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

Digger-Beispiel: Logging und Report

```
// Statistikpuffer
using odemx::data::buffer::StatisticsBuffer;
static std::tr1::shared_ptr < StatisticsBuffer > stats =
                                        StatisticsBuffer::create (sim, "Statistik", false);
sim.addConsumer (data::channel_id::statistics, stats);

//enable logging
odemx::data::output::XmlWriterPtr xmlWriter =
                                odemx::data::output::XmlWriter::create("xmltrace.xml");
sim.addConsumer(odemx::data::channel_id::trace, xmlWriter);
```

```
//remove trace comsumer
sim.removeConsumer(xmlWriter);

// Report
data::output::XmlReport xmlReport("digger.xml");
xmlReport.addReportProducer (*stats);
xmlReport.generateReport ();
```

```
int main () {
    Simulation& sim = getDefaultSimulation ();
     LtruckGenerator   *lt;
     StruckGenerator   *st;

  // Statistikpuffer und Loging- Trace- Preparation
  ...
  // Initialisierung
    nextl = new NegativeExponential(sim, "next large", 22.0);
    nexts = new NegativeExponential(sim, "next small", 10.0);
    q     = new Condq (sim, "sq");
    stq   = new Waitq (sim, "s_Truck_q");
    ltq   = new Waitq (sim, "l_Truck_q");

    s     = new Sdigger (sim);
    s->activate ();
    new Ldigger (sim)->activate ();
    new Ldigger (sim)->activate ();
    new LtruckGenerator (sim)->activate ();
    new StruckGenerator (sim)->activate ();

    sim.runUntil (8*60.0);  // 8 h
  // Report- Trace-Handling
  ...
    return 0;
}
```

Report (ODEMx 3.0) nicht gut interpretierbar

ODEMx XML Report

odemx::synchronization::WaitQ

Name	*Reset at*	*master queue*	*slave queue*	*synchronizations*
l_Truck_q	0	l_Truck_q.master-queue	l_Truck_q.slave-queue	223
s_Truck_q	0	s_Truck_q.master-queue	s_Truck_q.slave-queue	96

odemx::data::buffer::StatisticsBuffer Updates

Producer	*Property*	*Updates*	*Min*	*Max*	*Mean*	*Weighted Mean*	*Std Deviation*	*Weighted StDev*
l_Truck_q				0.61039	0.0839453	0.0665617	0.165067	0.145425
l_Truck_q				0.449572	0.143207	0.119505	0.13614	0.131988
l_Truck_q.master-queue	length	103	0	2	0.902913	0.525674	0.782262	0.825623
l_Truck_q.slave-queue	length	456	0	14		72608		3.83903
s_Truck_q	master wait time	96	0	0			0	0
s_Truck_q	slave wait time	96	0	0.740528	0.244685	0.219993	0.195846	0.18316
s_Truck_q.master-queue	length	1	0	0	0	0	0	0
s_Truck_q.slave-queue	length	197	0	9	2.58883	2.36805	2.16771	2.12359
sq	wait time	115	0	0.0174378	0.000188922	0.00117587	0.00166384	0.00435027
sq.queue	length	231	0	1	0.497835	0.00218234	0.499995	0.0466645

maximal wartende Fahrzeuge

maximal wartende Bagger

odemx::random::NegativeExponential

Name	*Reset at*	*inverse mean*	*seed*	*uses*
next large	0	22		215
next small	0	10		100

angekommene Fahrzeuge

odemx::synchronization::CondQ

Name	*Reset at*	*queue*	*signals*	*users*
sq	0	sq.queue	320	115

wünschenswert: Aktuell wartende Fahrzeuge und Bagger, Anzahl der “Durchläufer” wie in früheren ODEMx - Versionen

6. ODEMx-Modul Synchronisation:
WaitQ, CondQ

- Konzept WaitQ
  - Beispiel: Tankerflotte, Hafen, Raffinerie
- Konzept CondQ
  - Beispiel: Hafen, Schlepper, Gezeiten
- Weitere Anwendungsbeispiele für WaitQ u. CondQ
- Zusammenfassung/einheitliche Betrachtung
- Anwendungsbeispiel (Bin, Res, WaitQ)

Beispiel: Logistikproblem

Füllen: Normalverteilung (5 ZE, 1ZE)
Mahlen: 10 ZE
Entnahme: 15 ZE
Ruhezeit: 10 ZE

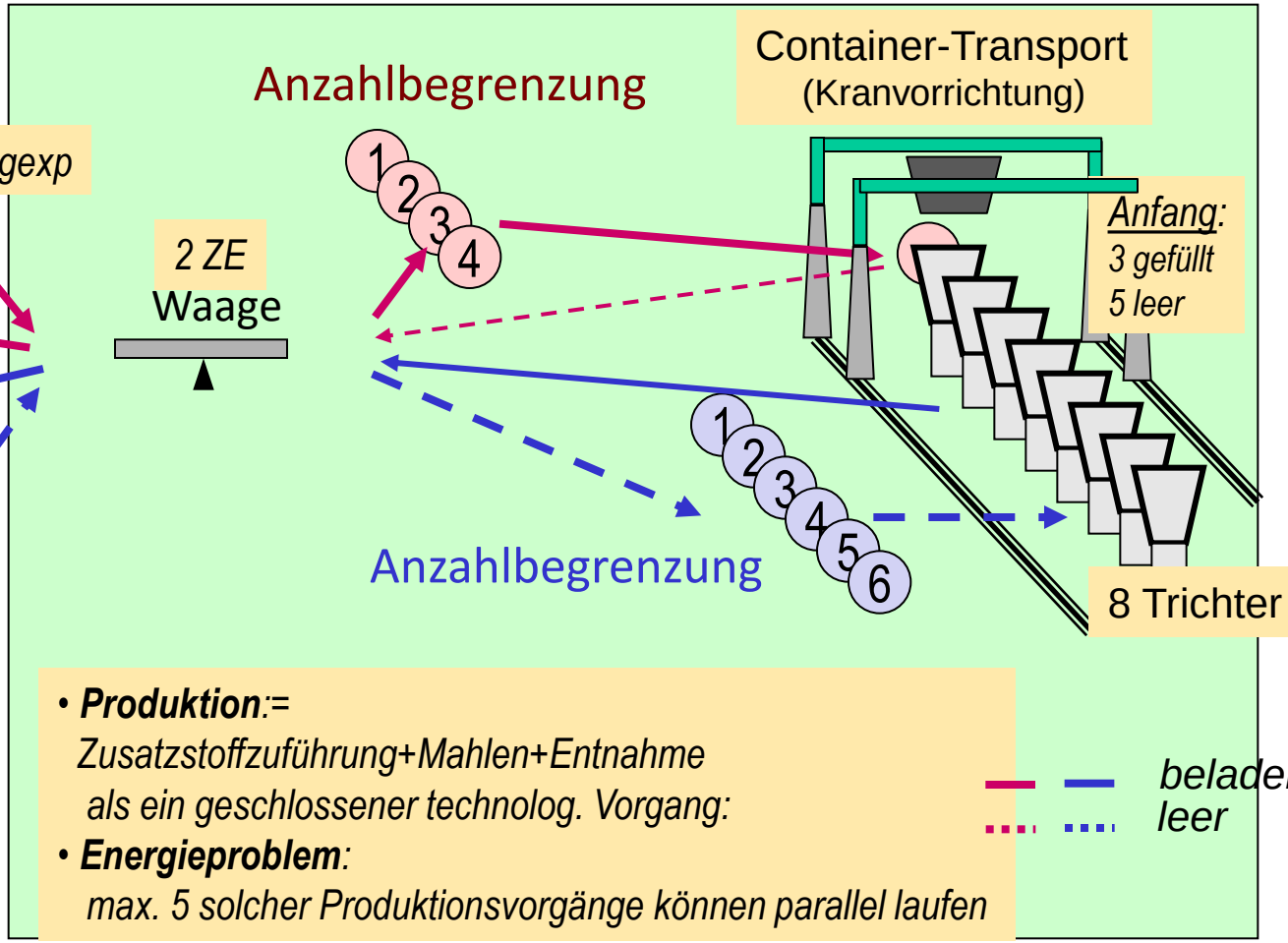
Container-Fahrzeug (füllt 3 Trichter)

