

# Teil I

## Modellierung und Analyse

- Kap. 2-7 Modellierung und Simulation diskreter Systeme
  - Einführung in die klassische ereignisdiskrete Simulation
  - Inkl. statistischer Grundlagen
- Kap. 8 Validierung von Modellen
  - Hier im Bezug auf ereignisdiskrete Modelle vorgestellt, aber allgemeiner anwendbar
- Modellierung und Analyse kontinuierlicher Systeme und analytische/numerische Methoden werden in der Vorlesung nicht behandelt

# Modellierung und Analyse diskreter Systeme

## Gliederung

2. Konzepte ereignisdiskreter Simulation  
inkl. Spezifikation von Simulatoren
3. Generierung und Bewertung von Zufallszahlen
4. Modellierung von Eingabedaten
5. Auswertung von Simulationsläufen
6. Simulationssoftware
7. Möglichkeiten und Grenzen der Simulation

# Literatur:

## Primär verwendet

- A. M. Law, Simulation Modeling and Analysis. McGraw Hill 5. Auflage 2015.  
Kapitel 1, 3-1.5, 3, 6.1-6.5, 7.1, 7.2, 8.1, 8.2, 9.1-9.5 oder
- Banks et al. Discrete-Event System Simulation. Prentice Hall 2000.  
Kap. 2-4, 7, 8, 9.1-9.4, 11.

Auch zu empfehlen (für Teilbereiche):

- A. Borschev. The Big Book of Simulation Modeling - Multimethod Modeling with AnyLogic 8 AnyLogic 2020.  
(<https://www.anylogic.de/resources/books/big-book-of-simulation-modeling/> )
- u.v.a.

In den folgenden Kapiteln werden Systeme untersucht,

- die zu diskreten Zeitpunkten ihren Zustand ändern
- bei denen zwischen Zustandsänderungen eine durch eine Zufallsvariable festgesetzte Zeit vergeht

Simulation dieser Systeme bedeutet Nachbilden und Beobachten des dynamischen Ablaufs

- programmtechnische Darstellung des Systemzustands
- zeitgerechte Ausführung der Ereignisse zur Änderung des Systemzustands
- Auswertung der Beobachtungen zur Ermittlung von Resultatgrößen

## Ziele:

- Erkennen des grundsätzlichen Ablaufs diskreter Simulationen
- Kennen lernen unterschiedlicher Realisierungskonzepte für ereignisdiskrete Simulatoren
- Anforderungen an Programmiersprachen zur Erstellung von Simulatoren formulieren können
- Übersicht über einige gängige Simulationswerkzeuge erhalten
- Wissen über die Realisierung von Zufallszahlen erlangen
- Methoden zur Darstellung gemessener Verteilungen durch Zufallszahlen kennen lernen
- Resultatschätzer und Konfidenzintervalle aus Simulationsdaten gewinnen können
- Überblick über die Struktur vorhandener Softwaresysteme erlangen
- Grenzen der Simulation einschätzen können

## 2 Konzepte ereignisdiskreter Simulation

### 2.1 Struktur und Ablauf der Simulation

Simulieren  $\approx$  Imitieren des Realsystems, d.h. (für jeden Zeitpunkt):

- Systemzustand kennen
- (nächste) Zustandsänderung kennen

Zustand im Simulator (imperative Sicht):

- Werte aller Programmvariablen

Nächste Zustandsänderung  $\Rightarrow$  Programmzeiger

Damit natürlich und naheliegend:

- Zustandsraum Realsystem  $\rightarrow$  Datenstruktur im Simulator

In Simulationsterminologie:

Menge der Zustandsvariablen bildet die **statische Struktur** des Modells

## Zustandsänderungen im Simulator:

- Werteänderungen von Zustandsvariablen
- d.h. per Wertzuweisung
- d.h. per Ausführung von Simulator-Code

⇒ zu jedem **Ereigniszeitpunkt** muss ein Stück Simulator-Code abgearbeitet werden

## Im Simulator müssen

- **Ereignistypen** festgelegt werden  
(wobei jeder Ereignistyp durch eine Menge von Zustandsänderungen charakterisiert ist)
- Code-Segmente für jeden Ereignistyp programmiert werden  
(**Ereignisroutinen**)
- die **zeitgerechte Ausführung** der Ereignisroutinen organisiert werden

## Zum Zeitablauf:

- Ablauf der Ereignisroutinen im Modell zeitlos
  - Zustandsänderungen sind atomar, finden in Nullzeit statt
  - Abarbeitung der zugehörigen Routine dauert in der Realität Zeit
- zwischen zwei Ereignissen vergeht in der Realität Zeit
  - Erstes Ereignis tritt zum Zeitpunkt  $t$ , nächstes zum Zeitpunkt  $t+\Delta$  ( $\Delta \geq 0$ ) ein ( $\Delta$  in Abhängigkeit vom Zustand-Ereignis)
  - Übergang von einer Ereignisroutine zur nächsten erfordert im Rechner (fast) keine Zeit

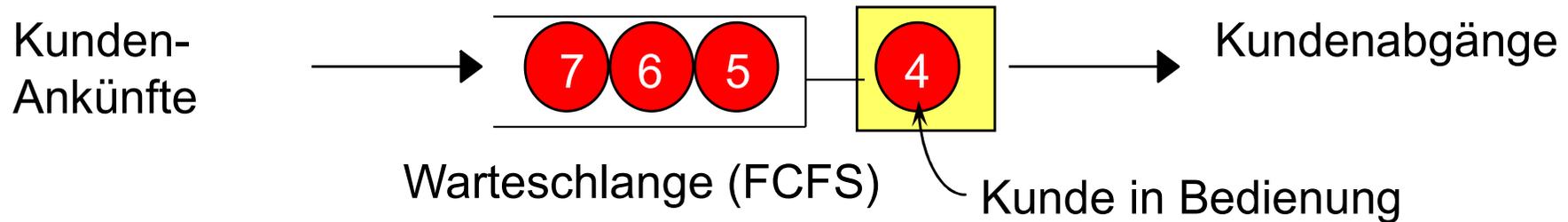
## **Damit besteht ein Simulator (sehr abstrakt):**

- **Abbildung des Zustands → statische Struktur**
- **Abbildung der Zustandsänderungen → dynamische Struktur**

## Erläuterung des prinzipiellen Vorgehens der ereignisdiskreten Simulation an einem einfachen Beispiel:

- Ein Bediener, der immer dann arbeitet, wenn Kunden warten
  - der Bediener kann genau einen Kunden zu einem Zeitpunkt bedienen
- Kunden kommen im System an und werden vom Bediener bedient
  - falls der Bediener belegt ist, warten die Kunden in einem Warteraum potenziell unendlicher Kapazität
  - Kunden im Warteraum werden nach First Come First Served (FCFS) abgearbeitet
  - nach Ende der Bedienung verlassen Kunden das System
- Ankunftszeitpunkte und Bedienzeiten resultieren aus Messungen oder werden aus vorgegebenen Verteilungsfunktionen generiert

# Graphische Darstellung des Systems:



## Statische Struktur: (es gibt unterschiedliche Möglichkeiten)

- Anzahl Kunden im Warteraum  $Q$  (integer)
- Bediener arbeitet  $B$  (Boolean, hier mit Werten 0 und 1)
- Zusätzlich vorhanden Variable  $t$  zur Beschreibung der aktuellen Simulationszeit
- Annahme: Zustand zu Beginn der Simulation  $t=0$  bekannt, Simulation bis zum Zeitpunkt  $T (> 0)$
- $B(t)$  und  $Q(t)$  Werte der Variablen zum Zeitpunkt  $t$
- $B(0)=\text{false}$  und  $Q(0)=0$

## Dynamische Struktur: Zwei Ereignisse Ankunft und Bedienende

- Bei einer Ankunft:
  - Wird  $B$  auf 1 gesetzt, falls  $B=0$   
(und startet damit implizit die Bedienung)
  - Wird  $Q$  auf  $Q+1$  falls  $B=1$
- Bei einem Bedienende
  - Wird  $B$  auf 0 gesetzt, falls  $Q=0$
  - Wird  $Q$  auf  $Q-1$  gesetzt, falls  $Q$  größer 0 ist

## Zusätzliche Aufgaben in der Simulation:

- Zeitgerechte Ausführung der Ereignisse (dazu später mehr)
- Beobachtung des Systemverhaltens  
z.B. durch Definition und Auswertung von Resultatgrößen

## Definition von Resultatgrößen:

- Anzahl der Kunden, die ihre Bedienung im Intervall  $[0, T]$  beendet haben  $P$
- Mittlere Population in der Warteschlange  $\int_0^T Q(t) dt / T$
- Maximal Population in der Warteschlange  $\max_{0 \leq t \leq T} Q(t)$
- Auslastung des Bedieners  $\int_0^T B(t) dt / T$
- Mittlere Verweilzeit eines Kunden  $\sum_{p=1}^P D_p / P$   
wobei  $D_p$  Verweilzeit des  $p$ -ten Kunden ist
- Durchsatz  $P/T$
- u.v.a.

Resultatgrößen sind während der Simulation neu zu berechnen/auszuwerten

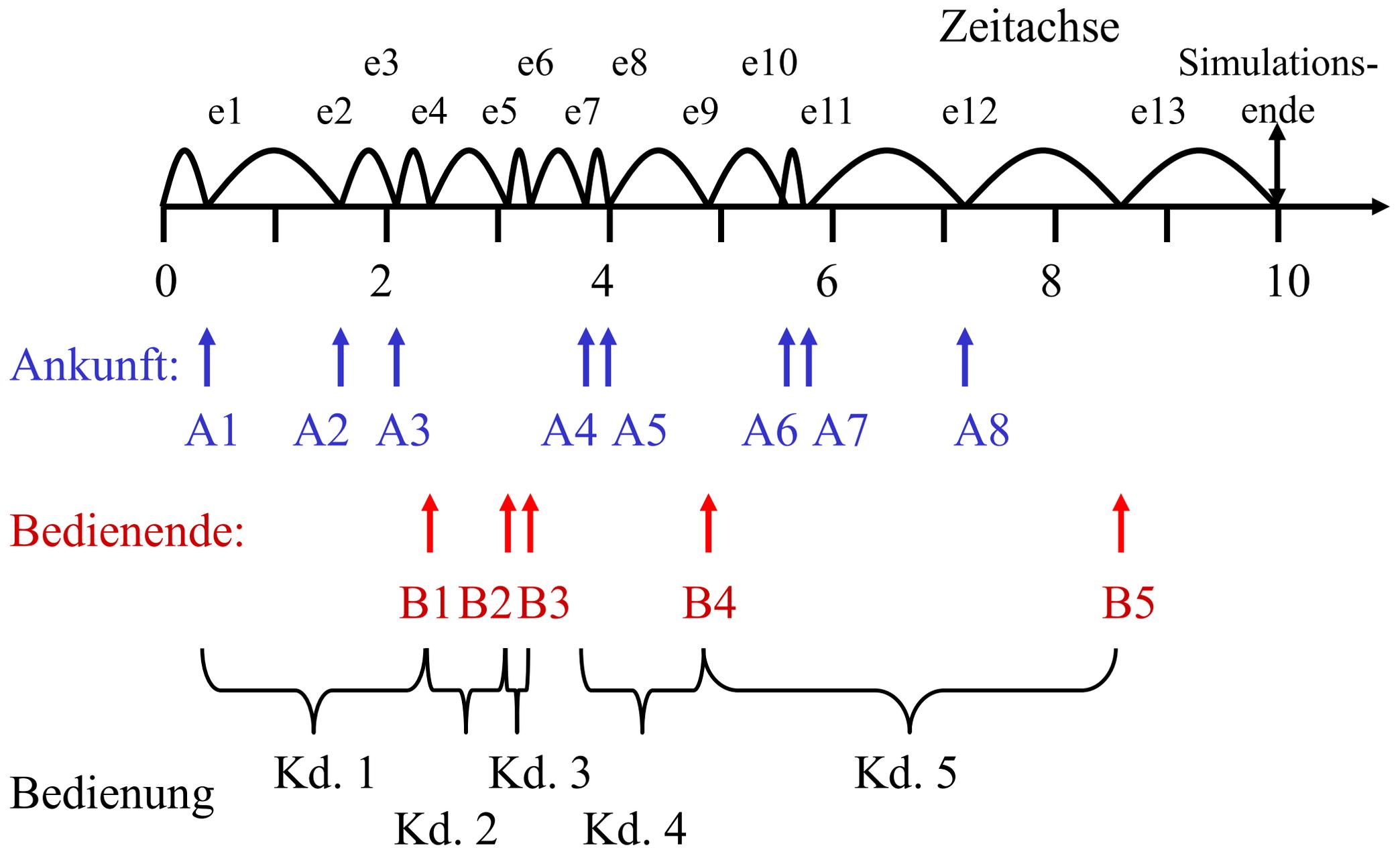
## Ablauf (Zeitangaben in Minuten, Simulationsdauer 10 Min.):

Kunde	Ankunftszeit	Zwischenankunftszeit	Bedienzeit
1	0.4	0.4	2.0
2	1.6	1.2	0.7
3	2.1	0.5	0.2
4	3.8	1.7	1.1
5	4.0	0.2	3.7
6	5.6	1.6	1.5
7	5.8	0.2	0.4
8	7.2	1.4	1.1

Es gilt für Kunden  $i$ :

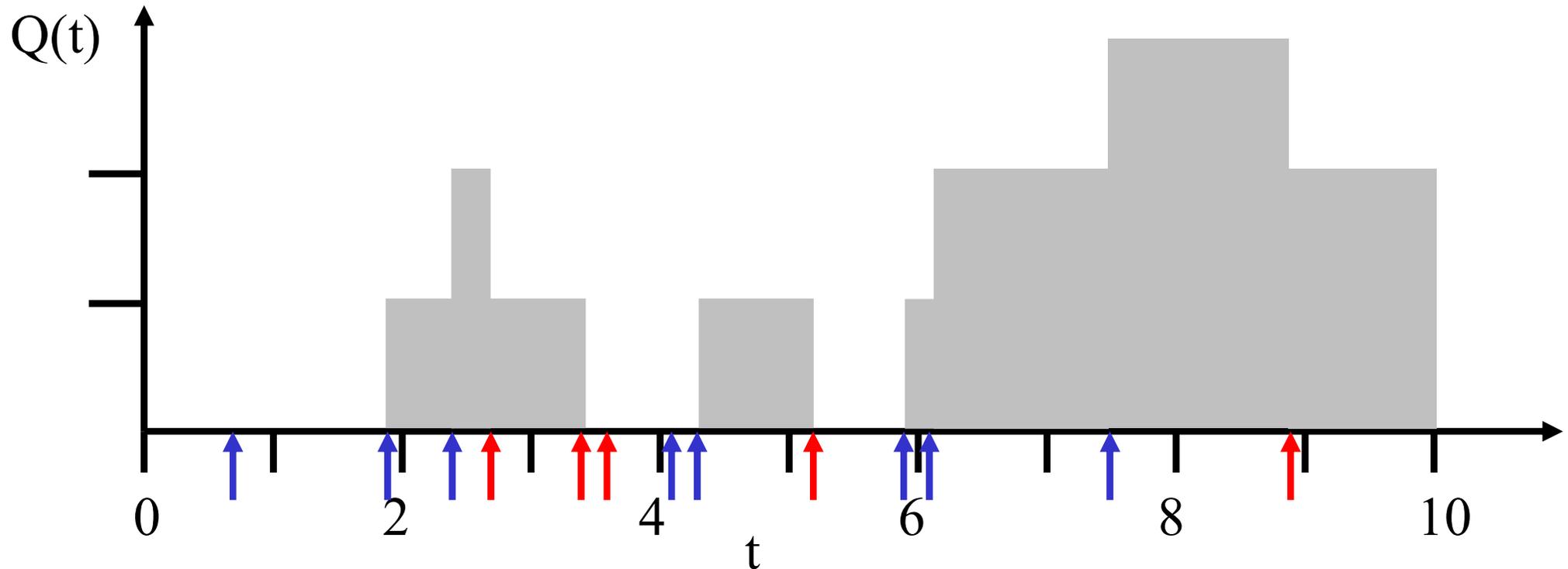
- $\text{Bedienende } i = \text{Bedienanfang } i + \text{Bedienzeit } i$
- $\text{Bedienanfang } i = \max(\text{Ankunftszeit } i, \text{Bedienende } i-1)$

Ziel: Zeitgerechte Ausführung der Ereignisse!



