

## 2. Modellbildung und Konzepte der ereignisorientierten Simulation

Ziel Modellbildung:

Erarbeitung "geordneter" Vorstellung (+ Darstellung), wie

- ein System **ist**
- ein System **sich verhält**

(sogar: "mathematisch konsistente" Darstellung dessen)

Man mache sich klar:

Sprechen wir über "ein System",  
so sprechen wir in Wahrheit immer nur über  
"unsere Vorstellung (Idee)  
von diesem System"

Ein System "ist", wie es eben ist !

Wir sprechen darüber,

wie unser **Gedankenmodell** des Systems ist

Gedankenmodell ist auf Betrachtungszweck bezogen,  
(conceptual model) kann hinreichend detailliert,  
hinreichend präzise, ... sein  
(muß es aber nicht)  
kann falsch sein !

zB "Tisch" ist blau / reflektiert Licht der Wellenlänge  $x$ ,  
hat Beine,  
wiegt  $y$  kg

Unbewußte Annahme:

- wenn wir (bei Modellbildung) "realistischer", präziser werden wollen, glauben wir i.allg., dies durch höhere Detaillierung erreichen zu können
- muß nicht zutreffen, kann undurchführbar sein

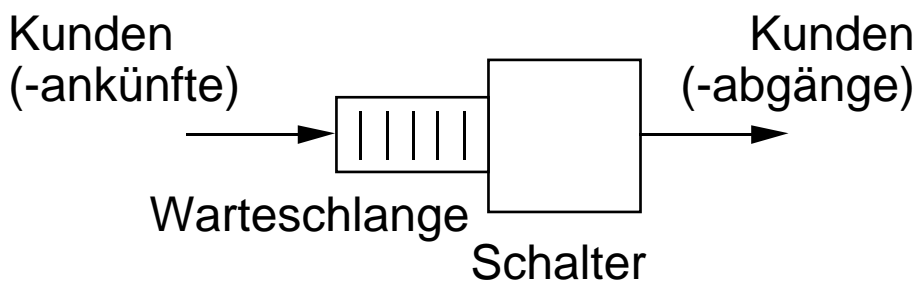


Auch unter dynamischen Systemen wird uns vorrangig eingeschränkte Klasse beschäftigen:

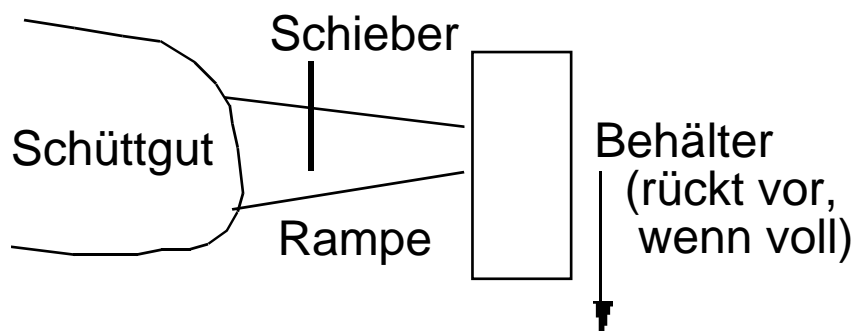
### ereignisorientierte (discrete event) Systeme

Zur Herausarbeitung Unterschied zwei Beispiele:

#### a) Bankschalter



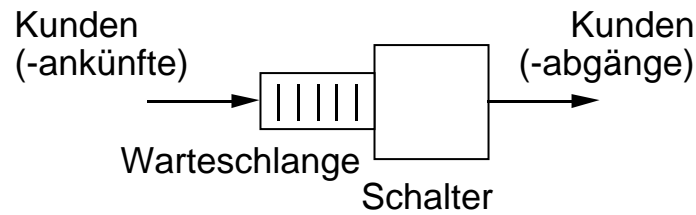
#### b) Abfüllvorrichtung



NB: hinsichtlich formaler Modell-Klassen damit die beiden "wesentlichsten" Familien berührt:

- (zeitbehaftete) Automaten
- (zeitbezogene) Differentialgleichungen

zu a)



- Objekte: Schalter,  
Warteschlange,  
(Menge von) Kunden
- Attribute: belegt / nicht belegt,  
Länge (Zahl wartender Kunden),  
(Menge von) Ankunftszeit, Bedienbedarf
- Attributänderungen:
 

	Schalter
Bei Ankunft	Ende Warteschlange
	nächster rückt vor ...
Bei Abgang	leer ...

