VORLESUNG

Automatisierung industrieller Workflows

Teil C: Die Sprache SLX

- Einführung und Grundkonzepte -

Joachim Fischer

Position

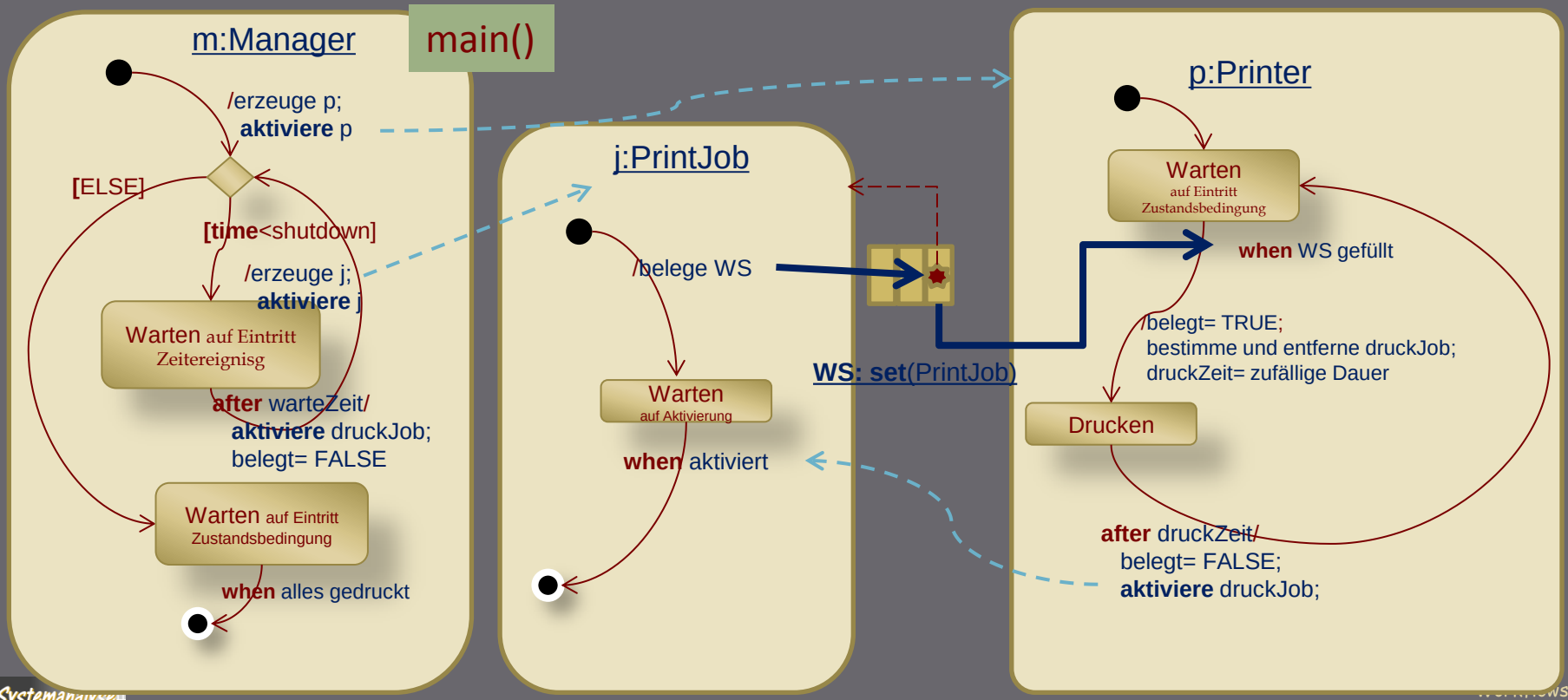
- **Teil A**
  Aspekte
  dynamis
- **Teil B**
  Die Moc
- **Teil C**
  Die ausf
  SLX
- **Teil D**
  Modellie

- **C.1**
  Einführung und Ba
- **C.2**
  Stochastische Pro
- **C.3**
  Vertiefung der SL
- **C.4**
  GPSS-Elemente
- **C.5**
  DISCO-Elemente
- **C.6**
  Basissprache (Erg

1. SLX-Überblick
2. Syntaxkonventionen
3. Elementare Datentypen
4. Klassen
5. Objekte
6. Typ Set
7. Anweisungen, Prozeduren
8. Einfache Ausgabe
9. Modellierungselemente in SLX (allgemein)
10. Prozesse und Pucks
11. Beispiel (Drucker, Druckjobs):
12. SLX- Laufzeitsystem (Simulator-Kern)
13. Zeitkonzepte
14. Scheduling-Operationen

Beispiel (Aufgabe, Prozess-Schema)

- zyklische Erzeugung von Druckjobs innerhalb eines vorgeg. Zeitintervalls
- Erfassung der Druckjobs in einer Warteschlange
- Realisierung (zeitliche Nachbildung) des Druckens durch einen Drucker
- Beendigung der Simulation, sobald letzter Job gedruckt worden ist
- Anzahl und mittlere Wartezeit der Druckjobs sind zu ermitteln



```
module basic {
    rn_stream arrivals, service ;
    int shutdown_time = 5*8*60, jobs_in;
    control integer  jobs_printed ;
    float total_queueing_time;
    pointer( Printer ) printer;
    set ( PrinterJob ) waiting_line; // Queue for arriving jobs
```

globale initialisierte Daten

1 ZE $_{Modell}$ = 1 min $_{Real}$

1 Objekt mit 1 Puck

```
class Printer {
  boolean printer_busy;
  pointer ( puck ) my_puck; // Puck for Printer Process
  pointer ( PrinterJob ) owner; // Current Print Job
  actions {
    my_puck = ACTIVE; // Store the Pointer of the Puck
   forever {
     wait; // Wait for Print Jobs
       // While Contents of Job Queue != 0
     while ( (first PrinterJob in waiting_line) != NULL ) {
        // Take First Print Job
        owner = first PrinterJob in waiting_line;
      remove owner from waiting_line ;
        printer_busy = TRUE;
        total_queueing_time += (time –
                                owner->my_puck->mark_time);
      advance rv_uniform ( service , 0.5, 15.0 ) ;
                                // printing time
        printer_busy = FALSE;
        jobs_printed ++;
        // Wake up the Sleeping Print Job Process
      reactivate owner->my_puck;
        owner = NULL;
     } // while
    } // forever
  } // actions
}
```

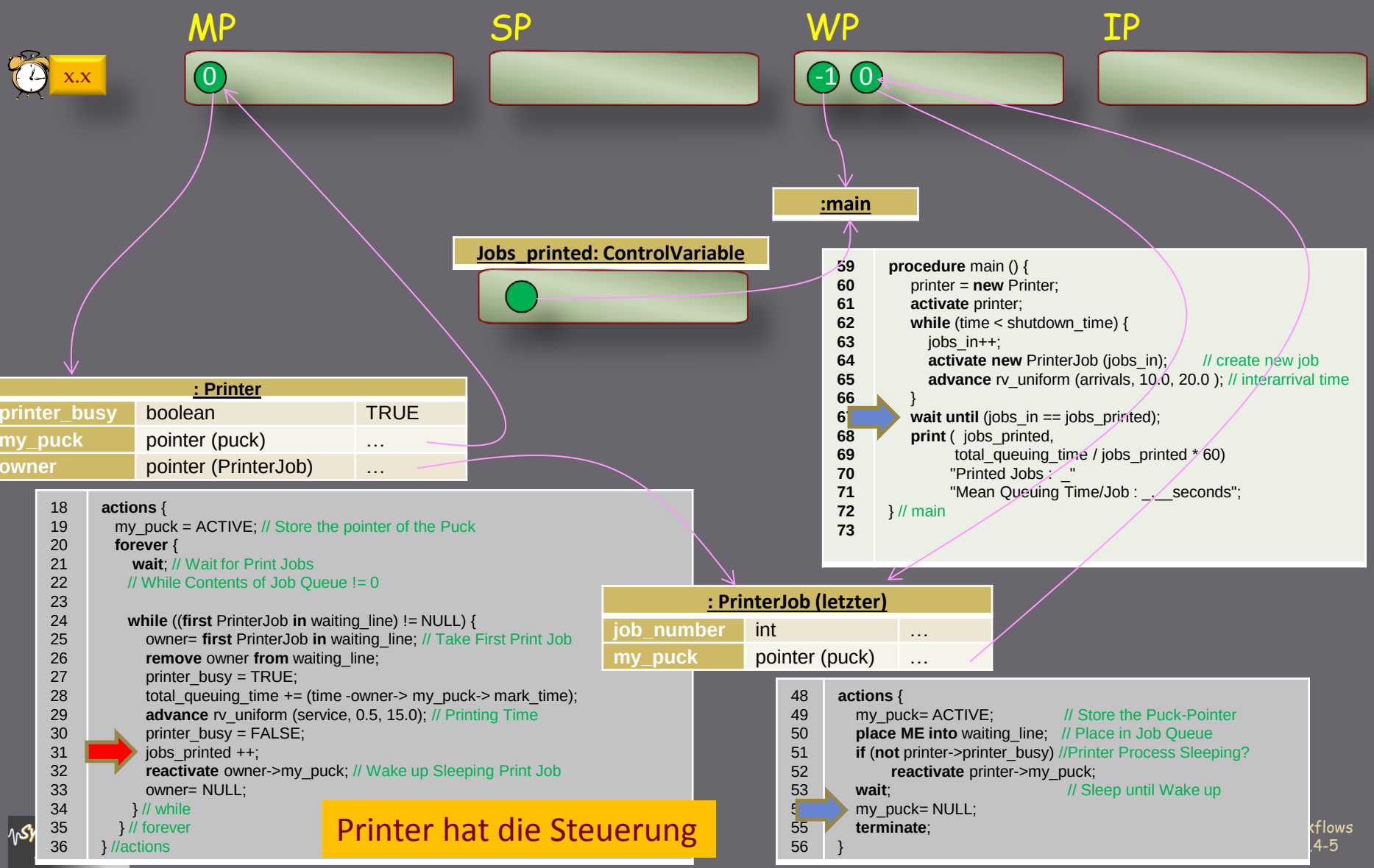
n Objekte mit 1 Puck pro Objekt

```
class PrinterJob ( in integer in_job_num ) {
        int job_number;
        pointer ( puck ) my_puck; // Puck for printer_job process
        initial {
          job_number = in_job_num;
        }
       actions {
            my_puck = ACTIVE; // Store the pointer of the puck
            place ME into waiting_line; // Place in Job Queue
            if ( not printer->printer_busy ) // Printer process sleeping?
                 reactivate printer->my_puck;
            wait; // Sleep until wake up
            my_puck = NULL;
            terminate ;
       }
    } // cl_printer_job
```