# Bestimmung von Resultaten während der Simulation

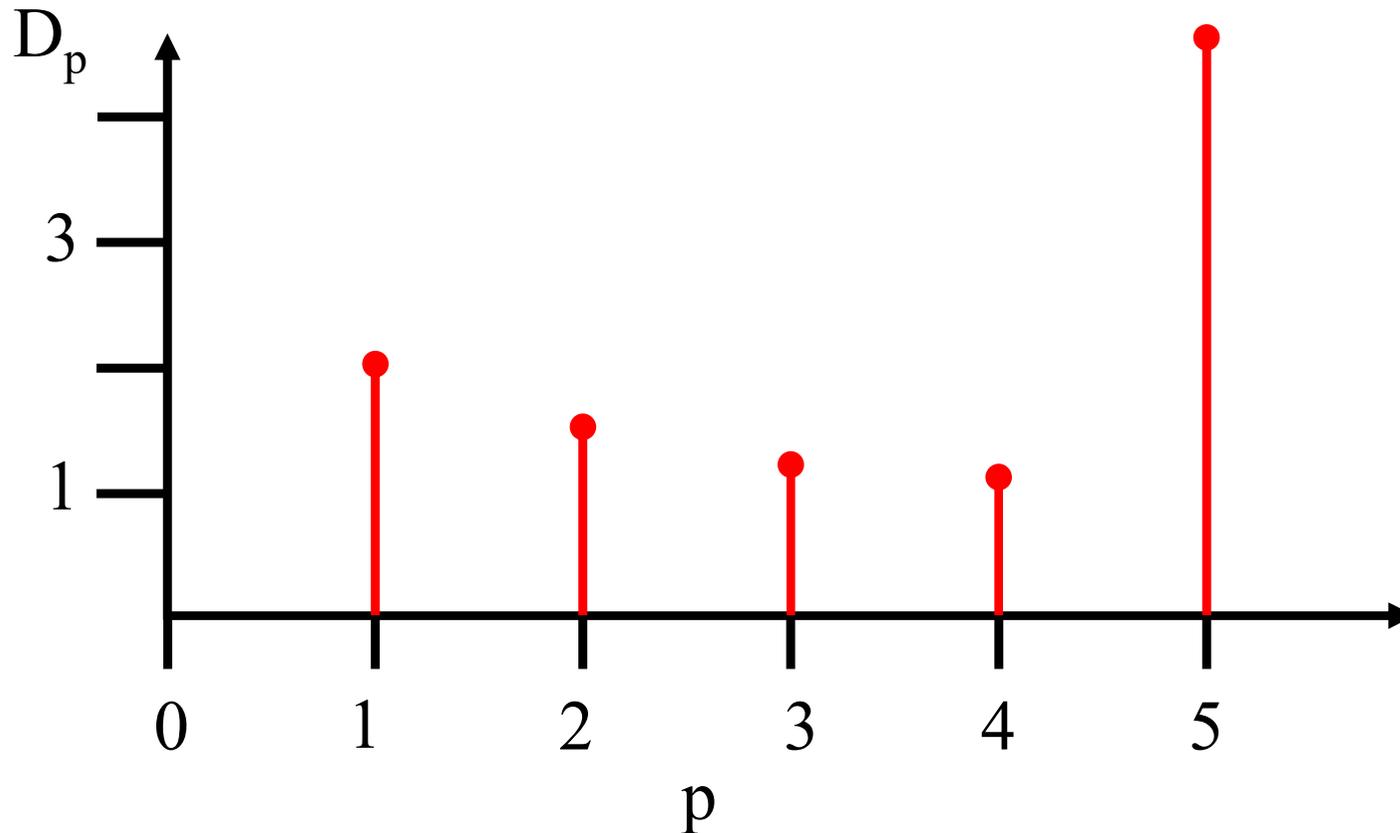
Beispiel:  $Q(t)$



Die mittlere Population in der Warteschlange ergibt sich aus der grauen Fläche im Verhältnis zur Länge der Beobachtung

Zur Ermittlung während der Simulation wird bei jedem Ereignis der Flächeninhalt seit dem letzten Ereignis ausgewertet!

## Beispiel: $D_p$ Verweilzeiten der Kunden



Die mittlere Verweilzeit ergibt sich aus der Summe der Verweilzeiten dividiert durch die Anzahl beobachteter Kunden

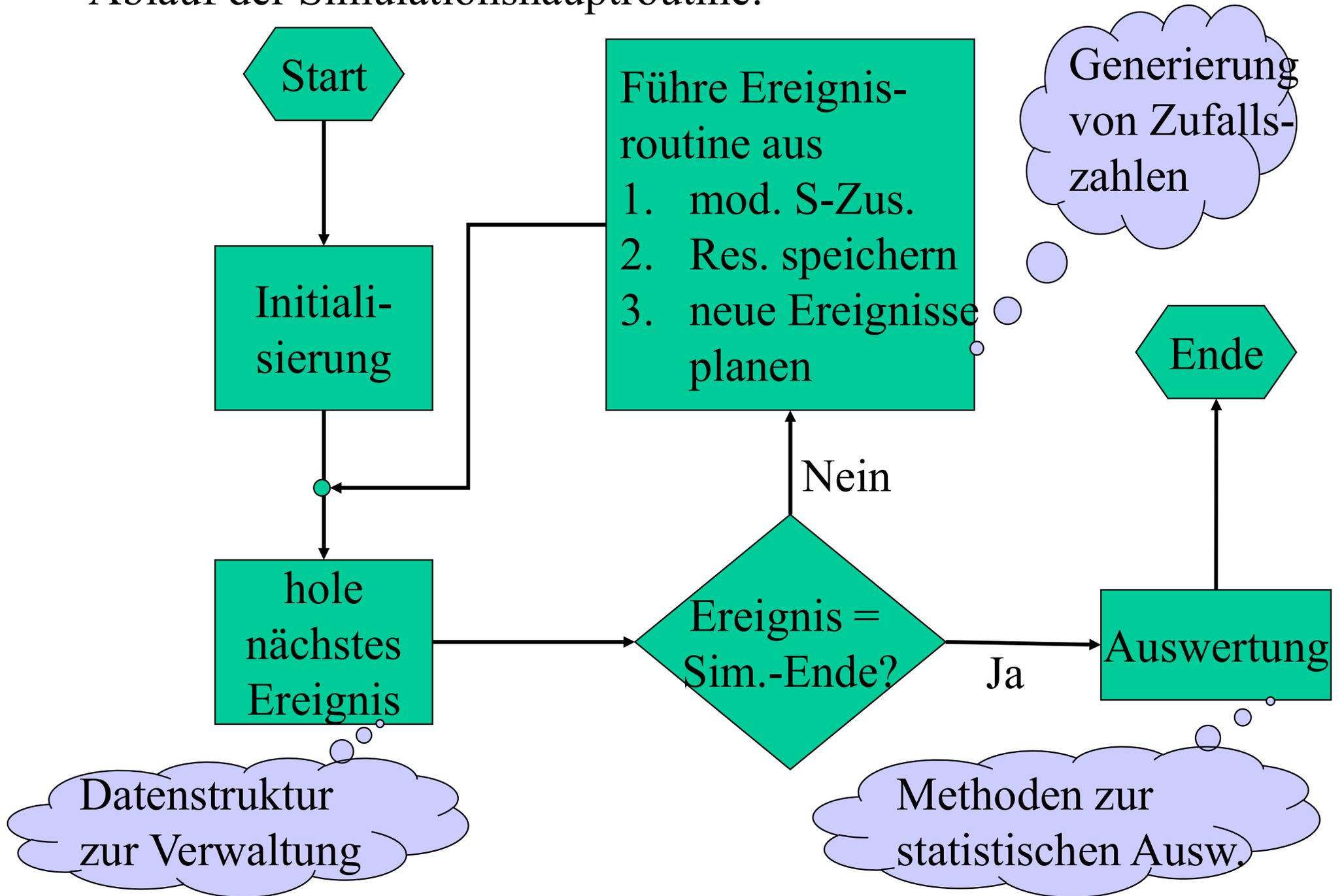
Zur Ermittlung während der Simulation wird bei jedem Kunden, der das System verlässt, die Verweilzeit ermittelt!

## 2.2 Dynamischer Ablauf

Einige Beobachtungen:

- Simulationsablauf durch Sprünge von einem Ereignis zum nächsten
- Simulationszeit  $t$  ergibt sich aus der Zeit des nächsten Ereignisses
- Ereigniszeitpunkte ergeben sich z. T. während der Simulation  
z.B. Bedienende bestimmt den nächsten Bedienanfang
- Zu jedem Ereigniszeitpunkt wird die zugehörige Ereignisroutine ausgeführt, um
  - 1. den Systemzustand zu modifizieren**
  - 2. Resultatwerte zu speichern**
  - 3. zukünftige Ereignisse einzuplanen**

# Ablauf der Simulationshauptroutine:



# Datenstruktur zur Verwaltung von Ereignissen

## Anforderungen:

- Jedes Ereignis  $i$  ist beschrieben durch den Ereigniszeitpunkt  $t_i$  und den Ereignistyp  $tp_i$
- Ereignisse müssen nach Zeit ihres Eintretens  $t_i$  geordnet sein
- Zugriffsfunktionen
  - Auslesen des ersten Elements
  - Einfügen eines neuen Elements
  - Löschen eines eingefügten Elements  
(im einfachen Beispiel nicht verwendet!)
- **Verwendete Datenstruktur Ereignisliste**

# Ereignisliste:

$t_i$	$t_1$	$t_2$	$t_3$	$t_4$	$t_5$	...
$tp_i$	$tp_1$	$tp_2$	$tp_3$	$tp_4$	$tp_5$	...

- Inhalt der Ereignisliste umfasst die bisher geplante Modellzukunft
- Simulationsende kann als spezielles Ereignis zu Beginn der Simulation eingefügt werden, falls die Abbruchzeit feststeht (es gibt andere Abbruchbedingungen!)
- Ereignisse können nur zu Zeitpunkten  $t \geq t_1$  eingefügt werden (Simulationszeit läuft nur in eine Richtung!)
- Für gleichzeitige Ereignisse (d.h.  $t_i = t_j$ ) existiert keine Reihenfolge (dies kann zu nicht-deterministischem Verhalten führen!)

Für unser Beispiel:

- $t$  aktuelle Simulationszeit

### Zustandsvariablen

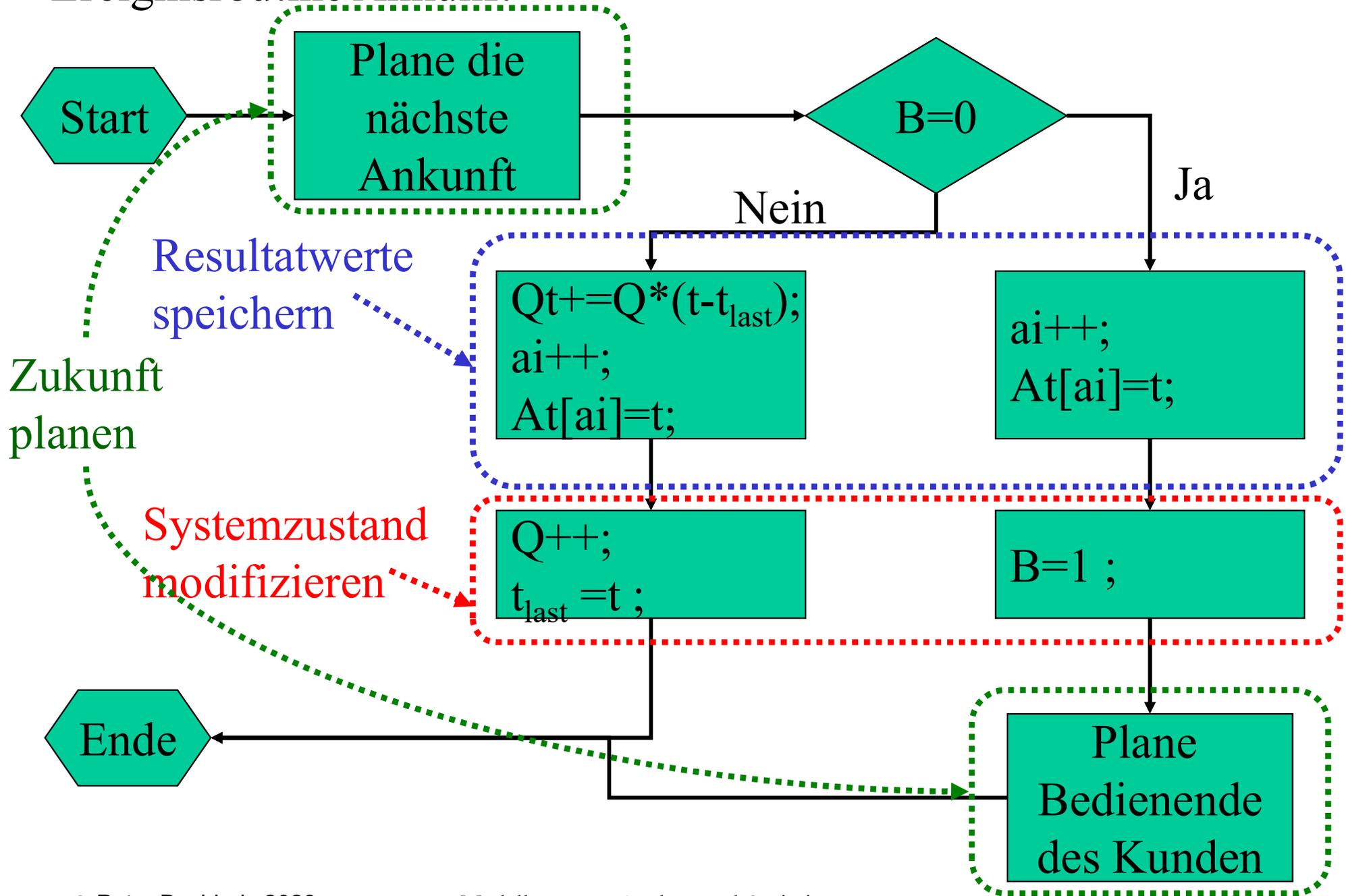
- $Q$  Anzahl Aufträge in der Warteschlange, initialisiert mit 0
- $B$  Status des Bedieners, initialisiert mit 0

### Variablen zur Auswertung der Resultate

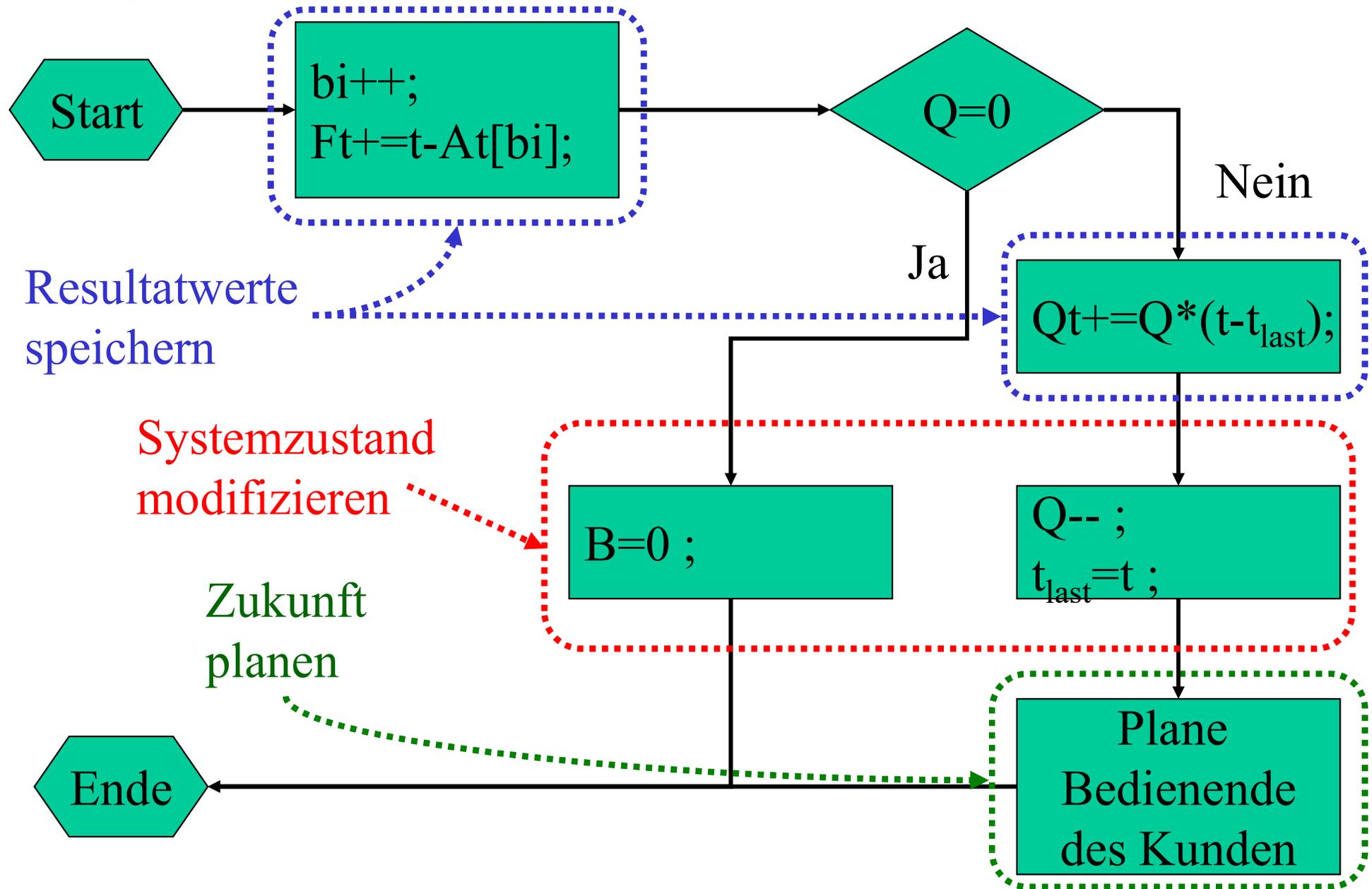
(alle Variablen werden mit 0 initialisiert)

- $Qt$  kumulierter Wert der Auftragszahl in der Warteschlange
- $t_{\text{last}}$  Zeitpunkt der letzten Änderung von  $Q$
- $ai$  Nummer des letzten Auftrags, der angekommen ist
- $bi$  Nummer des letzten Auftrags, der das System verlassen hat
- $At[i]$  Ankunftszeit des  $i$ -ten Auftrags
- $Ft$  kumulierter Wert der Verweilzeiten

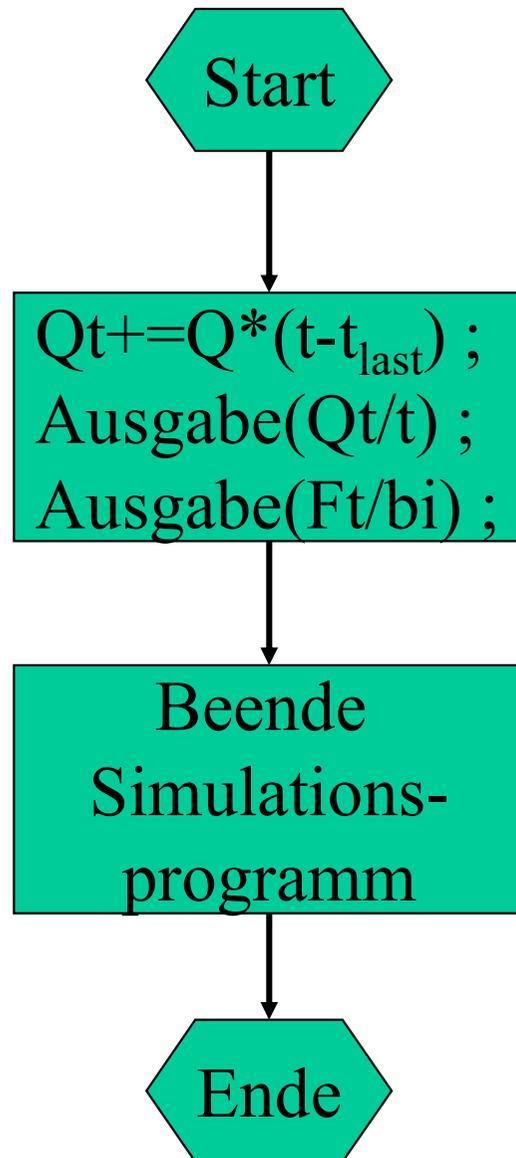
# Ereignisroutine Ankunft



# Ereignisroutine Bediende



# Simulationsende



- Falls nach T Zeiteinheiten abgebrochen werden soll, wird Ereignis *Simulationsende* zu Beginn der Simulation für den Zeitpunkt T eingeplant!
- Falls bzgl. einer zu erfüllenden Bedingung abgebrochen werden soll, wird sobald die Bedingung erfüllt ist, ein Ereignis *Simulationsende* sofort (d.h. für Zeit t) eingeplant

