

# ***Kurs OMSI*** ***im WiSe 2014/15***

## ***Objektorientierte Simulation*** ***mit ODEMx***

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

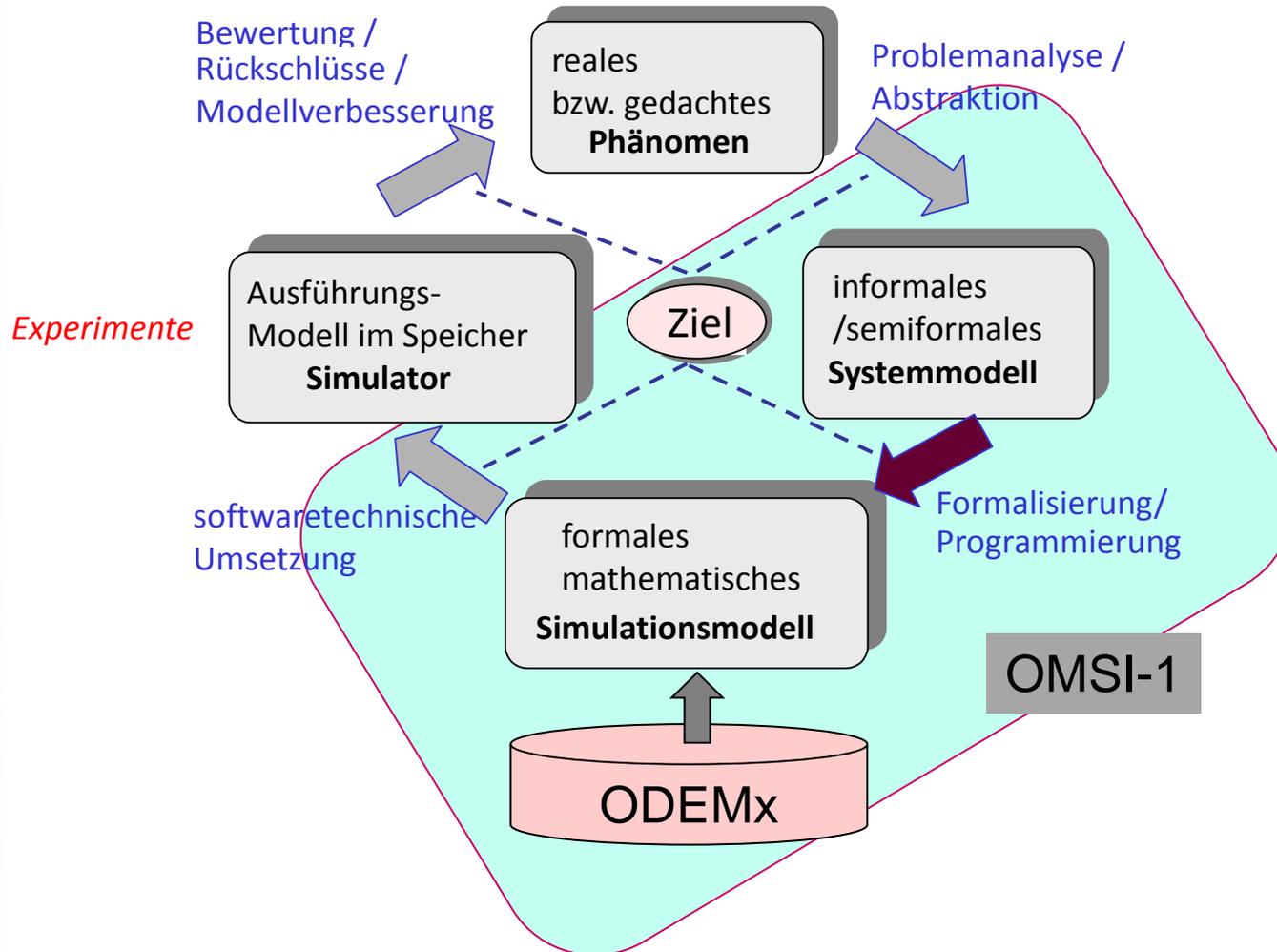
[fischer|ahrens|eveslage@informatik.hu-berlin.de](mailto:fischer|ahrens|eveslage@informatik.hu-berlin.de)

## 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)

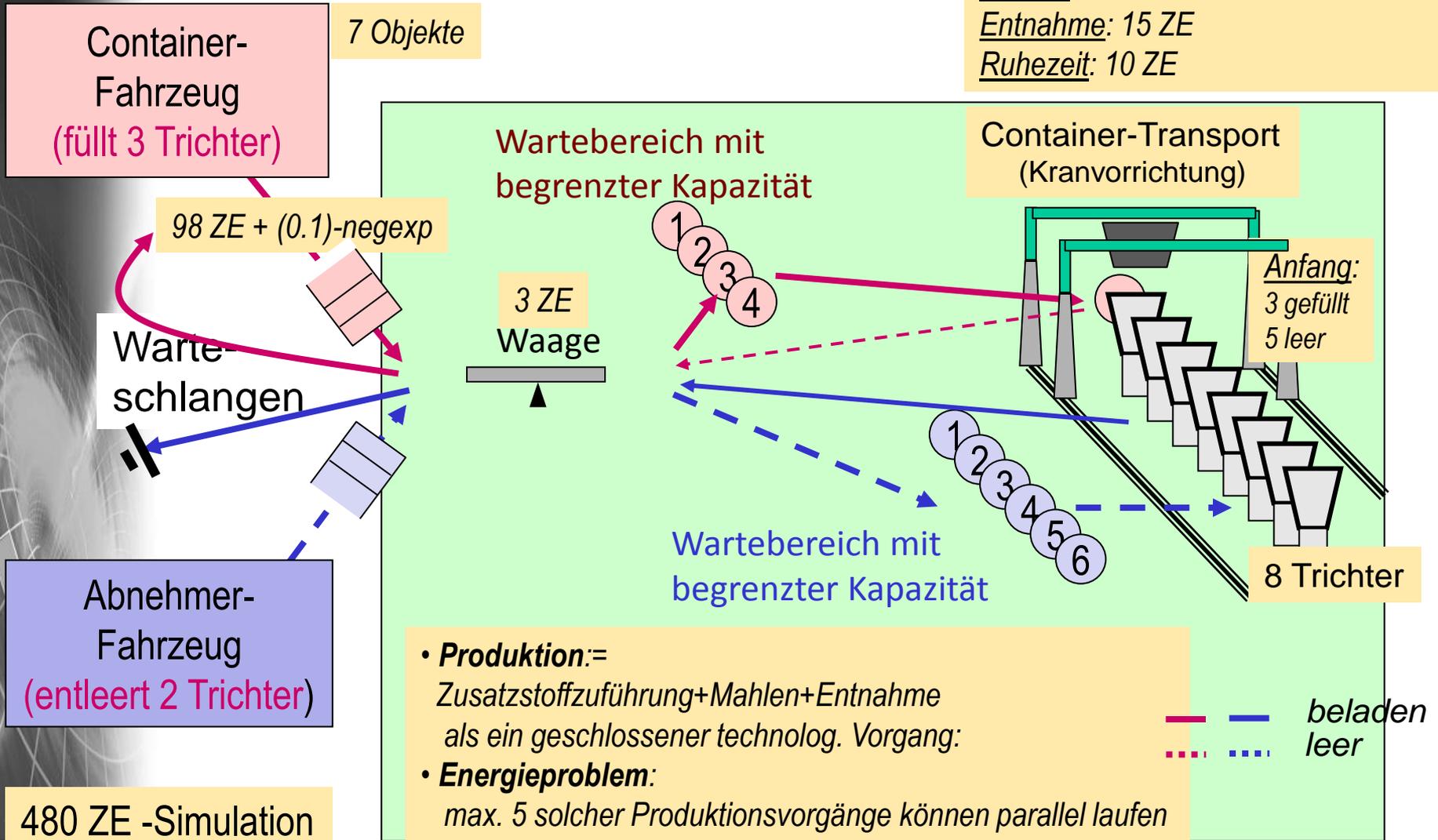
Zielstellung:

**Experimentelle Bestimmung einer geeigneten Zwischenankunftszeit von Transportfahrzeugen zur Abholung des Endprodukts**



# Beispiel: Logistikproblem

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



# 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

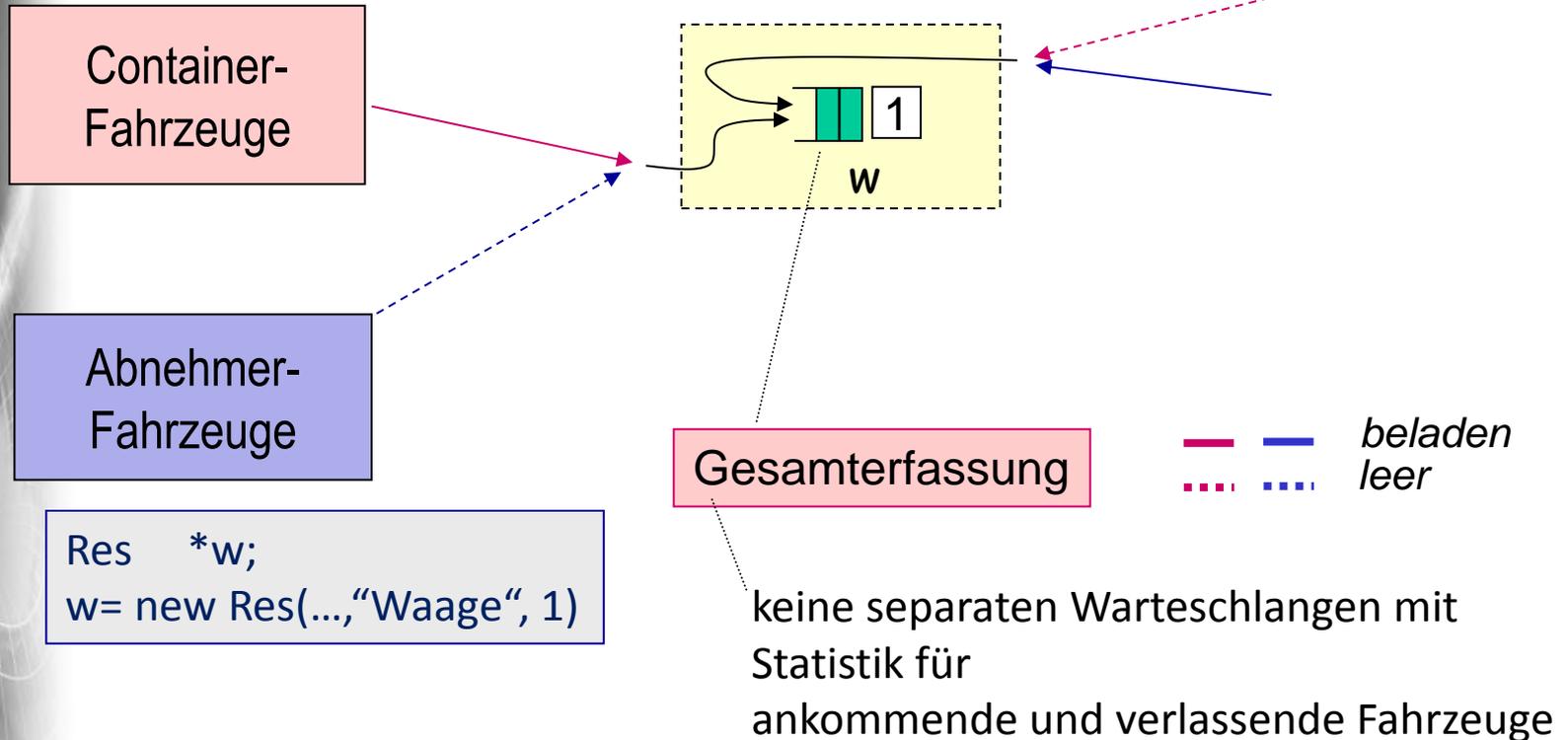
- **Res**-Objekt für **Stellplätze** der Container-Fahrzeuge  
mit 4 initialen Token
- **Res**-Objekt für **Stellplätze** der Endprodukt-Fahrzeuge  
mit 6 initialen Token
- **Res**-Objekt für **Waage**  
mit 1 initialen Token
- **Res**-Objekt für **Kranvorrichtung**  
mit 1 initialen Token
- **Trichter** (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