7 Objekte

98 ZE + (0.1)-negexp

Abnehmer-Fahrzeug (entleert 2 Trichter)

480 ZE -Simulation

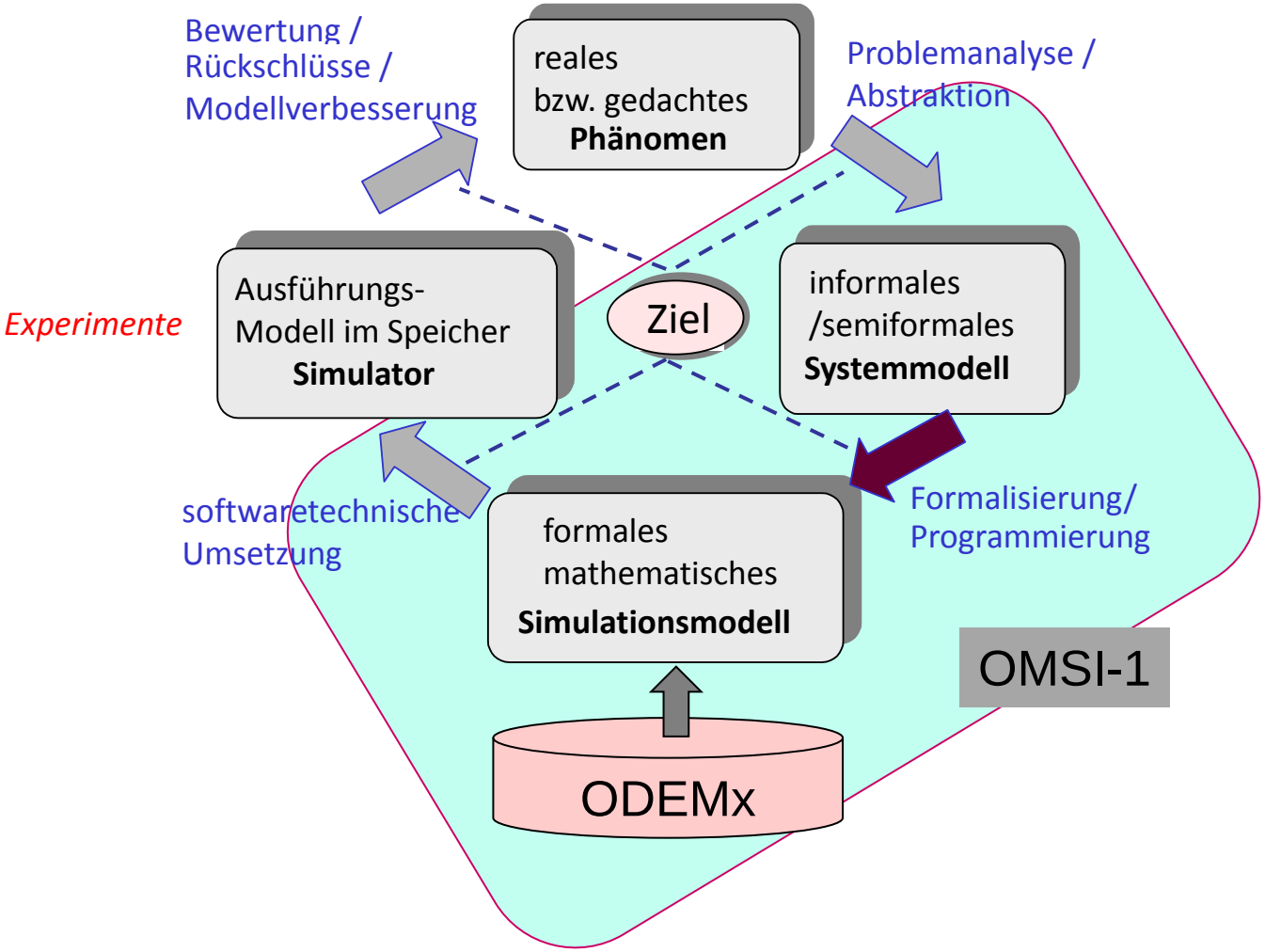


- ***Produktion***:= *Zusatzstoffzuführung+Mahlen+Entnahme als ein geschlossener technolog. Vorgang:*
- ***Energieproblem***: *max. 5 solcher Produktionsvorgänge können parallel laufen*

beladen
leer

Zielstellung:

Experimentelle Bestimmung einer geeigneten Zwischenankunftszeit von Transportfahrzeugen zur Abholung des Endprodukts



Spezifische Modellierungsprobleme

- Zuweisung mehrerer Ressourcen aus einem Spektrum unterschiedlicher **Ressourcenklassen**
  - Stellplätze,
  - Waage,
  - Kranvorrichtung,
  - Trichter (Grundzustand: leer, gefüllt, in Benutzung)
  - Energie

- Durchführung unterschiedlicher, aber feststehender **Folgen von Arbeitsgängen** mit Ressourcenforderungen
  - **a)** **Container**: Stellplatzreservierung – Waage – freier Trichter (3x) - Entladung – Waage – Stellplatzfreigabe
  - **b)** **Endprodukt-Transporter:** Stellplatzreservierung – Waage – voller Trichter (2x) - Beladung – Waage – Stellplatzfreigabe
  - **c)** **Produktion**: beladene Trichter – leeres Fahrzeug- Energie

- Unterschiedliche Arten von **Ankunftsströmen**
  - feste Anzahl von Fahrzeugen mit stochastischer Ankunftszeit
  - unbestimmte Anzahl von Fahrzeugen mit stochastischer Ankunftszeit