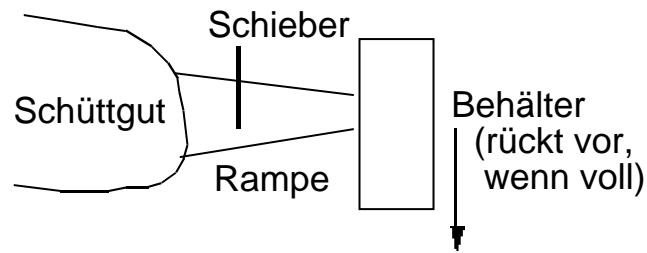
### Besonderheit:

Attributwerte ändern sich diskontinuierlich,  
zu **Ereignis**-Zeitpunkten

Ereigniszeitpunkte "vorhersehbar",  
zB Zeitpunkte:

$$\text{Abgang} = \text{Bedienanfang} + \text{Bedienbedarf}$$

zu b)



- Objekte: Behälter,  
Schieber
- Attribute: Menge abgefülltes Gut "a"  
Schieberstellung "b"
- Attributänderungen:

"Abfüllrate"

$$a(t) = \int_{t_A}^t f(b) dt \quad t \quad A < t < t_B$$

$$a(t_A) = 0$$

$$a(t_B) = \text{"voll"}$$

Definition für  $t_B$ 

### Fallunterscheidungen:

b1) Schieberstellung  $b$  konstant,  $f(b) = \text{const}$ :

$$t_B \text{ aus: } t_B = t_A + t_F$$

$$\text{mit } t_F \text{ aus: } f(b) \cdot t_F = K \quad (= \text{Behälterkapazität})$$

folglich: Ereigniszeitpunkt "voll" bekannt  
(vorhersehbar),  
und ereignisorientierte Sicht möglich  
(trotz "an sich" kontinuierlicher  
Zustandsänderungen)

- b2) Schieberstellung  $b$  nicht konstant,  
aber festliegende Funktion der Zeit

$$b = g(t - t_A)$$

$t_B$  aus:  $t_B = t_A + t_F$

mit  $t_F$  aus:

$$\begin{aligned} K &= \int_{t_A}^{t_B=t_A+t_F} f(b) dt \\ &= \int_{t_A}^{t_B} f(g(t-t_A)) dt \end{aligned}$$

erneut: Ereigniszeitpunkt "voll" vorhersehbar  
und ereignisorientierte Sicht möglich

- b3) Schieberstellung  $b$  nicht konstant,  
durch Füllmenge "geregelt" gemäß

$$b = h(a)$$

Differentialgleichung (-System)

$$\dot{a}(t) = f(h(a))$$

falls expliziter Lösung

(und damit expliziter Bestimmung von  $t_F$ ,  $t_B$ )

nicht zugänglich,

dann ereignisorientierte Sicht nicht mehr möglich

- ⇒ kontinuierliche Attributsänderungen zu beachten,  
Behandlung mit "kontinuierlicher" Simulation  
(Abtasten der Zeitfunktionen "in kleinen Schritten")

**Wir behandeln:**

**Dynamische Systeme / Modelle,**

deren Objekt-Attribute

- sich (über der Zeit) **in diskontinuierlichen Sprüngen** ändern, zumindest diese Sicht zulassen (Vorstellungswelt Gedankenmodell)
- sich **zu "Ereignis"-Zeitpunkten** ändern

deren Objekt-Menge sich (wenn überhaupt)

- zu "Ereignis"-Zeitpunkten ändert

damit insgesamt: **Ereignisorientierte Systeme / Modelle**

"Menge von Objekten + deren Attributwerte"

(zu festem Zeitpunkt t)

gibt an, wie System zum Zeitpunkt t "ist", seinen **Zustand**

"Änderungen der Objektmenge + der Werte von Attributen"