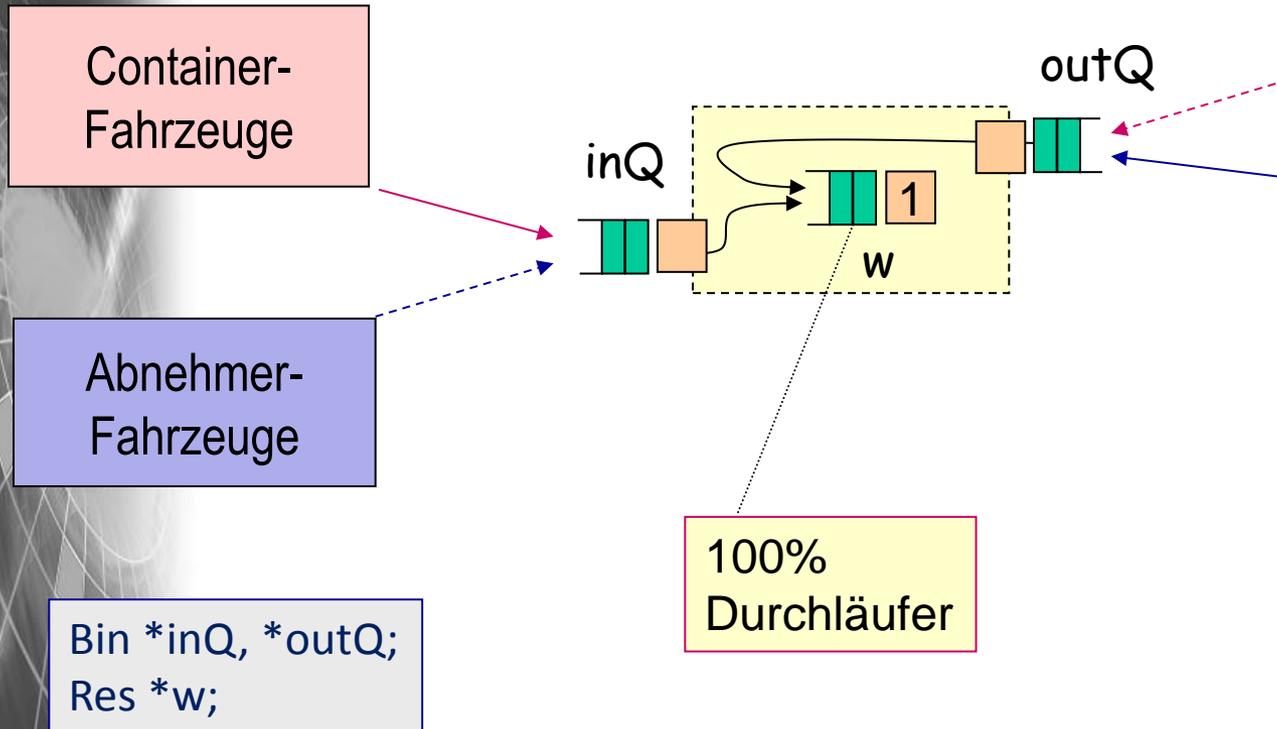
```
w->acquire(1);  
holdFor(...);  
w->release(...);
```



# Alternative Modellierung der Waage (2)

... als Ensemble aus

- 2 Bin-Objekten und
- 1 Res-Objekt



```
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);
```

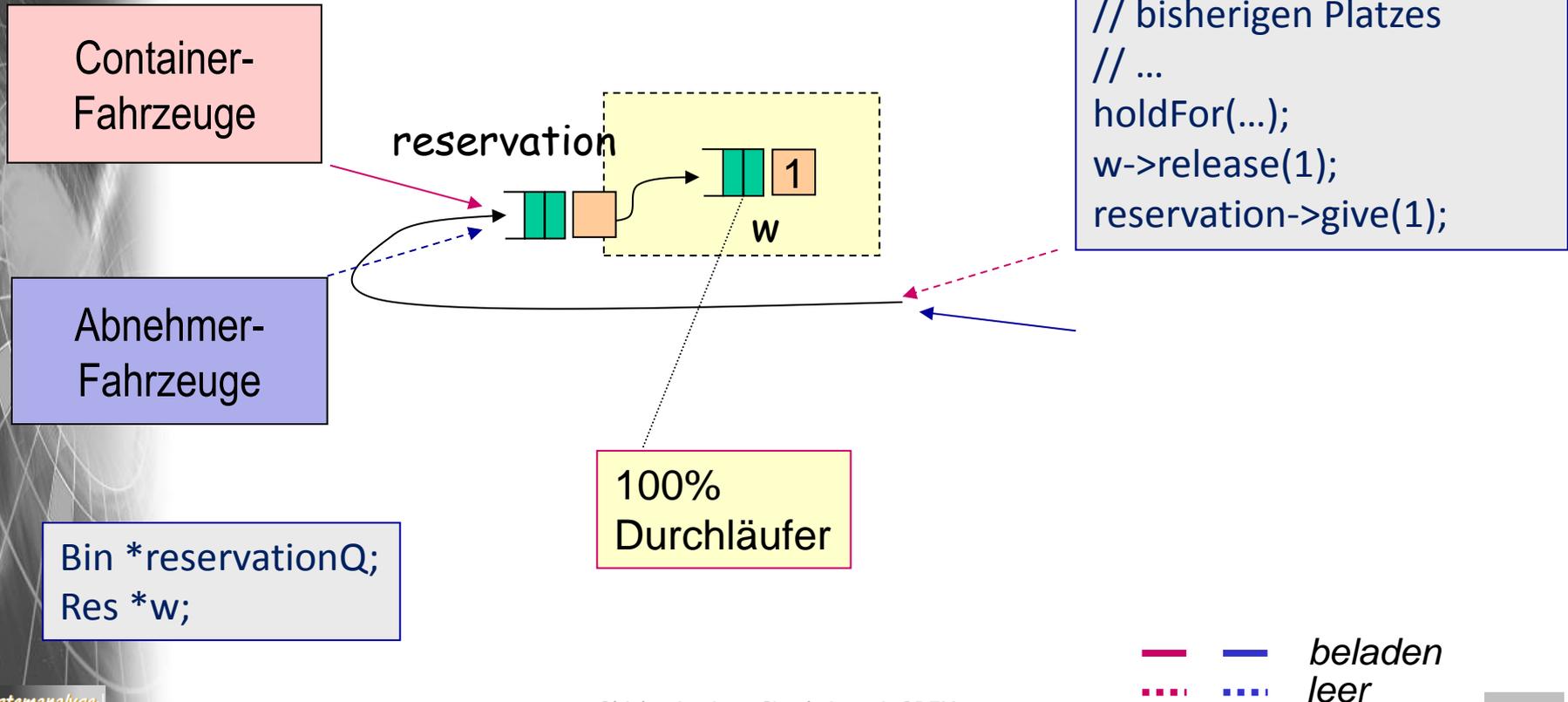
— beladen  
 ... leer

# Alternative Modellierung der Waage (3)

**Problem:** kein Platz vor und hinter der Waage

Lösung als Ensemble aus

- 1 Bin-Objekt (regelt Zugangsvergabe) und
- 1 Res-Objekt (Waage)