Modellierung der Systemelemente:
Fahrzeug, Stellplatz, Waage, Kran, Trichter, Produktion

- Res-Objekt für **Stellplätze** der Container-Fahrzeuge
  mit 4 initialen Token

- Res-Objekt für **Stellplätze** der Endprodukt-Fahrzeuge
  mit 6 initialen Token

- Res-Objekt für **Waage**
  mit 1 initialen Token

- Res-Objekt für **Kranvorrichtung**
  mit 1 initialen Token

- **Trichter** (als 8 Token),
  die dynamisch zwischen zwei Bin-Objekten ausgetauscht werden (frei – gefüllt)

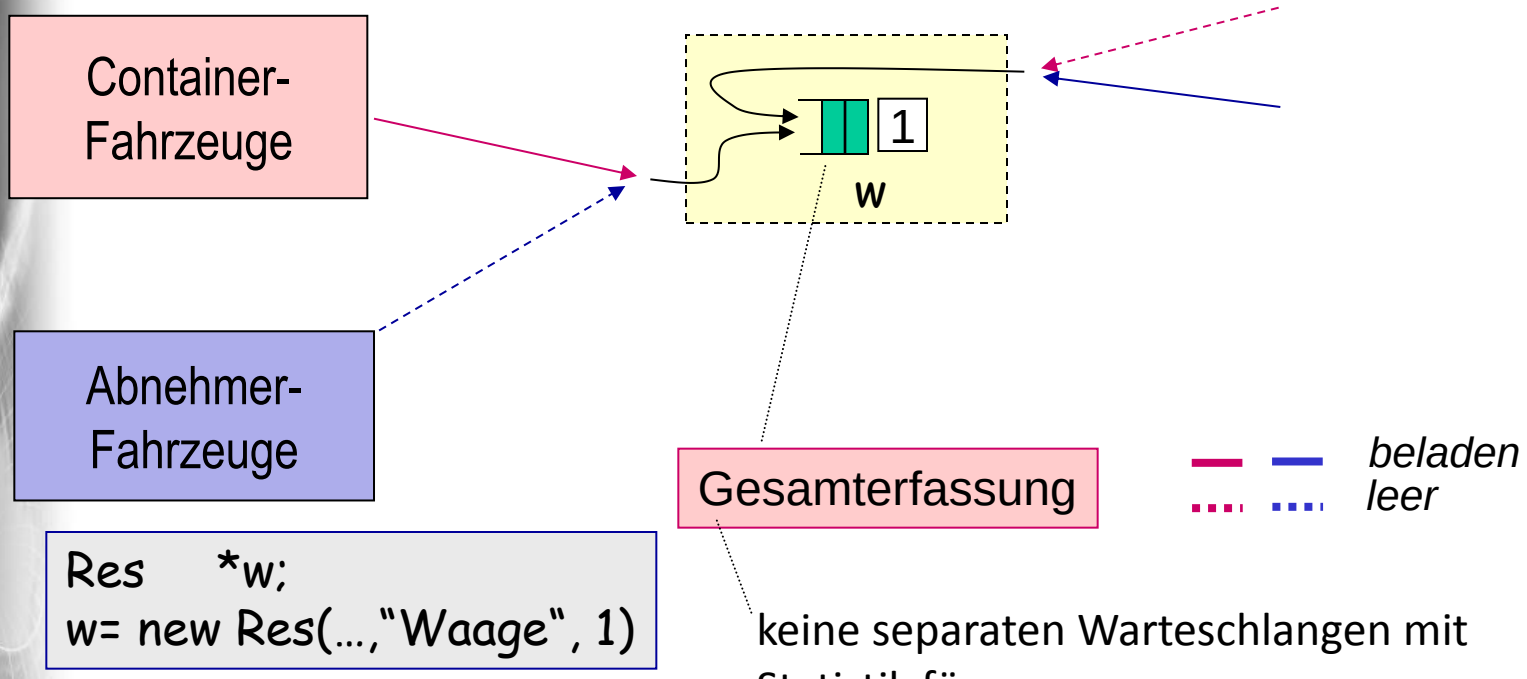
- Synchronisation von 5 **Produktionsobjekten** (Master) und **Endprodukt-Fahrzeugen** (Slave) per WaitQ

Modellierung der Waage (1)

… als

- Res-Objekt mit zugehöriger Warteschlange

```
w->acquire(1);
holdFor(…);
w->release(…);
```




```
Res     *w;
w= new Res(…,"Waage", 1)
```

keine separaten Warteschlangen mit Statistik für
ankommende und verlassende Fahrzeuge

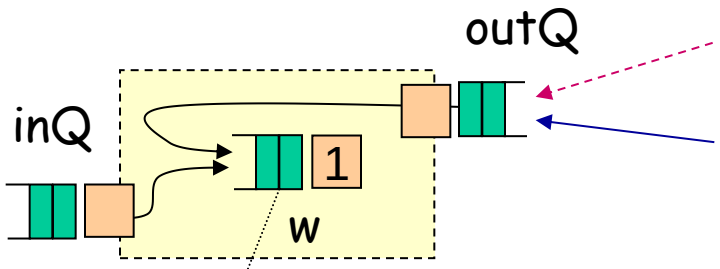
Alternative Modellierung der Waage (2)

… als Ensemble aus

- 2 Bin-Objekten und
- 1 Res-Objekt

Container-Fahrzeuge

Abnehmer-Fahrzeuge



100% Durchläufer

```
Bin *inQ, *outQ;
Res *w;
```

```
inQ->take(1)
w->acquire(1);
holdFor(…);
w->release(1);
outQ->give(1);
oder
inQ->give(1);
```