(zu Zeitpunkten von Ereignissen)

erfaßt **Zustandsänderungen**

**Simulation** solchen Systems heißt:

- von (gesetztem) Anfangszustand aus
- Ereignis nach Ereignis (in zeitlicher Reihenfolge), insbesondere dabei auftretende Zustandsänderungen "Schritt für Schritt" nachzuvollziehen,
- den somit "imitierten" Verlauf von Zustandsverläufen (Zustandstrajektorien) "geeignet" zu beobachten / zu notieren / zu protokollieren,
- den beobachteten Ablauf zu bewerten / zu beurteilen





Soweit "Zustand"  
 Nun "Zustandsänderungen":

Im Simulator zwangsweise  
 Wertveränderungen von Zustandsvariablen,  
 dh per (Menge von) Wertzuweisung(en),  
 dh per (Exekution von) Simulator-Code

Wann? "Zu Ereigniszeitpunkten" (Modellvorstellung)

Für den Simulator also:

- Identifikation aller Ereignis-"Typen",  
 wo jeder Ereignistyp durch eine spezifische Menge  
 von Zustands-(Variablen-)Änderungen ausgezeichnet ist
- Programmierung eines "Stückes Code" (**Ereignisroutine**)  
 für jeden Ereignistyp, das bei Exekution  
 die (für diesen Ereignistyp) spezifischen  
 Zustands-Variablen-Werteveränderungen durchführt  
 (Imitation Attributänderungen im Objektsystem, "zeitlos")
- Organisation der (Modell-)zeitgerechten Exekution von  
 Ereignisroutinen  
 (Imitation Eintreten von Ereignissen im Objektsystem)  
**Diskussion aufgeschoben**

Insgesamt bis jetzt:

- Abbildung Zustand (Zustands-)Datenstruktur,  
**statische Struktur**
- Abbildung Zustandsänderungen Ereignisroutinen,  
**temporale Struktur**

**Beispiel 2.1:** "Bankschalter",  
in simulativem Modell zu fassen

Mentales Modell ist "ohne viel Nachdenken" greifbar:

- hinter Bankschalter Angestellter,  
kann diverse Tätigkeiten verrichten

Unmittelbares Bedürfnis nach Präzisierung:

- Genau ein Angestellter,  
- verrichtet eine Tätigkeit nach der anderen  
jeweils vollständig,  
- geht nicht weg, solange er etwas zu tun hat,  
- legt auch keine sonstigen Pausen ein
- vor dem Bankschalter "Publikumsverkehr"

Präzisierung:

- Kunden treffen einzeln am Schalter ein,  
- stellen sich "hinten" an, wenn sie warten müssen,  
- rücken "diszipliniert" vor,  
- haben bestimmten "Auftrag" im Sinn,  
- gehen erst weg, wenn ihr Auftrag erfüllt ist

NB: Es handelt sich um nichts anderes als um eine  
"freundliche" Verkleidung des  
(Ihnen vielleicht wohlbekannten)  
"FCFS-single-servers"

## Statische Struktur? Beispielsweise wie folgt

### Objekte:

(mehrere  
Exemplare von:)

•Kunde

(genau ein Schalter,  
strukturiert in:)

•Bedienplatz

•Warteschlange

### Attribute:

(je Kunden-Exemplar:)

- Identifikation:

ID, Wertemenge NAMEN,  
beliebig codiert

- Auftrag:

TASK, Wertemenge  
AUFTRAGSARTEN,  
beliebig codiert

- Warteposition:

P, Wertemenge  
"dran" POSITIONEN,  
Codierungsvorschlag:  
P {0} {1,2,...}

- Tätigkeit:

B, Wertemenge  
"untätig" AUFTRAGSARTEN  
(s. oben)

- Bedienter:

K, Wertemenge  
"keiner" NAMEN (s. oben)

- Füllungsgrad:

N {0,1,2,...}

- Warteliste:

WS, Liste der Länge N aus  
(zB) Kunden ID's (s. oben),  
in Ankunftsreihenfolge sortiert

Bemerke:

Vieles "doppelt gemoppelt",  
Zustandsraum sehr redundant  
und von einem "minimalen" weit entfernt

## **Temporale Struktur?**

Zustandswechsel erfolgen genau anlässlich

- des Eintreffens eines Kunden (Ankunft),
- des Endes der Bedienung eines Kunden (Bedienende)

Also genau zwei Ereignistypen existent  
und zugehörige Ereignisroutinen gefragt

Wir müssen

**auf der Basis der gewählten statischen Struktur**

notieren:

**Ereignistypen:****Ereignisroutinen:**

- Ankunft:  
(eines bestimmten Kunden)

```
{ Notiere Attribute,
  dh ID- und TASK-Werte,
  dieses Kunden};
IF B="untätig"
THEN BEGIN
    P{-Attribut dieses Kunden}:=0
                                     {für "dran"};
    B:=TASK {dieses Kunden};
    K:=ID {dieses Kunden}
    END
ELSE BEGIN
    N:=N+1;
    P{-Attribut dieses Kunden}:=N;
    {Verlängere Liste WS
     hinten um einen Eintrag,
     ID dieses Kunden}
    END
```

- Bedienende:  
(eines bestimmten Kunden)

```
{ Lösche alles von jenem Kunden,
  dessen ID=K};
IF N=0 {keine Wartenden}
THEN BEGIN
    B:= "untätig";
    K:= "keiner"
    END
ELSE BEGIN
    B:=TASK {des vordersten
             in WS};
    K:=ID {des vordersten in WS};
    {Lösche vordersten
     WS-Eintrag};
    N:=N-1
    {noch was?}
    END
```

Sie bemerken:

- doch komplexer als erwartet
- Grund: unsere "selbstverschuldete"  
Wahl der statischen Struktur

Lerneffekt?

Zwar: Unser Verständnis von einem  
zu analysierenden / zu modellierenden System,  
unser mentales Modell,  
befähigt uns unmittelbar,  
statische und temporale Struktur eines zugehörigen  
ereignisorientierten Simulators  
zu notieren

Aber: Unbedachtes Vorgehen dabei hat seinen Preis:

- uU redundanter Zustandsraum,
- der komplexe (und irrtumsanfällige!)  
Ereignisroutinen nach sich zieht.

Dies ganz prinzipiell:

Redundanz heißt

- wechselseitige (nicht zwingend erforderliche)  
Abhängigkeit von Zustandsgrößen
- bestimmte Relationen über Zustandsgrößen  
müssen aufrechterhalten werden
- Veränderung einer Zustandsgröße zwingt  
zur Veränderung von anderen

Einer der diesbezüglichen Fälle aus dem Beispiel:  
Die ID des Kunden, dessen P "dran" signalisiert, muß in K  
des Bedienplatzes notiert sein.

Wir haben festgelegt

- statische Struktur (Zustandsraum)
  - temporale Struktur (Zustandsübergangsregeln)
- eines ereignisorientierten Simulators

Aufgeschoben war

- Stattfinden /Eintreten von Ereignissen zu Zeitpunkten  
(im ablaufenden Programm "Simulator")

Bei "Erfindung" programmtechnischen Mechanismus'  
ist "Ereignisorientiertheit" des Modells zentrale Stütze:

- Zustandsänderungen finden statt zu Ereigniszeitpunkten,  
zwischen aufeinanderfolgenden Ereignissen  
bleibt Zustand unverändert  
(programmtechnisch: nichts "am Zustand" zu tun)
- zwischen aufeinanderfolgenden Ereignissen  
verstreicht "Zeit"  
(programmtechnisch: nachzuvollziehen)
- da Zeitintervalle zwischen Ereignissen geschehnislos,

kann "**Zeitablauf**" modelliert werden als  
**diskontinuierlicher Vorgang**, der unmittelbar  
von einem E.-Zeitpunkt auf den folgenden "springt"  
(programmtechnisch: Zuweisungen an globale Variable  
t := "momentane Zeit"  
der Reihe nach, zu E.-Zeitpunkten)

Verschiedenene Implementierungen dieses Prinzips möglich

Einfachste: Realisierung eines "Kalenders",  
einer Liste aus Einträgen darüber,  
**wann welches Ereignis** eintritt,  
einer sog. **Ereignisliste**

(Ereignis-)

Zeitpunkte	$t_1$	$t_2$	$t_3$	...	...
Typen	$tp_1$	$tp_2$	$tp_3$	...	

