Projekt Erdbebenfrühwarnung
im WiSe 2010/11

Entwicklung verteilter eingebetteter Systeme

Prof. Dr. Joachim Fischer
Dipl.-Inf. Ingmar Eveslage
Dipl.-Inf. Frank Kühnlenz

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

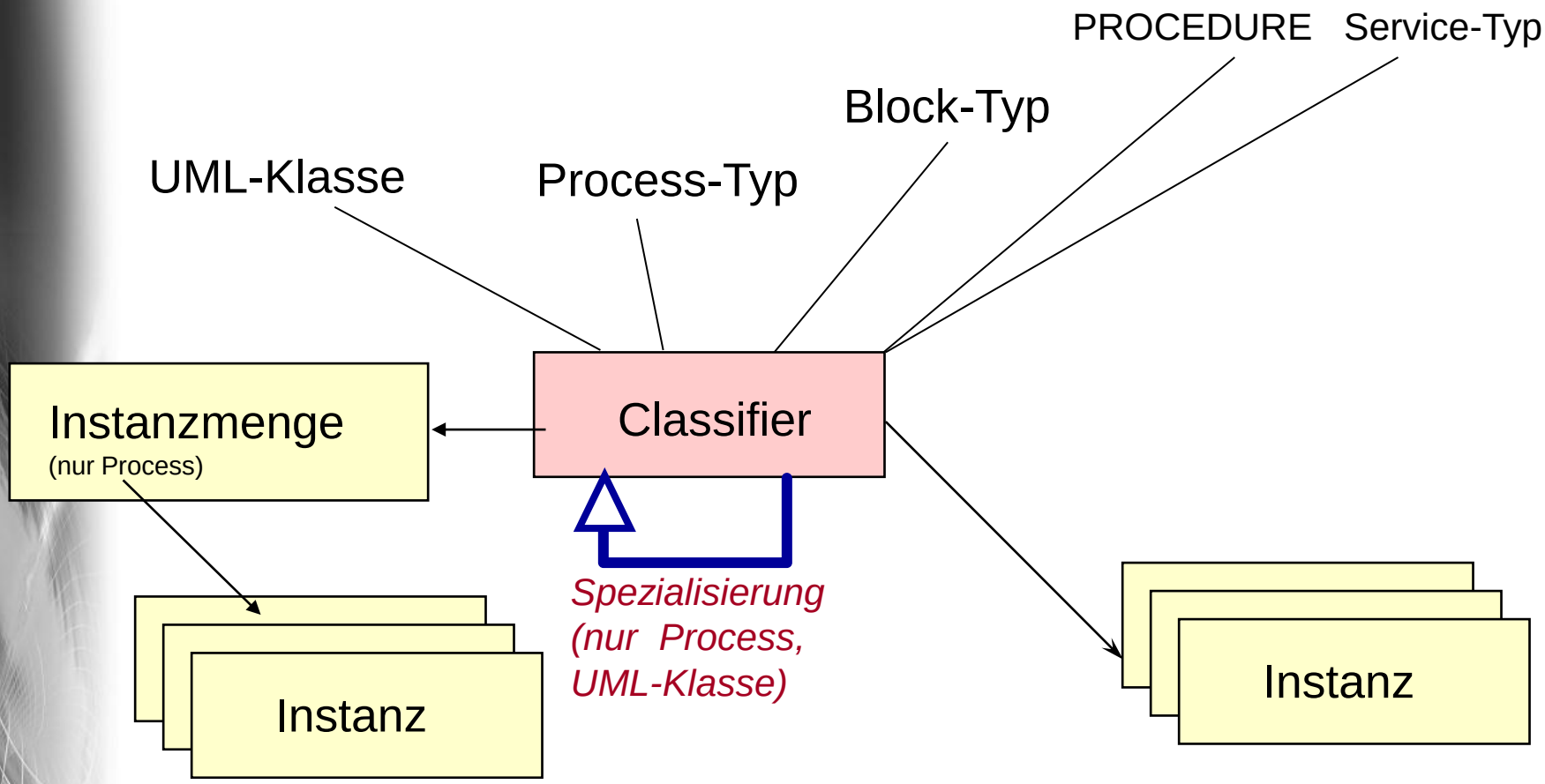
6. SDL-Konzepte (Präzisierung)