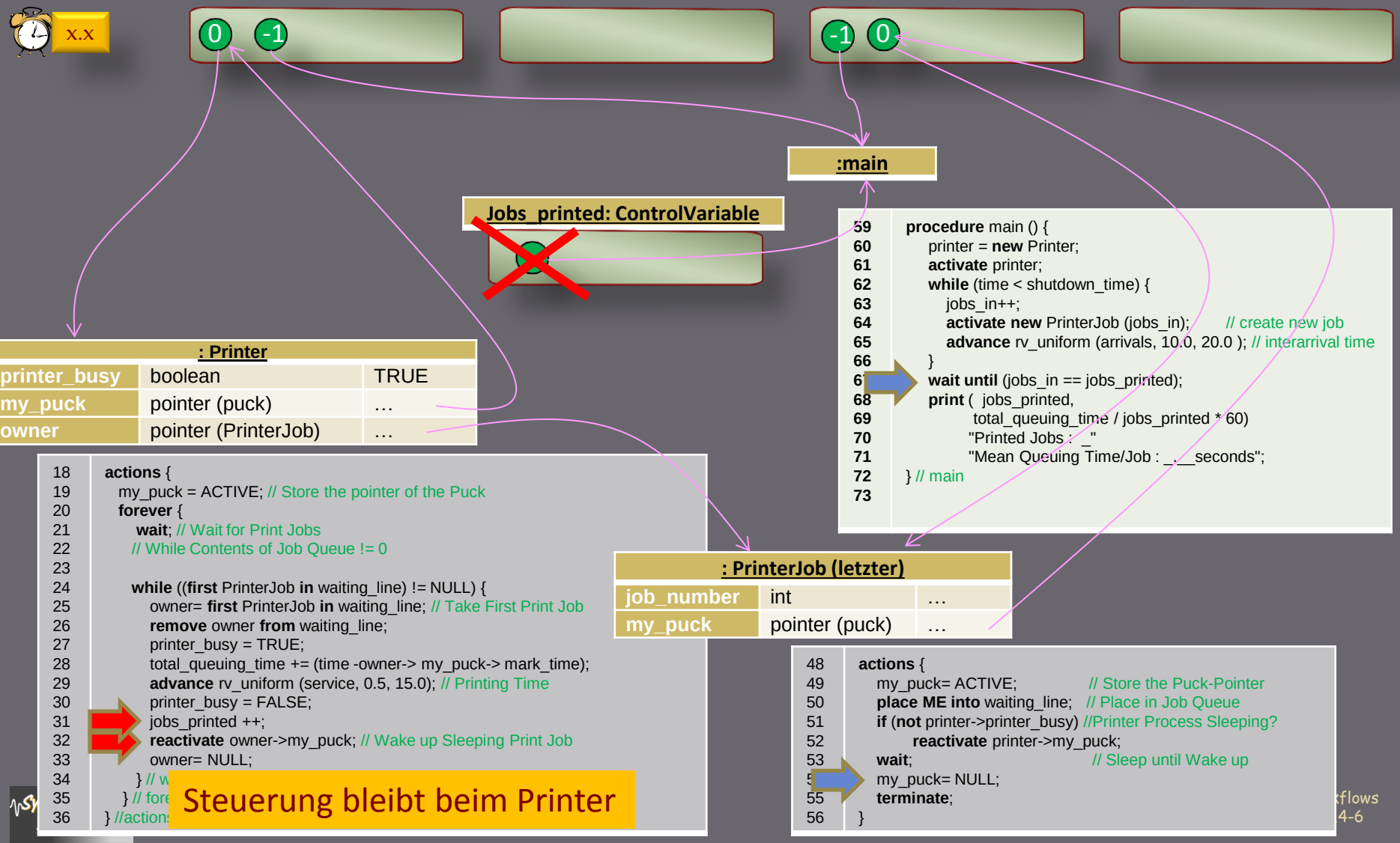
1 Instanz mit 1 Puck

```
  procedure main() {
     printer = new Printer ;
    activate printer;
    while (time < shutdown_time) {
       jobs_in++;
     activate new PrinterJob ( jobs_in ); // Create new jobs
     advance rv_uniform( arrivals , 10.0, 20.0 ) ;// interarrivel time
    }
    wait until ( jobs_in == jobs_printed);
    print ( jobs_printed ,
              total_queueing_time / jobs_printed * 60 )
              "Printed Jobs : _"
              "Mean Queueing Time/Job : _.__ seconds";
   }// main
  }
```

In Nähe der Abbruchsituation



Reaktivierung von main



Reaktivierung von PrinterJob (durch Printer)

MP | SP | WP | IP

MP: 0 | 0 | -1

WP: 0

:main

Jobs_printed: ControlVariable

```
59  procedure main () {
60      printer = new Printer;
61      activate printer;
62      while (time < shutdown_time) {
63          jobs_in++;
64          activate new PrinterJob (jobs_in);        // create new job
65          advance rv_uniform (arrivals, 10.0, 20.0 ); // interarrival time
66      }
67      wait until (jobs_in == jobs_printed);
68      print (  jobs_printed,
69               total_queuing_time / jobs_printed * 60)
70              "Printed Jobs : _"
71              "Mean Queuing Time/Job : _.__seconds";
72  } // main
73
```

: Printer		
printer_busy	boolean	TRUE
my_puck	pointer (puck)	…
owner	pointer (PrinterJob)	…

```
18  actions {
19    my_puck = ACTIVE; // Store the pointer of the Puck
20    forever {
21      wait; // Wait for Print Jobs
22      // While Contents of Job Queue != 0
23
24      while ((first PrinterJob in waiting_line) != NULL) {
25        owner= first PrinterJob in waiting_line; // Take First Print Job
26        remove owner from waiting_line;
27        printer_busy = TRUE;
28        total_queuing_time += (time -owner-> my_puck-> mark_time);
29        advance rv_uniform (service, 0.5, 15.0); // Printing Time
30        printer_busy = FALSE;
31        jobs_printed ++;
32        reactivate owner->my_puck; // Wake up Sleeping Print Job
33        owner= NULL;
34      } // w
35    } // fore
36  } //action
```

: PrinterJob (letzter)		
job_number	int	…
my_puck	pointer (puck)	…

```
48  actions {
49    my_puck= ACTIVE;              // Store the Puck-Pointer
50    place ME into waiting_line;   // Place in Job Queue
51    if (not printer->printer_busy) //Printer Process Sleeping?
52        reactivate printer->my_puck;
53    wait;                         // Sleep until Wake up
      my_puck= NULL;
      terminate;
    }
```

Steuerung bleibt immernoch beim Printer

Verlassen der While-Schleife (durch Printer)

