Modellbasierte Entwicklung von Technologien für selbstorganisierende dezentrale Informations-systeme im Katastrophen-management

METRIK

Projekt Erdbebenfrühwarnung
im SoSe 2011

HUMBOLDT-UNIVERSITÄT ZU BERLIN

Entwicklung verteilter echtzeitfähiger Sensorsysteme

Joachim Fischer
Klaus Ahrens
Ingmar Eveslage

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

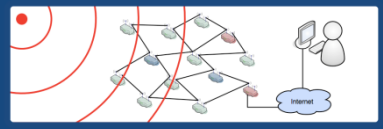
SAFER
Seismic eArly warning For EuRope

EDIM

SOSEWIN-extented

GFZ
Helmholtz-Zentrum
POTSDAM

Systemanalyse



7. SDL-Konzepte (Präzisierung)

1. Modellstruktur
2. Einfacher Zustandsautomat: Triggerarten
3. Ersetzungsmodell: Priorisierter Input
4. Nachrichtenadressierung
5. Prozeduren
6. Timer
7. Remote Prozeduren
8. Dynamische Prozessgenerierung
9. Spezialisierung von Zustandsautomaten
10. Lokale Objekte, Semaphore

Zwei Adressierungsarten

(1) OUTPUT *msg-name (param-List)* **TO_X** *receiver*
(2) OUTPUT *msg-name (param-List)* **VIA** *medium*

Nachrichtenparameter: Werte, Referenzen
bei Referenzen ist wichtig: Welcher Prozess legt Lebenslauf eines Parameter-Objektes fest?

typische Probleme:

- Sender besitzt Datum nicht mehr, nachdem gesendet worden ist,
- Empfänger gibt Speicherplatz frei

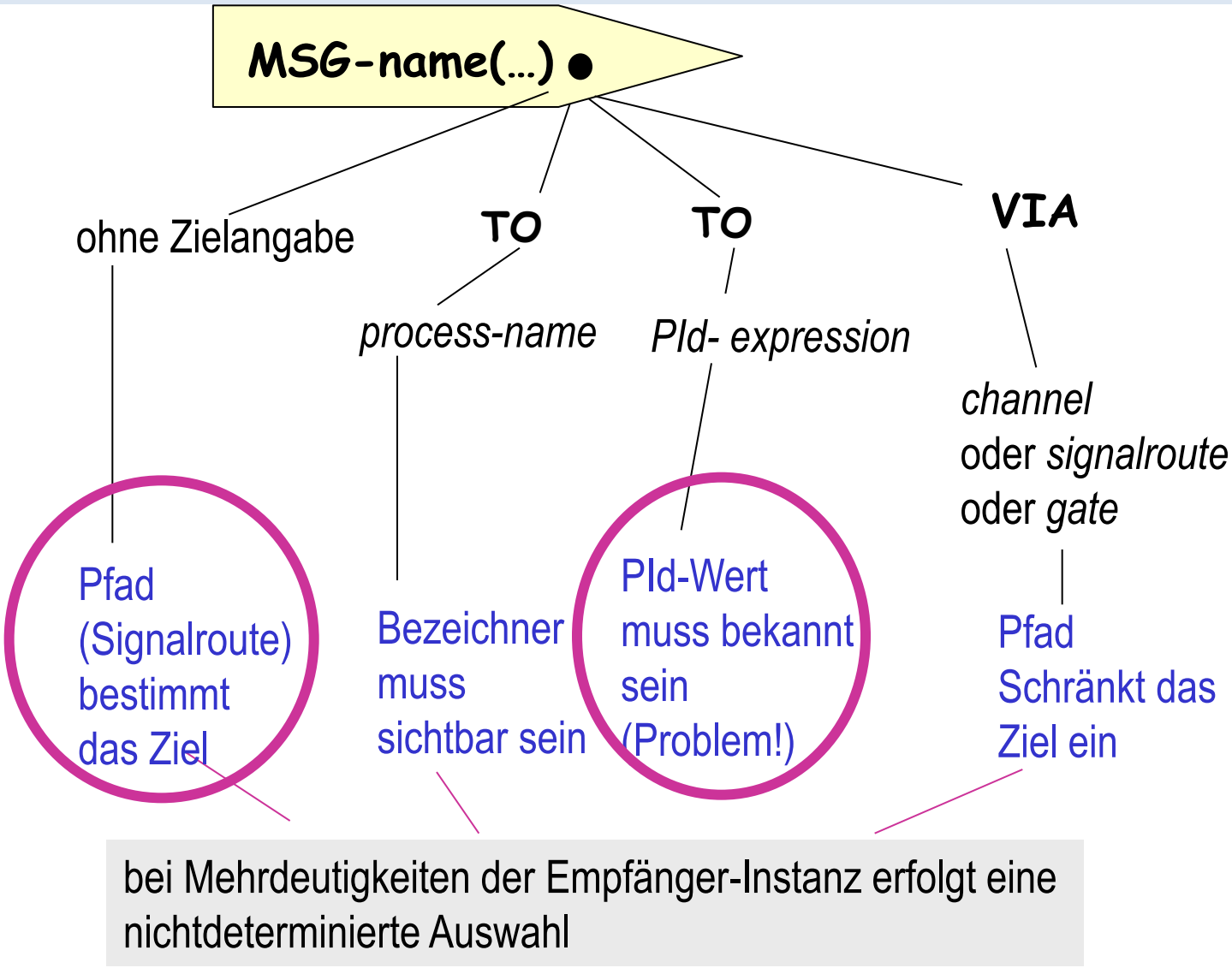
– **TO_X**:

- **ID**: PARENT, SELF, OFFSPRING, SENDER, oder RTDS_QueueId
- **NAME**: erster generierter Prozess einer Menge
  (nicht benutzen, wenn Name nicht 1-deutig)
- **ENV**

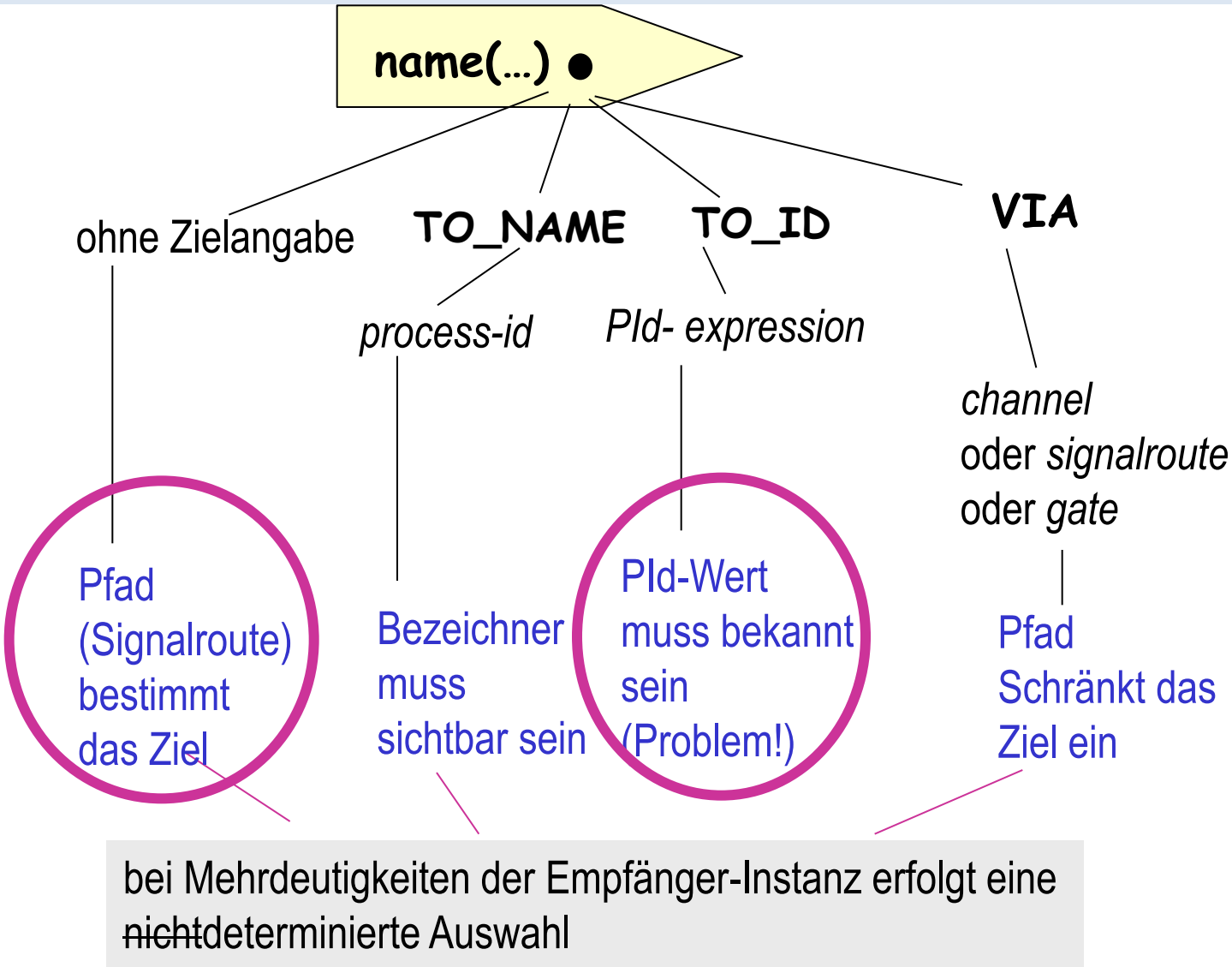
– **VIA** medium:

- Name von Channel oder Gate (verbunden mit aktuellem Prozess)
  keine Mehrdeutigkeit zulässig!

SDL-Standard: Nachrichten-Adressierungsarten



SDL-RT: Nachrichten-Adressierungsarten



7. SDL-Konzepte (Präzisierung)

1. Modellstruktur
2. Einfacher Zustandsautomat: Triggerarten
3. Ersetzungsmodell: Priorisierter Input
4. Nachrichtenadressierung
5. Prozeduren
6. Timer
7. Remote Prozeduren
8. Dynamische Prozessgenerierung
9. Spezialisierung von Zustandsautomaten
10. 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)

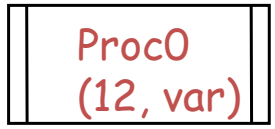
b) zustandslos (Spezialfall von a)

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

Diagrammarten und Aufruf in Standard-SDL:

Referenzsymbol
ohne Signaturangabe!

Prozedur-Diagramm (Kopf)



Prozedur-Aufruf

Prozeduren und Funktionen in SDL/RT

- SDL-Prozeduren
  - a) zustandslos : mit und ohne Rückgabewert , aber hier sind C-Funktionen besser
  - b) zustandsbehaftet: ohne Rückgabewert

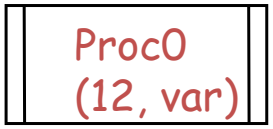
- C-/C++ Funktionen, Member-Funktionen

Diagrammarten und Aufruf in SDL/RT

Referenzsymbol
mit Signaturangabe!

Beispiel ohne Rückgabewert

Prozedur-Diagramm



Aufruf

Zustandsbehaftete Prozeduren in SDL

… sind parametrisierbare Zustandsübergangs-Teilgraphen eines Prozesses mit:

- dynamischen Kontextinformationen des aufrufenden Prozesses

  Timer, Inputpuffer, Empfangsnachrichten

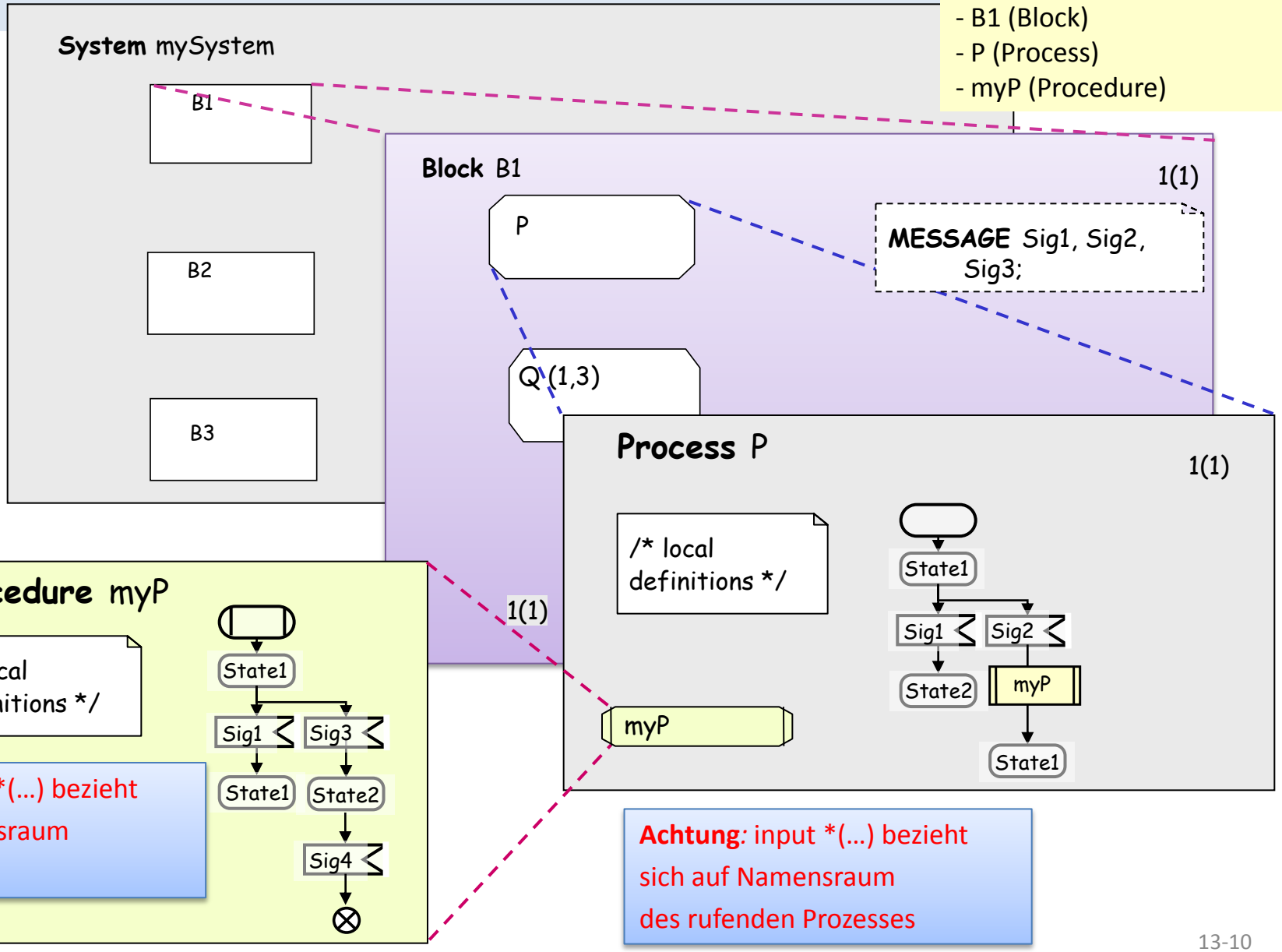
- statischen Kontextinformationen der Deklarationsumgebung

