VORLESUNG

Automatisierung industrieller Workflows

Teil C: Die Sprache SLX

- GPSS- Einführung -

Joachim Fischer

Inhalt C.3

- **Teil A**
  Aspekte
  dynamis
- **Teil B**
  Die Mod
- **Teil C**
  Die aust
  SLX
- **Teil D**
  Modellie

- **C.1**
  Einführung und Ba
- **C.2**
  Stochastische Pro
- **C.3**
  GPSS-Elemente
- **C.4**
  DISCO-Elemente
- **C.5**
  Basissprache (Erg

1. Einführung
2. Zyklische Erzeugung aktiver Objekte (generate)
3. Bedienungseinrichtung (facility) und Warteschlangenstatistik (queue)
   - Normale Bedienung
   - Verfügbarkeitsänderung
   - Facility: Control-Variablen, Makros
   - Queue: Operationen, Makros
   - gate-Anweisung für Facility
   - Verdrängende Bedienung
   - Standard-Report (Facility, Queue)
   - Zusammenfassung
4. ...

Barber-Shop: Facility, Queue

```
//*********************************************************************
// EX-00020: SLX Barbershop Model
//*********************************************************************
import <h7>
module barb13 {
//*********************************************************************
// Global Declarations
//*********************************************************************
      facility joe;
      queue joeq;
      rn_stream Arrivals seed=100000;
      rn_stream Service seed= 200000;
      constant float stop_time= 48000;
//*********************************************************************
// Customer
//*********************************************************************
 class customer {
      actions {
                  enqueue joeq;
            seize joe;
                  depart joeq;

                  advance rv_uniform(Service, 12.0, 18.0);
            release joe;
            terminate;
      }
}
//*********************************************************************
// Run Control
//*********************************************************************
procedure main {
      arrivals: customer
            iat = rv_uniform(Arrivals, 12.0, 24.0)
            until_time = stop_time;
      wait until (time >= stop_time and Q(joeq) == 0 and FNU(joe));
      report(system);
      exit(0);
}
} // End of barb13 module
```

Facility (Bedienungseinrichtung):

- bedient zu einem Zeitpunkt max. einen "Kunden" (besser: dessen Puck)
- spätere Kunden müssen warten (Pucks werden in einer von mehreren Puck-Listen der Facility (FIFO) verwaltet
- Statistik zur Warteschlange muss explizit organisiert werden
- Zur Erzeugung eines Stroms von Objekten aktiver Klassen (z.B. Kunden) steht die flexible **arrivals-** Anweisung zur Verfügung

Control-Attribute von Facility und Queue (spezielle Makros)

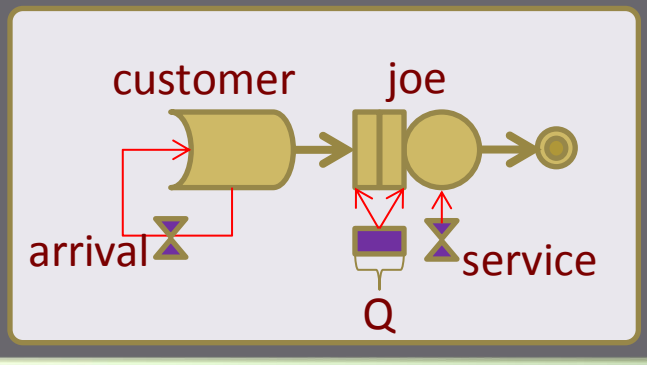
jeweils eingesetzte Objekte,wie

- **rn_stream**
- **queue**
- **facility**

registrieren sich (über ihre **initial**-Properties) in zentralen Listen zur Organisation der Standardausgabe

Systemreport wird erweitert

Standardausgabe



report(system);

Barber-Shop-31-01-2013.slx: SLX-64 UL211 Lines: 7,458 Errors: 0 Warnings: 0 Lines/Second: 185,145 Memory: 4 MB
Execution begins

System Status at Time 480007.1404

Random Stream	Sample Count	Initial Position	Current Position	Antithetic Variates	Chi-Square Uniformity
rna	26629	100000	126629	OFF	0.95
rns	26628	200000	226628	OFF	0.73

Facility	Total %Util	Avail %Util	Unavl %Util	Entries	Average Time/Puck	Current Status	Percent Avail	Seizing Puck	Preempting Puck
joe	83.17			26628	14.993	AVAIL	100.000	<NULL>	<NULL>

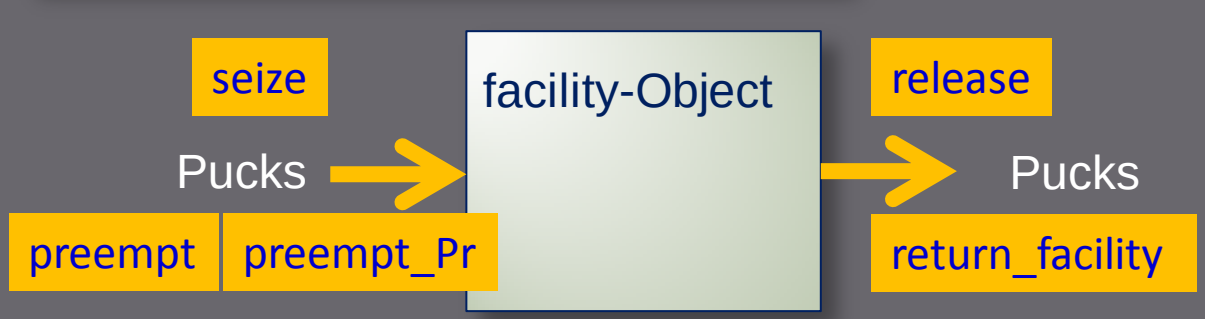
Queue	Current Contents	Maximum Contents	Average Contents	Total Entries	Zero Entries	Percent Zeros	Average Time/Item	Average > 0 Time/Item
joeq	0	2	0.05	26628	18220	68.42	0.853	2.701

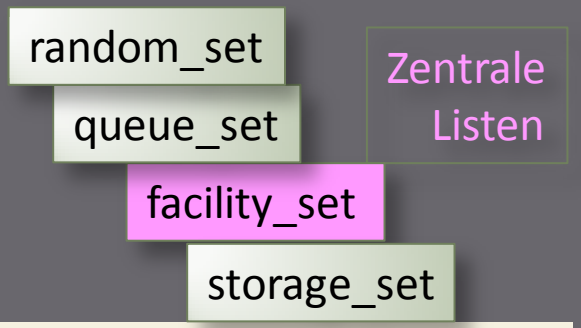
Execution time: 0.096 s
Execution complete
Objects created: 46 passive and 26,629 active Pucks created: 26,631 Memory: 4 MB Time: 0.10 seconds

Funktionsüberblick: Facility

Objekterzeugung:
facility *object_name* [**title**= *report_title*], …

sämtliche facility-Objekte
werden per **initial** in facility_set einsortiert





- Zu einem Zeitpunkt kann höchstens ein Prozess (Puck) die *Facility* in Benutzung haben, andere werden blockiert und müssen warten (**FIFO**).
- Der aktuelle "Besitzer/Nutzer"-Prozess kann von anderen Prozessen unterbrochen und verdrängt werden, er wird die Restzeit später nachholen können (**LIFO**).
- Prozesse können in der Fortsetzung ihrer Aktionen vom **Belegungszustand** einer *Facility* per **gate**-Anweisung abhängig gemacht werden, ohne dass sie selbst die Facility nutzen
- Darüber hinaus können in einer Anwendung Prozesse von (allen) **control**-Attributen einer Facilty direkt oder bei Anwendung vordefinierter Makros indirekt per **wait until** abhängig gemacht werden. (Das Laufzeitssystem setzt selbst kein **wait until** ein !)
- Alle Puck-Warteschlangen einer Facility (egal ob einfache Puck-Liste oder Set) bieten **keine Statistik (explizite Nutzung von Queue ist erforderlich**)

Layout von Facility

```
passive class facility(string(*) report_title) {
    read_only {
        string(12)    title;
        random_variable(time)   usage;
        interval      total_time,
                      avail_time,
                      unavail_time;
        ...
    }
    initial { ...}
    report{ ...}
    clear { ...}
} // class Facility
```

später

```
pointer (puck)              owner;
control enum facility_state state;
control pointer (puck)      preemptor;
control enum ...            availability;
```

```
set (puck)      preemption stack;
pointer (puck)  change_in seizability;
pointer (puck)  change_in_ownership;
pointer (puck)  change_in_availability;
pointer (puck)  change_in_preemption;
```

```
procedure SEIZE(inout facility f) ...
procedure RELEASE(inout facility f) ...
procedure PREEMPT(inout facility f) ...
procedure PREEMPT_PR(inout facility f) ...
procedure FAVAIL(inout facility f) ...
procedure FUNAVAIL(inout facility f) ...
```

