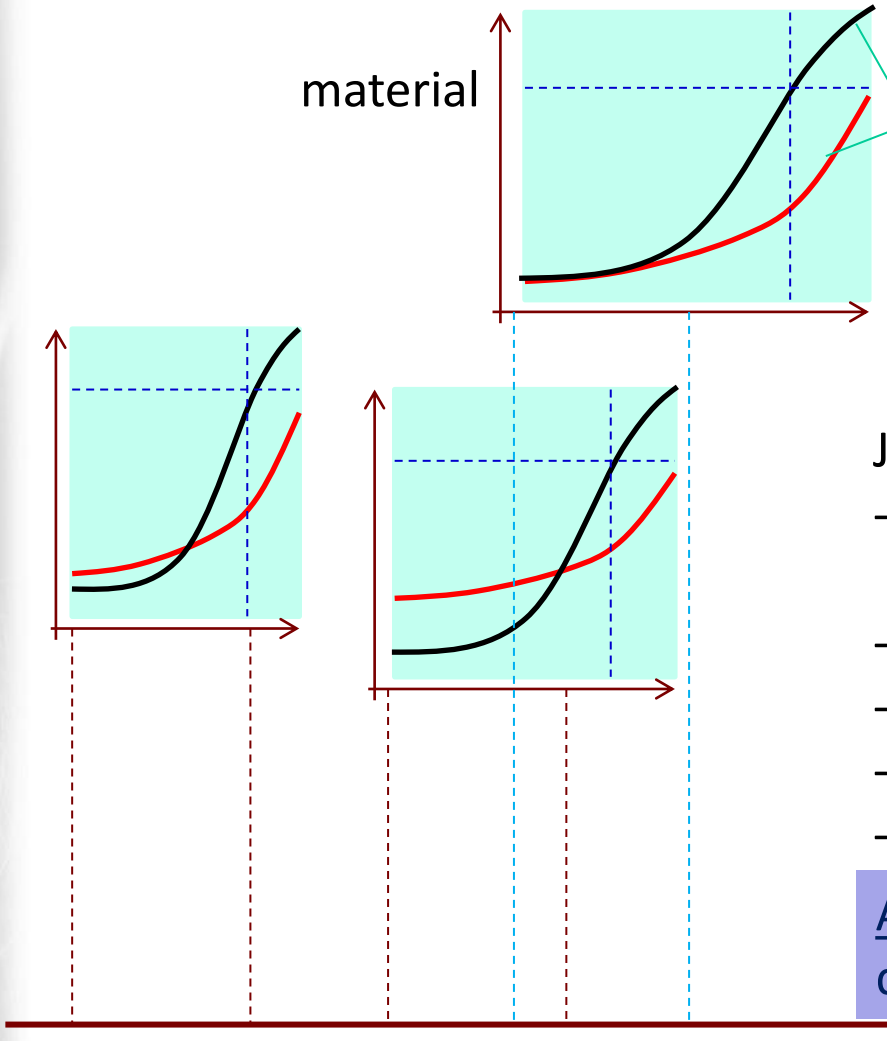
Behandlung zeitkontinuierlicher Zustandsänderungen

- Beispiel: Feuerwehreinsatz
- Konzept für die zeitkontinuierliche Simulation

Feuerwehreinsatz



Unkontrollierte Brände



size‘ (t)= c * size(t)
size (t_0)= initial_size

damage‘ (t)= size(t)

