

Kurs OMSI im WiSe 2014/15

Objektorientierte Simulation mit ODEMx

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

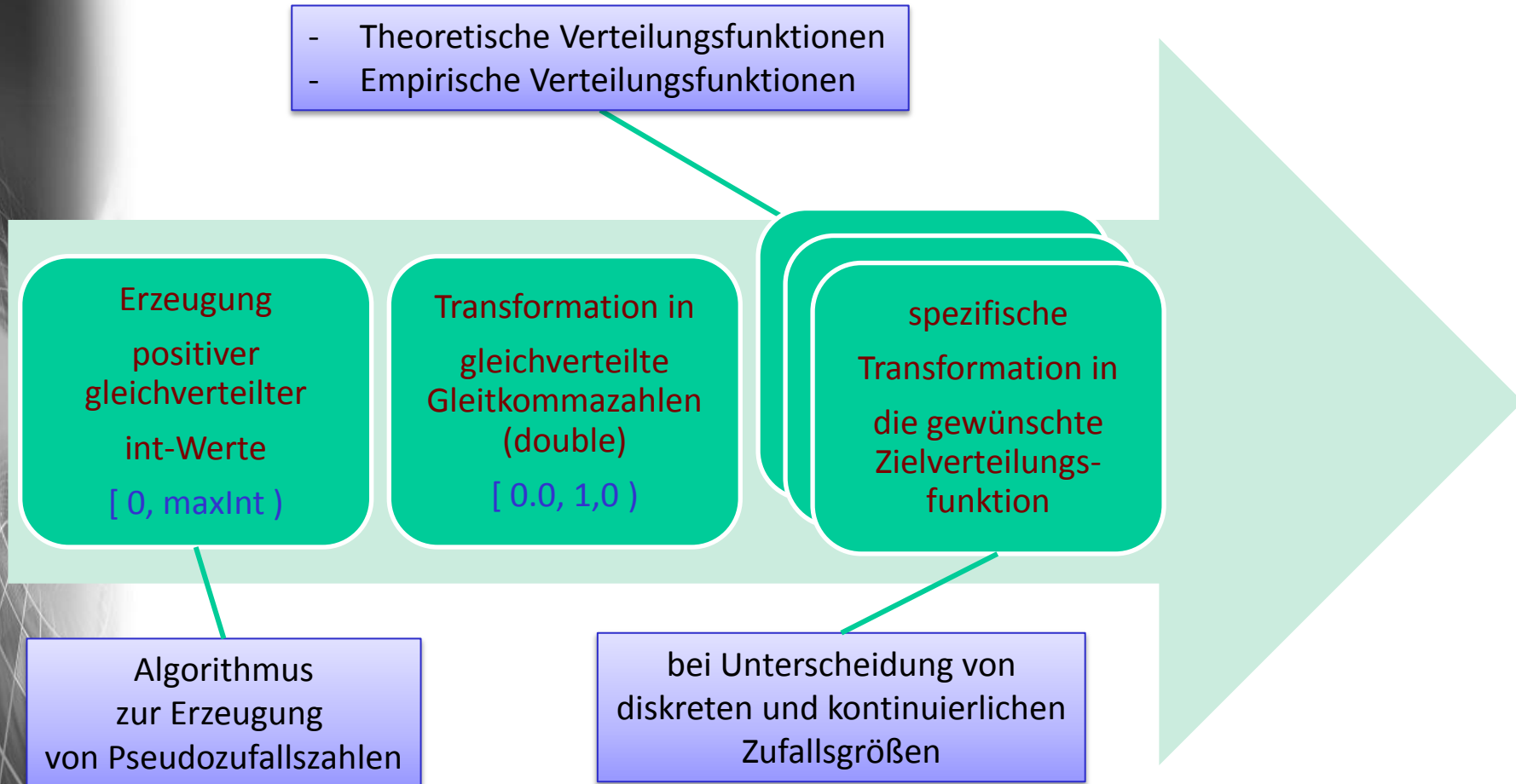
fischer|ahrens|eveslage@informatik.hu-berlin.de



5. ODEMx-Modul Random

1. Charakterisierung von Zufallsgrößen
2. Approximation von Zufallszahlen
3. ODEMx- Zufallszahlengeneratoren (Übersicht)
4. Einstellung von Startwerten
5. Protokollierung
6. Berechnung von Zufallszahlen ausgewählter Verteilungen

Schema zur Berechnung von Zufallszahlen



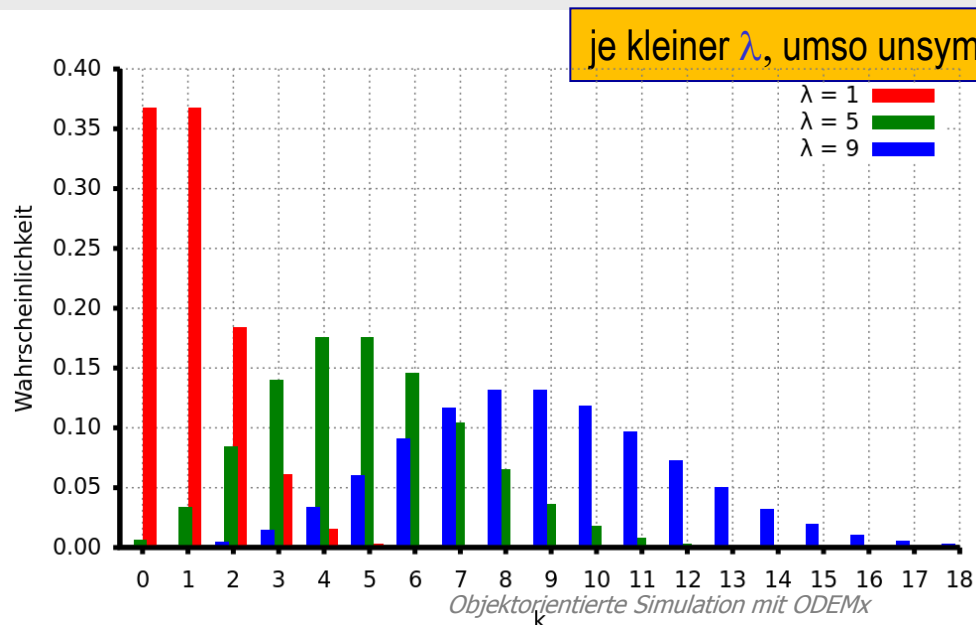
Diskrete Zufallsgrößen

Charakterisierung einer diskreten Zufallsgröße X :

- X beschreibt Ereignisregistrierungen (Anzahl) in einem bestimmten Zeitbereich (X nimmt bei jedem Versuch zufällig einen best. Wert an)
- Werte genügen einem bestimmten Typ von Verteilungsfunktion: **Poisson**
- λ ist die mittlere Anzahl (**Varianz $\sigma = \lambda$**)

Beispiel-Ereignisse

- Anzahl beobachteter Sternschnuppen in einer bestimmter Zeitspanne
- Anzahl registrierter Telefonanrufe in einer best. Zeitspanne
- Anzahl von **Ankünften von Kunden einer Service-Einrichtung** in best. Zeitspanne



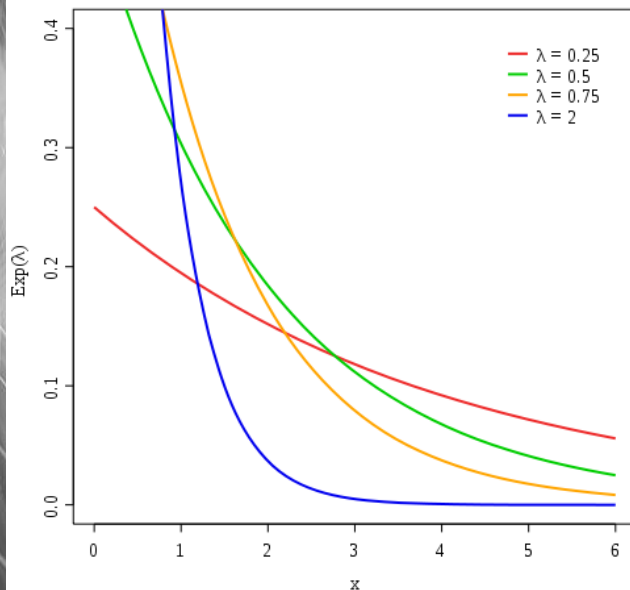
Stetige Zufallsgrößen

Beispiel für eine stetige Zufallsgröße X:

- X nimmt bei jedem Versuch zufällig einen bestimmten Wert x an
- Werte x_i genügen einer (negativen) Exponential-Verteilungsfunktion
- λ entspricht dabei einer Ereignisrate und $1/\lambda$ dem mittleren zeitlichen Abstand

Dichtefunktion $f(x) = \lambda e^{-\lambda x}$, falls $x \geq 0$, sonst 0

Verteilungsfunktion $F(x) = 1 - e^{-\lambda x}$, falls $x \geq 0$, sonst 0



- Zwischenankunftszeiten:
Zeitabstände zwischen Ankunftsereignissen von ... Ringstapeln in einer Schicht
- Dauer von Telefongesprächen
- Lebensdauer von Lebewesen, Maschinen, ...
- Modell für kleine und mittlere Schäden im Hausrat, für Kraftfahrzeug-Haftpflicht

$$\mu = 1/\lambda$$
$$\sigma^2 = 1/\lambda^2$$

5. ODEMx-Modul Random

1. Charakterisierung von Zufallsgrößen
2. Approximation von Zufallszahlen
3. ODEMx- Zufallszahlengeneratoren (Übersicht)
4. Einstellung von Startwerten
5. Protokollierung
6. Berechnung von Zufallszahlen ausgewählter Verteilungen

Pseudo-Zufallszahlen

Warum sind reproduzierbare Zufallszahlen enorm wichtig für die Simulation?