  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

Zustandsbehaftete Prozedur

Zustandsgraph ähnelt dem eines Prozesses

Besonderheiten

- eigenes **start-** Symbol

  ➔Graph kann **nicht** mit Input beginnen

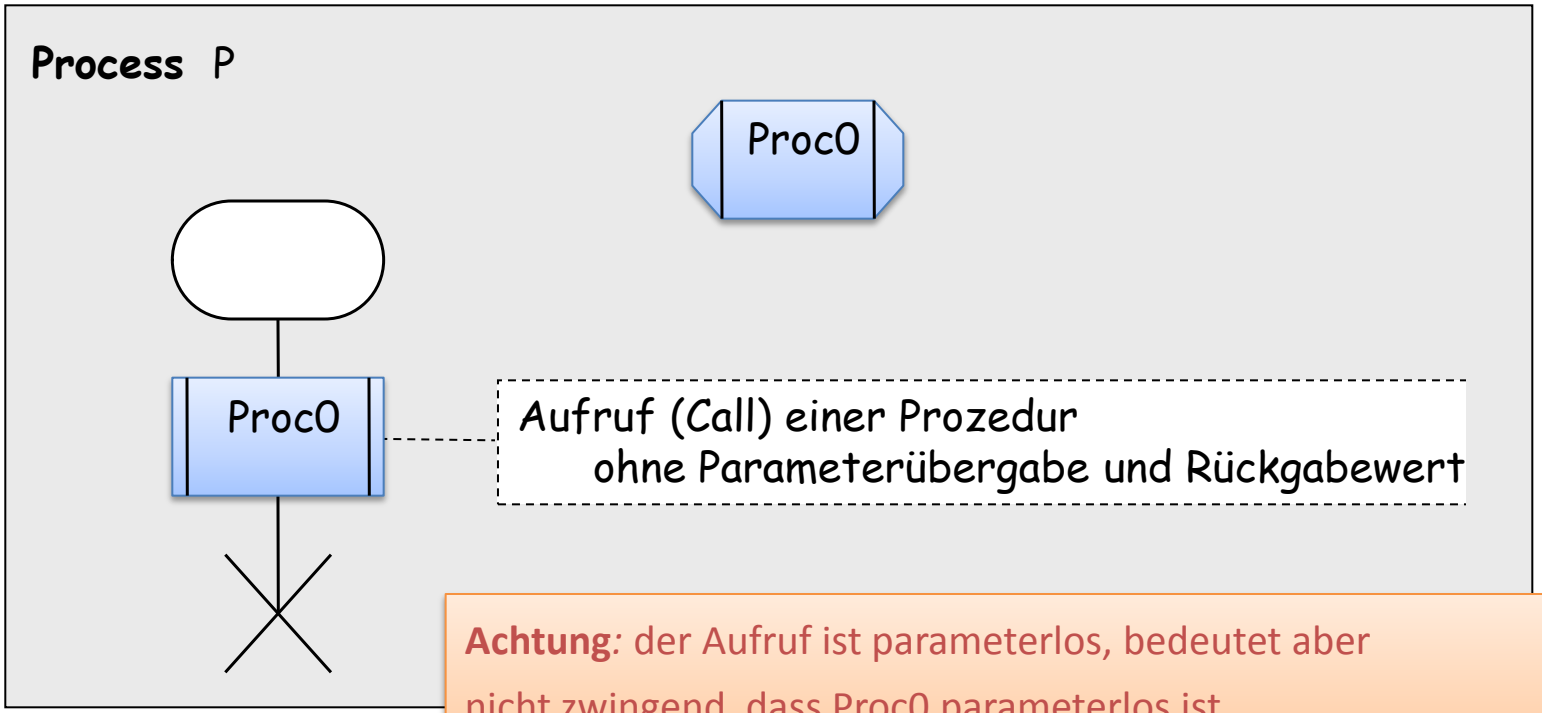
- eingener Namensraum (insb. eigene Zustandsmenge)
- eigenes *stop*-Symbol (**return**)
- Marken und Zustände des Rufers können **nicht** aktiviert werden

  ➔ 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

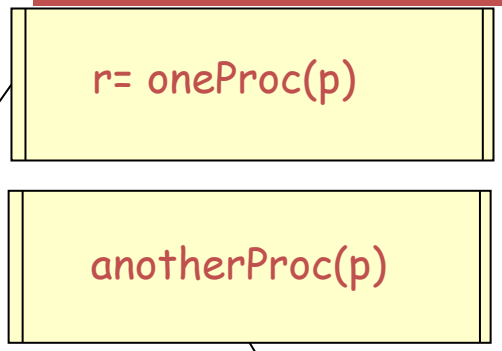
Aufruf **nicht** innerhalb von Tasks erlaubt

```
...;
x= myProc(p);
...
```

...es sei denn,

dass myProc eine C-Funktion ist

Achtung: Rekursivität ist erlaubt



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

Erlaubte Deklarationsniveaus von Prozeduren

- 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

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

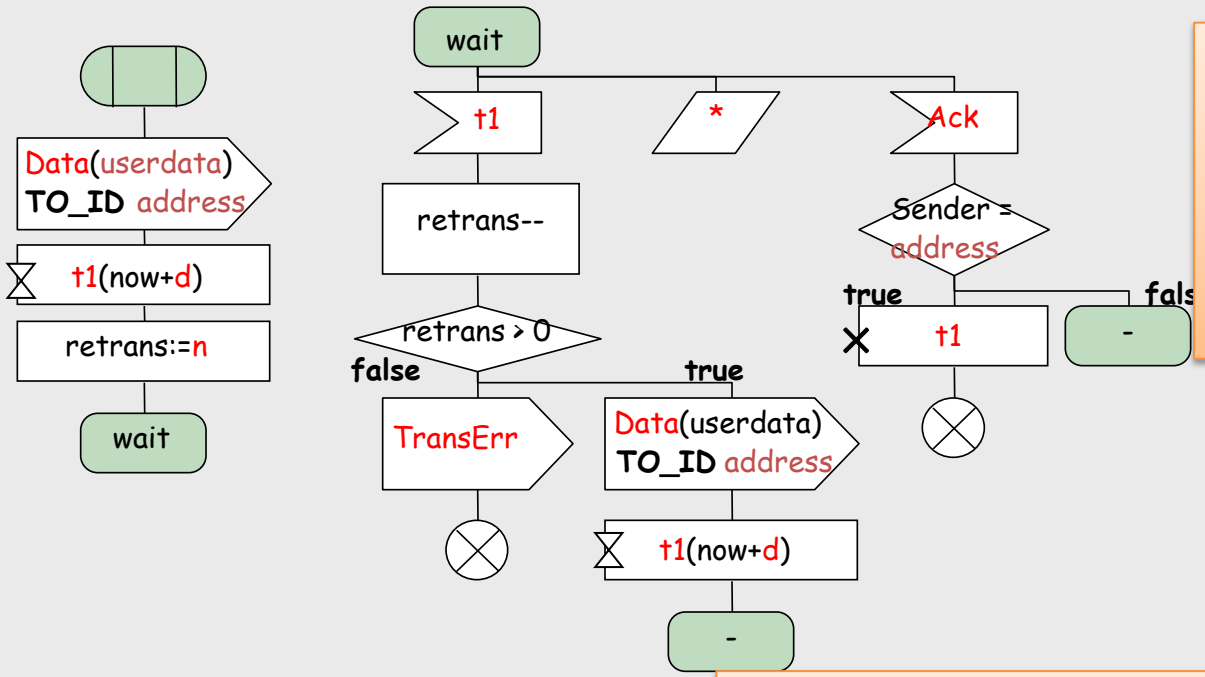
- **C-Konventionen**

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
auch als expliziter dynamischer Kontext-Parameter

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

7. SDL-Konzepte (Präzisierung)

1. Modellstruktur
2. Einfacher Zustandsautomat: Triggerarten
3. Ersetzungsmodell: Priorisierter Input
4. Nachrichtenadressierung
5. Prozeduren
6. Timer
7. Remote Prozeduren
8. Dynamische Prozessgenerierung
9. Spezialisierung von Zustandsautomaten
10. Lokale Objekte, Semaphore

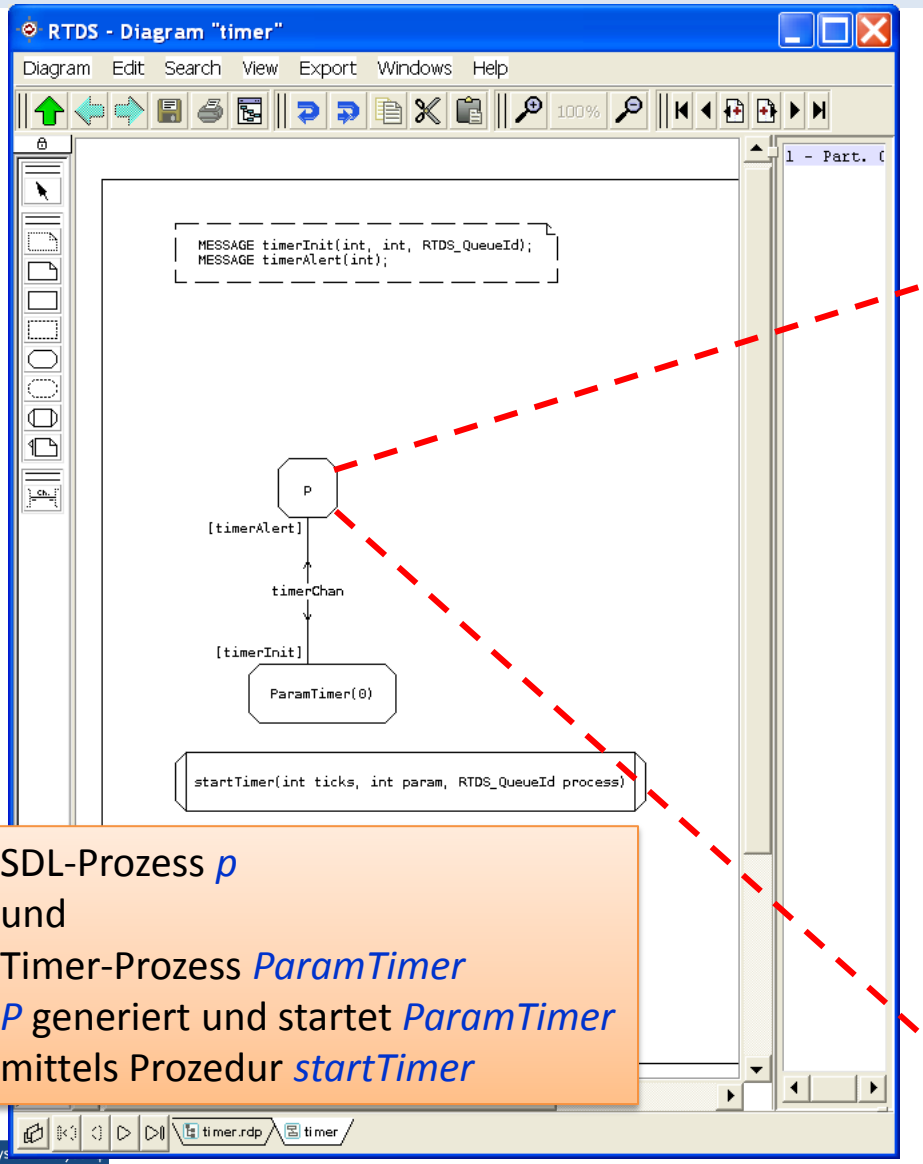
Timer-Konzept

- Grundkonzept zur Verhinderung von Deadlocks und Lifelocks in realen verteilten Systemen
  - Starten: start/restart bei Angabe der Timeoutzeit
  - Stoppen: stop
  - Timeout: Timout-Nachrichten unterscheidbar durch Namen des auslösenden Timers
  - jeder Prozess kann über beliebig viele Timer verfügen

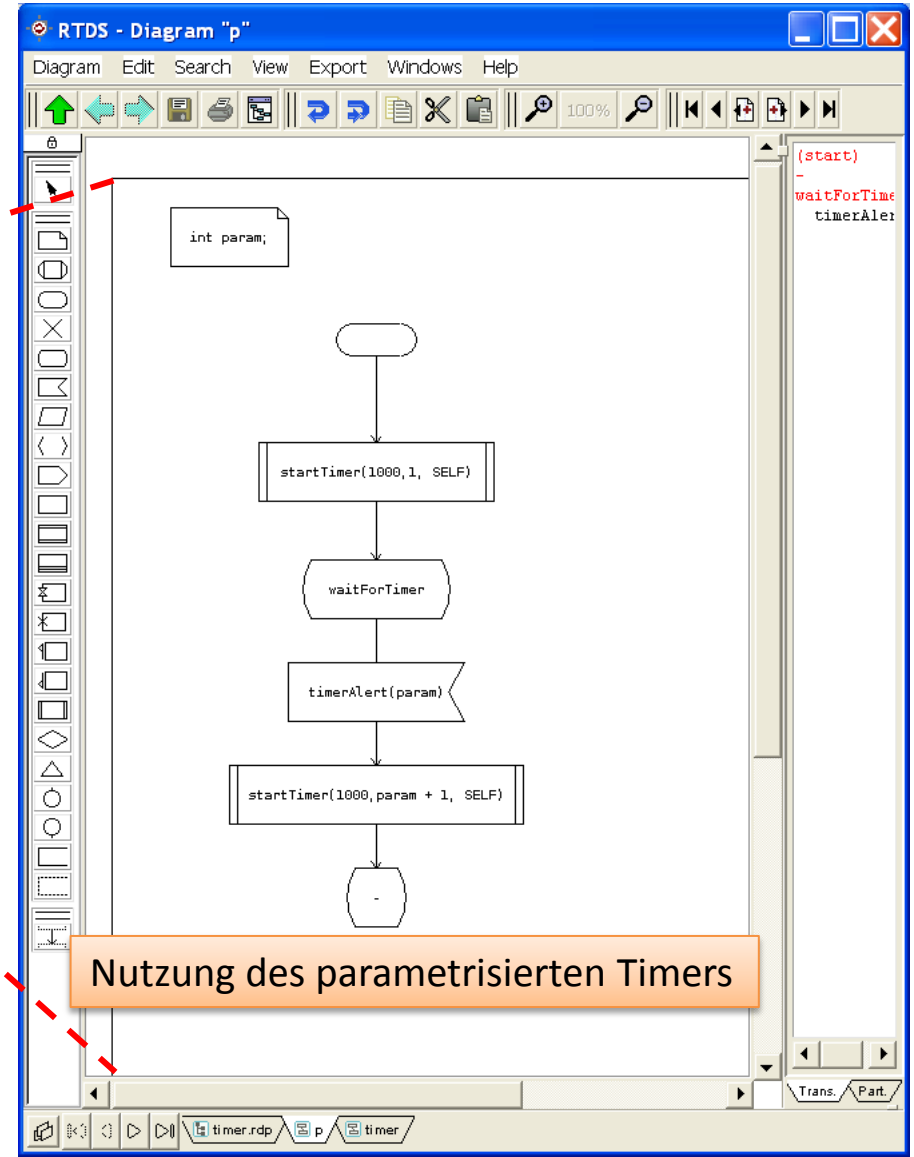
- Z.100-SDL erlaubt Parametrisierung/Indizierung von Timern zur Identifikation (SDL-RT nicht)

- In SDL/RT bleibt nur ein nutzereigenes Ersetzungsmodell

Prozess mit indiziertem Timer-Prozess (1)