MP | SP | WP | IP

MP: 0 0 -1

:main

Jobs_printed: ControlVariable

: Printer		
printer_busy	boolean	TRUE
my_puck	pointer (puck)	…
owner	pointer (PrinterJob)	NULL

```
59  procedure main () {
60      printer = new Printer;
61      activate printer;
62      while (time < shutdown_time) {
63          jobs_in++;
64          activate new PrinterJob (jobs_in);        // create new job
65          advance rv_uniform (arrivals, 10.0, 20.0 ); // interarrival time
66      }
67      wait until (jobs_in == jobs_printed);
68      print (  jobs_printed,
69               total_queuing_time / jobs_printed * 60)
70              "Printed Jobs : _"
71              "Mean Queuing Time/Job : _.__seconds";
72  } // main
73
```

```
18  actions {
19     my_puck = ACTIVE; // Store the pointer of the Puck
20     forever {
21        wait; // Wait for Print Jobs
22        // While Contents of Job Queue != 0
23
24        while ((first PrinterJob in waiting_line) != NULL) {
25           owner= first PrinterJob in waiting_line; // Take First Print Job
26           remove owner from waiting_line;
27           printer_busy = TRUE;
28           total_queuing_time += (time -owner-> my_puck-> mark_time);
29           advance rv_uniform (service, 0.5, 15.0); // Printing Time
30           printer_busy = FALSE;
31           jobs_printed ++;
32           reactivate owner->my_puck; // Wake up Sleeping Print Job
33           owner= NULL;
34        } // w
35     } // fore
36  } //action
```

Aufruf von wait

: PrinterJob (letzter)		
job_number	int	…
my_puck	pointer (puck)	…

```
48  actions {
49     my_puck= ACTIVE;              // Store the Puck-Pointer
50     place ME into waiting_line;   // Place in Job Queue
51     if (not printer->printer_busy) //Printer Process Sleeping?
52           reactivate printer->my_puck;
53     wait;                         // Sleep until Wake up
5
55     my_puck= NULL;
56     terminate;
   }
```

Steuerungswechsel zu PrinterJob

MP SP WP IP

0 -1 0

:main

Jobs_printed: ControlVariable

: Printer		
printer_busy	boolean	TRUE
my_puck	pointer (puck)	…
owner	pointer (PrinterJob)	NULL

```
59  procedure main () {
60      printer = new Printer;
61      activate printer;
62      while (time < shutdown_time) {
63          jobs_in++;
64          activate new PrinterJob (jobs_in);        // create new job
65          advance rv_uniform (arrivals, 10.0, 20.0 ); // interarrival time
66      }
67      wait until (jobs_in == jobs_printed);
68      print (  jobs_printed,
69               total_queuing_time / jobs_printed * 60)
70              "Printed Jobs :   _"
71              "Mean Queuing Time/Job : _.__seconds";
72  } // main
73
```

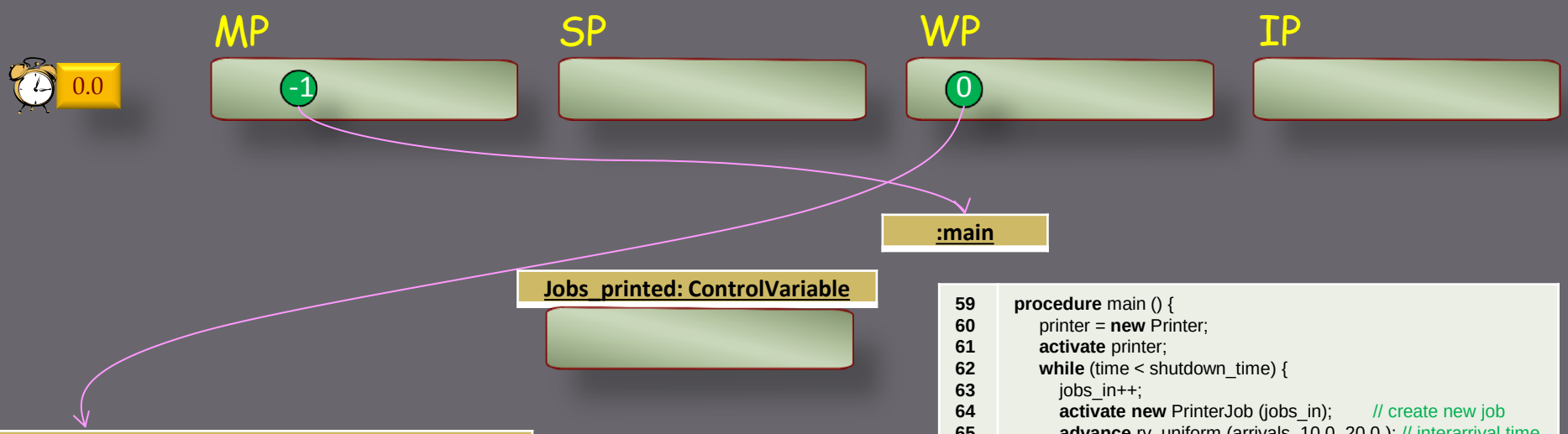
```
18  actions {
19    my_puck = ACTIVE; // Store the pointer of the Puck
20    forever {
21      wait; // Wait for Print Jobs
22      // While Contents of Job Queue != 0
23
24      while ((first PrinterJob in waiting_line) != NULL) {
25        owner= first PrinterJob in waiting_line; // Take First Print Job
26        remove owner from waiting_line;
27        printer_busy = TRUE;
28        total_queuing_time += (time -owner-> my_puck-> mark_time);
29        advance rv_uniform (service, 0.5, 15.0); // Printing Time
30        printer_busy = FALSE;
31        jobs_printed ++;
32        reactivate owner->my_puck; // Wake up Sleeping Print Job
33        owner= NULL;
34      } // while
35    } // forever
36  } //actions
```

: PrinterJob (letzter)		
job_number	int	…
my_puck	pointer (puck)	…

```
48  actions {
49    my_puck= ACTIVE;              // Store the Puck-Pointer
50    place ME into waiting_line;   // Place in Job Queue
51    if (not printer->printer_busy) //Printer Process Sleeping?
52        reactivate printer->my_puck;
53    wait;                         // Sleep until Wake up
54    my_puck= NULL;
55    terminate;
56  }
```

Puck-Vernichtung

Steuerungswechsel zu Main



: Printer		
printer_busy	boolean	TRUE
my_puck	pointer (puck)	…
owner	pointer (PrinterJob)	NULL

```
59  procedure main () {
60      printer = new Printer;
61      activate printer;
62      while (time < shutdown_time) {
63          jobs_in++;
64          activate new PrinterJob (jobs_in);        // create new job
65          advance rv_uniform (arrivals, 10.0, 20.0 ); // interarrival time
66      }
67      wait until (jobs_in == jobs_printed);
68      print (  jobs_printed,
69               total_queuing_time / jobs_printed * 60)
70              "Printed Jobs :   _"
71              "Mean Queuing Time/Job : _.__seconds";
72  } // main
73
```

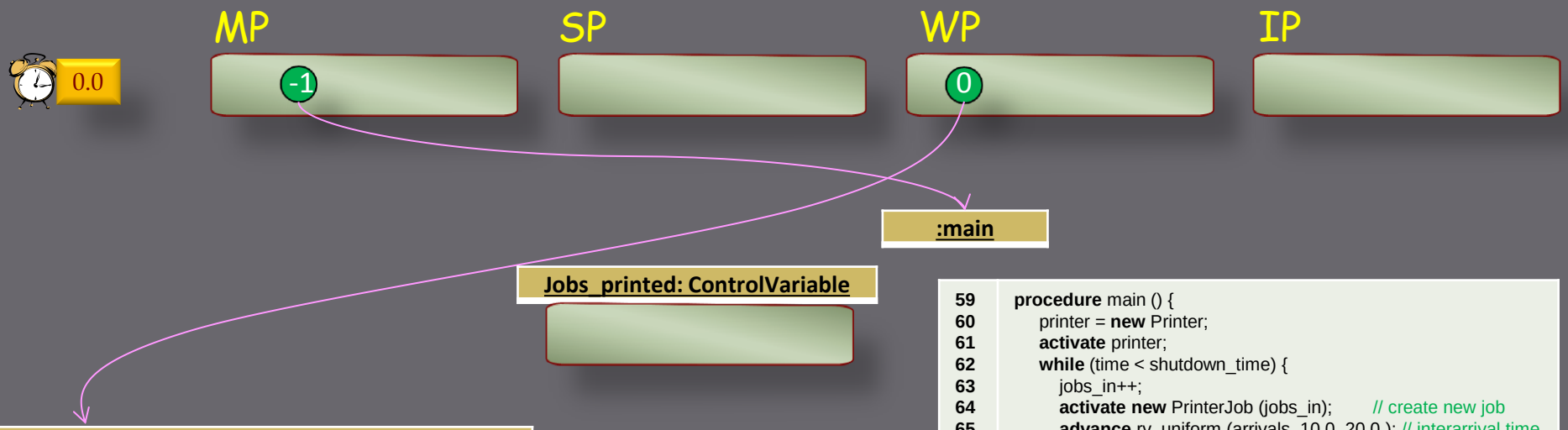
```
18  actions {
19    my_puck = ACTIVE; // Store the pointer of the Puck
20    forever {
21      wait; // Wait for Print Jobs
22      // While Contents of Job Queue != 0
23
24      while ((first PrinterJob in waiting_line) != NULL) {
25        owner= first PrinterJob in waiting_line; // Take First Print Job
26        remove owner from waiting_line;
27        printer_busy = TRUE;
28        total_queuing_time += (time -owner-> my_puck-> mark_time);
29        advance rv_uniform (service, 0.5, 15.0); // Printing Time
30        printer_busy = FALSE;
31        jobs_printed ++;
32        reactivate owner->my_puck; // Wake up Sleeping Print Job
33        owner= NULL;
34      } // while
35    } // forever
36  } //actions
```

