Modul OMSI-2
im SoSe 2010

Objektorientierte Simulation mit ODEMx

Prof. Dr. Joachim Fischer
Dr. Klaus Ahrens
Dipl.-Inf. Ingmar Eveslage
Dipl.-Inf. Andreas Blunk

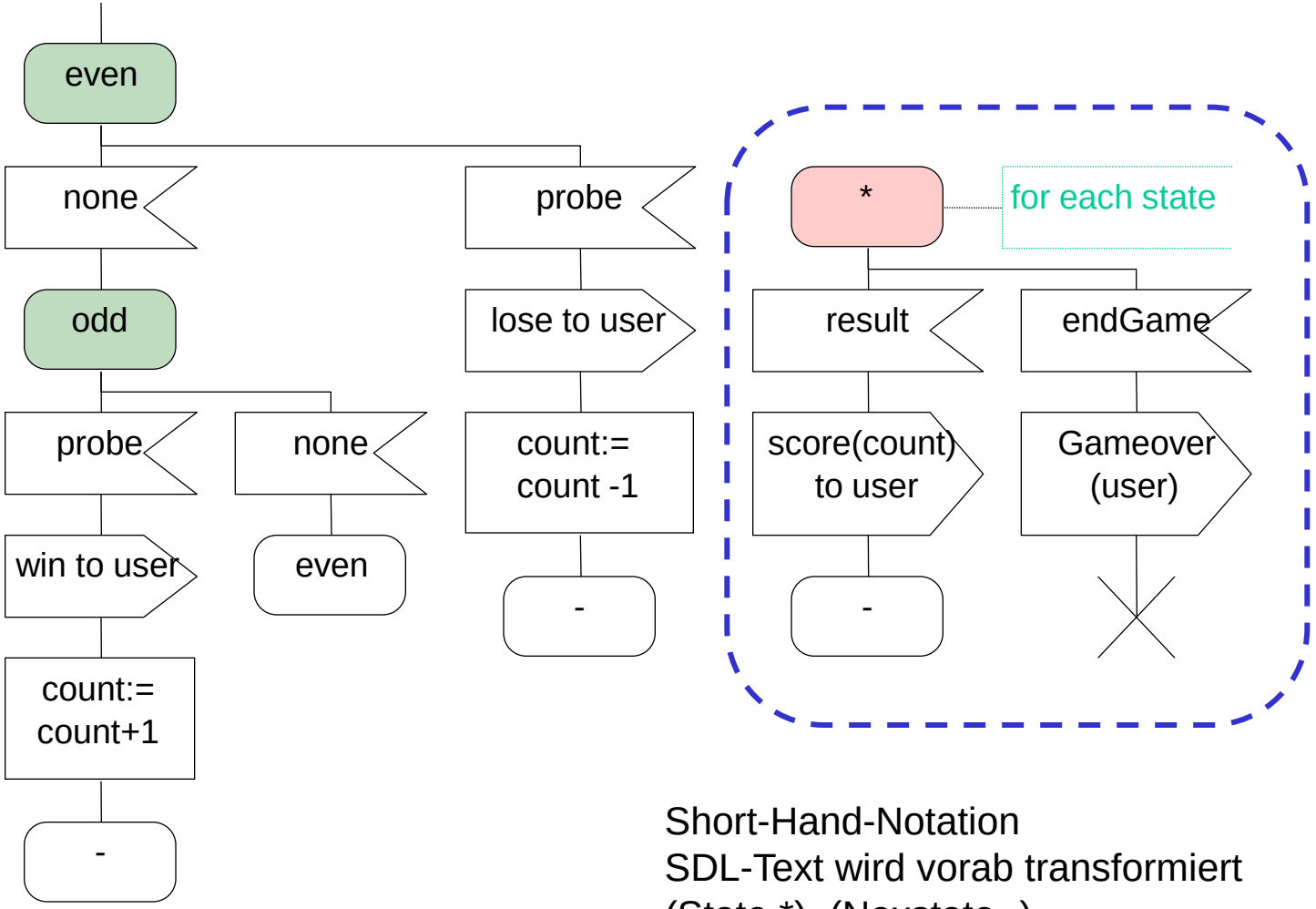
fischer|ahrens|eveslage|blunk@informatik.hu-berlin.de

9. SDL-Konzepte *(Präzisierung, 2.Teil)*

1. Ersetzungskonzept: Shorthand-Notation
2. Ersetzungskonzept: Priorisierter Input
3. Prozeduren, Ersetzungskonzept: Remote-Prozeduren
4. Spezialisierung von Zustandsautomaten
5. Lokale Objekte, Semaphore

… bereits bekannt:

Shorthand-Notationen



Short-Hand-Notation
SDL-Text wird vorab transformiert
(State *), (Nexstate -)

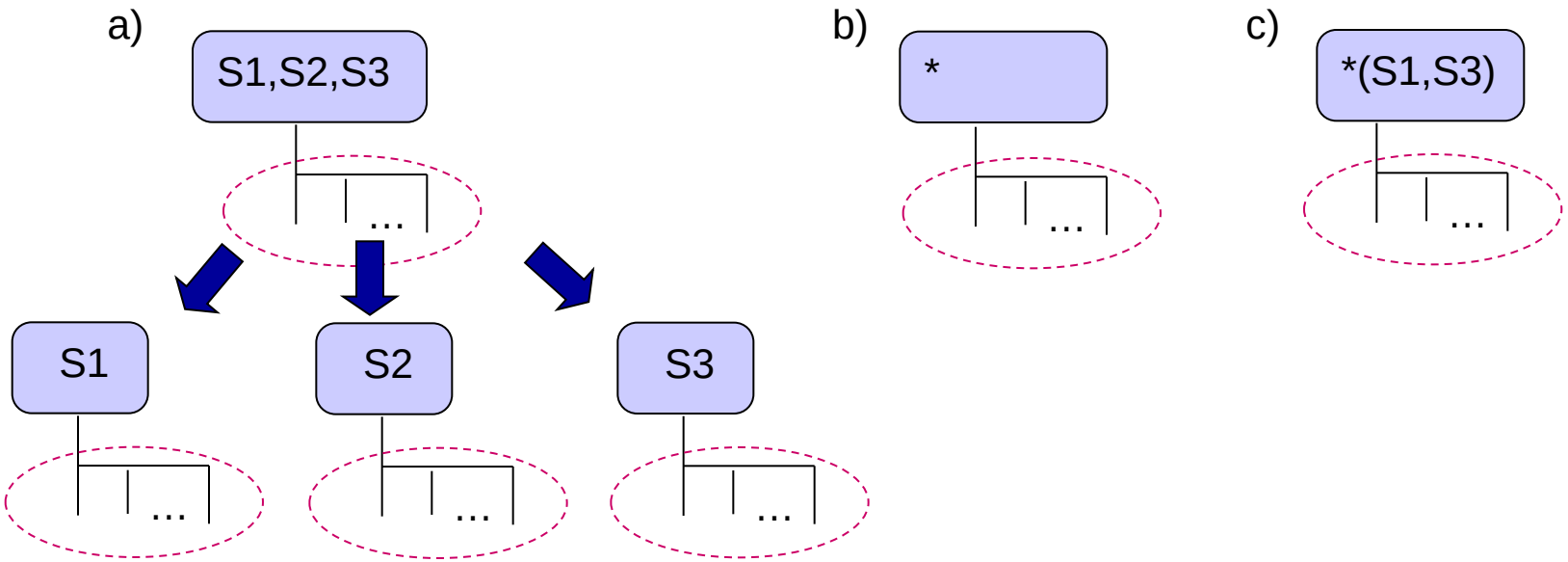
Shorthand- Notationen (Teil I)

- Allgemeines Prinzip der SDL- Semantikdefinition:

  Zurückführung komplexer Konstruktionen durch Quelltexttransformation auf einfachere Konstrukte (Basic-SDL)

- Z.100 enthält sog. Ersetzungsmodelle, die die Semantik der Konstrukte festlegen, z.B.
  - state *
  - input *
  - nextstate -
  - priority input
  - continous signal , ...

Vervielfachung von Transitionen für mehrere Zustände



Vervielfachung für

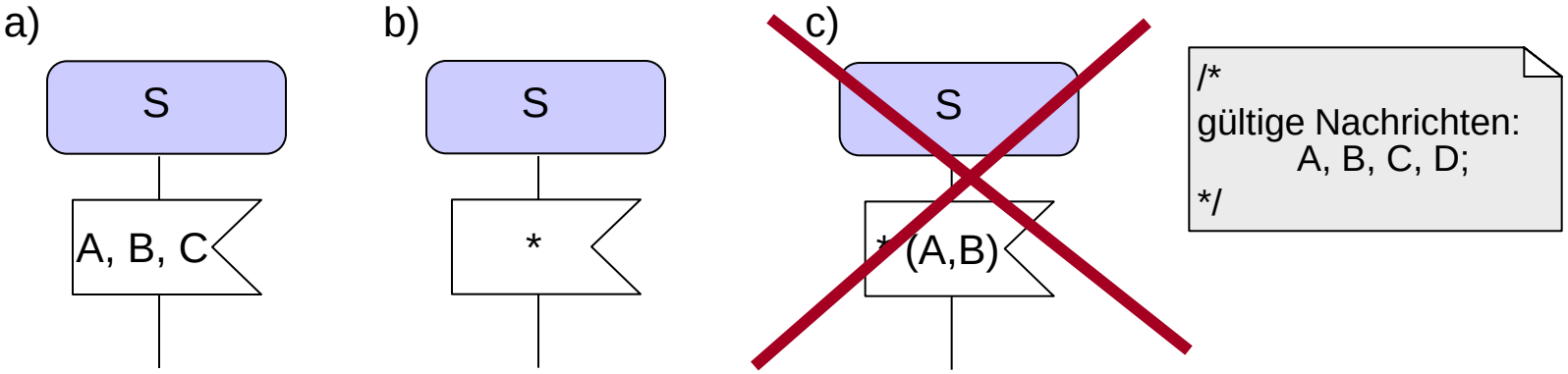
a) eine Menge von Zuständen (hier S1,S2,S3),

b) alle Zustände des jeweiligen Namensraumes:
process, (process type, service, service type), procedure)

c) alle, bis auf eine Menge von Zuständen
(hier: S1,S3)

Achtung: über alle möglichen Vererbungsstufen hinweg !!!

Vervielfachung von Transitionen für mehrere Trigger in einem Zustand



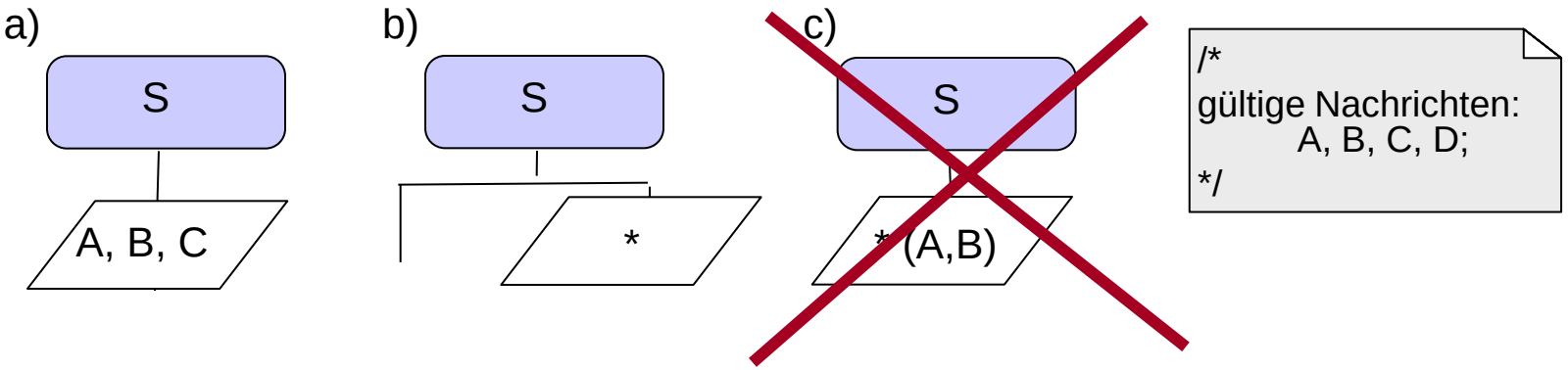
Vervielfachung für

a) eine Menge von Input-Trigger (hier A, B, C),

b) alle Trigger (bzgl. gültiger Eingabenachrichten)

Achtung: über alle möglichen Vererbungsstufen hinweg !!!