auch als SLX-Statement, definiert per Spracherweiterung in h7

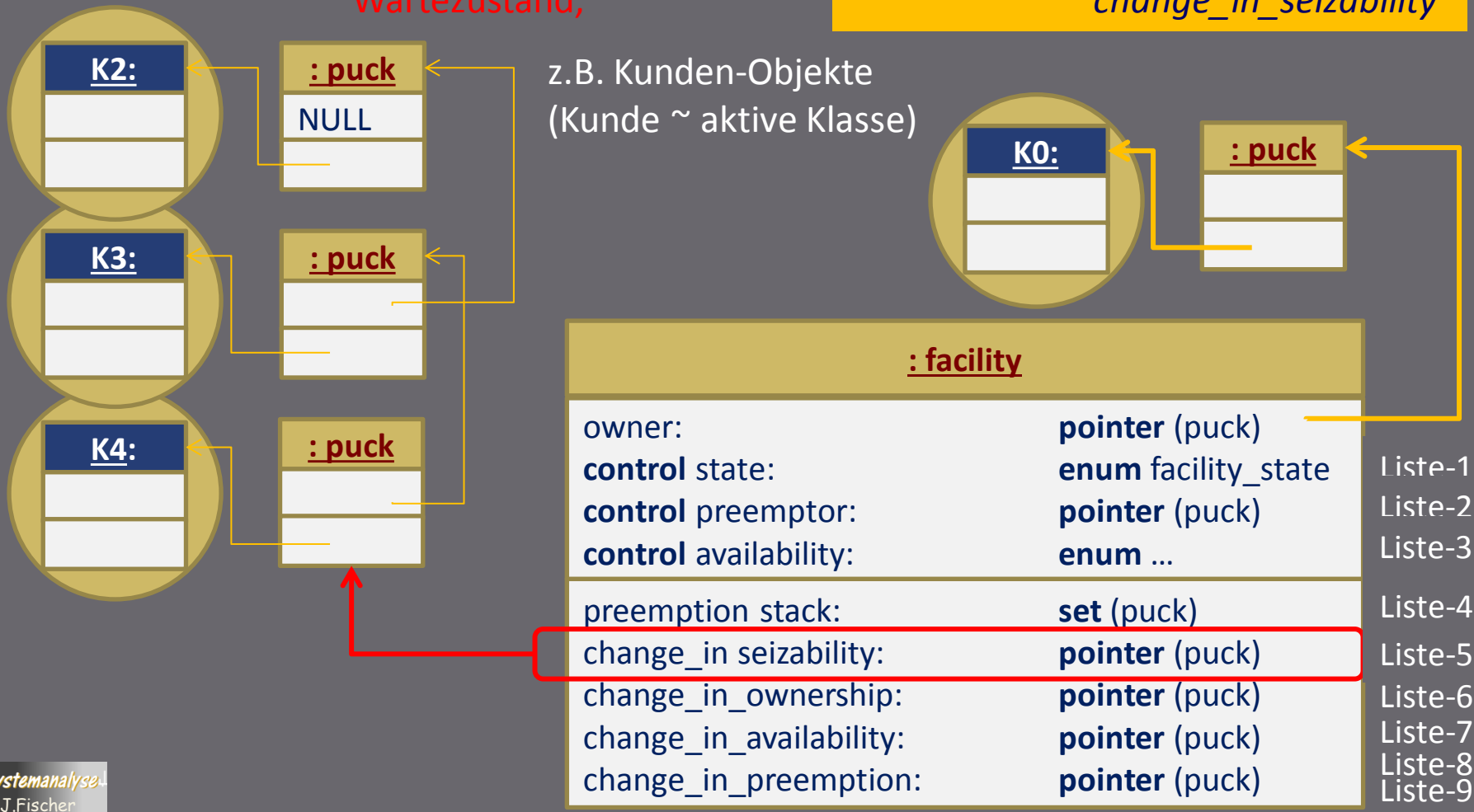
Facility ~ interne Sicht

Vielzahl von Puck-Listen, zur Puck-Synchronisation in speziellen Situationen

Pucks im Wartezustand,

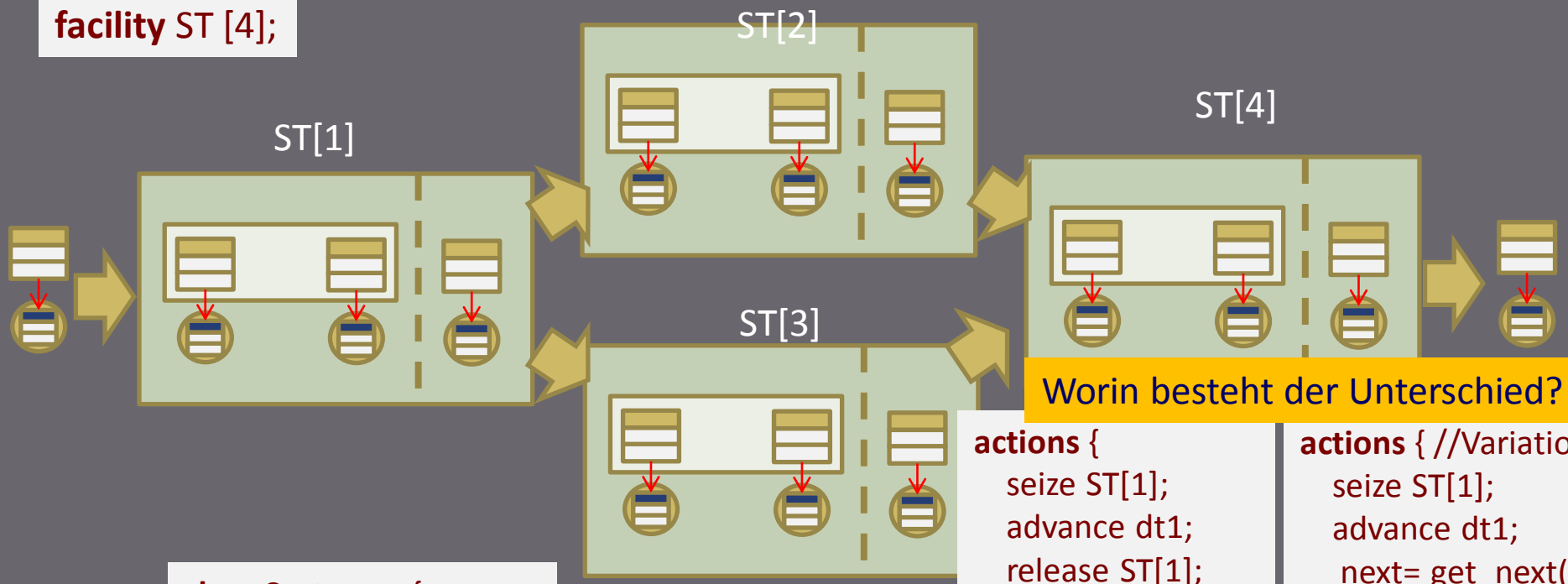
betrachten zunächst nur *change_in_seizability*

z.B. Kunden-Objekte
(Kunde ~ aktive Klasse)



Beispiel: Bedienungskette (in Varianten)

facility ST [4];



Worin besteht der Unterschied?

```
class Customer {
   double dt1, dt23, dt4;
   int next;

  procedure get_next():
         returning int {
   ...
       return ...;
  }
```

```
actions {
    seize ST[1];
    advance dt1;
    release ST[1];
    next= get_next();
    size ST[next];
     advance dt23;
     release ST[next];
     seize ST[4];
     advance dt4;
     release ST[4];
     terminate;
}
```

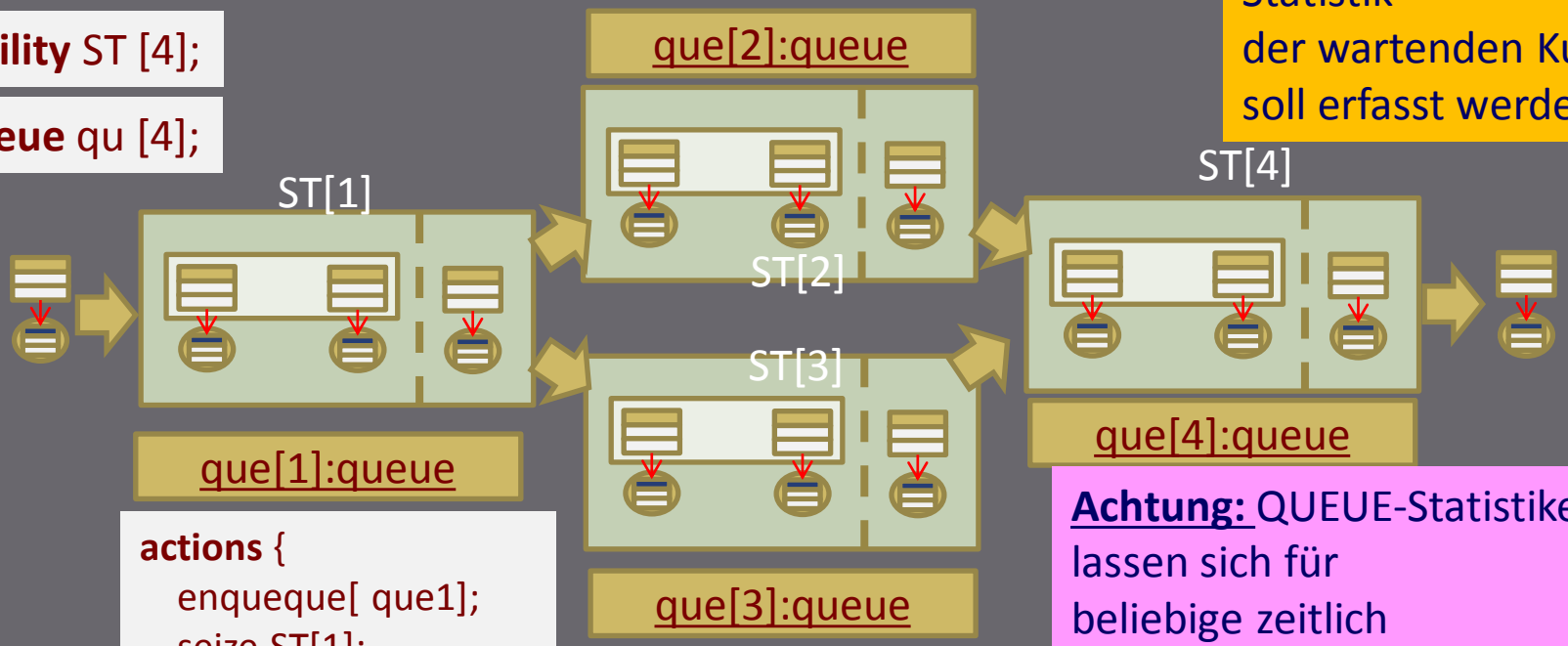
```
actions { //Variation
    seize ST[1];
    advance dt1;
     next= get_next();
    size ST[next];
    release ST[1];
     advance dt23;
     seize ST[4];
     release ST[next];
     advance dt4;
     release ST[4];
     terminate;
}
```

Beispiel: Bedienungskette (3. Variante)

facility ST [4];

queue qu [4];

Statistik
der wartenden Kunden
soll erfasst werden




```
actions {
   enqueque[ que1];
   seize ST[1];
   depart que[1];
   advance dt1;
   release ST[1];
   next= get_next();
   enqueque que[next];
   size ST[next];
   depart que[next]
    advance dt23;
    release ST[next];
      ...
```

```
   ...
    enqueue que[4];
    seize ST[4];
    depart que[4];
    release ST[next];
    advance dt4;
    release ST[4];
    terminate;
}
```

Achtung: QUEUE-Statistiken lassen sich für beliebige zeitlich begrenzte Aktionen der Aufrufer führen.

enqueue protokoll1;
wait until (...);
depart protokoll1;

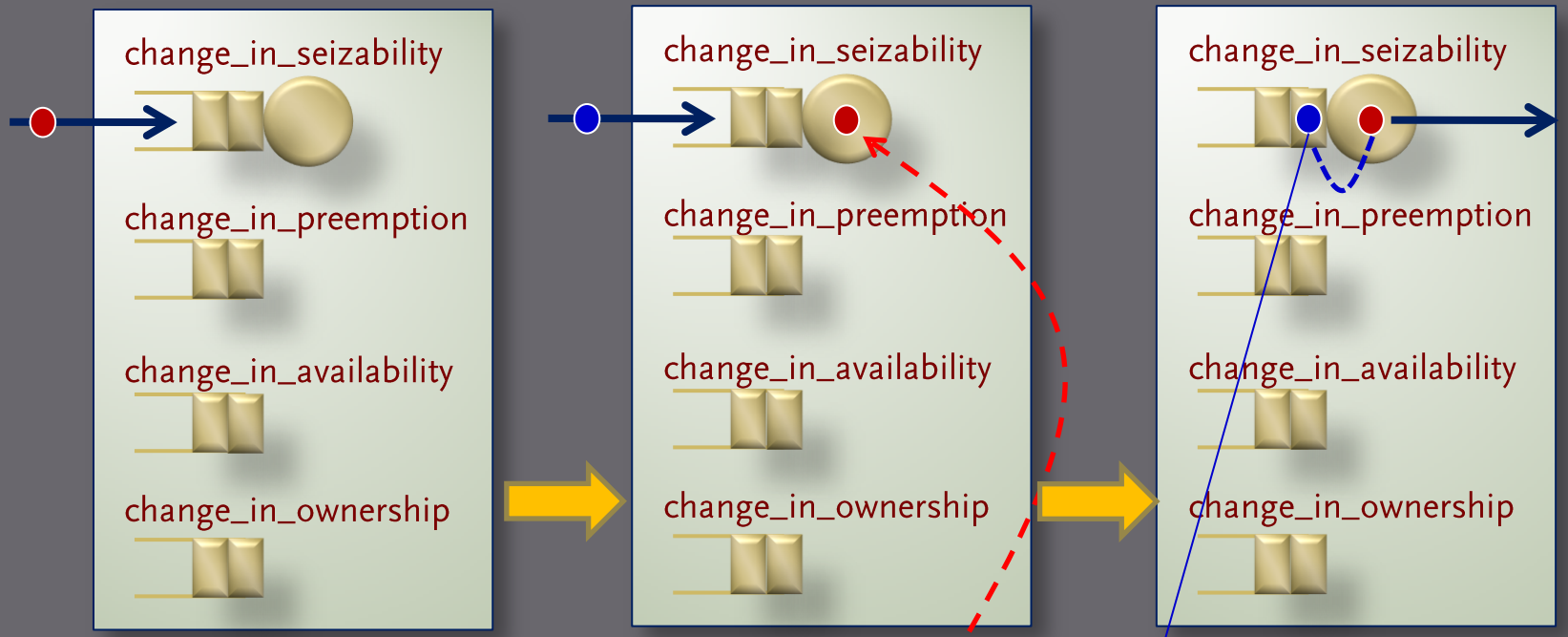
enqueue protokoll2;
advance(...); ...
depart protokoll2;