: PrinterJob (letzter)		
job_number	int	…
my_puck	pointer (puck)	…

```
48  actions {
49    my_puck= ACTIVE;              // Store the Puck-Pointer
50    place ME into waiting_line;   // Place in Job Queue
51    if (not printer->printer_busy) //Printer Process Sleeping?
52        reactivate printer->
53    wait;
54    my_puck= NULL;
55    terminate;
56  }
```

Puck-Freigabe
Objekt-Freigabe

Ausgabe und Programmabbruch



: Printer		
printer_busy	boolean	TRUE
my_puck	pointer (puck)	…
owner	pointer (PrinterJob)	NULL

```
59  procedure main () {
60      printer = new Printer;
61      activate printer;
62      while (time < shutdown_time) {
63          jobs_in++;
64          activate new PrinterJob (jobs_in);        // create new job
65          advance rv_uniform (arrivals, 10.0, 20.0 ); // interarrival time
66      }
67      wait until (jobs_in == jobs_printed);
68      print (  jobs_printed,
69               total_queuing_time / jobs_printed * 60)
70              "Printed Jobs :   _"
71              "Mean Queuing Time/Job : _.__seconds";
72  } // main
73
```

```
18  actions {
19    my_puck = ACTIVE; // Store the pointer of the Puck
20    forever {
21      wait; // Wait for Print Jobs
22      // While Contents of Job Queue != 0
23
24      while ((first PrinterJob in waiting_line) != NULL) {
25        owner= first PrinterJob in waiting_line; // Take First Print Job
26        remove owner from waiting_line;
27        printer_busy = TRUE;
28        total_queuing_time += (time - owner-> my_puck-> mark_time);
29        advance rv_uniform (service, 0.5, 15.0); // Printing Time
30        printer_busy = FALSE;
31        jobs_printed ++;
32        reactivate owner->my_puck; // Wake up Sleeping Print Job
33        owner= NULL;
34      } // while
35    } // forever
36  } //actions
```

mark_time ~ Generierungszeit des Pucks vom aktuellen Printer_Job-Objekt

Ausgabe

EX-0014-Printer-Printerjob-Main.slx: SLX-64 CT103 Lines: 1,797 Errors: 0 Warnings: 0 Lines/Second: 885,259 Memory: 2 MB
Execution begins
Printed Jobs : 163
Mean Queueing Time/Job : 12.39 seconds

System Status at Time 2412.1422

Random Stream	Sample Count	Initial Position	Current Position	Antithetic Variates	Chi-Square Uniformity
arrivals	163	200000	200163	OFF	0.32
service	163	400000	400163	OFF	0.79

Execution complete
Objects created: 10 passive and 165 active Pucks created: 166 Memory: 2 MB Time: 0.05 Seconds

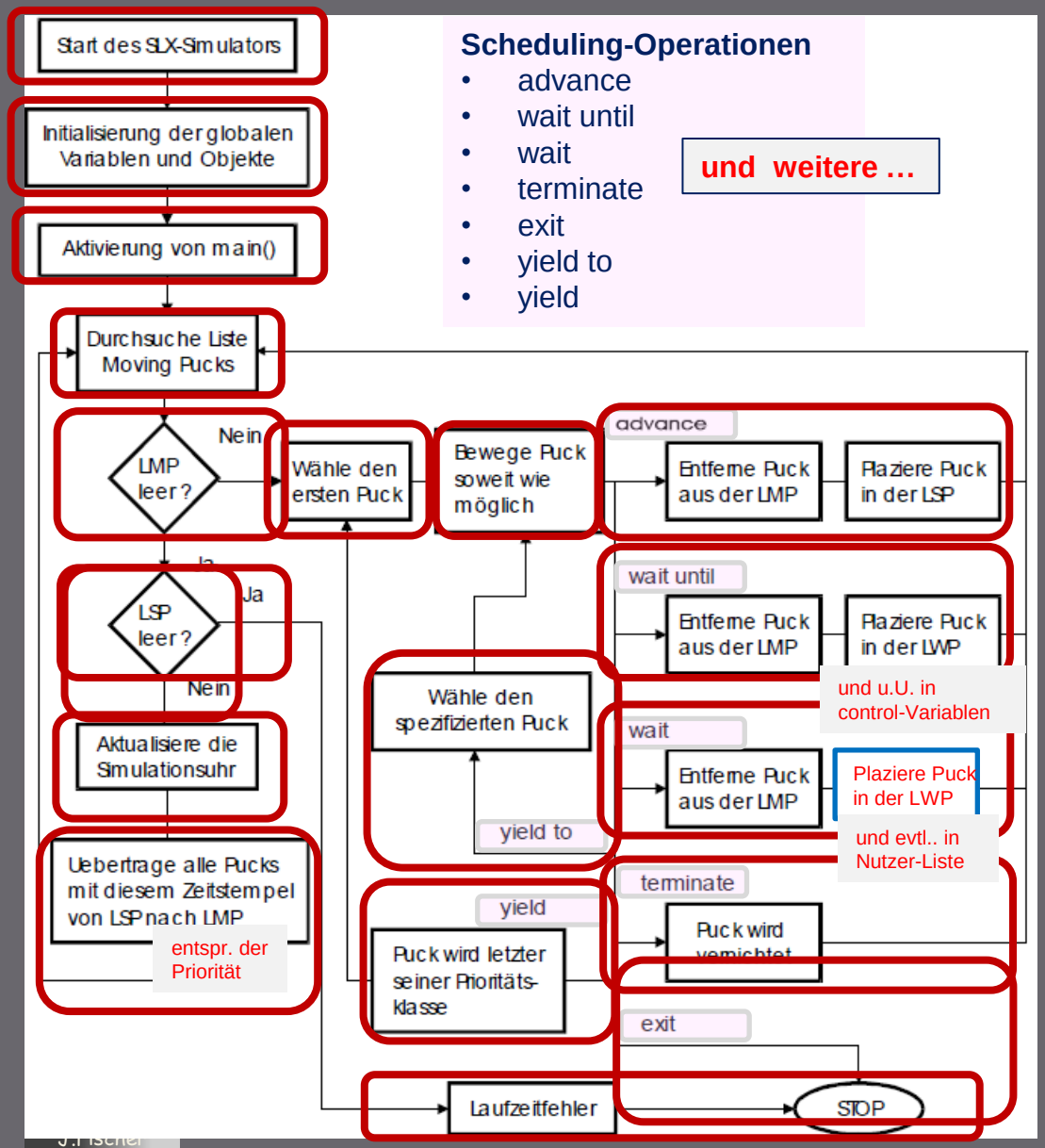
Position

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

- **C.1**
  Einführung und Ba
- **C.2**
  Stochastische Pro
- **C.3**
  Vertiefung der SL
- **C.4**
  GPSS-Elemente
- **C.5**
  DISCO-Elemente
- **C.6**
  Basissprache (Erg

1. SLX-Überblick
2. Syntaxkonventionen
3. Elementare Datentypen
4. Klassen
5. Objekte
6. Typ Set
7. Anweisungen, Prozeduren
8. Einfache Ausgabe
9. Modellierungselemente in SLX (allgemein)
10. Prozesse und Pucks
11. Beispiel (Drucker, Druckjobs)
12. SLX- Laufzeitsystem (Simulator-Kern)
13. Zeitkonzepte
14. Scheduling-Operationen