## Zum Start der Simulation

- Initialisierung der Variablen
- Planen der ersten Auftragsankunft

## Simulation liefert:

- Mittlere Population in der Warteschlange im Intervall  $[0, T]$
- Mittlere Verweilzeit fertig gewordener Aufträge im Intervall  $[0, T]$

## Bei stochastischen Simulationen jeweils für einen möglichen Ablauf!

- Um Aussagen über alle möglichen Abläufe zu erlangen, sind Methoden der Stochastik notwendig (dazu später mehr)

# Ablauf des Beispiels

Ereigniscodierung A Ankunft, B Bedienende, E Simulationseende

Im Beispiel hinzugefügt eine weitere Ankunft zum Zeitpunkt 10.2

## Start der Simulation

<b>Zustandsvariablen</b> Q=0    B=0    ai=0    bi=0 $t_{\text{last}}=0.0$	<b>Simulatorzustand</b> t=0.0    EL				
<b>Resultatvariablen</b> Qt=0.0            Ft=0.0            At=()	<table border="1"><tr><td>0.4</td><td>A</td></tr><tr><td>10</td><td>E</td></tr></table>	0.4	A	10	E
0.4	A				
10	E				

## Erste Ankunft

<b>Zustandsvariablen</b> Q=0   B=1   ai=1   bi=0 $t_{\text{last}}=0.0$	<b>Simulatorzustand</b> t=0.4   EL	<table border="1"><tr><td>1.6</td><td>A</td></tr><tr><td>2.4</td><td>B</td></tr><tr><td>10</td><td>E</td></tr></table>	1.6	A	2.4	B	10	E
1.6	A							
2.4	B							
10	E							
<b>Resultatvariablen</b> Qt=0.0   Ft=0.0   At=(0.4)								

## Zweite Ankunft

<b>Zustandsvariablen</b> Q=1   B=1   ai=2   bi=0 $t_{\text{last}}=1.6$	<b>Simulatorzustand</b> t=1.6   EL	<table border="1"><tr><td>2.1</td><td>A</td></tr><tr><td>2.4</td><td>B</td></tr><tr><td>10</td><td>E</td></tr></table>	2.1	A	2.4	B	10	E
2.1	A							
2.4	B							
10	E							
<b>Resultatvariablen</b> Qt=0.0   Ft=0.0   At=(0.4,1.6)								

## Dritte Ankunft

<p>Zustandsvariablen</p> <p><math>Q=2</math>   <math>B=1</math>   <math>ai=3</math>   <math>bi=0</math>   <math>t_{\text{last}}=2.1</math></p>	<p>Simulatorzustand</p> <p><math>t=2.1</math>   EL</p> <table border="1" style="margin-left: 20px;"> <tr><td style="background-color: #00ffcc;">2.4</td><td style="background-color: #00ffcc;">B</td></tr> <tr><td style="background-color: #00ffcc;">3.8</td><td style="background-color: #00ffcc;">A</td></tr> <tr><td style="background-color: #00ffcc;">10</td><td style="background-color: #00ffcc;">E</td></tr> </table>	2.4	B	3.8	A	10	E
2.4	B						
3.8	A						
10	E						
<p>Resultatvariablen</p> <p><math>Qt=0.5</math>   <math>Ft=0.0</math>   <math>At=(0.4,1.6,2.1)</math></p>							

## Erstes Bedienende

<p>Zustandsvariablen</p> <p><math>Q=1</math>   <math>B=1</math>   <math>ai=3</math>   <math>bi=1</math>   <math>t_{\text{last}}=2.4</math></p>	<p>Simulatorzustand</p> <p><math>t=2.4</math>   EL</p> <table border="1" style="margin-left: 20px;"> <tr><td style="background-color: #00ffcc;">3.1</td><td style="background-color: #00ffcc;">B</td></tr> <tr><td style="background-color: #00ffcc;">3.8</td><td style="background-color: #00ffcc;">A</td></tr> <tr><td style="background-color: #00ffcc;">10</td><td style="background-color: #00ffcc;">E</td></tr> </table>	3.1	B	3.8	A	10	E
3.1	B						
3.8	A						
10	E						
<p>Resultatvariablen</p> <p><math>Qt=1.1</math>   <math>Ft=2.0</math>   <math>At=(0.4,1.6,2.1)</math></p>							

## Zweites Bedienende

<p>Zustandsvariablen</p> <p><math>Q=0</math>   <math>B=1</math>   <math>ai=3</math>   <math>bi=2</math>   <math>t_{last}=3.1</math></p>	<p>Simulatorzustand</p> <p><math>t=3.1</math>   EL</p> <table border="1" style="margin-left: 20px;"> <tr><td style="background-color: #00FF99;">3.3</td><td style="background-color: #00FF99;">B</td></tr> <tr><td style="background-color: #00FF99;">3.8</td><td style="background-color: #00FF99;">A</td></tr> <tr><td style="background-color: #00FF99;">10</td><td style="background-color: #00FF99;">E</td></tr> </table>	3.3	B	3.8	A	10	E
3.3	B						
3.8	A						
10	E						
<p>Resultatvariablen</p> <p><math>Qt=1.8</math>   <math>Ft=3.5</math>   <math>At=(0.4,1.6,2.1)</math></p>							

## Drittes Bedienende

<p>Zustandsvariablen</p> <p><math>Q=0</math>   <math>B=0</math>   <math>ai=3</math>   <math>bi=3</math>   <math>t_{last}=3.1</math></p>	<p>Simulatorzustand</p> <p><math>t=3.3</math>   EL</p> <table border="1" style="margin-left: 20px;"> <tr><td style="background-color: #00FF99;">3.8</td><td style="background-color: #00FF99;">A</td></tr> <tr><td style="background-color: #00FF99;">10</td><td style="background-color: #00FF99;">E</td></tr> </table>	3.8	A	10	E
3.8	A				
10	E				
<p>Resultatvariablen</p> <p><math>Qt=1.8</math>   <math>Ft=4.7</math>   <math>At=(0.4,1.6,2.1)</math></p>					

## Vierte Ankunft

<p>Zustandsvariablen</p> <p><math>Q=0</math>   <math>B=1</math>   <math>ai=4</math>   <math>bi=3</math>   <math>t_{\text{last}}=3.8</math></p>	<p>Simulatorzustand</p> <p><math>t=3.8</math>   EL</p> <table border="1" style="margin-left: 20px;"> <tr><td style="background-color: #00FF99;">4.0</td><td style="background-color: #00FF99;">A</td></tr> <tr><td style="background-color: #00FF99;">4.9</td><td style="background-color: #00FF99;">B</td></tr> <tr><td style="background-color: #00FF99;">10</td><td style="background-color: #00FF99;">E</td></tr> </table>	4.0	A	4.9	B	10	E
4.0	A						
4.9	B						
10	E						
<p>Resultatvariablen</p> <p><math>Qt=1.8</math>   <math>Ft=4.7</math>   <math>At=(0.4, 1.6, 2.1, 3.8)</math></p>							

## Fünfte Ankunft

<p>Zustandsvariablen</p> <p><math>Q=1</math>   <math>B=1</math>   <math>ai=5</math>   <math>bi=3</math>   <math>t_{\text{last}}=4.0</math></p>	<p>Simulatorzustand</p> <p><math>t=4.0</math>   EL</p> <table border="1" style="margin-left: 20px;"> <tr><td style="background-color: #00FF99;">4.9</td><td style="background-color: #00FF99;">B</td></tr> <tr><td style="background-color: #00FF99;">5.6</td><td style="background-color: #00FF99;">A</td></tr> <tr><td style="background-color: #00FF99;">10</td><td style="background-color: #00FF99;">E</td></tr> </table>	4.9	B	5.6	A	10	E
4.9	B						
5.6	A						
10	E						
<p>Resultatvariablen</p> <p><math>Qt=1.8</math>   <math>Ft=4.7</math>   <math>At=(0.4, 1.6, 2.1, 3.8, 4.0)</math></p>							

## Viertes Bedienende

<p>Zustandsvariablen</p> <p><math>Q=0</math>   <math>B=1</math>   <math>ai=5</math>   <math>bi=4</math>   <math>t_{last}=4.9</math></p>	<p>Simulatorzustand</p> <p><math>t=4.9</math>   EL</p> <table border="1" style="margin-left: 20px;"> <tr><td>5.6</td><td>A</td></tr> <tr><td>8.6</td><td>B</td></tr> <tr><td>10</td><td>E</td></tr> </table>	5.6	A	8.6	B	10	E
5.6	A						
8.6	B						
10	E						
<p>Resultatvariablen</p> <p><math>Qt=2.7</math>   <math>Ft=5.8</math>   <math>At=(0.4,1.6,2.1,3.8,4.0)</math></p>							

## Sechste Ankunft

<p>Zustandsvariablen</p> <p><math>Q=1</math>   <math>B=1</math>   <math>ai=6</math>   <math>bi=4</math>   <math>t_{last}=5.6</math></p>	<p>Simulatorzustand</p> <p><math>t=5.6</math>   EL</p> <table border="1" style="margin-left: 20px;"> <tr><td>5.8</td><td>A</td></tr> <tr><td>8.6</td><td>B</td></tr> <tr><td>10</td><td>E</td></tr> </table>	5.8	A	8.6	B	10	E
5.8	A						
8.6	B						
10	E						
<p>Resultatvariablen</p> <p><math>Qt=2.7</math>   <math>Ft=5.8</math>   <math>At=(0.4,1.6,2.1,3.8,4.0,5.6)</math></p>							

## Siebte Ankunft

<p>Zustandsvariablen</p> <p><math>Q=2</math>   <math>B=1</math>   <math>a_i=7</math>   <math>b_i=4</math>   <math>t_{\text{last}}=5.8</math></p>	<p>Simulatorzustand</p> <p><math>t=5.8</math>   EL</p> <table border="1" style="margin-left: 20px;"> <tr><td>7.2</td><td>A</td></tr> <tr><td>8.6</td><td>B</td></tr> <tr><td>10</td><td>E</td></tr> </table>	7.2	A	8.6	B	10	E
7.2	A						
8.6	B						
10	E						
<p>Resultatvariablen</p> <p><math>Q_t=2.9</math>   <math>F_t=5.8</math></p>	<p><math>A_t=(0.4, 1.6, 2.1, 5.8, 3.8, 4.0, 5.8)</math></p>						

## Achte Ankunft

<p>Zustandsvariablen</p> <p><math>Q=3</math>   <math>B=1</math>   <math>a_i=8</math>   <math>b_i=4</math>   <math>t_{\text{last}}=7.2</math></p>	<p>Simulatorzustand</p> <p><math>t=7.2</math>   EL</p> <table border="1" style="margin-left: 20px;"> <tr><td>8.6</td><td>B</td></tr> <tr><td>10</td><td>E</td></tr> <tr><td>10.2</td><td>A</td></tr> </table>	8.6	B	10	E	10.2	A
8.6	B						
10	E						
10.2	A						
<p>Resultatvariablen</p> <p><math>Q_t=5.7</math>   <math>F_t=5.8</math></p>	<p><math>A_t=(0.4, 1.6, 2.1, 3.8, 4.0, 5.6, 5.8, 7.2)</math></p>						

## Fünftes Bedienende

<p>Zustandsvariablen</p> <p><math>Q=2</math>   <math>B=1</math>   <math>a_i=8</math>   <math>b_i=5</math>   <math>t_{\text{last}}=8.6</math></p>	<p>Simulatorzustand</p> <p><math>t=8.6</math>   EL</p> <table border="1" style="background-color: #00b050; color: white;"> <tr><td>10</td><td>E</td></tr> <tr><td>10.1</td><td>B</td></tr> <tr><td>10.2</td><td>A</td></tr> </table>	10	E	10.1	B	10.2	A
10	E						
10.1	B						
10.2	A						
<p>Resultatvariablen</p> <p><math>Q_t=9.9</math>   <math>F_t=10.4</math>   <math>A_t=(0.4, 1.6, 2.1, 3.8, 4.0, 5.6, 5.8, 7.2)</math></p>							

## Simulationsende

<p>Zustandsvariablen</p> <p><del><math>Q=3</math>   <math>B=1</math>   <math>a_i=8</math>   <math>b_i=5</math>   <math>t_{\text{last}}=7.2</math></del></p>	<p>Simulatorzustand</p> <p><math>t=10</math>   EL</p> <table border="1" style="background-color: #00b050; color: white;"> <tr><td><del>10.1</del></td><td><del>B</del></td></tr> <tr><td><del>10.2</del></td><td><del>A</del></td></tr> </table>	<del>10.1</del>	<del>B</del>	<del>10.2</del>	<del>A</del>
<del>10.1</del>	<del>B</del>				
<del>10.2</del>	<del>A</del>				
<p>Ausgabe</p> <p><math>Q_t=12.7/10=</math>   <b>1.27</b></p> <p><math>F_t=10.4/5=</math>   <b>2.08</b></p>					

## Einige Beobachtungen für dieses Beispiel

- Werte von  $Q$  und  $B$  werden implizit in  $a_i$  und  $b_i$  codiert  
( $Q = \max(0, a_i - b_i - 1)$ ,  $B = \delta(a_i > b_i)$ )
- Länge der Ereignisliste auf maximal 3 Elemente beschränkt  
(ein Element pro Ereignistyp)
- Es werden keine Elemente aus der Ereignisliste gelöscht
- $At$  speichert alle bisherigen Ankunftszeiten
  - Länge wächst mit steigender Simulationszeit/Ankunftsanzahl
  - Benötigt werden aber nur Ankunftszeiten  $b_i, \dots, a_i$   
andere Speicherstruktur wäre sinnvoll,  
aber Anzahl zu speichernder Elemente ist a priori unbekannt

Grundsätzliches Vorgehen beim einfachen Beispielmmodell ist auf komplexere Modelle übertragbar, aber komplexere Modelle verlangen komplexere Datenstrukturen

Beispiel:

- Kunden haben Eigenschaften, die die Bedienzeit beeinflussen
- Jeder Kunde wird durch eine Datenstruktur  $K_d$  zur Codierung der Eigenschaften dargestellt
- Population im System ist eine Menge von Kunden (Liste von Variablen vom Typ  $K_d$ )
- Bei Ankunft eines Kunden muss eine Variable vom Typ  $K_d$  initialisiert werden
- Verlässt ein Kunde das System, so wird die zugehörige Variable nicht mehr benötigt

auch hier dynamische Belegung und Freigabe von Speicherplatz!

## Zeitablauf in der Simulation:

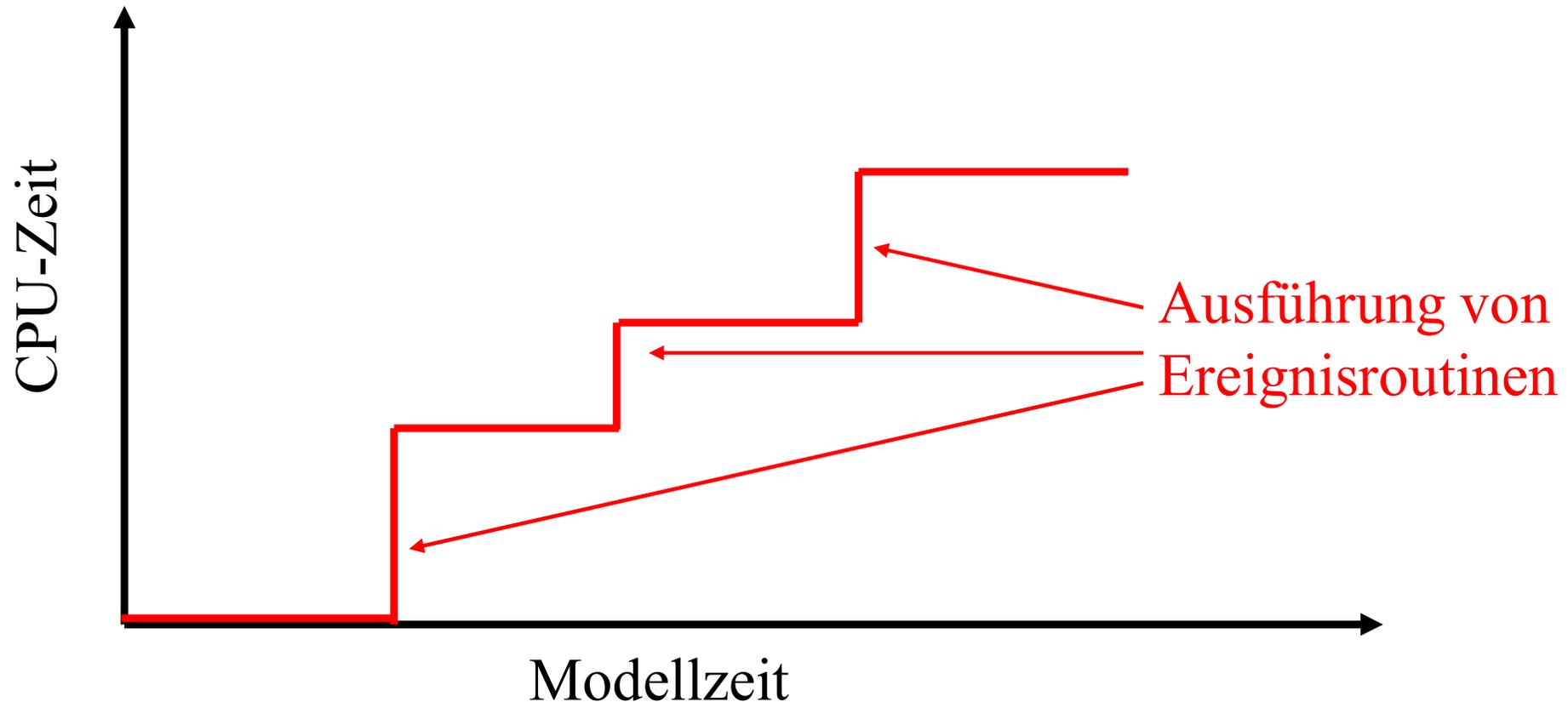
- Zeit läuft diskontinuierlich in Sprüngen ab
  - Zustand ist nur zu Ereigniszeitpunkten definiert (d.h. Zustand direkt nach Eintreten des atomaren Ereignisses)
  - Zwischen Ereigniszeitpunkten ist der Zustand nicht (vollständig) definiert
  - Falls der Zustand zu einem Zeitpunkt  $t$  relevant ist, so ist für diesen Zeitpunkt ein Ereignis vorzumerken
  - Da mehrere Ereignisse zu einem Zeitpunkt stattfinden können, kann das System zu einem Zeitpunkt mehrere Zustände haben
- Ein Ablauf eines Simulators über ein Zeitintervall  $[0, T]$  wird als **Trajektorie** bezeichnet
  - In stochastischen Simulationen können (unendlich) viele unterschiedliche Trajektorien auftreten