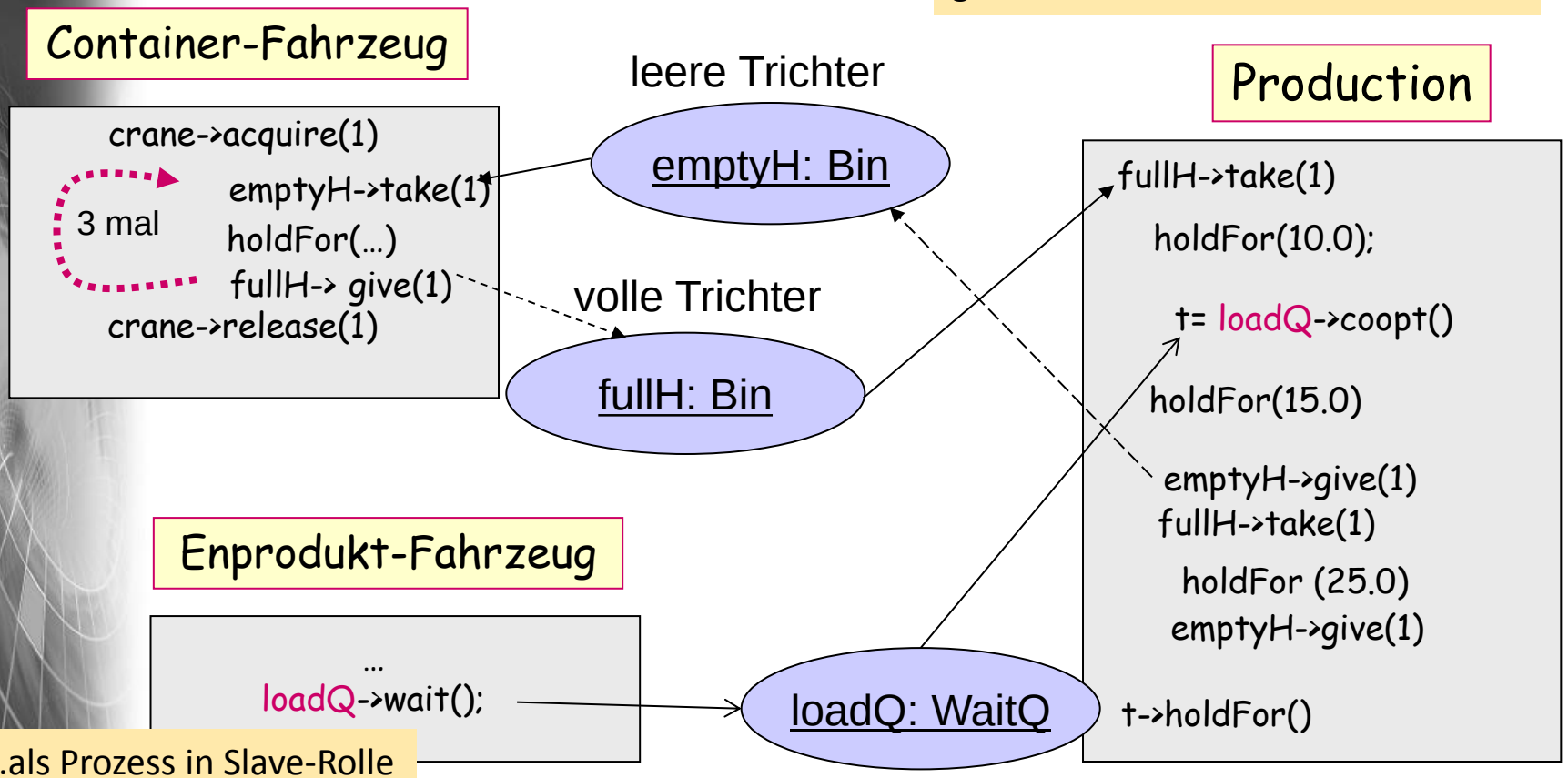
```
outQ->take(1)
w->acquire(1);
holdFor(…);
w->release(1);
outQ->give(1);
oder
inQ->give(1);
```

beladen
leer

Modellierung der Trichter und Produktion

…als feste Anzahl von Token (**8**), die in Abhängigkeit ihres Zustandes in unterschiedlichen Bin-Objekten verwaltet werden.

…als Prozess in Master-Rolle unter Nutzung von Token, die Trichter repräsentieren
5 Instanzen von Production werden generiert



…als Prozess in Slave-Rolle

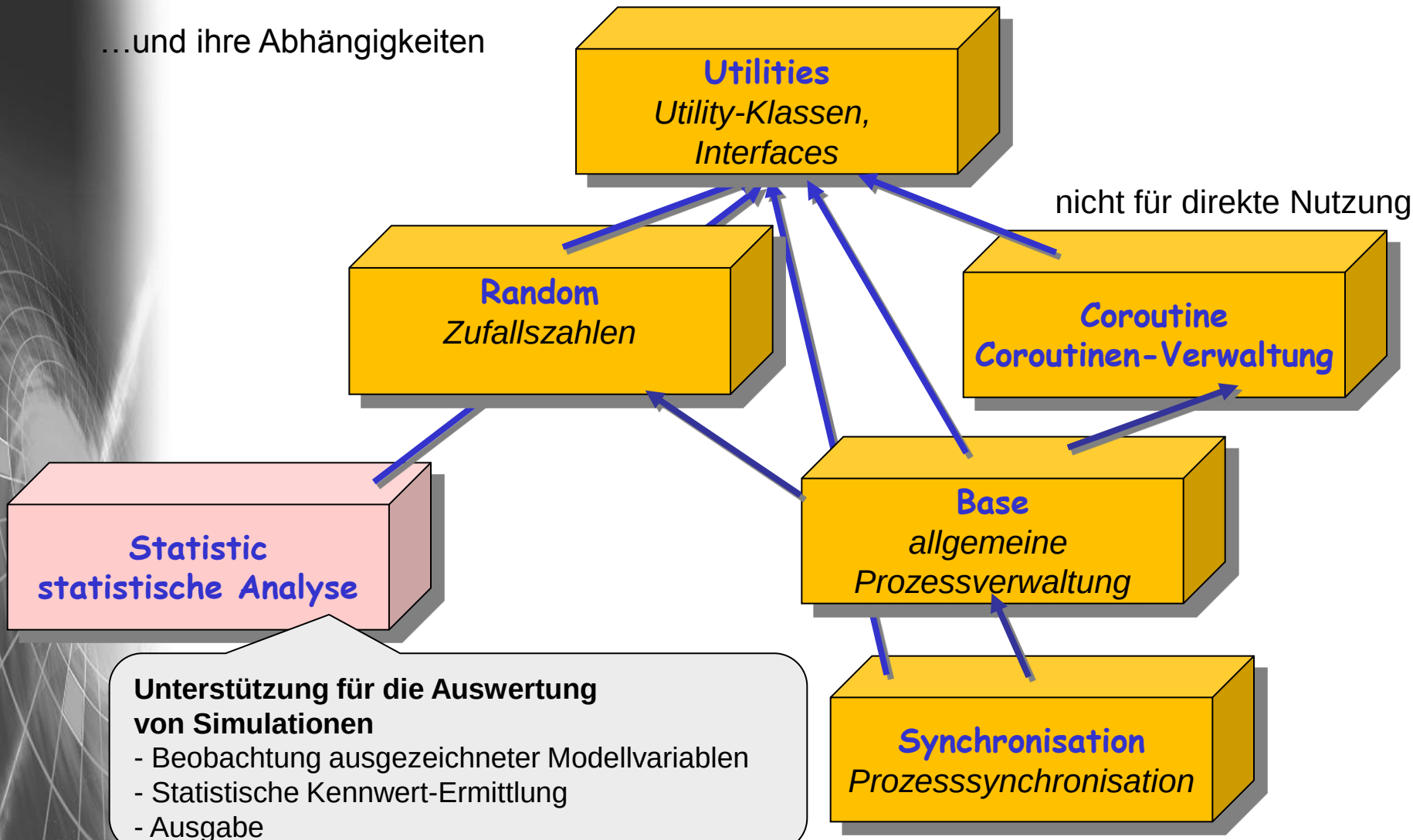
7. ODEMx-Modul Statistik:

Count, Tally, Accum, Histo, Regress

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

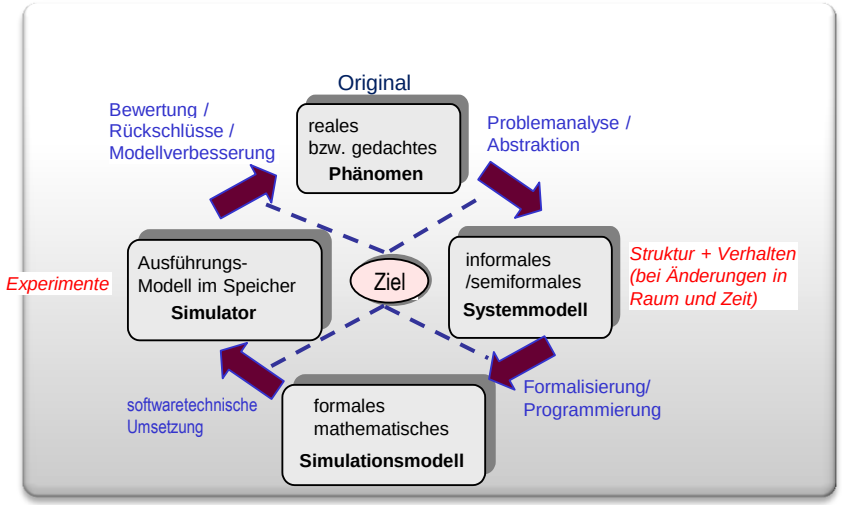
Die ODEMx-Module

…und ihre Abhängigkeiten



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

Modellauswertung



Typischer Ablauf

1. Sammlung von Beobachtungsdaten
   je ausgezeichnete Modellvariable
   (Ergebnisgröße)
   in einem <u>ersten</u> Simulationslauf [0, t_{max}]

2. Verdichtung der gewonnenen Rohdaten zu statistischen Kenngrößen
   (Mittelwert, Streuung/Standardabweichung) und deren Speicherung

3. Wiederholte Durchführung weiterer Simulationsläufe (Schritt 1 und 2) bei Variation identifizierter Experimentierparameter (z.B. Startwerte von Zufallsgeneratoren) :
   Kennwerte $_{SL-1}$, Kennwerte $_{SL-2}$, … , Kennwerte $_{SL-n}$

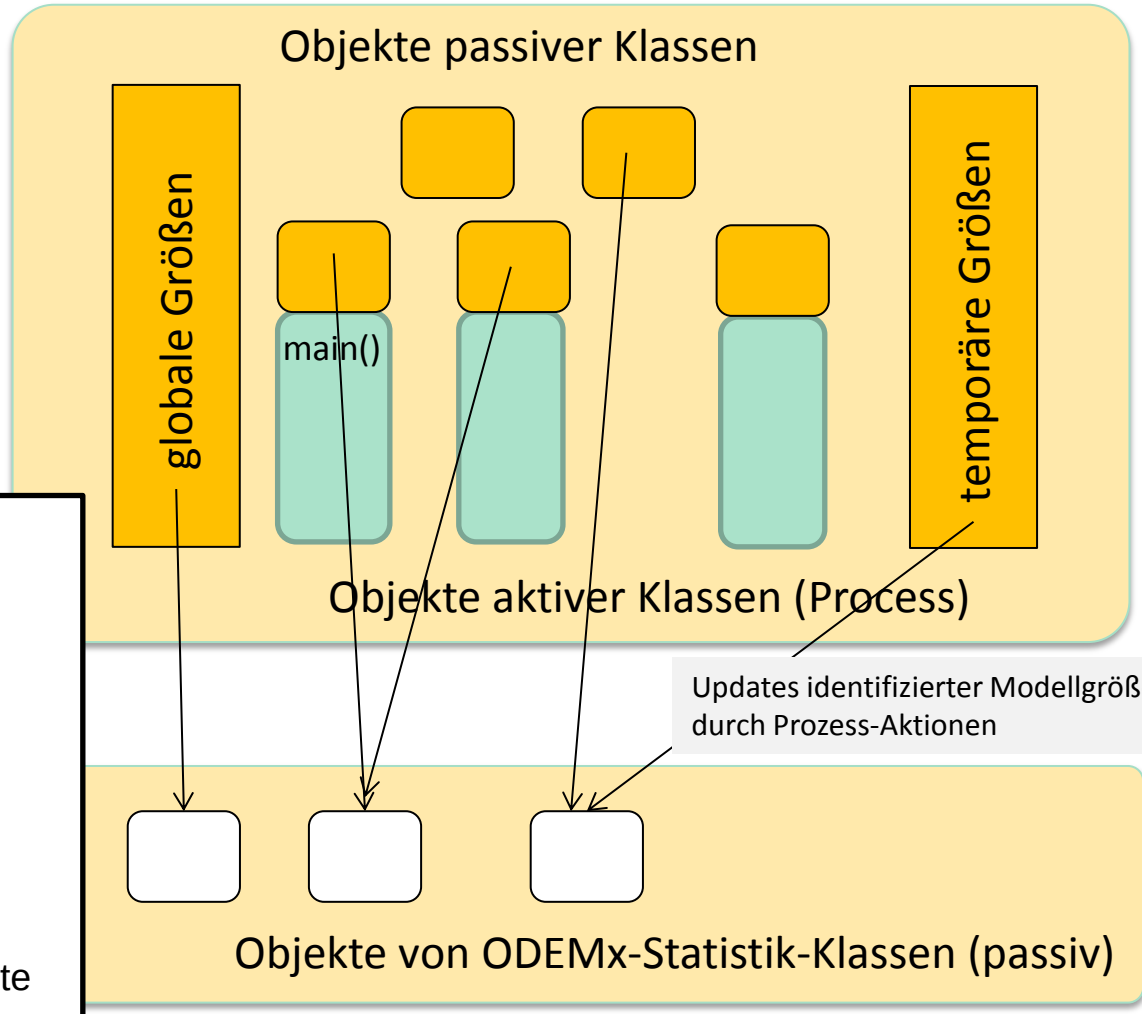
   Berechnung von statistischen Parametern wie Mittelwert und Konfidenzintervall für die Ergebnisgrößen über <u>alle</u> bisherigen Simulationsläufe

   Hauptprogramm startet nacheinander verschiedene Kontexte mit dem gleichen Modell (aber unterschiedlichen Urstartwert für ZZ-Generatoren)

   Sammelt Ergebnisdaten (als Stichprobe) ein und berechnet Kennwerte

nicht mehr
In diesem
Semester !