Iterationsverfahren

$x_0 \sim$ beliebig aus $[1, 10]$, z.B.: 2

$$x_{k+1} \equiv 2 * x_k \bmod 11 \quad (k=0, 1, 2, \dots)$$

→ Zahlenfolge mit Periode $p=10$:

$x_0 = 4, 8, 5, 10, 9, 7, 3, 6, 1, 2, 4, \dots$

Wiederholung

*In Abhängigkeit vom Startwert
ergibt sich eine periodische Folge von determinierten Zahlen,
die als Zufallszahlenfolge interpretiert werden kann*

Ein einfacher Pseudozufallszahlengenerator

```
class Random {  
    int u;  
    Random( int uu) u(uu) {  
        if u < 1 or u>10 { error(...); ...};  
    }  
    double next () {  
        u:= 2*u;  
        if u>11 { u= u-11; };  
        next= double(u) / 11.0;  
    }  
}
```

Werte aus dem Intervall [0.0, ..., 1.0)

Individuelle Startwerte: eigene Folgen von Zufallszahlen

```
Random *s1, *s2, *s3;  
s1= new Random(2); s2= new Random(9); s3 = new Random(7);
```

```
s1->next(); // .636, .273, .545, ...  
s2->next(); // .364, .727, .455, ...  
s3->next(); // .273, .545, .091, ...
```

Linearer Generator

Generator für gleichverteilte 26-Bit-Zufallswerte

$$q = 67.099.547 = 2^{26} - 1$$

Kongruenzmethode

- multiplikative Variante (Iterationsverfahren mit Startwert x_0)

$$x_{k+1} \equiv c x_k \bmod q$$

$$x_{k+1} \equiv 8.192 x_k \bmod 67.099.547 \quad (j = 0, 1, 2, 3, \dots)$$

→ Zahlenfolge mit **sehr großer** Periode **p = 67.099.546**:

$$x_0, x_1, x_2, \dots, x_{p-1}, x_p = x_0, x_1, x_2, \dots$$

Algorithmus für Startwerte (unabhängiger) Generatoren

$u_0 = 907$ (erster Startwert eines Generators)

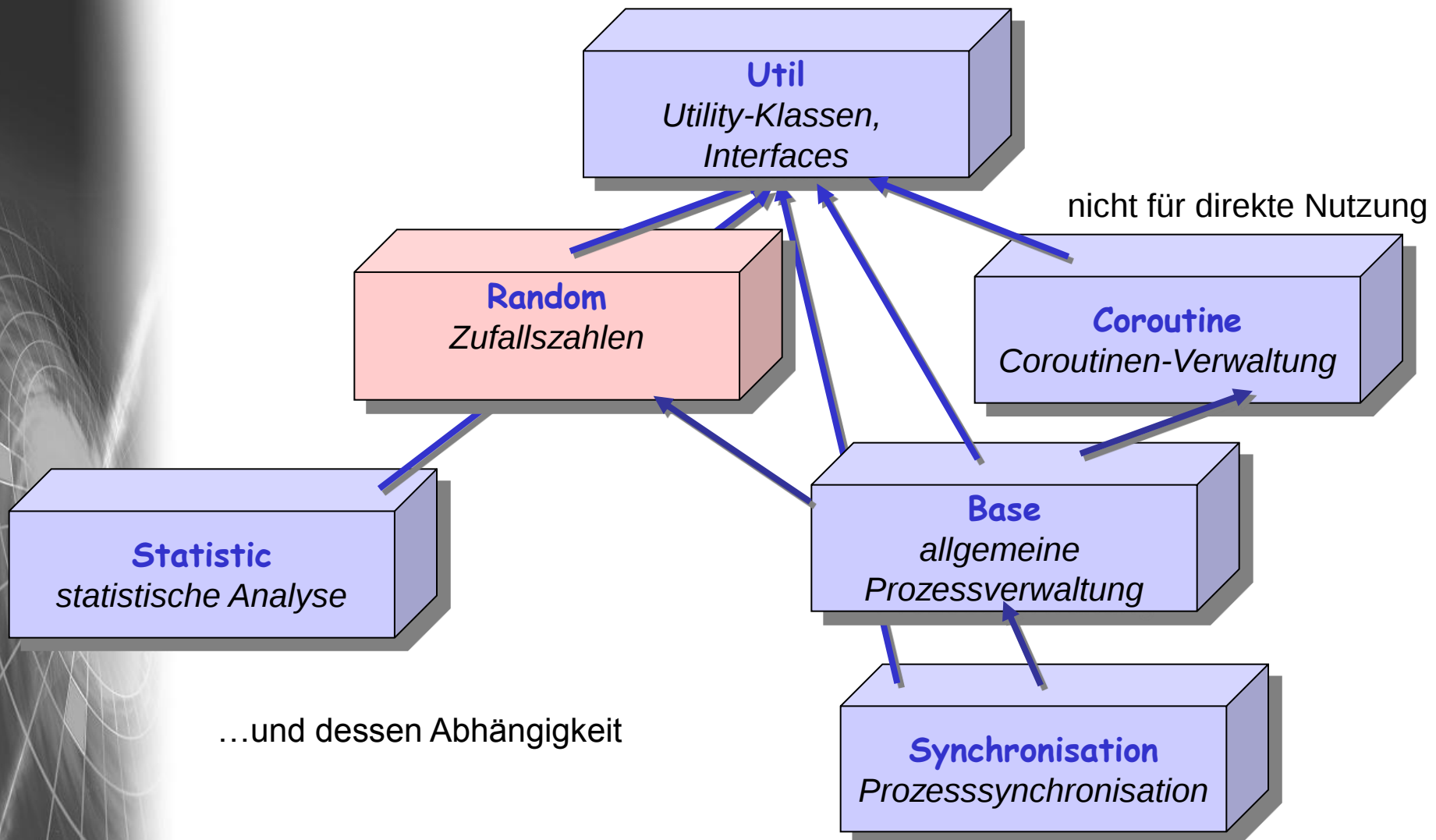
$$u_{i+1} \equiv 36.855 * u_i \bmod 67.099.547$$

→ 500 Generatoren für unabhängige Zahlenfolgen der Länge 120.000 (ohne dass eine Folge, in die andere „hineinläuft“)

5. ODEMx-Modul Random

1. Charakterisierung von Zufallsgrößen
2. Approximation von Zufallszahlen
3. ODEMx- Zufallszahlengeneratoren (Übersicht)
4. Einstellung von Startwerten
5. Protokollierung
6. Berechnung von Zufallszahlen ausgewählter Verteilungen