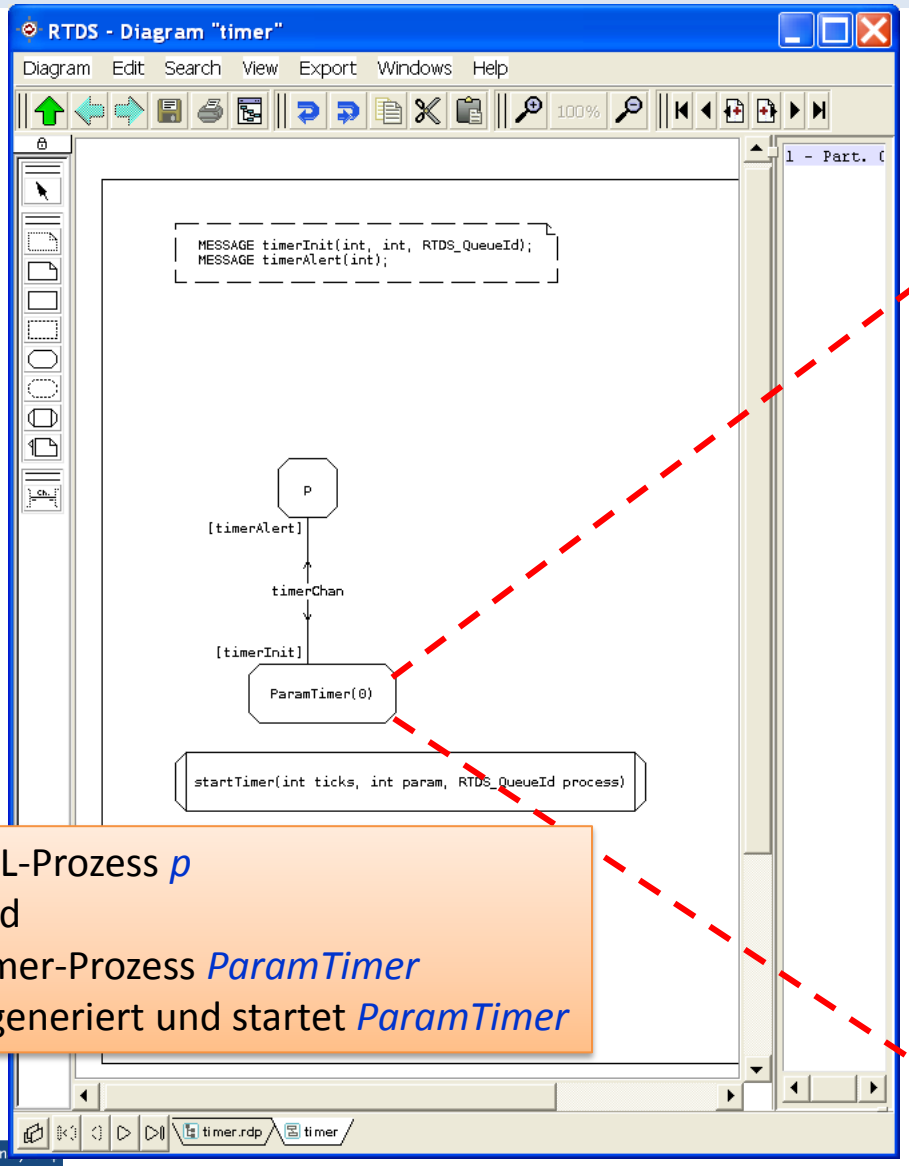


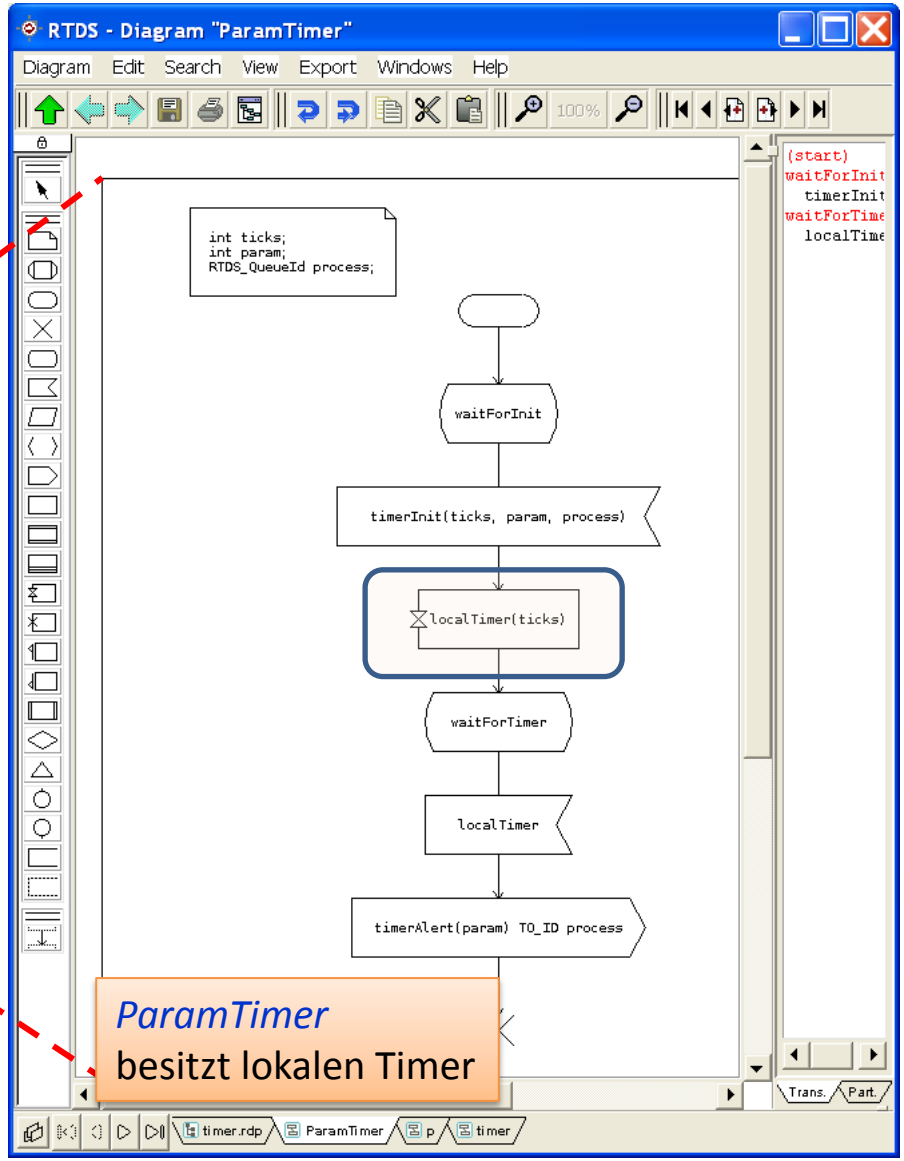
SDL-Prozess *p*
und
Timer-Prozess *ParamTimer*
P generiert und startet *ParamTimer*
mittels Prozedur *startTimer*



Nutzung des parametrisierten Timers

Prozess mit indiziertem Timer-Prozess (2)

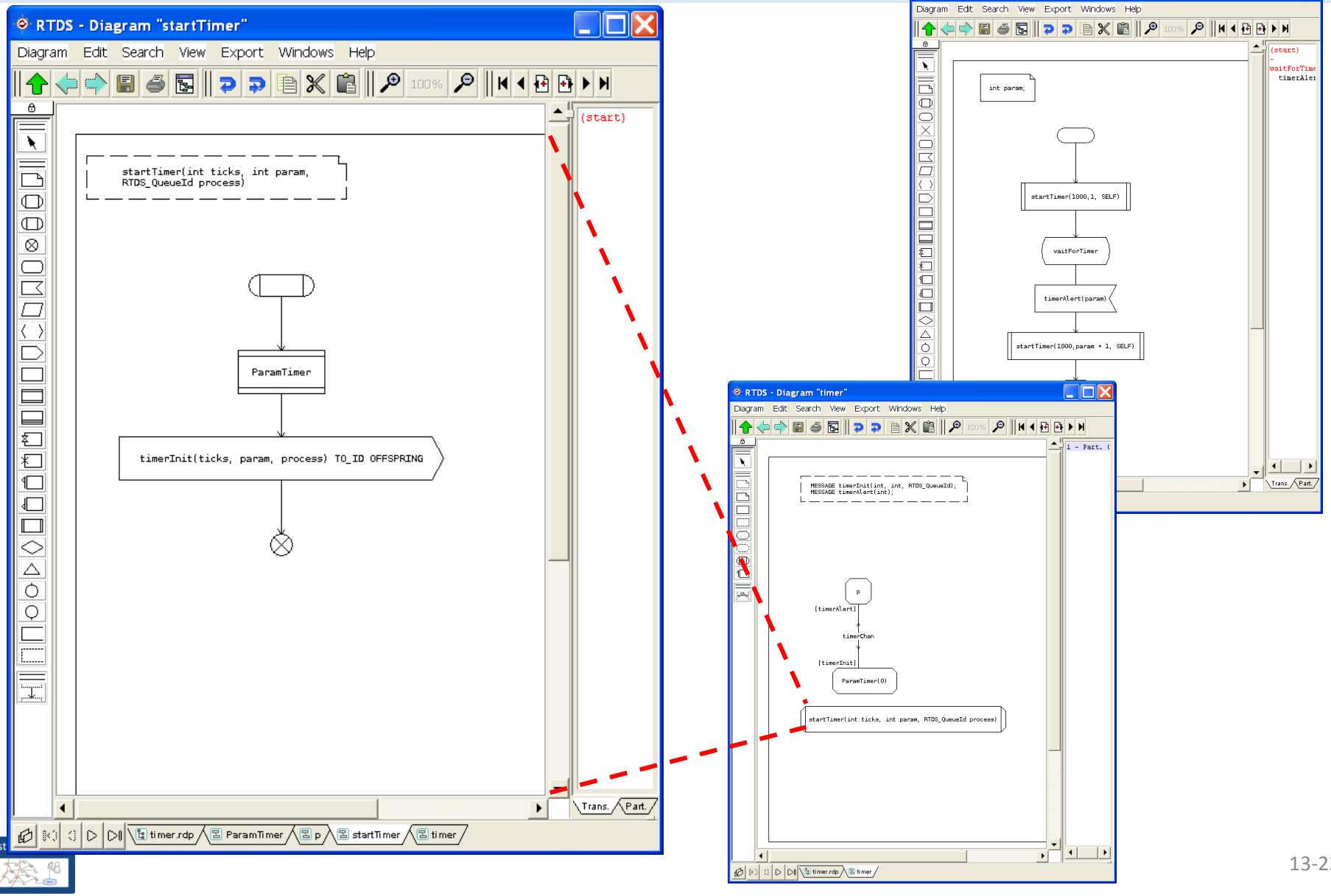




SDL-Prozess *p*
und
Timer-Prozess *ParamTimer*
P generiert und startet *ParamTimer*

ParamTimer
besitzt lokalen Timer

Prozess mit indiziertem Timer-Prozess (3)



7. SDL-Konzepte (Präzisierung)

1. Modellstruktur
2. Einfacher Zustandsautomat: Triggerarten
3. Ersetzungsmodell: Priorisierter Input
4. Nachrichtenadressierung
5. Prozeduren
6. Timer
7. Remote Prozeduren
8. Dynamische Prozessgenerierung
9. Spezialisierung von Zustandsautomaten
10. Lokale Objekte, Semaphore

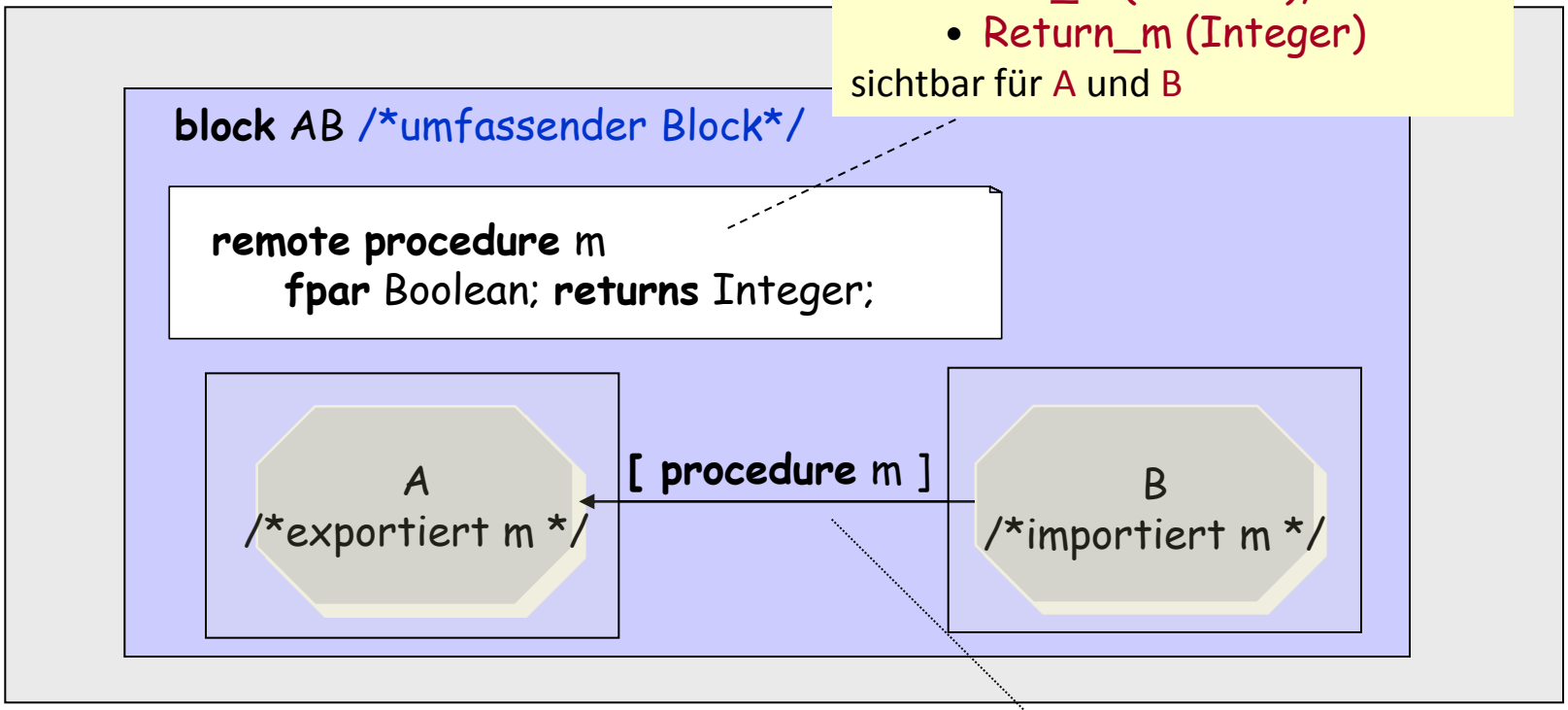
Allgemeines Konzept: Remote- Prozeduren (in Standard-SDL als Ersetzungsmodell)

- **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



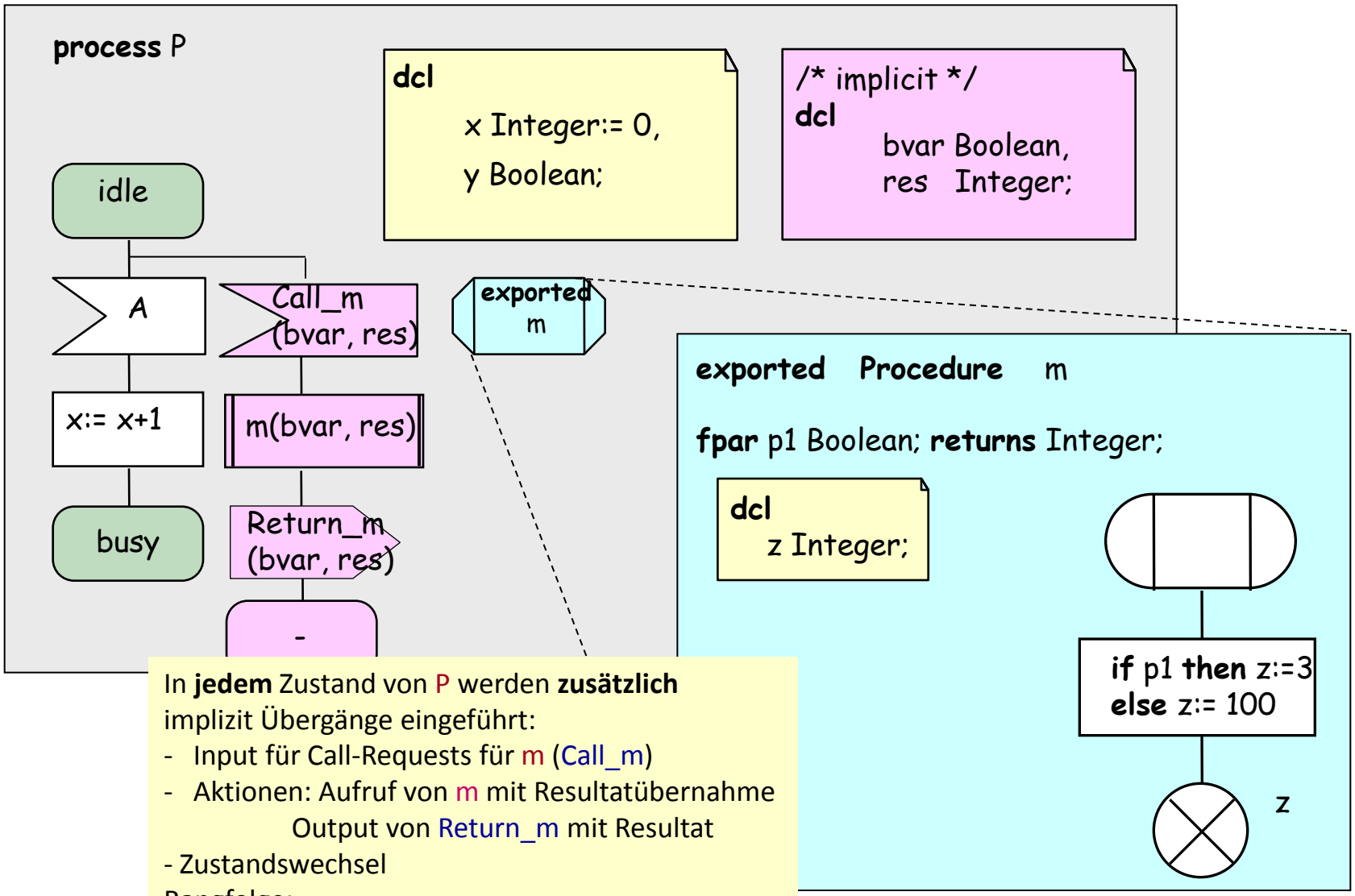
Namen von Remote-Prozeduren können in
- Signallisten,
- Kanälen, Routen, Gates und
- Signalsets

erscheinen (wenn nicht, dann implizit)