1. Modellstruktur
2. Einfacher Zustandsautomat: Triggerarten
3. Nachrichtenadressierung
4. Dynamische Prozessgenerierung
5. Prozeduren
6. Lokale Objekte
7. Ersetzungsmodelle
8. Semaphore
9. Spezialisierung von Zustandsautomaten

Kap. 7 INRES-Protokoll

Fortsetzung jetzt

Typen, parametrisierte Typen, Spezialisierung



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

Spezialisierung (Vererbung im Z.100)

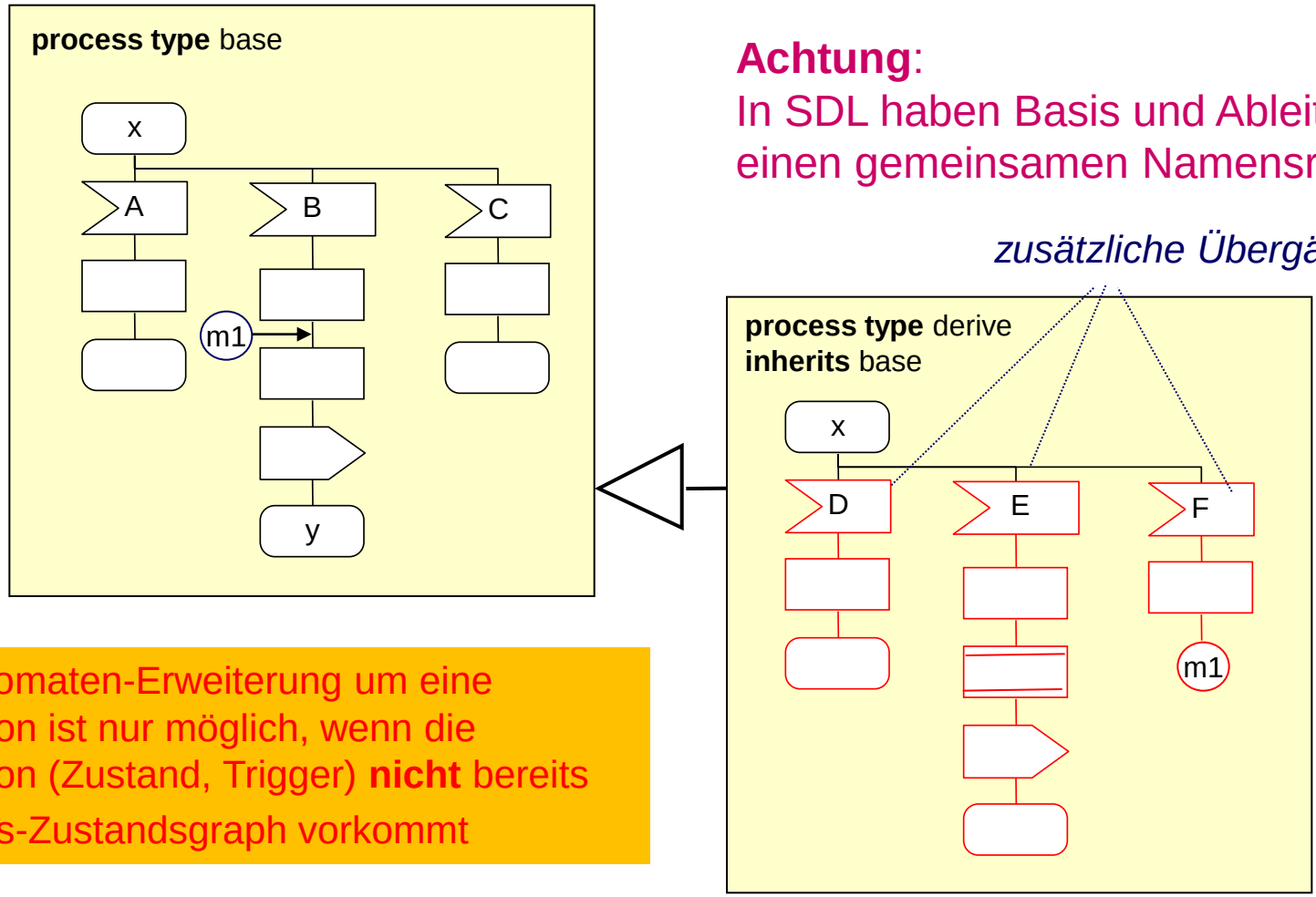
- Mechanismus, um einen neuen Klassentyp von einem bereits existenten Basistyp abzuleiten

- 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 !!!