## Zeitbegriffe in der Simulation:

Es gibt eine Reihe unterschiedlicher Zeitbegriffe

- **Objektzeit**, jene Zeit, in der (zumindest hypothetisch) der Betrieb des realen Systems abläuft
- **Modell-/Simulationszeit**, jene Zeit, die im Programm „Simulator“ manipuliert wird (indem  $t$  gesetzt wird);
  - Modellzeit imitiert Objektzeit, d.h. ist identisch zu ihr (bis auf Translation, etwa Start bei „0“)
- **Realzeit** des Simulationsprogramms, (das als Programm real in einem Zeitintervall abläuft) ist für uns belanglos
- **Ausführungszeit/CPU-Zeit** (des Simulationsprogramms), jene Zeit, welche die CPU zur Ausführung benötigt
  - CPU-Zeit ist Teil des Ressourcenbedarfs
  - Simulatoren sind notorische Langläufer

# Zusammenhang Modellzeit-CPU-Zeit



Auf den ersten Blick verwirrender Zusammenhang:

- Immer dann, wenn Modellzeit verstreicht, verstreicht keine CPU-Zeit
- und umgekehrt

- Endliche Modellzeitintervalle (zwischen Ereignissen) werden in verschwindender CPU-Zeit nachvollzogen
- Verschwindende Modellzeitintervalle (zu Ereigniszeitpunkten) werden in endlicher CPU-Zeit realisiert

Ausführungszeit eines Simulators ist

- von der Anzahl der simulierten Ereignisse,
- der Komplexität der zugehörigen Ereignisroutinen
- und nicht allein von der verstrichenen Modellzeit abhängig

Erhöhung der Effizienz muss an der Ereigniszahl und -komplexität ansetzen

⇒ am Abstraktionsniveau unseren mentalen Modells

Im allgemeinen gilt:

**höheres Abstraktionsniveau → weniger Ereignisse**

## 2.3 Spezifikation von Simulatoren

Vorgestelltes Prinzip der ereignisdiskreten Simulation ist strukturiert und verständlich, aber Programmierung von Simulatoren erfordert Abstraktion und ist damit komplex!

Grundsätzliche Fragestellungen

- Welche sprachlichen Konstrukte (Syntax + Semantik) sind notwendig, um Simulatoren zu implementieren?
- Wie werden diese Konstrukte umgesetzt oder realisiert?
- Welche Ziele verfolgt die Simulation?  
(Systemanalyse, Anbindung an reale Systeme, Animation)
- Welche Anwendungsgebiete sollen abgedeckt werden?  
(universell, spezifisch)
- Welche Darstellungsformen werden gewählt?  
(programmiersprachlich, graphisch, menügesteuert, ...)

Heutige Simulationssoftware oft mit graphischer Benutzerschnittstelle

- Modellspezifikation aus Standardkomponenten (Baukastenprinzip anwendungsspezifisch) oder
- graphische Spezifikation der Ereignisroutinen (ähnlich Flussdiagrammen)

Umsetzung erfolgt dann in eine Programmiersprache, so dass ein lauffähiges Simulationsprogramm entsteht

# Ereignisliste als spezielle Liste

## Primäre Operationen:

- Herauslesen des ersten Elements
- Einfügen eines Elements nach Schlüssel

## Seltener gebraucht

- Löschen oder Suchen eines beliebigen Elements

## Datenstruktur für die Listenelemente

- Einfaches Feld mit zwei Elementen
  - $t$  zur Darstellung der Ereigniszeit  
Typ (nicht negative) reelle Zahl
  - $tp$  zur Darstellung des Ereignistyps  
i.d.R. als ganze Zahl codiert

## Aufwand der Operationen bei Listenlänge $n$

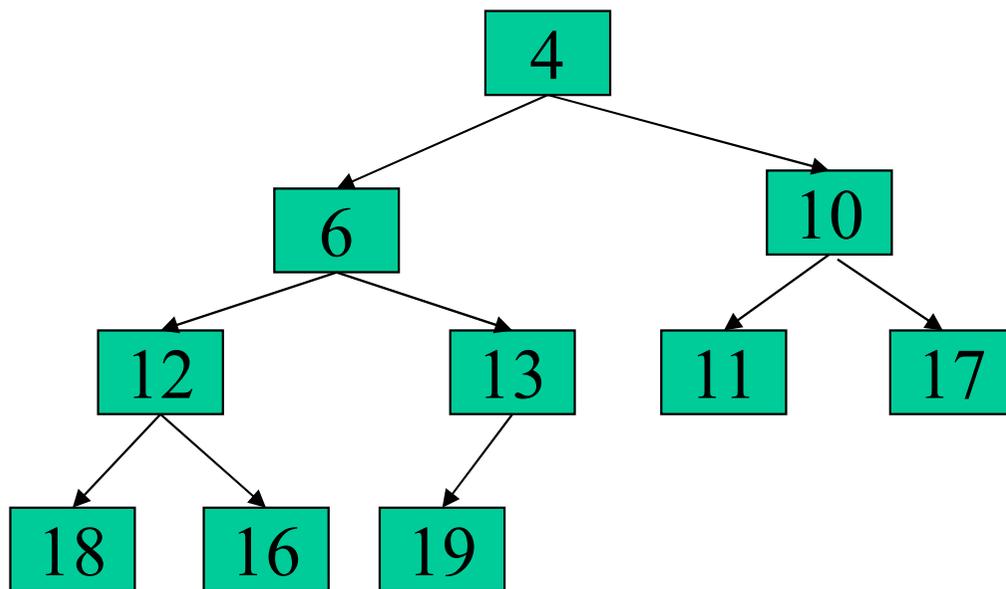
- Herauslesen  $O(1)$
- Einfügen und Löschen  $O(n)$

# Andere Möglichkeiten zur Realisierung von Ereignislisten

Darstellung als Heap (Binärbaum):

- Wurzel beinhaltet minimales Element
- Vorgänger kleiner als Nachfolger
- Unterste Ebene von links gefüllt

Bsp. nur Werte von  $t$  angegeben



Auslesen des ersten Elements:

- Wurzel entfernen
- Letzten Knoten der untersten Ebene auf die Wurzel setzen
- Wurzel solange mit kleinstem Nachfolger vertauschen, bis kein kleinerer Nachfolger mehr existiert

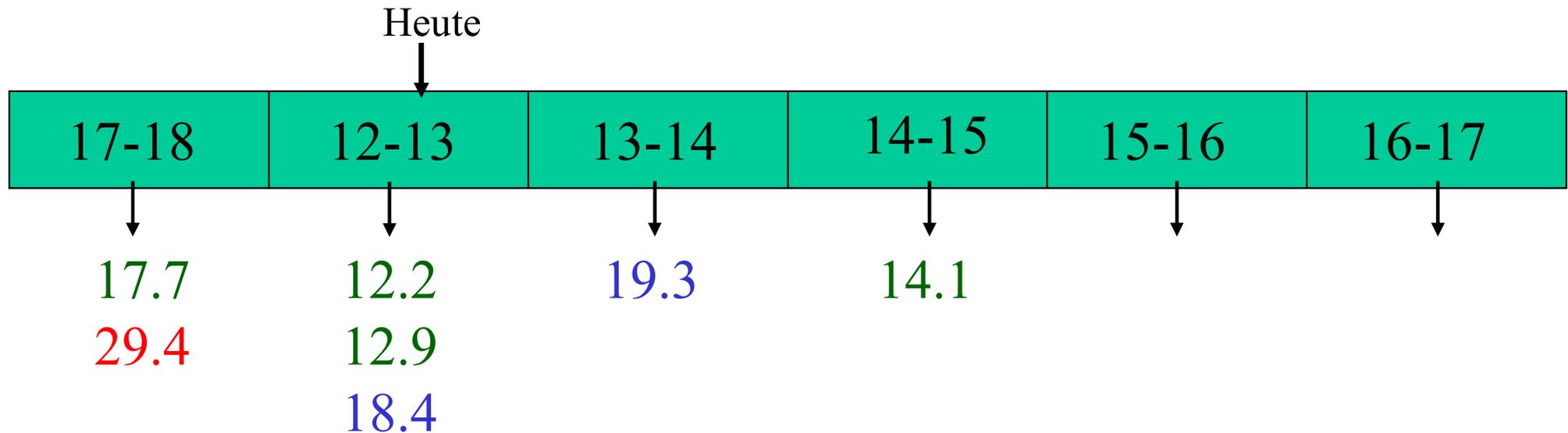
Einfügen eines Elements:

- Einhängen als letztes Element der untersten Ebene
- Solange mit Vorgänger vertauschen, bis dieser kleiner

Aufwand Auslesen und Einfügen jeweils  $O(\log n)$ !

## Darstellung als Kalender: Datenstruktur Array von Listen

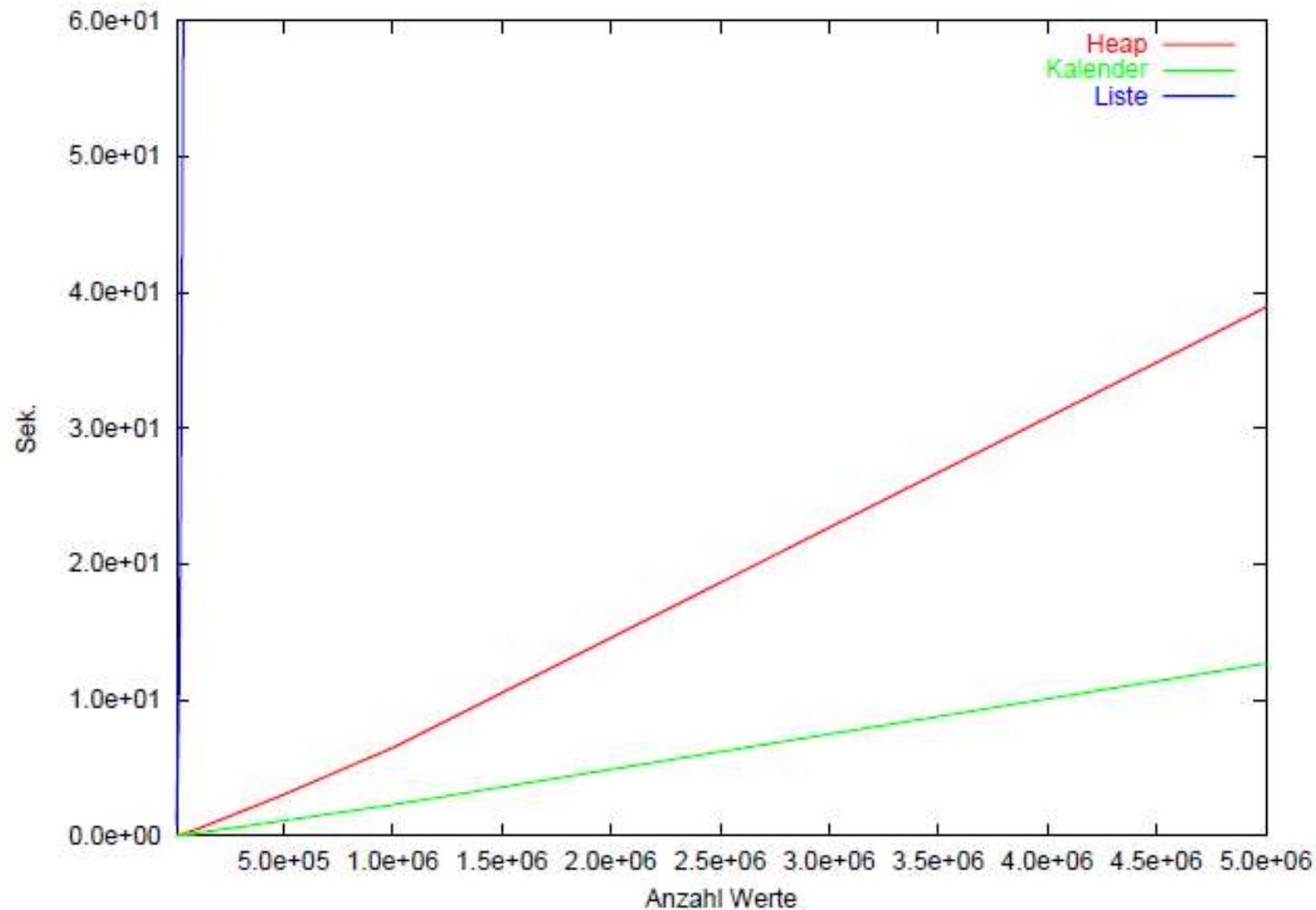
- Jede Zelle enthält Ereignisse aus einem festen Zeitraum
- Ereignisse aus jedem Zeitraum werden in geordneter Liste gehalten



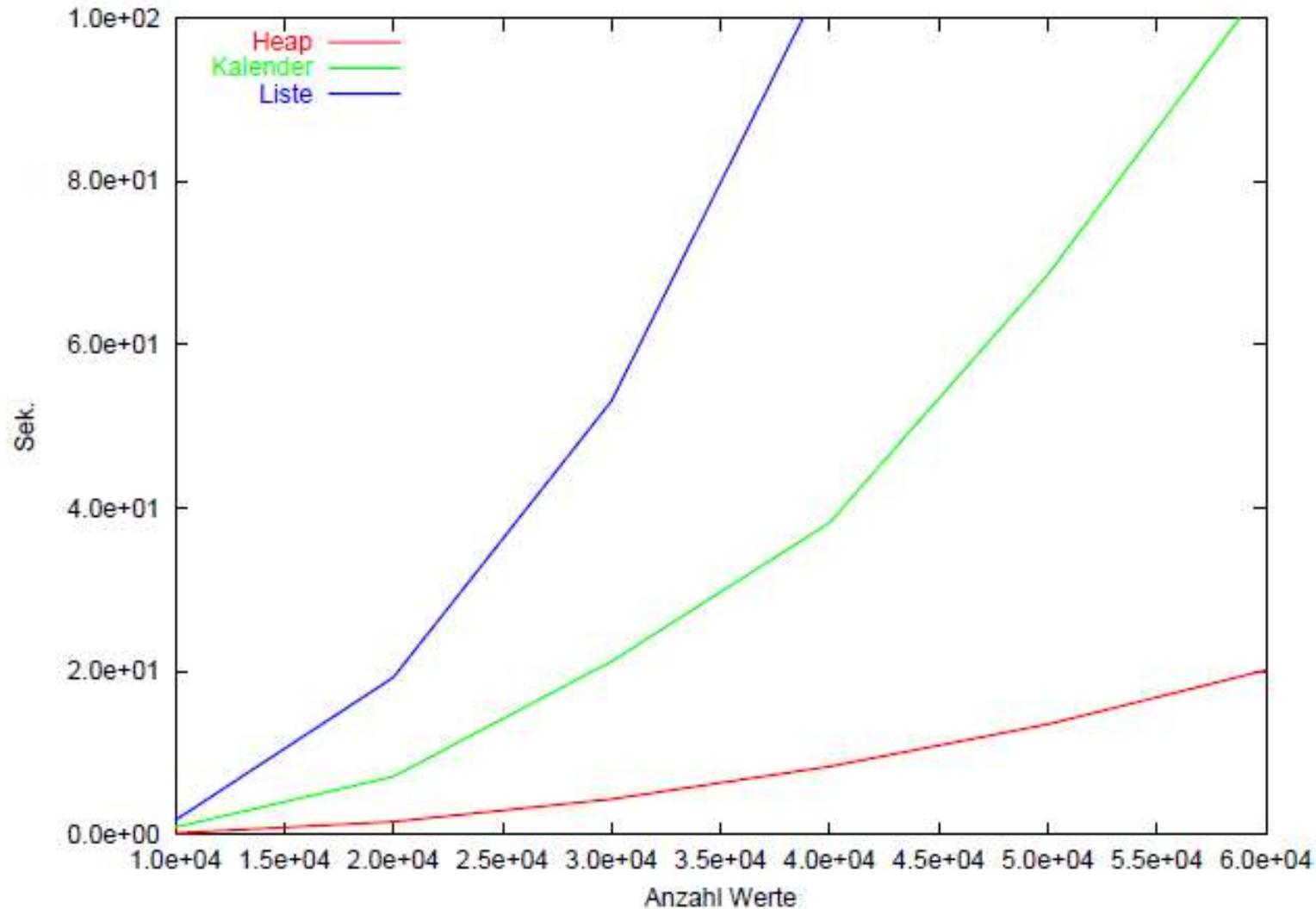
- Dynamische Anpassung der Zellbreite, so dass ca. 75% der Einfügeoperationen „in diesem Jahr stattfinden (Jahr hier 12-18)“
- Dynamische Anpassung der Zellzahl
  - Halbieren, falls Anzahl Elemente  $< 0.5 \cdot$  Anzahl Zellen
  - Verdoppeln, falls Anzahl Elements  $> 2 \cdot$  Anzahl Zellen

Experimente zeigen fast konstanten Aufwand für Entfernen und Einfügen!