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

Exporteur der Prozedur

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

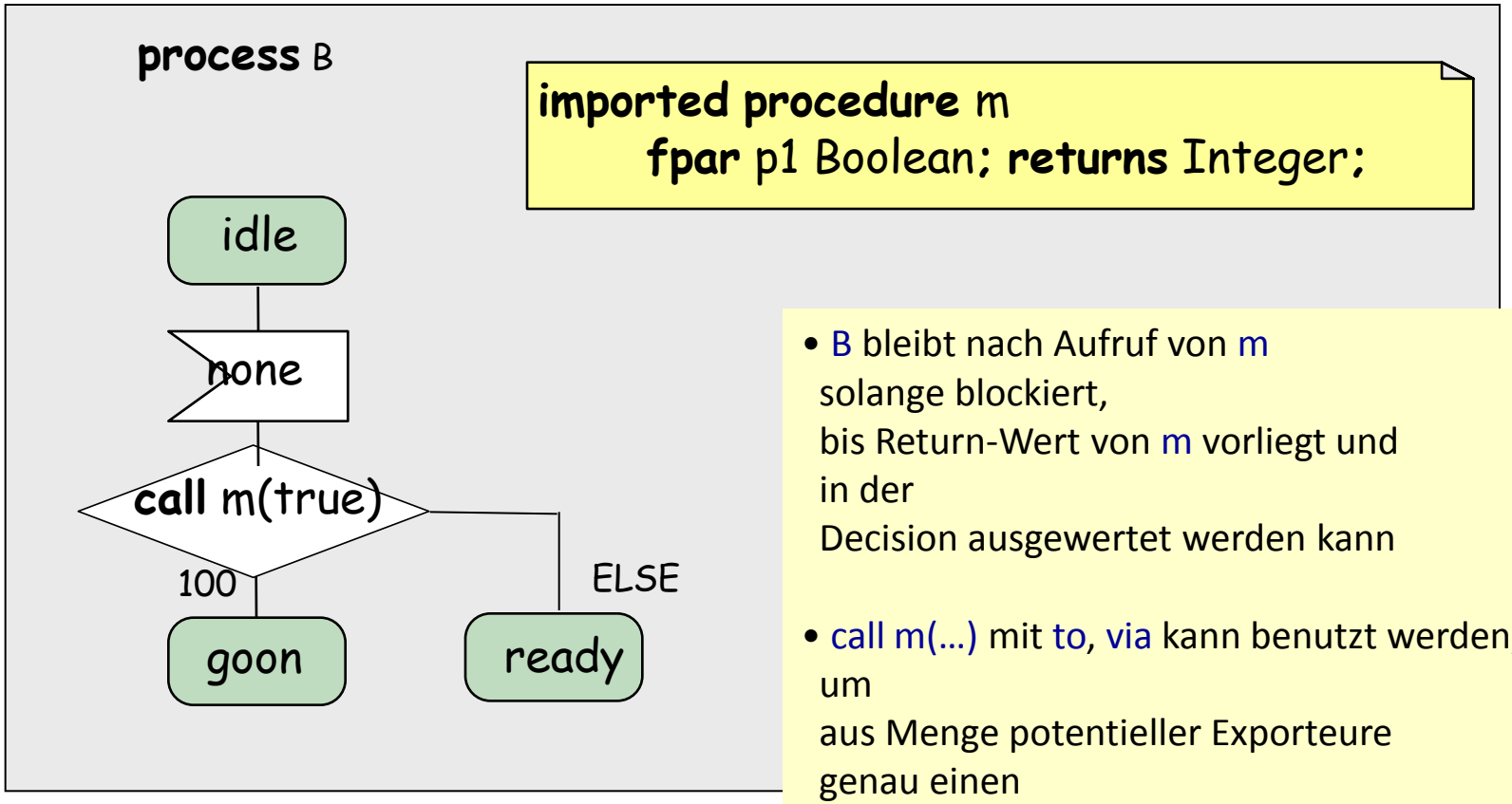


In **jedem** Zustand von P werden **zusätzlich** implizit Übergänge eingeführt:

- 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

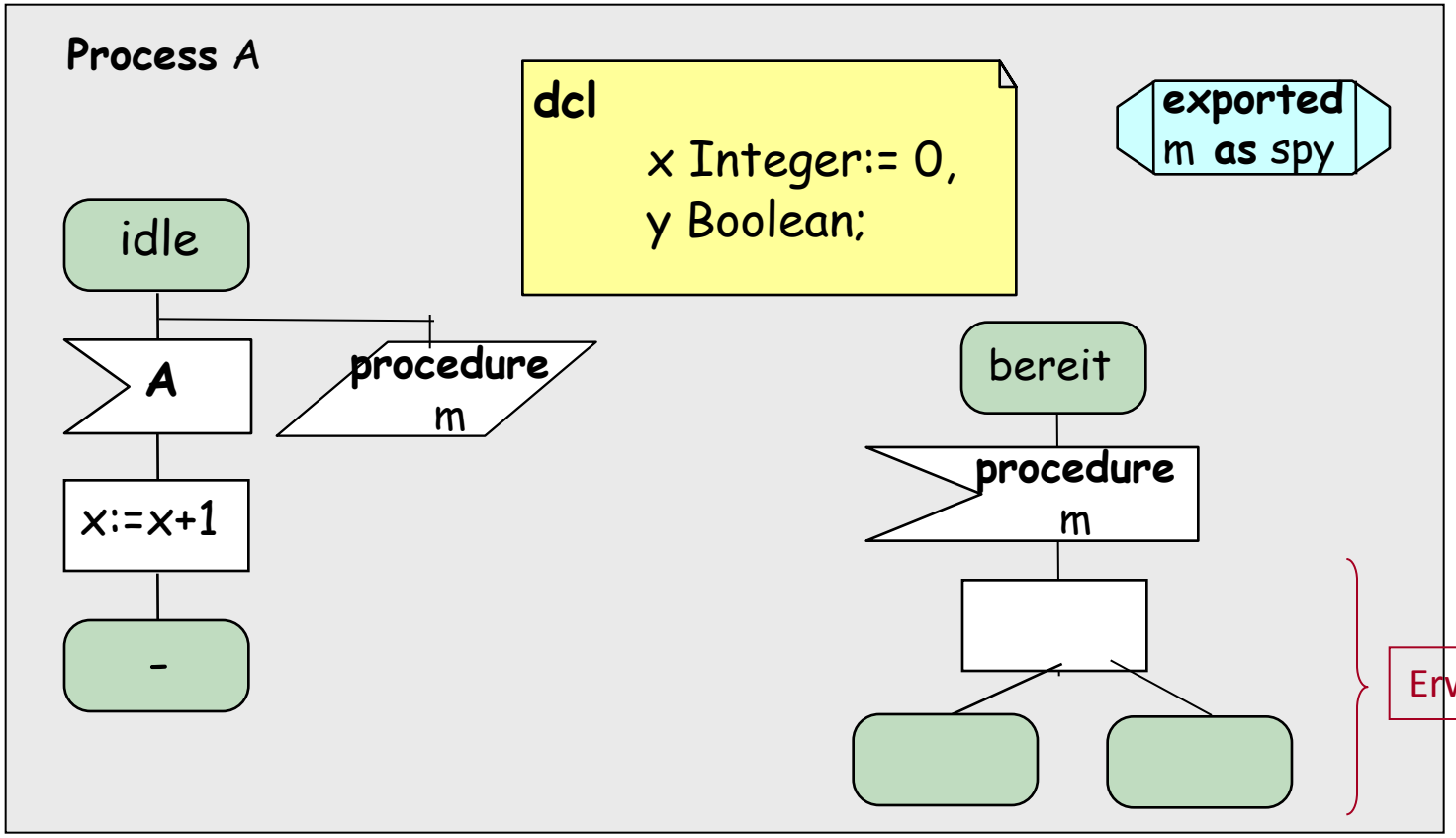
Importeur der Prozedur



- 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)

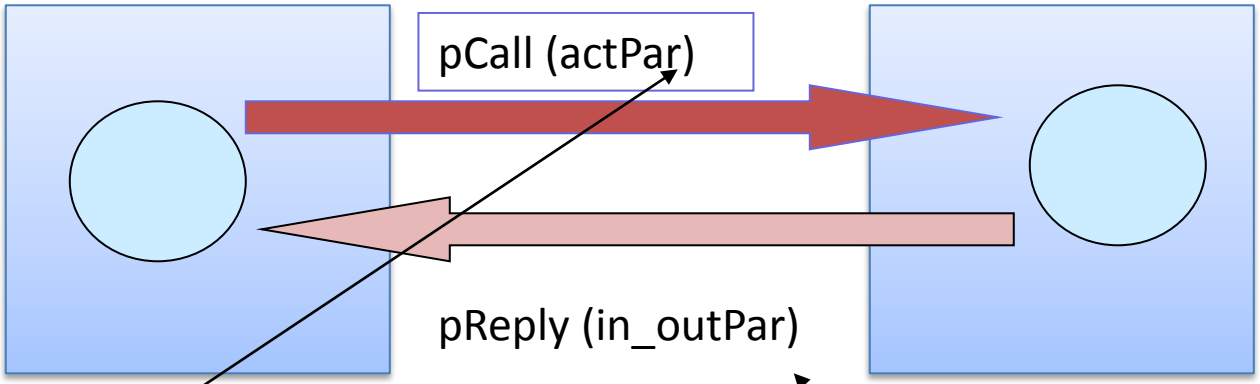
RPC-Ersetzungsmodell

Remote procedure p (…)

1 Aus RPC-Deklaration →
implizite Signaldefinition + implizite biderektionaler Kanal

Client

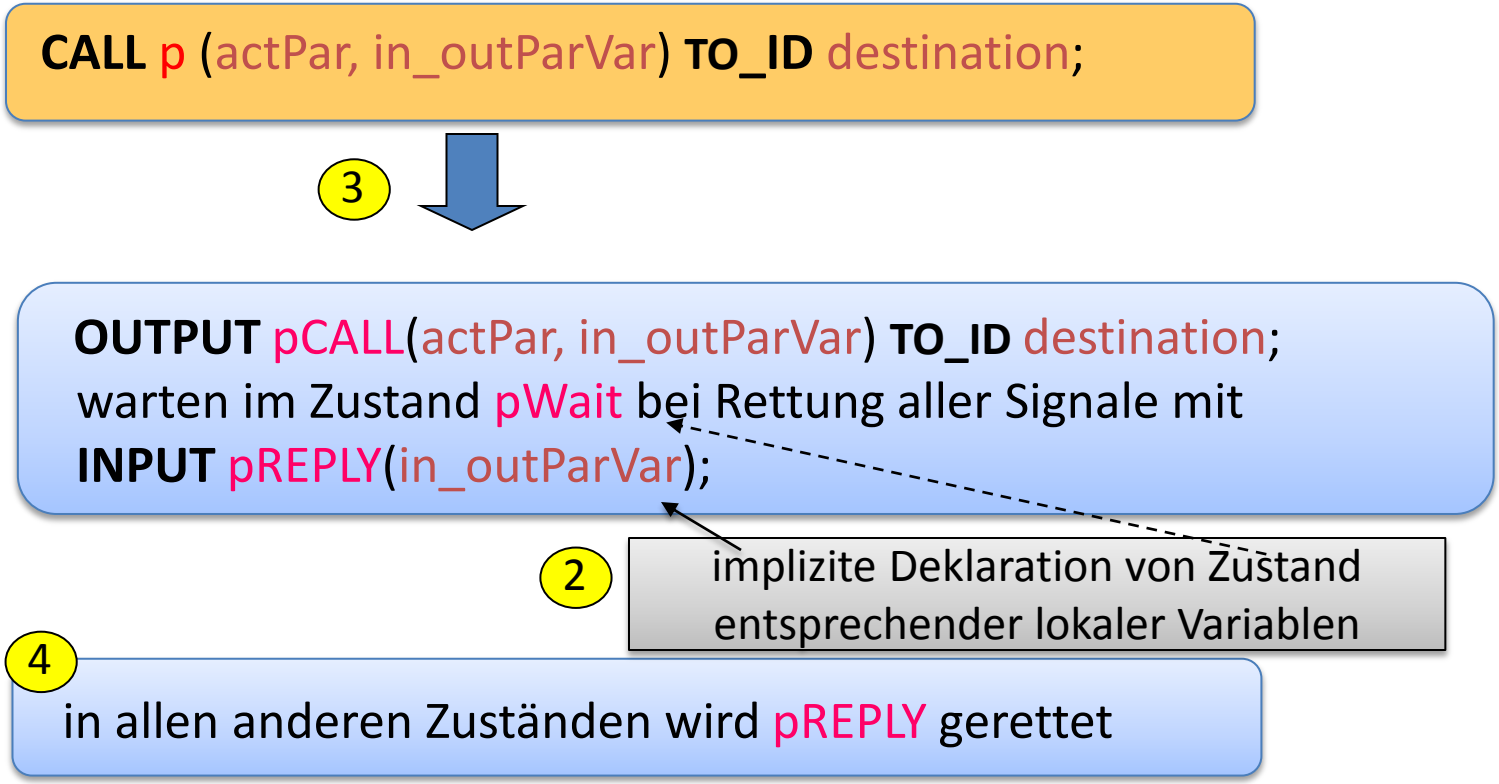
Server



Liste aktueller In/Out-Parameter

Liste aktueller InOut-Parameter (inkl. Return-Wert)

RPC-Ersetzungsmodell des Client



RPC-Ersetzungsmodell des Client

bei allen Zuständen des Servers mit
INPUT procedure p
<Transition>

6

wird wird folg. Input-Teil hinzugefügt

INPUT pCALL(actParVar);
TASK senderVar:= SENDER;
CALL local_p (actParVar, in_outParVar);
OUTPUT pREPLY(in_outParVar) **TO_ID** senderVar;
<Transition>

5 bei impliziter Deklaration entsprechender Variablen

7

bei allen Zuständen des Servers mit
SAVE procedure p
erfolgt eine Ersetzung mit **SAVE** pCALL;

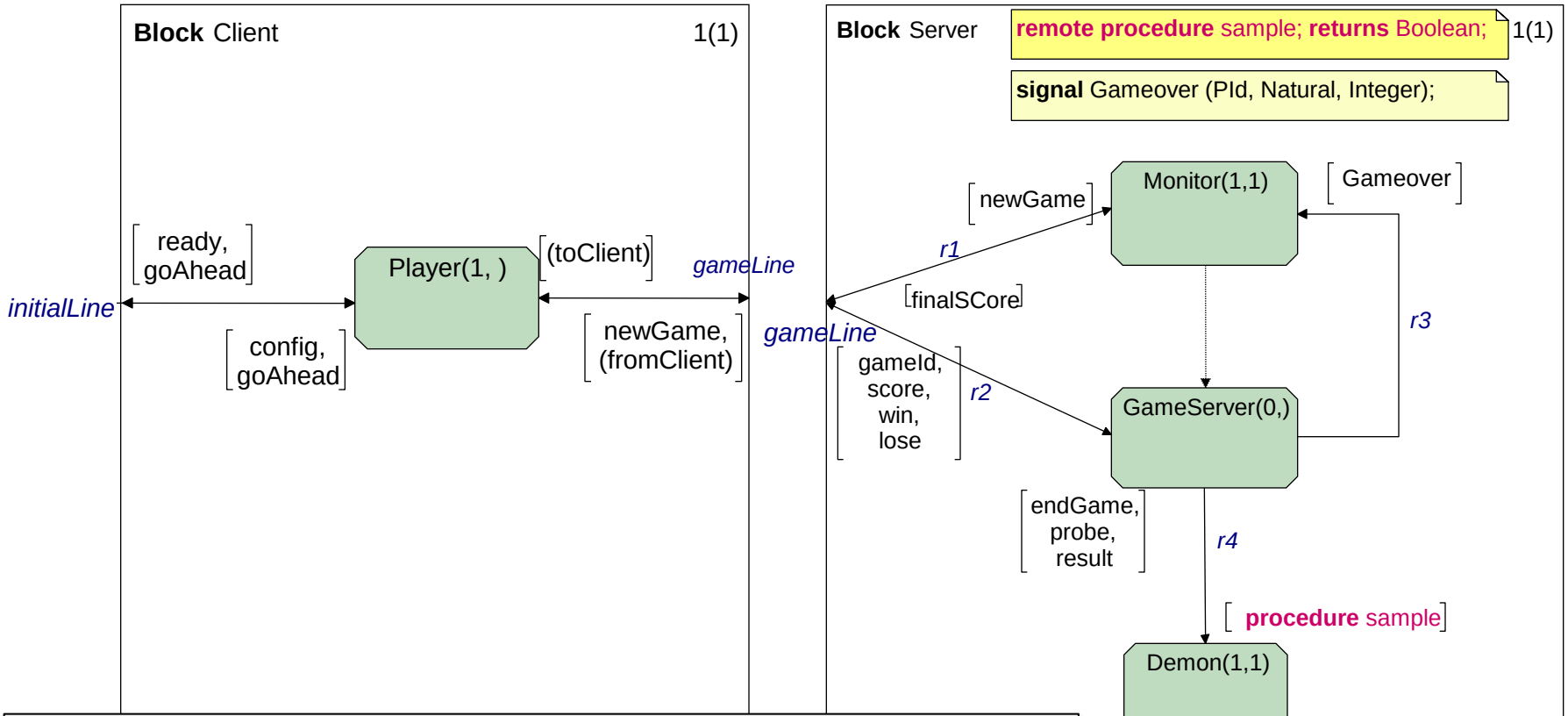
RPC-Ersetzungsmodell des Client

bei allen anderen Zuständen des Servers
(mit Ausnahme der impliziten Zustände)
wird folg. Input-Teil ergänzt:

8

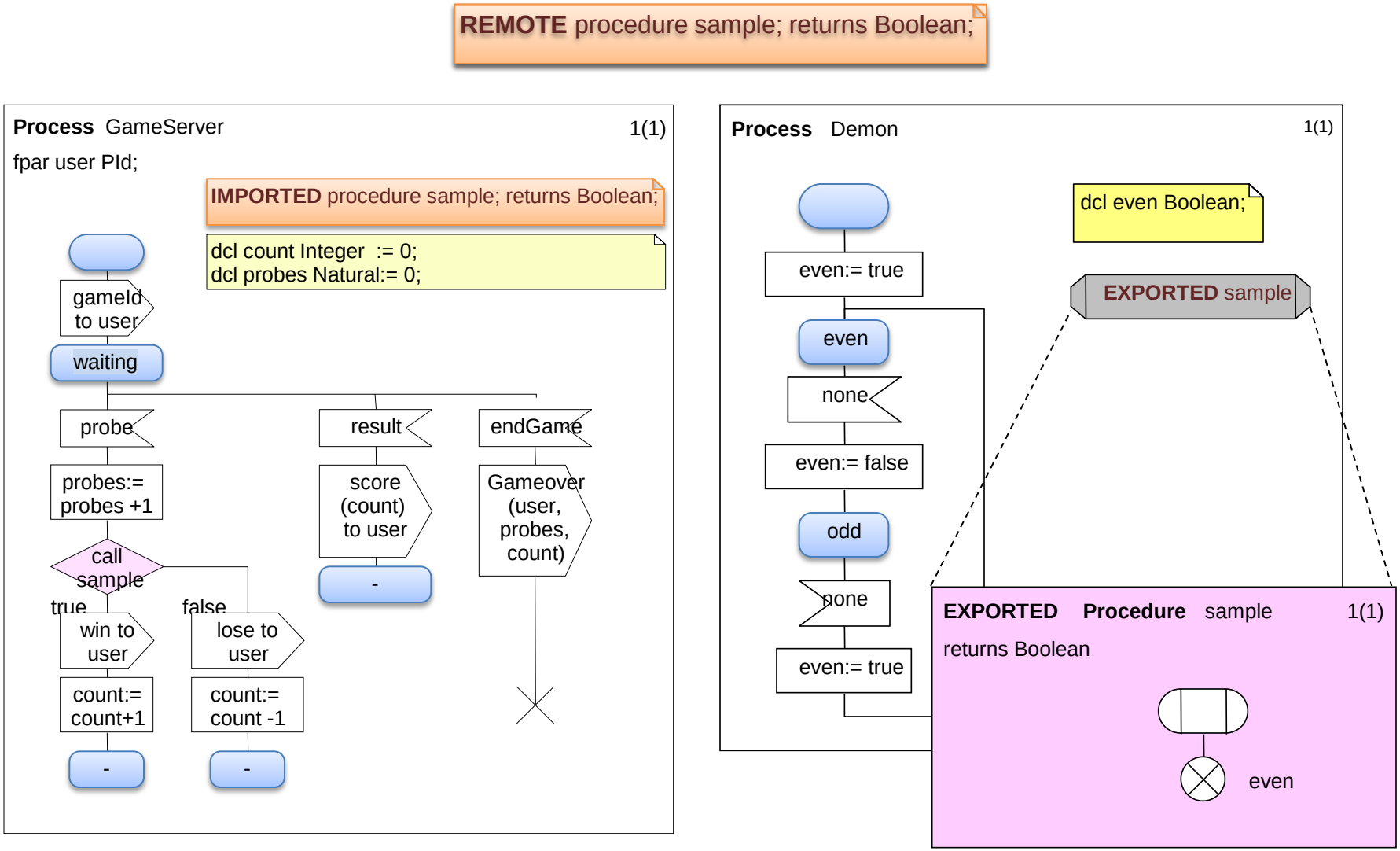
```
INPUT pCALL(actParVar);
TASK senderVar:= SENDER;
CALL local_p (actParVar, in_outParVar);
OUTPUT pREPLY(in_outParVar) TO_ID senderVar;
NEXTSTATE -;
```

RPC-Beispiel: Demon-Game in Standard-SDL

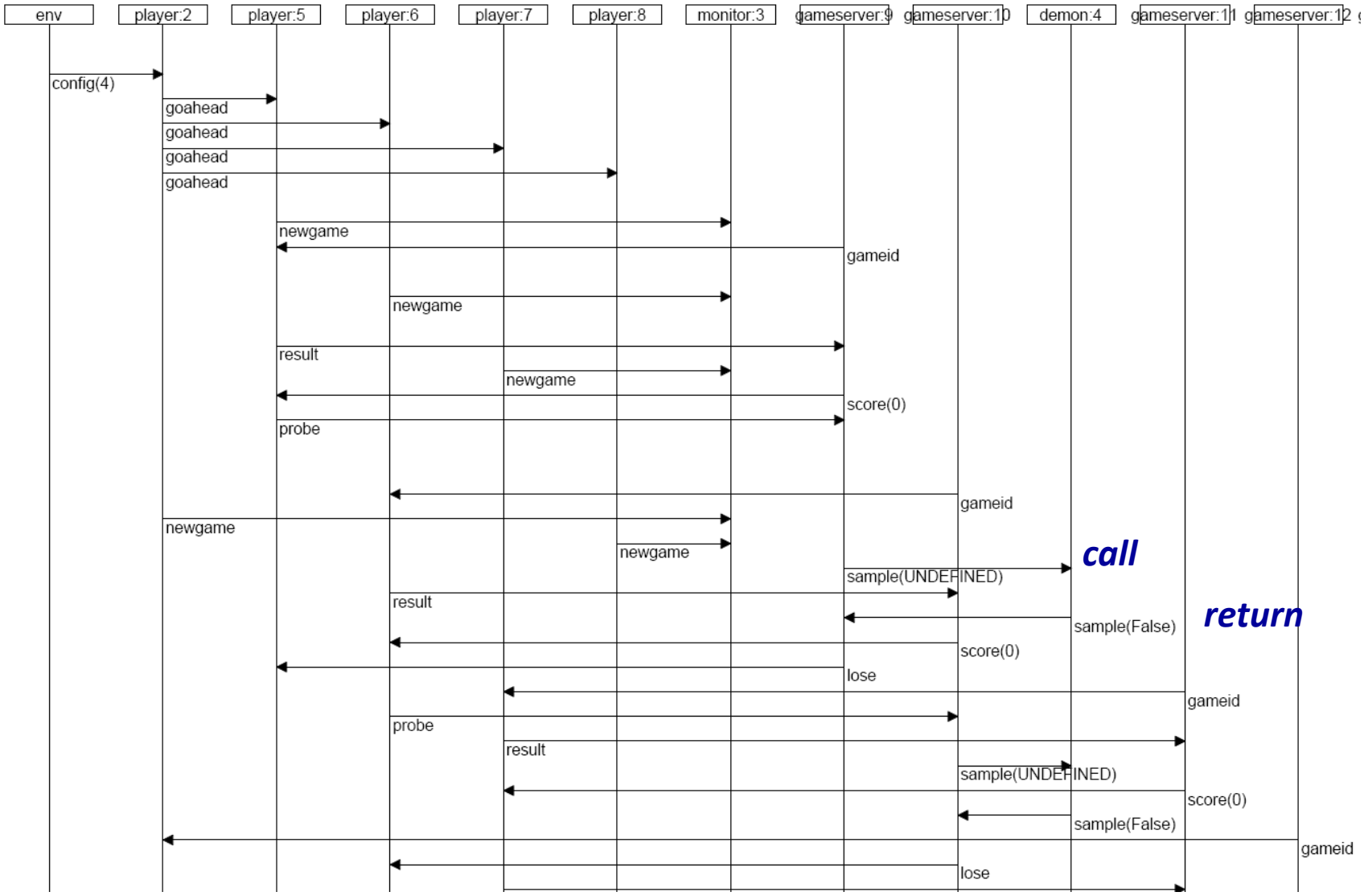


Achtung: Für alle beteiligten Spieler gibt es eine gemeinsame geteilt genutzte Variable (private Prozessvariable von Demon)
Ein gleichzeitiger Zugriff per Remote Prozedur
muss zwangssequentialisiert werden

Dreiklang: Remote – Imported - Exported



Ablauf mit Ruf und Return einer Remote-Prozedur

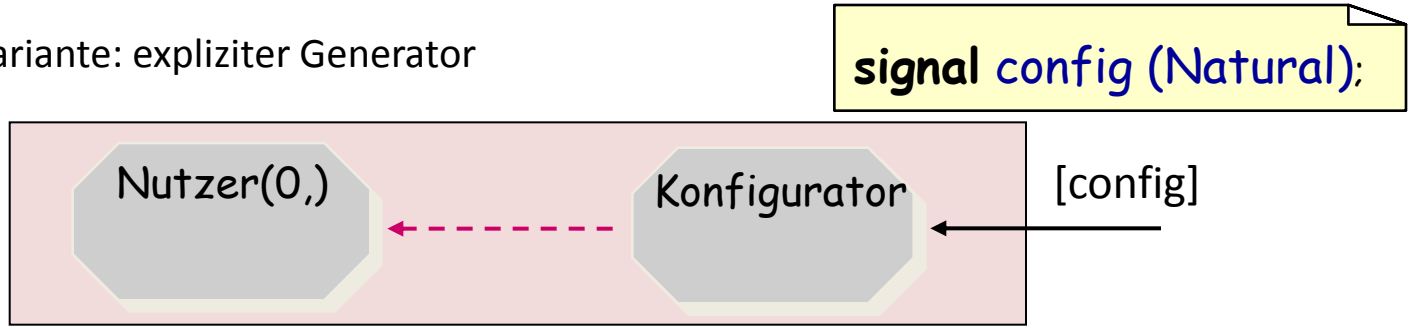


7. SDL-Konzepte (Präzisierung)

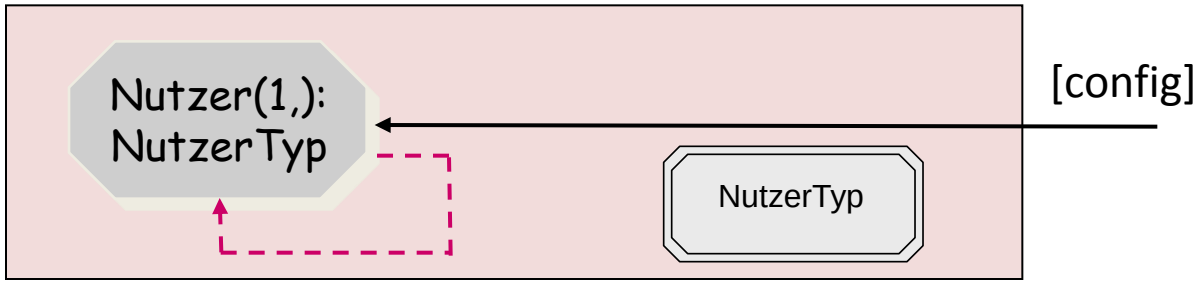
1. Modellstruktur
2. Einfacher Zustandsautomat: Triggerarten
3. Ersetzungsmodell: Priorisierter Input
4. Nachrichtenadressierung
5. Prozeduren
6. Timer
7. Remote Prozeduren
8. Dynamische Prozessgenerierung
9. Spezialisierung von Zustandsautomaten
10. Lokale Objekte, Semaphore

Dynamische Konfiguration von Prozessen

- 1. Variante: expliziter Generator



- 2. Variante: Nutzertyp-Instanz ist in der Lage, Kopien von sich selbst in der jeweiligen Instanzmenge (hier Nutzer) zu erzeugen

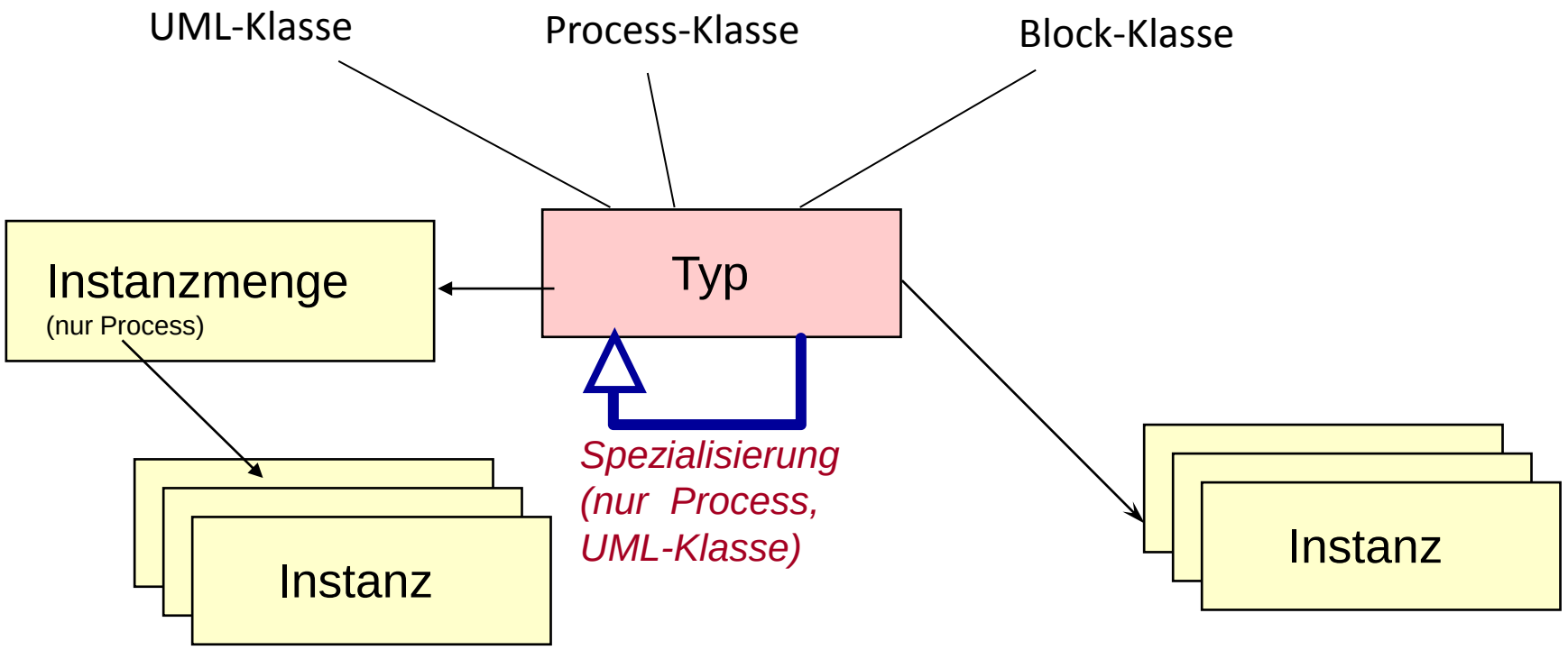


- 3. Variante: Nutzer ist in der Lage, Kopien von sich selbst zu erzeugen

7. SDL-Konzepte (Präzisierung)

1. Modellstruktur
2. Einfacher Zustandsautomat: Triggerarten
3. Ersetzungsmodell: Priorisierter Input
4. Nachrichtenadressierung
5. Prozeduren
6. Timer
7. Remote Prozeduren
8. Dynamische Prozessgenerierung
9. Spezialisierung von Zustandsautomaten
10. Lokale Objekte, Semaphore

Typen, parametrisierte Typen, Spezialisierung



Vererbung in SDL/RT stark eingeschränkt:
Process Typ und UML-Klasse

Spezialisierung (Vererbung)