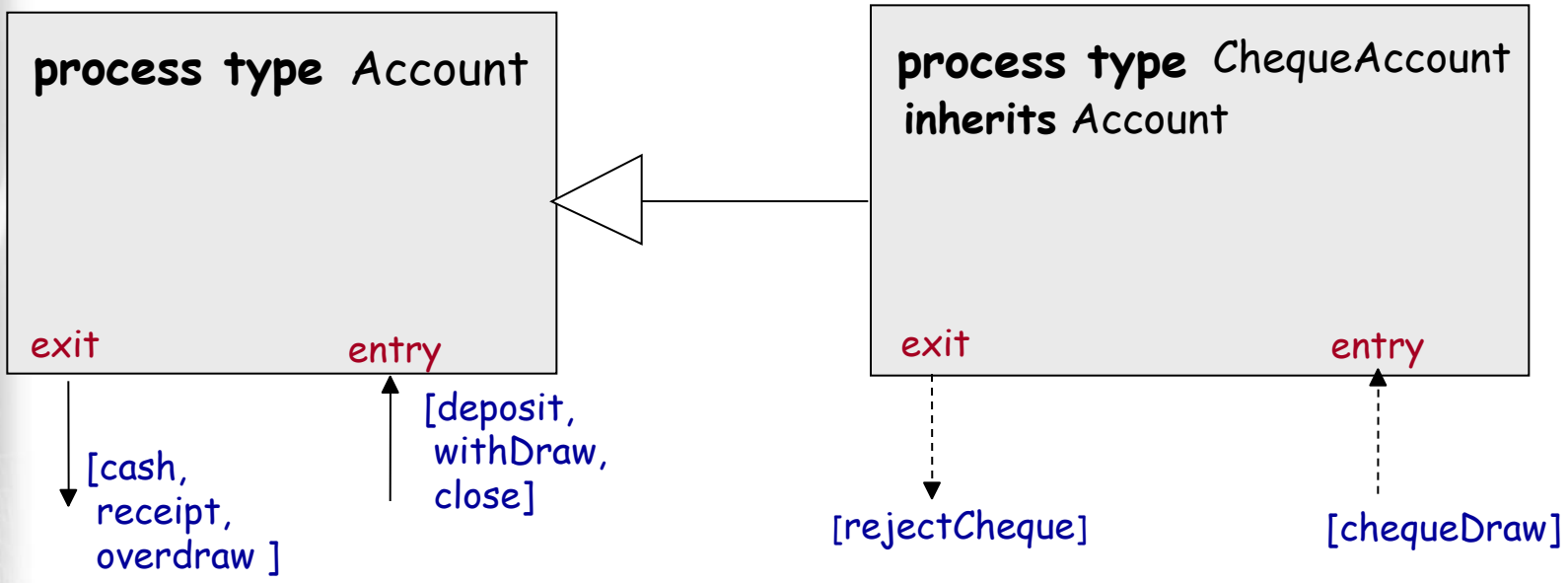
zusätzliche Übergänge

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

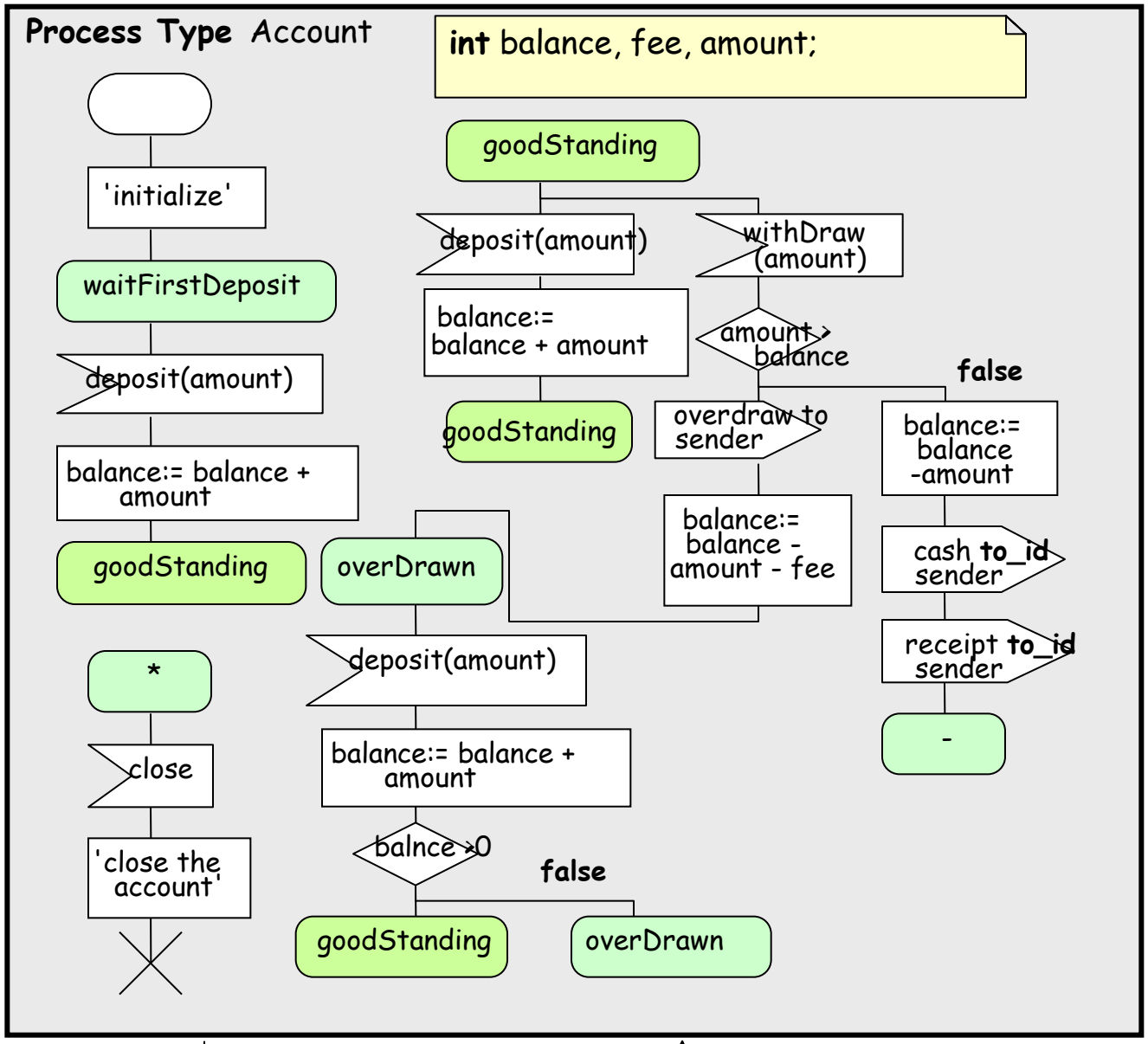
Spezialisierung/ abgeleiteter Typ (mit Gate-Erweiterung)