Vervielfachung von Save-Aktionen in einem Zustand



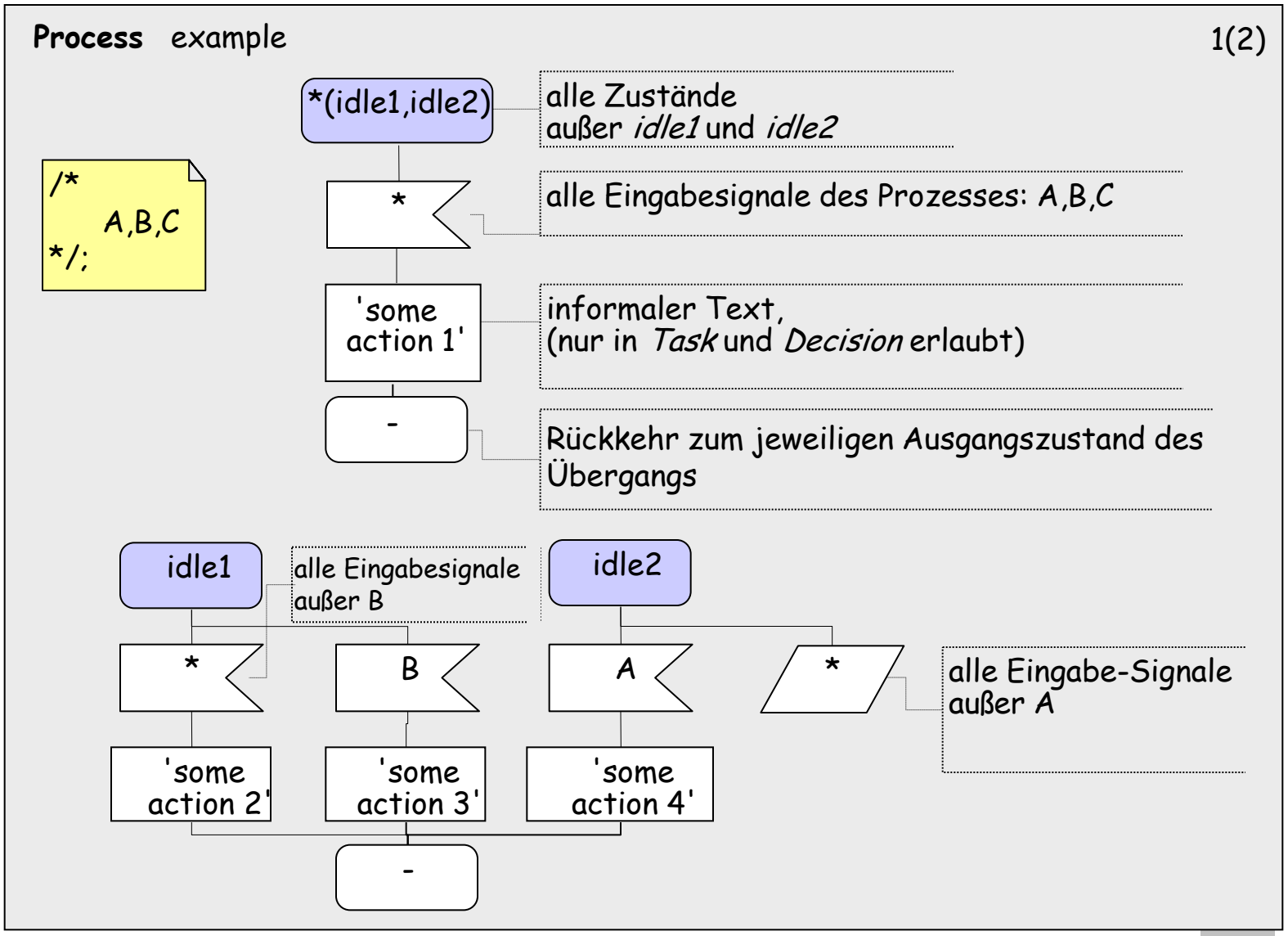
Vervielfachung

a) einer Menge von Save-Trigger (hier A, B, C),

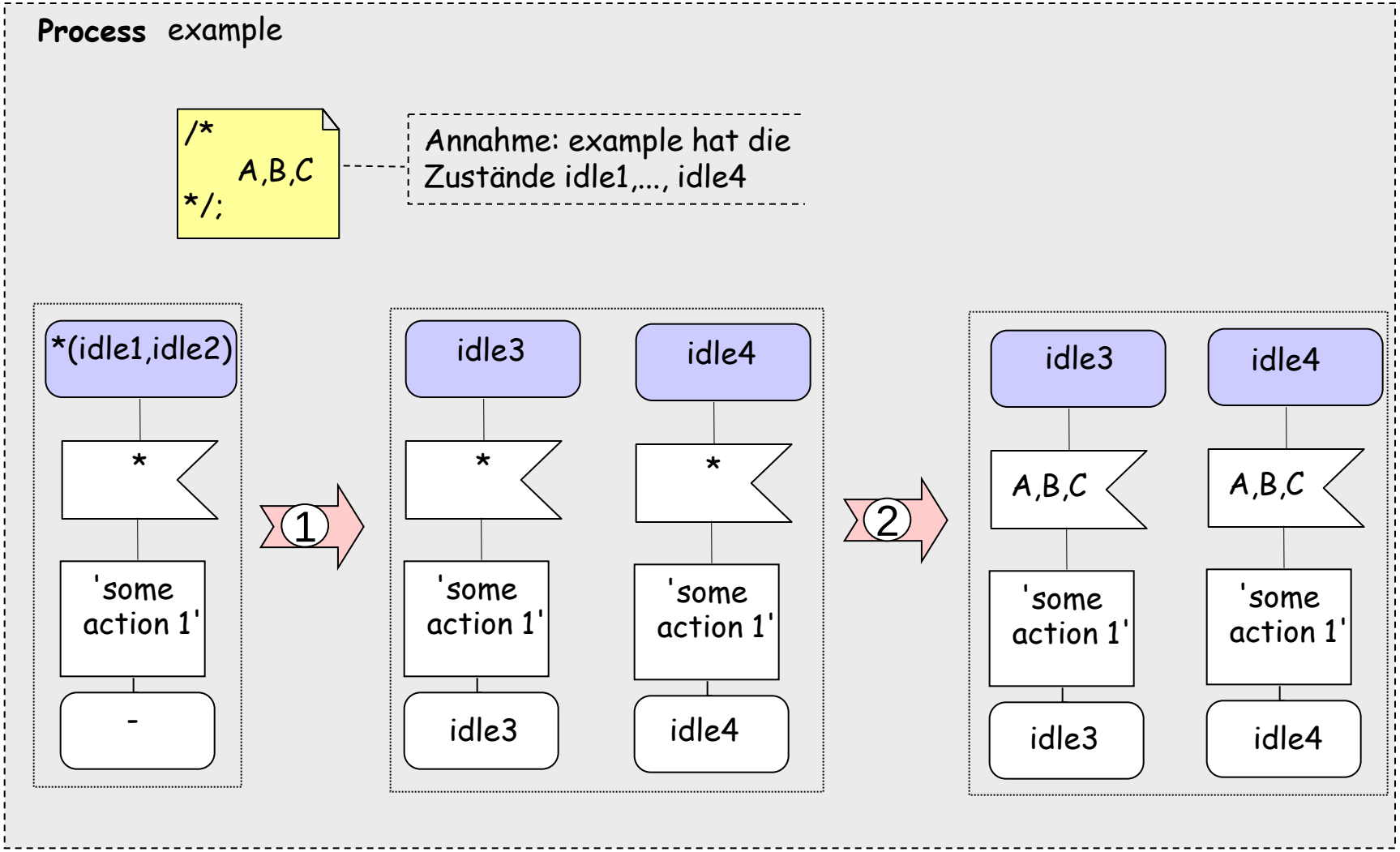
b) von Save-Triggern für alle Signale
(bzgl. gültiger Eingabenachrichten),
die nicht bereits in S mit alternativen Triggern verarbeitet werden

Achtung: über alle möglichen Vererbungsstufen hinweg !!!

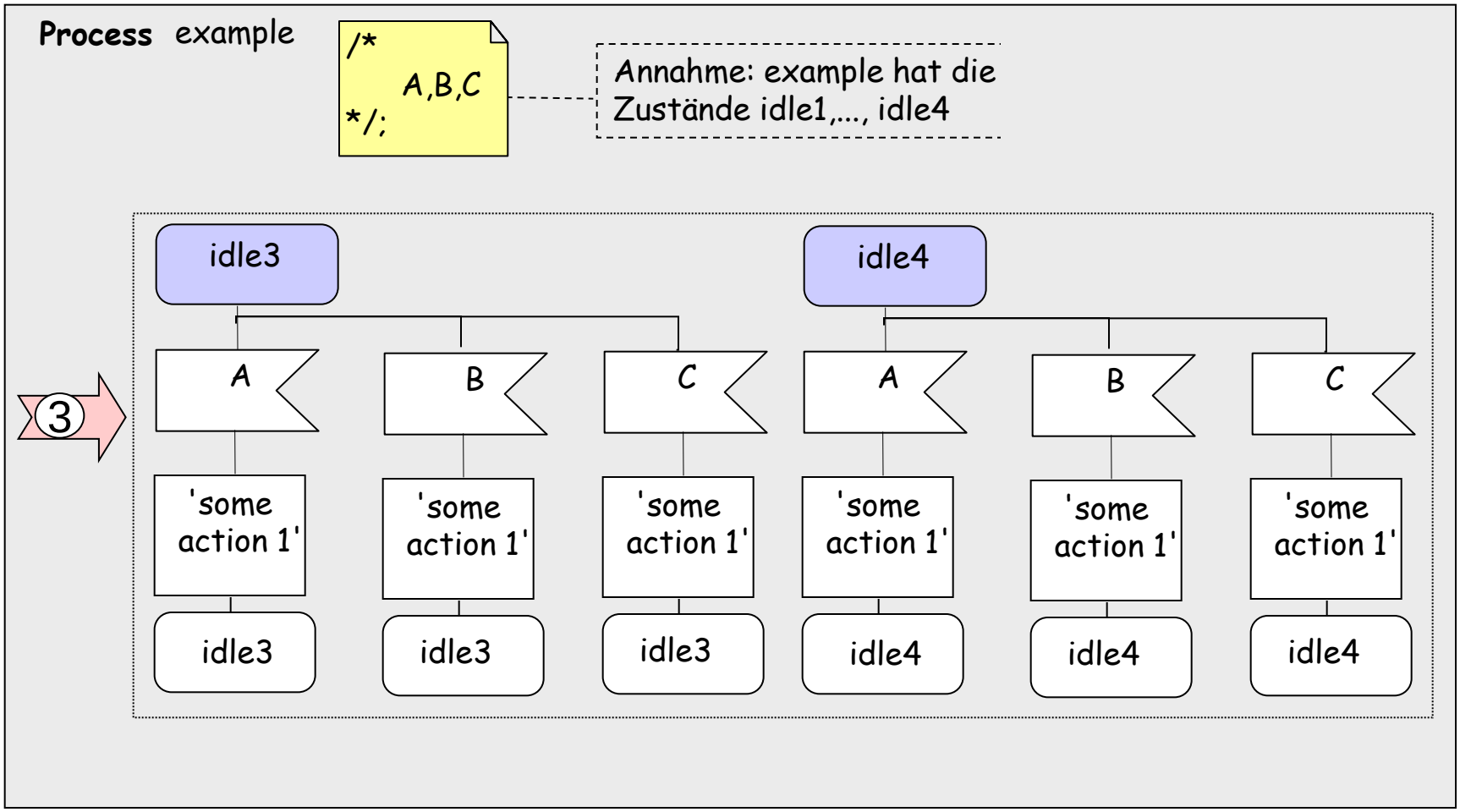
Beispiel: Anwendung von Shorthand-Notationen



Beispiel: Auflösung von Shorthand-Notationen (1)



Beispiel: Auflösung von Shorthand-Notationen (2)



Wiederholung: RTDS (SDL-Z.100)

- eingeschränkte Trigger:
  - priority input
  - (normal) input
  - continuous signal

abnehmende Priorität

- eingeschränkte Short-Hand-Notationen
  - state -, state *, state *(..,..)
  - input *, save *

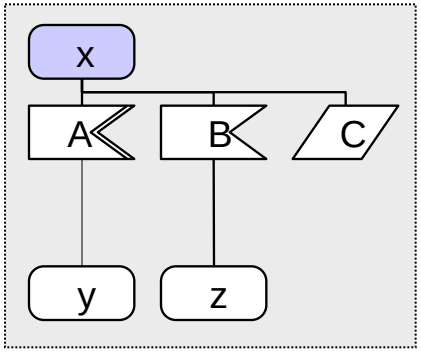
- eingeschränkte Prozeduren
  - erlaubte Dekl.-Niveaus: System/Block, Process, Procedure
  - keine Kontextparameter

9. SDL-Konzepte *(Präzisierung, 2.Teil)*

1. Ersetzungskonzept: Shorthand-Notation
2. Ersetzungskonzept: Priorisierter Input
3. Prozeduren, Ersetzungskonzept: Remote-Prozeduren
4. Spezialisierung von Zustandsautomaten
5. Lokale Objekte, Semaphore

Ersetzungsmodell für priorisierten Input (1)

Annahme (o.B.d.A.)



zu betrachtender Zustand x

Prinzip

– Aufteilung der Signal-Triggermenge (A, B, C) für den Zustand x:
  (1) für priorisierter Trigger: { A } und
  (2) den Rest: { B, C }

– Ersetzung des Zustandes x durch zwei (neue implizite) Zustände s1 und s2:

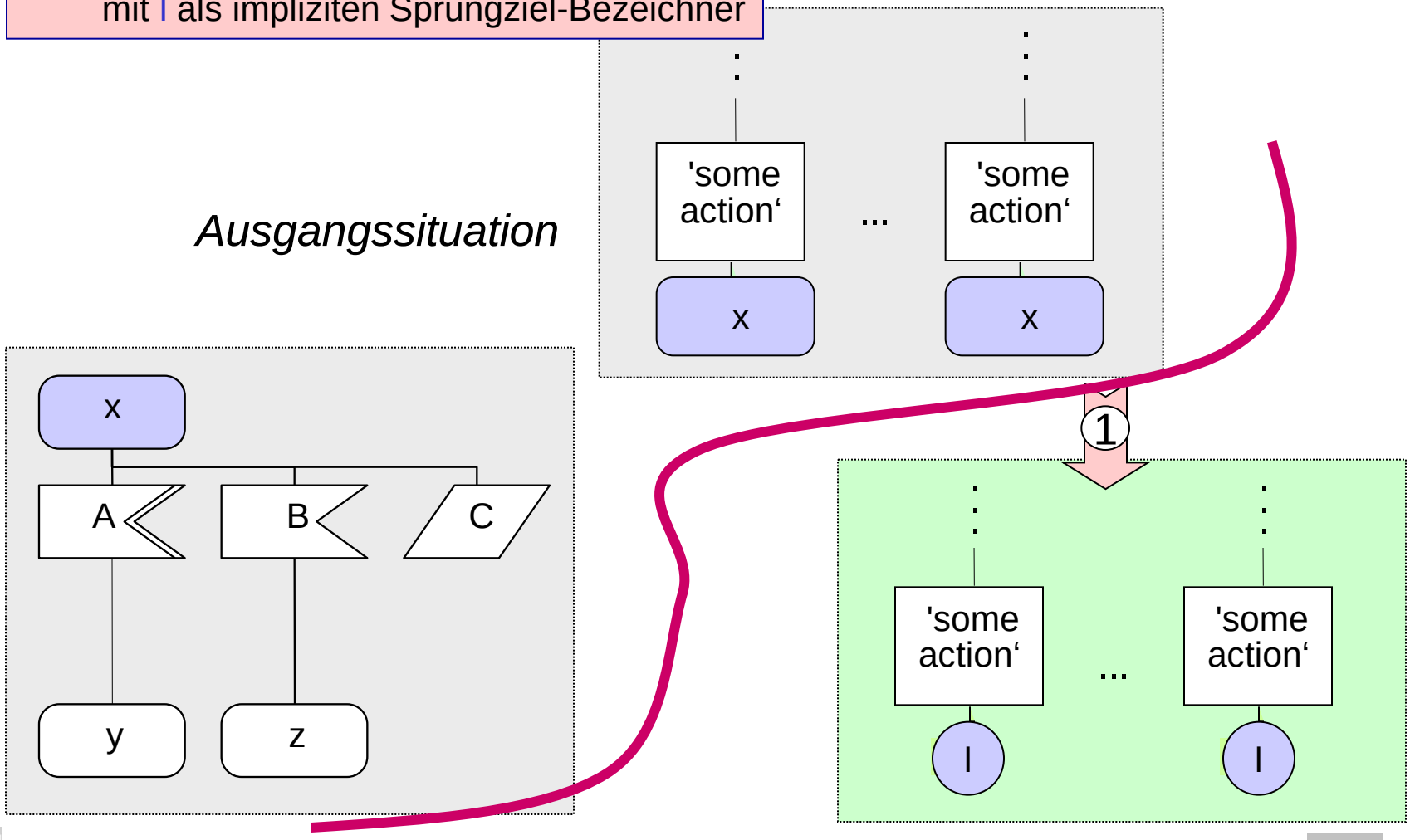
  s1: behandelt { A } als normale Inputs nach FCFS und
  stellt alle anderen ankommenden Signale mit SAVE zurück

  s2: behandelt den Rest { B,C } nach FCFS

Priorisierte Trigger müssen in SDL/RT per Hand transformiert werden !