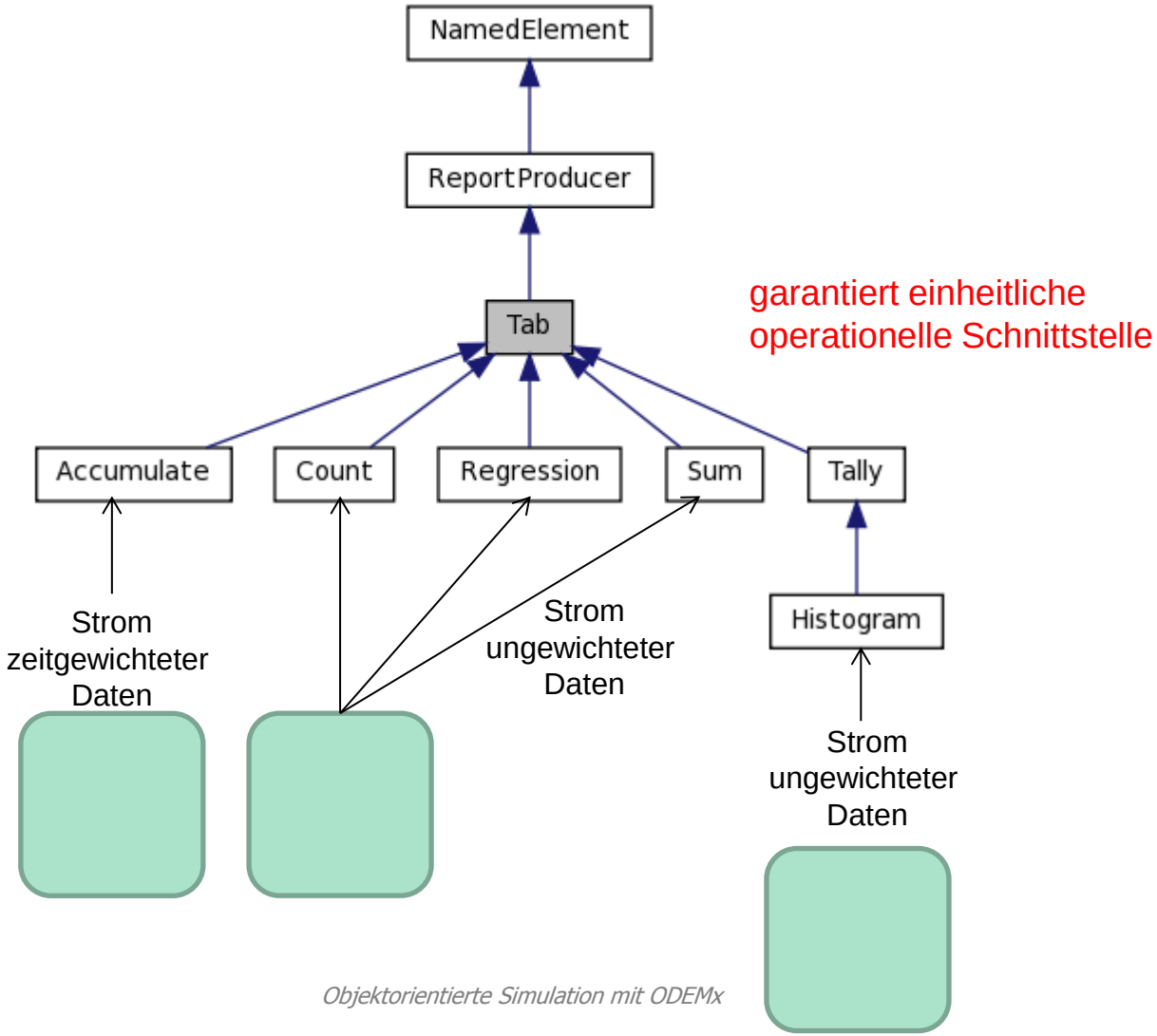
Erfassung von Beobachtungsdaten (Datenströme)



Kategorien von Ergebnisgrößen

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

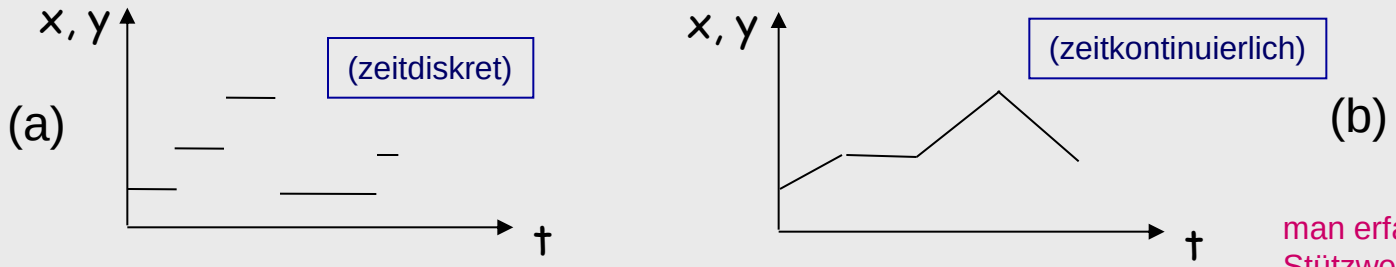
Tab – abstrakte Basisklasse *für alle Statistik-Klassen*



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

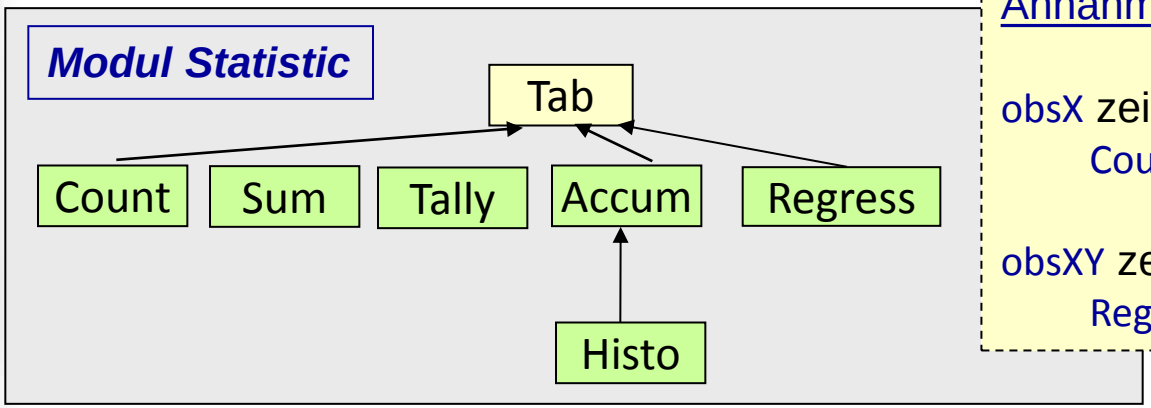
(1) Modellbeobachtungen: x1, x2, x3, … y1, …

(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 Beobachtungsanzahl, Mittelwert, Standardabweichung, Minimum, Maximum