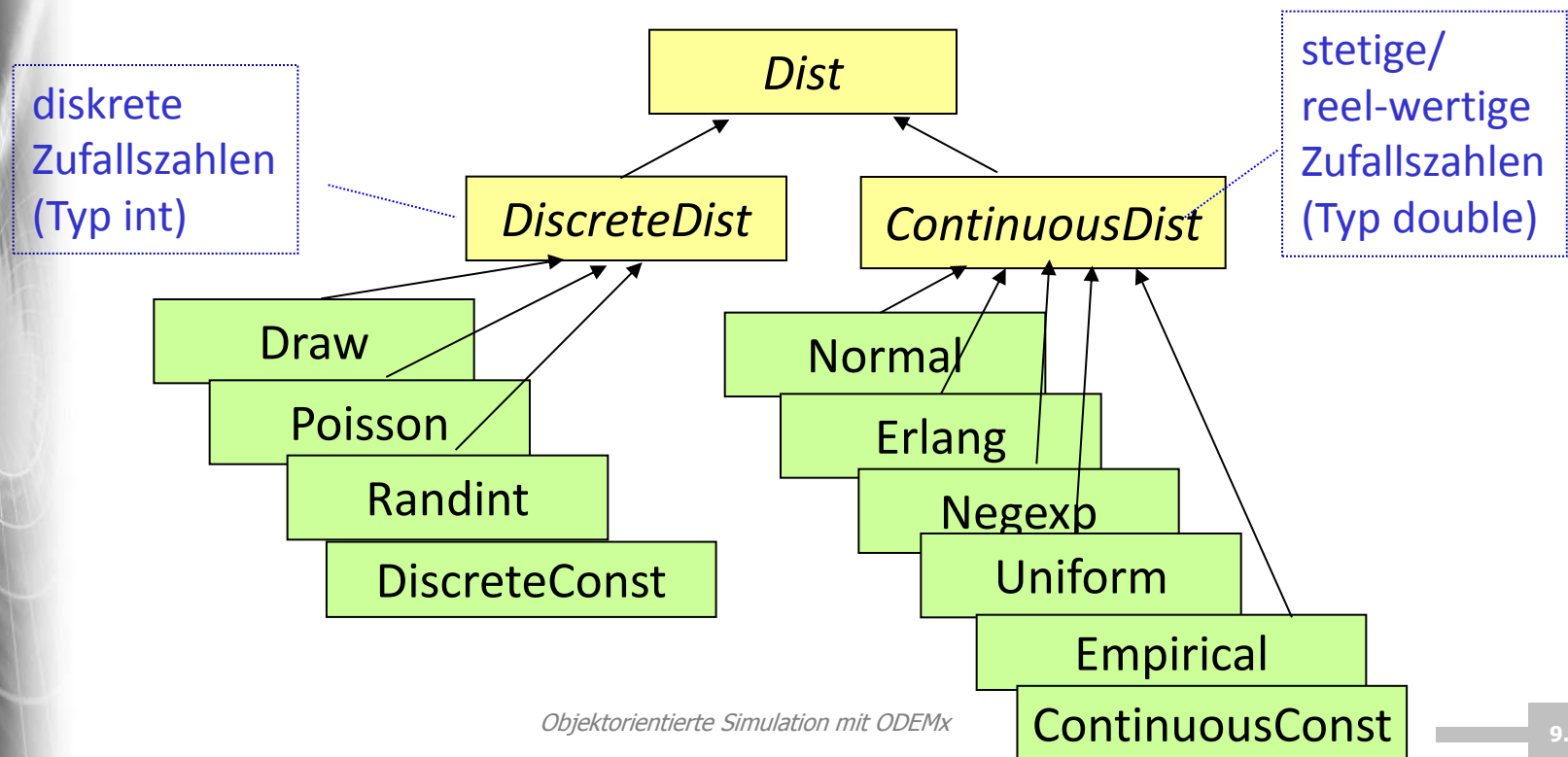
Der ODEMx- Modul Random



Klassen des Random-Moduls

Dist ist **abstrakte** Basisklasse aller Zufallszahlengeneratoren

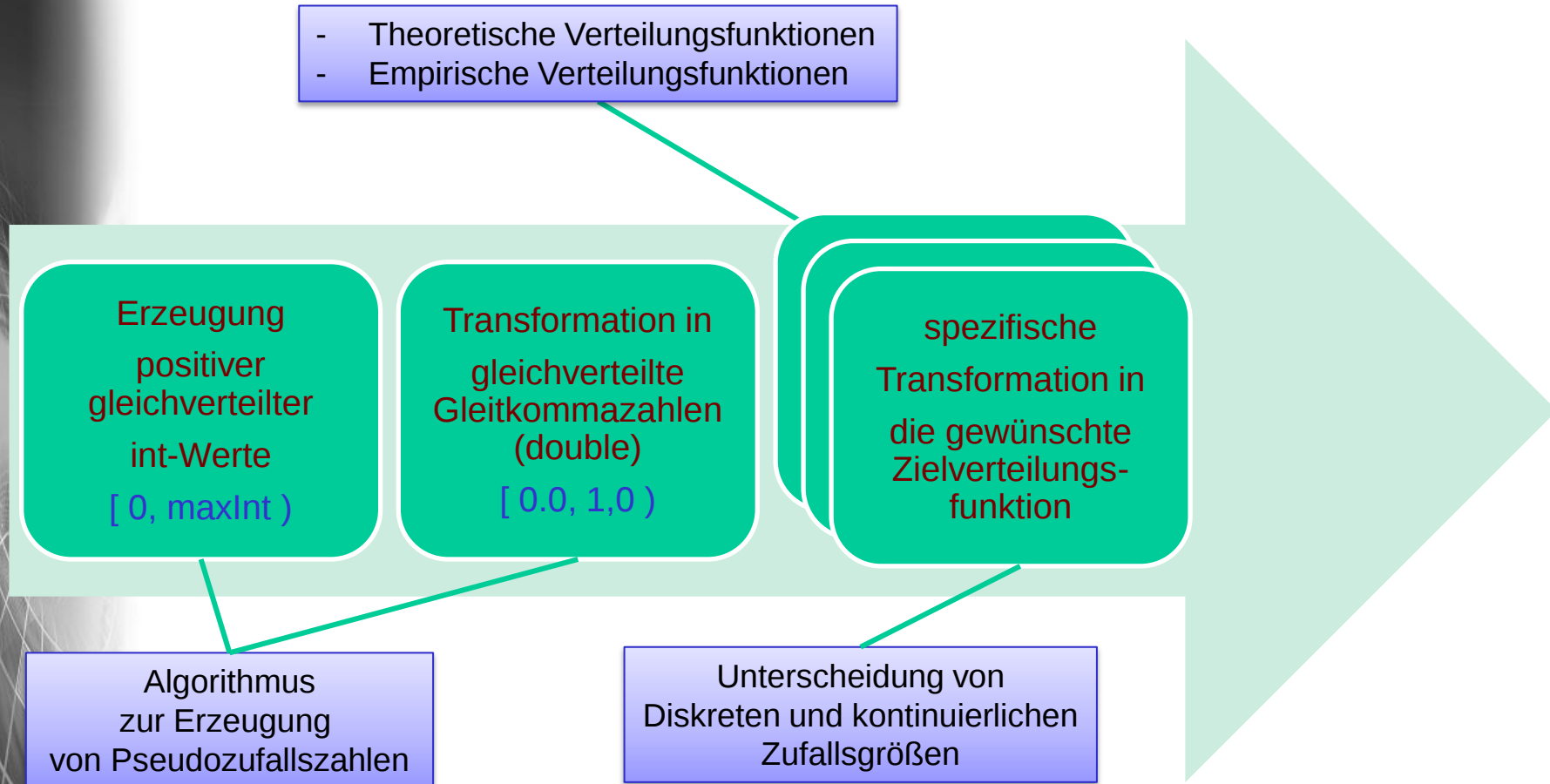
- bietet ganzzahligen Generator (lineares Kongruenzverfahren) für (0,1)- gleichverteilte Zufallsgrößen
- **Dist**- Ableitungen transformieren (0,1)- Folge in Folgen verschiedener Verteilungsfunktionen



5. ODEMx-Modul Random

1. Charakterisierung von Zufallsgrößen
2. Approximation von Zufallszahlen
3. ODEMx- Zufallszahlengeneratoren (Übersicht)
4. Einstellung von Startwerten
5. Protokollierung
6. Berechnung von Zufallszahlen ausgewählter Verteilungen
7. Lösung: Autofähre

Schema zur Berechnung von Zufallszahlen



5. ODEMx-Modul Random

1. Charakterisierung von Zufallsgrößen
2. Approximation von Zufallszahlen
3. ODEMx- Zufallszahlengeneratoren (Übersicht)
4. Einstellung von Startwerten
5. Protokollierung
6. Berechnung von Zufallszahlen ausgewählter Verteilungen
7. Lösung: Autofähre

(0,1)- Pseudo-Zufallszahlen in ODEMX

Iterationsverfahren mit Startwert x_0

$$x_{j+1} \equiv k x_j \bmod q \quad (j = 0, 1, 2, \dots)$$

liefert Zahlenfolge mit Periode p :

$$x_0, x_1, x_2, \dots, x_{p-1}, x_p = x_0, x_1, x_2, \dots$$

Generator für **gleichverteilte**
(31 Bit-) Zufallswerte
mit **maximaler Periode**

$$q = 2^{31} - 1,$$

$$k = 7^5 = 16.807,$$

$$p = 2^{31} - 2 = 2.147.483.646$$

Generator für **(0,1)-gleichverteilte** Zufallswerte

per Transformation : $y_i = x_i/q$

Urstartwert (aus Simulation-Context-Objekt)

Startwert für **ersten Generator** berechnen

Startwert x_0 für **zweiten Generator** berechnen

Berechnung von x_i und Transformation zu y_i

Berechnung von x_i und Transformation zu y_i

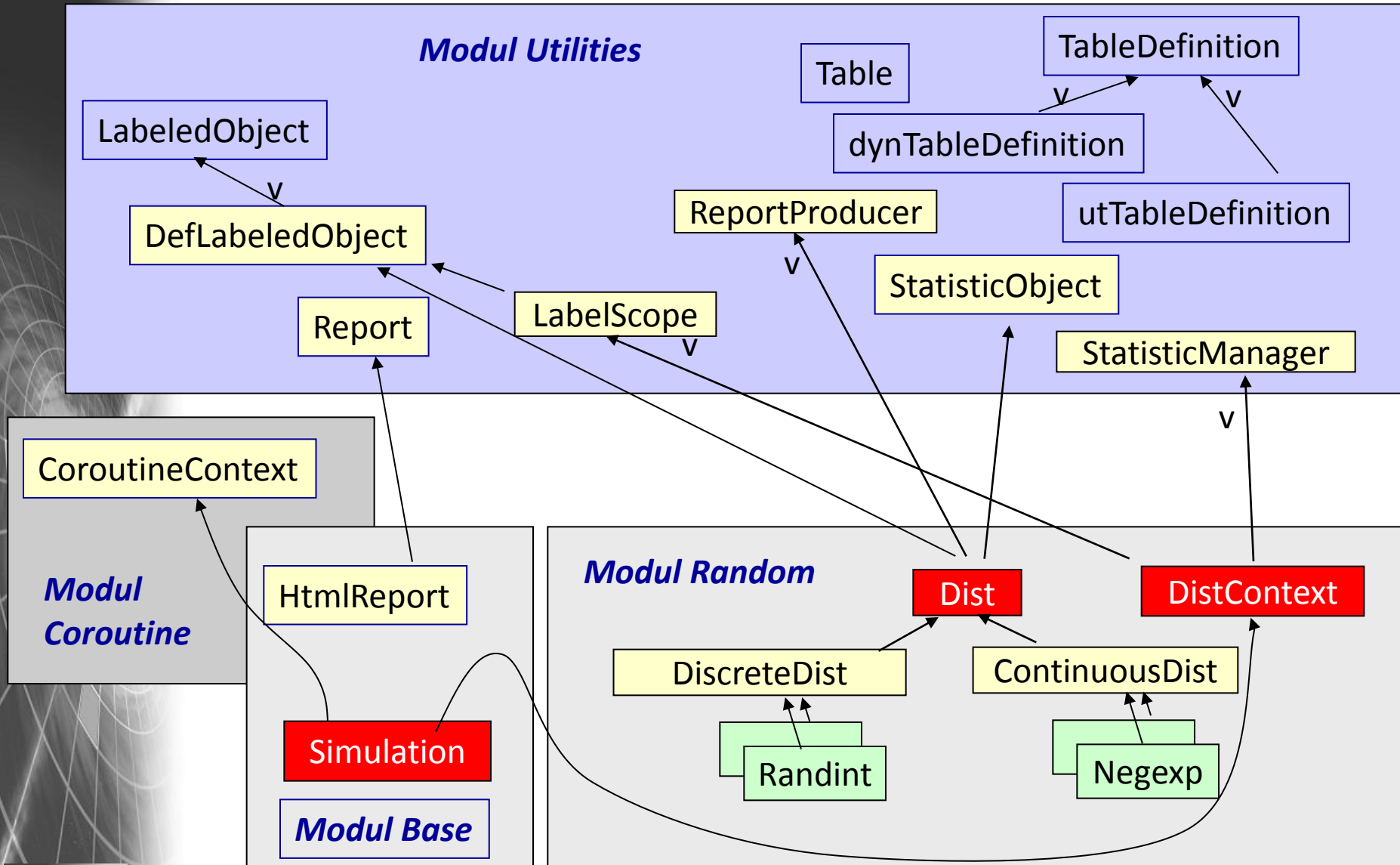
Transformation von y_i (0,1)-gleichverteilt zu z_i
(entspr. Zielverteilungsfunktion)

Transformation von y_i (0,1)-gleichverteilt zu z_i
(entspr. Zielverteilungsfunktion)

$i++$

$i++$

ZZ-Generatoren in der ODEMx-Klassenhierarchie



Abstrakte Klasse: Dist

... liefert Funktionalität zur Erzeugung einer (0,1)-gleichverteilten Zahlenfolge
ODEMx-ZZ-Generatoren sind Objekte von Dist- Ableitungen

```
class Dist : public DefLabeledObject, public StatisticObject,  
            public virtual ReportProducer {
```

```
protected:
```

```
    Dist(DistContext* c=0, Label label="");
```

```
    virtual ~Dist();
```

```
    ...
```

```
public:
```

```
    virtual void setSeed( int n = 0);
```

```
protected:
```

```
    double getSample(); //nächster (0,1)-Zufallswert
```

```
private:
```

```
    DistContext* context;
```

```
    unsigned long u, ustart;
```

```
    unsigned int antithetics;
```

Frage:
woher bezieht
ein Dist-Objekt
seinen individuellen
Startwert ?