# Modellierung der Trichter und Produktion

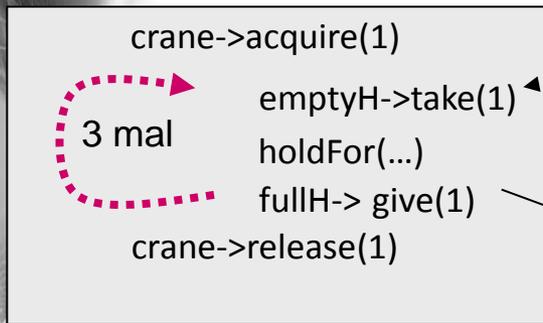
Trichter ~ Hopper

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

Kran

...als Res-Objekt

Container-Fahrzeug



leere Trichter

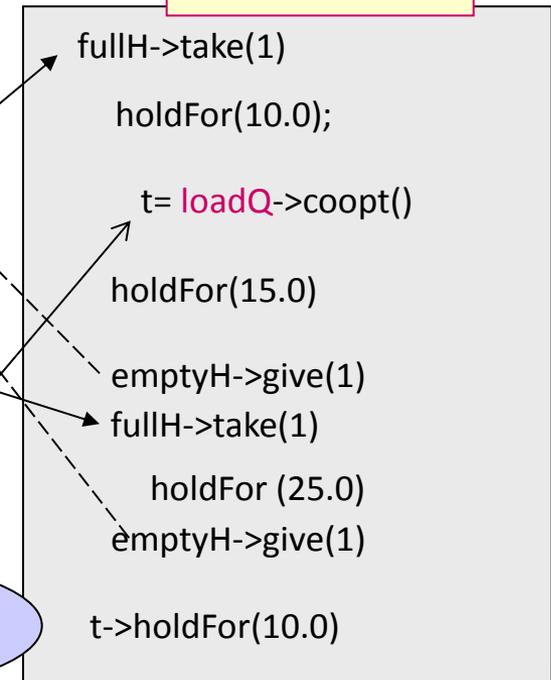
emptyH: Bin

volle Trichter

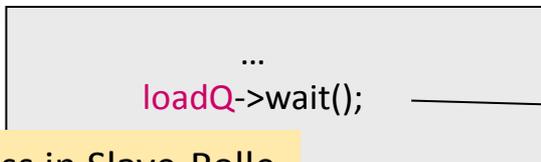
fullH: Bin

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

Production



Enprodukt-Fahrzeug



loadQ: WaitQ

...als Prozess in Slave-Rolle

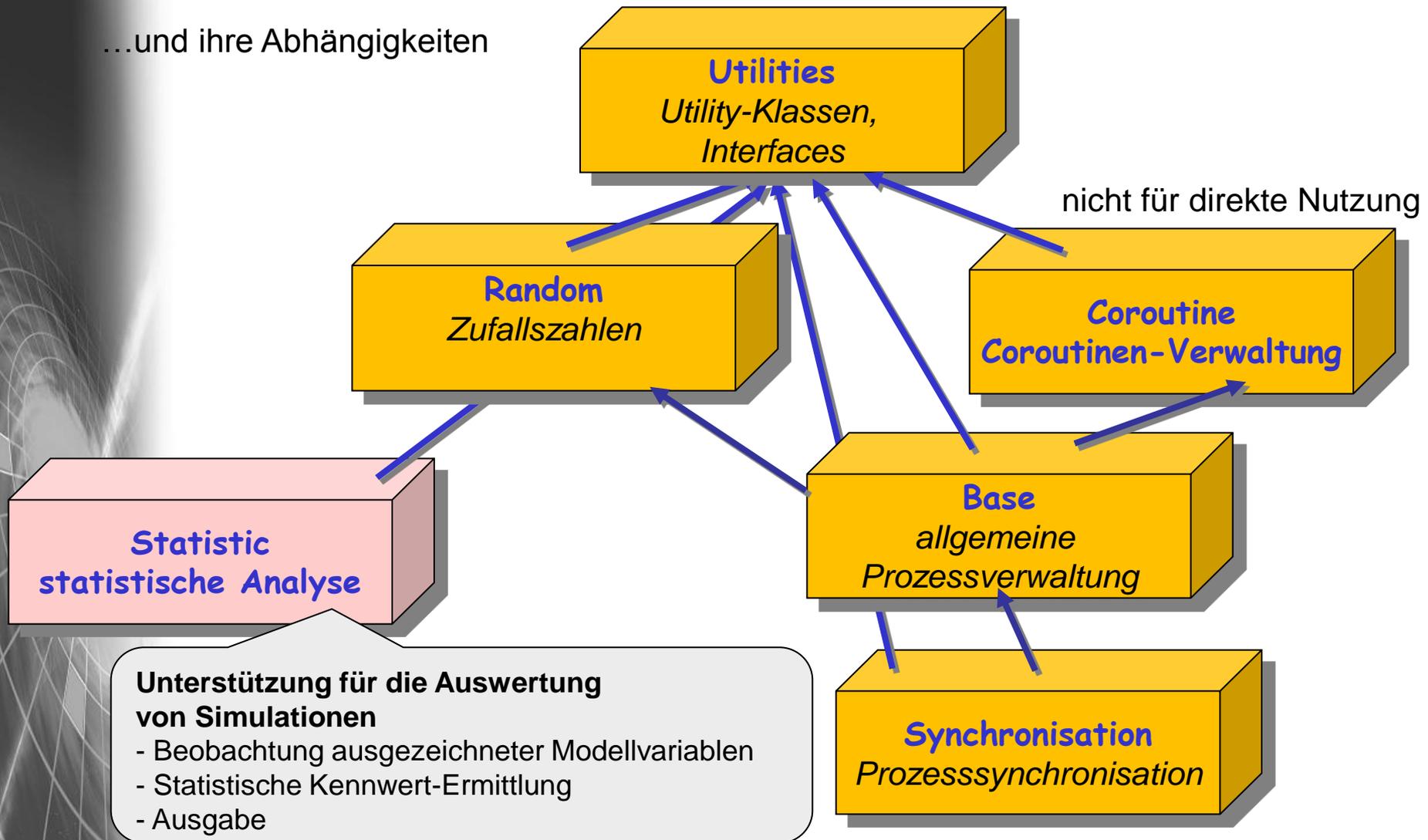
# **7. ODEMx-Modul Statistik:**

## ***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



# Modellauswertung

## Typischer Ablauf

1. Sammlung von Beobachtungsdaten je ausgezeichnete Modellvariable (Ergebnisgröße) in einem ersten Simulationslauf  $[0, t_{\max}]$