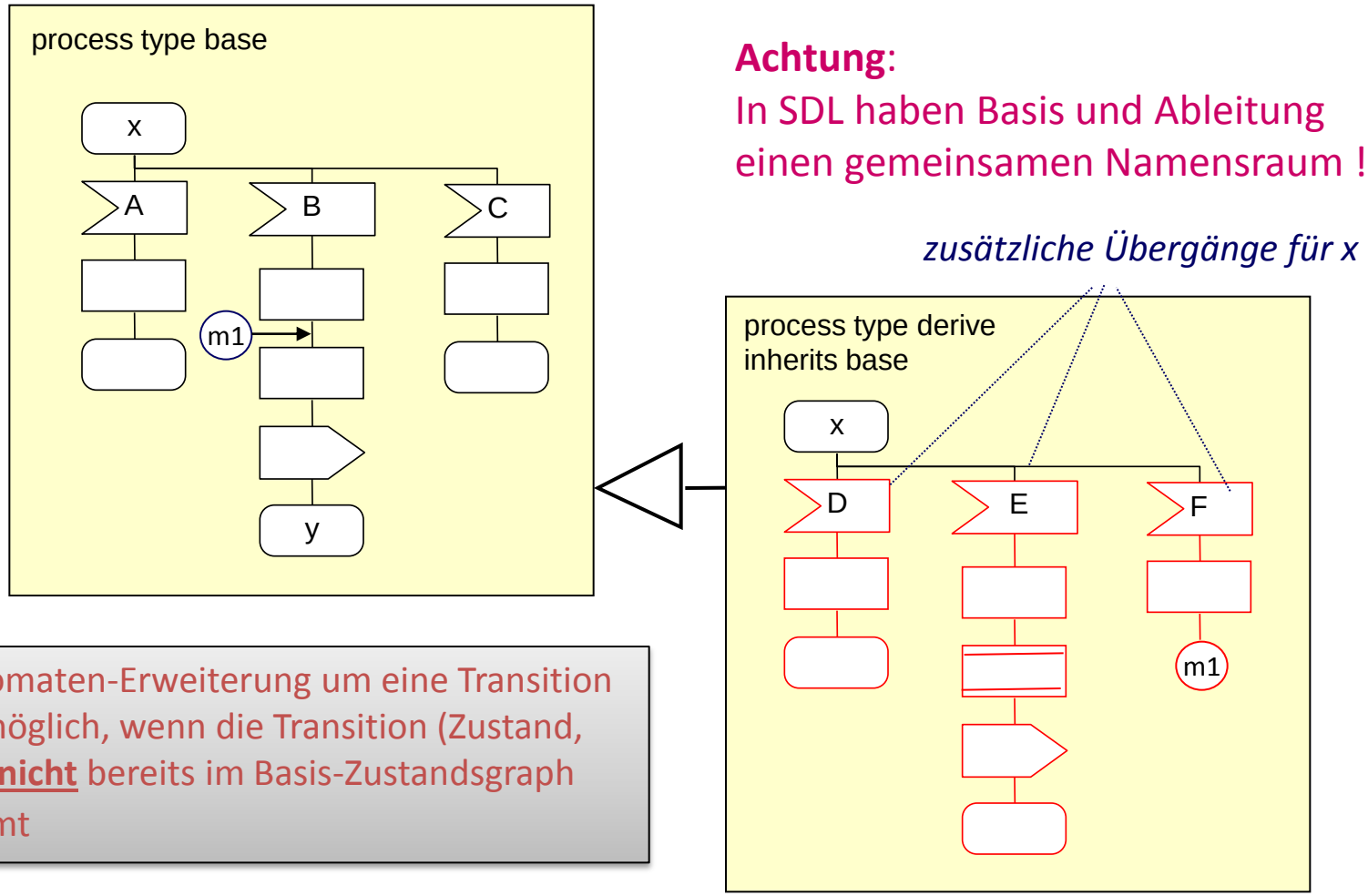
- Mechanismus,
  um einen neuen Typ von einem bereits existenten Typ (Basistyp) abzuleiten

- dabei können Konzepte des Basistyps
  - übernommen (geerbt)
  - modifiziert und
  - erweitert

  werden

- der Basistyp legt fest, was modifiziert werden kann und was nicht
  - **virtual** kennzeichnet Urdefinition und kann redefiniert werden
  - **redefined** kennzeichnet Redefinition und kann **erneut** redefiniert werden
  - **finalized** kennzeichnet Redefinition und kann **nicht mehr** redefiniert werden

Process-Typ-Spezialisierung: Erweiterung des Zustandsgraphen



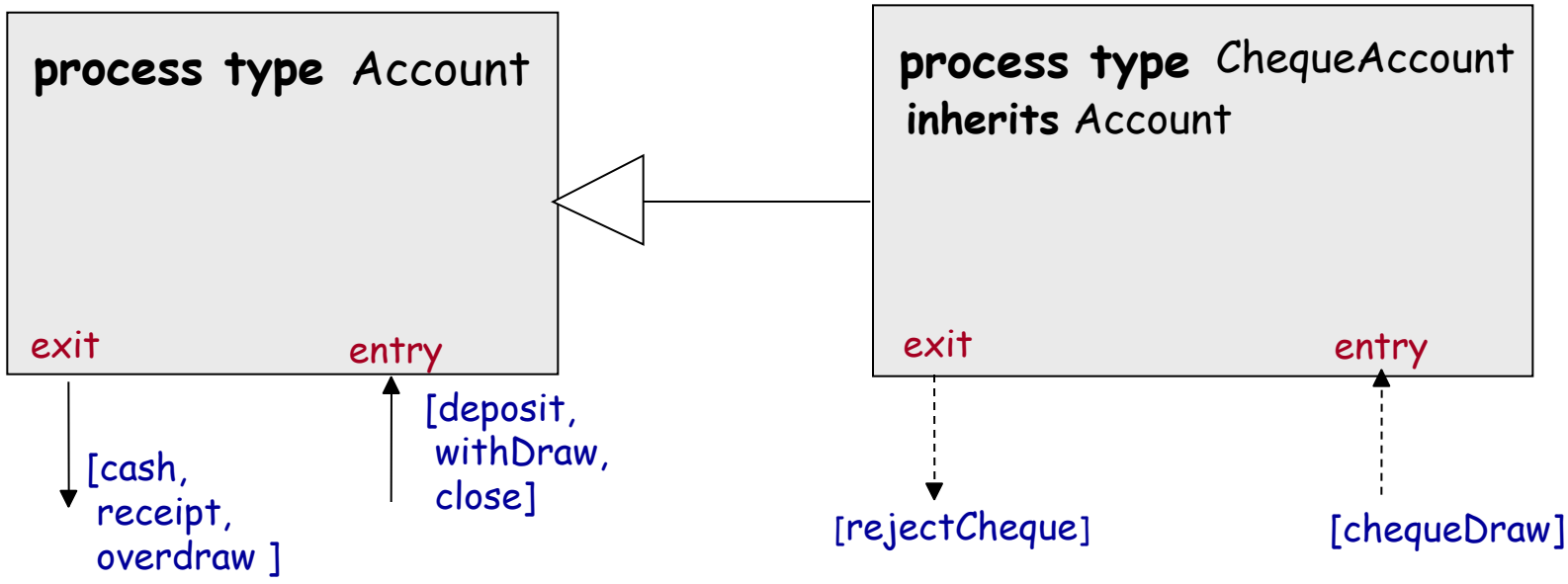
Achtung:
In SDL haben Basis und Ableitung einen gemeinsamen Namensraum !!!

Die Automaten-Erweiterung um eine Transition ist nur möglich, wenn die Transition (Zustand, Trigger) **nicht** bereits im Basis-Zustandsgraph vorkommt

Process-Typ-Spezialisierung: Vererbungsbeispiel

Basistyp

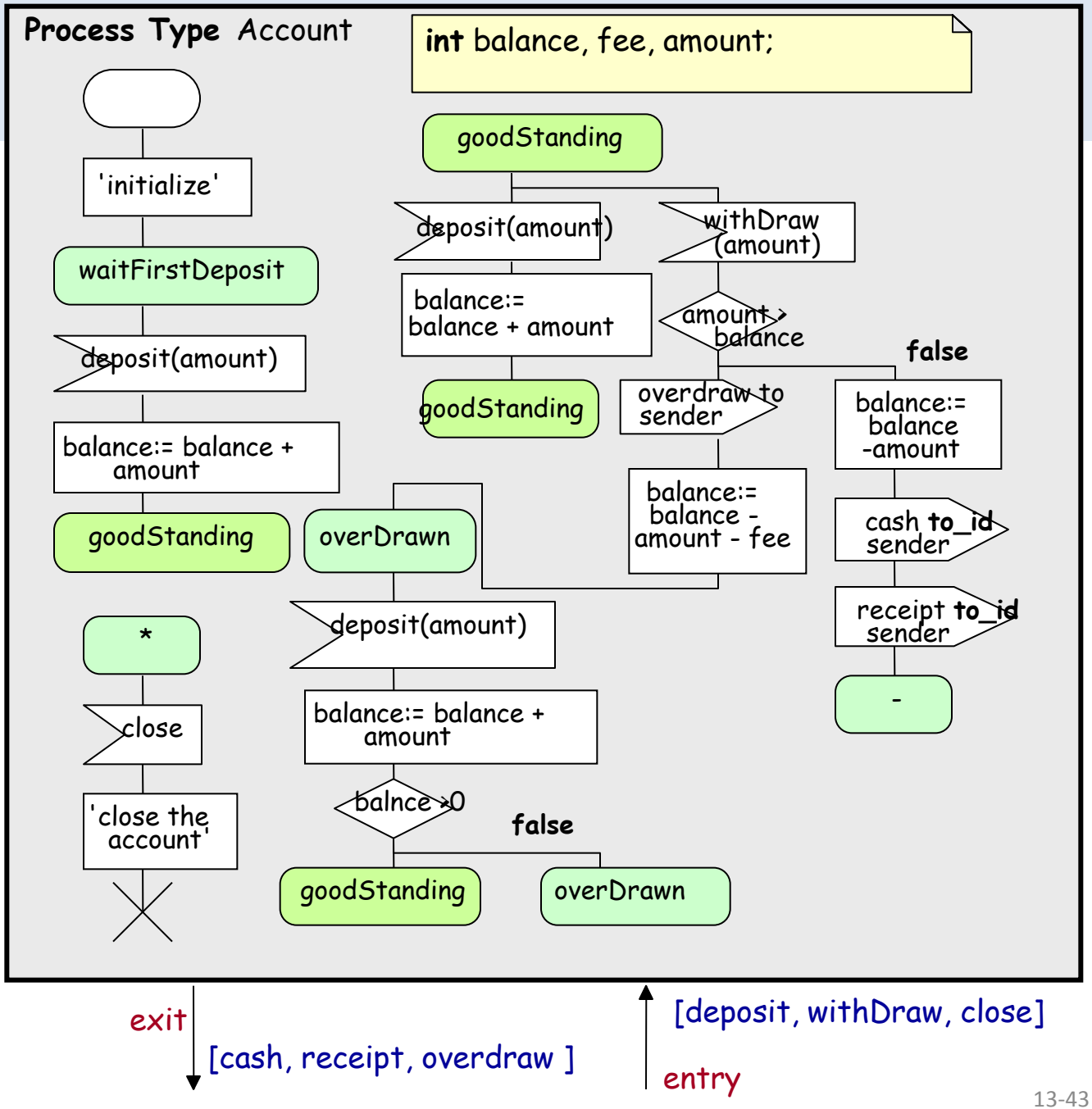
Spezialisierung/ abgeleiteter Typ (mit Gate-Erweiterung)



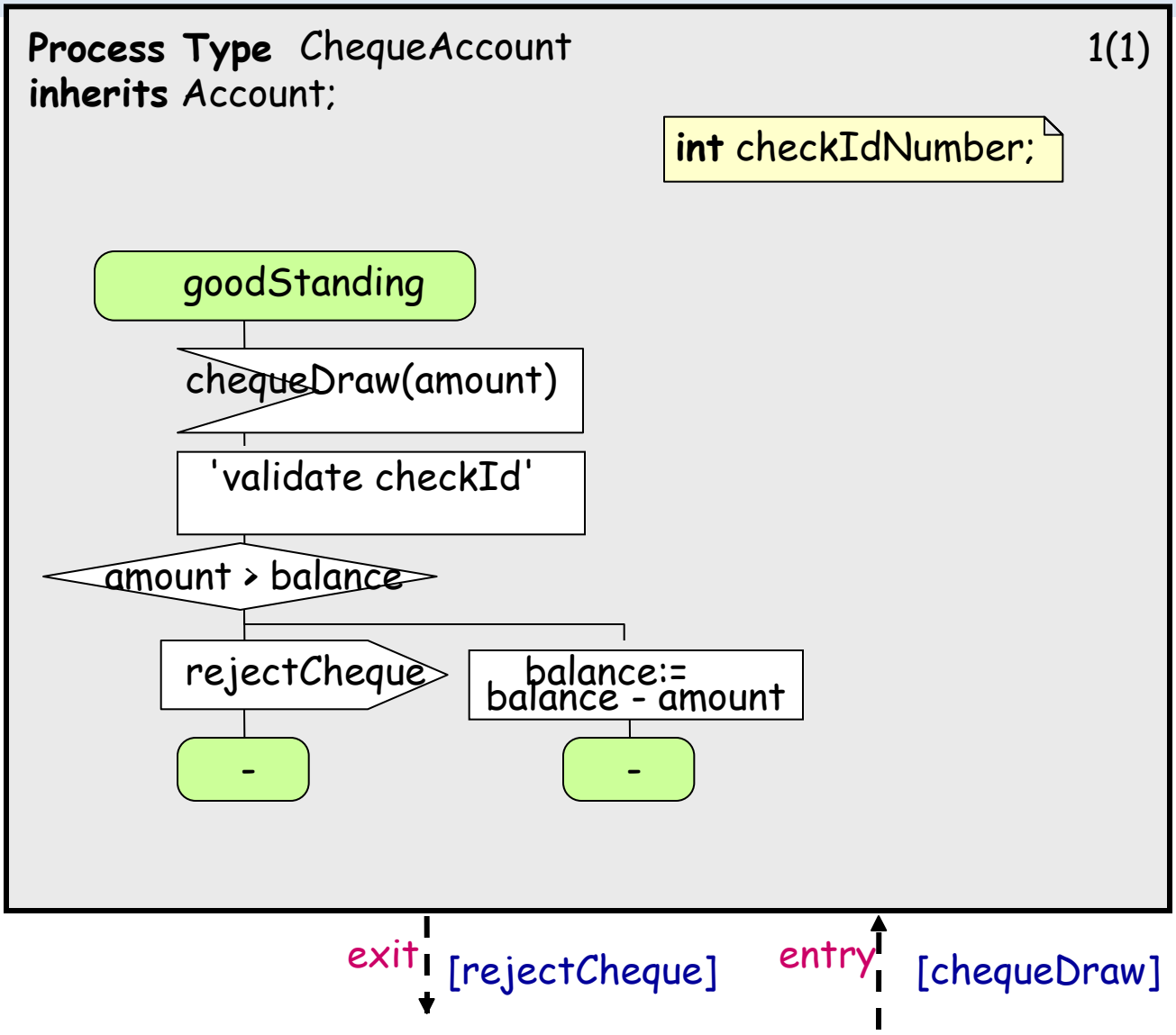
Account- Funktionalität
- Einzahlen
- Abheben → (Cash,Quittung) | (Überziehungsmitteilung)
- Konto schließen

zusätzliche ChequeAccount-Funktionalität
- Überweisung→ (Annullierung)