Seize-Operation

normale Benutzung

```
seize einrichtung;
  advance ...;
release einrichtung;
```

Blockierung der Pucks nachfolgender seize-Aufrufer

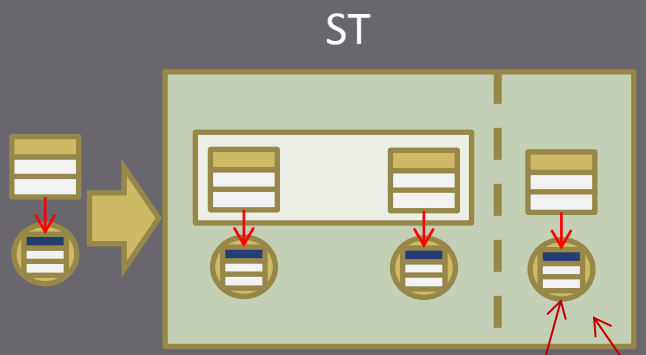


pointer (puck) owner
pointer (puck) preemptor
control state = **IDLE**
control availabilty = AVAIL

pointer (puck) owner
pointer (puck) preemptor
control state = **SEIZED**
control availabilty = AVAIL

Fortsetzungsbedingung:
state== IDLE && availability == AVAIL

Dynamische Änderung der Verfügbarkeit



Was passiert mit weiteren Belegungswünschen per size-Aufruf während der Nichtverfügbarkeit ?
s. übernächstes Bild

Kunden
actions {
…
seize ST;
advance …;
release ST;
…
}

Schadensauslöser
actions {
…
advance …;
funavail ST;
interrupt ST.owner;
}

Schadensbehebung
actions {
…
wait until ! ST.availability;
advance …;
favail ST;
resume ST.owner;
}

Operationen zur Verfügbarkeitsänderung

```
procedure FAVAIL(inout facility f)
        {
        if (f.availability == AVAIL)
                return;

        f.availability = AVAIL;
        reactivate list=f.change_in_availability;          // awaken pucks waiting for facility unavail
        if (f.state == IDLE)
                reactivate list=f.change_in_seizability; // awaken pucks waiting for facility seizable

        stop_interval f.unavail_time;
        start_interval f.avail_time;
        return;
        }
```

betrachten erst später
change_ in_availability

getrennte Intervalle für Statistiken werden gestartet und gestoppt

```
procedure FUNAVAIL(inout facility f)
        {
        if (f.availability == UNAVAIL)
                return;

        f.availability = UNAVAIL;
        reactivate list=f.change_in_availability;          // awaken pucks waiting for facility avail
        if (f.state == IDLE)
                reactivate list=f.change_in_seizability; // awaken pucks waiting for facility not seizable

        start_interval f.unavail_time;
        stop_interval f.avail_time;
        return;
        }
```

```
procedure SEIZE (inout facility f) {
        while (f.state != IDLE || f.availability != AVAIL)
            wait list=f.change_in_seizability;              // sleep

        f.owner = ACTIVE;
        f.state = SEIZED;
        reactivate list=f.change_in_ownership;              // awaken pucks waiting for facility in use
        if (f.availability == AVAIL)
                reactivate list=f.change_in_seizability;    // awaken pucks waiting for facility not seizable

        tabulate f.usage = 1.0;
        return;
}
```

kein **waituntil**-Einsatz,
d.h. die Control-Variablenfunktionalität wird nicht genutzt

Pucks, die auf Zustandsänderung warten

betrachten
change_ in_seizability
change_in_ownership

```
procedure RELEASE (inout facility f) {
        pointer(preemption) p;

        if (f.owner != ACTIVE)
                diagnose f run_time error(ACTIVE, f.title, f.owner) "_ cannot release _ (owner = _)";

        for (p = each preemption in f.preemption_stack)
                if (p -> preemptee == ACTIVE) {
                        ACTIVE -> preemption_count --;
                        remove p from f.preemption_stack;
                        return;
                }
        tabulate f.usage = 0.0    count = 0;
        f.owner = NULL;
        f.state = IDLE;

        reactivate list=f.change_in_ownership;          // awaken pucks waiting for facility not in use
        if (f.availability == AVAIL)
          reactivate list=f.change_in_seizability;      // awaken pucks waiting for facility seizable
        return;
}
```

Inhalt C.3

- **Teil A**
  Aspekte
  dynamis
- **Teil B**
  Die Mod
- **Teil C**
  Die aust
  SLX
- **Teil D**
  Modellie

- **C.1**
  Einführung und Ba
- **C.2**
  Stochastische Pro
- **C.3**
  GPSS-Elemente
- **C.4**
  DISCO-Elemente
- **C.5**
  Basissprache (Erg

1. Einführung
2. Zyklische Erzeugung aktiver Objekte (generate)
3. Bedienungseinrichtung (facility) und Warteschlangenstatistik (queue)
   - Normale Bedienung
   - Verfügbarkeitsänderung
   - Facility: Control-Variablen, Makros
   - Queue: Operationen, Makros
   - gate-Anweisung für Facility
   - Verdrängende Bedienung
   - Standard-Report (Facility, Queue)
   - Zusammenfassung
4. ...

Facility-Makros

- Typische Statusabfragen einer Facility:
  können bei wait until-Anweisungen benutzt werden
  da hier Control-Variablen zum Einsatz kommen

Makro	Bedeutung
FU(*facility_ident* **)**	Einrichtung ist belegt (in use)
FNU(*facility_ident* **)**	Einrichtung ist frei (not in use)
FS(*facility_ident* **)**	Einrichtung kann belegt werden (seizable)
FNS(*facility_ident* **)**	Einrichtung kann nicht belegt werden (not seizable)

hier: state

```
macro FU(#facility)              // Facility in use (boolean)
 definition
        {
        expand(#facility)                "((#).state != IDLE)";
        }

describe @FU                     "FU(x) is TRUE if facility x is in use.";

macro FNU(#facility)     // Facility not in use (boolean)
 definition
        {
        expand(#facility)                "((#).state == IDLE)";
        }

describe @FNU                    "FNU(x) is TRUE if facility x is not in use.";
```

Facility-Control-Variablen

state	Bedeutung
IDLE	Einrichtung ist frei
SEIZED	Einrichtung ist normal belegt
PREEMPTED	Einrichtung ist vorrangig belegt

availabilty	Bedeutung
AVAIL	erzeugt und einsatzbereit
UNAVAIL	erzeugt, aber nicht einsatzbereit

preemptor	Bedeutung
Puck-Zeiger	NULL oder aktueller verdrängender Puck

Nutzer kann eigene Abhängigkeiten bei Einsatz von **wait until** definieren und
dabei vorige Makros einsetzen oder direkt Attribute einer Facility

Barber-Shop: Facility, Queue

```
//*****************************************************************************
// EX-00020: SLX Barbershop Model
//*****************************************************************************
import <h7>
module barb13 {
//*****************************************************************************
// Global Declarations
//*****************************************************************************
        facility joe;
        queue joeq;
        rn_stream Arrivals seed=100000;
        rn_stream Service seed= 200000;
        constant float stop_time= 48000;
//*****************************************************************************
// Customer
//*****************************************************************************
 class customer {
        actions {
                        enqueue joeq;
                seize joe;
                        depart joeq;

                        advance rv_uniform(Service, 12.0, 18.0);
                release joe;
                terminate;
        }
}
//*****************************************************************************
// Run Control
//*****************************************************************************
procedure main {
        arrivals: customer
                iat = rv_uniform(Arrivals, 12.0, 24.0)
                until_time = stop_time;
        wait until (time >= stop_time and Q(joeq) == 0 and FNU(joe));
        report(system);
        exit(0);
}
} // End of barb13 module
```

Zur Erinnerung …

Makro für Facility
Zugriff auf Control-Attribut

Makro für Queue
Zugriff auf Control-Attribut

Inhalt C.3

- **Teil A**
  Aspekte
  dynamis
- **Teil B**
  Die Moc
- **Teil C**
  Die aust
  SLX
- **Teil D**
  Modellie

- **C.1**
  Einführung und Ba
- **C.2**
  Stochastische Pro
- **C.3**
  GPSS-Elemente
- **C.4**
  DISCO-Elemente
- **C.5**
  Basissprache (Erg

1. Einführung
2. Zyklische Erzeugung aktiver Objekte (generate)
3. Bedienungseinrichtung (facility) und Warteschlangenstatistik (queue)
   - Normale Bedienung
   - Verfügbarkeitsänderung
   - Facility: Control-Variablen, Makros
   - Queue: Operationen, Makros
   - gate-Anweisung für Facility
   - Verdrängende Bedienung
   - Standard-Report (Facility, Queue)
   - Zusammenfassung
4. ...

Queue: Operationen, Control-Variablen, Makros

```
passive class queue(string(*) report_title) {
    read_only {
            string(12)                  title;
            random_variable(time)       usage;
            interval                    total_time;
            double                      deltat;
    }

            int                         zero_count;
            initial { ... }
            report { ... }
            clear { ... }
};          // End of queue class

procedure QUEUE (inout queue q, int units) { ... }
procedure DEPART (inout queue q, int units) { ... }
```

Makros

Q(*queue-ident* **)** — *Aktuelle Länge, bei Zugriff auf usage.current_value*

SLX-Bezeichner	Bedeutung	SLX-Datentyp
weight_type	Gewichtungstyp	wtype
current_value	Aktueller Wert	control double
tlast	Zeitpunkt des letzten Updates	double
smallest_max	Kleinstes Maximum aller Statistiken	double
largest_min	Größtes Minimum aller Statistiken	double
histo	Verweis auf ein Histogramm	pointer(histogram)
title	Titel	string(16)
by_interval	Zugeordnete Intervalle	set(statistics)
master	Statistik über der gesamten Zeit	statistics

Anweisungen

enqueue *queue_ident* [**units=** *queue_units*]
depart *queue_ident* [**units=** *queue_units*]

Inhalt C.3

- **Teil A**
  Aspekte
  dynamis
- **Teil B**
  Die Mod
- **Teil C**
  Die aust
  SLX
- **Teil D**
  Modellie

- **C.1**
  Einführung und Ba
- **C.2**
  Stochastische Pro
- **C.3**
  GPSS-Elemente
- **C.4**
  DISCO-Elemente
- **C.5**
  Basissprache (Erg

1. Einführung
2. Zyklische Erzeugung aktiver Objekte (generate)
3. Bedienungseinrichtung (facility) und Warteschlangenstatistik (queue)
   - Normale Bedienung
   - Verfügbarkeitsänderung
   - Facility: Control-Variablen, Makros
   - Queue: Operationen, Makros
   - gate-Anweisung für Facility
   - Verdrängende Bedienung
   - Standard-Report (Facility, Queue)
   - Zusammenfassung
4. ...