Kern des Simulationslaufzeitssystems



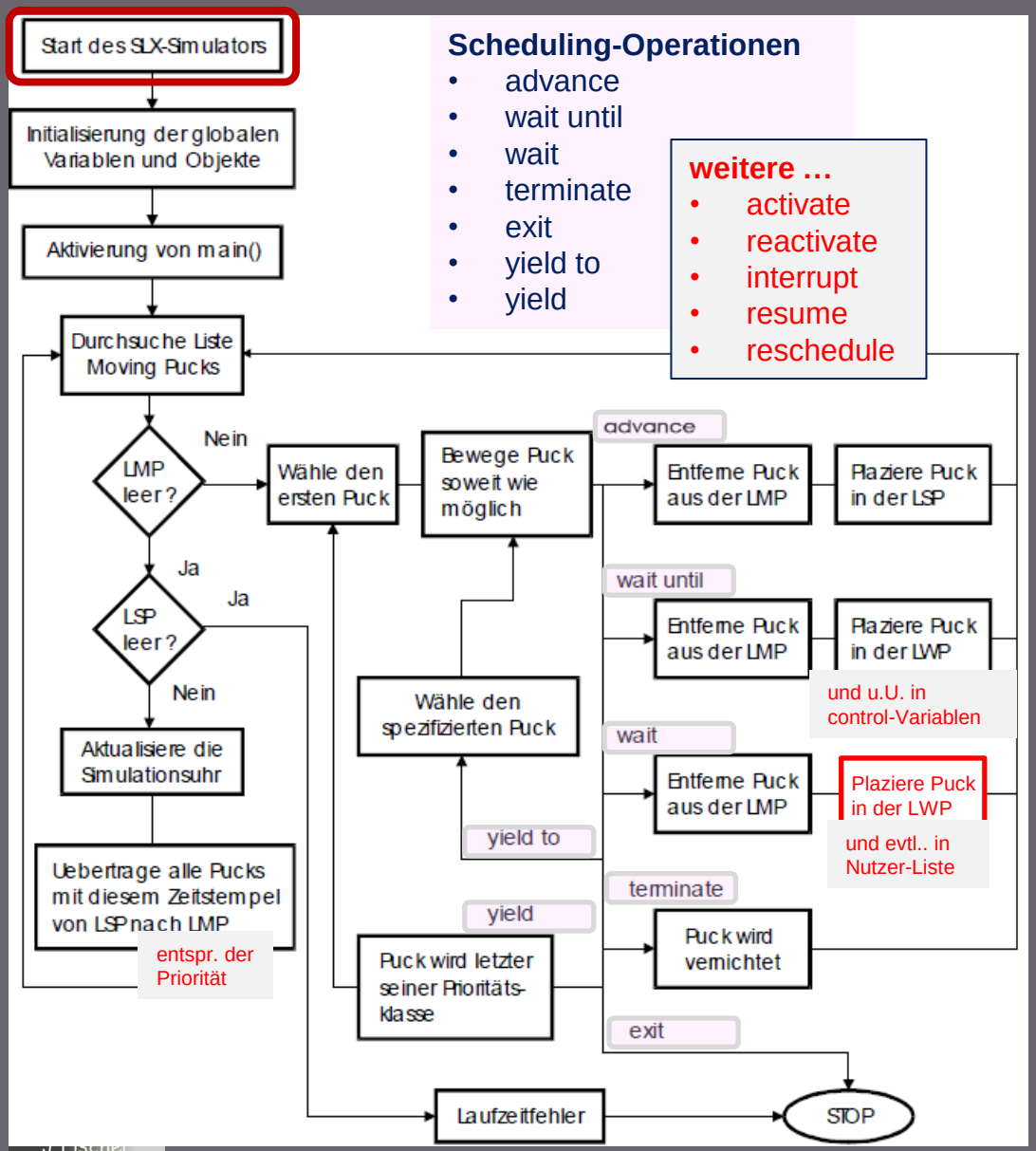
Scheduling-Operationen

- advance
- wait until
- wait
- terminate
- exit
- yield to
- yield

und weitere …

Achtung:
Schema aus „Simlation neeeds SLX“ wurde ergänzt !

Kern des Simulationslaufzeitssystems



Scheduling-Operationen

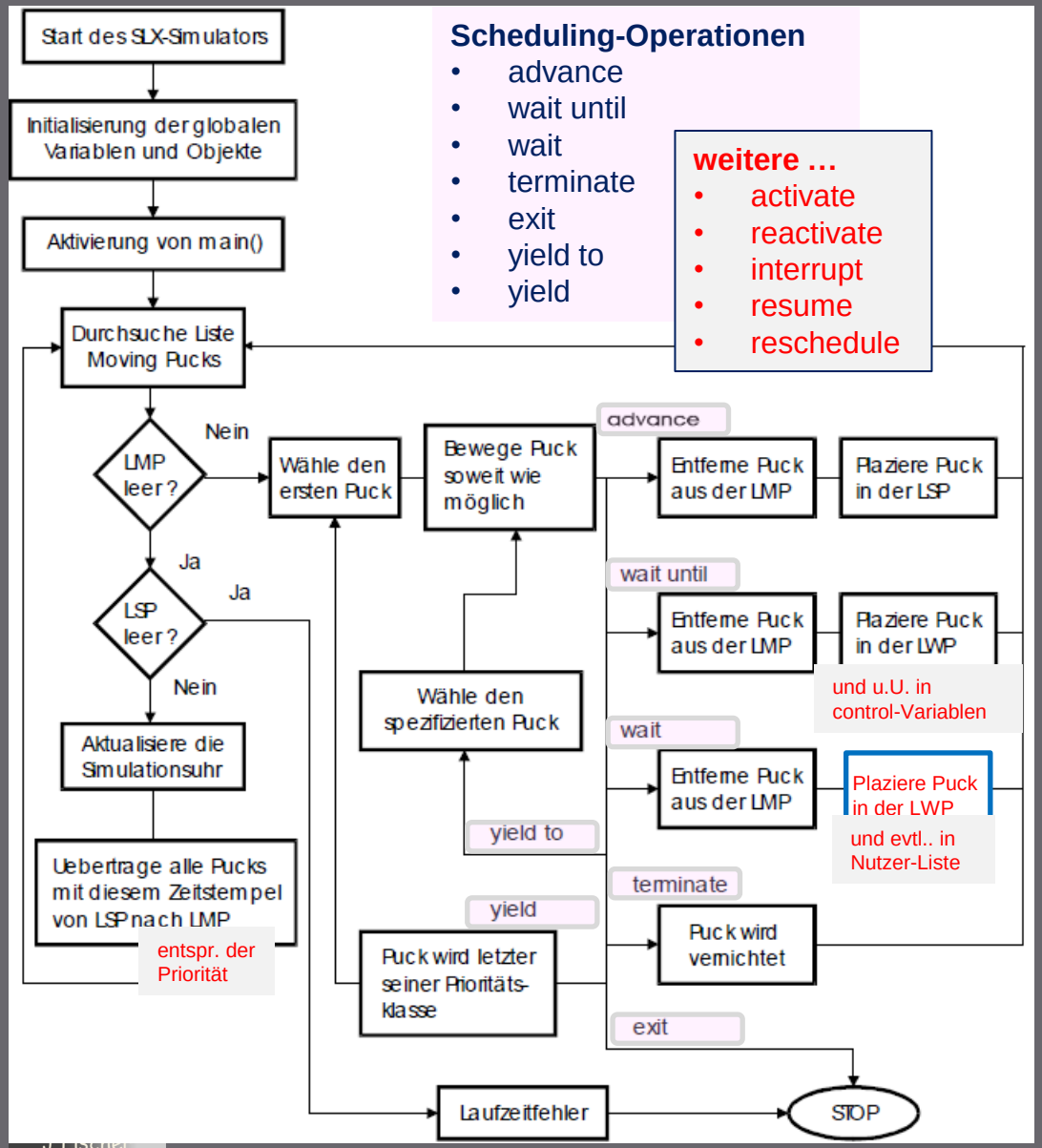
- advance
- wait until
- wait
- terminate
- exit
- yield to
- yield

weitere …

- activate
- reactivate
- interrupt
- resume
- reschedule

Achtung:
Schema aus „Simlation neeeds SLX“ wurde ergänzt !

Kern des Simulationslaufzeitssystems



Scheduling-Operationen

- advance
- wait until
- wait
- terminate
- exit
- yield to
- yield

weitere …

- activate
- reactivate
- interrupt
- resume
- reschedule

Aufgaben und häufige Struktur des Hauptprogramm:

- die Konfiguration des Systems anzustoßen
- und sich zu suspendieren/ oder zu blockieren
- bei finaler Aktivierung des Hauptprogramms werden Reportausgaben angestoßen
- bei Erreichen des Endes des Hauptprogramms (exit) bricht die Simulation ab

Zwischenfazit (1)

- vom SLX-Laufzeitsystem kann zu einem Zeitpunkt nur ein Puck verarbeitet werden

- das Problem der zeitlichen Parallelität von Abläufen in einem Simulationsmodell wird durch die Sequentialisierung der Puck-Verarbeitung gelöst

  dafür dienen die Scheduling- und andere Laufzeitlisten von Pucks

- der Modellierer kann die sequentielle Verarbeitung gleichzeitiger (im Sinne von Modellzeitgleichheit) Puck-Aktivitäten beeinflussen durch:
  - statische/dynamische Prioritätsvergabe und mittels **yield**
  - gezielte Auswahl des nächsten zu verarbeitenden Pucks mittels **yield to**

Zwischenfazit (2)

- Pucks sind in Liste MP nach Prioritätsklassen absteigend sortiert
- Die Interne Steuerung wählt immer den 1.Eintrag von MP (höchste Priorität) bei einer erneuten Prozessaktivierung
- Der aktuelle Prozess (verbunden mit ACTIVE puck) kann durch seine action-Realisierung neue Einträge in MP vornehmen,
  wobei er dadurch seinen ersten Platz in MP verlieren kann
  dies bedeutet jedoch nicht zwingend einen Steuerungswechsel

Position

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

- **C.1**
  Einführung und Ba
- **C.2**
  Stochastische Pro
- **C.3**
  Vertiefung der SL
- **C.4**
  GPSS-Elemente
- **C.5**
  DISCO-Elemente
- **C.6**
  Basissprache (Erg

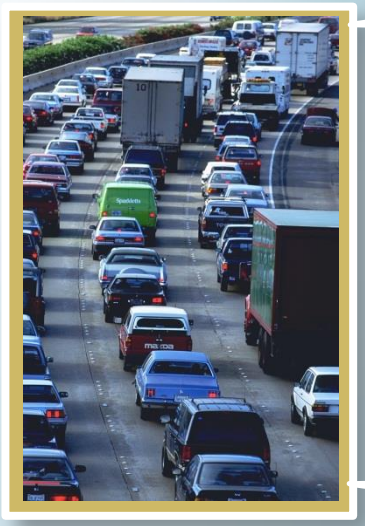
1. SLX-Überblick
2. Syntaxkonventionen
3. Elementare Datentypen
4. Klassen
5. Objekte
6. Typ Set
7. Anweisungen, Prozeduren
8. Einfache Ausgabe
9. Modellierungselemente in SLX (allgemein)
10. Prozesse und Pucks
11. Beispiel (Drucker, Druckjobs):
12. SLX- Laufzeitsystem (Simulator-Kern)
13. Zeitkonzepte
14. Behandlung von Zeit- und Zustandsereignissen

Zeitkonzepte

reale Welt

abstrakte (Modell-)Welt

ausführbares Programm



Fahrspur
Bahn
...