Antwort: i.d.R.
iterativ von seinem
(Simulations-)kontext
über seinen Konstruktor
aus einem Ur-Startwert

oder
nutzerspezifisch

u wird mit
ustart initialisiert
und dann
iterativ verändert

DistContext (abstrakte Basisklasse von Simulation)

- Ein **Simulation**-Objekt stellt damit **auch** einen gemeinsamen Kontext für all seine (verschiedenen) ZZ-Generatoren dar
die Zuordnung vollzieht sich bei Generierung der ZZ-Generator-Objekte

```
class DistContext : public virtual LabelScope,  
                  public virtual StatisticManager {  
public:  
    DistContext ();  
    virtual ~DistContext();  
    unsigned long getNextSeed(); // berechnet weiteren neuen  
                                // Startwert im Kontext  
protected:  
    friend class Dist;  
    getSeed(); //liefert aktuellen Startwert  
private:  
    unsigned long zyqseed; //aktueller Startwert  
                        //bzw. Urstartwert  
};
```

$zyqseed^{k+1} = f(zyqseed^k)$

$zyqseed^0 = 907$
(UrGeneratorstartwert)

5. ODEMx-Modul Random

1. Charakterisierung von Zufallsgrößen
2. Approximation von Zufallszahlen
3. ODEMx- Zufallszahlengeneratoren (Übersicht)
4. Einstellung von Startwerten
5. Protokollierung
6. Berechnung von Zufallszahlen ausgewählter Verteilungen
7. Lösung: Autofähre

Report-File

Zeitpunkt der Reporterstellung

Bezug zum jeweiligen Simulationskontext

alle erzeugten ZZ-Generatoren des zugehörigen Kontextes

Simulation: DefaultSimulation					SimTime: 1318.53					
HtmlReport					ODEMx Version: 1.0					
Random Number Generators										
Name	Reset at	Type	Uses	Seed	Parameter 1	Parameter 2	Parameter 3			
mainland	0	Negexp	95	33427485	0.1	0	0			
island	0	Negexp	94	22276755	0.1	0	0			
crossing	0	Normal	72	46847980	7.5	0.5	0			
Queue Statistics										
Name	Reset at	Min queue length	Max queue length	Now queue length	Avg queue length					
mainland_queue	0	0	1	0	0					
island_queue	0	0	1	0	0					
ferry load_queue	0	0	1	0	0					
Bin Statistics										
Name	Reset at	Queue	Users	Provider	Init number of token	Min number of token	Max number of token	Now number of token	Avg number of token	Avg waiting time
mainland_1	0	mainland_queue	92	94	3	0	7	5	1.99533	0
island_1	0	island_queue	93	93	1	0	8	1	1.4375	0

Anzahl der Autos ~ 95-1 Autos

Anzahl der Überfahrten ~ 72 Überfahrten

es wurde nur eine Fähre eingesetzt

bislang keine Wartezeiten

Anzahl der Beladungen: 92

Anzahl der Aufrufe:
~ 95-1 Autos auf Mainland

Anzahl der Aufrufe:
~ 72 Überfahrten

es wurde nur
eine Fähre eingesetzt

bislang keine
Wartezeit auf Autos

Anzahl der Beladungen: 92

5. ODEMx-Modul Random

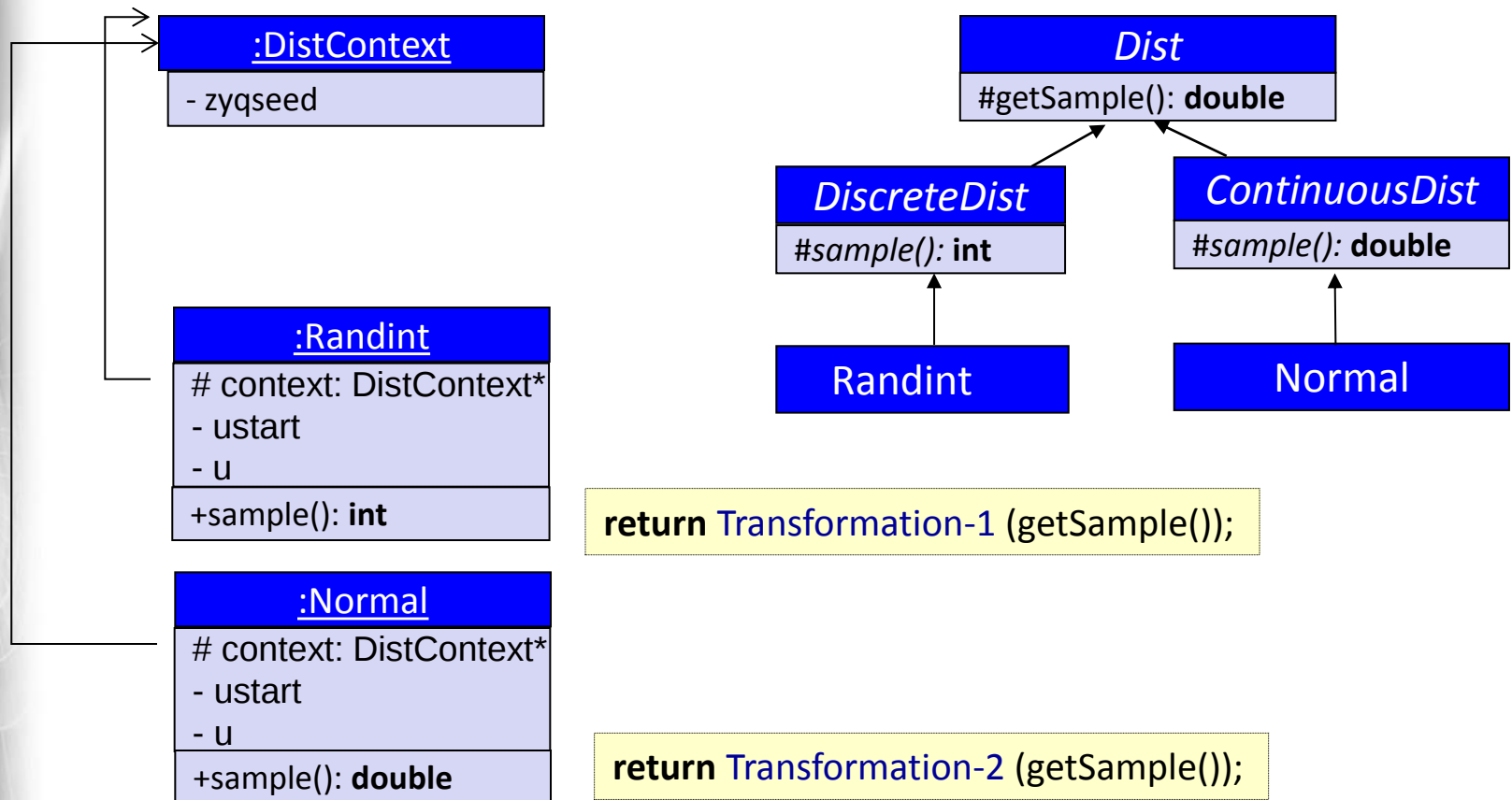
1. Charakterisierung von Zufallsgrößen
2. Approximation von Zufallszahlen
3. ODEMx- Zufallszahlengeneratoren (Übersicht)
4. Einstellung von Startwerten
5. Protokollierung
6. Berechnung von Zufallszahlen ausgewählter Verteilungen
7. Lösung: Autofähre

Schema zur Berechnung von Zufallszahlen

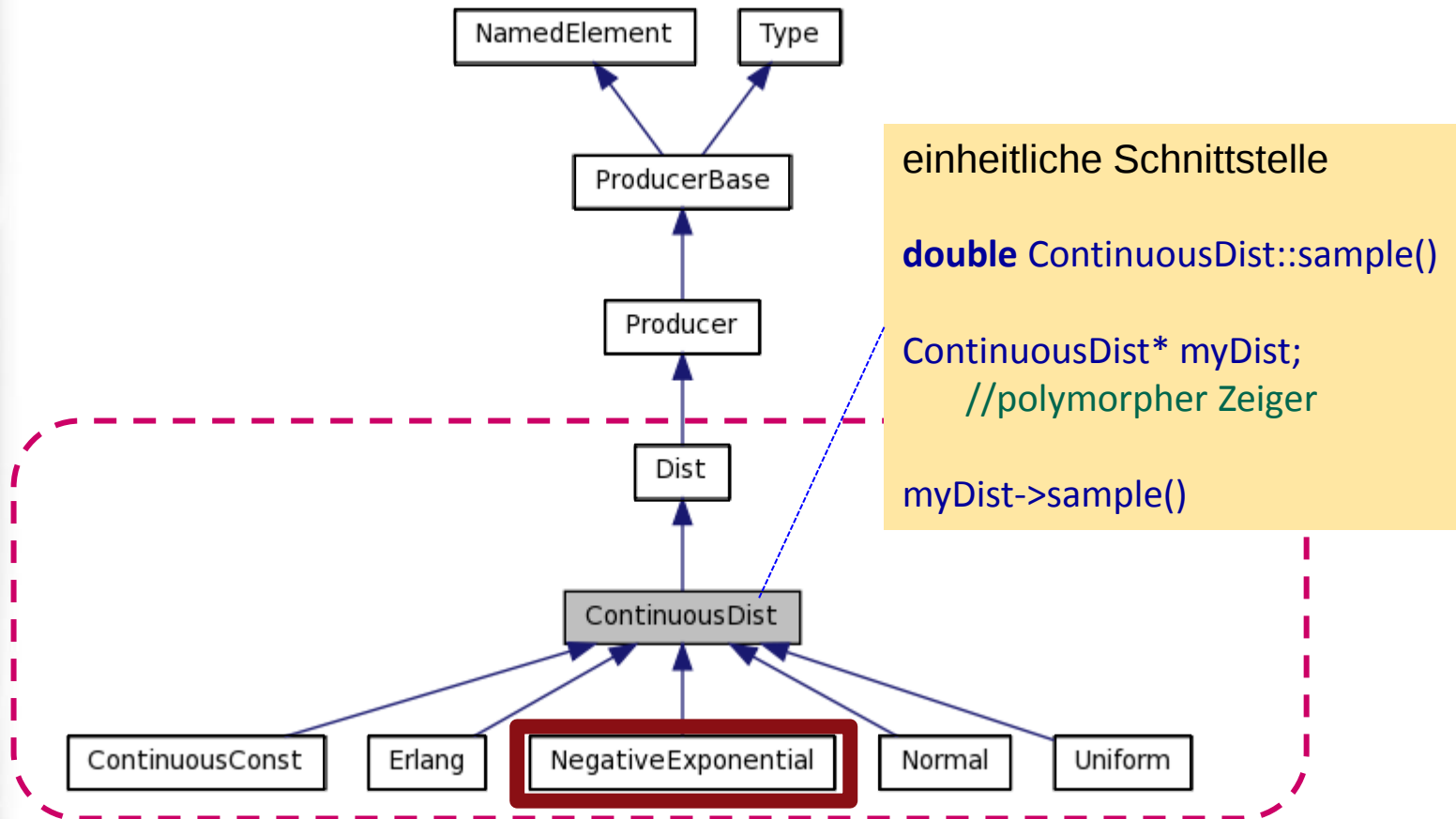
redefinierte Member-Funktion `sample()` einer