Gate-Anweisung

- SLX bietet

  wait until zur Blockierung/Deblockierung aufrufender Prozesse

  in Abhängigkeit beliebiger Zustandsbedingungen

- h7_Modul definiert daneben die

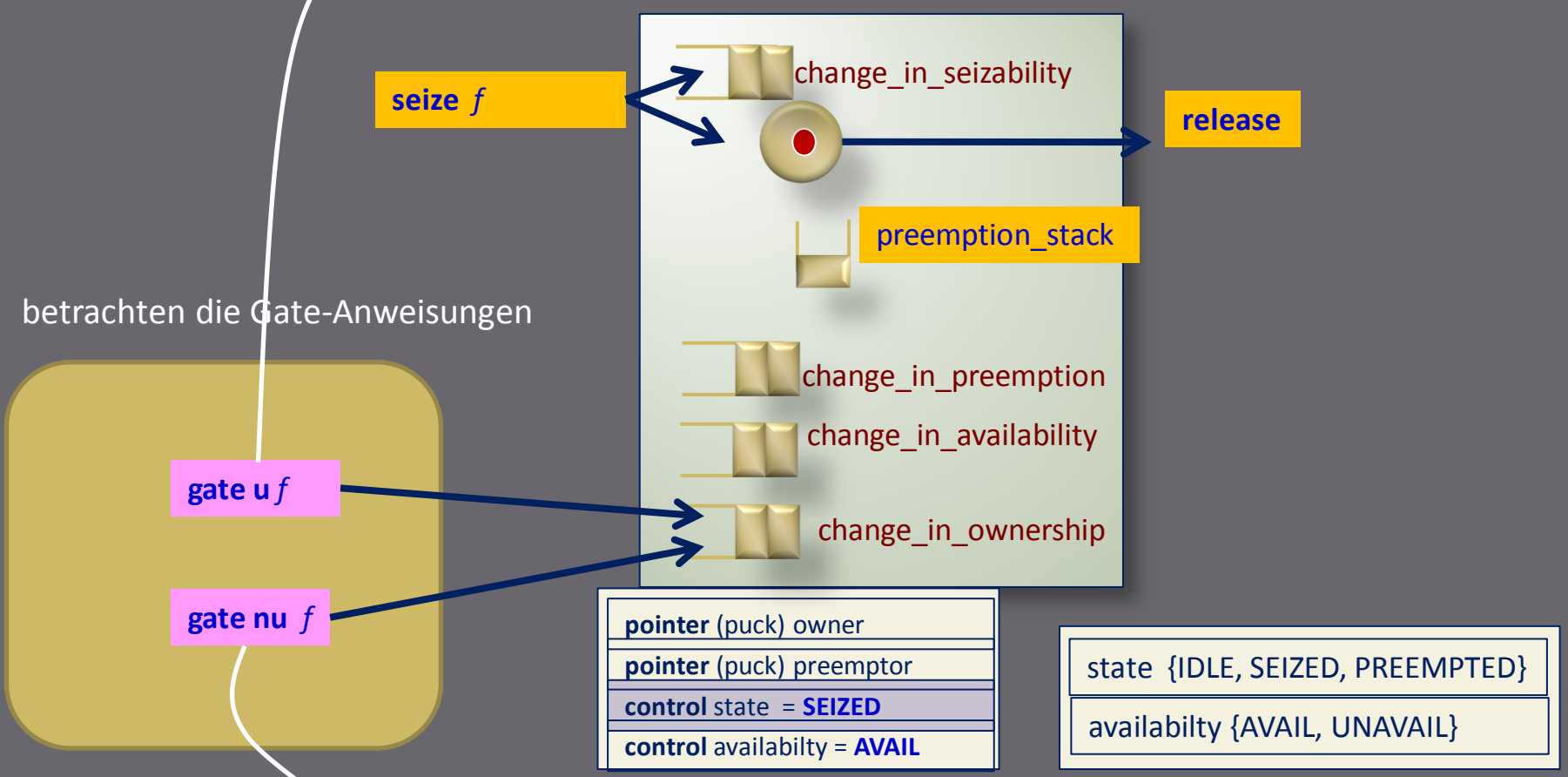
  gate-Anweisung (stammt aus GPSS)
  zur Blockierung/Deblockierung aufrufender Prozesse

  in Abhängigkeit von gewissen Zustandsbedingungen von Facility, Storage, Switch, …

Facility: GATE u und GATE nu

u ~ used
nu ~ not used

while (f.state < SEIZED)
wait list= f.change_in_ownership;



while (f.state >= SIZED)
wait list= f.change_in_ownership;

```
procedure SEIZE (inout facility f) {
        while (f.state != IDLE || f.availability != AVAIL)
            wait list=f.change_in_seizability;              // sleep

        f.owner = ACTIVE;
        f.state = SEIZED;
        reactivate list=f.change_in_ownership;              // awaken pucks waiting for facility in use
        if (f.availability == AVAIL)
                reactivate list=f.change_in_seizability;    // awaken pucks waiting for facility not seizable

        tabulate f.usage = 1.0;
        return;
}
```

kein **waituntil**-Einsatz,
d.h. die Control-Variablenfunktionalität wird nicht genutzt

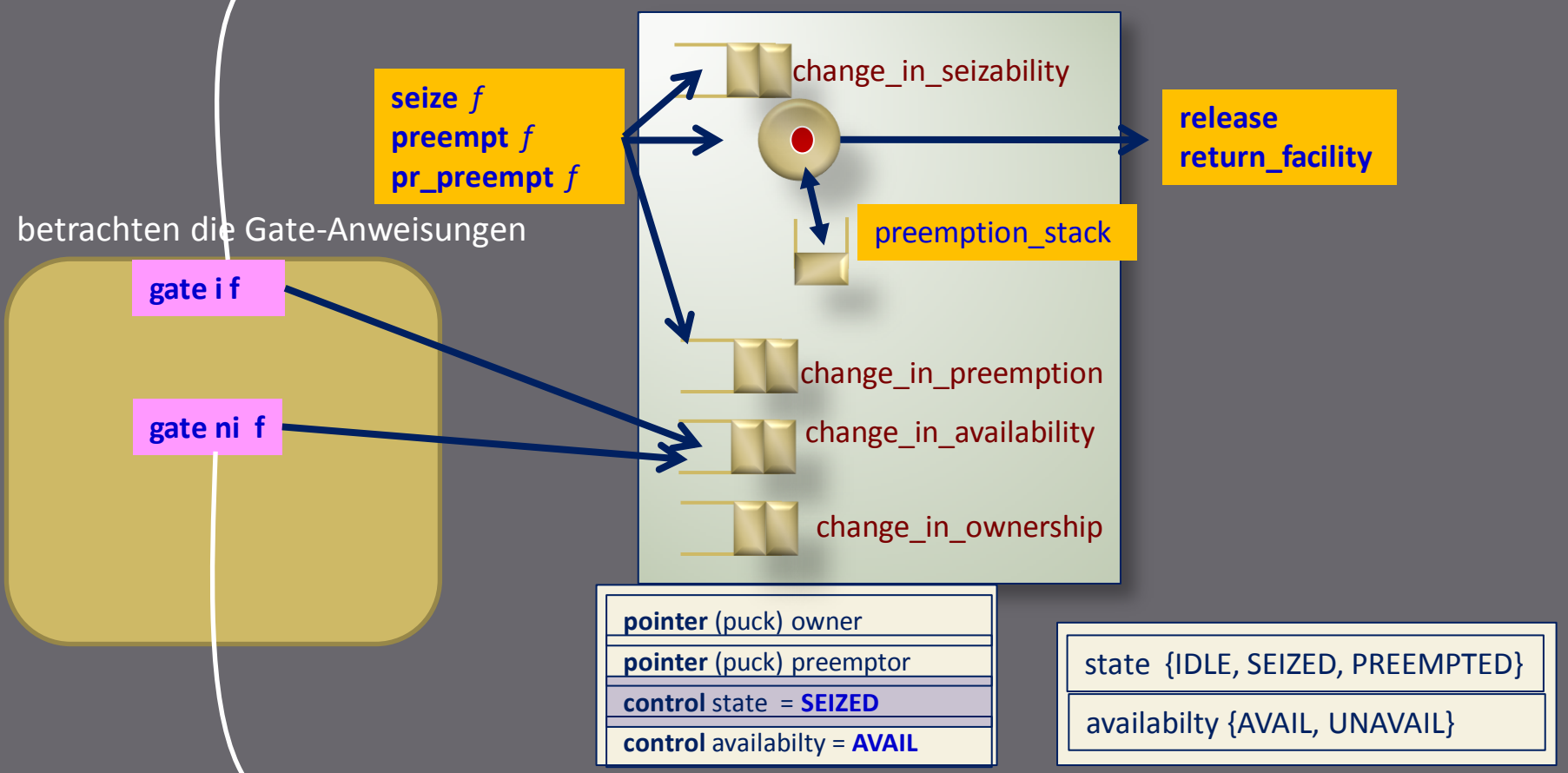
Pucks, die auf Zustandsänderung warten

```
procedure RELEASE (inout facility f) {
        pointer(preemption) p;

        if (f.owner != ACTIVE)
                diagnose f run_time error(ACTIVE, f.title, f.owner) "_ cannot release _ (owner = _)";

        for (p = each preemption in f.preemption_stack)
                if (p -> preemptee == ACTIVE) {
                        ACTIVE -> preemption_count --;
                        remove p from f.preemption_stack;
                        return;
                }
        tabulate f.usage = 0.0    count = 0;
        f.owner = NULL;
        f.state = IDLE;

        reactivate list=f.change_in_ownership;          // awaken pucks waiting for facility not in use
        if (f.availability == AVAIL)
          reactivate list=f.change_in_seizability;      // awaken pucks waiting for facility seizable
        return;
}
```

GATE i und GATE ni

i ~ interrupted
ni ~ not interrupted

while (f.state != PREEMPTED)
wait list= f.change_in_preemption;



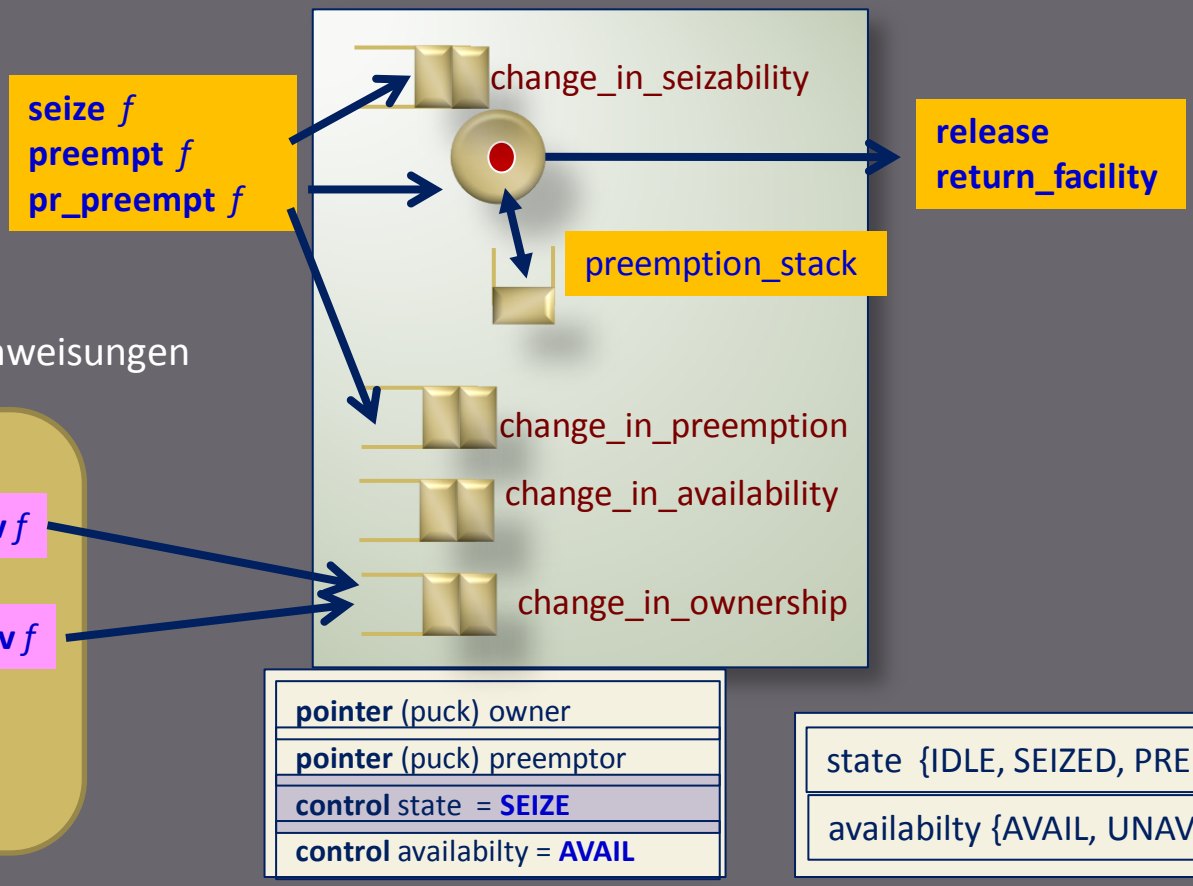
while (f.state == PREEMPTED)
wait list= f.change_in_preemption;