Prinzipielle Anwendung



Annahme

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

obsXY zeige auf
Regress- Objekt

- pro Modellvariable x bzw. Variablenpaar (x,y) vom Typ **int** oder **double** ist
  - ein Beobachter-Objekt obsX bzw. obsXY zu konstruieren,
    zur expliziten Erfassung der x-Werte in obsX bzw.
    der (x,y)-Werte in obsXY

- zu den Zeitpunkten, wo sich x bzw.ändert,
  wird x per obsX->update(x) beobachtet, bzw.

  x und y per obsXY->update(x,y)

- zu gewünschten Zeitpunkten können die Kennwert-Profile der Größen als Tabellen in Reports generiert werden obsX->report(), …
- zu gewünschten Zeitpunkten können Einschwingphasen (die Kennwertprofile verfälschen) 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 {
    protected:
        std::size_t  updateCount_;
        base::SimTime  resetTime_;

    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;

};
```

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

erst Ableitungen stellen eigentliche update()- Funktionalität mit spezifischer Signatur bereit
hier nur: Zählung der Beobachtungen

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

7. ODEMx-Modul Statistik:

Count, Tally, Accum, Histo, Regress

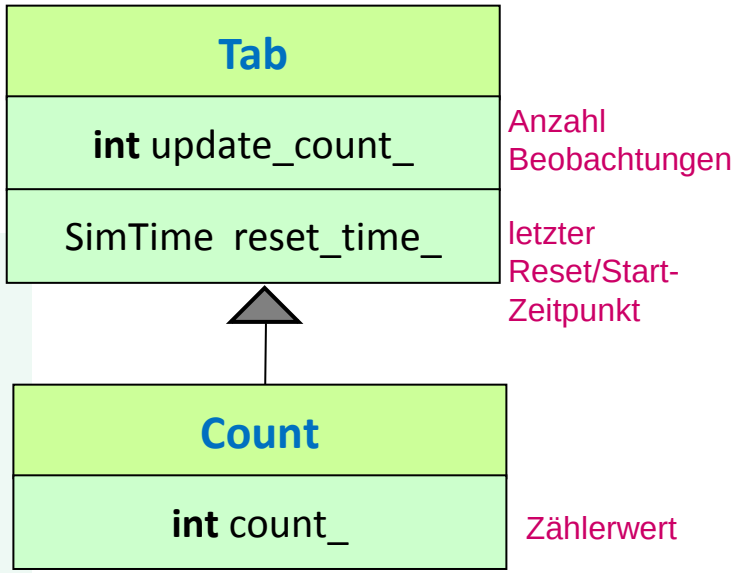
- Motivation und Konzept
- Allgemeine operationelle Schnittstelle (Tab)
- Zähler (Count, Sum)
- Ungewichtete Modelldaten (Tally, Histo)
- Gewichtete Modelldaten (Accum)
- Lineare Regression

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



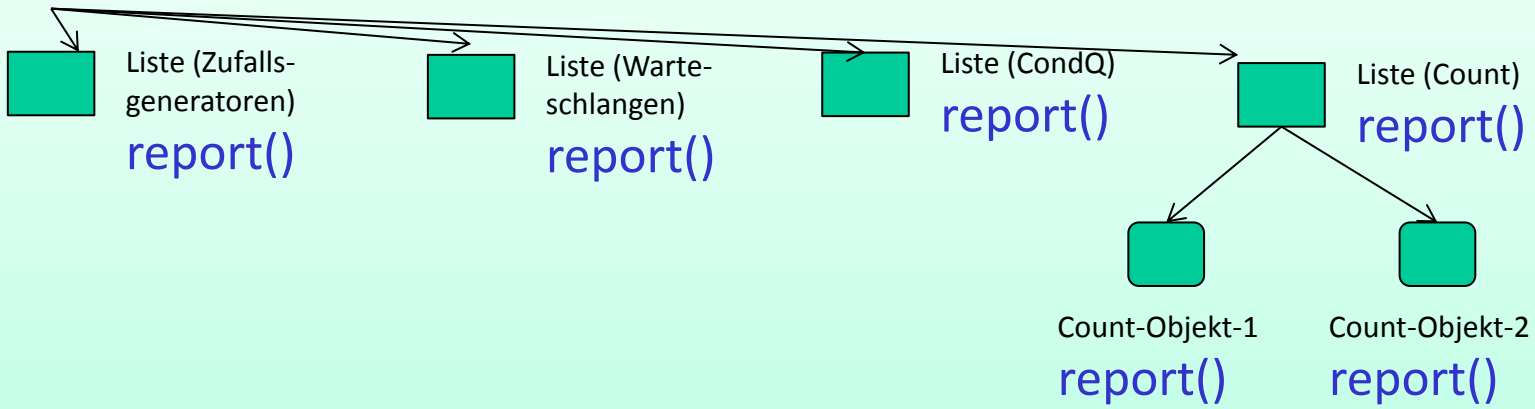

```
class Count : public Tab  {
public:
        Count( base::Simulation& sim,
                const data::Label& label );

        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;
}
```

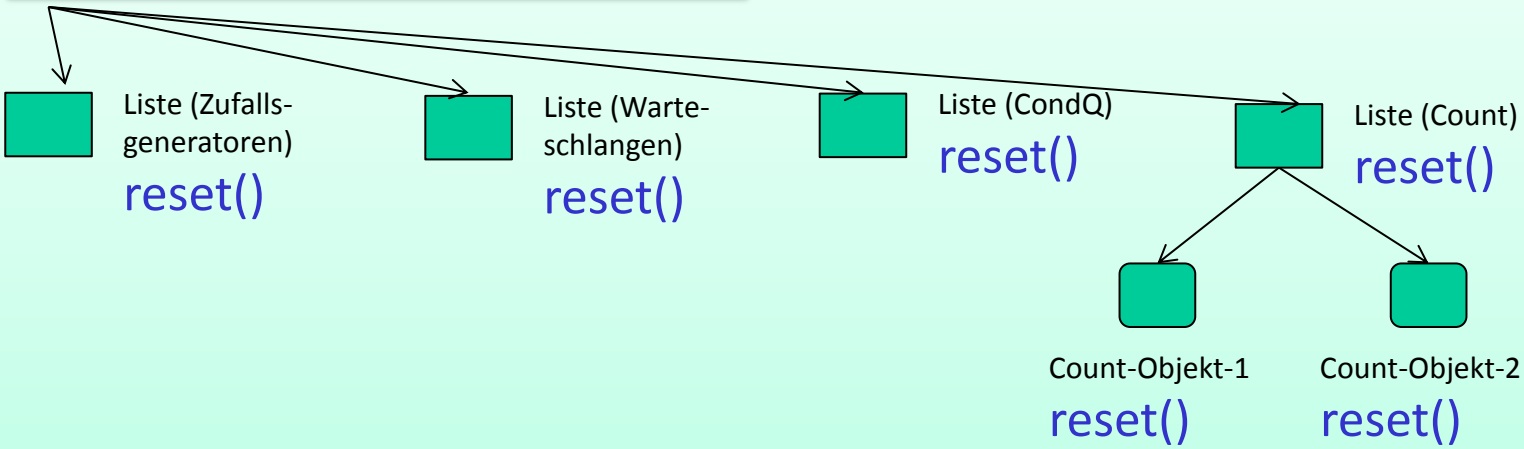
```
void Count::reset ( base::SimTime value ) {
        Tab::reset(time);
        count_ = 0;
}
```

Hierarchie von Report und Reset

report() // vom gesamten System



reset() // vom gesamten System



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
…
s->report();
```