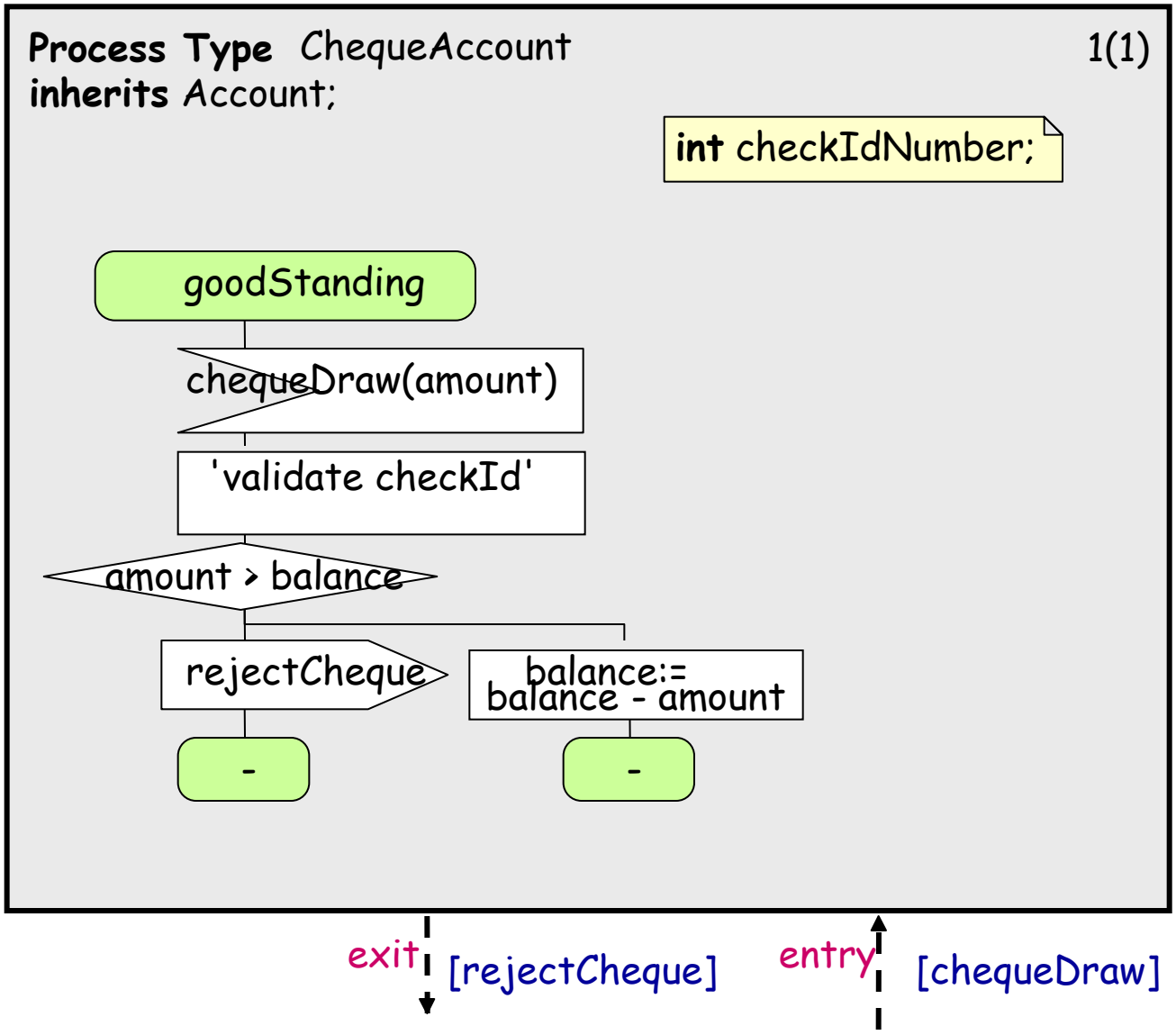
Account- Funktionalität
- Einzahlen
- Abheben → (Cash,Quittung) | (Überziehungsmitteilung)
- Konto schließen

zusätzliche ChequeAccount-Funktionalität
- Überweisung→ (Annullierung)