- `DiscreteDist`- bzw.
- `ContinuousDist`-Ableitung

transformiert gleichverteilte (0,1)-Folge von `Dist`



Stetige Verteilungsfunktionen



Generator für *exponentialverteilte* Pseudo-Zufallszahlen

Transformationsgenerator für **exponential- verteilte** Zufallswerte
mit Erwartungswert $1/\lambda$

(Dichtefunktion: $f(x) = \lambda e^{-\lambda x}$)

$\{y_i\}$ sei (0, 1)- verteilte Zufallszahlenfolge von Dist

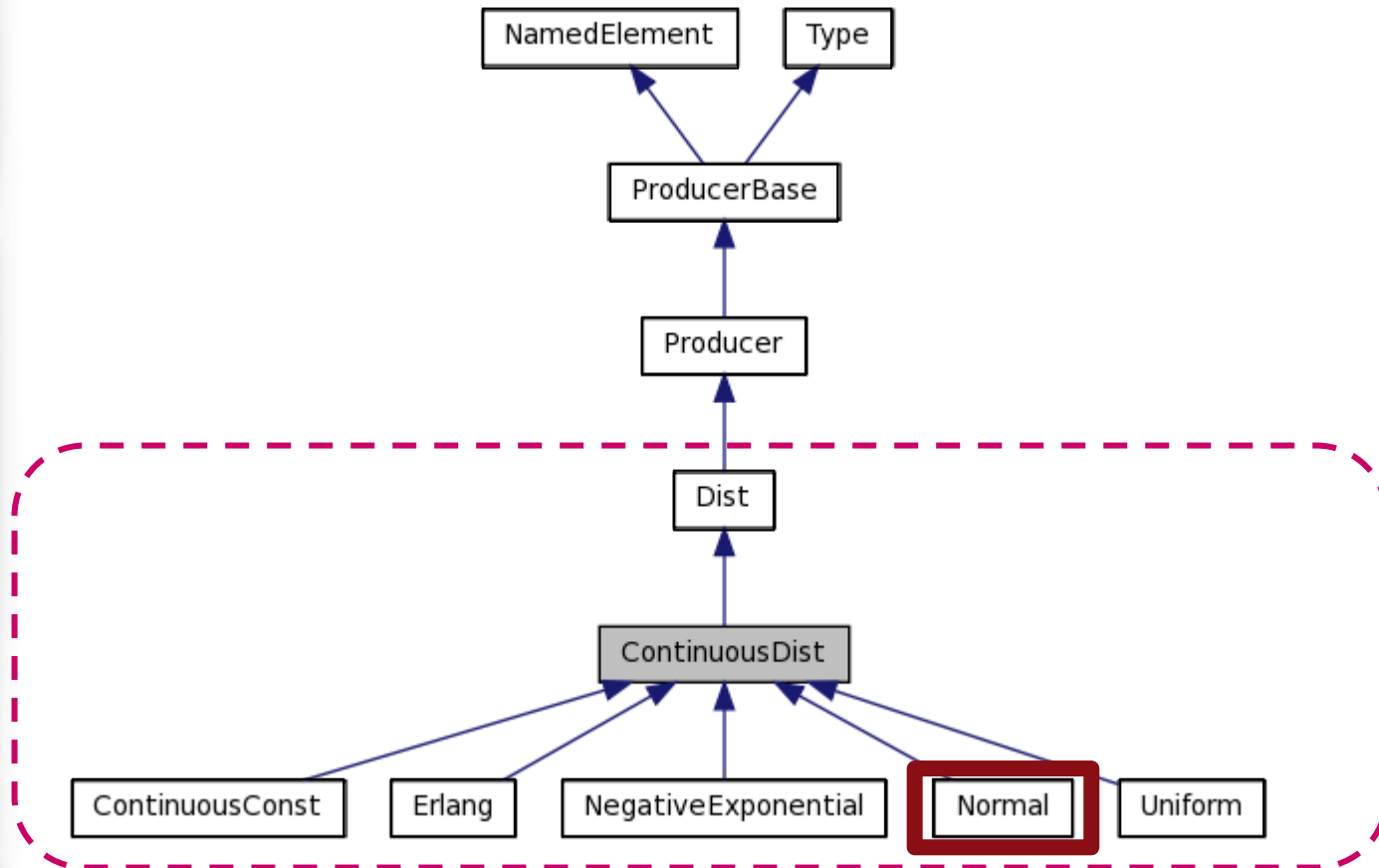
$$x_i = (-1/\lambda) * \ln(1 - y_i) \quad (i = 0, 1, 2, 3, \dots)$$

$$x_i = (-1/\lambda) * \ln(y_i) \quad (i = 0, 1, 2, 3, \dots)$$

→ $x_0, x_1, x_2, \dots, x_{p-1}, x_p = x_0, x_1, x_2, \dots$
exponential-verteilte Zufallswerte mit Erwartungswert $1/\lambda$

Parameter vom **negexp**-Konstruktor ist jeweilige die mittlere Zwischenankunftszeit

Stetige Verteilungsfunktionen



Generator für *normalverteilte* Pseudo-Zufallszahlen

*Transformation
erfolgt über
mehrere Schritte*

ODEMx-Lösung:

seien y_i und y_{i+1} zwei aufeinander folgende Werte einer (0, 1)-verteilten Zufallszahlenfolge, dann entsteht nach folg. Alg. eine neue Folge

$$x_i = \sqrt{-2 \ln(y_i)} \sin(2\pi y_{i+1})$$

$$z_i = \mu + \sigma x_i$$

→ $x_0, x_1, x_2, \dots, x_{p-1}, x_p = x_0, x_1, x_2, \dots$
normal-verteilte Zufallswerte
mit Erwartungswert $\mu = 0.0$ und
Standardabweichung $\sigma = 1.0$

normal-verteilte
Zufallswerte
mit Erwartungswert μ
und
Standardabweichung σ

Random Variable	#Observed	Mean or -Value	Std Dev or -Error	Sig. Digits	Minimum	Maximum
norm_full	100000	0.483	5.014		-21.79	22.82
Lower	Upper	Frequency	Percent			
-15.0	-14.0	76	0.076			
-14.0	-13.0	160	0.160			
-13.0	-12.0	255	0.255	*		
-12.0	-11.0	469	0.469	**		
-11.0	-10.0	735	0.735	***		
-10.0	-9.0	1099	1.099	*****		
-9.0	-8.0	1619	1.619	*****		
-8.0	-7.0	2275	2.275	*****		
-7.0	-6.0	3092	3.092	*****		
-6.0	-5.0	3874	3.874	*****		
-5.0	-4.0	4800	4.800	*****		
-4.0	-3.0	5829	5.829	*****		
-3.0	-2.0	6635	6.635	*****		
-2.0	-1.0	7313	7.313	*****		
-1.0	0.0	7748	7.748	*****		
0.0	1.0	8087	8.087	*****		
1.0	2.0	7802	7.802	*****		
2.0	3.0	7342	7.342	*****		
3.0	4.0	6552	6.552	*****		
4.0	5.0	5885	5.885	*****		
5.0	6.0	4759	4.759	*****		
6.0	7.0	3785	3.785	*****		
7.0	8.0	2926	2.926	*****		
8.0	9.0	2210	2.210	*****		
9.0	10.0	1652	1.652	*****		

leistet ODEMx
nicht

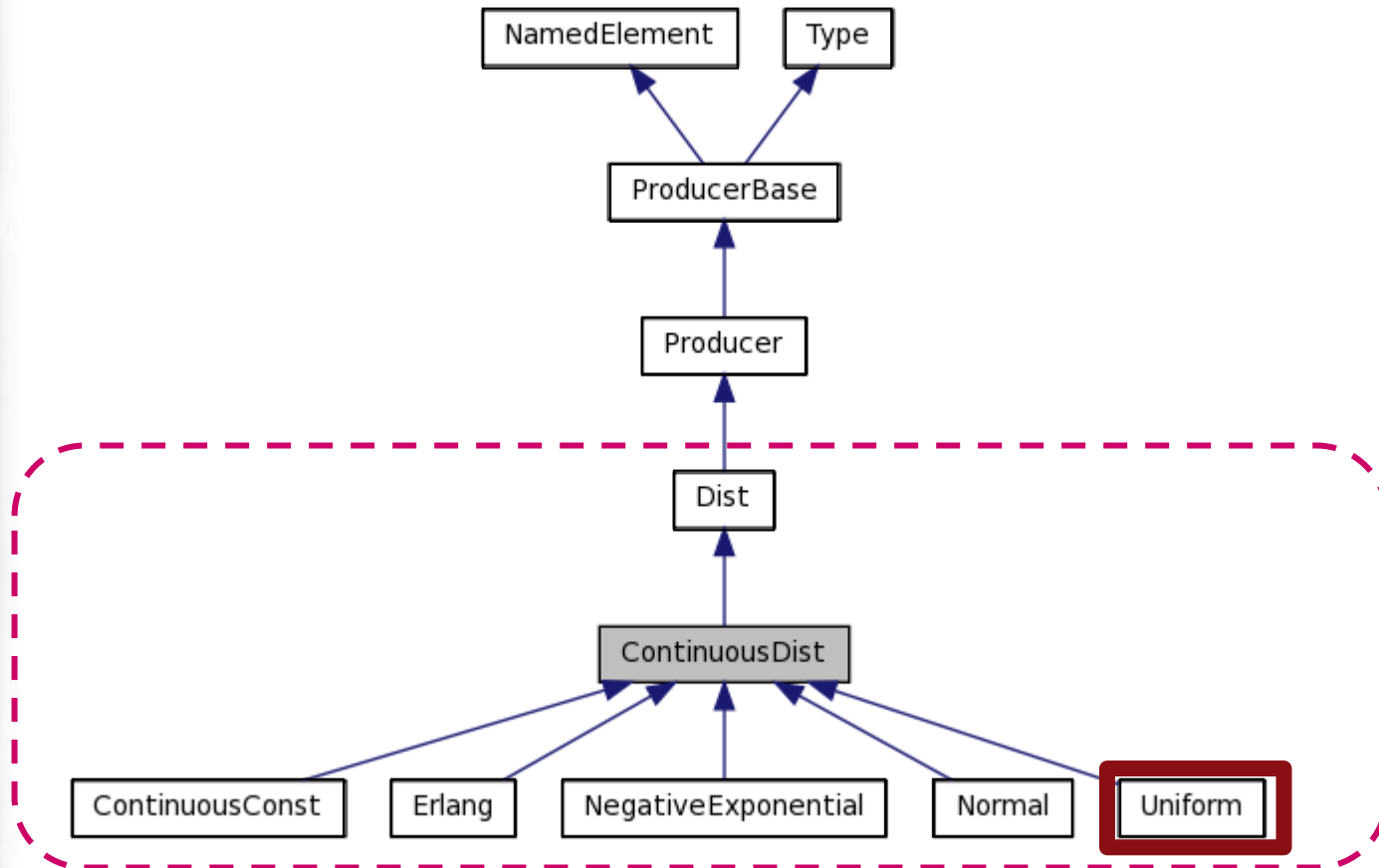


Underflow: 109
 Overflow: 197
 Average Underflow: -16.50
 Average Overflow: 16.44

falls Zufallsgröße negativ,
dann verwerfen und
neuen Wert generieren
solange bis positives Resultat