Fahrzeug
(Typ,
Ausstattung,
Beschaffenheit,
Position /
Positionsänderung,
Fahrweise)

Struktur-baustein
passive class

Struktur-/ Verhaltens-baustein
active class

Realzeit
(reale Uhr)

Modellzeit
(streng monton wachsende reelle Variable)

(1) Modell- oder Simulationszeit (monton wachsende Variable)
(2) Ausführungszeit (reale Uhr)

Sonderfall einer **Echtzeitsimulation**: Realzeit gleich Ausführungszeit der Simulation
(allgemeiner: echtzeitproportionales Verhalten)

Zeitbestimmung

Prozeduren zur Zeitbestimmung (float/double-Wert)

(1) **time** (Schlüsselwort) liefert moving_time der Pucks in der LMP

(2) **next_immenent_time()**
liefert moving_time des ersten Pucks in der LSP

(2) **real_time()**
liefert Zeit des Betriebssystems in s (seit Programmstart)

Position

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

- **C.1**
  Einführung und Ba
- **C.2**
  Stochastische Pro
- **C.3**
  Vertiefung der SL
- **C.4**
  GPSS-Elemente
- **C.5**
  DISCO-Elemente
- **C.6**
  Basissprache (Erg

1. SLX-Überblick
2. Syntaxkonventionen
3. Elementare Datentypen
4. Klassen
5. Objekte
6. Typ Set
7. Anweisungen, Prozeduren
8. Einfache Ausgabe
9. Modellierungselemente in SLX (allgemein)
10. Prozesse und Pucks
11. Beispiel (Drucker, Druckjobs):
12. SLX- Laufzeitsystem (Simulator-Kern)
13. Zeitkonzepte
14. Scheduling-Operationen

Prozesszustände und Übergänge

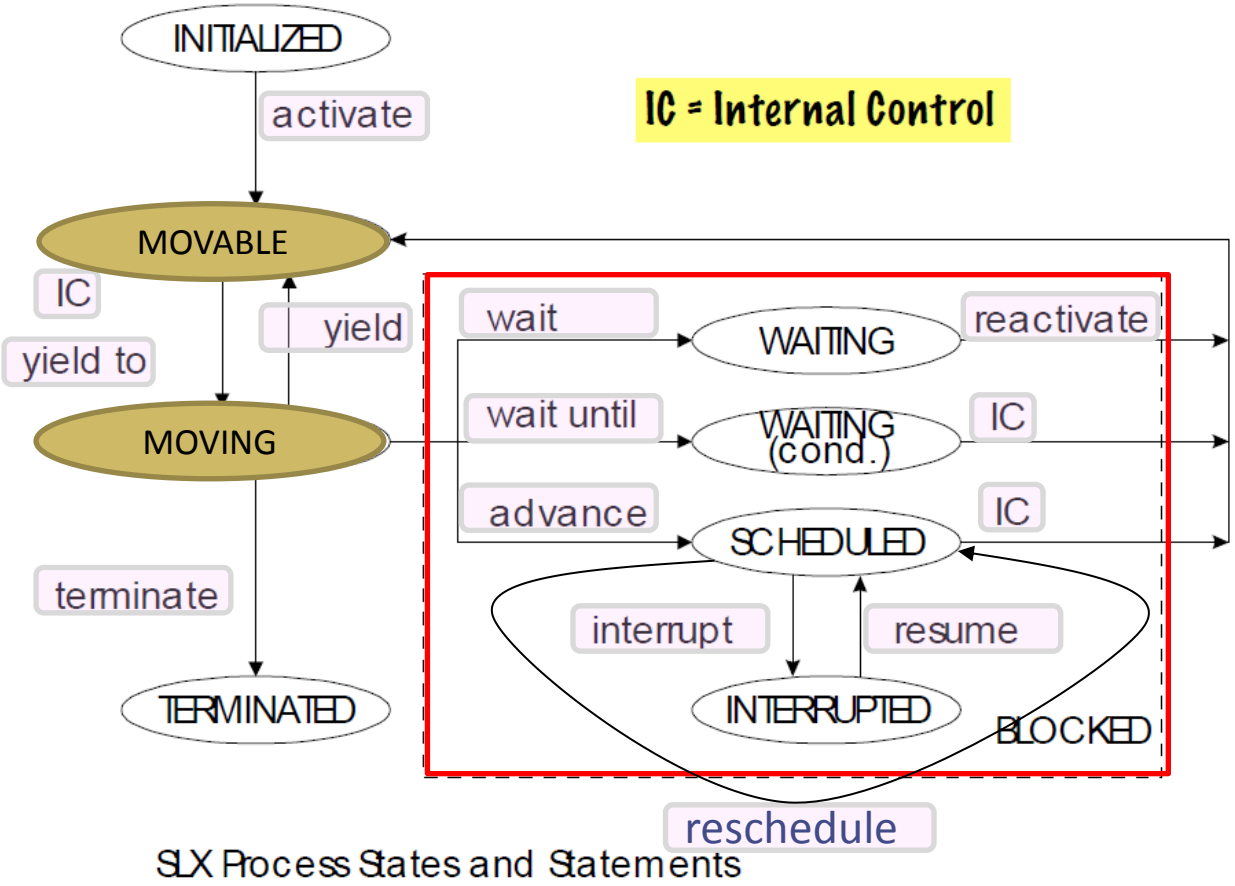
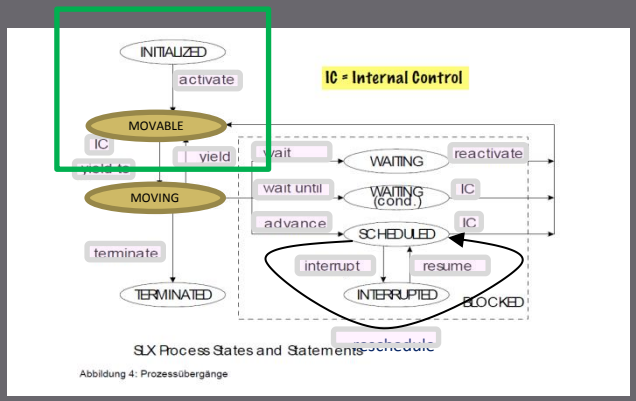


Abbildung 4: Prozessübergänge

Achtung:
Schema aus „Simlation neeeds SLX“ wurde ergänzt !

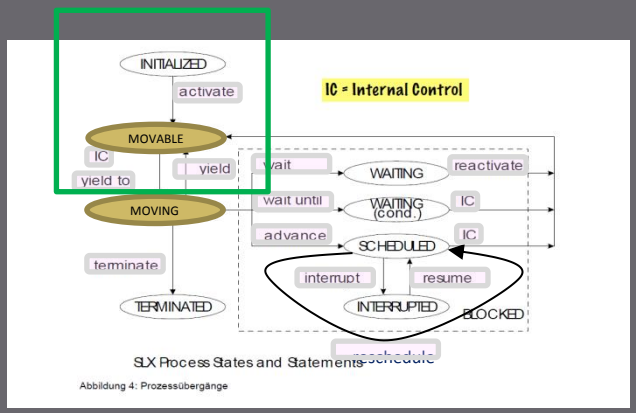
Scheduling-Operation (1): *activate, yield_to*



INITIALZED → MOVABLE

activate new process_class (…);
Zeitpunkt der Aktivierung ist aktuelle Modellzeit

Eintrag in List MP entspr. Priorität,
aber **kein** Steuerungswechsel

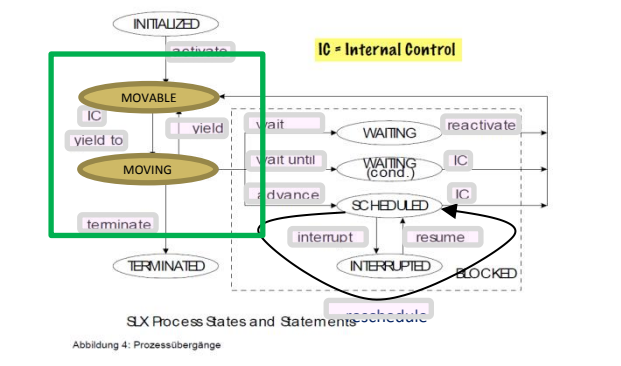


READY→ ACTIVE

(1) **Interne Steue**rung (IC) wählt den ersten
MP-Eintrag als aktiven Prozess
garantierter Steuerungswechsel