**Abbildung 2.2:** Ereignisliste;  
 $t_i, i=1,2,\dots$ , monoton nichtfallend

Zusätzlich nötig: "**zentrale Simulatorschleife**"

welche eine globale Variable Zeitvariable "t"

- der Reihe nach auf die Werte  $t_i$  setzt
- und anschließend in die  
gemäß  $tp_i$  "zugehörige" Ereignisroutine verzweigt

...

WHILE {Simulator läuft}

DO BEGIN

{lies "vordersten" (Abb. 2.2: "ganz linken")

Eintrag der Ereignisliste, Werte seien:  $t_x, tp_x$ };

$t:=t_x$ ; {"momentane Zeit" t "springt auf"  $t_x$ }

{verzweige zu Ereignisroutine,

die durch  $tp_x$  gekennzeichnet ist,

Annahme: Ereignisroutine gibt nach Ablauf

Kontrolle **hierher** zurück};

{lösche "vordersten" Eintrag der Ereignisliste}

END;

...

**Abbildung 2.3:** Zentrale Simulatorschleife



Insgesamt:

- "Zeitablauf" simuliert durch diskontinuierliches "Vorrücken" der Variablen  $t$
- Zustandsveränderungen ("Trajektorie") simuliert durch Exekutieren (zu allen Ereigniszeitpunkten) der jeweils zugehörige Ereignisroutine

Quelle von häufigen Mißverständnissen ist  
**Reihe verschiedener Zeitbegriffe**

**Objektzeit**, jene Zeit, in der (zumindest hypothetisch) der Betrieb eines realen Systems abläuft

**Modell-** oder **Simulationszeit**, jene Zeit, die im Programm "Simulator" manipuliert (indem Zeitvariable  $t$  gesetzt);  
 Modellzeit imitiert Objektzeit, ist also "identisch" zu ihr (bis auf Translationen, etwa: "Start bei 0")

**Realzeit** des exekutierenden Simulatorprogramms (das ja in realem Zeitintervall abläuft), nur der Vollständigkeit halber, hier belanglos

**Ausführungszeit** (des exekutierenden Simulatorprogramms) jene Zeit, welche (exekutierende) CPU benötigt; diese **CPU-Zeit** ist interessant, da Ressourcenbedarf zu bedenken: Simulatoren sind notorische "Langläufer", ihre "Effizienz" (wieviel Modellzeit in wieviel CPU-Zeit) ist regelmäßiges Problem

Zwischen den beiden als interessant verbliebenen

## **Modellzeit , CPU-Zeit**

besteht (auf den ersten Blick) verwirrender Zusammenhang:

Modellzeit verstreicht "zwischen" Ereigniszeitpunkten;  
Im Simulator von E.-zeitpunkt zu E.-zeitpunkt "gesprungen",  
keine Code-Ausführung, kein CPU-Zeitverbrauch

Zustandsübergänge geschehen anlässlich  
Eintreten von Ereignissen "spontan", in "0"-Modellzeit;  
Vollzug der Ereignisübergänge im Simulator per  
Programmcode-Durchlaufen (Ereignisroutinen),  
Verbrauch CPU-Zeit

Kurz: Endliche Modellzeitintervalle (zwischen Ereignissen)  
werden in verschwindender CPU-Zeit nachvollzogen,  
verschwindende Modellzeitintervalle  
(zu Ereigniszeitpunkten) in endlicher CPU-Zeit

Aus dieser Klärung unmittelbar praktisch Verwertbares:

Exekutionszeit Simulator (CPU-Zeit)

- von Anzahl simulierter Ereignisse bestimmt
- nicht von überstrichener Modellzeit  
(= nachzubildender Objektzeit)

Erhöhung Simulationseffizienz

(zB:  $\text{Modellzeit} / \text{CPU-Zeit}$ )

muß, wo erforderlich, an Ereigniszahlen ansetzen  
damit am Abstraktionsniveau unseres mentalen Modells

Im allgemeinen gilt:

höheres Abstraktionsniveau      weniger Ereignisse

Wir wissen:

Technischer Ablauf einer ereignisorientierten Simulation organisierbar auf Basis "gefüllter" Ereignisliste

Wie kommen Ereigniseinträge "dorthin"?