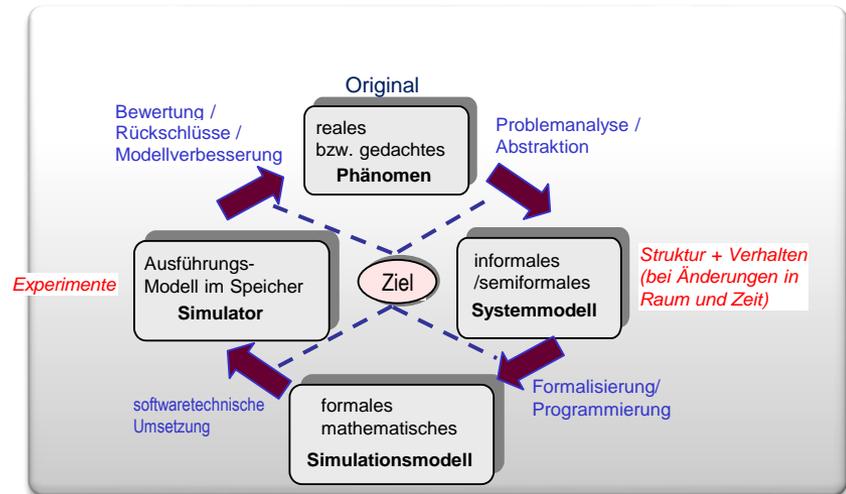
2. Verdichtung der gewonnenen Rohdaten zu statistischen Kenngrößen (Mittelwert, Streuung/Standardabweichung) und deren Speicherung

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 alle bisherigen Simulationsläufe

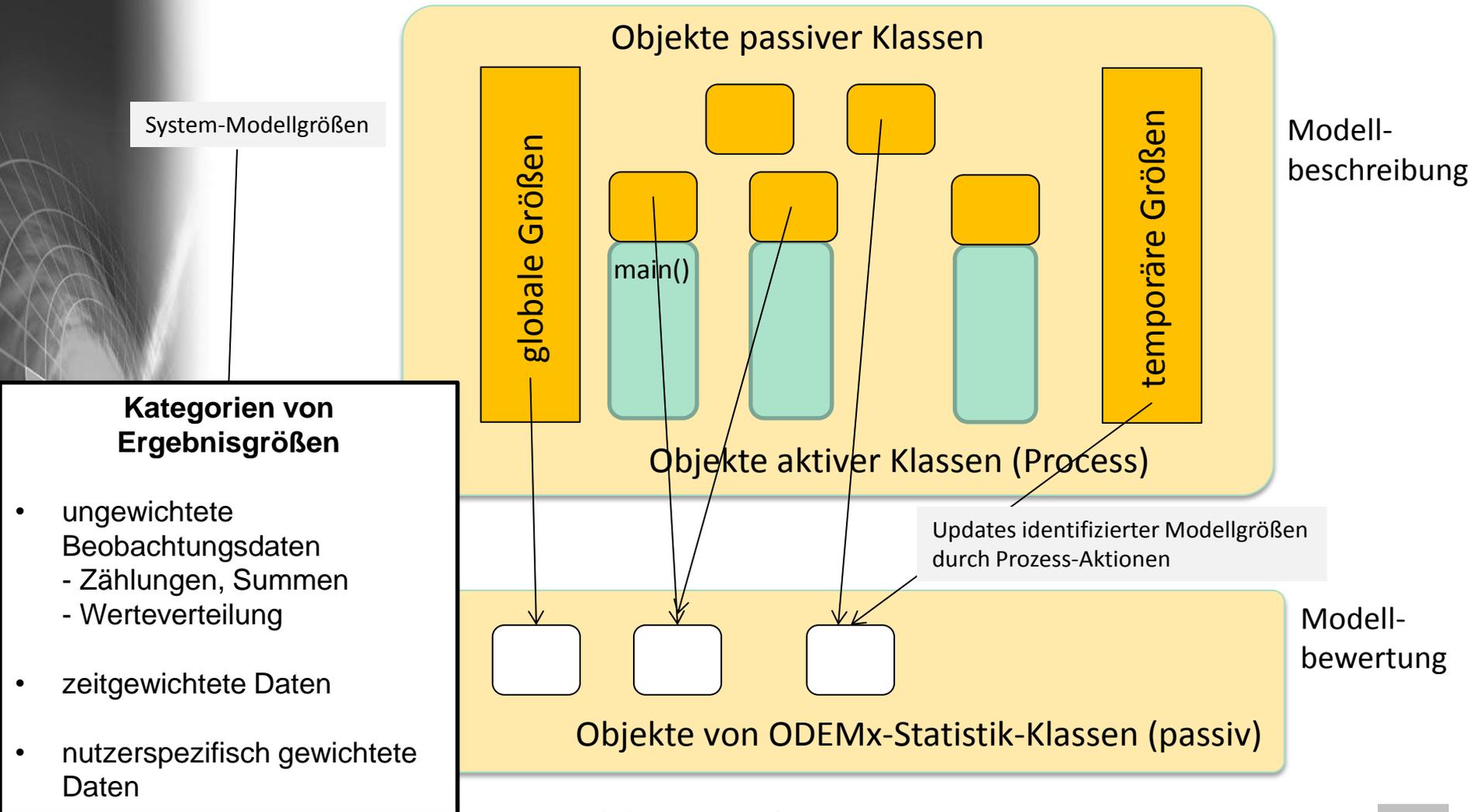
Hauptprogramm startet nacheinander verschiedene Kontexte mit dem gleichen Modell (aber unterschiedlichen Urstartwert für ZZ-Generatoren)