Random Variable	#Observed	Mean or -Value	Std Dev or -Error	Sig. Digits	Minimum	Maximum
norm_half	100000	3.732	2.551		0.00	10.00
Lower	Upper	Frequency	Percent			
0.0	1.0	15859	15.859	*****		
1.0	2.0	15387	15.387	*****		
2.0	3.0	14290	14.290	*****		
3.0	4.0	12936	12.936	*****		
4.0	5.0	11246	11.246	*****		
5.0	6.0	9397	9.397	*****		
6.0	7.0	7514	7.514	*****		
7.0	8.0	5805	5.805	*****		
8.0	9.0	4389	4.389	*****		
9.0	10.0	3177	3.177	*****		

Stetige Verteilungsfunktionen



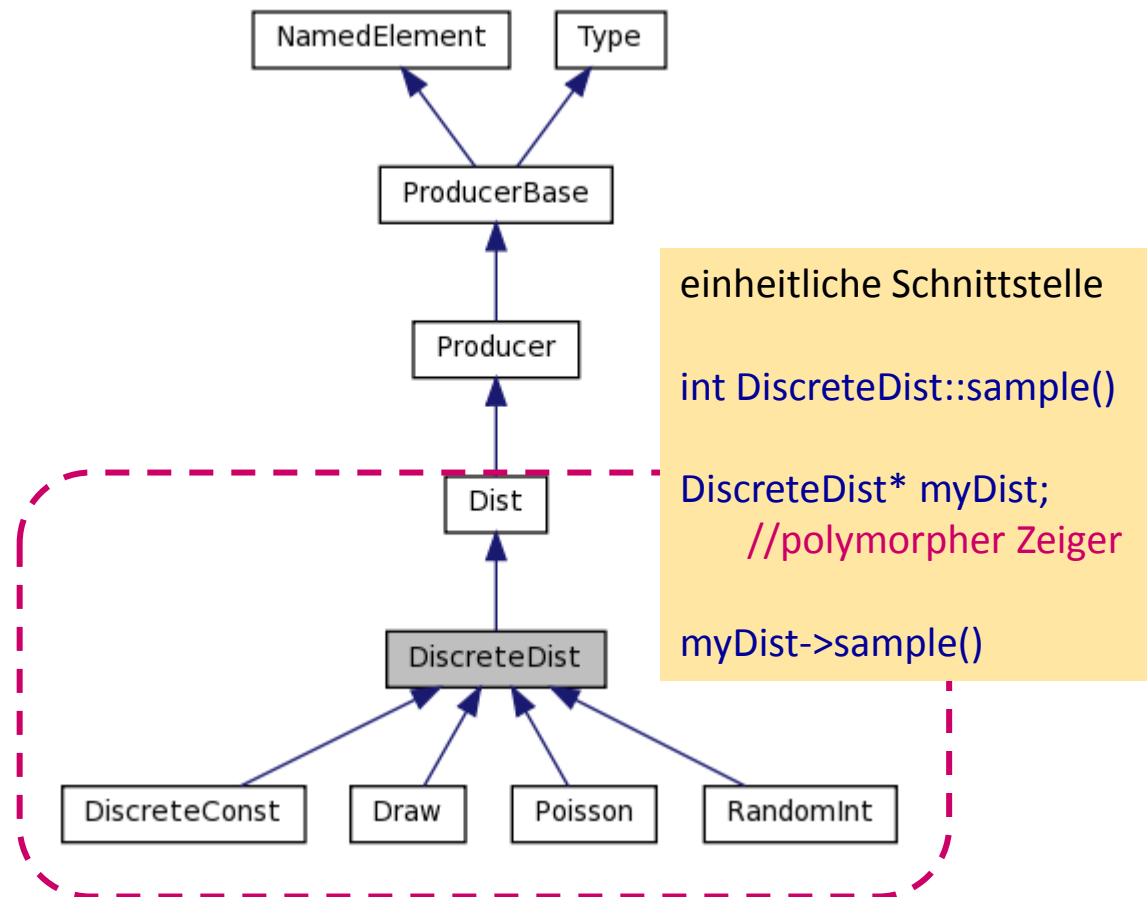
Generator für *gleichverteilte reelle Pseudo-Zufallszahlen*

Transformationsgenerator für **gleich-verteilte stetige Zufallswerte** aus dem Intervall $[a, b)$

$\{y_i\}$ sei $(0, 1)$ - verteilte Zufallszahlenfolge
(erzeugt durch bekannten Generator)

$$x_i = a + (b-a) * y_i$$

Diskrete Zufallszahlengeneratoren



Generator für *gleichverteilte* diskrete Pseudo-Zufallszahlen

- Konstruktor

```
Randint::Randint(DistContext* c, Label title, int na, nb);  
// trägt Objekt in Liste des Kontextes ein
```

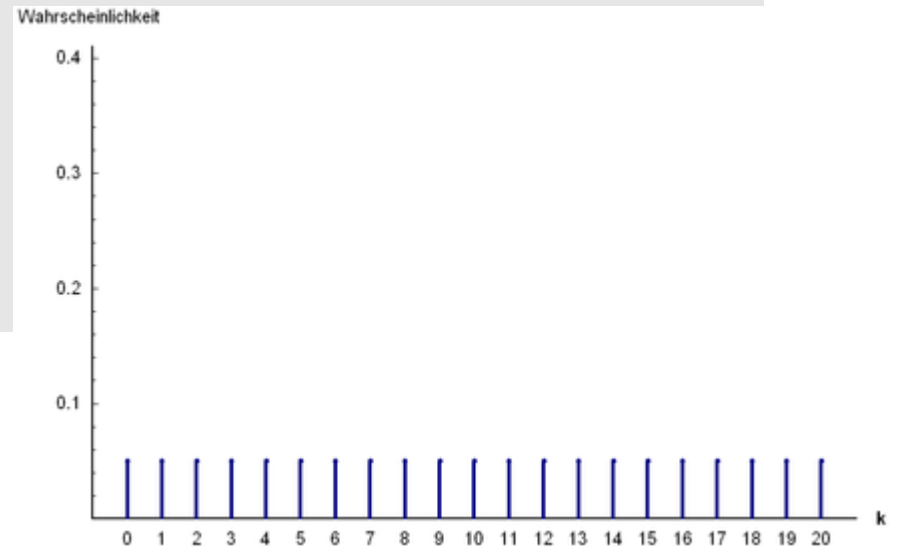
linker Rand

rechter Rand

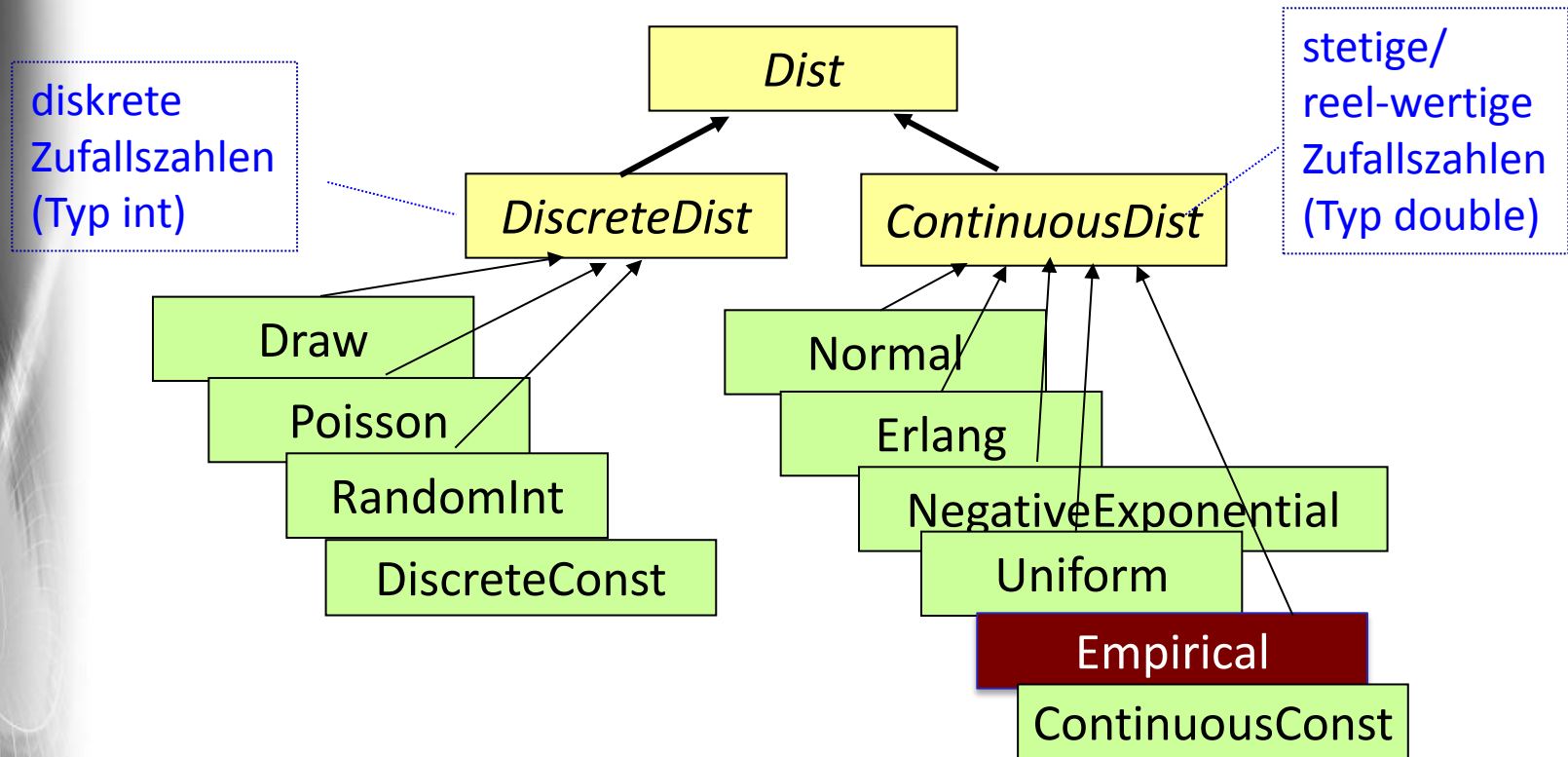
- Generator-Objekt

```
Dist* wuerfel; Simulation* sim;  
wuerfel= new RandomInt (sim, "Los", 0, 20);
```