Basistyp

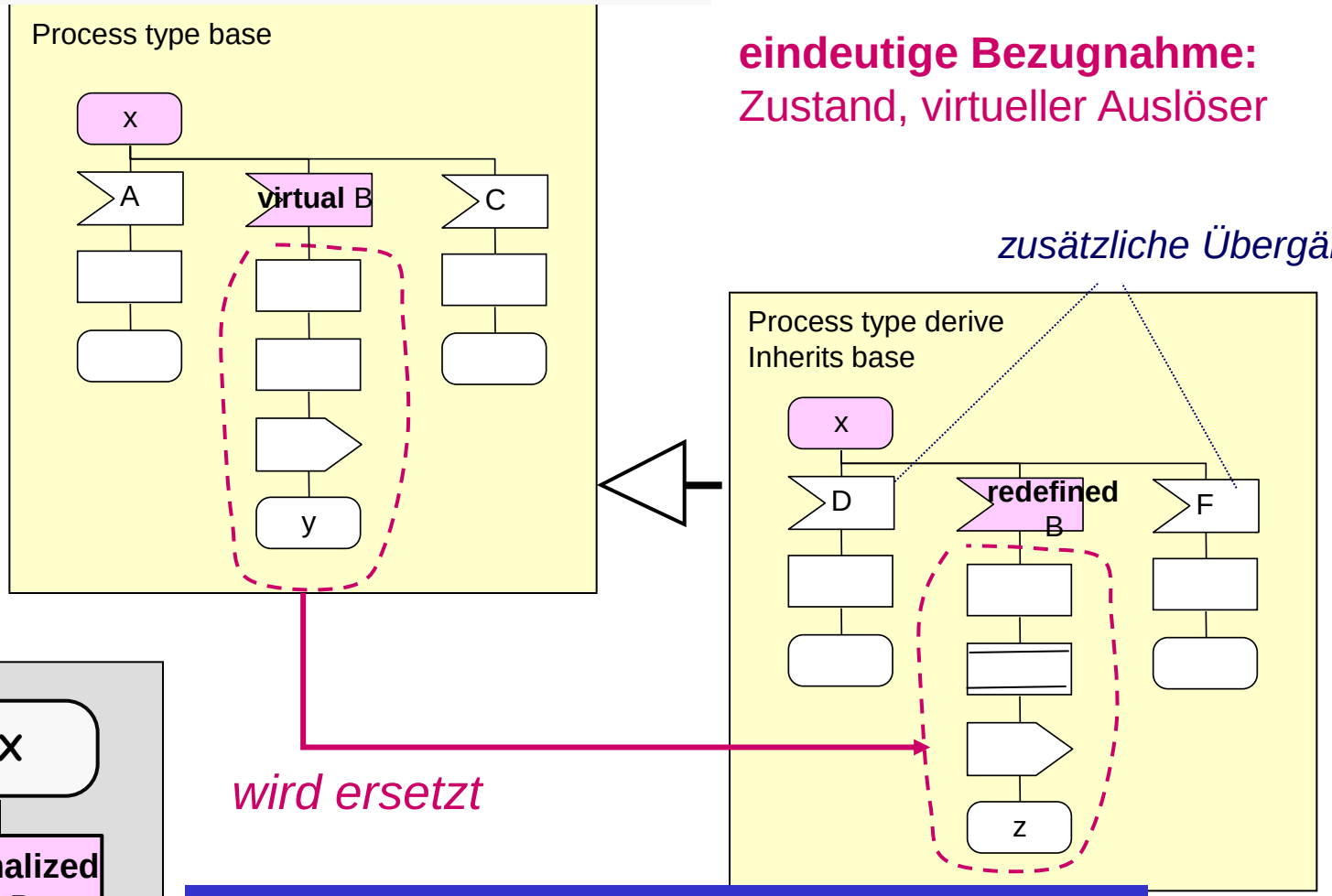

Process Type Account
int balance, fee, amount;
'initialize'
waitFirstDeposit
deposit(amount)
balance:= balance + amount
goodStanding
goodStanding
deposit(amount)
balance:= balance + amount
goodStanding