Sammelt Ergebnisdaten (als Stichprobe) ein und berechnet Kennwerte

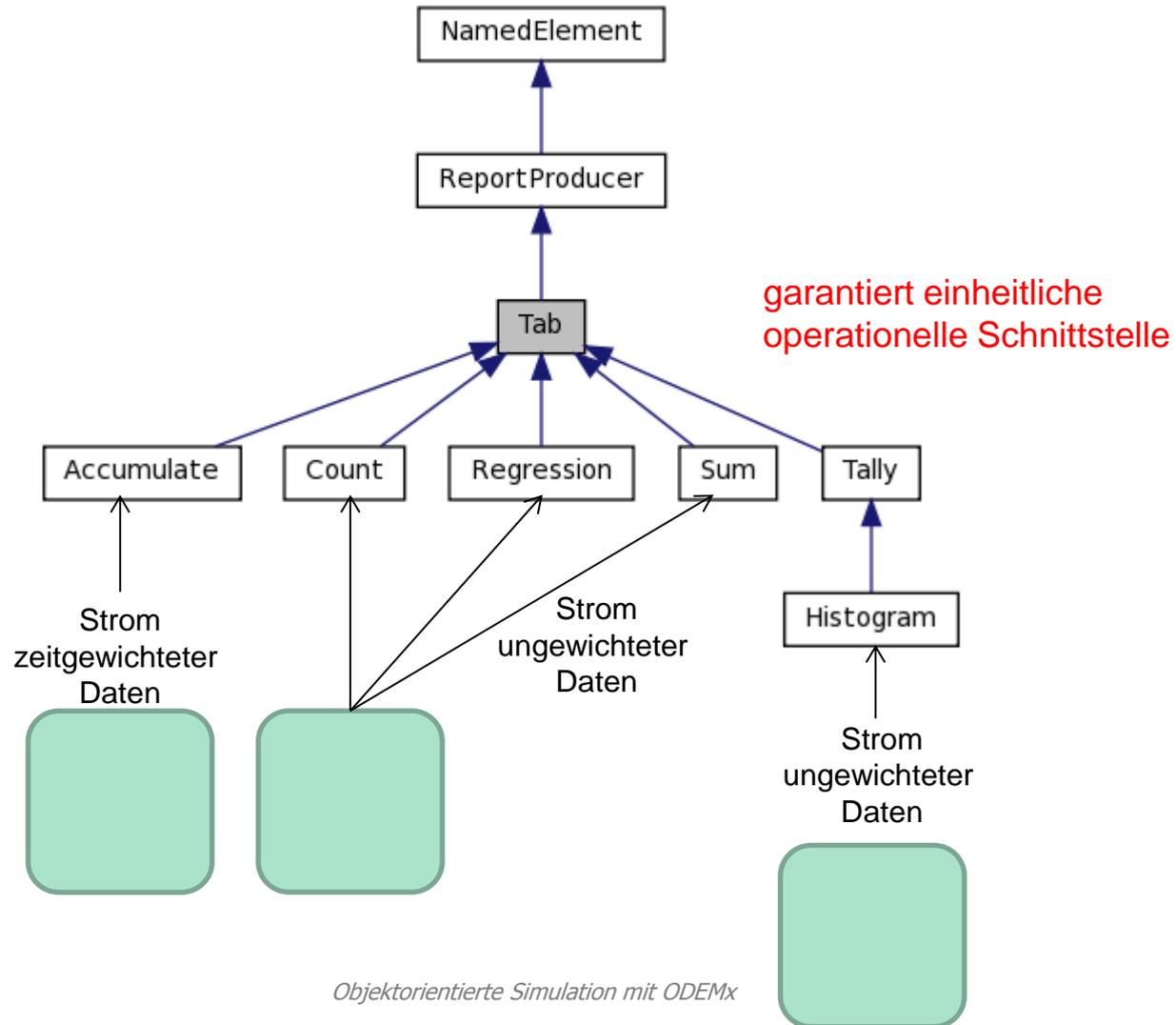


nicht mehr  
In diesem  
Semester !

# Erfassung von Beobachtungsdaten (Datenströme)



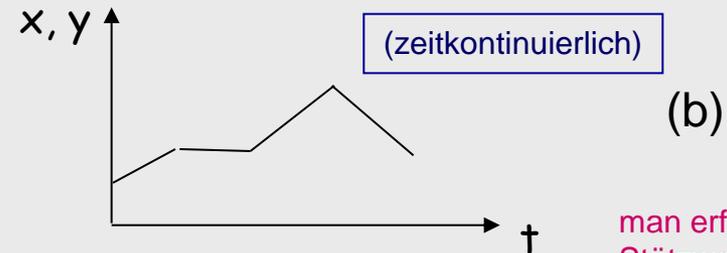
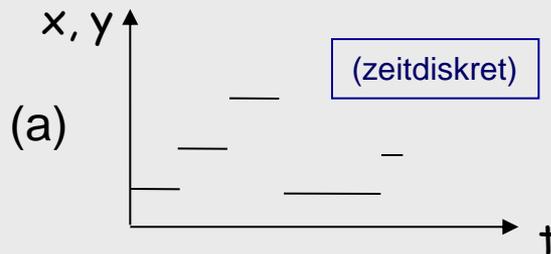
# Tab – abstrakte Basisklasse für alle Statistik-Klassen



# 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 z.B. lineare Übergänge an

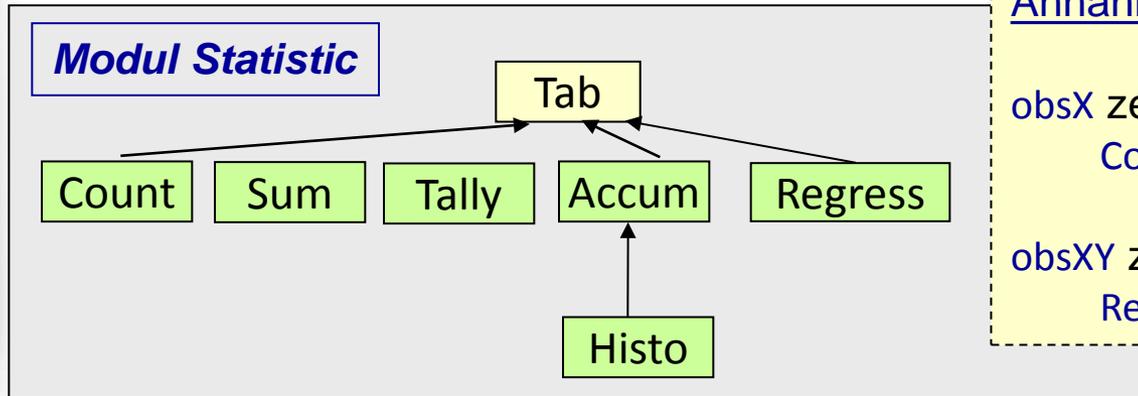
(1) Modellbeobachtungen:  $x_1, x_2, x_3, \dots$   $y_1, \dots$