# Laufzeit für das Einfügen exp. verteilter Werte und seq. Auslesen



# Laufzeit für das Einfügen exp. verteilter Werte und zufälliges Auslesen



Generierung von Zufallszahlen:

Methoden zur Generierung von Zufallszahlen vorgegebener Verteilungen, also

```
atime = ziehe_zz(Verteilungstyp, Parameter) ;
```

Später mehr zur Realisierung dieser Funktionen

Weitere Funktionen zur Realisierung von Simulationen:

```
plane(Ereignistyp, Ereigniszeit) ;
```

Zum Einhängen eines Ereignisses in die Ereignisliste

```
ereignis = erstes_ereignis() ;
```

Zum Auslesen des ersten Ereignisses aus der Liste

```
zeit = t ;
```

Zum Auslesen der Zeit des nächsten Ereignisses in der Ereignisliste

**Funktionen existieren so oder ähnlich in vielen Simulationssprachen!**

## Auswertung von Simulationsergebnissen:

- Schreiben von Traces
  - Ausgabe von Informationen zu Ereignissen
  - Spezielle Ereignisse zur Beobachtung definierbar
  - Grundlage der Animation
- Erstellen aggregierter Statistiken
  - Definition von Resultatvariablen  
(z.B. Population in einer Warteschlange, Auslastung einer Ressource)
  - Bestimmung von Mittelwerten + statistische Auswertung  
(Standardabweichung, Konfidenzintervalle)
  - Graphische Darstellung ermittelter Werte

## 2.4 Struktur von Simulatoren

Vorgestellte Konstrukte reichen aus, um ereignisdiskrete Simulatoren auf dem bisher beschriebenen Niveau zu realisieren!

Dazu notwendig

- Aufbau der Datenstruktur zur Abbildung der statischen Struktur
- Definition aller Ereignisse und Realisierung der zugehörigen Ereignisroutinen

Ansatz wird als „**event scheduling**“ bezeichnet

Für komplexe Probleme ist das Vorgehen sehr aufwändig und fehleranfällig, da

- sämtliche Ereignisse und deren (globale) Auswirkungen erkannt und kodiert werden müssen
- flache aber stark vernetzte und damit unübersichtliche Modelle entstehen

## Geht es einfacher/besser/sicherer ? Wenn ja, wie?

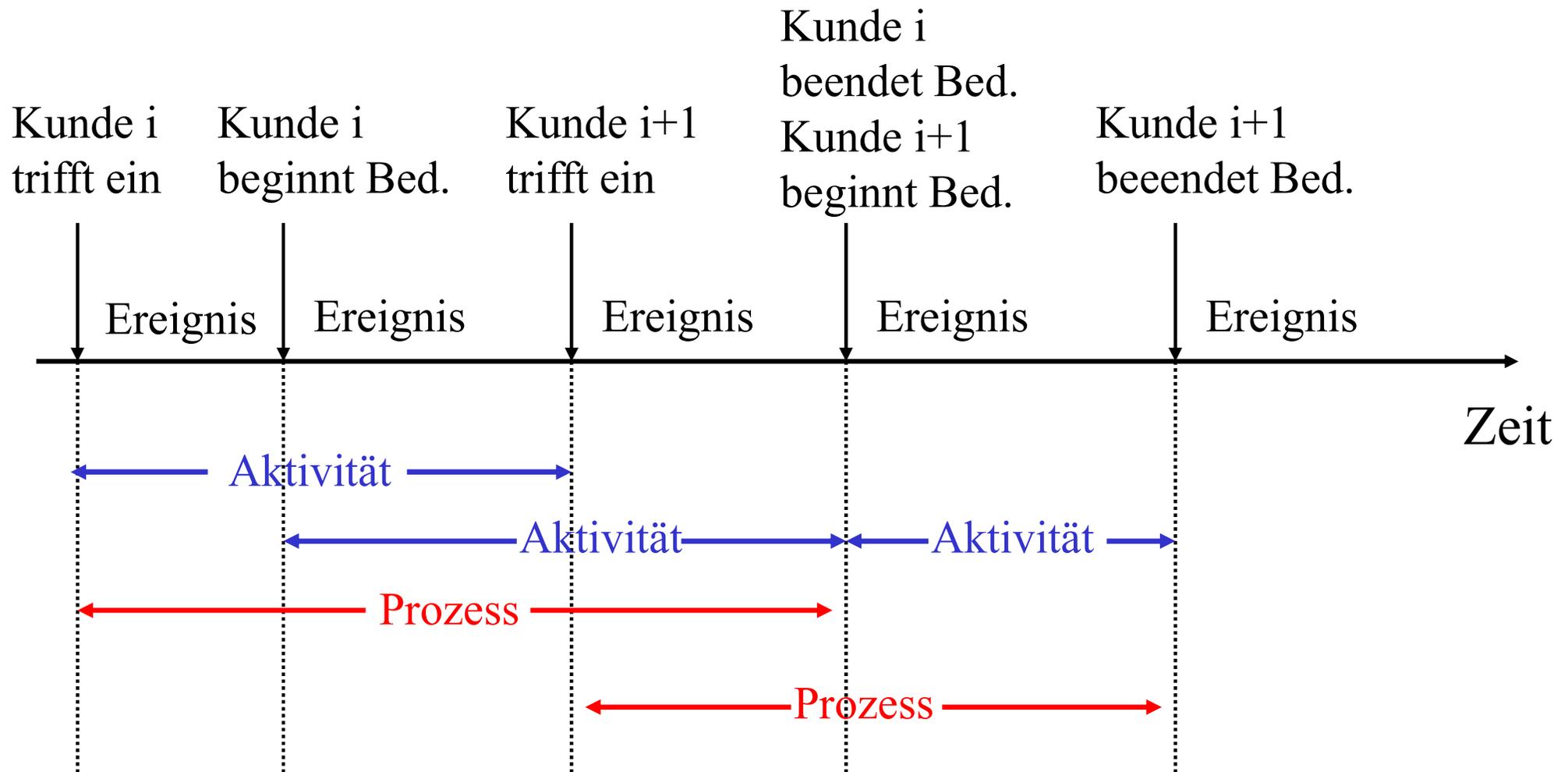
Basis der Modellierung ist unser mentales oder semi-formales Modell!

- Einfachere Simulationsmodelle müssen „näher“ am mentalen Modell sein
- Viele Ereignisse haben „lokale“ Auswirkungen, die sich nur auf einzelne Zustandsvariablen beziehen
- Denken in elementaren Ereignissen ist nicht abstrakt genug
- Ereignisse müssen zusammengefasst/strukturiert werden

Verschiedene Strukturierungsmethoden existieren in der Literatur, weit verbreitet ist die Unterteilung in

- event scheduling
- activity scanning
- process interaction
- agentenorientiert

# Unterscheidung am Beispiel des einfachen Schalters:



Entsprechen der Abbildung

- drei „Sichten“ und
- zugehörige konzeptuelle Rahmen der Zeitsteuerung
- mit steigender Abstraktion

Verknüpfung zwischen Zustand und Zeit durch:

1. **Ereignis:** Veränderung der Zustandsvariablen zu bestimmten Zeitpunkten
2. **Aktivität:** Menge von Operationen während eines Zeitintervalls
3. **Prozess:** Folge von Aktivitäten eines Objekts über eine Zeitspanne

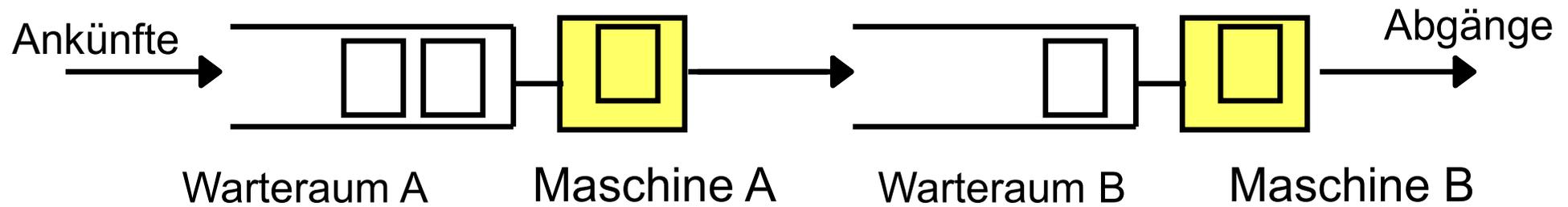
Darüber hinaus weitere Freiheitsgrade für den Modellierer:

Was ist aktiv, was ist passiv?

Im Ergebnis identisch, aber unterschiedliche Sichtweisen des Systemverhaltens

- Kunden als aktive Elemente, Bediener als passive Ressourcen  
Kunde bewegt sich von einem Bediener zum nächsten
- Bediener als aktive Elemente, Kunden nur passiv  
Bediener übergibt Kunden an den nächsten Bediener
- Kombinationen aus beiden Sichtweisen

# Vorstellung der Konzepte an einem einfachen Beispiel



## Annahmen:

- Aufträge passieren nacheinander Maschine A und B
- Bearbeitungszeiten  $T_A$  und  $T_B$  (Zufallsvariablen)
- $T_I$  Zeitabstand zwischen zwei Ankünften (ebenfalls ZV)
- Maschinen besitzen unbeschränkte Puffer und bedienen nach FCFS
- nach Bearbeitungsende in B, verlässt ein Auftrag das System

## Analyseziele (Beispiele):

- Durchlaufzeiten der Aufträge (kundenorientiert)
- Anzahl Aufträge im Warteraum (ressourcenorientiert)
- Auslastung der Maschinen (kostenorientiert)

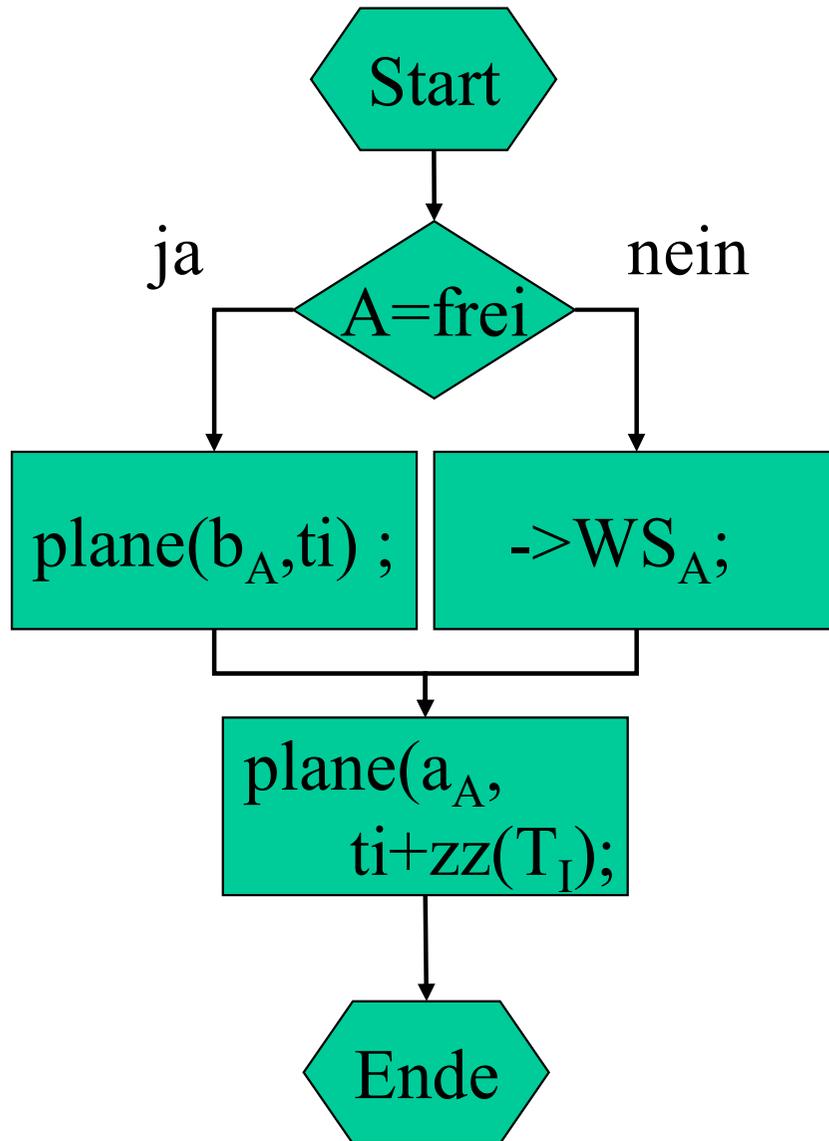
## Statische Modellstruktur:

- Zwei Maschinen A und B als permanent existierende Objekte
  - jede Maschine im Zustand frei oder belegt
  - Warteräume vor den Maschinen als FIFO-Queues (von Aufträgen)
- Aufträge als temporär existierende Objekte

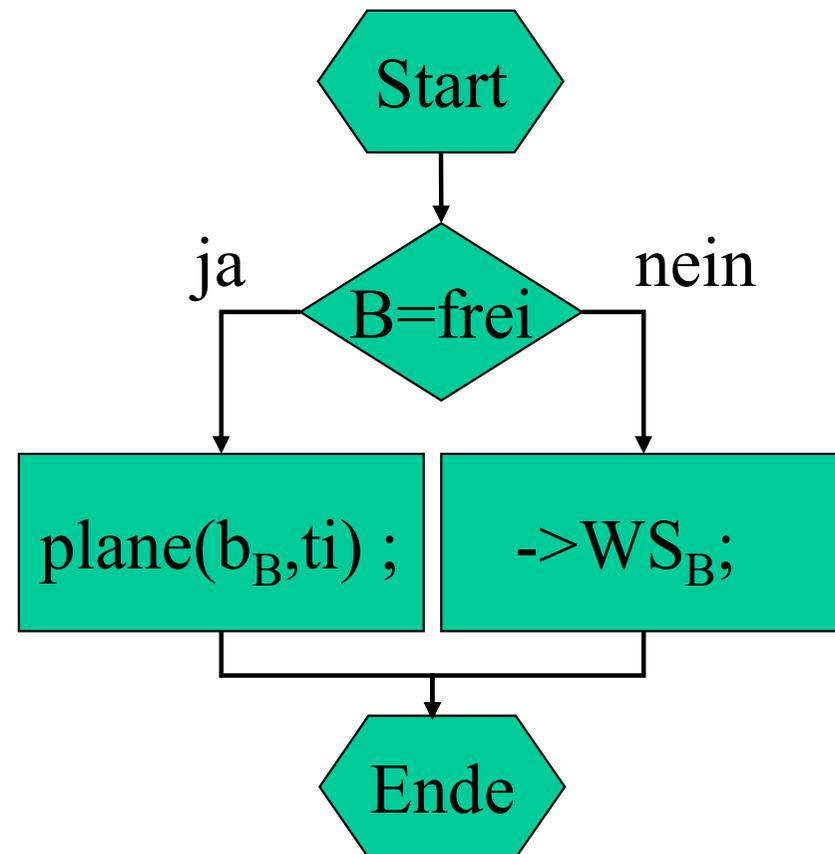
## Ereignisse im event scheduling-Ansatz (aktive Aufträge)

1. Ankunft an Maschine A ( $a_A$ )
2. Ankunft an Maschine B ( $a_B$ )
3. Bearbeitungsbeginn an Maschine A ( $b_A$ )
4. Bearbeitungsbeginn an Maschine B ( $b_B$ )
5. Bearbeitungsende an Maschine A ( $e_A$ )
6. Bearbeitungsende an Maschine B ( $e_B$ )