(2) **yield_to** *puck-pointer*
garantierter Steuerungswechsel (erzwungen)
ohne Positionsberücksichtigung in LMP

Scheduling-Operation (2): *yield*



SLX Process States and Statements

Abbildung 4: Prozessübergänge

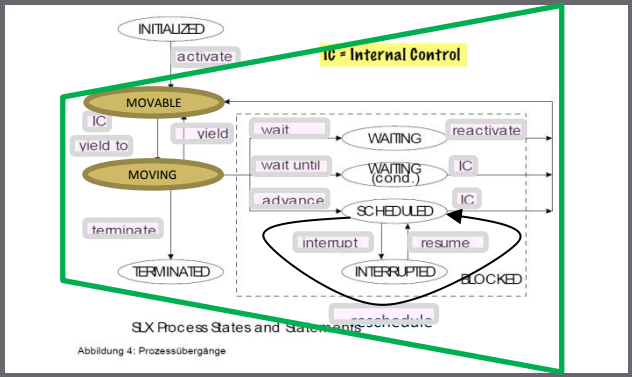
ACTIVE → READY

yield

Steuerung geht an den 1.Eintrag von LMP

evtl. Steuerungswechsel (erzwungen)

Scheduling-Operation (3): *advance*



RÜCKKEHR: SCHEDULED → MOVABLE/MOVING

IC: LMP wird leer und Ereigniszeit
der Pucks ergeben Minimum (aller Pucks in SP)

MOVING → SCHEDULED

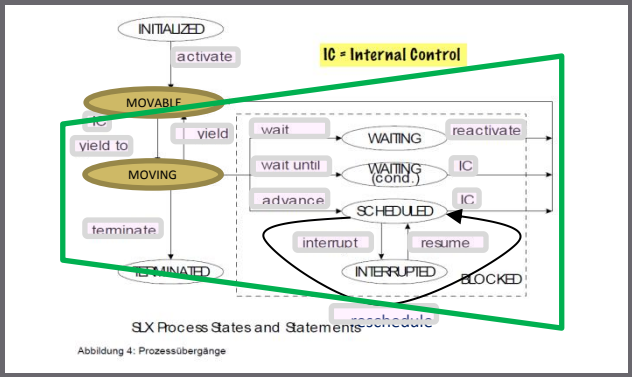
advance *delay-expression*; //Zeitverschiebung

Prozess des aktiven Pucks (Aufrufer von **advance**, …) wird unterbrochen und in LSP entspr. der neuen Ereigniszeit einsortiert

evtl. Steuerungswechsel

delay-expression vom Typ **float** oder **double** muss positiv sein
(negativ nicht erlaubt, Null=leere Anweisung)

Scheduling-Operation (4): *wait until*



Bedingung über globale Zustandsgrößen und Attribute referenzierbarer Objekte

ACHTUNG: mindestens ein Operand muss als Control-Größe vereinbart sein, damit
- Reaktivierung möglich und
- Deadlock verhindert werden kann

MOVING→ BLOCKED: WAITING(cond)

wait until *boolean-Ausdruck*; //bedingte Blockierung

Prozess des aktiven Pucks (Aufrufer von **wait until**) wird unterbrochen, falls Bedingung **nicht** erfüllt ist und wird
(1) in LWP als letzter und
(2) in jeweils die Puck-Liste aller im boolean-Ausdruck vorkommenden Control-Variablen eingetragen
(3) Steuerungswechsel

sonst Fortsetzung

Scheduling-Operationen (5)

- Anweisung

wait until (*condition* **)**

Boolscher Ausdruck
- globale Größen
- Attribute von Objekten aktiver/ passiver Klassen

ACHTUNG
mindestens eine Variable muss mit Präfix **control** definiert worden sein

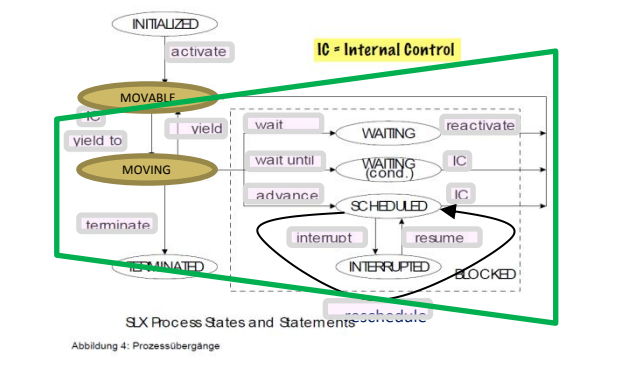
- Control-Variable

- *Jede Control-Variable verfügt über eine Puck-Liste*
- *Erfasst werden Pucks von Prozessen, deren Bedingungen von dieser Variable abhängen*
- *Bei jeder tatsächlichen Wertänderung wird zeitgleich geprüft, ob erfasste blockierte Prozesse jetzt eventuell fortgesetzt können*

ACHTUNG: Sonderfälle
a) bei Verwendung von **time** im Ausdruck
b) Bei Aktualisierung der Control-Variable durch **Eingabe**
c) Beachtung weiterer **Einschränkungen**

später

Scheduling-Operation (6)



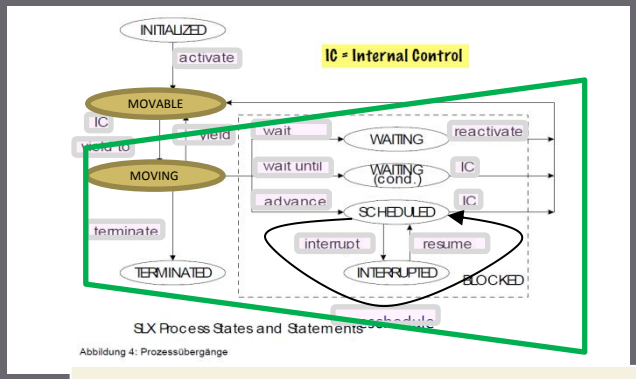
RÜCKKEHR: **WAITING(con) → MOVABLE**
Änderung mindestens einer Control-Variablen

puck_ptr wird in LWP und in der entspr.
Puck-Liste der Control-Variablen-Listen gestrichen
und in LMP einsortiert

kein Steuerungswechsel

Neuauswertung des Ausdrucks erfolgt somit erst,
*wenn **alle** potentiellen Änderungen zu diesem Zeitpunkt durch*
Prozesse mit Pucks in der LMP ausgeführt worden sind

Scheduling-Operation (7)



MOVING → BLOCKED: **WAITING**

wait; //unbedingte Blockierung

Prozess des aktiven Pucks (Aufrufer von **wait**) wird unterbrochen und sein Puck wird in Liste LWP als letzter eingetragen

garantierter Steuerungswechsel

RÜCKKEHR: **WAITING** → **MOVABLE**

reactivate puck_ptr; // puck_ptr wird in LMP einsortiert

kein Steuerungswechsel

Scheduling-Operation (8)

- unterbrochener Prozess kann sich nicht selbst aktivieren

```
class P1 {
   pointer (puck) myPuck;
   actions {
              myPuck= active;
              …
              wait;
                … //Fortsetzung bei reactivate
```

```
class P2 (pointer (P1) ptr) {
    pointer (P1) partner;
    initial {
                partner= P1;
    }
    actions {
              …
              reactivate partner->myPuck;
                …
}
```

Scheduling-Operation (9) *wait* mit *list*-Option

Prozess des aktiven Pucks (Aufrufer von **wait**) wird unterbrochen und sein Puck wird in Liste LWP als letzter eingetragen

wait list = *list_ident* **;**

*list_ident***:** Bezeichner der Liste

Puck wird zusätzlich in benutzerdefinierte Liste eingeordnet

Pointer (puck) list_ident ; muss vom Nutzer definiert werden
verweist auf letzten Puck der Liste

Beispiel:

pointer (puck) paperQ; // Used for wait lists
…
wait list = paperQ; // Wait in paper Queue

Verwendung dieser Option gestattet
die Reaktivierung“ der gesamten Liste

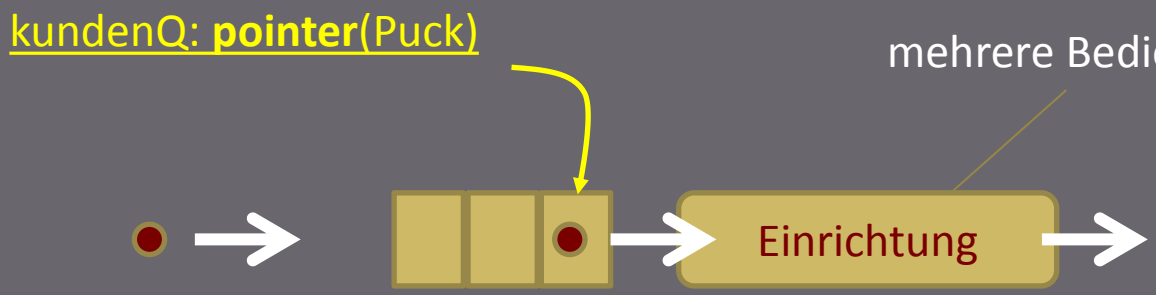
Scheduling-Operation (10) *reactivate* mit *list*-Option

reactivate list = *list_ident* **;**

reaktiviert alle „schlafenden“ Pucks innerhalb der Liste

die Pucks werden in der LMP wieder in ihrer ursprünglichen Reihenfolge eingeordnet