(2) Modellbeobachtungen:  $(x_1, t_1), (x_2, t_2), (x_3, t_3)$

(3) Modellbeobachtungen:  $(x_1, y_1)^{t_1}, (x_2, y_2)^{t_2}, (x_3, y_3)^{t_3}, \dots$

**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 **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.  $y$  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**

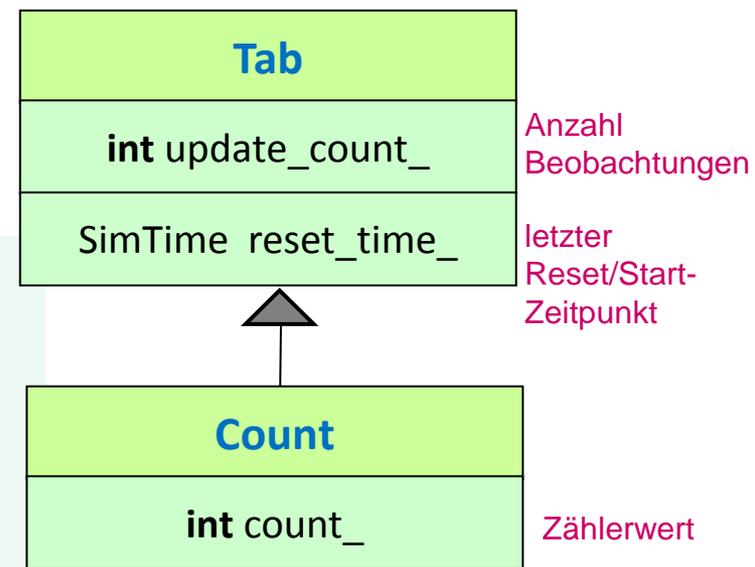
```
Simulation *sim;
```

```
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_;  
};
```

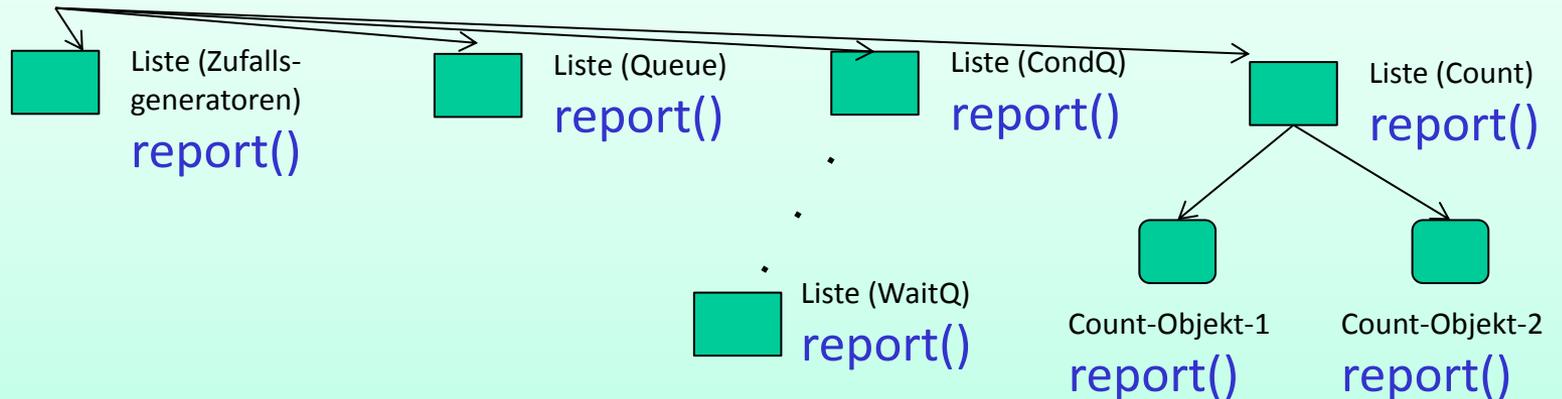


```
void Count::update( int value ) {  
    Tab::update();  
    count_ += value;  
}
```

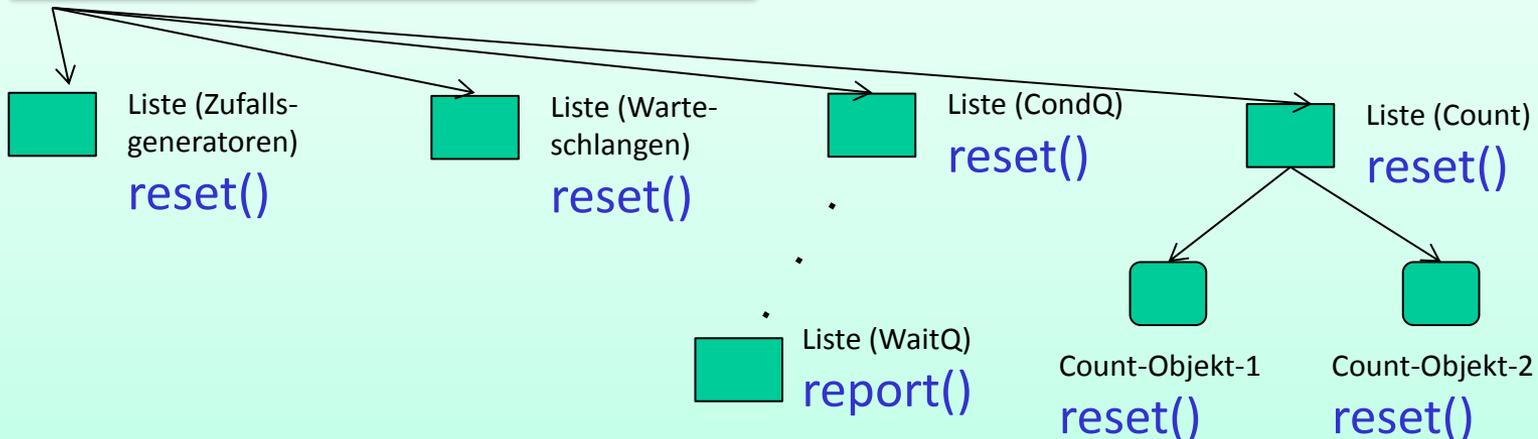
```
void Count::reset ( base::SimTime value ) {  
    Tab::reset(time);  
    count_ = 0;  
}
```