- Anwendung
`wuerfel->sample();`



Empirische Verteilungsfunktion



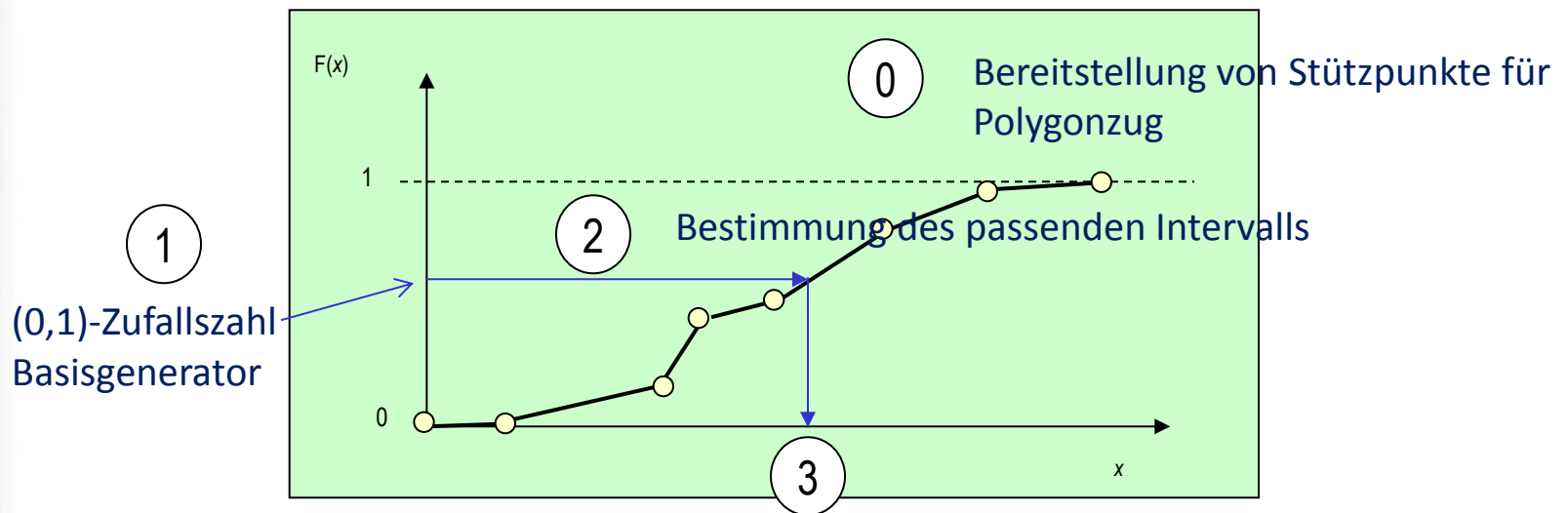
Generator für empirischverteilte Pseudo-Zufallszahlen

Vor.: aufgezeichnete Werte einer beobachteten Größe

- a) Bestimmung der Häufigkeit der Werte in **äquidistanten** Intervallen
- b) daraus die kumulative Häufigkeit $F(x)$

→ erhalten **Verteilungsfunktion**: Polygonzug über $(x, F(x))$ -Stützpunkte

Methode zur Ermittlung einer Zufallszahl entsprechend einer empirischen Verteilung $F(x)$: Schritte 0 bis 3



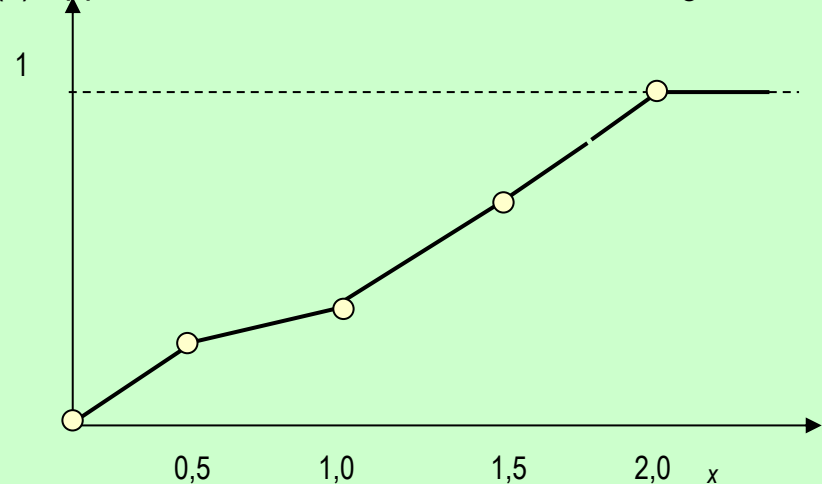
Bestimmung des x -Wertes (Zufallsgröße, die der empirischen $F(x)$ genügt)

Empirische Verteilungen

Beispiel: Aufzeichnung von 100 Reparaturzeiten x

Intervall(h)	Häufigkeit	relative Häufigkeit	kummulative Häufigkeit
$0 \leq x \leq 0.5$	31	0.31	0.31
$0.5 < x \leq 1.0$	10	0.10	0.41
$1.0 < x \leq 1.5$	25	0.25	0.66
$1.5 < x \leq 2.0$	34		

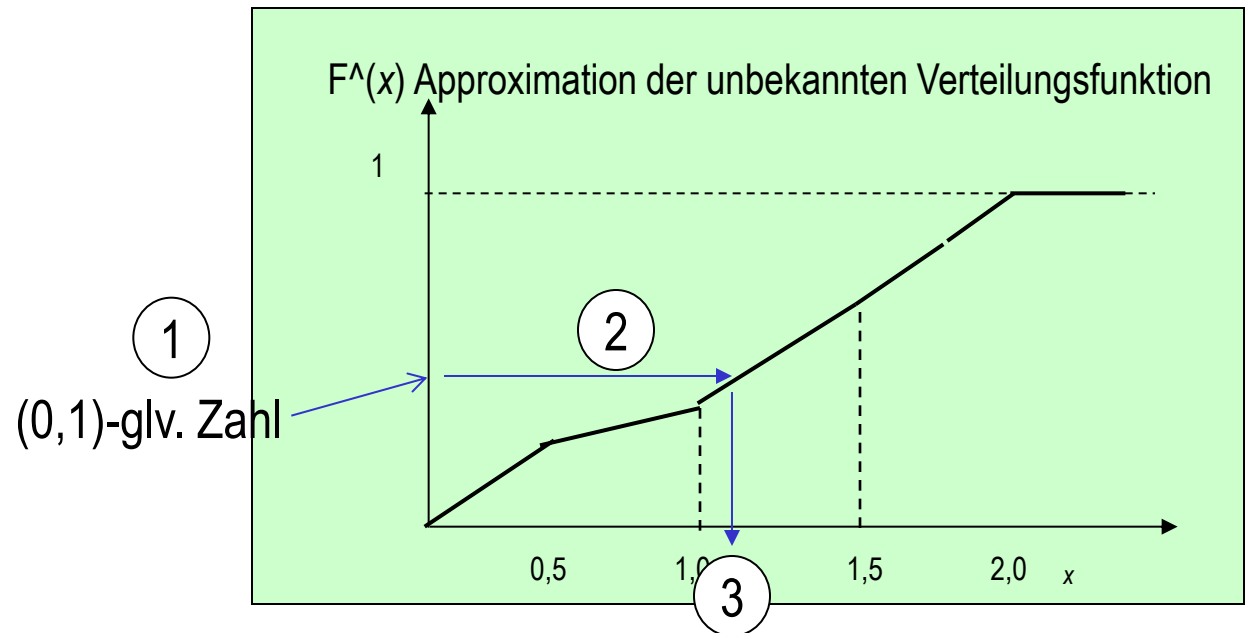
$F(x)$ Approximation der unbekannten Verteilungsfunktion



Empirische Verteilungen (Forts.)

Vorgehensweise:

- Erzeugung einer (0,1)- verteilten Zufallszahl
- Bestimmung des passenden Intervalls (Funktionsgleichung)
- Bestimmung der Zufallsgröße (Reparaturzeit)



5. ODEMx-Modul Random

1. Charakterisierung von Zufallsgrößen
2. Approximation von Zufallszahlen
3. ODEMx- Zufallszahlengeneratoren (Übersicht)
4. Einstellung von Startwerten
5. Protokollierung
6. Berechnung von Zufallszahlen ausgewählter Verteilungen
7. Lösung: Autofähre

- Bin, Res,
- Negexp, Normal,
- Statistikprofile beobachteter Modellgrößen (später)