**Vor Ablauf** der eigentlichen Simulation eintragen?

In sich widersprüchlich:

Wenn alle Ereignisse (Zeitpunkt + Typ) bekannt, dann Simulieren nicht erforderlich, Ereignislisten und Trajektorien aufeinander abbildbare Darstellungen des System-"Verhaltens"; eventuell "Bewerten", aber nicht mehr "Simulieren"

Bleibt: Ereignisliste **während** Simulation füllen

Wie das?

- Wenn Betrieb realen Systems schrittweise simuliert, wird (über mentales Modell) immer wieder "ein Stückchen Zukunft" sichtbar

zB Bankschalter Bsp. 2.1

Sicherlich zu spezifizieren:

Wann erfolgen Kundenankünfte?

- Angenommen, deterministisch (genau) alle 5 min wir wissen anlässlich Ankunft (Ereignis), daß 5 min später wieder Ankunft (Ereignis)
- Angenommen, stochastisch mit u.i.v. Abstands-ZV  $A_i$  ("u.i.v.": **unabhängig identisch verteilt**, gemäß **Zufalls-Variable A**) wir können anlässlich Ankunft Realisierung  $a$  von  $A$  "beschaffen",  $t+a$  ist wieder Ankunft

Ähnlich: Schließen von Bedienbeginn auf Bedienende

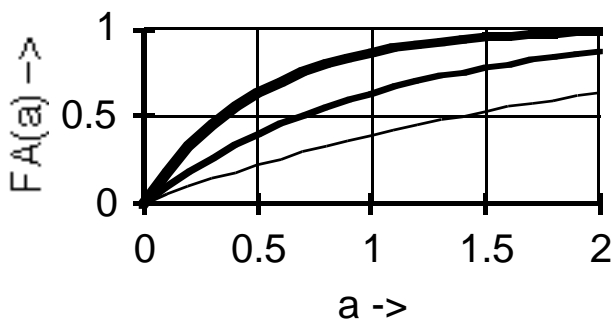
- Wenn "Stückchen Zukunft" sichtbar, dann "vormerken";  
technisch, indem  
Eintrag (Ereigniszeitpunkt, Ereignistyp) Ereignisliste  
**Einsortieren**
- Wo (im Simulatorcode) steht "Vormerk"-Operation?  
Einzige Möglichkeit: in den Ereignisroutinen!
  - werden "anlässlich" eines Ereignisses durchlaufen,
  - bei ihrer Codierung Kenntnisse vorhanden  
über neu aufscheinendes "Stückchen Zukunft"
 E.-Routinen erhalten damit **zweite** wesentliche **Aufgabe**:  
(neben der Veränderung Modellzustand)  
**Planung der Modellzukunft** vornehmen

**Beispiel 2.4:** Bankschalter aus Bsp. 2.1,  
in abstrahierter Form (zusätzliche Annahmen)  
"M/M/1-FCFS-System":

u.i.v., exponentielle Ankunftsabstände (erstes "M"),  
gemäß Zufallsvariable A, Parameter  $\lambda$ ,  
u.i.v., exponentielle Bedienzeiten (zweites "M"),  
gemäß Zufallsvariable S, Parameter  $\mu$   
First-Come-First-Served Bedienstrategie: "FCFS"