Typisches Modellierungsmuster



Strom von Pucks (Kunden)

pointer (puck) kundenQ;

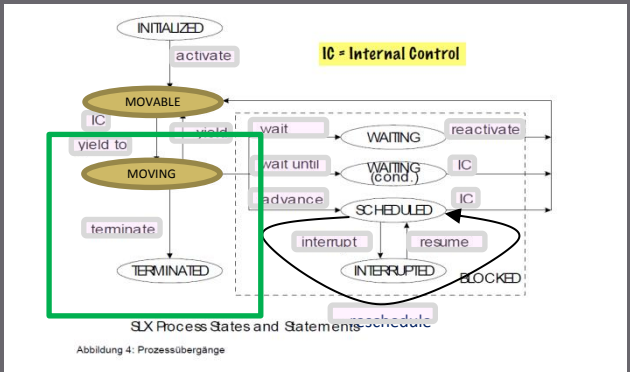
Actions von Kunde

```
while (Station->belegt)
        wait list = kundenQ;
//…Fortsetzung bei freier Station

Station->belegt=TRUE;
advance (dt);
Station->belegt= FALSE;

if not kundenQ = NULL reactivate list= kundenQ
```

Scheduling-Operation (11): *terminate*



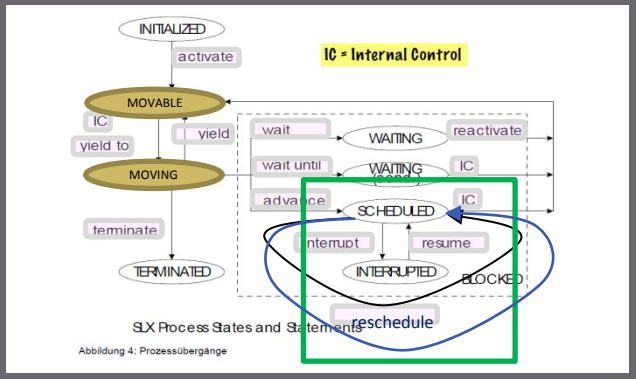
MOVING → TERMINATED

terminate;

Aufrufer von **terminate** führt letzte Prozess-Aktion aus, Löschen des aktiven Pucks

garantierter Steuerungswechsel

Scheduling-Operation (11): *reschedule*



SCHEDULED→ SCHEDULED

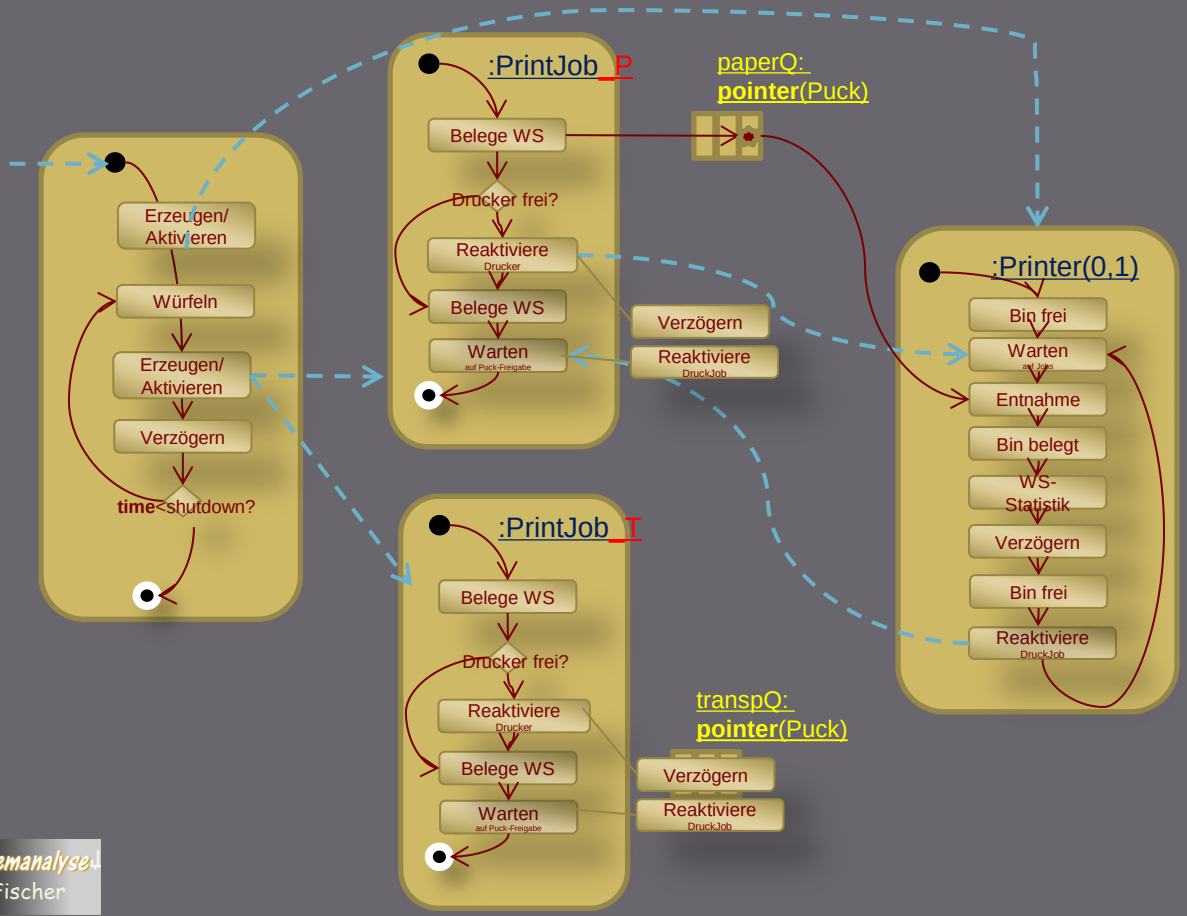
reschedule *puck_ptr* **at** *new_time*;

Puck des suspendierten Prozesses wird in LSP umsortiert

Kein Steuerungswechsel

2.Beispiel (Aufgabe, Prozess-Schema)

- zwei Arten von Printjobs (Papier- u. Folienausgabe)
- separate Erfassung



sehr flexibel
funktioniert
für mehrere
mehrere
Printer-Objekte

Vereinfachung:
nur ein Printer

bei Verzicht auf
Printer-Objekt

```
module basic {
  type t_printType enum {paper, transp };

  rn_stream arrivals, service, jType ; // (0,1)-random gen
  int shutdown_time = 5*8*60;
  control boolean printer_busy;
  t_printType printer_case = paper ; // initialization with paper
  pointer (puck) paperQ, transpQ; // used for wait lists
  int nPaper, nTransp; // Counter for waiting jobs
```

```
procedure main() {
  while (time < shutdown_time) {
    if ( frn ( jType ) >0.5 )
      activate new cl_printer_job_P ; // create new jobs
    else
      activate new cl_printer_job_T;
    advance rv_uniform( arrivals , 10.0,20.0 ) // arrival time
  }
} // main
```

```
class cl_printer_job_P  {
  t_printType  jobType;    //job type
  initial {
      nPaper++; jobType = paper;
  }

  actions {
    while ( ( printer_busy ) || ( jobType != printer_case )) {
      if ( !printer_busy )
        printer_case = jobType ; // reset the printer
      else
        wait list= paperQ; // wait in paper Queue
    }
    printer_busy = TRUE; // start printing
    nPaper--;
    advance rv_uniform ( service , 10, 18.0 ) ; // printing time
    printer_busy = FALSE; // finish printing
    if ( nPaper >0 ) // more paper printing?
      reactivate list = paperQ; //Yes
    else
      reactivate list = transpQ; // perhaps transp. printing
  }
} // cl_printer_job_P
```

```
class cl_printer_job_T {
  t_printType  jobType;
  initial {
    nTransp++; jobType = transp;
  }
  actions {
   while ( ( printer_busy ) || ( jobType != printer_case )) {
    if ( !printer_busy )
        printer_case = jobType ;
    else
      wait list= transpQ;
   }
   printer_busy = TRUE;
   nTransp--;
   advance rv_uniform ( service , 10, 18.0 ) ; // printing time
   printer_busy = FALSE;
   if ( nTransp >0 )
      reactivate list = transpQ;
   else
      reactivate list = paperQ;
  } // actions
} // cl_printer_job_P
```