vertagt

6. ODEMX-Modul Synchronisation: *WaitQ, CondQ*

- Konzept **WaitQ**

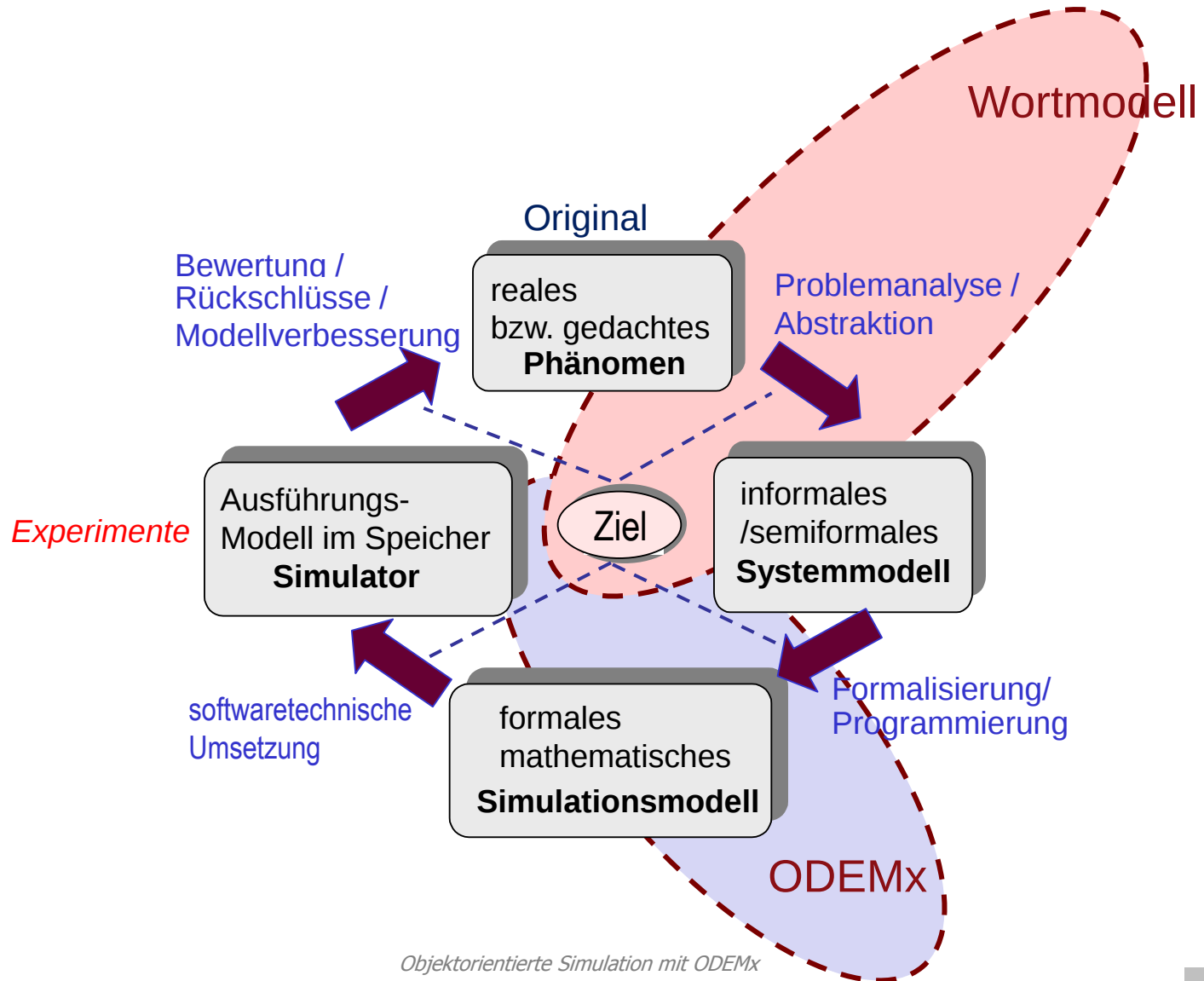
Beispiel: Tankerflotte / Hafen / Raffinerie

- Konzept **CondQ**

Beispiel: Hafen / Schlepper / Gezeiten

- Weitere Anwendungsbeispiele für **WaitQ** u. **CondQ**
- Zusammenfassung/einheitliche Betrachtung

Erinnerung: Vorgehensweise bei der Systemsimulation



Tanker – Tank – Raffinerie: Wortmodell

1. Durchführung einer 1000h-Simulation logistischer Abläufe eines Ölhafen
2. **Tanker** unterschiedlichen Fassungsvermögens (gleichverteilt 15 tb, 20 tb, 25 tb)
 - treffen zufällig im Öl-Hafen ein und
 - können unter best. Bedingungen parallel entladen werden
3. die **Zwischenankunftszeit** der Tanker ist neg.exponential verteilt
 - im Mittel soll alle 8h ein weiterer Tanker eintreffen
4. zur Entladung stehen maximal 5 **Tankbehälter** bereit, von denen pro Tanker jeweils immer nur einer zugeordnet wird, und zwar der
 - der die längste Zeit zur Befüllung bereit war und
 - dessen freie Kapazität die Schiffsladung komplett übernehmen kann
 - steht kein solcher Tank zur Verfügung, muss der Tanker **warten**
5. die **Befüllung** des Tankbehälters (Entladung des Tankers) erfolgt mit einer konstanten Pumprate
 - von 1 tb/h;
 - zum Anschluss eines Tankers an einen Tank werden
 - 0,5 h als **Vorbereitungszeit**
 - benötigt.

1 b $\approx$ 159 l

Wortmodell (Forts.)

5. das maximale Fassungsvermögen eines jeden Tanks beträgt 70 tb
6. während der Befüllung des Tankbehälters erfolgt keine Entnahme durch die angeschlossene Raffinerie
7. die Entnahme von Öl durch die Raffinerie erfolgt
 - (nach Abschluss der Befüllung durch einen Tanker), sobald der Tank nur noch 20 tb oder weniger aufnehmen kann
 - mit einer konstanten Pumprate von 4 tb/h;
8. es liegt eine besondere Ausgangskonfiguration vor
 - a) zwei Tanks sind leer
 - b) einer ist an der Raffinerie angeschlossen und wird in 8h leer
 - c) zwei werden gefüllt (d.h. zwei Tanker haben angelegt), wobei
 - ein Tank in 3,5h fertig wird mit verbleibender Aufnahmekapazität von 25tb und
 - der andere in 12h mit verbleibender Aufnahmekapazität von 45tb
 - d) der nächste Tanker wird zur Zeit 0.0 erwartet

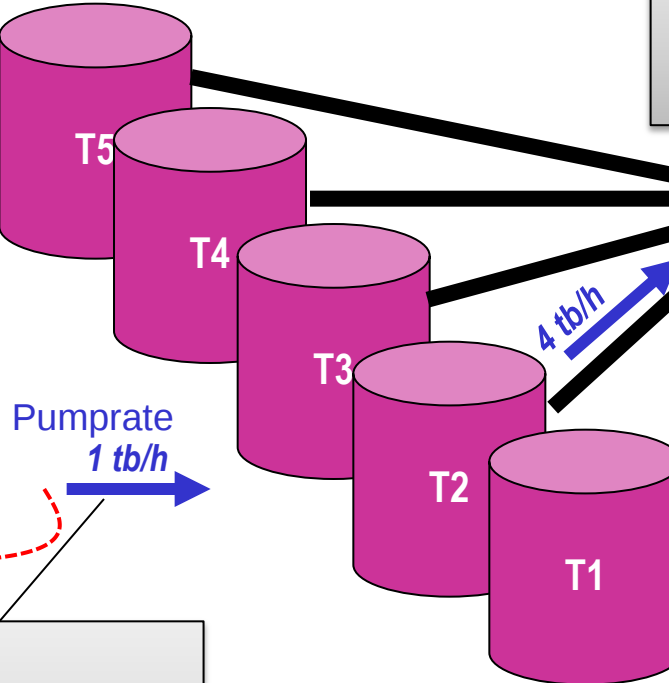
Beispiel: Tanker – Tank – Raffinerie

Ladung: Gleichverteilung
15tb, 20tb, 25tb

Zwischenankunftszeit:
n.-exponential verteilt

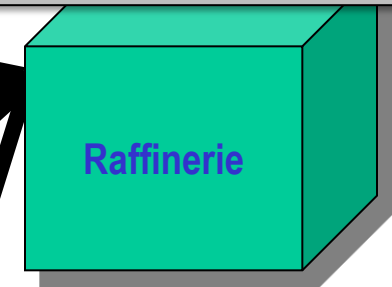


(Kap 70 tb)



Bed.2: Ende der Tank-Befüllung

- ist Tank nahezu voll ($\approx 70\%$), wird Öl zur Raffinerie abgepumpt
- Beladung wird gestoppt



Bed. 1: Tankauswahl

- (1) Tank, der die längste Zeit leer war und
- (2) dessen frei gebliebene Kapazität die Schiffsladung komplett übernehmen kann
- (3) konstante Vorbereitungszeit: 0,5h

Ziel:

1000 h Simulation, bei besonderer Ausgangssituation:

(1) Füllstand der Tankbehälter

- zwei sind leer (70tb frei)
- einer wird in 8h leer (70tb frei)
- einer wird in 12 h mit Befüllung fertig (45tb frei)
- einer wird in 3.5h mit Befüllung fertig (25tb)

(2) erste Tankerankunft: 0.0

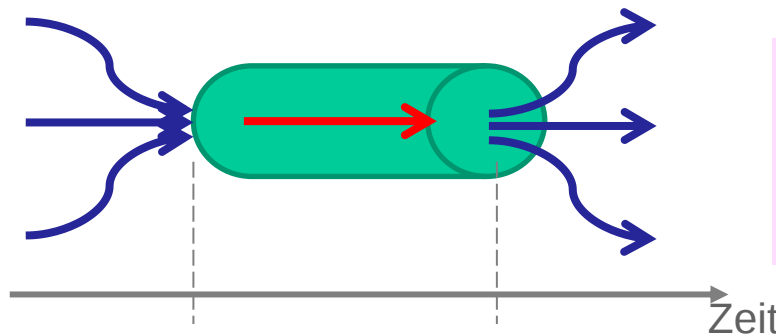
Beispiel zeigt typisches Problem

- Prozess-Objekte kooperieren zeitweilig miteinander (sogar starke Kopplung, da Rückkopplung)
 - hier: Tanker $\leftrightarrow$ Tankbehälter
Tankbehälter $\leftrightarrow$ Raffinerie
(falls Raffinerie als eigenständiges Objekt)
 - Zustandsänderung des einen Objektes (Tanker-Inhalt) ist abhängig von Zustandsänderung des anderen Objektes (Tank-Inhalt) und umgekehrt
- Daraus entsteht ein allg. Synchronisationsproblem:
bei quasiparalleler Ausführung von **n Prozessen** in ihrer Kooperationsphase
- Ein Prozess-Objekt kann in seiner Lebenszeit verschiedene zeitweilige Kooperationsbeziehungen eingehen

Nützliches Modellierungsmuster

- $n+1$ (≥ 2) Prozesse kooperieren ab einem Zeitpunkt für eine gewisse Dauer
- **Bed.:** (1) Zum Startzeitpunkt der Kooperation sind alle $n+1$ Prozesse verfügbar/für die Kooperation bereit
falls nicht, müssen die bereits verfügbaren auf die anderen warten

(2) Zustandsänderungen der Prozesse sind voneinander abhängig)



Entschärfung der Parallelität
der synchronen Wechselwirkungen
bei Zustandsänderungen
im Simulator

- Effiziente simulative Umsetzung auf einer Ein-Prozessor-Maschine
 - **einer** der $n+1$ Prozesse übernimmt als **Master** (aktiv) die Ausführung der Zustandsänderungen sämtlicher Prozesse in Abhängigkeit der Modellzeit
 - **alle anderen** n Prozesse warten als **Slave** (passiv) auf die Beendigung der Kooperation durch den **Master**

ACHTUNG: Master und Slave sind nur Rollen, die Prozesse zeitweilig spielen