Ersetzungsmodell für priorisierten Input (2)

1. Schritt: jedes **nextstate** x; wird zu **join** l;
mit l als impliziten Sprungziel-Bezeichner



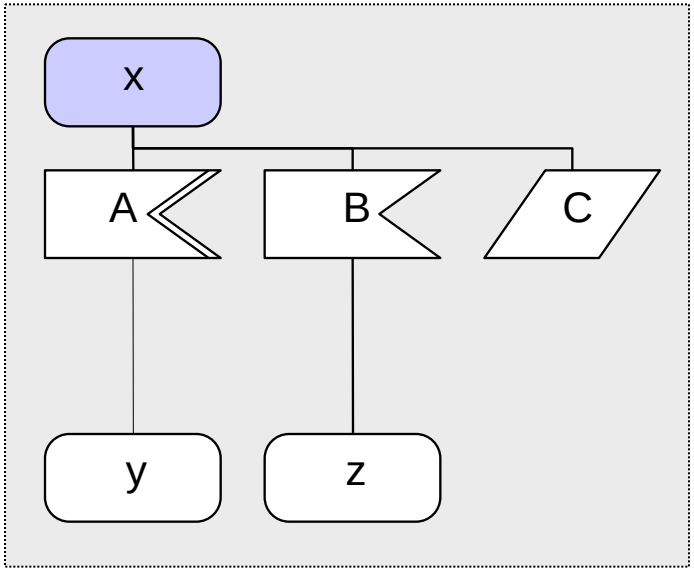
Ersetzungsmodell für priorisierten Trigger (3)

2. Schritt:
a) Einführung impliziter Variablen und Nachrichten

2a

```
int n = 0;
int newn;
```

MESSAGE Xcont (**int**);



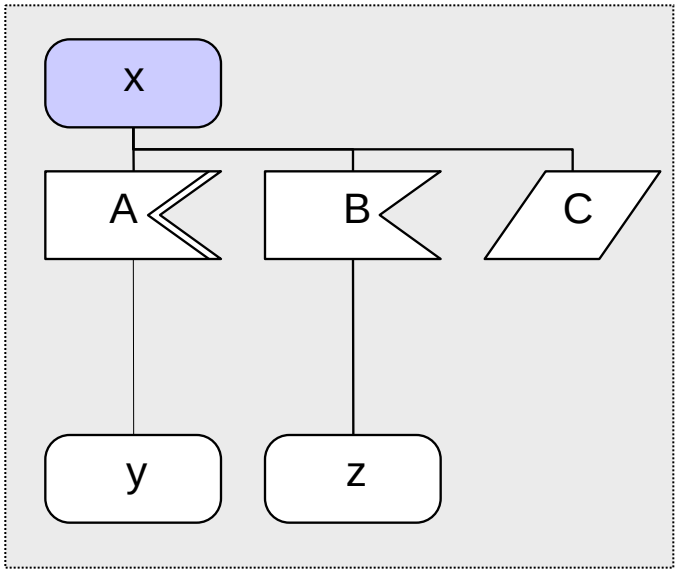
b) Einführung zweier impliziter Zustände

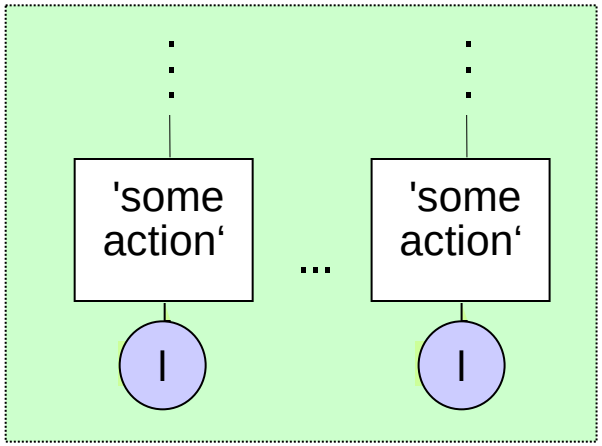
Behandlung der priorisierten Nachrichten

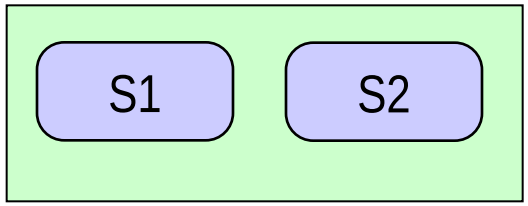
Behandlung der restlichen Nachrichten

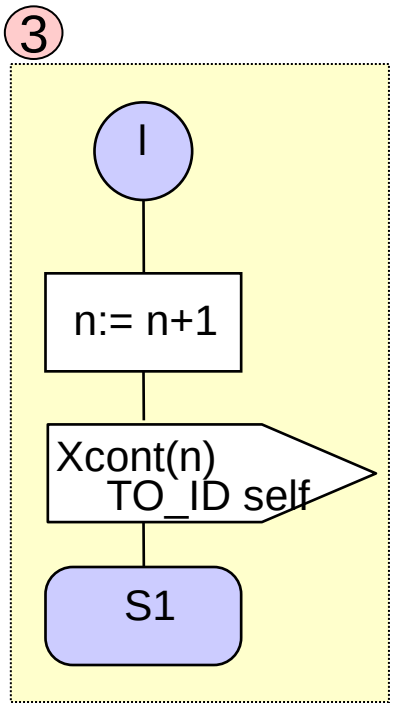
Ersetzungsmodell für priorisierten Input (4)

3.Schritt:
Übergang von der generierten Sprungmarke */* zum impliziten Zustand *S1*









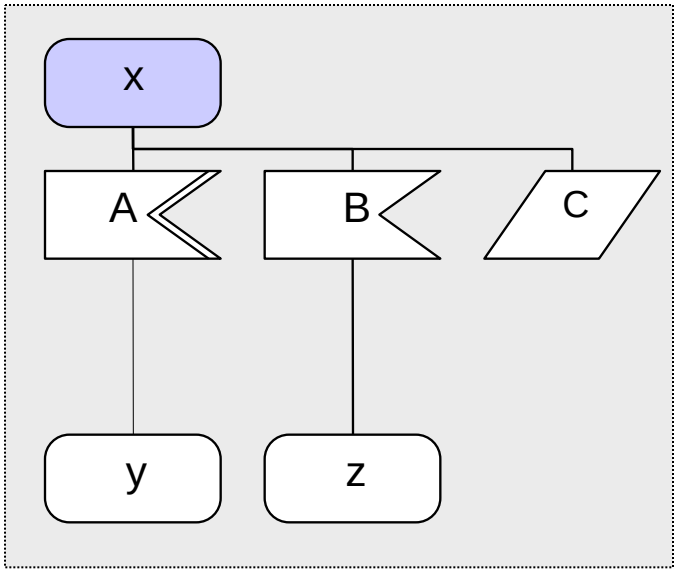

```
int n = 0;
int newn;
```

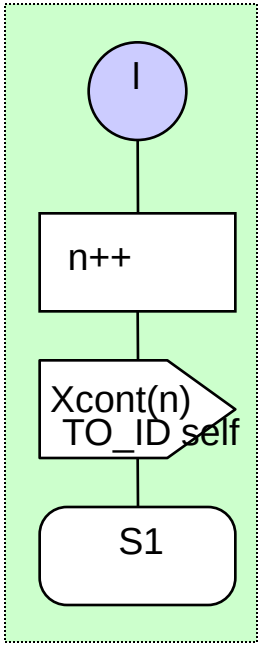
Zustandsübergänge aus S1

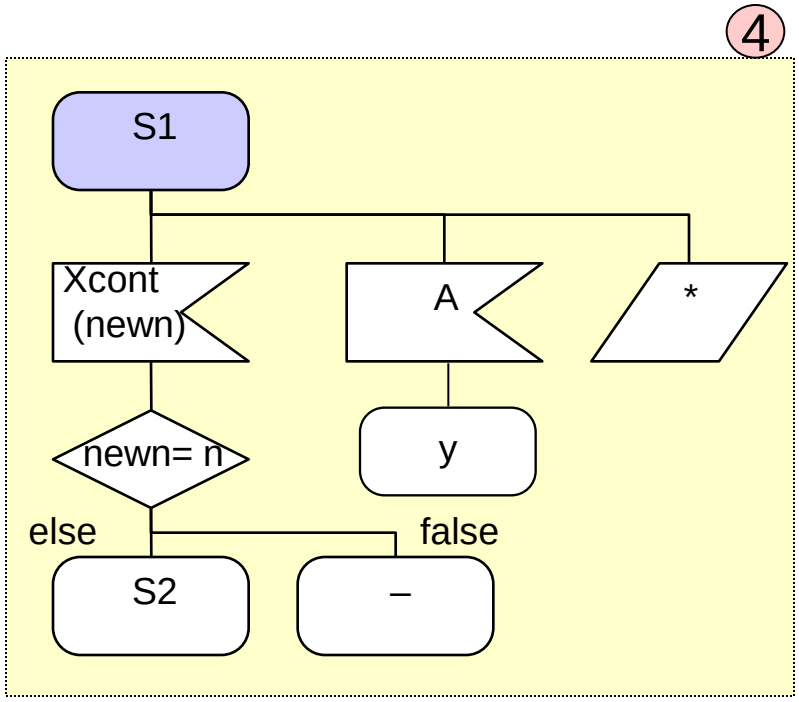
Endmarkierung der Nachrichten
des n-ten Übergangs

Ersetzungsmodell für priorisierten Input (5)

4.Schritt: Definition der Übergänge für *S1*







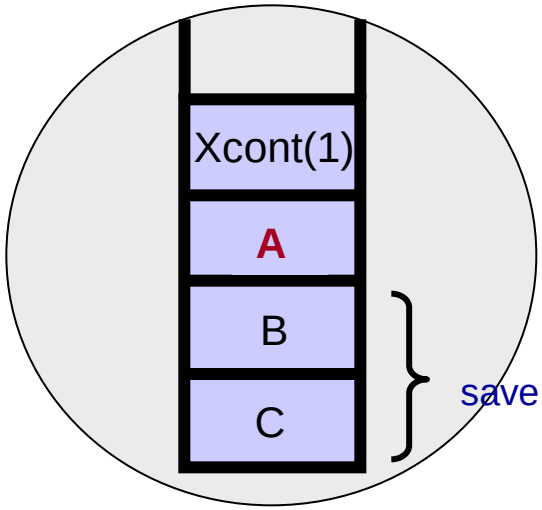
Zustandsübergange aus S1

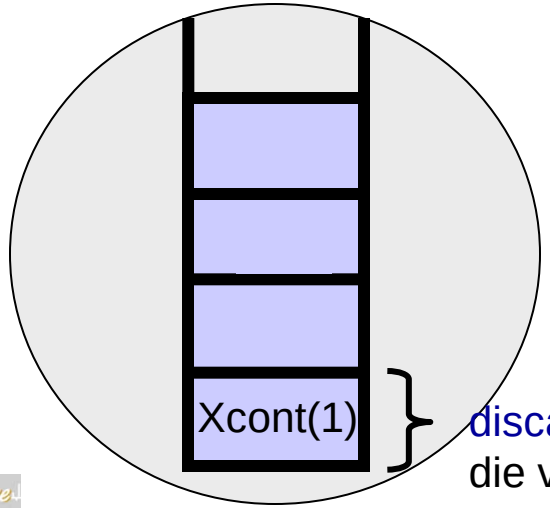
```
int n = 0;
int newn;
```

Endmarkierung der Nachrichten
des n-ten Übergangs

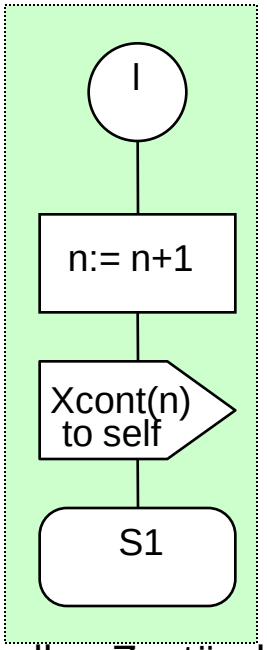
Zur Plausibilität des 4.Transformationsschrittes (1)

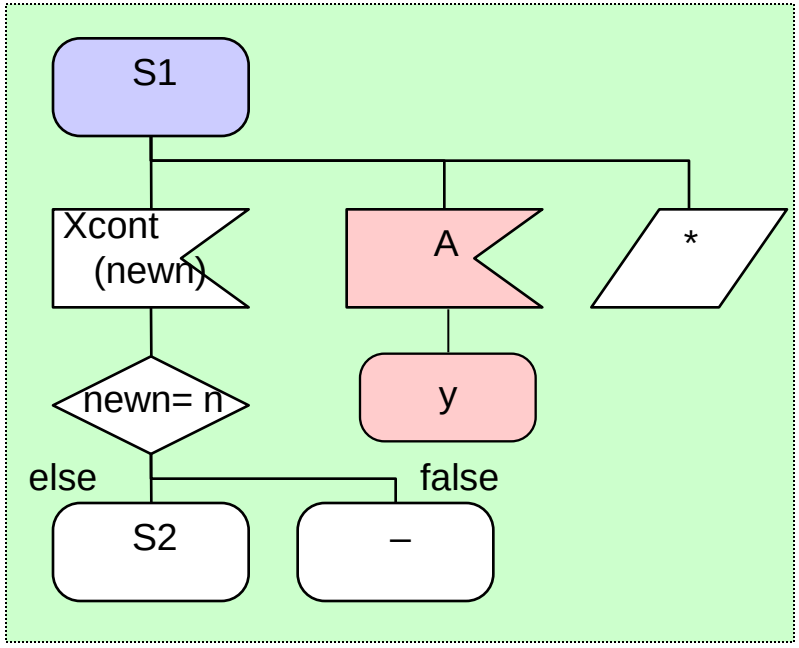
1.Szenario zur Bearbeitung der Nachrichten im Zustand S1: A **sei** vorhanden





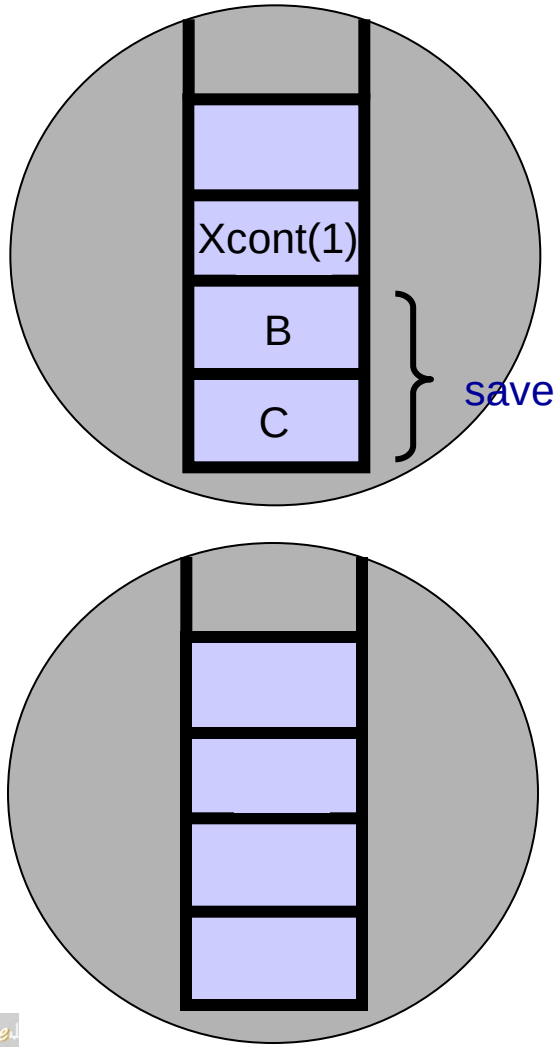
discard in allen Zuständen,
die verschieden von S1 sind