withDraw (amount)
amount > balance
false
overdraw to sender
balance:= balance - amount - fee
overDrawn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

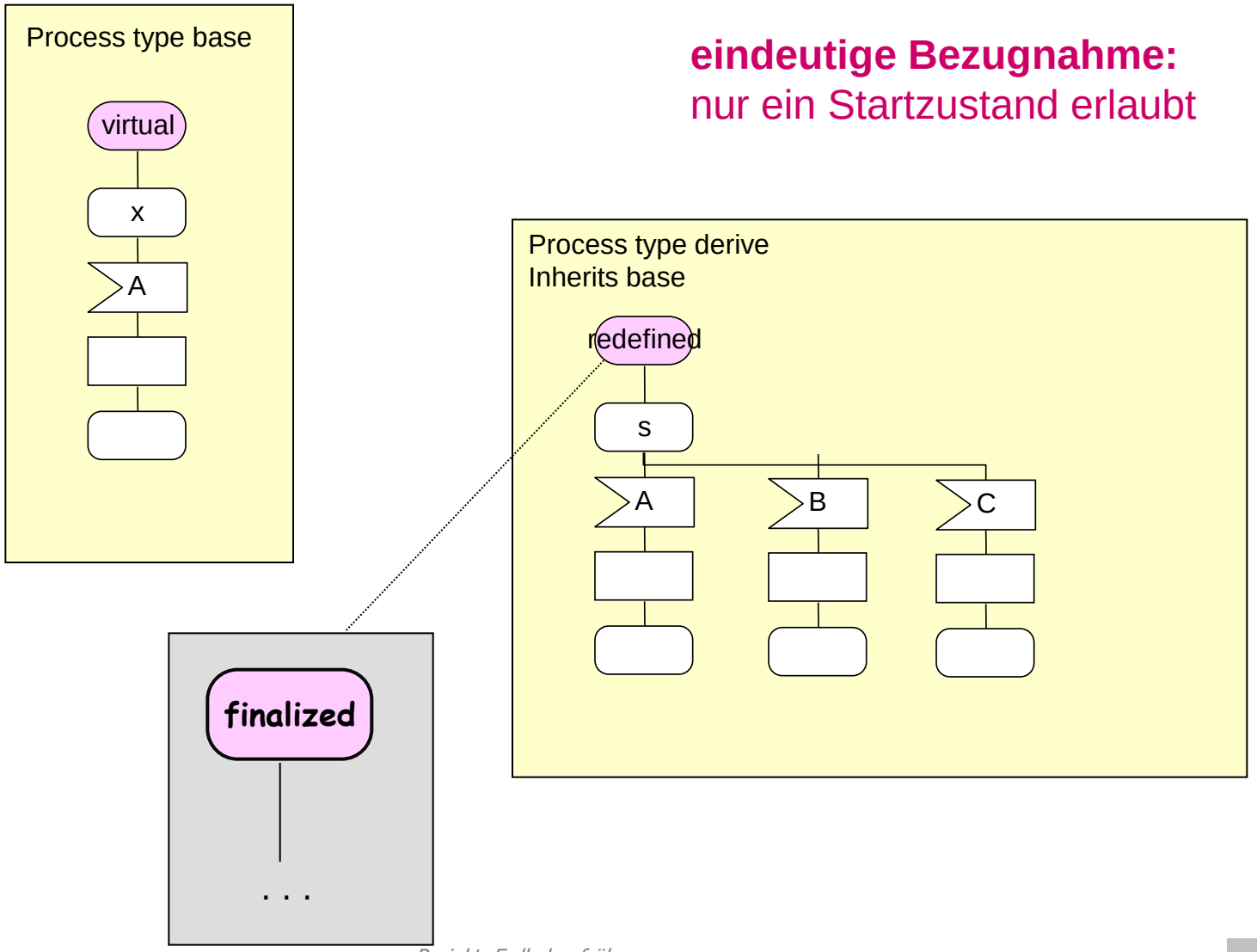
Redefinition einzelner Transitionen: Prinzip