Basistyp

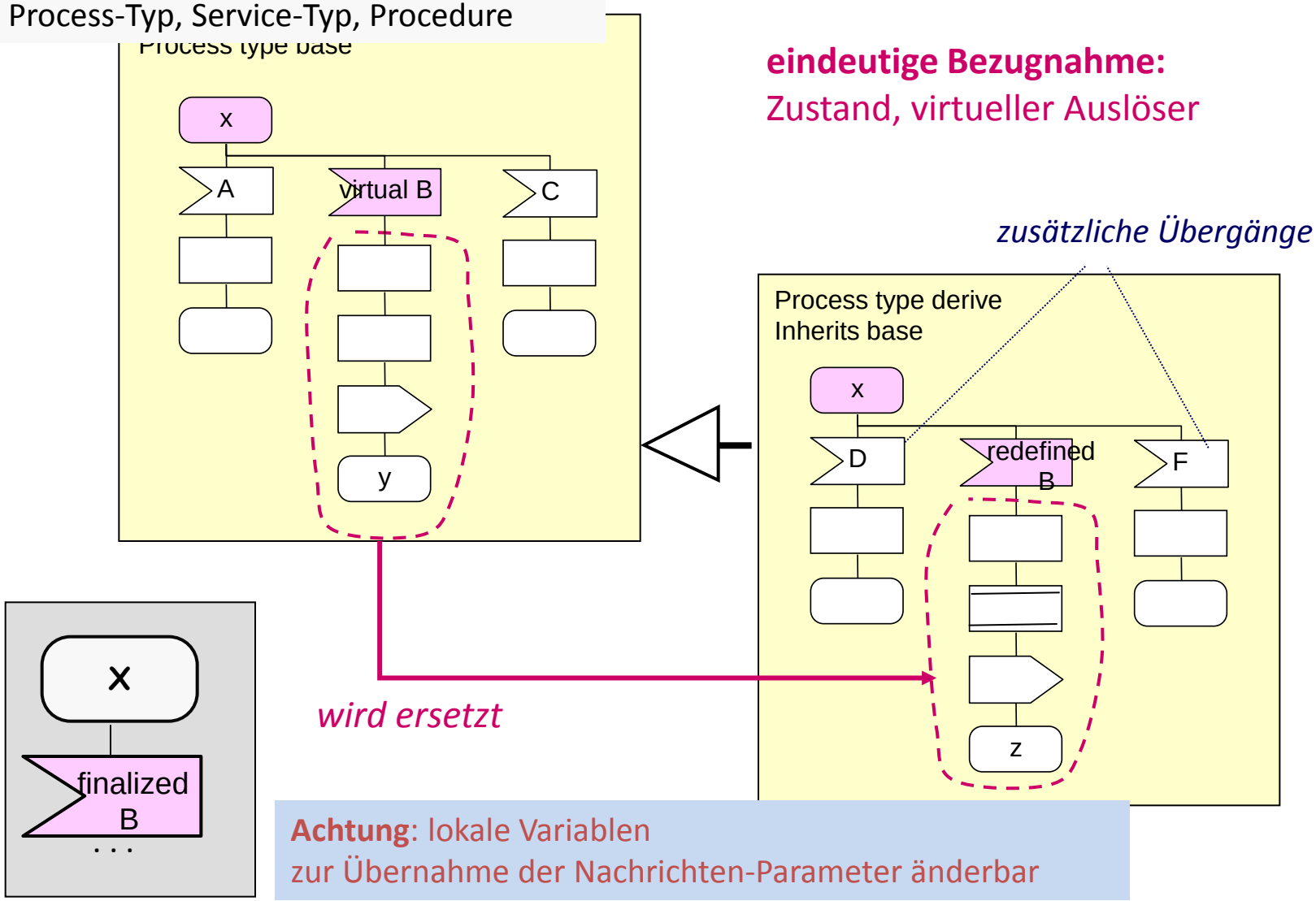

Process Type Account
int balance, fee, amount;
'initialize'
waitFirstDeposit
deposit(amount)
balance:= balance + amount
goodStanding
goodStanding
deposit(amount)
balance:= balance + amount
withDraw (amount)
amount > balance
false
overdraw to sender
balance:= balance -amount
goodStanding
overDrawn
balance:= balance - amount - fee
cash to_id sender
receipt to_id sender
-
*
close
'close the account'
deposit(amount)
balance:= balance + amount
balnce >0
false
goodStanding
overDrawn
exit
[cash, receipt, overdraw]
[deposit, withDraw, close]
entry

Ableitung

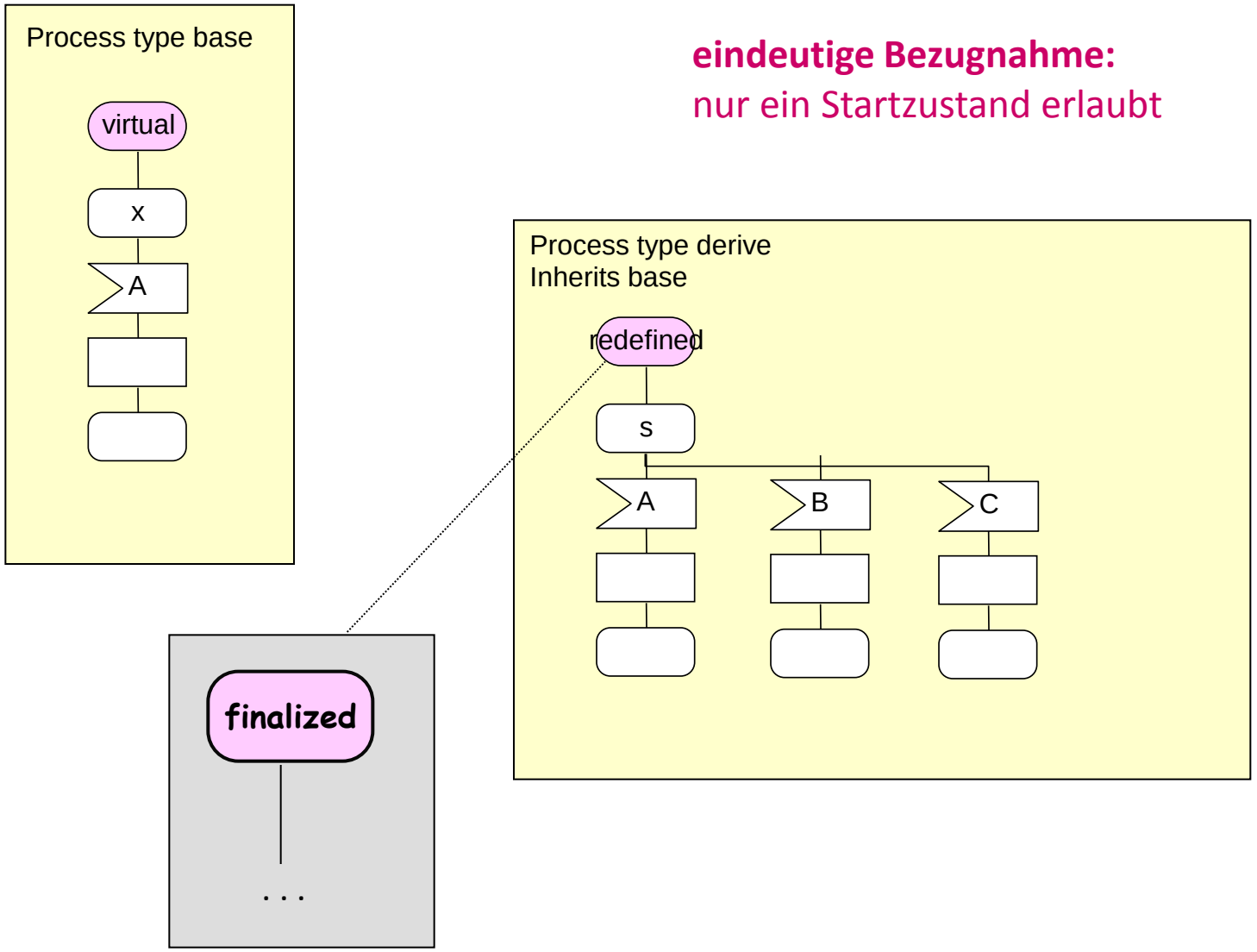

Process Type ChequeAccount
inherits Account;
1(1)
int checkIdNumber;
goodStanding
chequeDraw(amount)
'validate checkId'
amount > balance
rejectCheque
balance:=
balance - amount
-
-
exit
[rejectCheque]
entry
[chequeDraw]

Redefinition einzelner Transitionen: Prinzip

gilt für Process-Typ, Service-Typ, Procedure

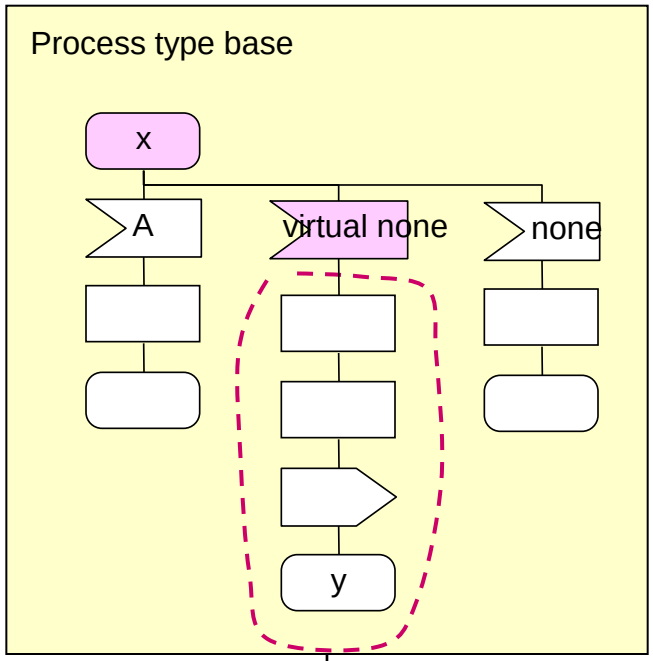


Redefinition des gesamten Zustandsautomaten



eindeutige Bezugnahme:
nur ein Startzustand erlaubt

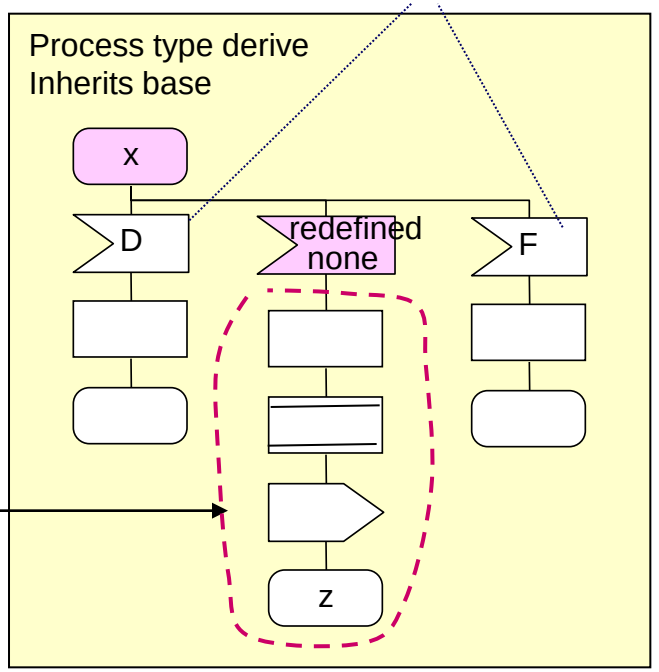
Redefinition spontaner Transitionen



eindeutige Auflösung
macht Einschränkung erforderlich:

→ pro Zustand ist im Basistyp **nur eine** virtuelle spontane Transition erlaubt

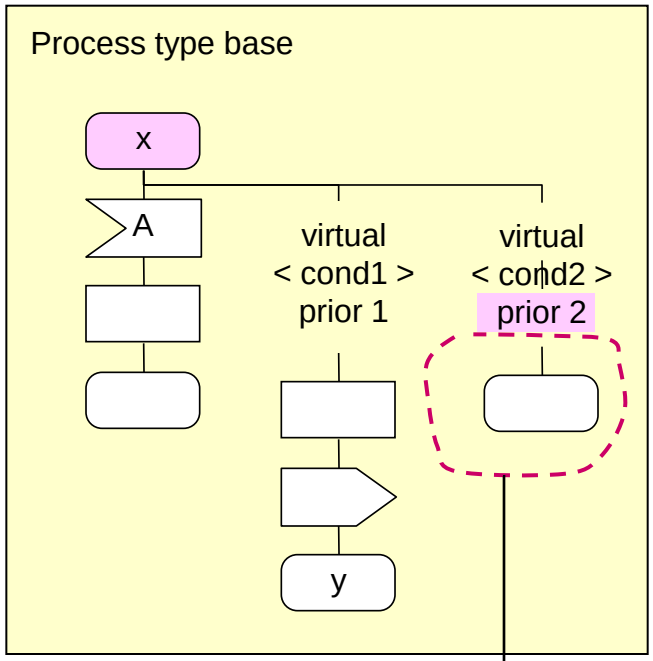
zusätzliche Übergänge



wird ersetzt

nicht in SDL/RT !

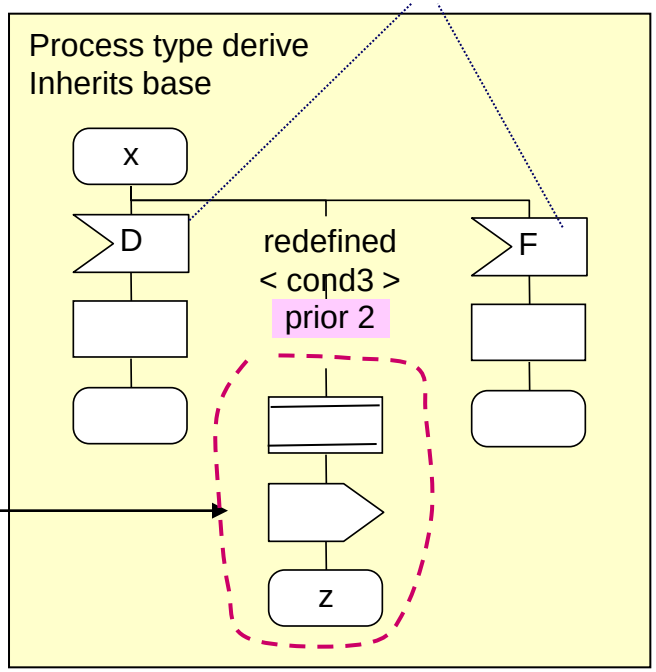
Redefinition von Continous-Transitionen



eindeutige Auflösung
macht Einschränkung erforderlich:

→ pro Zustand müssen sich im Basistyp die virtuellen Continuous-Transitionen in der **Priorität** unterscheiden

zusätzliche Übergänge



wird ersetzt,
sogar die Bedingung
kann sich ändern !

Flexible Redefinition von Transitionen mit Signal-Triggern

- ein virtueller Trigger für einen Zustand S eines Basis-Process-Typs X der folgenden Art

  - input virtual *sig*
  - save virtual *sig*
  - ~~priority input virtual *sig*~~
  - ~~input virtual *sig* provided <expr-1>~~

(Zustandsname, Signalname)

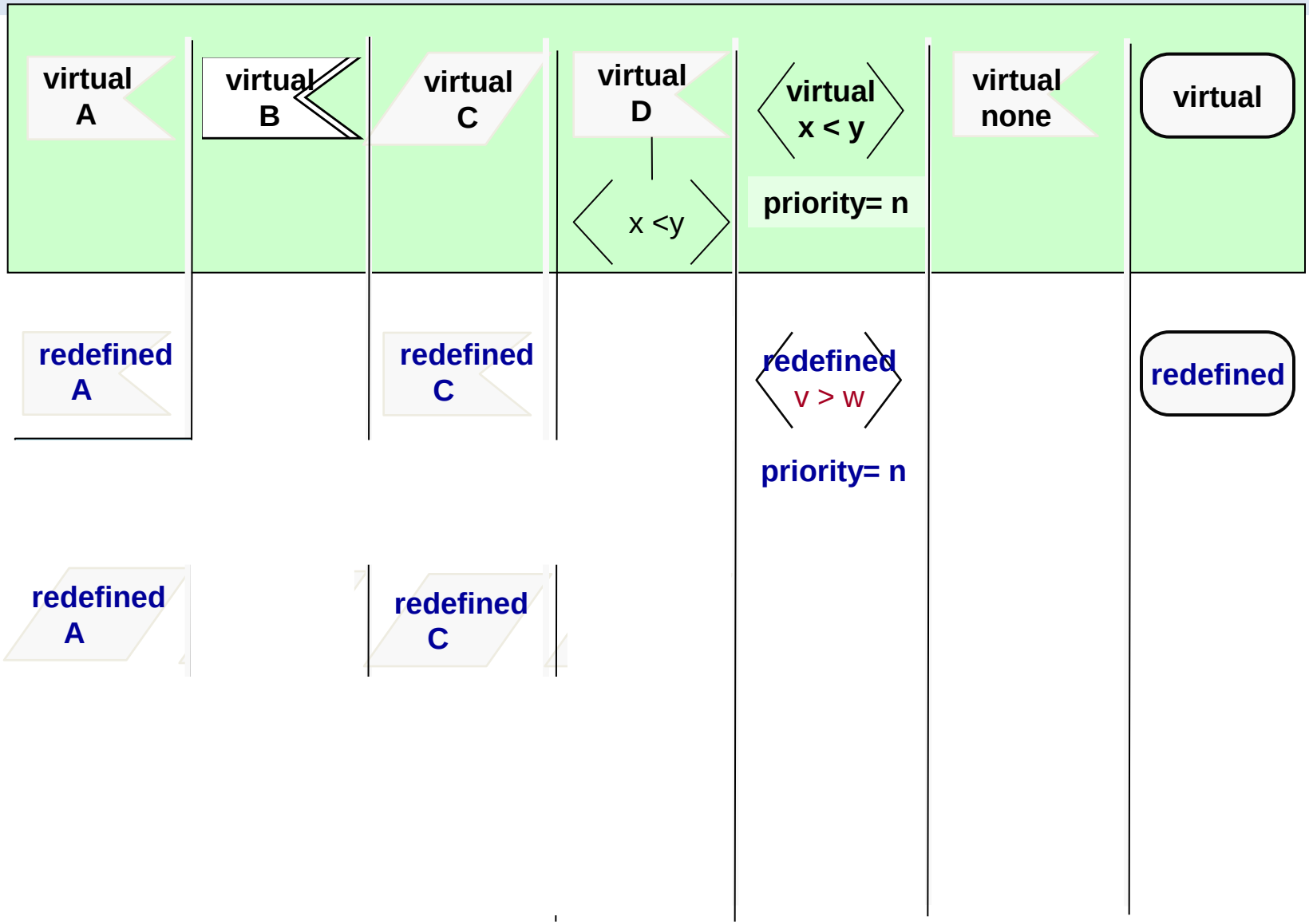
1-deutige Zuordnung

- kann in einer Ableitung von X im Zustand S zu einem beliebigen Trigger der folgenden Art umfunktioniert werden