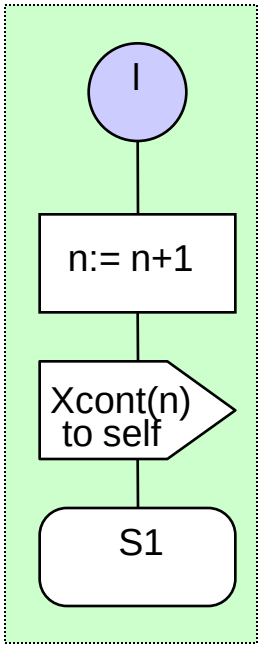
Zur Plausibilität des 4.Transformationsschrittes (2)

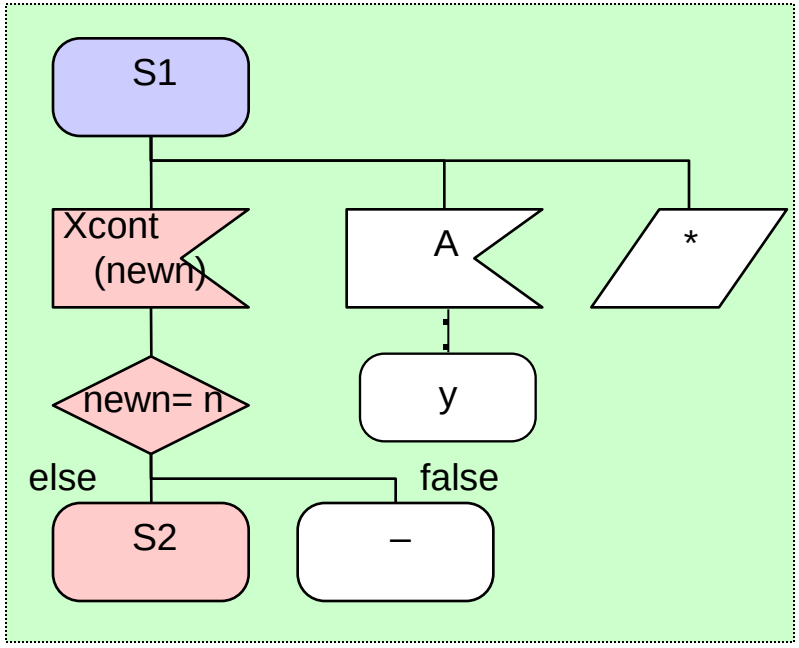
2.Szenario zur Bearbeitung der Nachrichten im Zustand S1: A **sei nicht** vorhanden



int newn 1

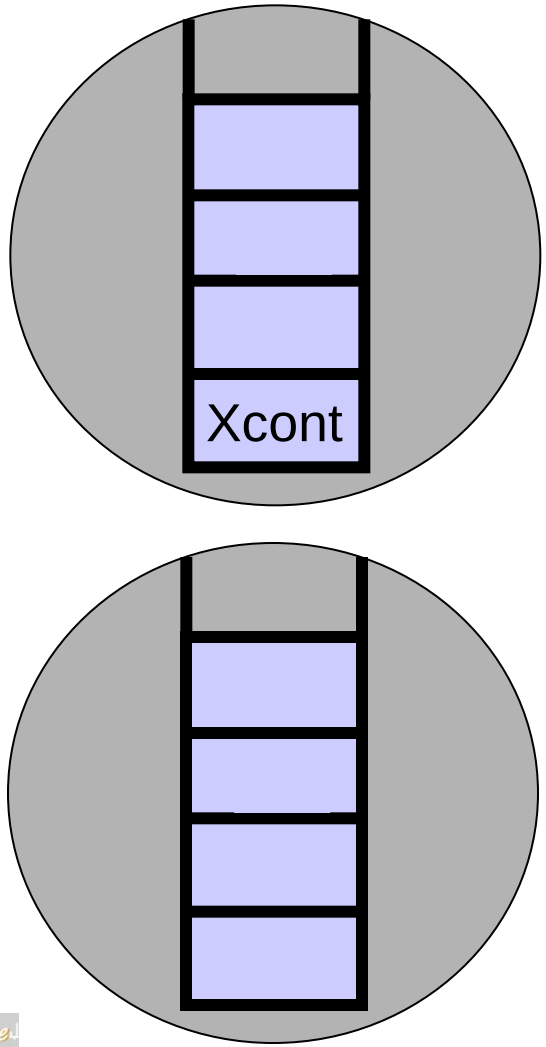
int n 1

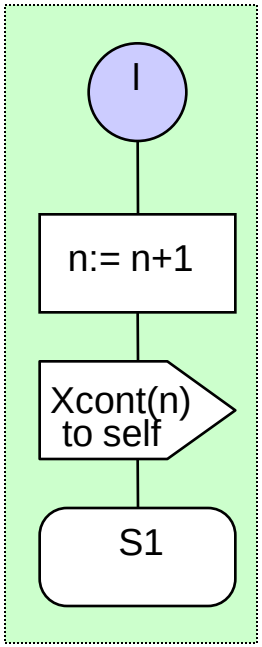


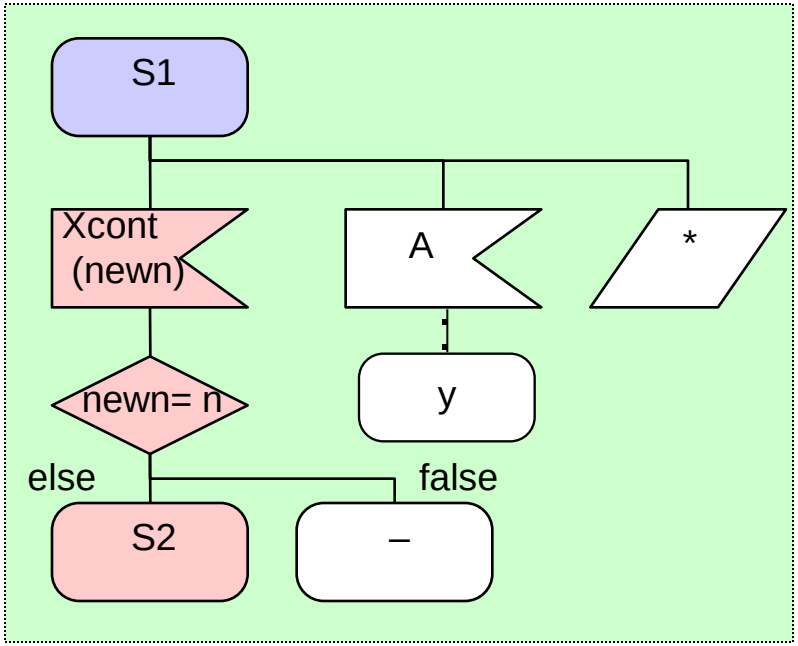


Zur Plausibilität des 4.Transformationsschrittes (3)

3.Szenario zur Bearbeitung der Signale im Zustand S1: sei „kein“ Signal vorhanden

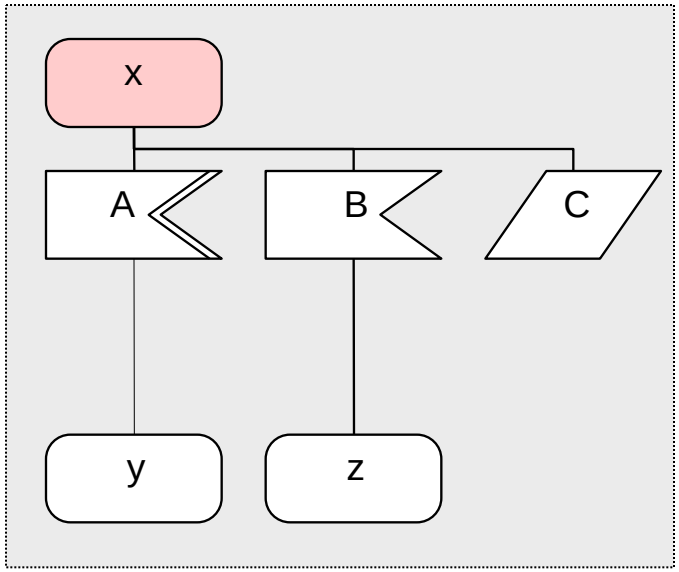


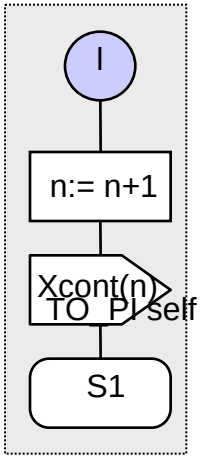


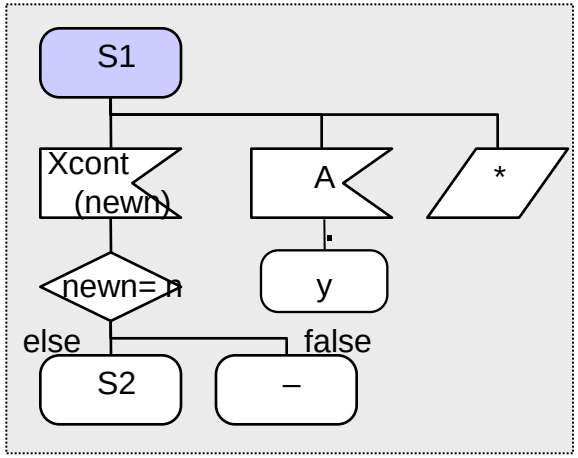


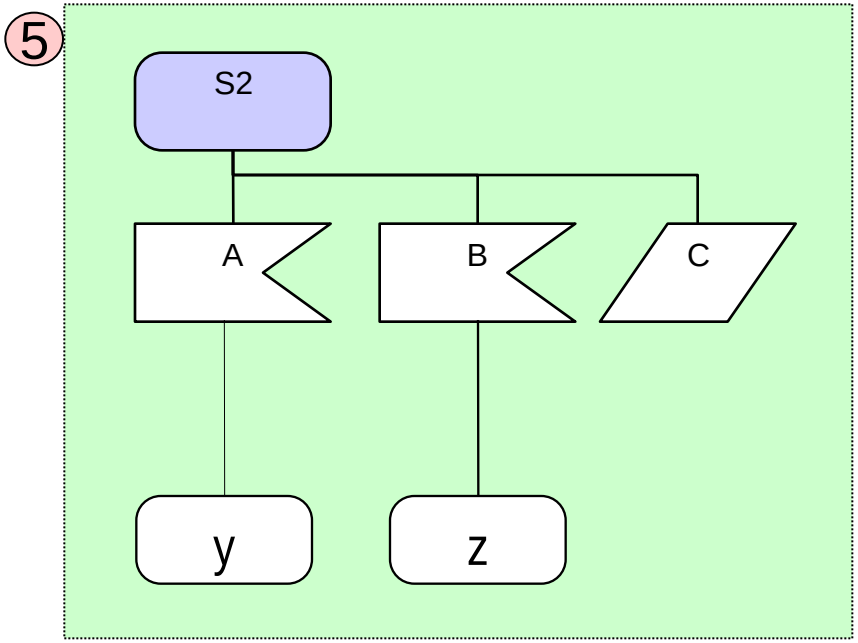
Ersetzungsmodell für priorisierten Input (9)

5. Schritt: Definition der Übergänge für S2



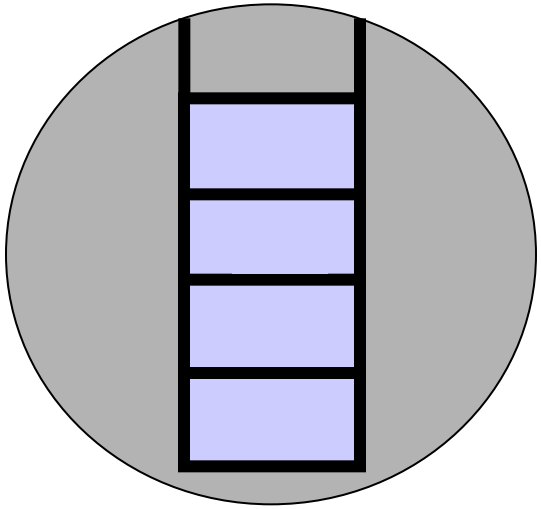


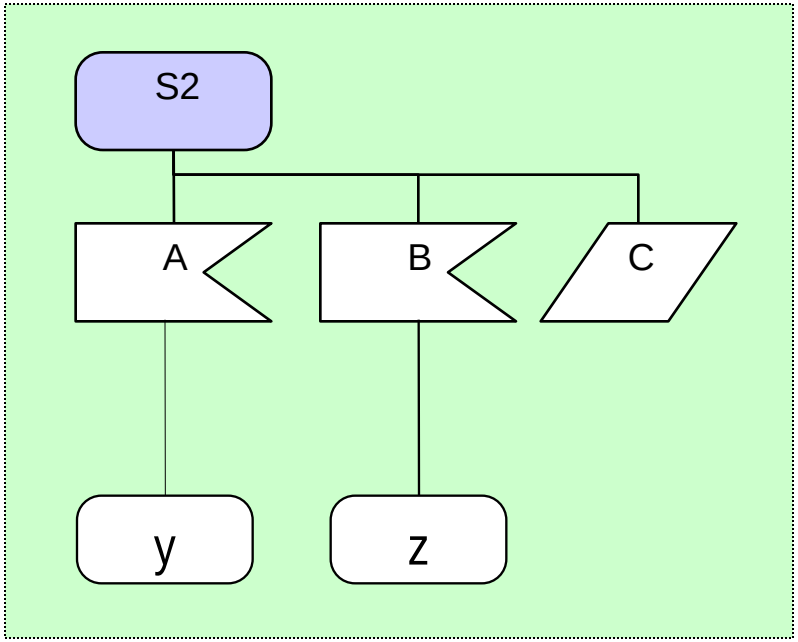




Zur Plausibilität des 5.Transformationsschrittes (1)