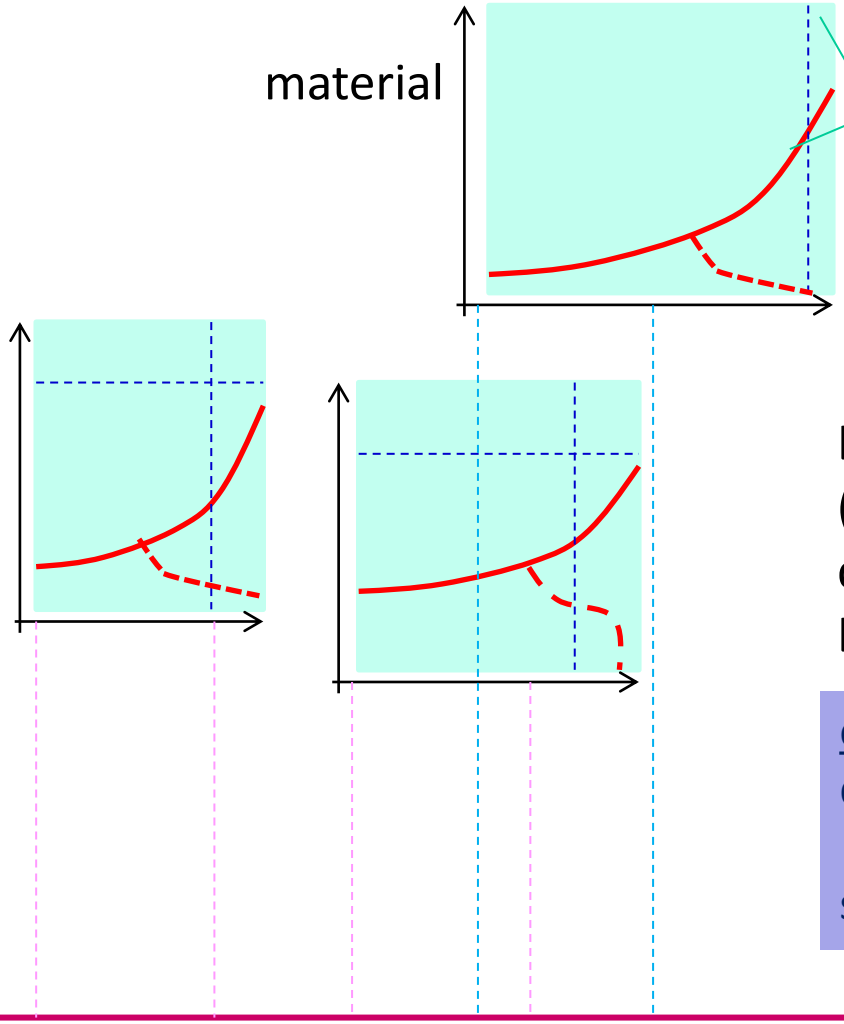
Jeder Brand ist individuell

- brennbare Materialmenge (Konstante material)
- Brennwert (Konstante c)
- initiale Feuergröße size (t_0)
- Startzeitpunkt
- Branddauer

Abbruchbedingung:
damage(t) > material

Kontrollierte Brände

ab Ankunft der Feuerwehr



size‘ (t)= c * size(t) - extinguish
size (t_0)= initial_size

damage‘ (t)= size(t)

Löschrate extinguish
(als Summe
der Löschraten der eingesetzten
Löschfahrzeuge)

erweiterte Abbruchbedingung:
damage(t) > material
||
size(t) <= 0.0

Zeitkontinuierliche *Zustandsänderungen in ODEMX*

- werden vorab unterstützt (historisch: verschiedene Konzepte)

- hier zunächst ein vereinfachter Ansatz:
  - Einführung einer aktiven Klasse Monitor
    - verwaltet „kontinuierliche“ Variablen anderer Prozesse
      (hier: die Variablen size, damage eines jeden Brand-Objektes)

    - verwaltet „beschreibende“ DGLs
      (hier: die Berechnungsvorschriften für jeden Brand)

    - 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 danach mit holdFor(h) verzögert.

Integrationsverfahren

- sehr einfaches Verfahren (Euler-Heun-Verfahren)

- Vektor x(t) gegeben
  hier: (size, damage) (t) = (Feuergröße, Schadensgröße)(t)

- Berechnung der Änderungen zum Zeitpunkt t
  hier (size‘, damage‘) (t) , nach Anwendung der DGLs

- Prädiktor-Schritt:
  Berechnung der neuen Werte zum Zeitpunkt t+h

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

  Berechnung der Änderungen zum Zeitpunkt t+h
  hier (size‘, damage‘) (t) , nach Anwendung der DGLs

  // r2(t)= (size‘, damage‘) (t)

- Korrektor-Schritt:
  abermalige Berechnung der neuen Werte zum Zeitpunkt t+h

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