## Ereignis $a_A$

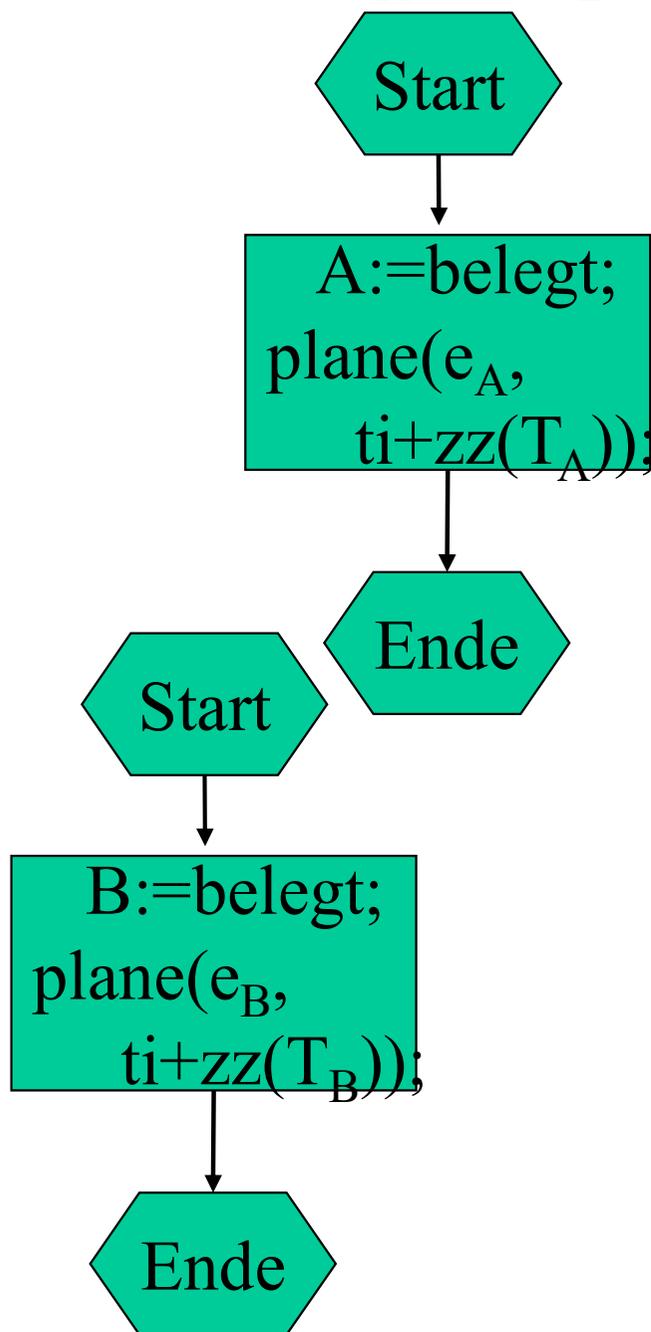


## Ereignis $a_B$

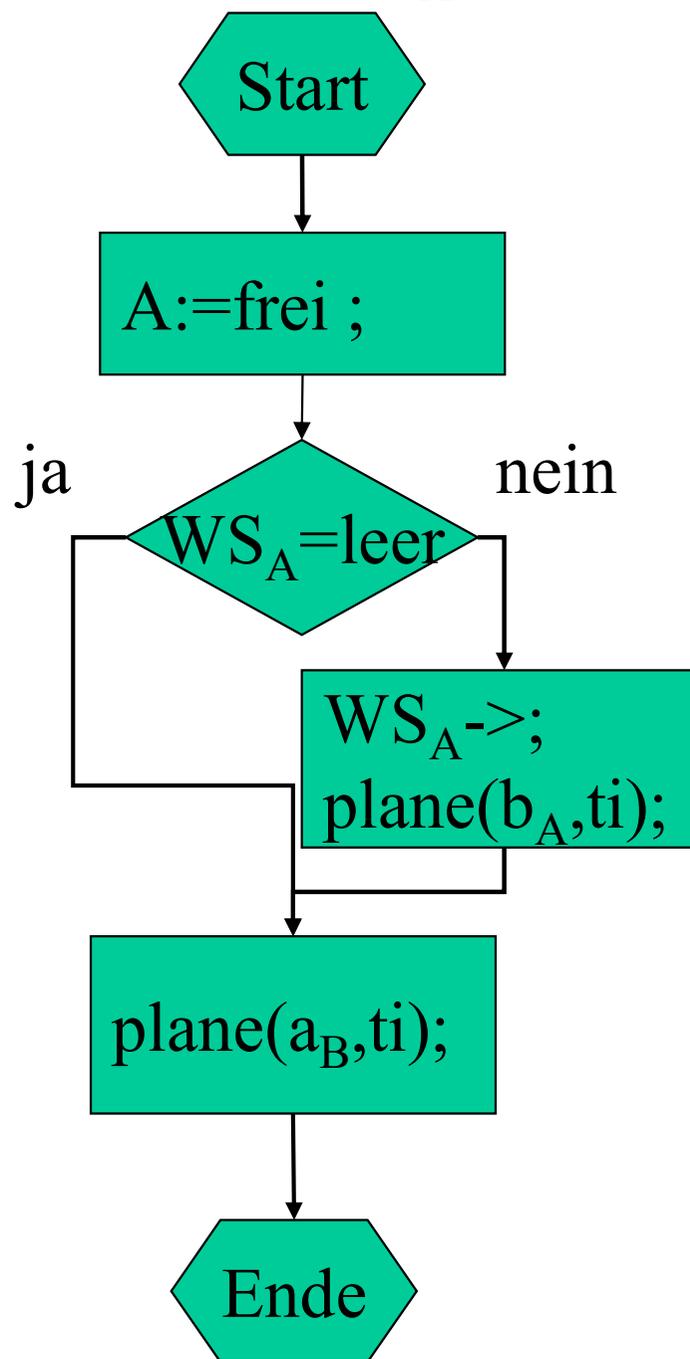


Abkürzende Schreibweisen:  $ti$  für  $simzeit()$ ,  $->WS_A$  für Auftrag in Warteschlange Maschine A,  $zz(T_A)$  für Generierung einer Zufallszahl gemäß  $T_A$ .

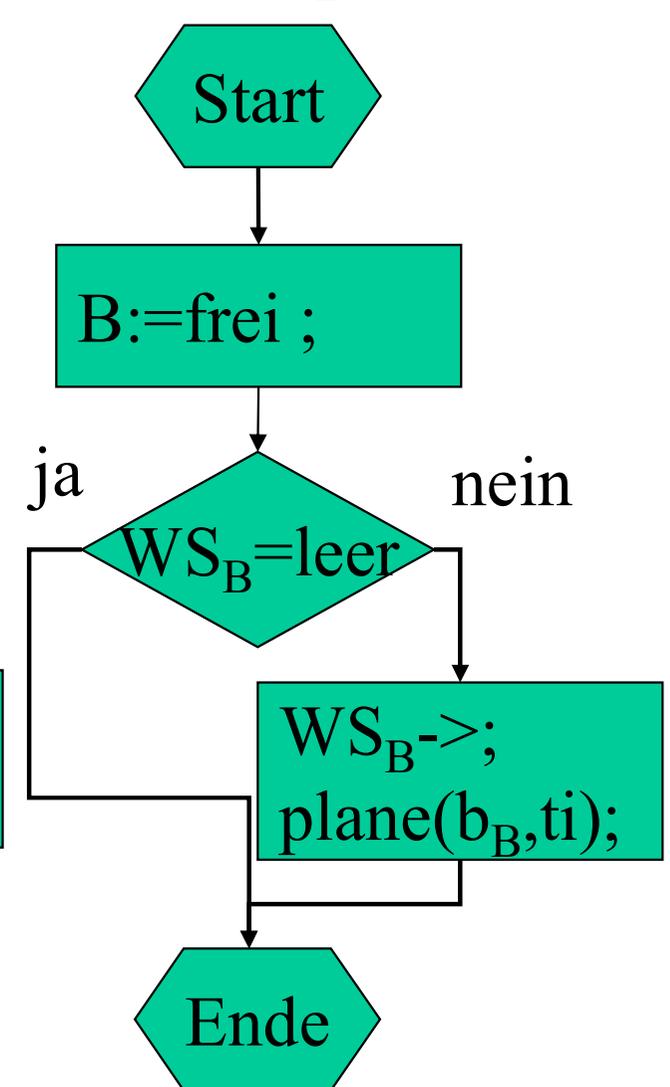
Ereignis  $b_A$  und  $b_B$



Ereignis  $e_A$



Ereignis  $e_B$



Was fehlt noch?

Wie beginnt der Ablauf und wann endet er?

Simulationshauptroutine, die

- Zustandsvariablen initialisiert  
(z.B.  $t_i := 0$ , Maschinen frei, Warteräume leer)
- erstes Ereignis plant ( $\text{plane}(a_A, \text{zz}(T_A))$ )
- Simulationslauf beendet  
(einfachste Form Einhängen eines Ereignisses *Simulationsende*)

und zu Beginn der Simulation aufgerufen wird.

# Beobachtung der Simulation und Resultatermittlung

Aufzeichnung von Beobachtungen in Verbindung mit Ereignissen verbunden mit

- Aufträgen (z.B. Verweilzeiten, Wartezeiten, ...)
  - Auftrag „merkt sich“ in der Ereignisroutine, wann er angekommen ist
  - Bei Abgang des Auftrags wird in der Ereignisroutine dafür gesorgt, dass die verbrauchte Zeit gespeichert wird
- Maschinen (z.B. Auslastung, Population, ...)
  - Station hat für ein bestimmtes Zeitintervall einen festen Zustand
  - Zustand ändert sich durch Ereignisse
  - In der Ereignisroutine wird der alte Zustand plus dessen Dauer und der neue Zustand plus dessen Beginn gespeichert

I.a. werden Daten in aggregierter Form (z.B. als Mittelwerte) gespeichert und ausgewertet (später dazu mehr)

## Process interaction-Ansatz

Weitere Strukturierung durch Zusammenfassung von Ereignissen

Im Beispiel:

Ein (jeder!) Auftrag

- trifft an Maschine A ein und stellt sich an (Ereignis)
- kommt dran (Ereignis)
- ist fertig und geht weg (Ereignis)
- trifft an Maschine B ein und stellt sich an (Ereignis)
- kommt dran (Ereignis)
- ist fertig und geht weg (Ereignis)

Sechs Ereignisse bestimmen „Leben“ eines Objekts „Auftrag“

**Dies ist nur eine Strukturierungsmöglichkeit!**

## Alternative zur Struktur nach Aufträgen, Struktur nach Maschinen:

- Die Maschine (A oder B)
  - beginnt eine Bedienung (Ereignis)
  - beendet diese später (Ereignis)
  - beginnt die nächste Bedienung (Ereignis)
  - .....
- Die Umwelt
  - schickt einen Auftrag in das System (Ereignis)
  - schickt später einen weiteren Auftrag (Ereignis)
  - ...

Eine potenziell unbegrenzte Zahl von Ereignissen bestimmt das Leben der Objekte Maschine und Umwelt

Weiter auf dem eingeschlagenem Weg

⇒ Zusammenfassung größerer Ereignismengen

Lohnenswerter Versuch:

- Verschiedene „Objekt-Leben“ als Strukturierungsmittel nutzen (Einzelereignisse sind „Unterstrukturen“)

Hoffnung:

- Abstraktere, näher an der Vorstellung liegende Modellwelt

Vorstellung:

- Zeitabhängiges Verhalten eines dynamischen Systems ist generiert durch eine Menge dynamischer Objekte (Prozesse), deren jedes (jeder) sich entsprechen festgelegter Vorschriften (Prozessmuster) verhält

⇒ **System „paralleler“ Prozesse**

Prozesse laufen unabhängig ab und müssen von Zeit zu Zeit interagieren

Im Beispiel:

- „Bearbeitetwerden“ im Auftrags-Prozess und
  - „Bearbeiten“ im Maschinen-Prozess
- Vorgänge in beiden Prozessen simultan

⇒ **Prozesse müssen sich synchronisieren**

Anforderungen zur Beschreibung solcher Systeme

1. Prozessmuster notieren
2. Prozesse (bestimmter Muster) starten
3. Prozesse „interagieren“ lassen

# Unterscheidung in **temporäre** und **permanente** Prozesse

Ablauf eines Prozessmusters:

- Operationen ausführen
- Warten (d.h. nichts tun, Zeit verstreichen lassen)
- Operationen ausführen
- ....

Am Beispiel Auftrag

- Stell dich an
- Warte bis du dran bist (1.)
- Warte bis Bedienzeit abgelaufen (2.)

(1.) und (2.) beschreiben mögliche Wartebedingungen

1. Warten auf eine Bedingung (weiter, wenn Bedingung erfüllt)

2. Warten für eine Zeit (weiter, wenn Zeit abgelaufen)

Programmiersprachliche Formulierung in der Form:

`wait_until` (Boolsche-Bedingung)

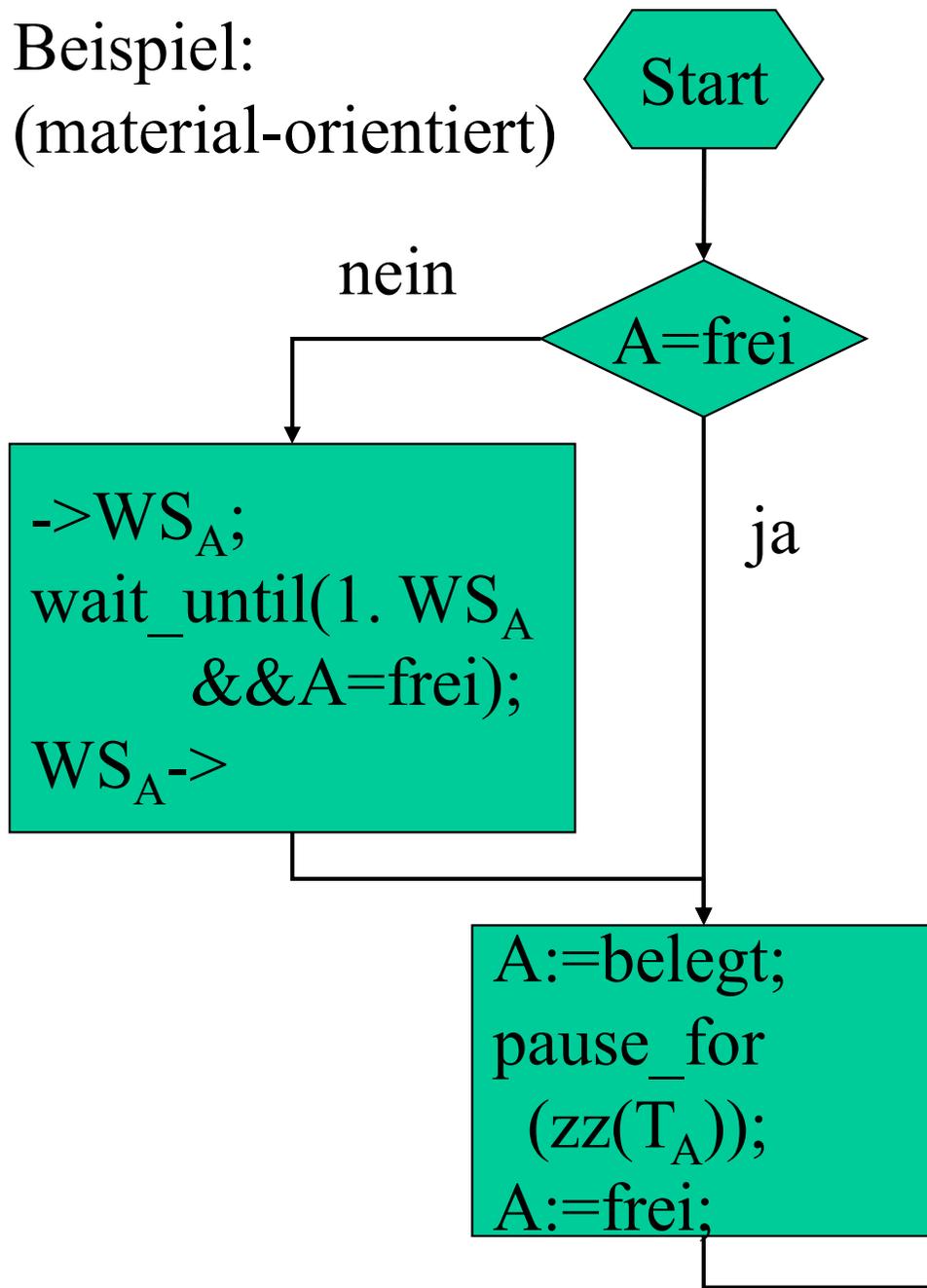
- Formuliert die 1. Bedingung
- Prozess wird inaktiv, bis Bedingung erfüllt
- Bedingung wird durch Zustandsänderung, d.h. Ereignisse in anderen Prozessen erfüllt

`pause_for` (t)

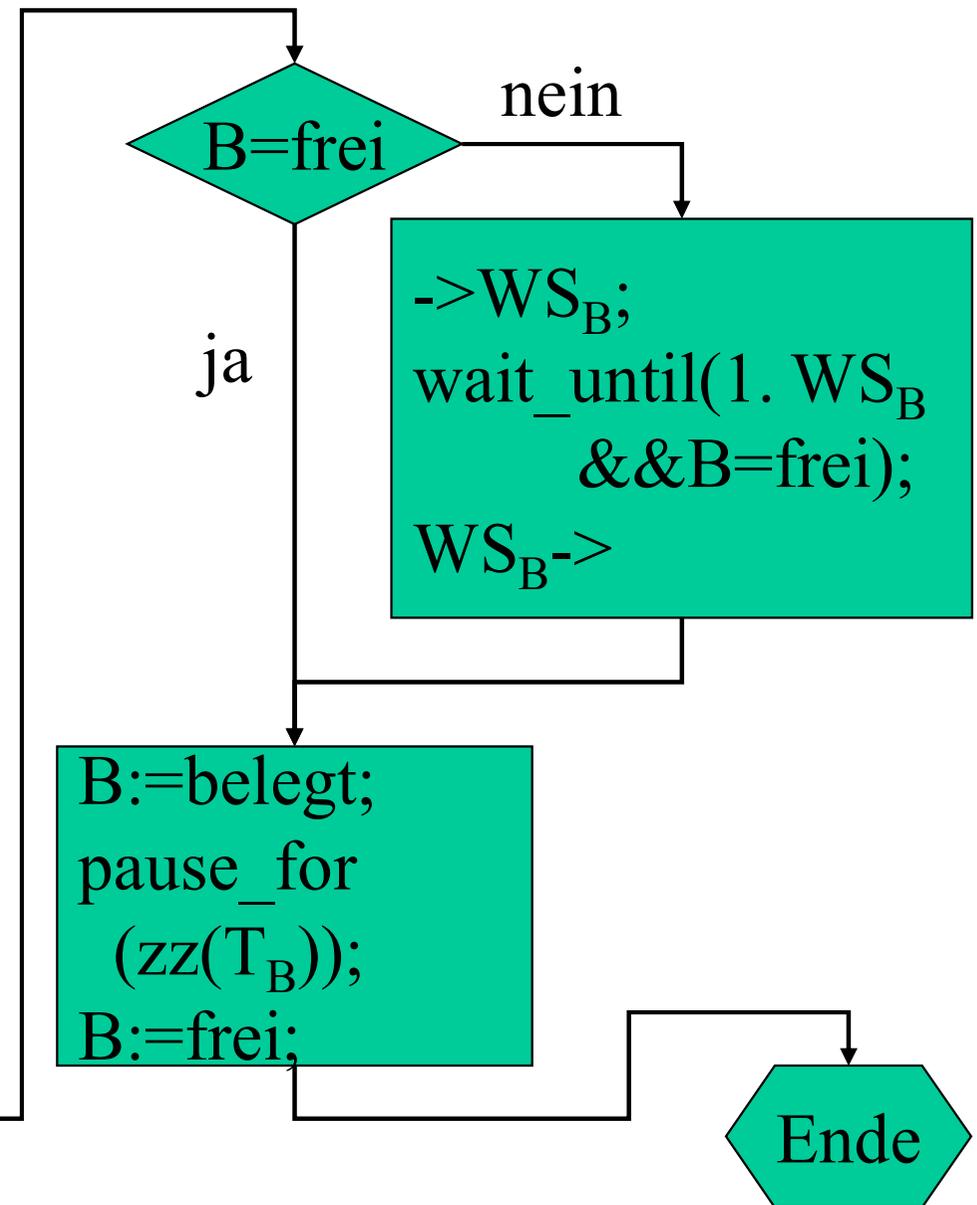
- Formuliert die 2. Bedingung
- Prozess wird für die Dauer t inaktiv
- danach wird er von selbst wieder aktiv