GATE fv und GATE fnv

fv ~ available
fnv ~ not available

verdrängend belegt

while (f.availability != AVAIL)
wait list= f.change_in_availability;



betrachten die Gate-Anweisungen

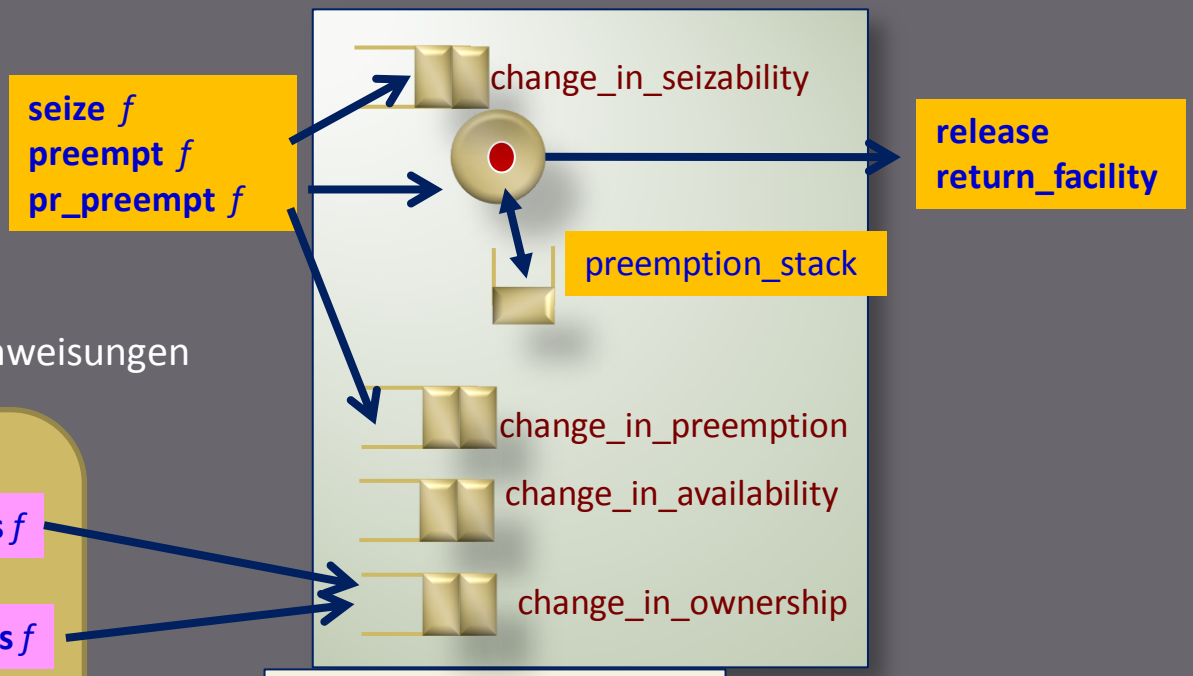
while (f.availability == AVAIL)
wait list= f.change_in_availability;

GATE fs und GATE fns

fs ~ available
fns ~ not available

normal belegt

while (f.availability != AVAIL || f.state >= SEIZED)
wait list= f. change_in_seizability;



betrachten die Gate-Anweisungen

gate fs *f*

gate fns *f*

pointer (puck) owner
pointer (puck) preemptor
control state = **SEIZE**
control availabilty = **AVAIL**

state {IDLE, SEIZED, PREEMPTED}
availabilty {AVAIL, UNAVAIL}

while (f.availability == AVAIL && f.state < SEIZED)
wait list= f.change_in_availability;

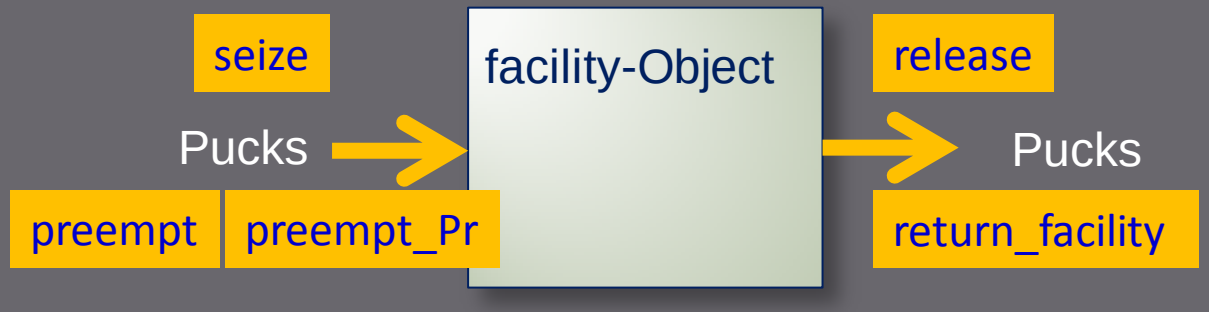
Inhalt C.3

- **Teil A**
  Aspekte
  dynamis
- **Teil B**
  Die Mod
- **Teil C**
  Die aust
  SLX
- **Teil D**
  Modellie

- **C.1**
  Einführung und Ba
- **C.2**
  Stochastische Pro
- **C.3**
  GPSS-Elemente
- **C.4**
  DISCO-Elemente
- **C.5**
  Basissprache (Erg

1. Einführung
2. Zyklische Erzeugung aktiver Objekte (generate)
3. Bedienungseinrichtung (facility) und Warteschlangenstatistik (queue)
   - Normale Bedienung
   - Verfügbarkeitsänderung
   - Facility: Control-Variablen, Makros
   - Queue: Operationen, Makros
   - gate-Anweisung für Facility
   - Verdrängende Bedienung
   - Standard-Report (Facility, Queue)
   - Zusammenfassung
4. ...

Funktionsüberblick: Facility



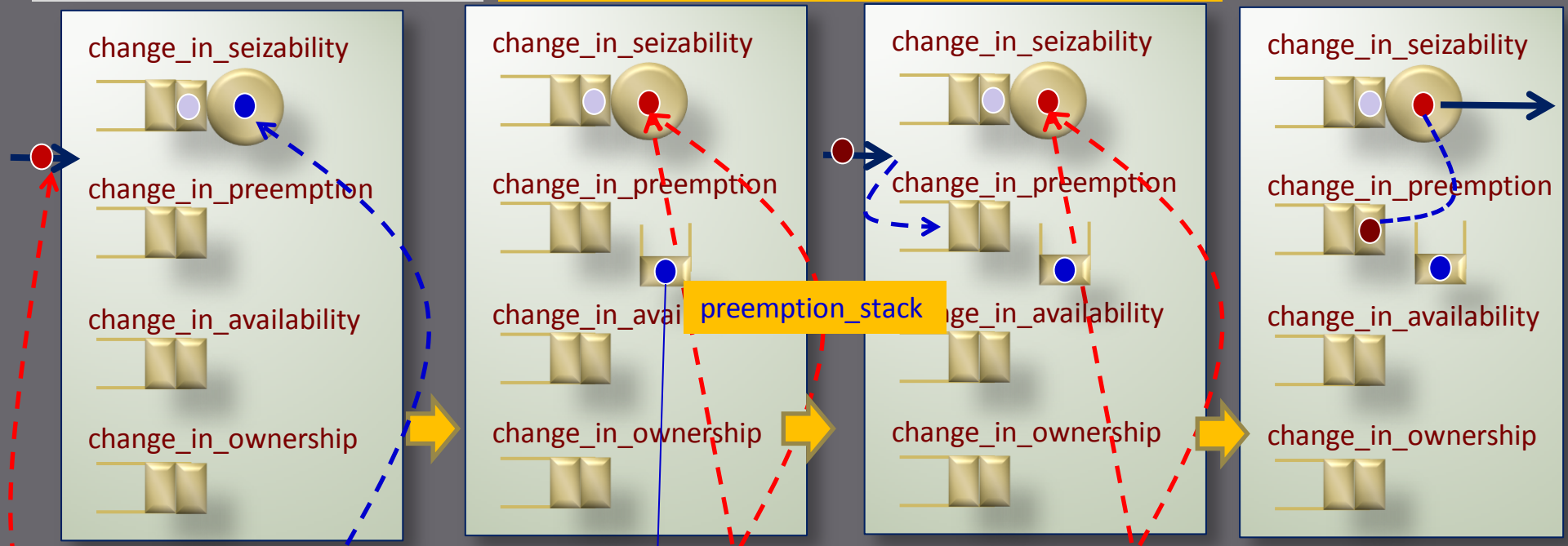
- Zu einem Zeitpunkt kann höchstens ein Prozess (Puck) die *Facility* in Benutzung haben, andere werden blockiert und müssen warten (**FIFO**).
- Der aktuelle "Besitzer/Nutzer"-Prozess kann von anderen Prozessen unterbrochen und verdrängt werden, er wird die Restzeit später nachholen können (**LIFO**).
- Prozesse können in der Fortsetzung ihrer Aktionen vom **Belegungszustand** einer *Facility* per **gate**-Anweisung bei Makronutzung abhängig gemacht werden, ohne dass sie selbst die Facility nutzen
- Darüber hinaus können in einer Anwendung Prozesse von (allen) **control**-Attributen einer Facilty per **wait until** abhängig gemacht werden. (Das Laufzeitssystem setzt kein **wait until** ein !)
- Alle Puck-Warteschlangen einer Facility (egal ob einfache Puck-Liste oder Set) bieten **keine Statistik (Nutzung von Queue erforderlich**)

Preempt-Operation

verdrängende Benutzung

preempt einrichtung;
  advance …;
return_facility einrichtung;