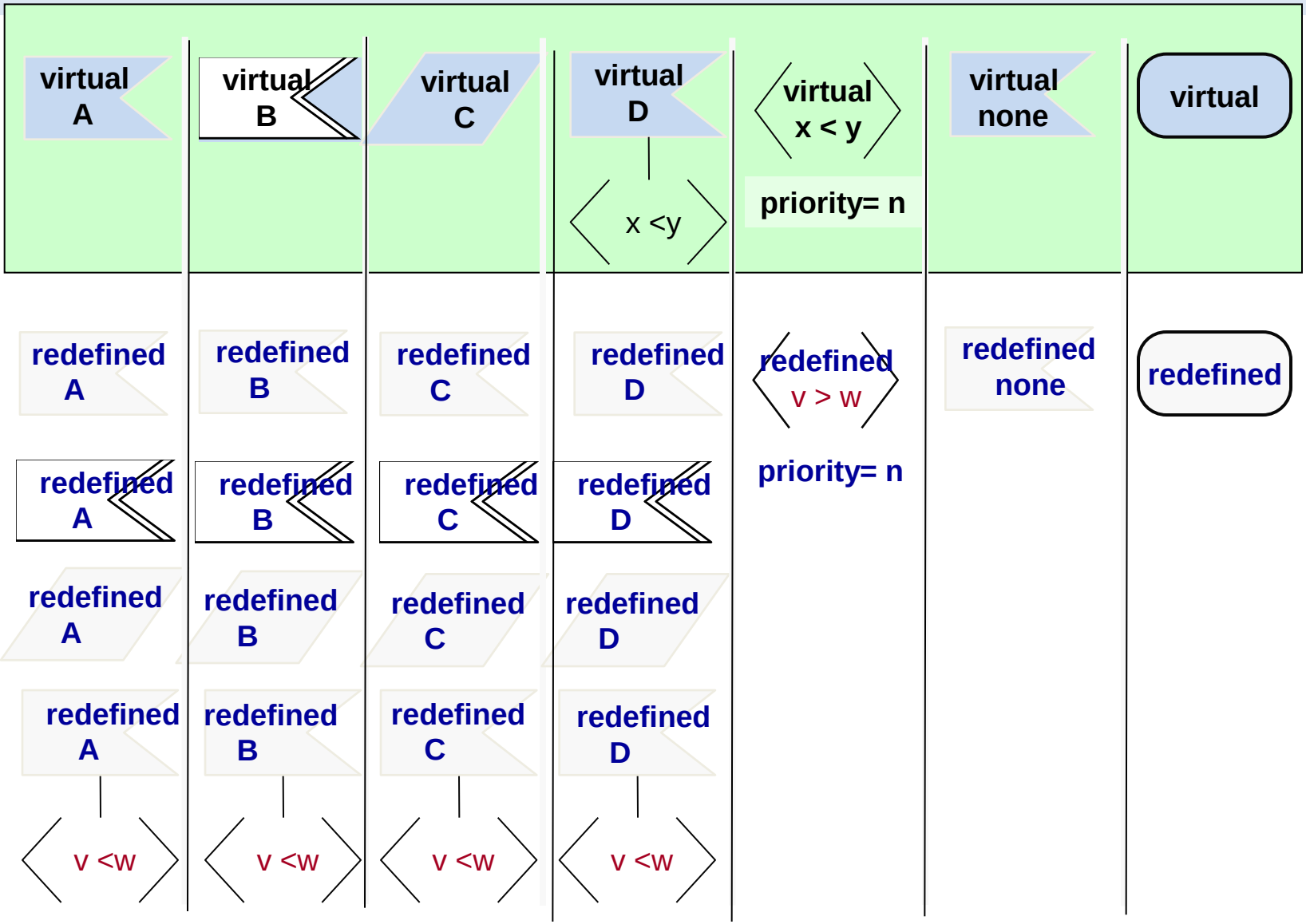
  - input redefined *sig*
  - save redefined *sig*
  - ~~priority input redefined *sig*~~
  - ~~input redefined *sig* provided <expr-2>~~

(Zustandsname, Signalname)

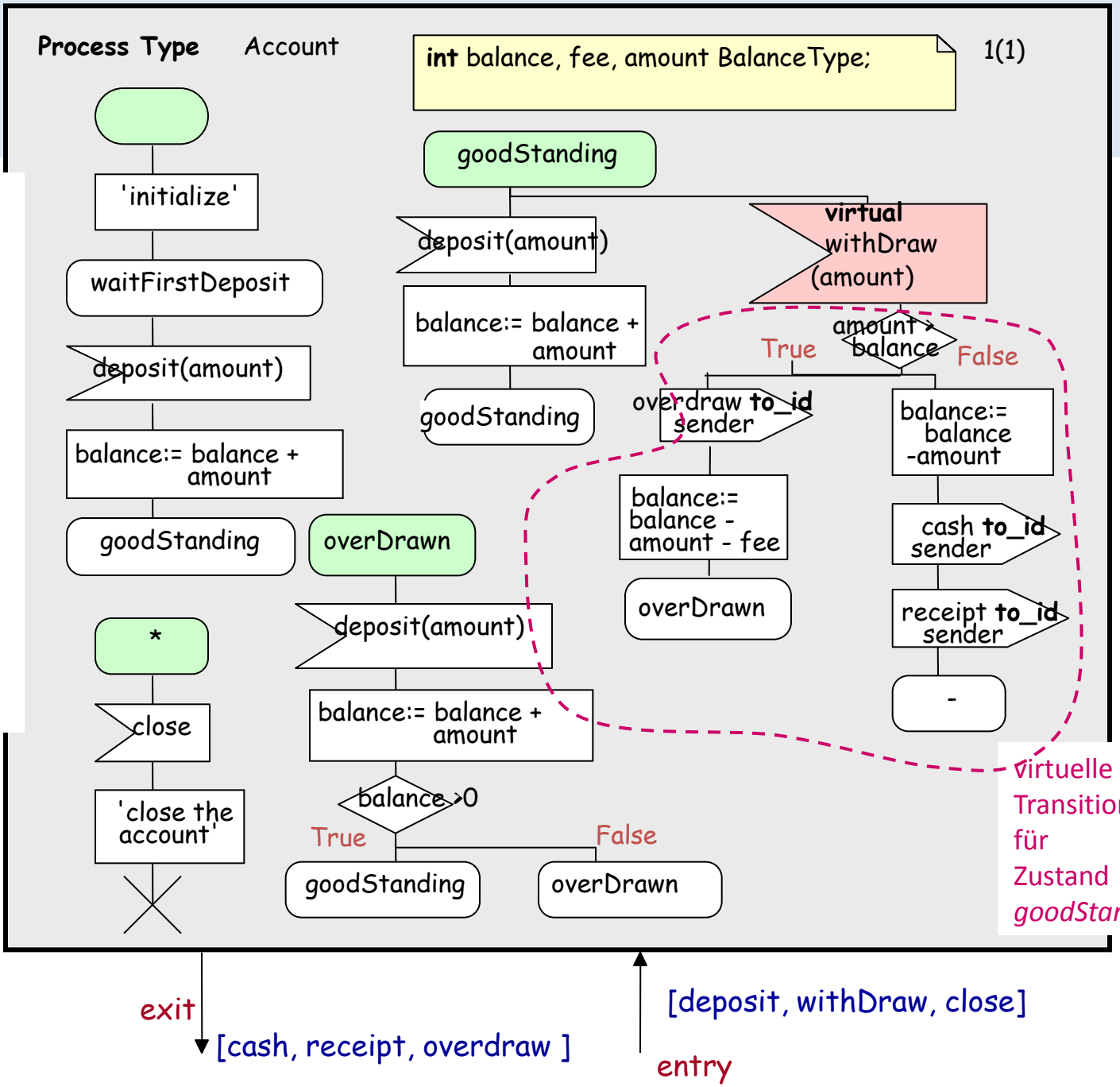
Redefinition von Transitionen: Zusammenfassung



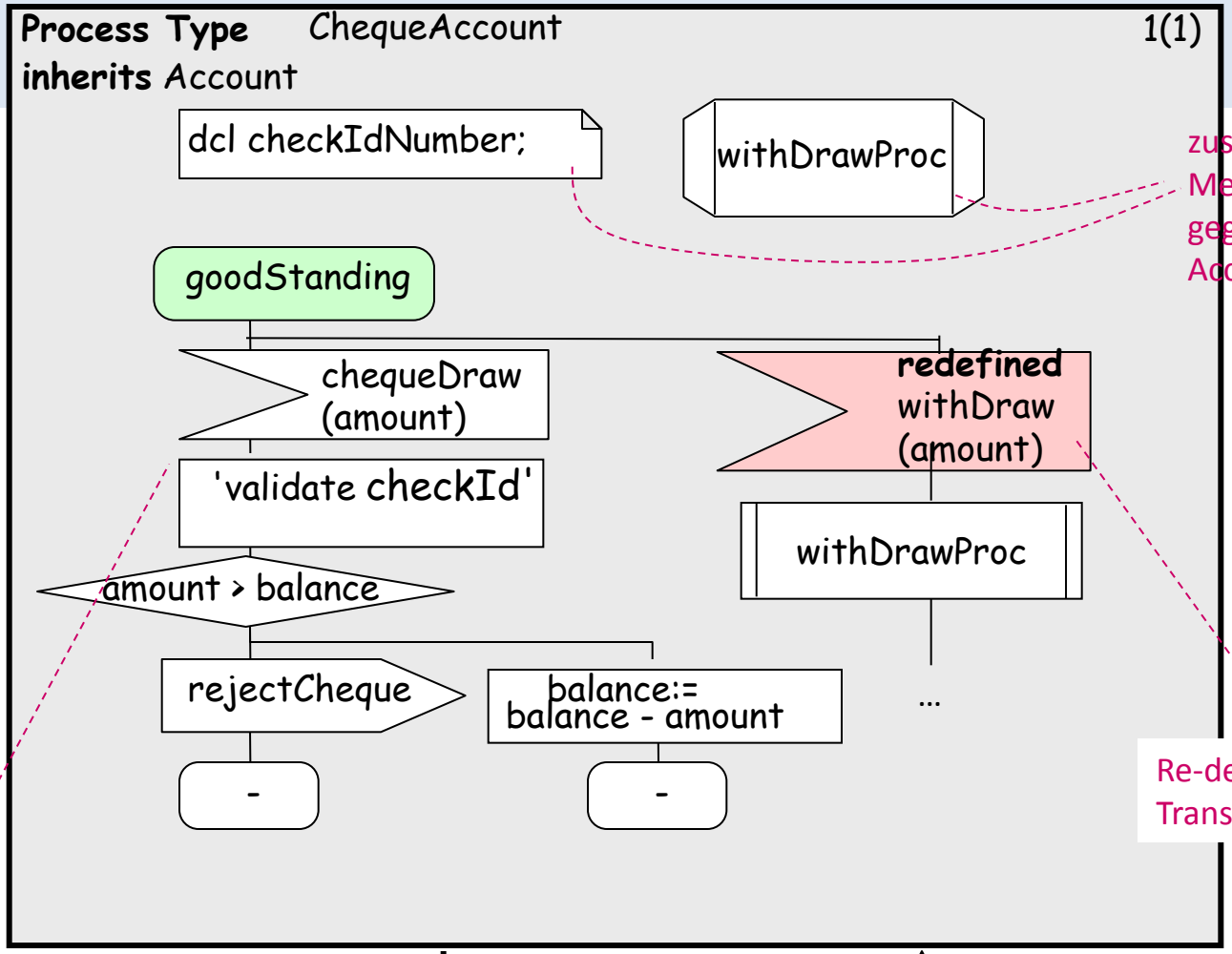
Redefinition von Transitionen: Standard-SDL



Basistyp


Process Type Account
int balance, fee, amount BalanceType;
1(1)
'initialize'
waitFirstDeposit
deposit(amount)
balance:= balance + amount
goodStanding
goodStanding
deposit(amount)
balance:= balance + amount
goodStanding
virtual withDraw (amount)
amount > balance
True
False
overdraw to_id sender
balance:= balance -amount
balance:= balance - amount - fee
overDrawn
cash to_id sender
receipt to_id sender
-
overDrawn
deposit(amount)
balance:= balance + amount
balance >0
True
False
goodStanding
overDrawn
*
close
'close the account'
virtuelle Transition für Zustand goodStanding
exit
[cash, receipt, overdraw]
[deposit, withDraw, close]
entry

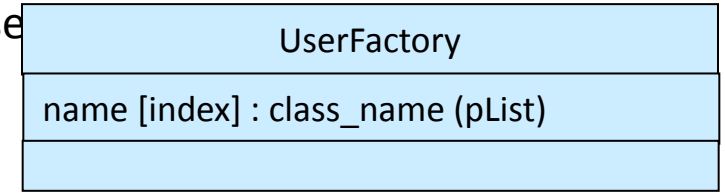
Ableitung


Process Type ChequeAccount
1(1)
inherits Account
dcl checkIdNumber;
withDrawProc
zusätzliche Merkmale gegenüber Account
goodStanding
chequeDraw (amount)
redefined withDraw (amount)
'validate checkId'
withDrawProc
amount > balance
rejectCheque
balance:= balance - amount
...
-
-
Re-definierte Transition
zusätzliche Transition
exit
[rejectCheque]
entry
[chequeDraw]
zusätzliche Signale für geerbte Gates

7. SDL-Konzepte (Präzisierung)

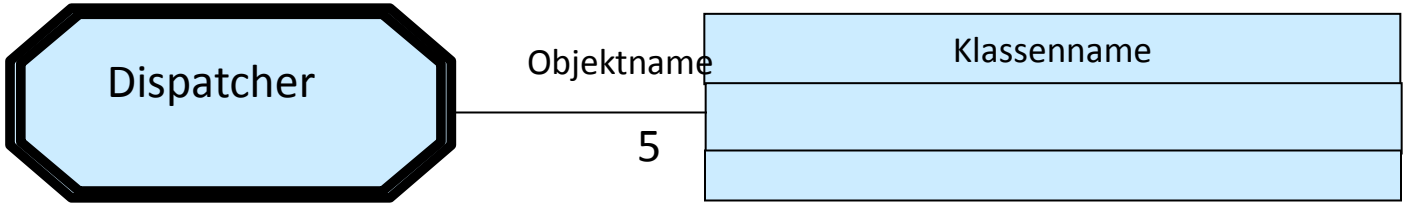
1. Modellstruktur
2. Einfacher Zustandsautomat: Triggerarten
3. Ersetzungsmodell: Priorisierter Input
4. Nachrichtenadressierung
5. Prozeduren
6. Timer
7. Remote Prozeduren
8. Dynamische Prozessgenerierung
9. Spezialisierung von Zustandsautomaten
10. Lokale Objekte, Semaphore

– In Klasse



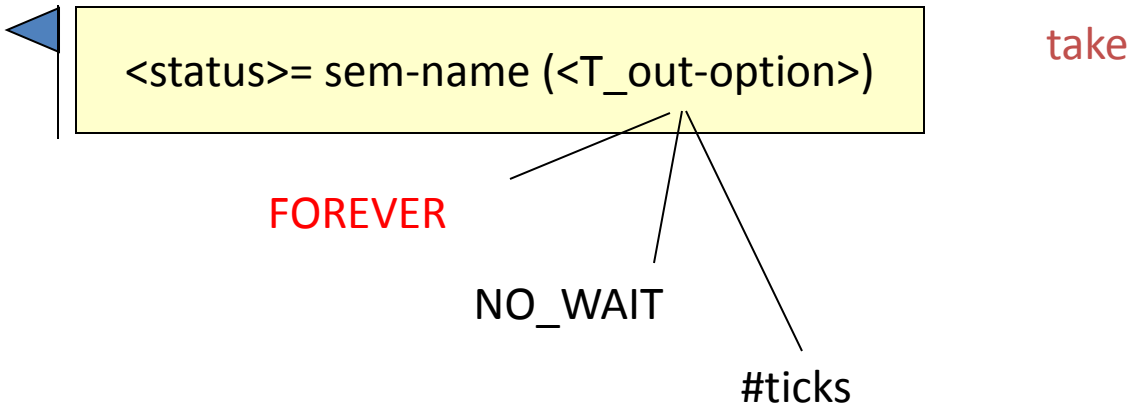
– In aktiven Klassen

(1) Attributdeklaration



(2) Objekterzeugung (darf nur in Starttransition erfolgen)

Semaphore-Nutzung



- RTDS_OK: Token wurde erfolgreich entnommen
- RTDS_ERROR: kein Token vorgefunden bzw. Timeout