In beiden Fällen fährt der Prozess dort fort, wo er aufgehört hat, also hinter der `wait_until/pause_for`-Anweisung

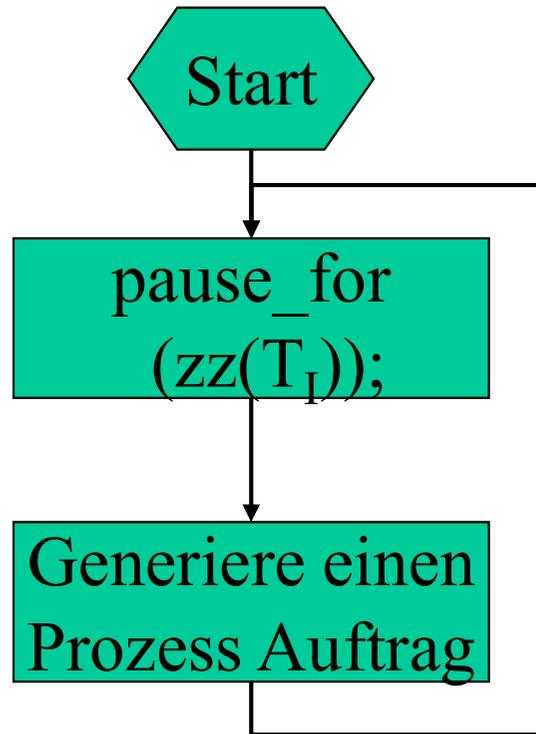
Beispiel:  
(material-orientiert)



Prozessmuster Auftrag



# Generierung von Aufträgen durch Prozess Umwelt



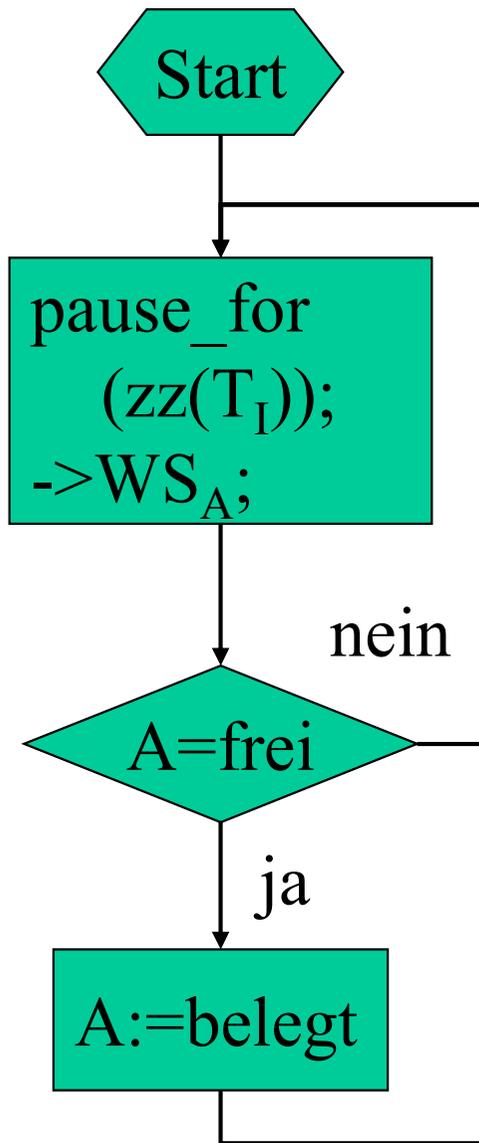
Zum Start der Simulation

- Einen Prozess Umwelt generieren
- Bis zum Simulationsende warten

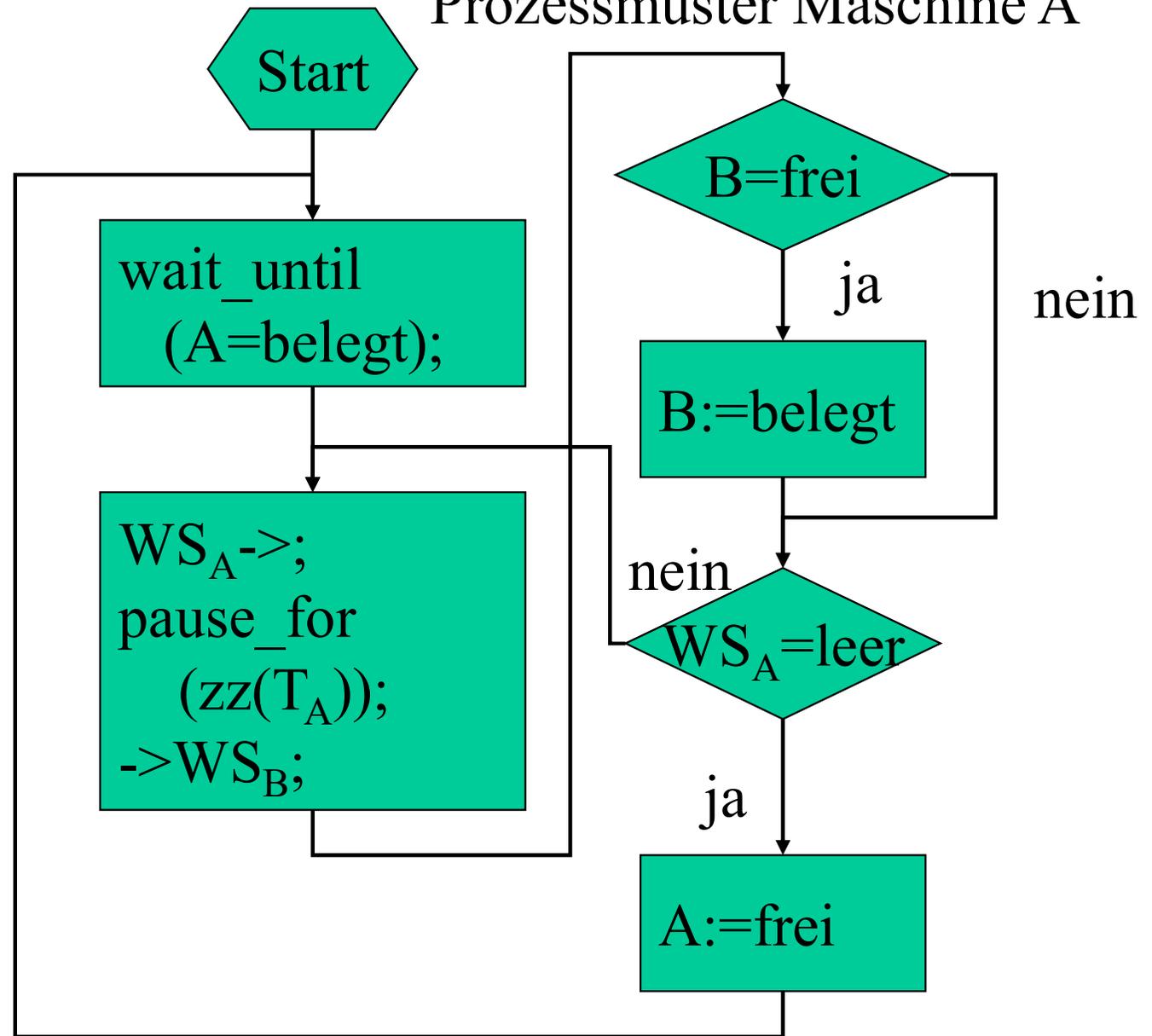
## Einige Beobachtungen

- Prozess Umwelt läuft permanent
- Aufträge sind temporäre Prozesse
- Beobachtung und Speicherung von Resultatwerten muss in den Auftragsprozessen ergänzt werden
- Maschinen treten nur implizit durch ihren Zustand und die Warteräume auf

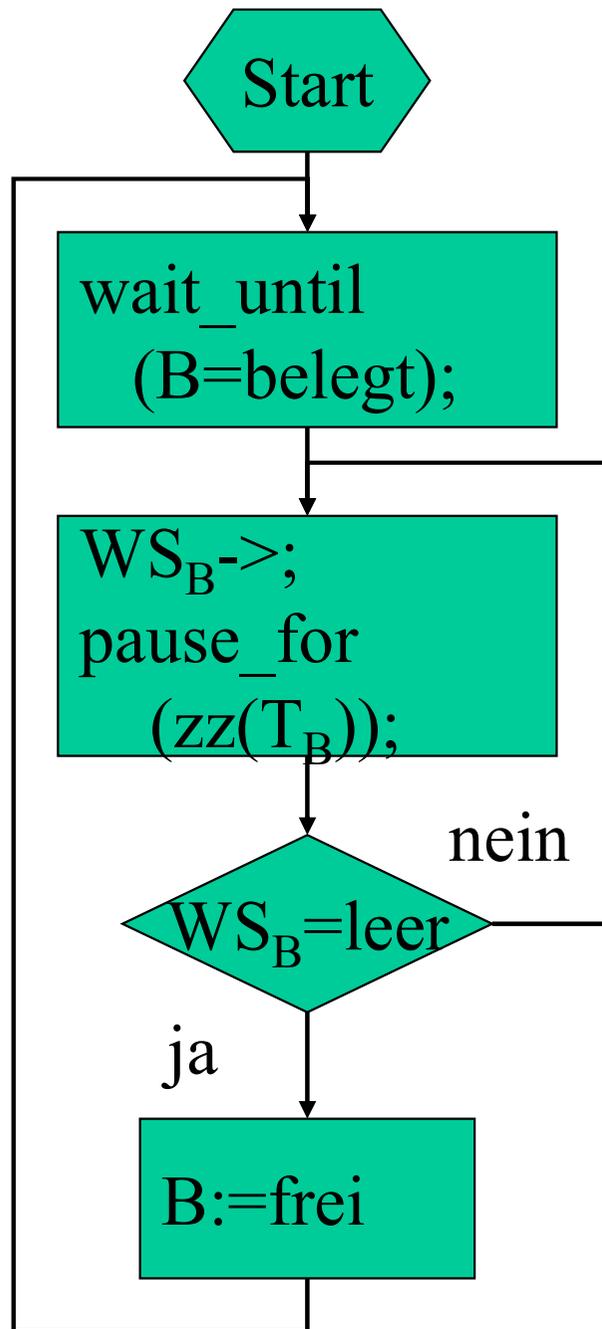
Beispiel:  
(maschinen-orientiert)



Prozessmuster Maschine A



# Prozessmuster Maschine B



## Einige Beobachtungen

- Alle Prozess laufen permanent
- Beobachtung und Speicherung von Resultatwerten muss in den Maschinenprozessen ergänzt werden
- Aufträge treten nur implizit in den Warteräumen auf
- Start des Modells durch Erzeugung der drei Prozesse (Umweltprozess generiert den ersten Auftrag, der die wait\_until-Bedingung der ersten Maschine erfüllt, ....)

# Unterscheidung in maschinen- und material-orientierte Sicht

Im Beispiel beides möglich aber

maschinen-orientierte Sicht  
erlaubt

- unterschiedliche Scheduling-Strategien
- maschinen-orientierte Leistungsmaße (z.B. Auslastung)
- Leistungsschwankungen auf Grund von Fehlern, Ausfällen etc.

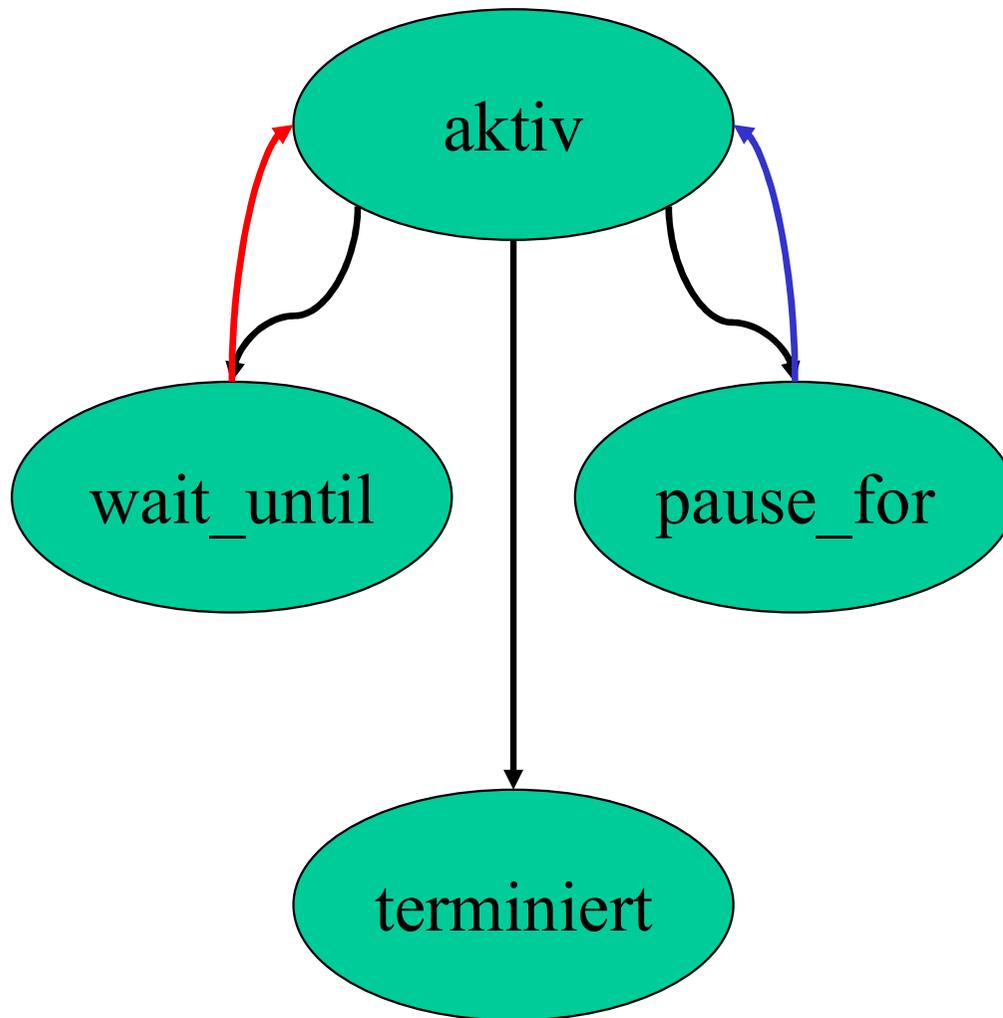
material-orientierte Sicht  
erlaubt

- simultane Ressourcenbelegung
- material-orientierte Leistungsmaße (z.B. Durchlaufzeiten)
- Synchronisation von Prozessen

In realen Modellen oft Kombination von maschinen- und material-orientierter Sichtweise, um Flexibilität zu erlangen.

# Realisierung des process interaction-Ansatzes

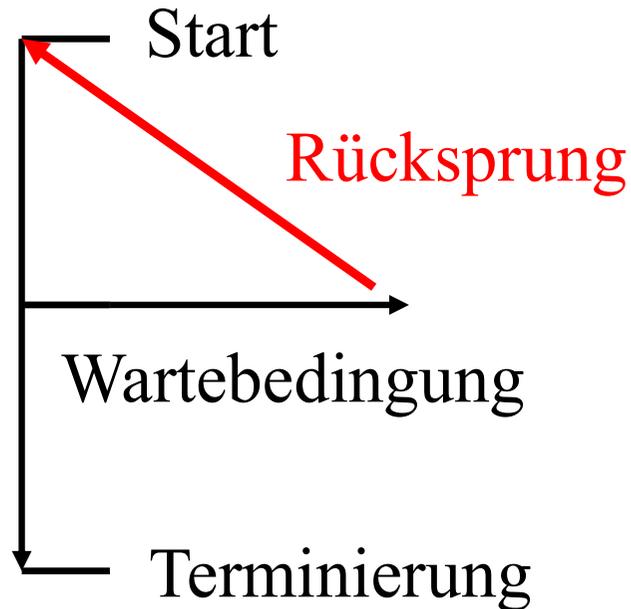
## Prozesszustandsdiagramm



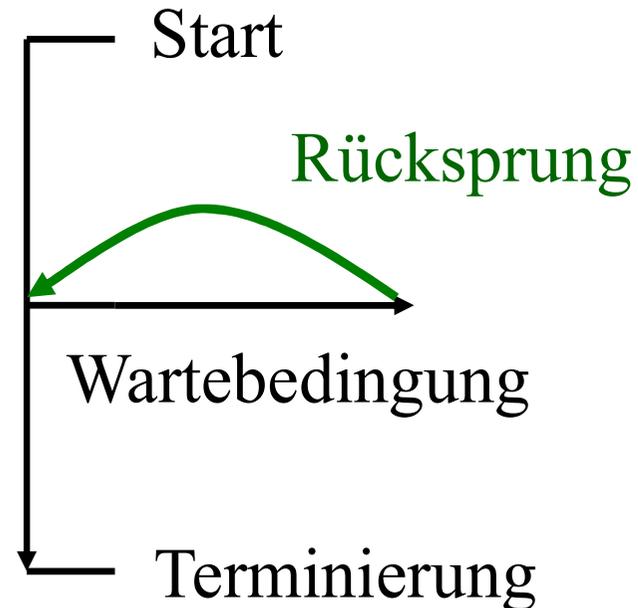
- Mehrere Prozesse laufen „parallel“, d.h. können auch zur gleichen Zeit im Zustand *aktiv* sein
- Realisierung auf Einzelprozessoren bedingt, dass nur jeweils ein Prozess aktiv sein kann (Realisierung auf Multiprozessoren deutlich komplexer, wird hier nicht betrachtet!)
- Parallelität „vorgaukeln“  
Ausführungsreihenfolge zeitgleicher Prozesse kann das Verhalten beeinflussen!