# 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);

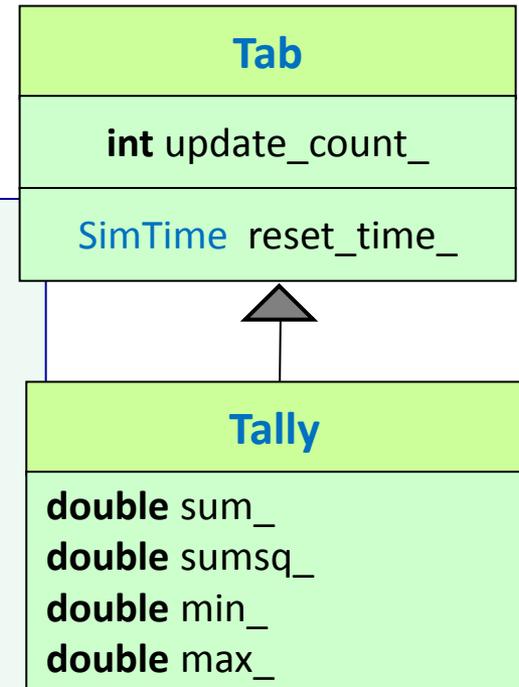
    virtual void reset(base::SimTime time);

    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

```
class Tally : public Tab {
public:
    Tally (base::Simulation* s, data::Label l);
    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;}

    double getDivergence();
    virtual void report (data::Report r);

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

Wieviel Speicherplätze werden zur Profilbestimmung benötigt ?

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

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

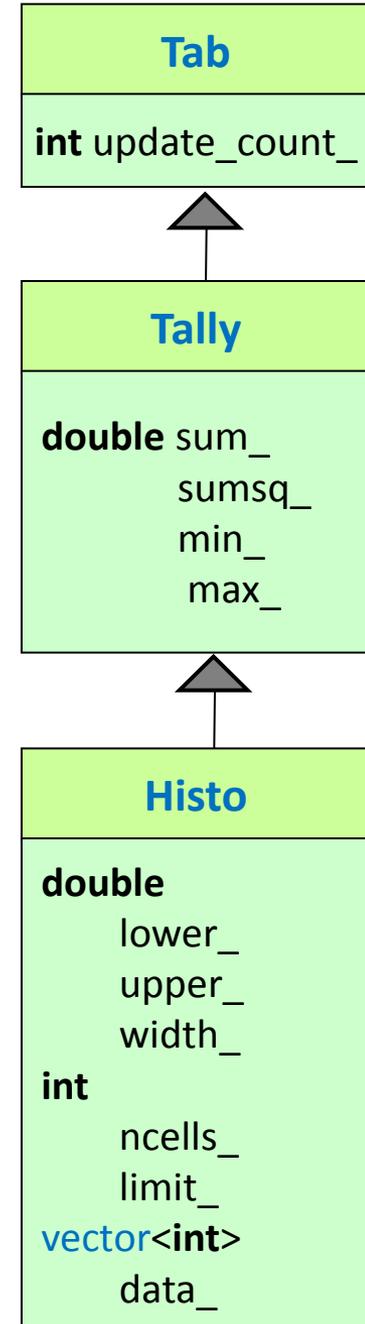
```
class Histo : public Tally {
public:
    Histo(Simulation* s, Label title,
          double low, double up, int n);
    virtual ~Histo();
    virtual void update(double v);
    virtual void reset(SimTime time);

    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

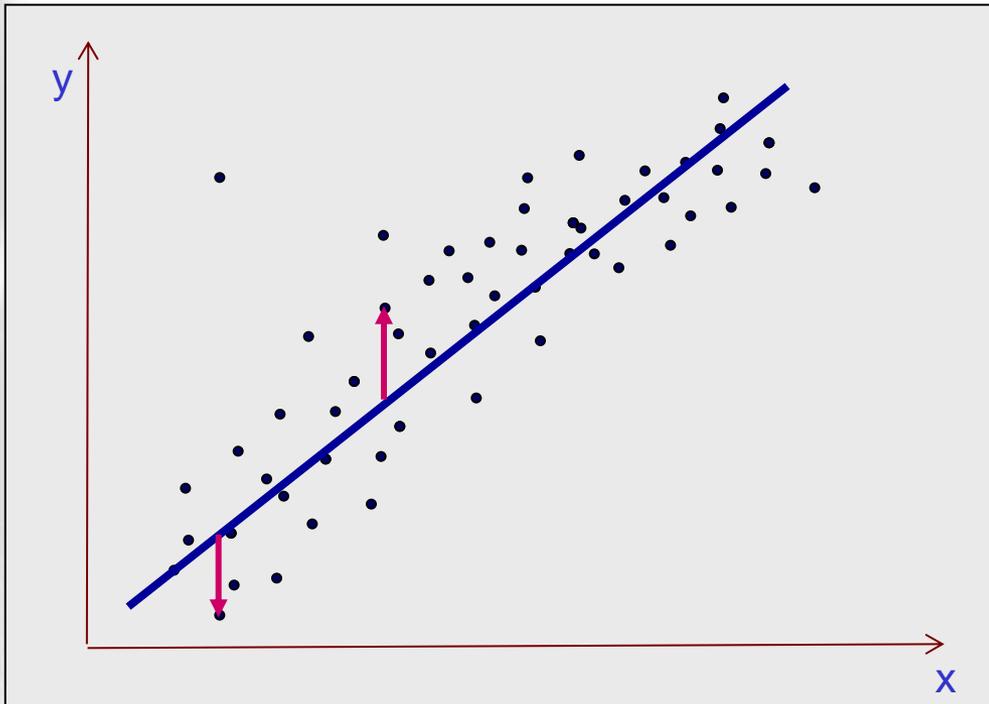
- **Vor.:** Paar zeitdiskreter Variablen vom Typ **double**
- **Funktion**  
Erfassung von Werteänderungen des Variablenpaares  $(x, y)$   
Bestimmung einer angenommenen linearen Abhängigkeit für ein Beobachtungsintervall
- **Kennwertprofil**  
**unabhängig von der jeweiligen Dauer einer Wertebelegung**  
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

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

## Regressionskoeffizient

$$-1 \leq r \leq 1$$

# Regressions- oder Korrelationskoeffizient

- **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