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)

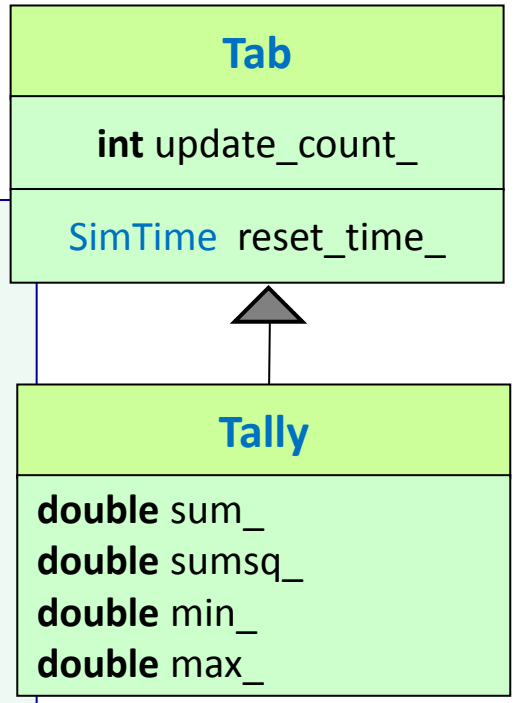
  Stichprobenumfang, Min, Max, Mittelwert, Standardabweichung

- **Beobachtung**

```
double x;   // Wartezeit eines Autos auf eine Fähre
Simulation *sim;

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

t->update (x);   // 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::Labe
        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/ getUpdateC

        double getDivergence();
        virtual void report (data::Rep

    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

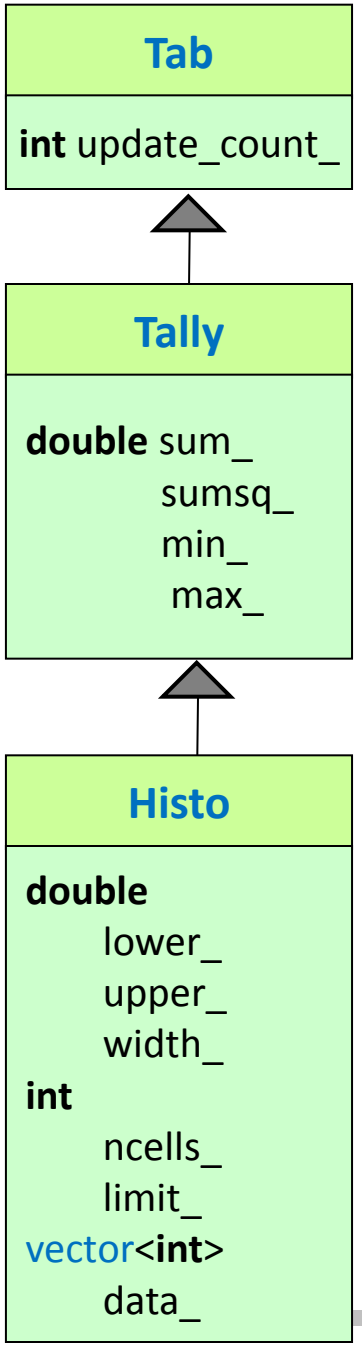
- **Kennwertprofil** (wie bei Tally)
  unabhängig von der jeweiligen Dauer einer Wertebelegung
  mit zusätzlicher Erfassung in Tabelle vorzugebener Werteklassen

- **Beobachtung**

```
double x;   // Wartezeit eines Autos auf eine Fähre
Simulation *sim;

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

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 maxElem();
        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, // zeitdiskrete Größe
          y; //zeitkontinuierliche Größe, die abgetastet wird
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); // linearen Interpolation 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_;
};
```

```
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

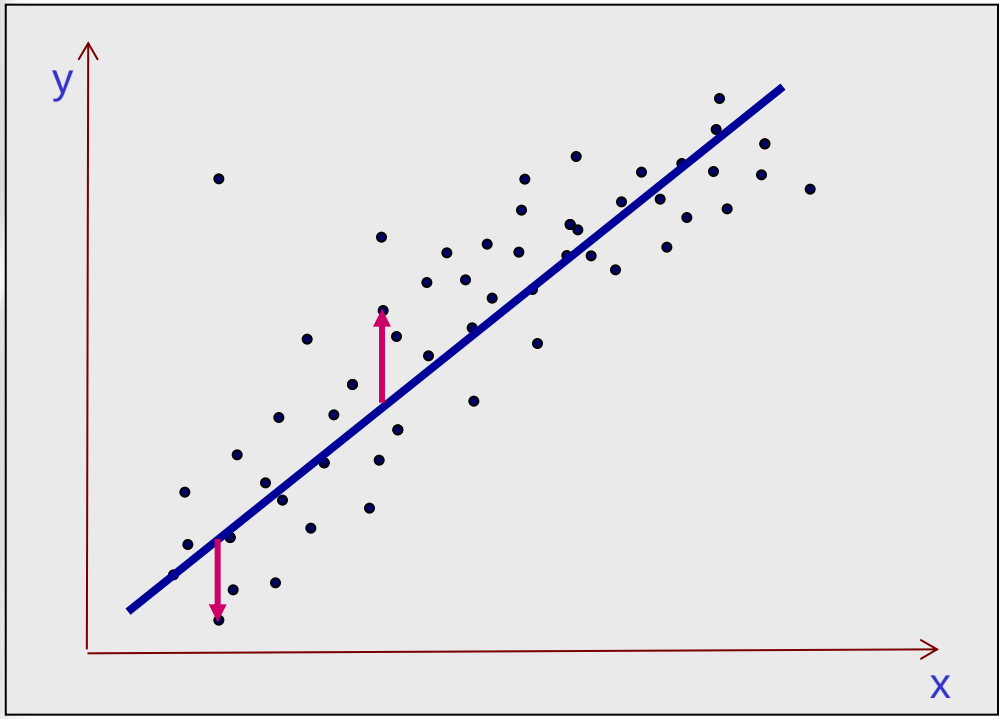
- **Kennwertprofil**
  unabhängig von der jeweiligen Dauer einer Wertebelegung
  Parameterschätzung des linearen Zusammenhangs:
  - Erwartungswert, Standardabweichung für Schätzungen von m und b (bei Annahme von y= mx + b),
  - Regressionskoeffizient

- **Beobachtung**

```
double x,
       y;
Simulation *sim;

Regress*r= new Regress(sim, "lineare Abhängigkeit");
..
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