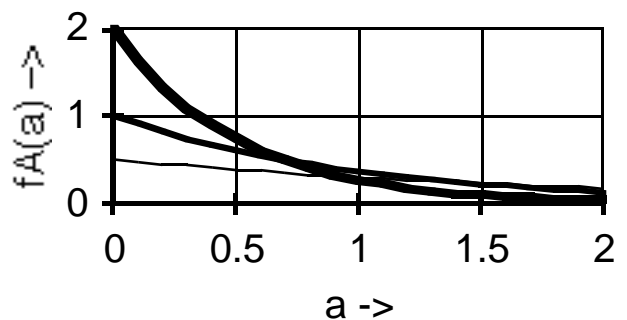
**Skizze 2.5:** Exponentialverteilung

Verteilungsfunktion



—  $\lambda=0.5$  —  $\lambda=1$  —  $\lambda=2$

Verteilungsdichte



—  $\lambda=0.5$  —  $\lambda=1$  —  $\lambda=2$

**Statische Struktur:**

Diesmal Wahl minimalen Zustandsraums

"Zahl Anwesender"

(unter Nutzung analytischer Kenntnisse: hinreichend)

**Objekte:**

- Wartesystem

**Attribute:**

- Füllungsgrad:

$n \in \mathbb{N}_0$

(zählt Anwesende incl. Bedientem)

**Temporale Struktur:****Ereignistypen:**

- Ankunft:

**Ereignisroutinen:**

$n := n + 1;$

{ sortiere Ereignis (Ankunft,  $t+a$ )

in Ereignisliste ein;

$t$  ist gegenwärtiger Zeitpunkt,

$a$  ist Realisierung der

exponentiell, mit Parameter  $\lambda$ ,

verteilten Zwischenankunfts-ZV  $A$ };

IF  $n=1$  {"vorher" war Wartesystem leer}

THEN BEGIN

{ sortiere Ereignis (Bedienende,  $t+s$ )

in Ereignisliste ein;  $t$  ist "jetzt",

$s$  ist Realisierung der

exponentiell, mit Parameter  $\mu$ ,

verteilten Bedienzeit-ZV  $S$ }

END

- Bedienende:

$n := n - 1;$

IF  $n > 0$  {System ist nicht leer}

THEN

{sortiere Ereignis (Bedienende,  $t+s$ )

in Ereignisliste ein}

**Bemerke:** Dank wohlüberlegten Zustandsraums (keinerlei Redundanz) sind Manipulationen des Zustands einfach; Zukunftsplanung nimmt vergleichsweise großen Raum ein.

Vereinbarung zur **Schreiberleichterung:**

- Einsortieren Ereignis des Typs "type" zum Zeitpunkt "time" in die Ereignisliste mittels Anweisung

PLAN(type,time)

- "Ziehen" einer ZV-Realisierung "wert" einer ZV mit Verteilung "verteilungskennung" und etwa nötigen Parametern "parameterliste" mittels

wert:= DRAW(verteilungskennung(parameterliste))

Ereignisroutine "Bedienende" des Bsp. 2.4 damit:

n:=n-1;

IF n>0

THEN PLAN(Bedienende, t+DRAW(negexp( $\mu$ )))

Prinzipien ereignisorientierte Simulation damit fast fertig.  
Zwei wesentliche "Kleinigkeiten" noch:

Wie fängt Simulation an?  
Und wie hört sie auf?

Wir erfinden schnell ein Stück Code

### **Simulationshauptroutine**

die bei "Start" begonnen wird zu exekutieren; im Beispiel

BEGIN

```
t:=0 {Modellzeit startet bei 0};
PLAN(Ankunft,DRAW(negexp(l)))
      {eine erste Ankunft wird festgelegt};
      {sorge für Simulationsabbruch, vgl. unten}
      {zentrale Simulatorschleife der Abb. 2.3};
      {nochwas?}
```

END

"sorge für Simulationsabbruch" in Zusammenhang mit

WHILE {Simulator läuft} (zentrale Simulatorschleife)

zu codieren - dem Sinn nach zB "bis Modellzeit > Grenze"  
- es gibt andere Möglichkeiten dafür

Letzte Bemerkungen:

Bis jetzt läuft Simulator "spurlos" ab, da  
keinerlei Beobachtungen aufgezeichnet

Aber "Meßstellen" für **Aufzeichnungen** (völlig natürlich)  
in den Ereignisroutinen vorhanden

(**dritte Aufgabe** für Ereignisroutinen)

Zusammenfassung, Bewertung der Aufzeichnungen,  
falls unmittelbar benötigt,  
im "nochwas" der Simulationshauptroutine