1.Szenario zur Bearbeitung der Signale im Zustand S2: sei kein Signal vorhanden

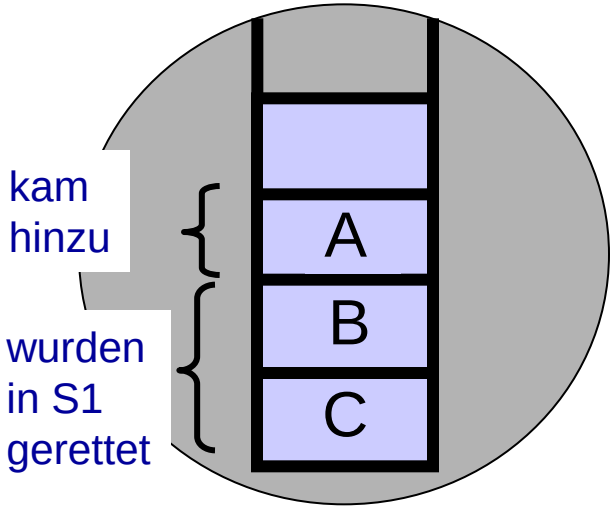


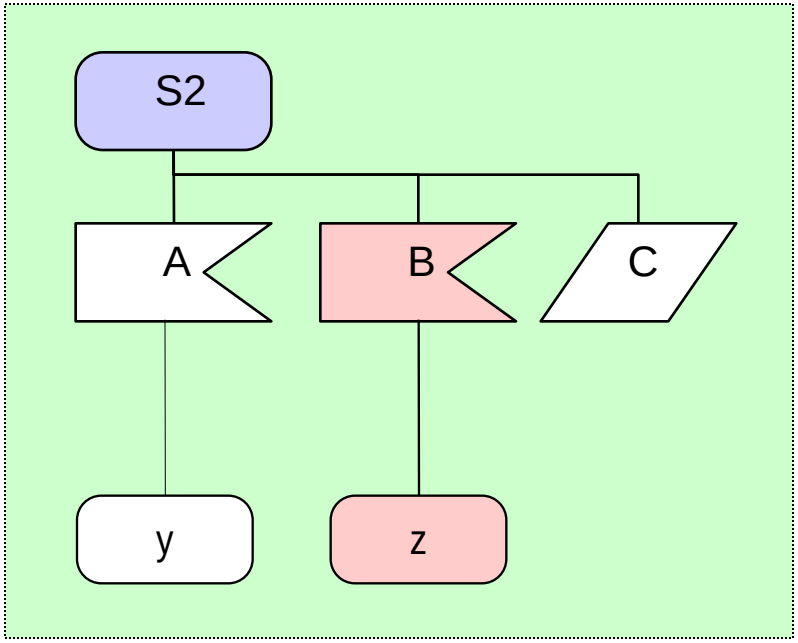


- Das erste ankommende Signal entscheidet die Fortsetzung
- Sollte nun zuerst A eintreffen, darf es nicht verloren gehen

Zur Plausibilität des 5.Transformationsschrittes (2)

2.Szenario zur Bearbeitung der Signale im Zustand S2: seien Signale vorhanden

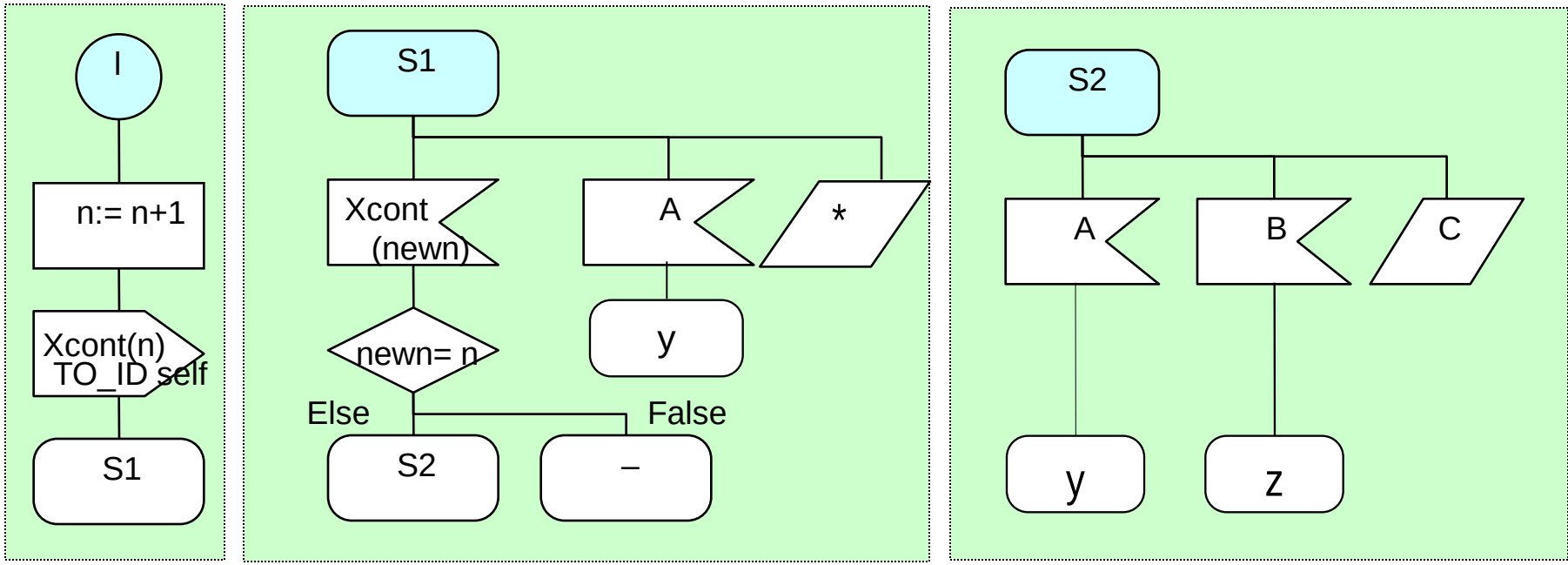




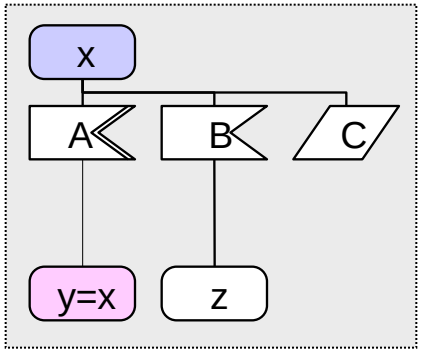
- Das erste ankommende Signal entscheidet die Fortsetzung
- A wird nicht berücksichtigt: zum Zeitpunkt des Eintritts in den Zustand x war nämlich A noch nicht vorhanden

Ersetzungsmodell für priorisierten Input (10)

offene Fragen:
- Notwendigkeit der Parametrisierung von *Xcont*
- ~~Behandlung spontaner Transitionen im Zustand, der priorisierte Transitionen verarbeitet~~

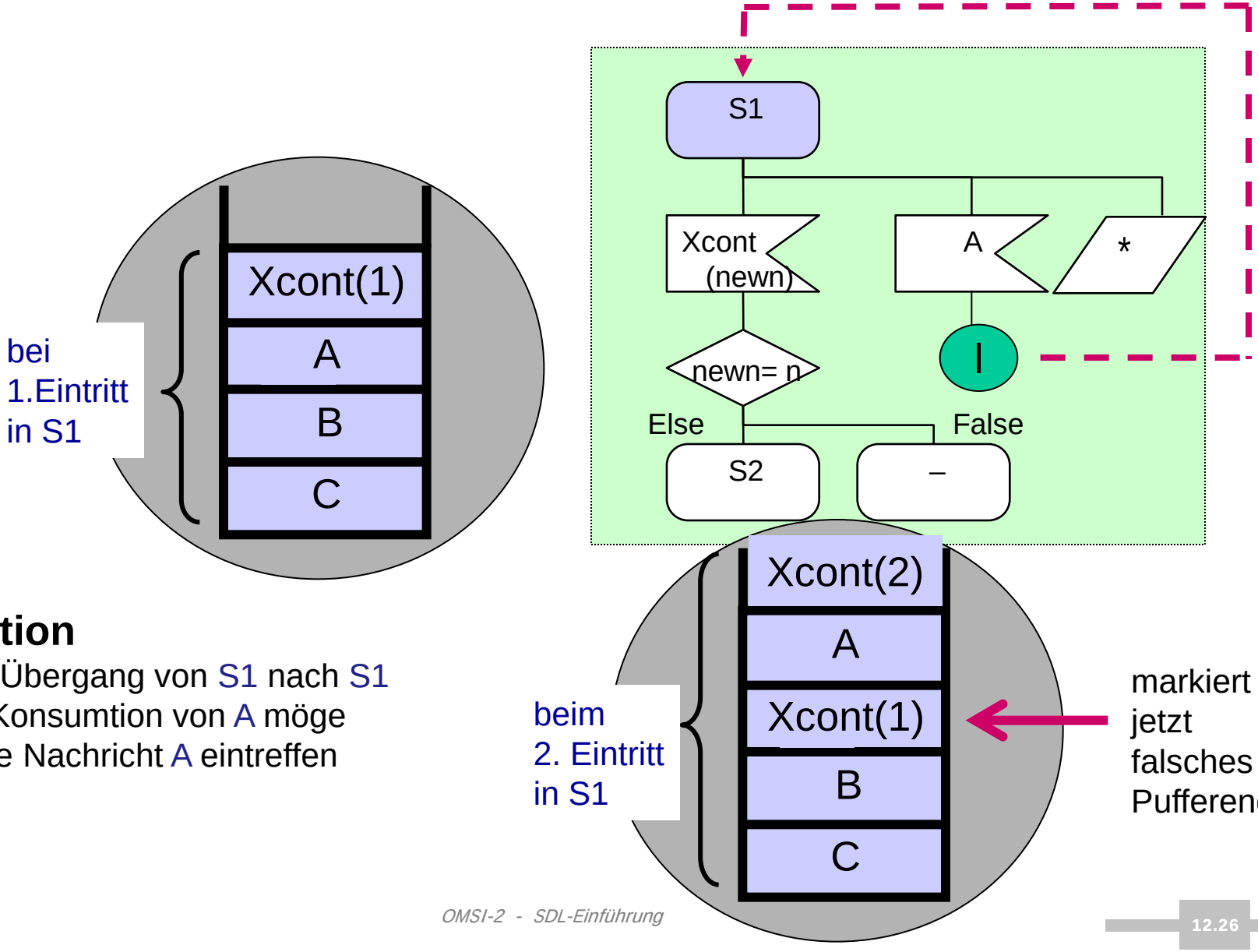


Zur Plausibilität der Xcont-Parametrisierung



notwendig für Spezialfall:
Folgezustand *y* für priorisierten A-Trigger ist wieder Ausgangszustand *x*

Zur Plausibilität der Xcont-Parametrisierung



Situation

- beim Übergang von S1 nach S1 nach Konsumtion von A möge weitere Nachricht A eintreffen

9. SDL-Konzepte *(Präzisierung, 2.Teil)*

1. Ersetzungskonzept: Shorthand-Notation
2. Ersetzungskonzept: Priorisierter Input
3. Prozeduren, Ersetzungskonzept: Remote-Prozeduren
4. Spezialisierung von Zustandsautomaten
5. Lokale Objekte, Semaphore

Prozeduren in SDL

… in zwei Ausprägungen, die zweckmäßiger Weise unterschiedliche Codegenierungsvarianten nach sich ziehen

a) zustandsbehaftet
   - Urform: ohne Rückgabewert
   - erweiterte Form: mit Rückgabewert (Spezialfall der Urform)

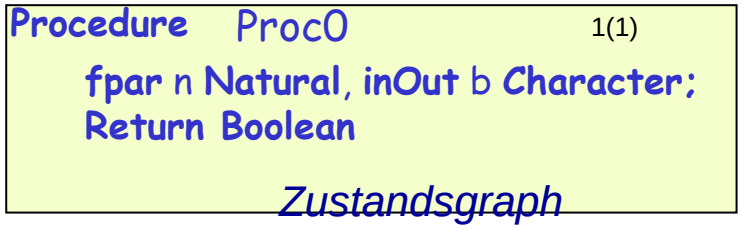
b) zustandslos (Spezialfall von a)

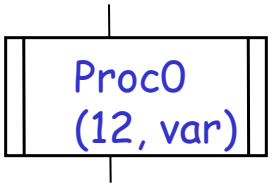
c) Funktionen/Memberfunktionen gab es vor SDL-2000 nicht (nur Operatoren, algebraisch def. Datentypen)

Diagrammarten und Aufruf in Standard-SDL:

Referenzsymbol
ohne Signaturangabe!

Prozedur-Diagramm





Prozedur-Aufruf

Prozeduren und Funktionen in SDL/RT

- SDL-Prozeduren
  - a) zustandslos / zustandsbehaftet: ohne Rückgabewert
  - b) zustandslos / zustandsbehaftet: mit Rückgabewert

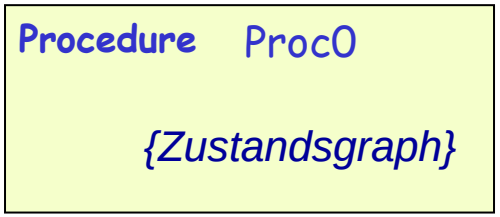
- C-/C++ Funktionen, Member-Funktionen

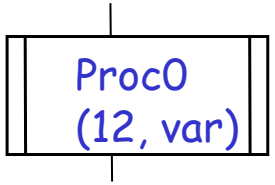
Diagrammarten und Aufruf in SDL/RT

Referenzsymbol
mit Signaturangabe!

ohne Rückgabewert

Prozedur-Diagramm





Aufruf

Zustandsbehaftete Prozedur

Zustandsgraph ähnelt dem eines Prozesses

Besonderheiten

- eigenes `start`- Symbol

  ➔Graph kann **nicht** mit Input beginnen

- eigene Zustandsmenge
- eigenes *stop*-Symbol (`return`)
- Marken und Zustände des Rufers können **nicht**

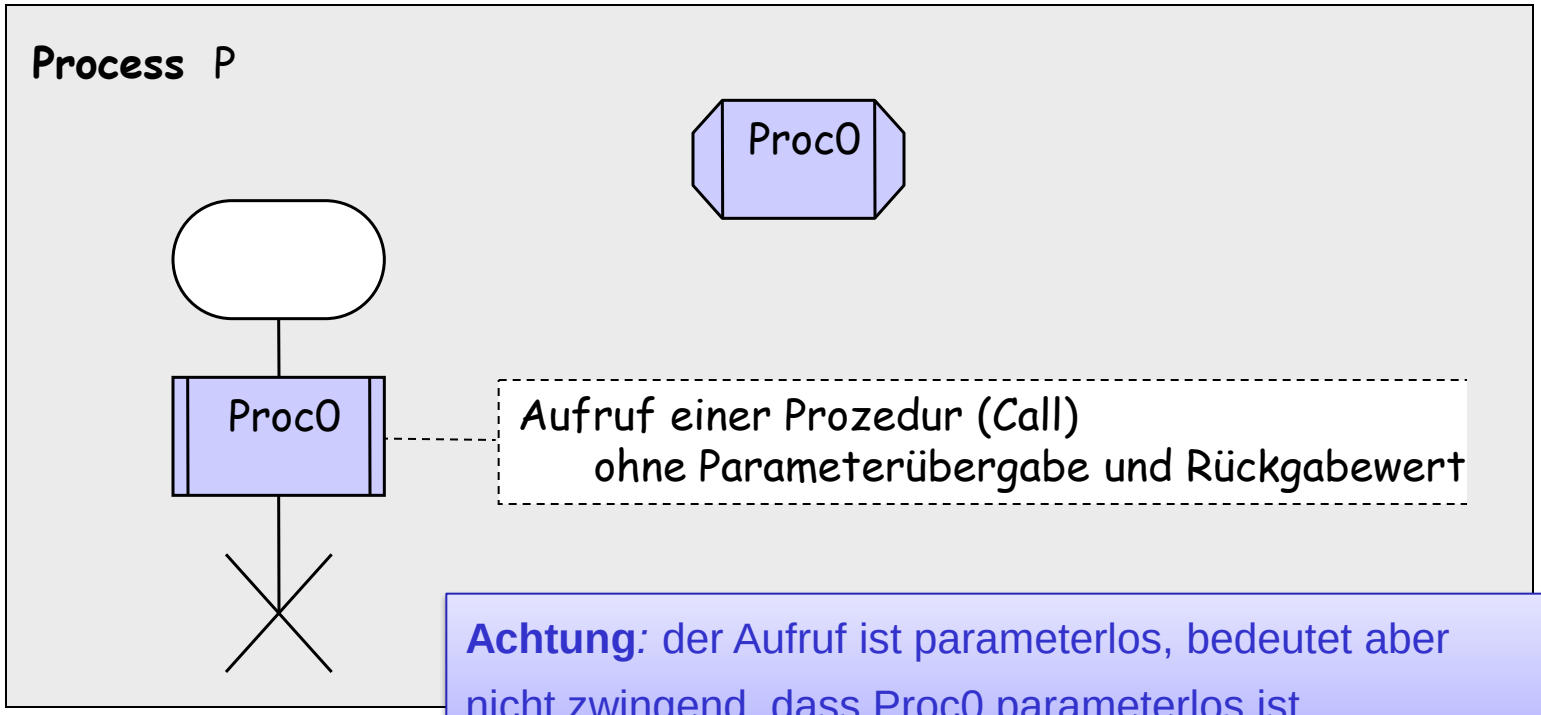
  aktiviert werden

  ➔ `state *(…)` bezieht sich **nur** auf Namensraum der Prozedur

- Zuordnung des Signalpuffers: rufender Prozess
- gültige Signal-Inputmenge stammt vom rufenden Prozess

  ➔ input *(...), save *(...)