Unterbrechung von owner per interupt
(Eintrag in IP-List und in preemption_stack)



pointer (puck) owner
pointer (puck) preemptor
control state = **SEIZED**
control availabilty = A

pointer (puck) owner
pointer (puck) preemptor
control state = **PREEMPTED**
control availabilty =

pointer (puck) owner
pointer (puck) preemptor
control state = **PREEMPTED**
control availabilty = AVAIL

```
augment puck {
    int  preemption_count;
};
```

preemption_
count= 1

```
procedure PREEMPT (inout facility f) {
        pointer(preemption) p;
        forever {
                if (f.state >= PREEMPTED) {
                        wait list=f.change_in_preemption;       // sleep
                        continue;                               // and try again
                }
                if (f.availability != AVAIL)  {
                        wait list=f.change_in_availability;     // sleep
                        continue;                               // and try again
                }
                break;                                          // can be preempted
        }
        if (f.state == SEIZED) {
                p = new preemption;
                p -> preemptee = f.owner;
                f.owner -> preemption_count++;
                if (f.owner -> state == SCHEDULED)
                                interrupt f.owner;
                else            diagnose f run_time warning(f.owner) "Unable to interrupt _ (not currently SCHEDULED)";

                place p into f.preemption_stack;                // LIFO list
        }
        else {
                reactivate list=f.change_in_ownership;          // awaken pucks waiting for facility in use
                if (f.availability == AVAIL)
                     reactivate list=f.change_in_seizability;   // awaken pucks waiting for facility not seizable
        }
        f.preemptor = ACTIVE;
        f.state = PREEMPTED;
        reactivate list=f.change_in_preemption;                 // awaken pucks waiting for facility interrupted
        tabulate f.usage = 1.0;
        return;
}
```

control state {IDLE, SEIZED, PREEMPTED}

control availabilty {AVAIL, UNAVAIL}

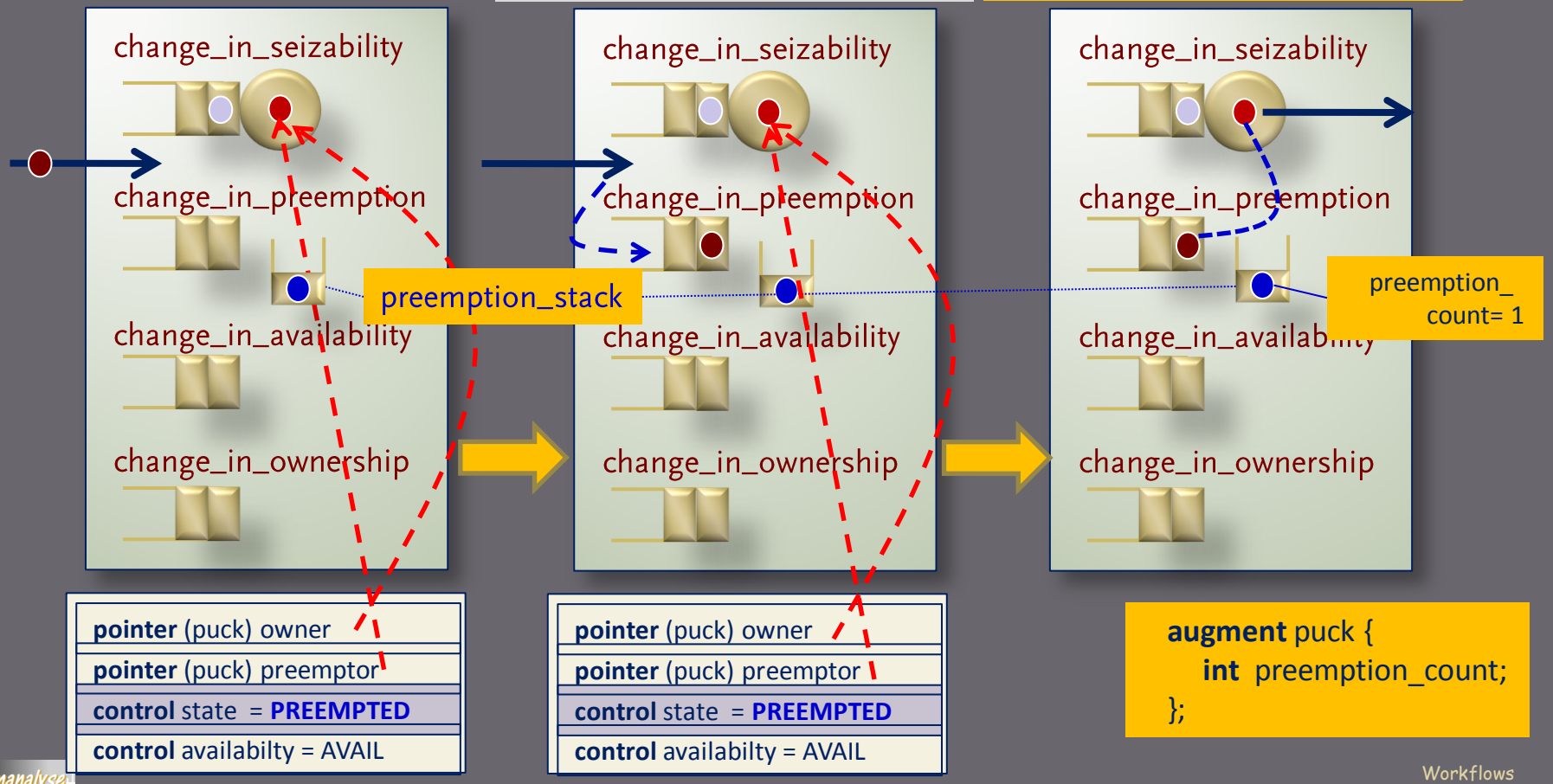
Preempt_pr-Operation (1)

verdrängende priorisierte Benutzung

priority des Aufrufers
kleiner gleich als die von owner

pr_preempt einrichtung;
  advance …;
return_facility einrichtung;

Blockierung des Pucks
des pr_preempt-Aufrufers




```
augment puck {
    int  preemption_count;
};
```

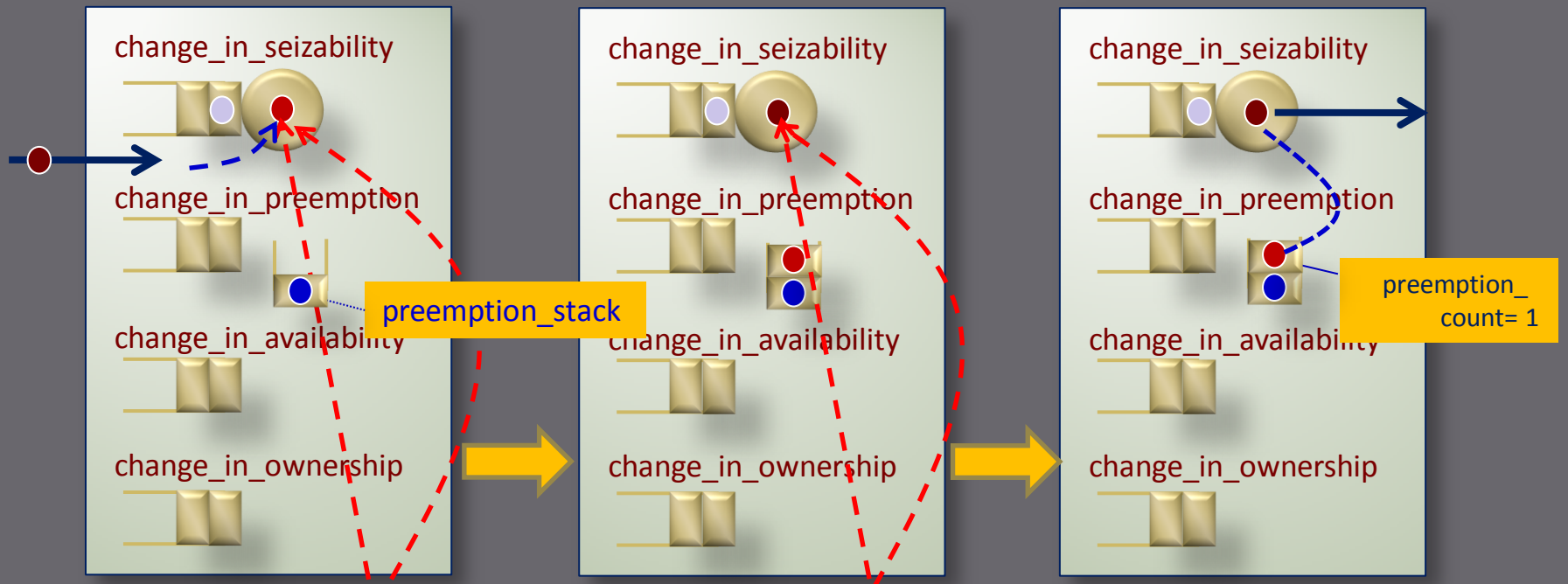
Preempt_pr-Operation (2)

verdrängende priorisierte Benutzung

priority des Aufrufers größer als die von owner

pr_preempt einrichtung;
  advance …;
return_facility einrichtung;

interupt von owner, Eintrag in: IP-List u. preemption_stack



pointer (puck) owner
pointer (puck) preemptor
control state = PREEMPTED
control availabilty = AVAIL

pointer (puck) owner
pointer (puck) preemptor
control state = PREEMPTED
control availabilty = AVAIL

augment puck {
  int preemption_count;
};

Inhalt C.3

- **Teil A**
  Aspekte
  dynamis
- **Teil B**
  Die Mod
- **Teil C**
  Die aust
  SLX
- **Teil D**
  Modellie

- **C.1**
  Einführung und Ba
- **C.2**
  Stochastische Pro
- **C.3**
  GPSS-Elemente
- **C.4**
  DISCO-Elemente
- **C.5**
  Basissprache (Erg

1. Einführung
2. Zyklische Erzeugung aktiver Objekte (generate)
3. Bedienungseinrichtung (facility) und Warteschlangenstatistik (queue)
   - Normale Bedienung
   - Verfügbarkeitsänderung
   - Facility: Control-Variablen, Makros
   - Queue: Operationen, Makros
   - gate-Anweisung für Facility
   - Verdrängende Bedienung
   - Standard-Report (Facility, Queue)
   - Zusammenfassung
4. ...

Standard-Ausgabe

```
class system_reporter {
    set(*) entity_reporters;  // inhomogene Liste mit

    report {
      print options=bold(time) "\n\nSystem Status at Time |_.____\n\n";
      report(entity_reporters);
    }

    clear {
      clear(entity_reporters);
    }
} system;
```

Jedes SLX-Programm verfügt über ein System_Reporter-Objekt

```
procedure main() {
        double real_start;
        double real_end;
        real_start = real_time();
        arrivals: customer
                iat = rv_uniform(rna, 12.0, 24.0)
                until_time = stop_time;
        wait until (time >= stop_time and Q(joeq) == 0 and FNU(joe));
        report(system);
        real_end = real_time();
        print(real_end - real_start) "\nExecution time: _.___ s\n";
        exit(0);
}
```

Barber-Shop-31-01-2013.slx: SLX-64 UL211 Lines: 7,458 Errors: 0 Warnings: 0 Lines/S
Execution begins

System Status at Time 480007.1404

Random Stream	Sample Count	Initial Position	Current Position	Antithetic Variates	Chi-Squar Uniformit
rna	26629	100000	126629	OFF	0.95
rns	26628	200000	226628	OFF	0.73

Facility	Total %Util	Avail %Util	Unavl %Util	Entries	Average Time/Puck	Current Status	Percent Avail	Seizing Puck	Preempting Puck
joe	83.17			26628	14.993	AVAIL	100.000	<NULL>	<NULL>

Queue	Current Contents	Maximum Contents	Average Contents	Total Entries	Zero Entries	Percent Zeros	Average Time/Item	Average > 0 Time/Item
joeq	0	2	0.05	26628	18220	68.42	0.853	2.701

Execution time: 0.081 s
Execution complete
Objects created: 46 passive and 26,629 active Pucks created: 26,631 Memory: 4 MB Time: 0.11 seconds

Flexibler Report und Clear

- Vordefinierte passive Modellelemente der Module h7 und statistics werden per **initial**-Property in spezifische Laufzeitlisten einsortiert
- globale Prozeduren **clear** und **report** benutzen diese Listen,

z.B. für Klassen von h7:

set (queue)	queue_set
set (facility)	facility_set
set (storage)	storage_set
set (user_chain)	user_chain_set
set (logic_switch)	logic_switch_set

z.B. für Klassen von Statistics:

set (random_variable) random_set

oder für Klassen von Buffer:
im Modul muss eine Liste

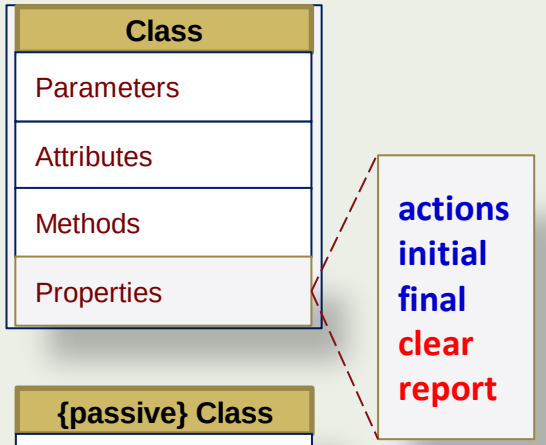
set (Buffer) buffer_set

eingerichtet sein

- Eine nutzerspezifische Reportgestaltung setzt eine Spezifikation der vom Nutzer gewünschten Ausgabeelemente voraus
- Anweisung **entity_class** in Kombination mit der Klasse **system_reporter** bieten Unterstützung für
  - **report** und
  - **clear**