# Wie werden Prozessmuster beschrieben? Als Prozeduren?

Verhalten einer Prozedur



Benötigt wird aber



Realisierung in Programmiersprachen durch Koroutinen, Threads

Beispiele

- Simula67 mit Koroutinenkonzept,
- Java mit Threadkonzept,
- C ohne solches Konzept, aber Zusatzbibliotheken existieren

## Ausführung gleichzeitiger Ereignisse:

Ein einfaches Beispiel mit Prozessen Schalter und Kunde:

Schalter mit Warteschlange WS, Boolescher Variable belegt und Referenz Kd auf einen Kunden

Kunde mit Boolescher Variable dran, Referenz Sc auf den Schalter und Referenz Current auf sich selbst

Schalter

Repeat

wait\_until(WS nicht leer) ; (S1)

Repeat

Kd := WS.first ; (S2)

belegt := true ; (S3)

WS.first.dran := true ; (S4)

wait\_until (not belegt) ; (S5)

Until WS leer ; (S6)

Until (Simulationsende) ;

Kunde

dran := false ; (K1)

Current->Sc.WS ; (K2)

wait\_until(dran) ; (K3)

pause\_for(Bedienz.) ; (K4)

Sc.belegt := false; (K5)

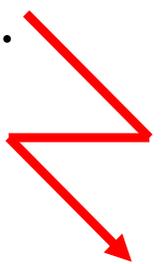
Sc.WS-> ; (K6)

## Erster Versuch der Festlegung einer Semantik:

- Reihenfolge der Ereignisse eines Prozesses müssen beachtet werden
- Zeitgleiche Ereignisse in unterschiedlichen Prozessen können beliebig angeordnet werden

Wir betrachten das System bei der Ankunft des ersten Kunden K.  
Eine mögliche Ereignisreihenfolge:

K1, K2, S1, S2, S3, S4, K3, K4, K5, S5, S6, S2, S3, S4, S5, K6



Schalter ist belegt und referenziert den Kunden, der das System bereits verlassen hat!  
⇒ Deadlock-Situation!!

## Übliche Vereinbarung in prozessorientierten Simulationssprachen:

Ein Prozess läuft so lange, bis er eine hemmende Bedingung erreicht oder terminiert.

Auswahl des nächsten Prozesses aus einer Menge aktiver Prozesse bleibt willkürlich und kann die Semantik beeinflussen!

## Realisierung der Wartebedingungen

### **pause\_for(t):**

- Prozess pausiert für  $t$  Zeiteinheiten und wird danach wieder aktiv
- Ereignis Aktivierung kann für Zeitpunkt  $t_i+t$  eingeplant werden
- Realisierung über Ereignisliste, die alle Prozesse enthält, die pausieren

## Realisierung der Wartebedingungen

### **wait\_until(b):**

- Bedingung b kann über beliebige Zustandsvariablen formuliert werden
  - Damit kann jede Wertzuweisung jedes wait\_until eines Wartenden ändern
  - Damit muss nach jedem Anhalten eines Prozesses jede Wartebedingung überprüft werden
- Extrem aufwändig, führt zu ineffizienten Simulatoren, deshalb in kaum einem Simulationssystem in dieser Allgemeinheit verfügbar

# Alternativen zum ineffizientem wait\_until

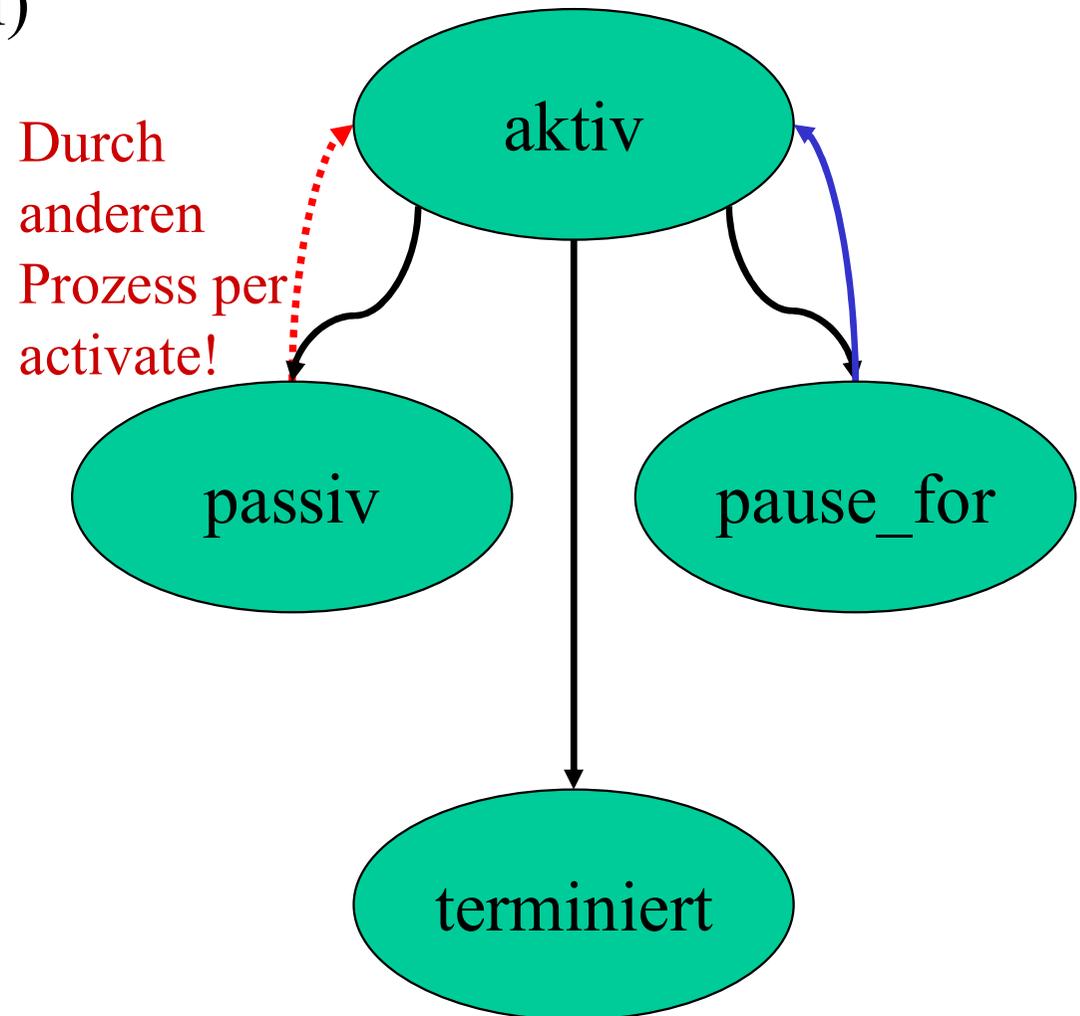
Zwei mögliche, „entgegengesetzte“ Richtungen

## Variante 1

Näher zur Implementierung  
(d.h. weg vom mentalen Modell)

- `passivate`-Anweisung, die den aufrufenden Prozess anhält  
`if not b then passivate;`
- Prozess kann nur außen aufgeweckt werden
- `active`-Anweisung, die einen schlafenden Prozess explizit aufweckt  
`activate(P);`
- Simulator-Code wird länger/komplexer, aber effizienter

## Prozesszustandsdiagramm



## Variante 2:

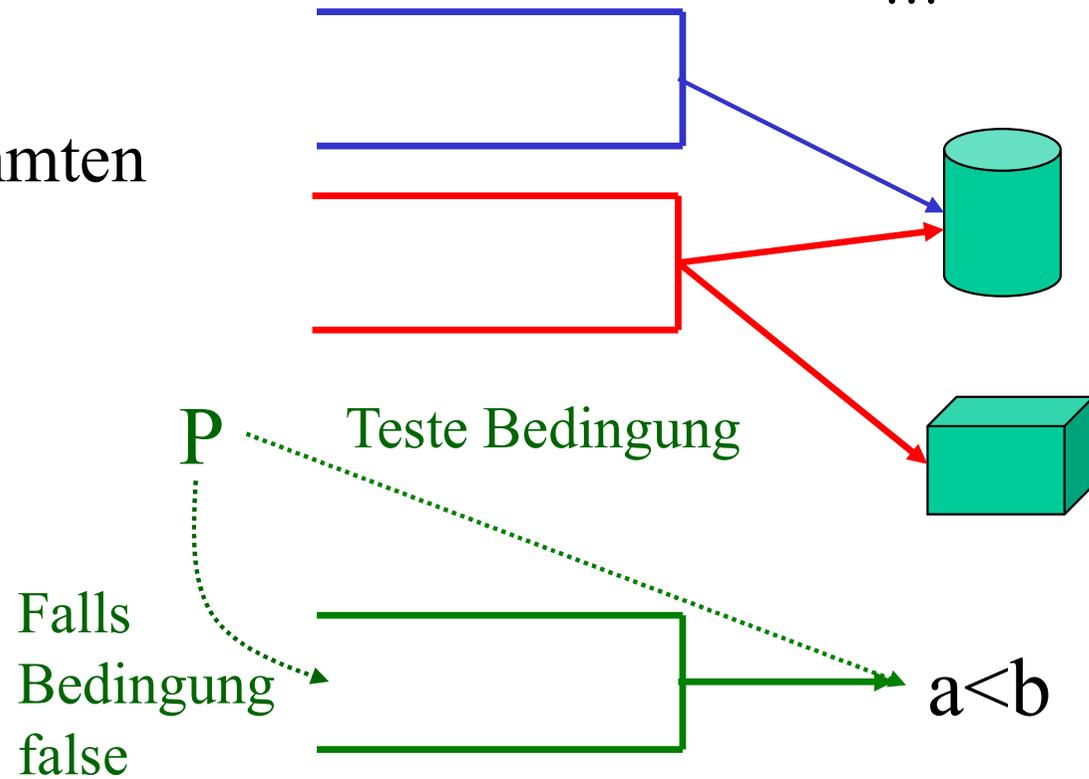
Näher zum mentalen Modell  
(d.h. weg von der  
Implementierung)

- Szenario-Sprachen  
d.h. Simulation in bestimmten  
Problemklassen
- Warten auf bestimmte  
Bedingungen  
z.B. warten auf freien  
Bediener
- Implementierung durch  
Bildung von lokalen  
Warteschlangen für  
Bedingungsmengen

## Implementierung

Warteschlangen für  
Prozesse

Ressourcen,  
Bedingungen,  
...



P' verändert a oder b  $\Rightarrow$   
Teste bei nächstem Prozesswechsel  
Prozesse in WS bzgl.  $a < b$

## Agentenorientierte Simulation

- Software-Agenten als autonome Computerprogramme  
Ursprünge in der Künstlichen Intelligenz  
Eigenschaften von Agenten
  - autonom
  - kognitiv – adaptiv
  - reaktiv – proaktiv
  - sozial – kommunikativ
- Definition des Agenten schwammig  
klare Abgrenzung zu sonstigen Programmen fehlt
- Einsatzgebiete
  - E-Commerce
  - Informationsrecherche
  - Computerspiele
  - **Simulation**

# Zentrale Ideen der agentenorientierten Simulation

- Software-Agenten beschreiben
  - Lokales Verhalten + Lernfähigkeit (if .. then–Regeln + Historie)
  - Kommunikationsbeziehungen zu anderen Agenten
- Keine explizite Beschreibung des Gesamtmodells (top down)
- Simulationsmodell entsteht aus Interaktionen der Agenten (bottom up)

Ist das mehr als prozessorientierte Simulation?

Sehr unterschiedliche Standpunkte in der Literatur:

- ✓ Discrete-event simulation is dead, long live agent-based simulation!  
Titel einer Podiumdiskussion
- ✓ Agent-based simulation is just a special case of discrete-event simulation!  
D. Pegden zitiert im Buch von A. Law

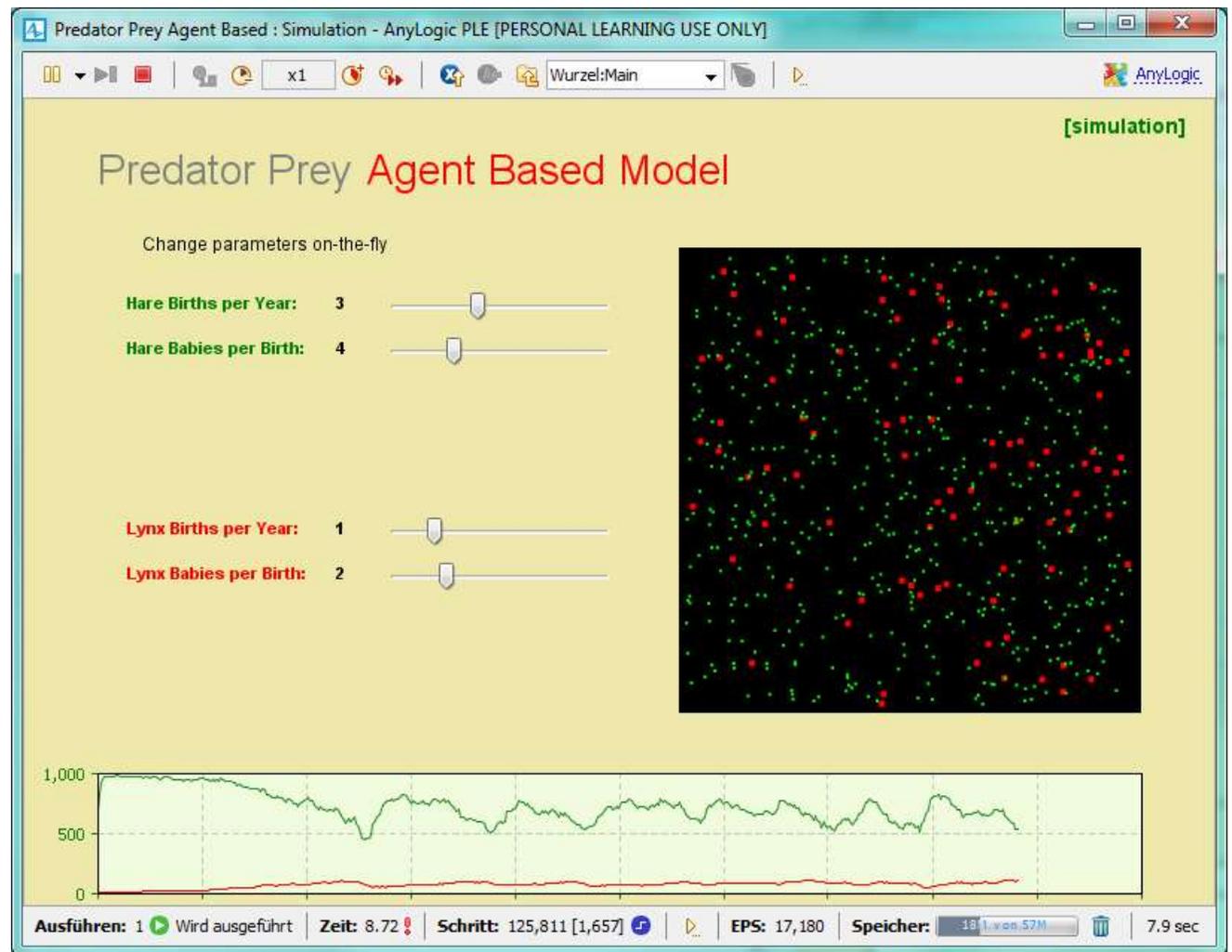
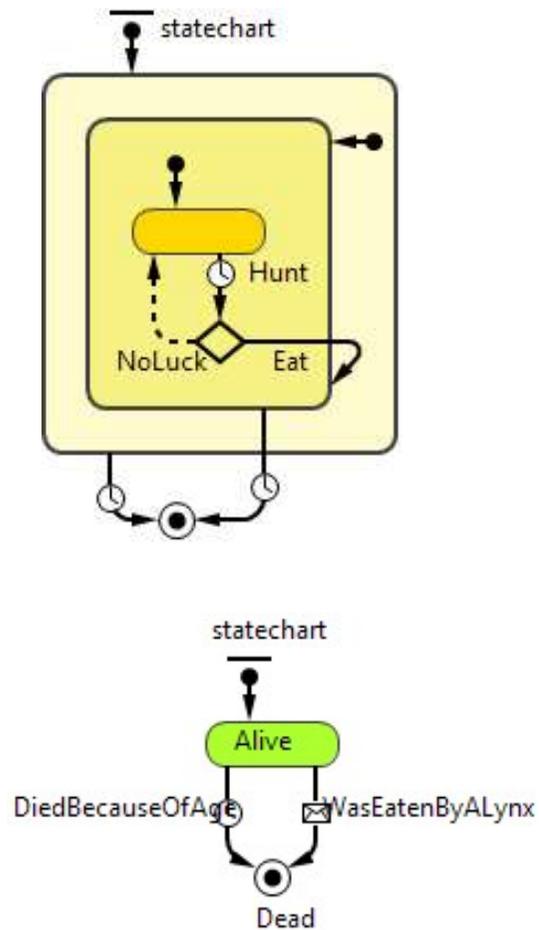
## Implementierung agentenorientierter Simulatoren:

- Objektorientierte Ansätze zur Beschreibung von Agenten
- Agenten haben einen Zustand
  - lokale Zeit
  - Position im Raum (bestimmt Umgebung + Kommunikation)
  - lokale Historie
- Simulationsfortschritt basiert auf
  - globaler getakteter Zeit (zeitdiskret) oder
  - globaler Ereignisliste (ereignisdiskret)
- Auswertung
  - für einzelne Agenten einfach realisierbar
  - für das gesamte System nur über globale Variablen

# Anwendungsgebiete agentenorientierter Simulation

- Systeme, deren Dynamik aus der Interaktion von Individuen entsteht
  - Biologische Systeme  
(z.B. Ausbreitung von Infektionen, Verhalten von Fischeschwärmen)
  - Ökologie  
(z.B. Räuber-Beute Modelle, Ausbreitung von Schadstoffen)
  - Sozialwissenschaften  
(z.B. Autofahrer im Stau, Evakuierungsszenarien)
  - Wirtschaftswissenschaften  
(z.B. Verhalten von Marktteilnehmern)
  - Ingenieurwissenschaften  
(z.B. Nutzer von Mobilfunknetzen)

# Räuber-Beute-Modell agentenorientiert



# Agentenorientierte Simulation

## Grenzen

- Synchronisation durch gemeinsame Ereignisliste oder synchrone Ereignisse
- Globale Beobachtung des Systems

Größe + Komplexität simulierbarer Modelle ist stark eingeschränkt

## Vision

- Autonome Agenten, parallel simulierbar
- Keine globale Zeit
- Verteilte Beobachtung des Zustand

## Probleme

- Konzepte zur effizienten Parallelisierung fehlen (Forschungsgebiet seit mehr als drei Jahrzehnten)
- Alle realisierten Konzepte basieren auf globaler Zeit und vollständiger Beobachtbarkeit der Simulation