Deklaration und Aufruf einer Prozedur ohne Rückgabewert (1)



Achtung: der Aufruf ist parameterlos, bedeutet aber nicht zwingend, dass Proc0 parameterlos ist
(es gilt die gleiche Konvention wie bei Signalparametern)

PROCEDURE-Call in SDL/RT

Aufruf **nicht** innerhalb von Tasks erlaubt

```
…;
x= myProc(p);
…
```

…es sei denn,
dass myProc eine C-Funktion ist

Achtung: Rekursivität ist erlaubt

```
r= oneProc(p)
```

```
anotherProc(p)
```

Aufruf einer
zustandslosen SDL-Prozedur
mit Rückkehrwert
entspricht einer C-Funktion
(empfohlen in SDL/RT)

Aufruf einer
echten SDL-Prozedur
(zustandsbehaftet,
ohne Rückkehrwert)

Aufruf an allen Stellen erlaubt, wo Tasks im Zustandsdiagramm möglich sind

Zustandsbehaftete Prozeduren in SDL

… sind parametrisierbare Zustandsübergangs-Teilgraphen eines Prozesses mit:

- dynamischen Kontextinformationen des aufrufenden Prozesses

  Timer, Inputpuffer, Empfangsnachrichten

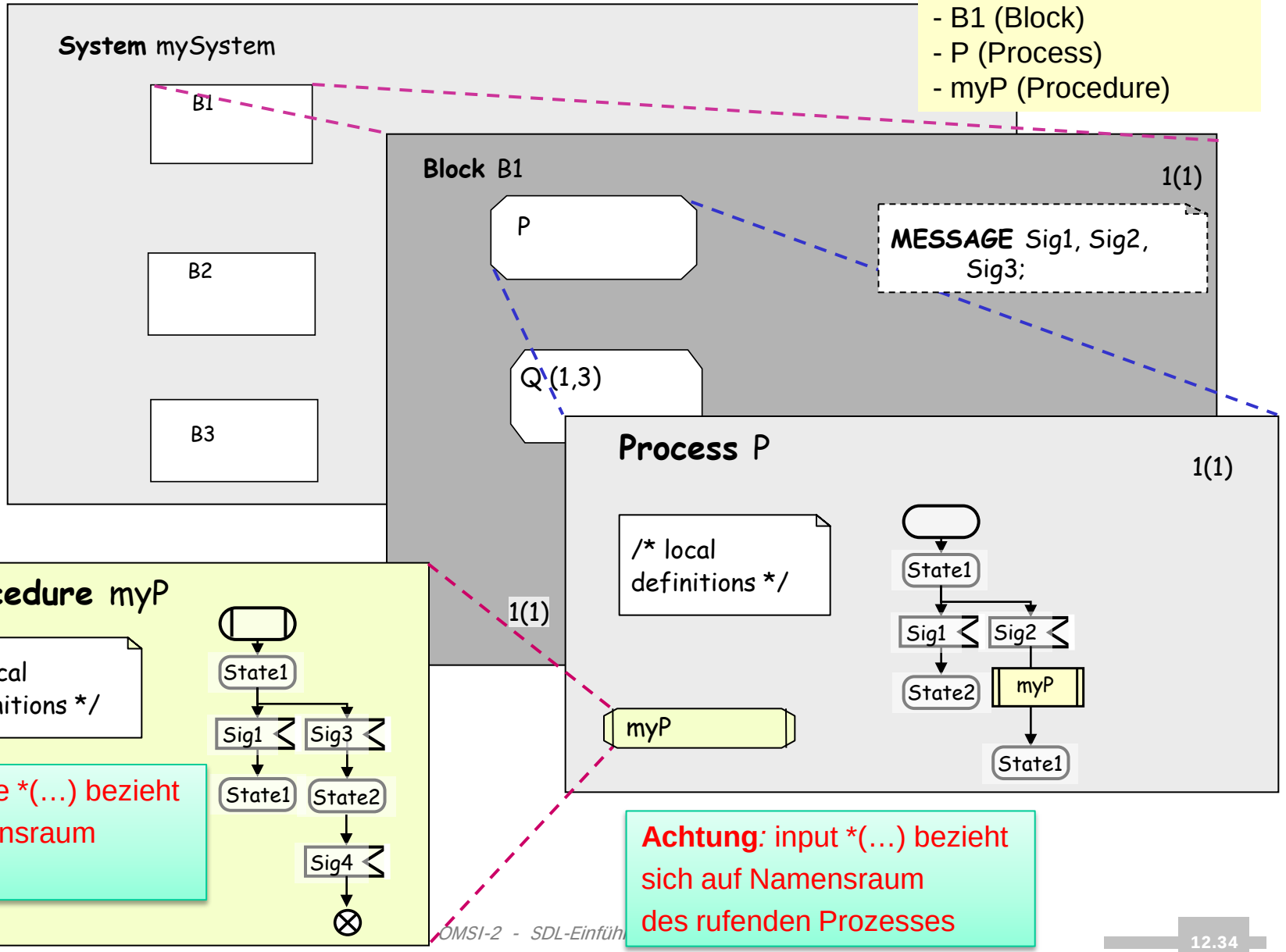
- statischen Kontextinformationen der Deklarationsumgebung

  Variablen, Funktionen, Typen

- eigenem Namensraum für *Zustände, Marken, Variablen, Datentypen*

Geschachtelte Namensräume

separate Namensräume
- mysystem (Sytem)
- B1 (Block)
- P (Process)
- myP (Procedure)



Achtung: state *(…) bezieht sich auf Namensraum der Prozedur

Achtung: input *(…) bezieht sich auf Namensraum des rufenden Prozesses

Erlaubte Deklarationsniveaus

- Package
- System (Systemtyp): system-global
- Block (Blocktyp): block-global
- Process (Processtyp): prozess-global
- ~~Service (Servicetyp): service-global~~
- Procedure: prozedur-global

Erlaubte Aufrufkontexte

Verhaltensgraph von einem

- Process
- ~~Service~~
- Procedure

Prozedur-Verhaltensgraph
operiert über Eingangspuffer des Prozesses, zu dem der Aufrufkontext gehört

Impliziter Parameter
Bezug zum Kontext vom Aufrufer-Prozess

Parameterübergabearten

für Prozeduren in Standard-SDL

- **in** (default)
  call-by-value: Ausdruckswerte der aktuellen Parameter werden in Kopie den formalen Parametern zugewiesen
- **in/out**
  call-by-reference: formale Parameter agieren als Synonyme der aktuellen Parameter während der Prozedur-Ausführung

für Prozesse und Nachrichten

- nur **in** möglich

für Prozeduren in SDL/RT

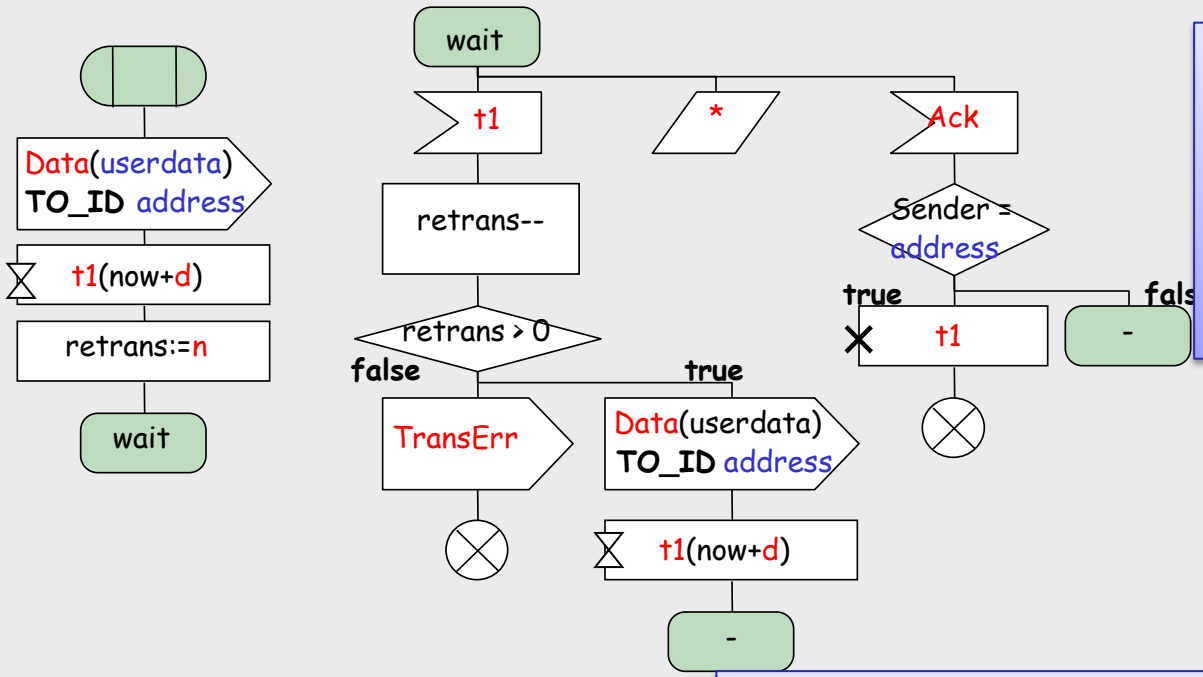
- **C-Konventionen**

Procedure sendData (Userdata userdata ; RTDS_QueueId address)

```
int retrans ; /* Anzahl möglicher Übertragungswiederholungen */

/* Kontextinformationen
   Duration d;
   int n;
   struct UserDataType ...;
   MESSAGE Data(UserdataType), Ack, TransErr;
   timer ...;
   signalset ...;
*/
```

Bereitstellungs-möglichkeiten



SDL/RT
statischer globaler Namensraum (System, Block, Prozess)

Standard-SDL
als expliziter dynamischer Kontext-Parameter

→ In Standard-SDL ist deshalb separate Prozedur-Definition möglich

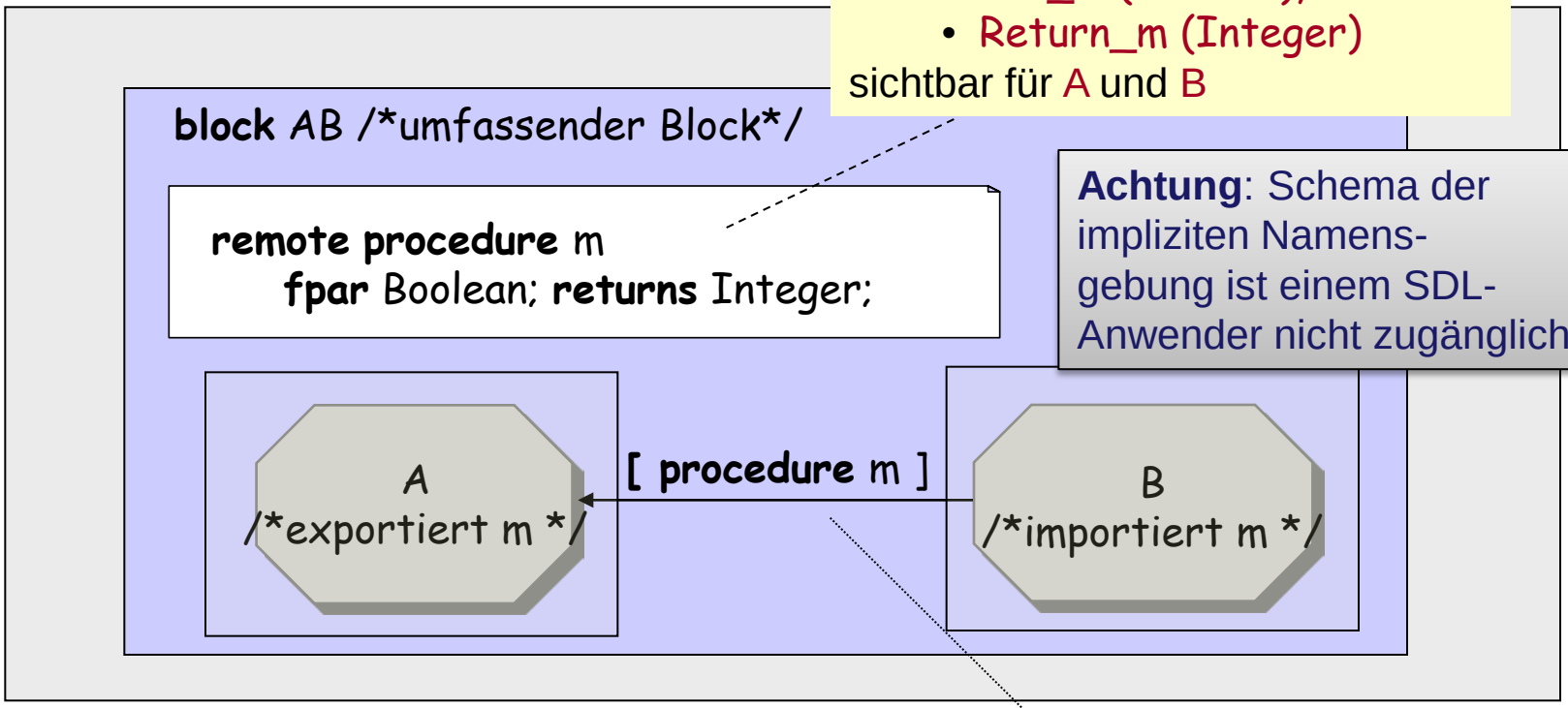
Allgemeines Konzept: Remote- Prozeduren (in Standard-SDL)

- **Entfernter Methoden-Ruf**
  - Ruf einer Methode eines anderen Objektes
- **Rollen** der beteiligten Prozess-Instanzen
  - Exporteur bietet Dienst als Prozedur an
  - Importeur verwendet die Prozedur als Dienst
- **Lokalisierung** der Partner
  - Prozess-Instanzen (Exporteur und Importeur-Instanzmengen) können verschiedenen Blöcken angehören
  - es muss aber immer einen umfassenden Block geben, der sowohl Importeur als auch Exporteur enthält (spätestens: System)
  - dieser umfassende Block hat die jeweilige Remote-Prozedur (Name und Signatur) zu deklarieren