Motivation

- Ein Simulationsexperiment besteht i.Allg. aus mehreren Simulationsläufen
- Häufig wird nach jedem Lauf eine **report**-Ausgabe gewünscht, wobei das Modell auf einen neuen Lauf vorbereitet werden muss
- **clear** dient der Vorbereitung
  ~ Löschen der Statistiken passiver Modellelemente, die im Vorgängerlauf bestimmt worden sind
- **report** der Ausführung eines Reports
  ~ Ausgabe bestimmter Modellgrößen/Statistiken



Hierarchie von Prozedur-Rufen für Report *bzw.* Clear

report (system)

report (queue_set)
report (storage_set)
report (facility_set)
...
report (nutzerKategory_set)

report (facility_object-1)
...
report (facility_object-n)

Objekt system der Klasse system_reporter

```
//*********************************************************************************
// SLX Barbershop Model
//*********************************************************************************
import <h7>
module barb13 {
//*********************************************************************************
// Global Declarations
//*********************************************************************************
        facility joe;
        queue joeq;
        rn_stream Arrivals seed=100000;
        rn_stream Service seed= 200000;
        constant float stop_time= 48000;
//*********************************************************************************
// Customer Object
//*********************************************************************************
 object customer {
        actions {
                enqueue joeq;
                seize joe;
                depart joeq;
                advance rv_uniform(Service, 12.0, 18.0);
                release joe;
                terminate;
        }
}
//*********************************************************************************
// Run Control
//*********************************************************************************
procedure main {
        arrivals: customer
        iat = rv_uniform(Arrivals, 12.0, 24.0)
        until_time = stop_time;
        wait until (time >= stop_time and Q(joeq) == 0 and FNU(joe));
        report(system);
        exit(0);
}
} // End of barb13 module
```

```
class system_reporter {
    set(*) entity_reporters;  // inhomogene Liste

    report {
       print options=bold(time) "\n\nSystem Status at Time |_.____\n\n";
       report(entity_reporters);
     }

     clear {
        clear(entity_reporters);
     }
} system;
```

- Die Standardausgabe für die GPSS-Objekte erfolgt per

  report (system);

- **report** (system) → system.report()
  ruft für alle Elemente
  der Liste entity_reporters
  die Report-Property auf

Blick in den h7-Modul

```
//*****************************************************************************
//        "Facilities"
//*****************************************************************************

    entity_class   facility;   // set=All_facilities;

    OEM static string(*) fac_format =

"|_____________   ___.__|  _____|  _____|  _________|  _______.___|  ____|____ ___.___|  _____|_____ _____|_____\n";

    typedef enum { IDLE, SEIZED, PREEMPTED }     facility_state;

    passive class preemption
        {
        pointer(puck)          preemptee;
        };

    augment puck
        {
        int              preemption_count;
        };

    passive class facility(string(*) report_title)
        {
        read_only
            {
            string(12)                          title;
            random_variable(time)               usage;
            interval                            total_time,
                                                avail_time,
                                                unavail_time;
            pointer(puck)                       owner;
            control pointer(puck)               preemptor;
            control facility_state              state;
            control enum { AVAIL, UNAVAIL }     availability;

            set(preemption) ranked LIFO preemption_stack;
            }

        pointer(puck) change_in_ownership,                  // read_write puck freezers
                      change_in_availability,
                      change_in_seizability,
                      change_in_preemption;

        initial
            {
            state = IDLE;
```

Nutzung der Anweisung entity_class

{passive} Puck

pointer (Puck*)	frozen_successor
float	mark_time
float	move_time
int	priority
pointer (*)	puck_object
enum Puckstatus	state
boolean	wait_incurred

Puck-Erweiterung

verschiedene Puck-Listen

Generierung am Beispiel

- Sei A nutzereigene Klasse
  (oder h7-Klasse facility, …
  die im Standard-Report aufgenommen
  werden soll
- **entity_class** A title="Report for all A-Elemente :" **;**

```
class system_reporter {
    set(*) entity_reporters;   // inhomogene Liste

    report {
        ...
        report(entity_reporters);
    }

    clear {
        clear(entity_reporters);
    }
} system;
```

erzeugt

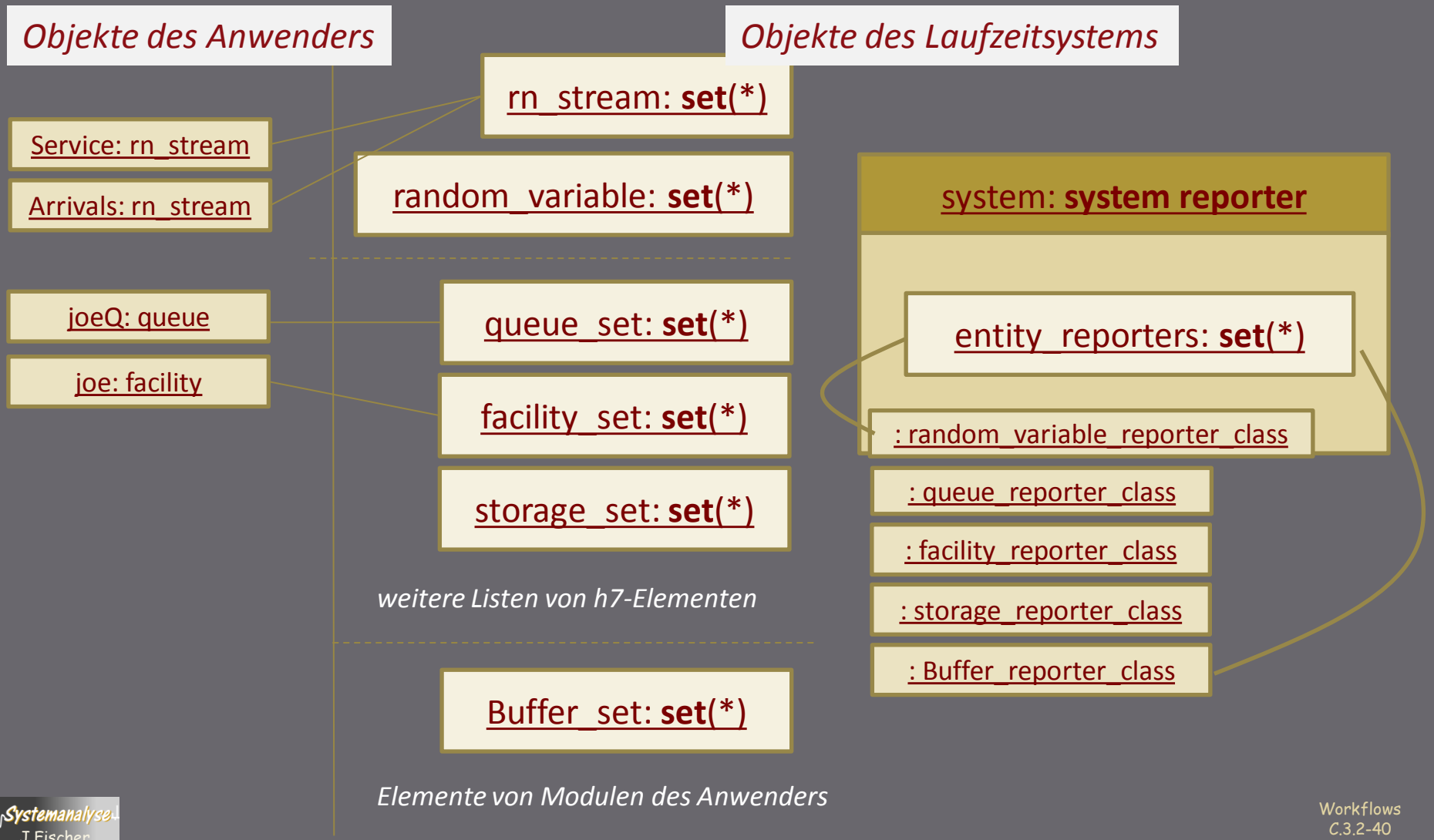
1
```
set(A) ranked(ascending title) A_set;
```

2
```
class A_reporter_class {
    initial {
            place ME into system.entity_reporters;
    }
    report {
            if (A_set.size > 0) {
                print options=bold, underline ("Report for all A's\n";
                report(A_set);
            }
    }
    clear {
            clear(A_set);
    }
};
```

3
```
A_reporter_class    A_reporter;
```

Nutzung der SLX-Spracherweiterung

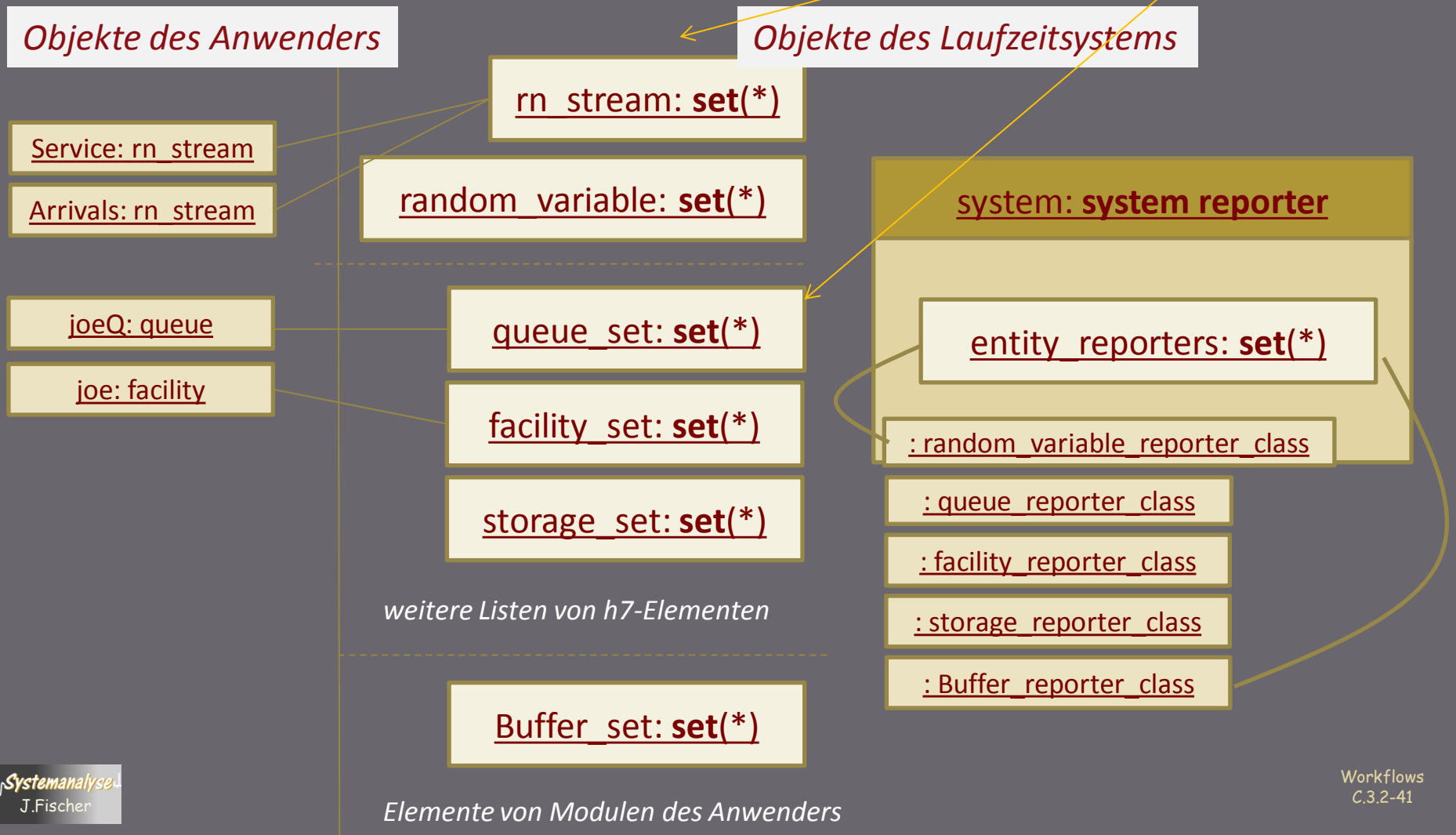
- durch Aufruf von entity_class
  für jede Kategorie von Modellkomponenten für report und clear
  entsteht im SLX-Laufzeitsystem:



- Aufruf von report (system

  führt zum Aufruf von report jedes Elements von entity_reporters

  dies führt zum spezifischen report aller Elemente der dazugehörigen globalen Liste



Hierarchische Organisation von report und clear

- Einrichtung von Objekten nutzerspezifischer Reporter-Klassen

- Anweisung

  entity_class *class_ident* [**title** = *report_titel*] [**set** = *set_ident*]

  erzeugt:

  a) ein homogenes Set von Objekten als *class_ident*_**set** mit dem Namen *set_ident*
  in Analogie zu queue_set,
  facility_set usw.