Deklaration: Remote- Prozeduren

Transformation in implizite Signale:
- Call_m (Boolean),
- Return_m (Integer)

sichtbar für A und B



Achtung: Schema der impliziten Namens-gebung ist einem SDL-Anwender nicht zugänglich

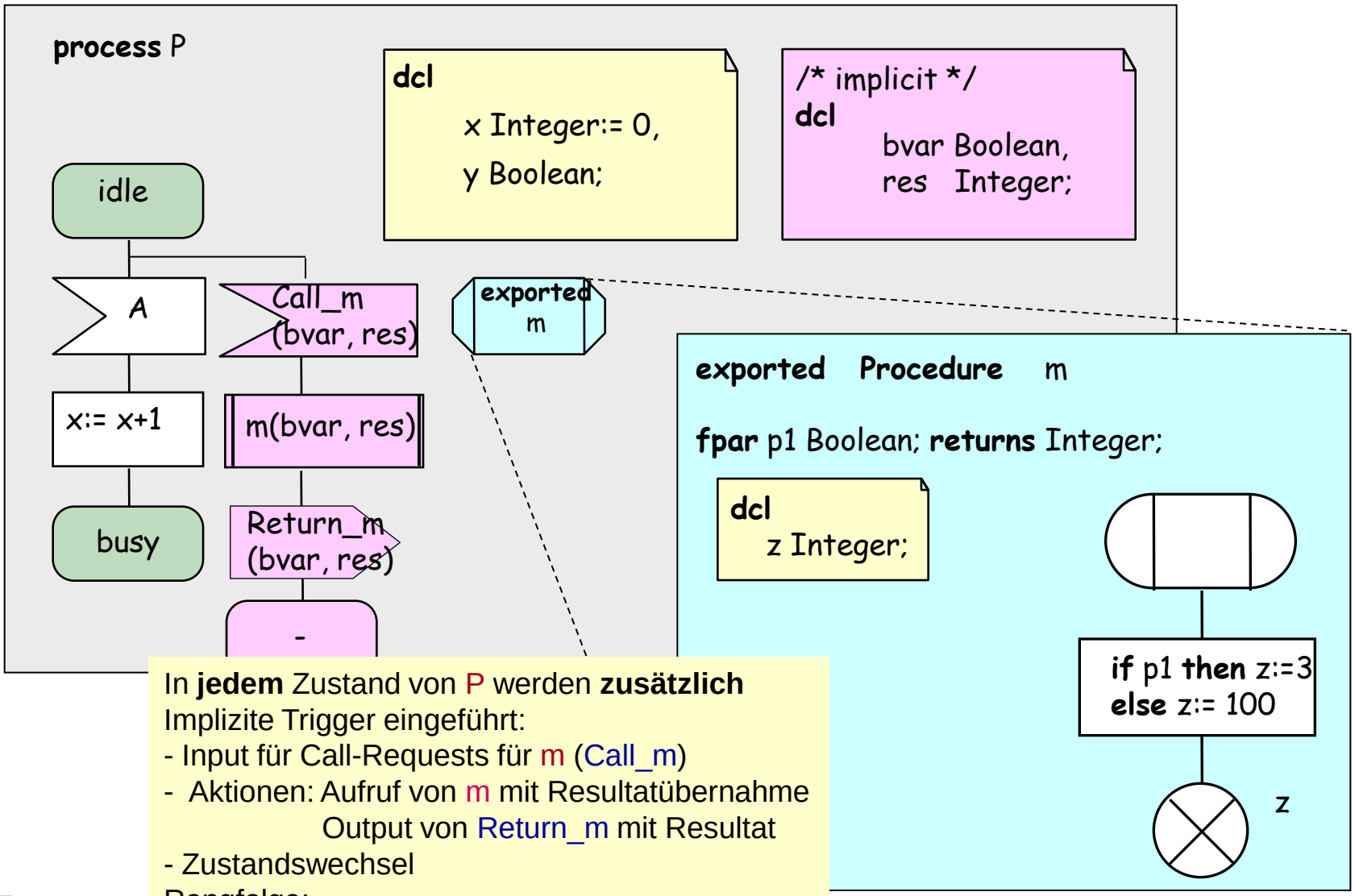
Namen von Remote-Prozeduren können in
- Signallisten,
- Kanälen, Routen, Gates und
- Signalsets

erscheinen (wenn nicht, dann implizit)

Angabe aus Richtung des Rufers/Importeurs (obwohl bi-direktional)

Exporteur einer Methode

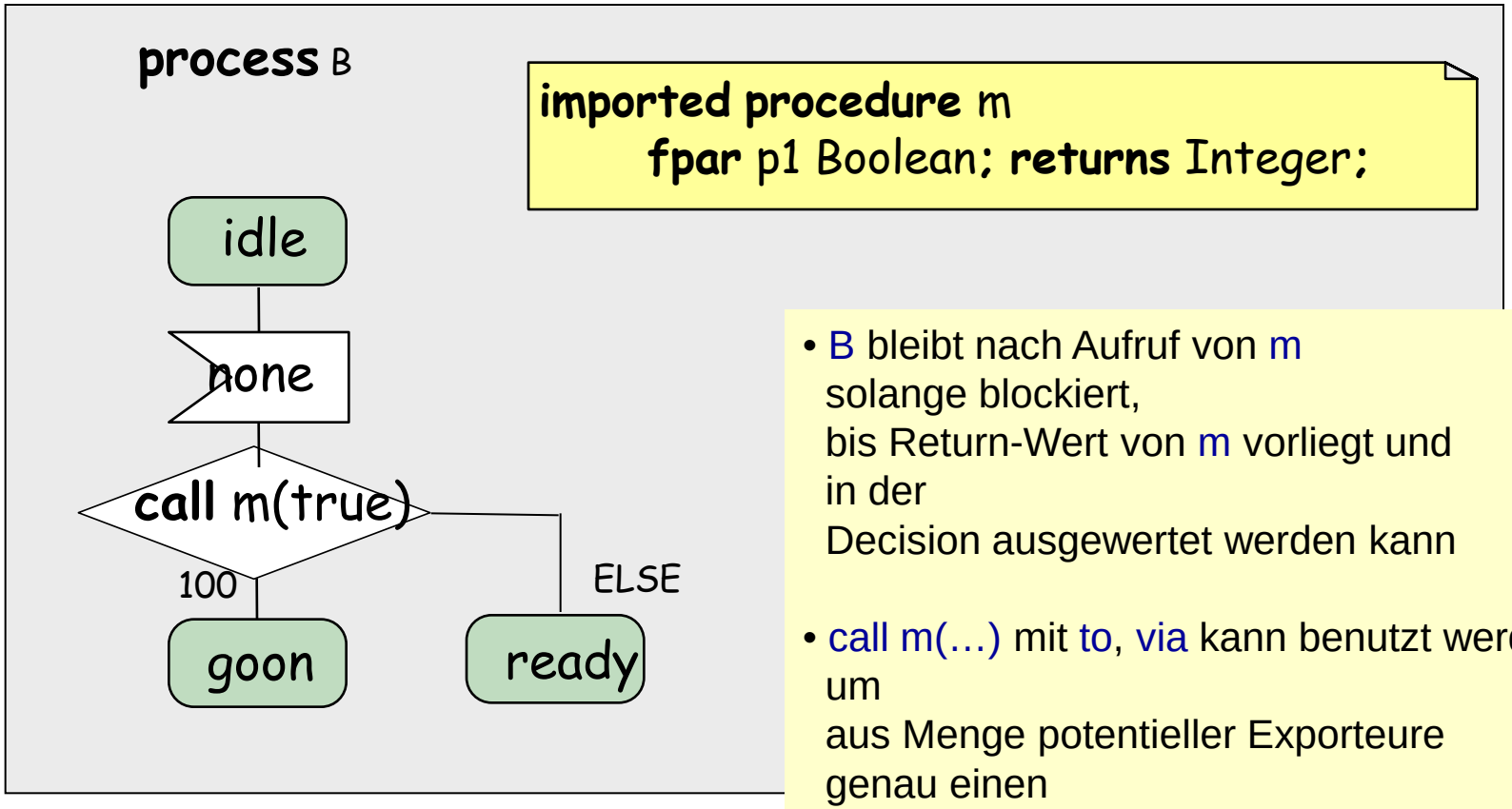
```
/* implicit */
signal
  Call_m (Boolean, Integer),
  Return_m (Boolean, Integer)
```



In **jedem** Zustand von P werden **zusätzlich** Implizite Trigger eingeführt:
- Input für Call-Requests für m (Call_m)
- Aktionen: Aufruf von m mit Resultatübernahme
  Output von Return_m mit Resultat
- Zustandswechsel

Rangfolge:
FCFS, wie normale Input-Signal-Trigger

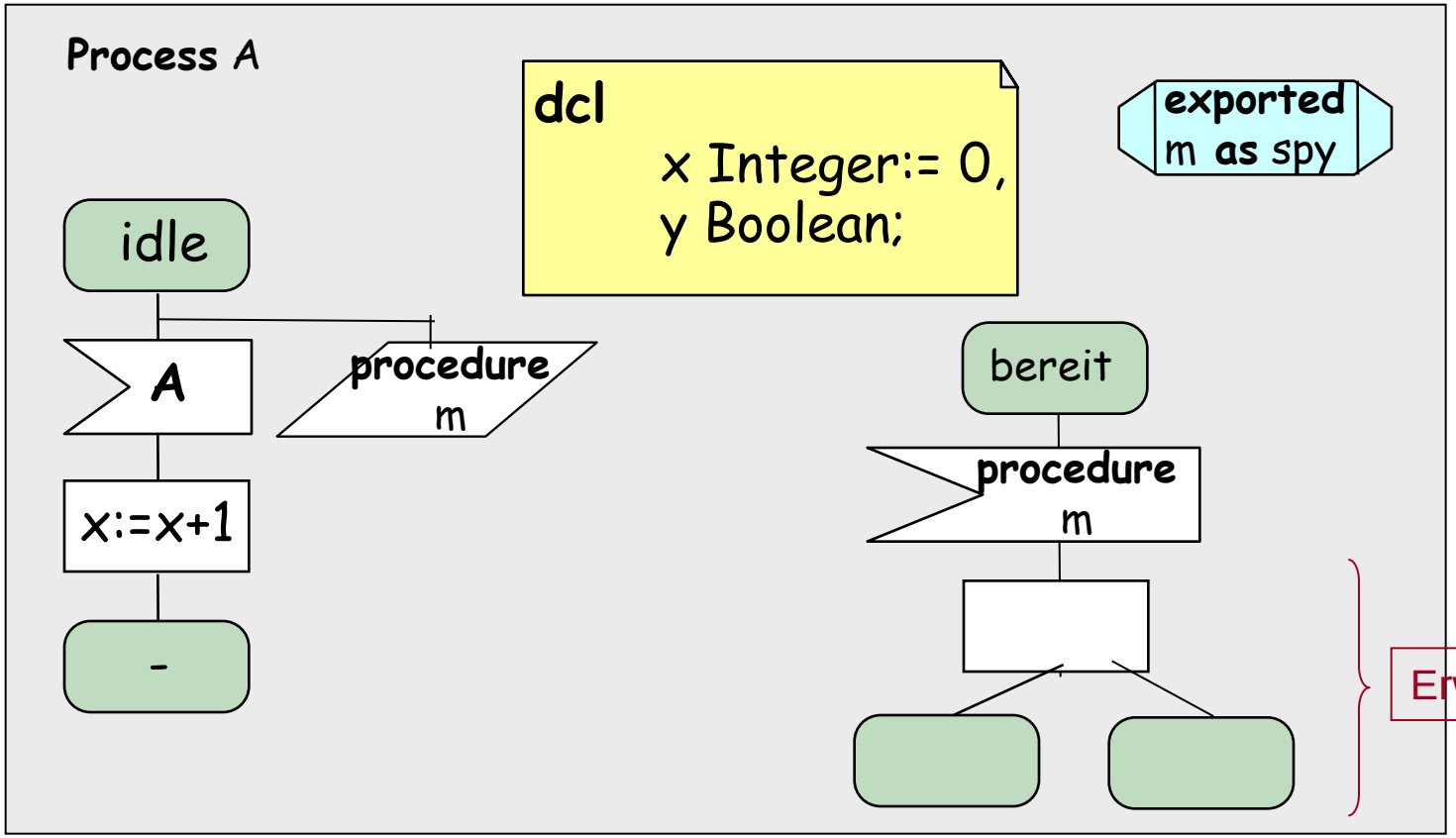
Importeur einer Methode



- B bleibt nach Aufruf von m
  solange blockiert,
  bis Return-Wert von m vorliegt und
  in der
  Decision ausgewertet werden kann

- call m(…) mit to, via kann benutzt werden, um
  aus Menge potentieller Exporteure
  genau einen
  Exporteur oder eine Teilmenge
  (Prozessinstanz-
  Menge) auszuwählen,
  aus der der Exporteur
  nichtdeterministisch bestimmt wird

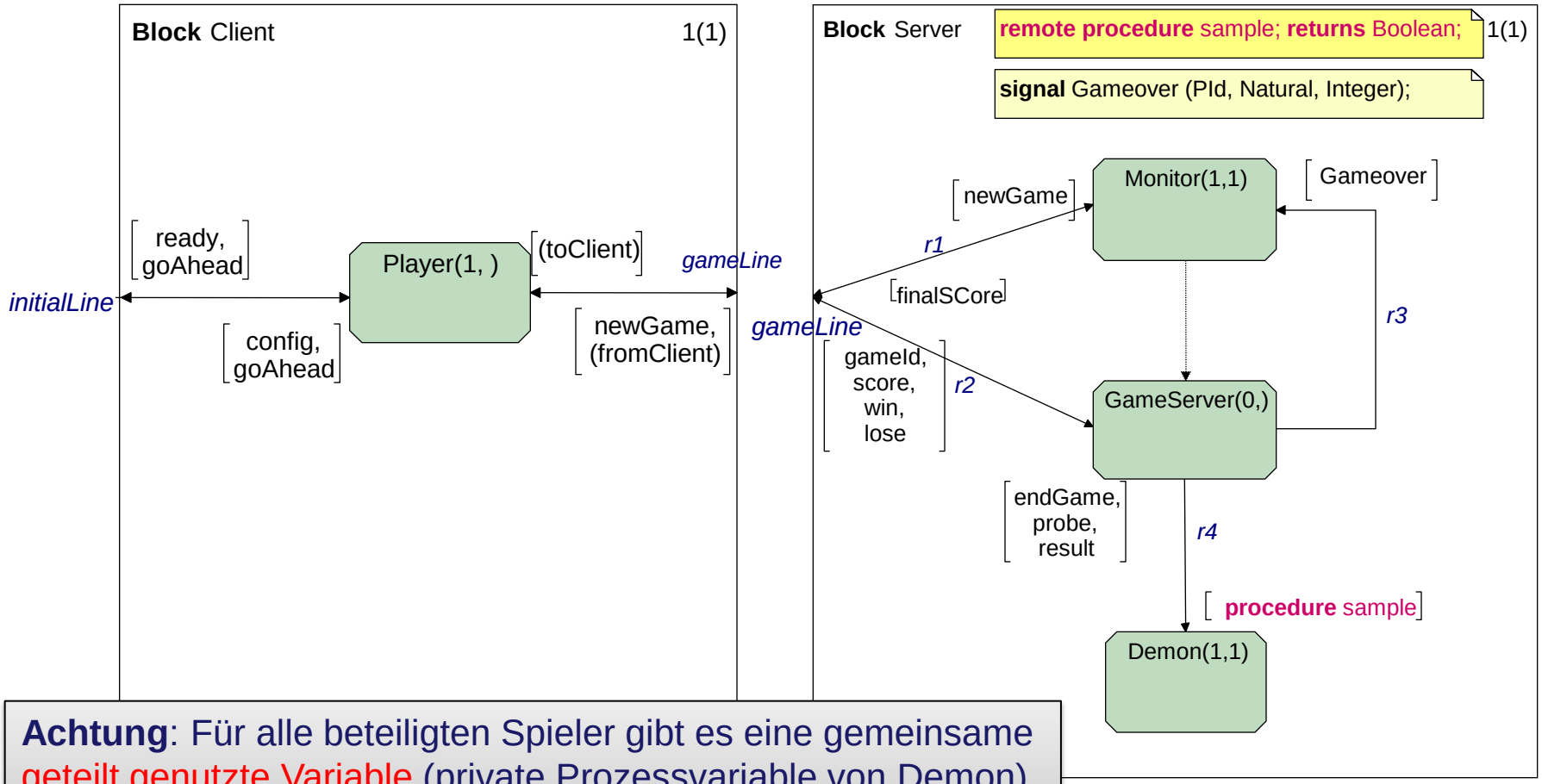
Verhaltensflexibilität des Exporteurs



Erweiterung der Standardsemantik:

- Prozedur kann unter anderem Namen exportiert werden (hier: spy)
- explizite Angabe von Zuständen, die Call-Requests akzeptieren (hier: bereit bei zusätzlicher Zustandsgraph-Erweiterung
- explizite Angabe von Zuständen, die Call-Requests zurückstellen (hier: idle)

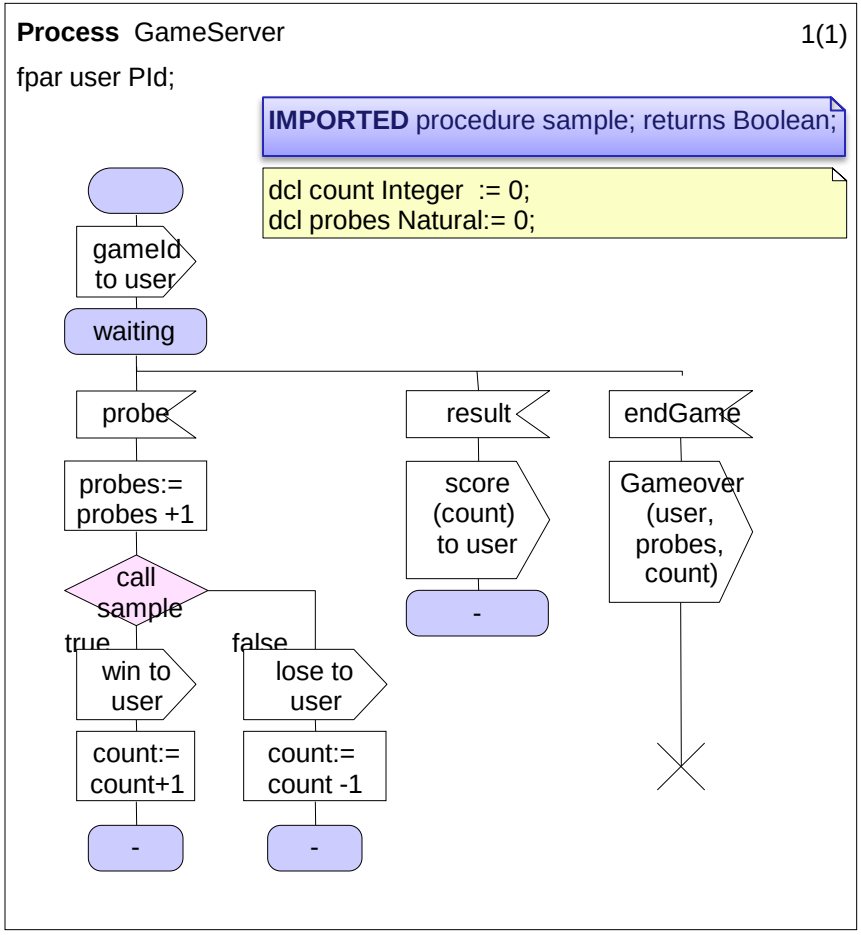
RPC-Beispiel: Demon-Game in Standard-SDL

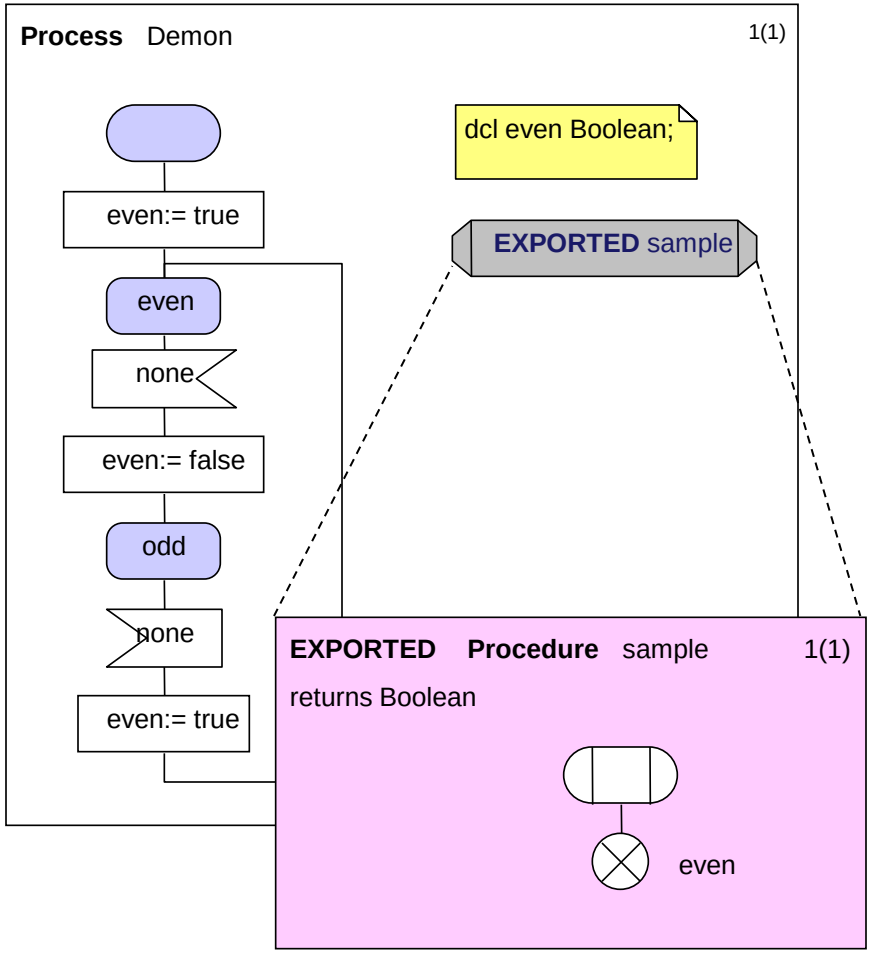


Achtung: Für alle beteiligten Spieler gibt es eine gemeinsame geteilt genutzte Variable (private Prozessvariable von Demon)
Ein gleichzeitiger Zugriff per Remote Prozedur
muss zwangssequentialisiert werden

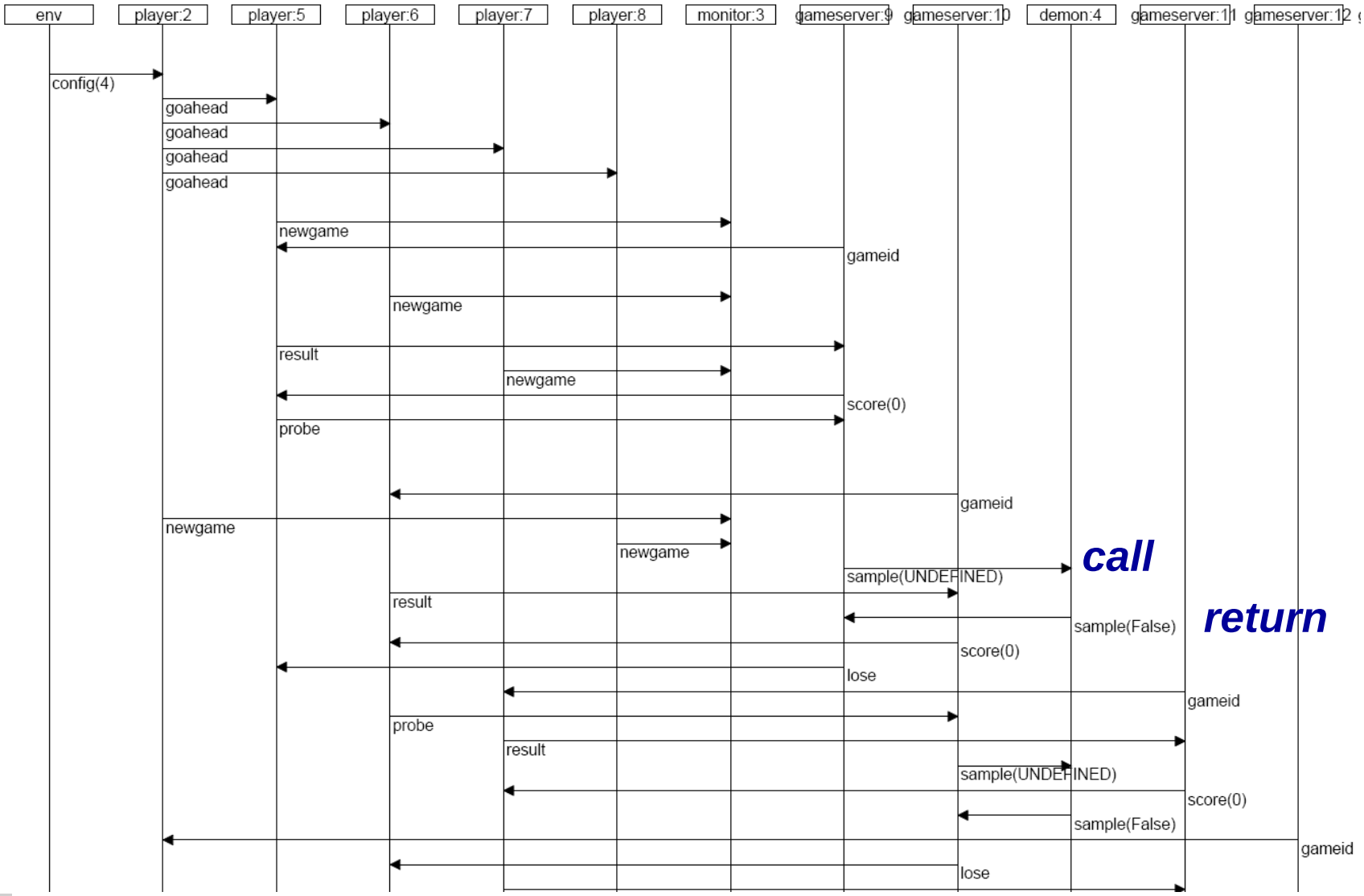
Dreiklang: Remote – Imported - Exported

REMOTE procedure sample; returns Boolean;

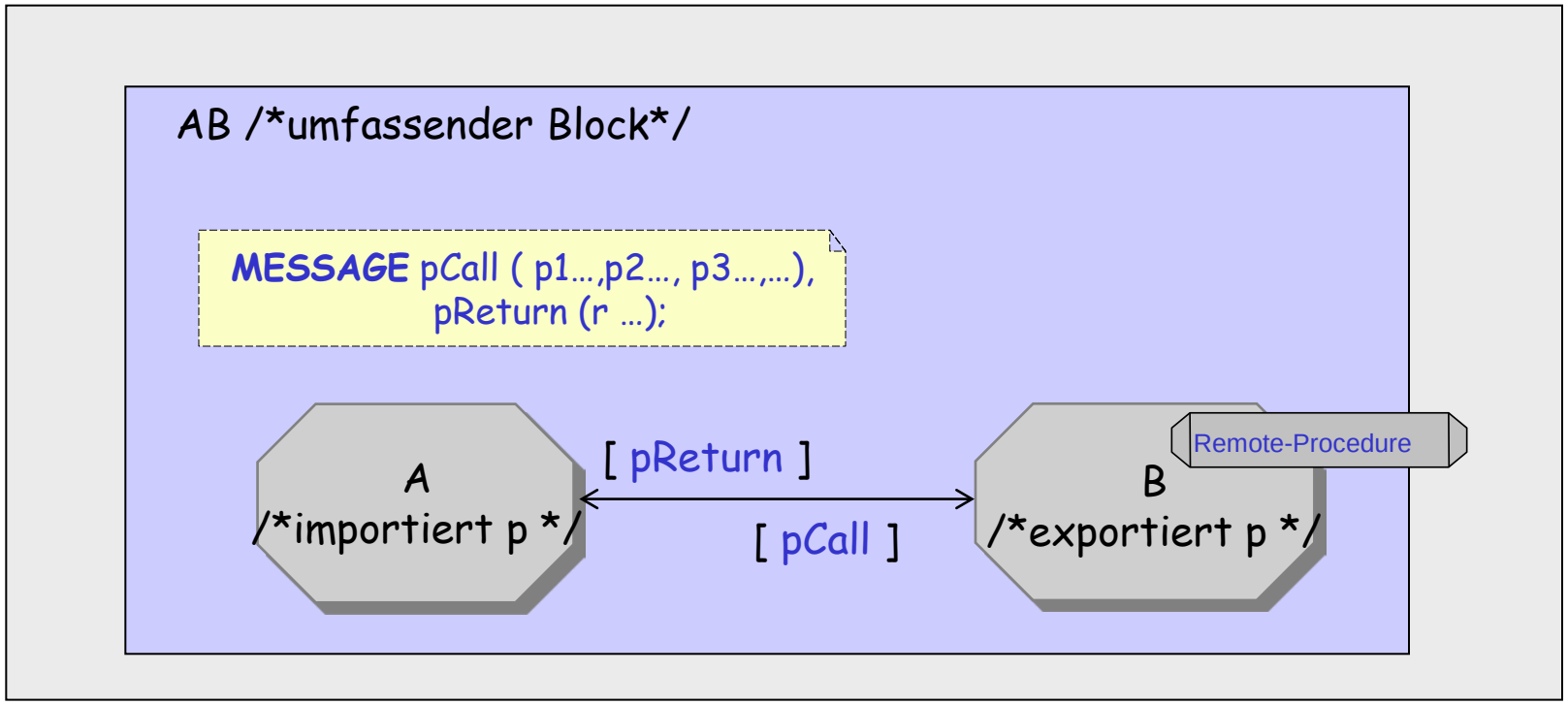




Ablauf mit Ruf und Return einer Remote-Prozedur



RPC-Nachbildung in SDL/RT



ÜA:???