  Sortierungsangabe

  b) eine Klasse *class_ident*_reporter_class
  in Analogie zur Klasse queue_reporter, ...
  und

  keine Attribute, aber mit
  initial, report, clear

  c) eine Instanz der Klasse *class_ident*_reporter_class
  mit dem Bezeichner *class_ident*_reporter

  fügt die Instanz in
  system. entity_reporters
  ein

  ruft für alle Elemente von
  set_ident die
  report-Property

Individuelle Report-Zusammenstellung

… von GPSS-Modellelementen

- **report** (system)
- **report** (facility_set)
  report (A_set)
- **report** (*facility_object*)
  report (*A_object*)

```
procedure main() {
        double real_start;
        double real_end;
        real_start = real_time();
        arrivals: customer
                iat = rv_uniform(rna, 12.0, 24.0)
                until_time = stop_time;
        wait until (time >= stop_time and Q(joeq) == 0 and FNU(joe));
        //report(system);
        //real_end = real_time();
        //print(real_end - real_start) "\nExecution time: _.___ s\n";

        report (queue_set);
        report (joe);
        exit(0);
}
```

Barber-Shop-31-01-2013.slx: SLX-64 UL211 Lines: 7,455 Errors: 0 Warnings: 0 Lines/Second: 1,108,743 Memory: 4 MB
Execution begins

Queue	Current Contents	Maximum Contents	Average Contents	Total Entries	Zero Entries	Percent Zeros	Average Time/Item	Average > 0 Time/Item
joeq	0	2	0.05	26628	18220	68.42	0.853	2.701

Facility	Total %Util	Avail %Util	Unavl %Util	Entries	Average Time/Puck	Current Status	Percent Avail	Seizing Puck	Preempting Puck
joe	83.17			26628	14.993	AVAIL	100.000	<NULL>	<NULL>

Execution complete
Objects created: 46 passive and 26,629 active Pucks created: 26,631 Memory: 4 MB Time: 0.06 seconds

Clear

- Versetzen des Modells in seinen **Anfangszustand** als Vorbereitung eines neuen Simulationslaufes

- Ausführung der **clear**-Property
  - Modellzeit wird **Null**
  - alle Pucks werden beseitigt,
    bis auf den Puck, der den **clear**-Aufruf ausgelöst hat

    ACHTUNG: sollte **clear** nicht durch **main** ausgelöst worden sein, wird auch der **main-Puck** vernichtet.
    Dies hat wiederum den Abbruch der Simulation zur Folge
    (Mitteilung eines Laufzeitfehlers)

- Das Löschen der eigentlichen **Prozess-Objekte** muss vom Nutzer organisiert werden (Aktionen von **clear** der betroffenen **Active-Classes**)
  ebenso die Re-initialisierung von Attributen

  SLX unterstützt dies aber bereits für alle SLX-Systemklassen (inkl. GPSS)

clear system

… bewirkt: die Freigabe von

n

- Zufallszahlengeneratoren
- Einrichtungen
- Speicher
- Warteschlangen
- Nutzerketten
- Logische Schalter
- Statistik-Objekte (s. später)
- nutzerdefinierte Objekte, die per **initial** in
  system.entity_reporters
  aufgenommen worden sind und über ein nutzerdefiniertes **clear**-Property verfügen

Individuelle Clear-Organisation

- unterschiedliche Möglichkeiten
  1. Einfügung einer clear-Property in die entsprechenden Nutzerklassen und Aufnahme in system. entity_reporters
     (s. letzte Folie)
  2. Definition einer speziellen clear-Klasse, in deren clear-Property die nutzerspezifischen Anweisungen des clearing beschrieben sind
  3. Expliziter (individueller) Aufruf von clear, nach dem Aufruf Aufruf von clear system

Inhalt C.3

- **Teil A**
  Aspekte
  dynamis
- **Teil B**
  Die Mod
- **Teil C**
  Die aust
  SLX
- **Teil D**
  Modellie

- **C.1**
  Einführung und Ba
- **C.2**
  Stochastische Pro
- **C.3**
  GPSS-Elemente
- **C.4**
  DISCO-Elemente
- **C.5**
  Basissprache (Erg

1. Einführung
2. Zyklische Erzeugung aktiver Objekte (generate)
3. Bedienungseinrichtung (facility) und Warteschlangenstatistik (queue)
   - Normale Bedienung
   - Verfügbarkeitsänderung
   - Facility: Control-Variablen, Makros
   - Queue: Operationen, Makros
   - gate-Anweisung für Facility
   - Verdrängende Bedienung
   - Standard-Report (Facility, Queue)
   - Zusammenfassung
4. ...

Puck-Listen von Facility

IDLE, SEIZED, PREEMPTED

- Preemption stack vom Typ: set
  - Erfassung verdrängter Pucks,
  - Registrierung in der Facility bleibt erhalten

AWAIL, UNAWAIL

- Change_in seizabilty vom Typ: pointer(puck)
  a) Erfassung von Pucks, die belegte Einrichtung per Seize nutzen wollen
  b) Erfassung von Pucks, die in Abhängigkeit eines Zustandswechsels der Einrichtung (IDLE – SEIZED) per gate warten
- Change_in_ownership vom Typ: pointer(puck)
  - von Pucks, die in Abhängigkeit eines Zustandswechsels der Einrichtung (IDLE – SEIZED/PREEMPT) per gate warten
- Change_in_availability vom Typ: pointer(puck)
  - von Pucks, die in Abhängigkeit eines Zustandswechsels der Einrichtung (AWAIL – UNAWAIL) per gate warten
- Change_in_preemption vom Typ: pointer(puck)
  - von Pucks, die in Abhängigkeit eines Zustandswechsels der Einrichtung (! PREEMPTED – PREEMPTED) per gate warten

facility-Operationen

```
typedef enum { IDLE, SEIZED, PREEMPTED }          facility_state;

passive class preemption {
        pointer (puck)                            preemptee;
};

augment puck {
        int                                       preemption_count;
};

passive class facility (string(*) report_title)  {
  read_only {
        string (12)                               title;
        random_variable (time)                    usage;
        interval                                  total_time, avail_time, unavail_time;
        pointer (puck)                            owner;
        control pointer (puck)                    preemptor;
        control facility_state                    state;
        control enum { AVAIL, UNAVAIL }           availability;
        set (preemption) ranked LIFO              preemption_stack;
  }
        pointer(puck)     change_in_ownership,          // read_write puck freezers
                          change_in_availability,
                          change_in_seizability,
                          change_in_preemption;
        initial { ... }
        report { ... }
        clear { ... }
};      // End of facility class

procedure SEIZE (inout facility f) { ... }
procedure RELEASE (inout facility f) { ... }
procedure PREEMPT (inout facility f) { ... }
procedure PREEMPT_PR (inout facility f) { ... }
procedure RETURN_FACILITY (inout facility f) { ... }
procedure FAVAIL (inout facility f) { ... }
procedure FUNAVAIL (inout facility f) { ... }
```

Makros (liefern logische/arithmetische Werte)

FU (*fac-ident* **)** — *used/not used*
FNU (*fac-ident* **)**
FS (*fac-ident* **)** — *seizable/ not seizable*
FNS (*fac-ident* **)**
FR (*fac-ident* **)** — *Auslastung: [0,1]*

Anweisungen (evtl. Aufrufer-Blockierung)

seize {*fac_ident* | **ALL (** *list_fac-ident* **)** }
release {*fac_ident* | **ALL (** *list_fac-ident* **)** }
preempt *fac_ident*
preempt_pr *fac_ident*
return_facility *fac_ident*
favail *fac_ident*
funavail *fac_ident*

Anweisungen
(evt. Aufrufer-Blockierung

gate fs *fac_ident*
gate nfs {*fac_ident*
gate i *fac_ident*
gate ni *fac_ident*
gate fv *fac_ident*
gate nfv *fac_ident*
gate u *fac_ident*
gate nu *fac_ident*

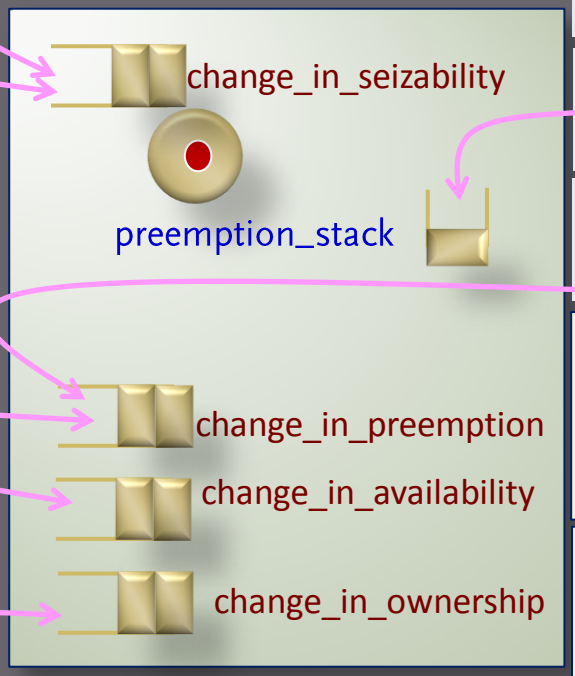
Facility-Operationen, Bedingungen und ihre Puck-Listen

Prozesse, die per **gate** auf (availability != AVAIL oder state >= SEIZED) bzw. (availability == AVAIL oder state < SEIZED) warten

gate fs *f*
gate nfs *f*

seize *f*

Prozesse, die per **seize** nach **FIFO** warten müssen



Prozesse, die verdrängt und unterbrochen worden sind

preempt *f*
pr_preempt *f*

Prozesse, die per **preempt** oder **pr_preempt** nach **FIFO** warten müssen

pr_preempt *f*

pointer (puck) owner
pointer (puck) preemptor
control state = **IDLE**
control availabilty = **AVAIL**

Statusanzeige

state {IDLE, SEIZED, PREEMPTED}

availabilty {AVAIL, UNAVAIL}

gate u *f*
gate nu *f*

Prozesse, die per **gate** auf state< SEIZED bzw. state>=SIZED warten

Prozesse, die per **gate** auf availability==AVAIL bzw. availability==UNAVAIL warten

gate i *f*
gate ni *f*

Prozesse, die per **gate** auf state==PREEMPTED oder state != PREEMPTED warten

gate fv *f*
gate fnv *f*