gilt für Process-Typ, Service-Typ, Procedure



Achtung: lokale Variablen
zur Übernahme der Nachrichten-Parameter änderbar

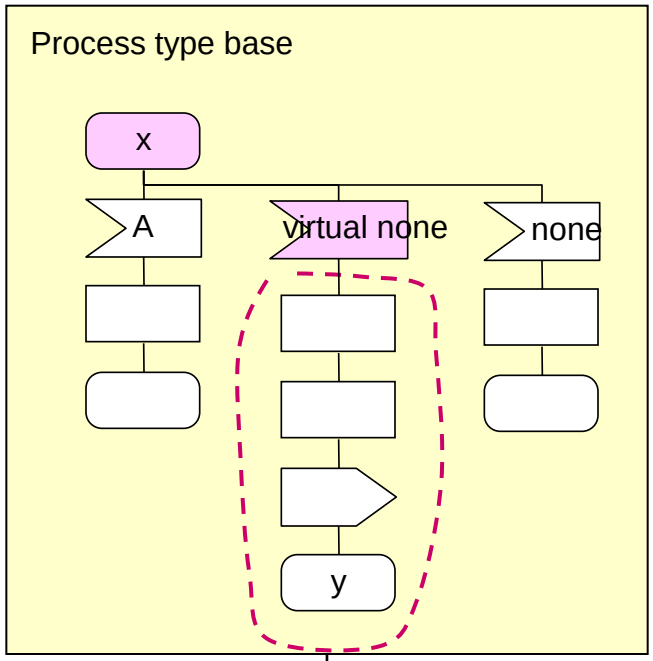
Redefinition des gesamten Zustandsautomaten



eindeutige Bezugnahme:
nur ein Startzustand erlaubt

nicht in SDL/RT !

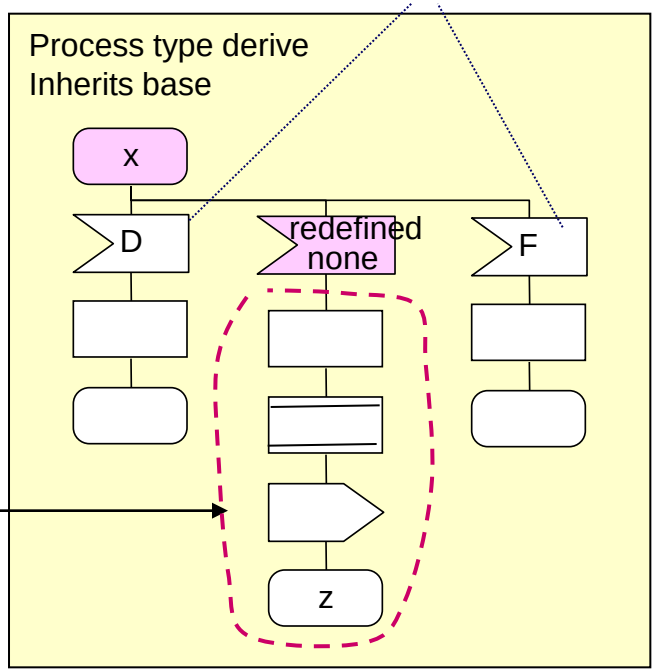
Redefinition spontaner Transitionen



eindeutige Auflösung
macht Einschränkung erforderlich:

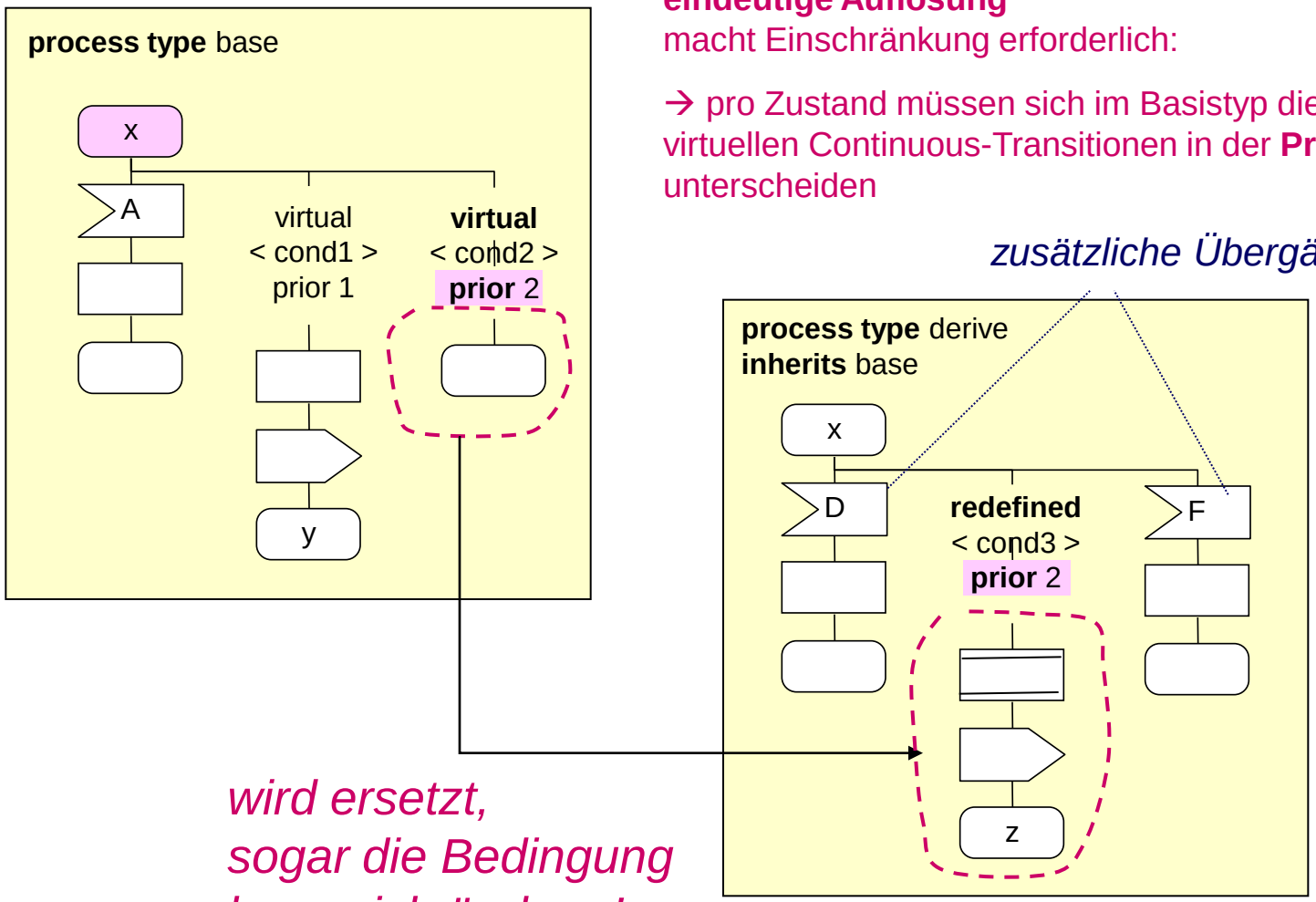
→ pro Zustand ist im Basistyp **nur eine** virtuelle spontane Transition erlaubt

zusätzliche Übergänge



wird ersetzt

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>~~

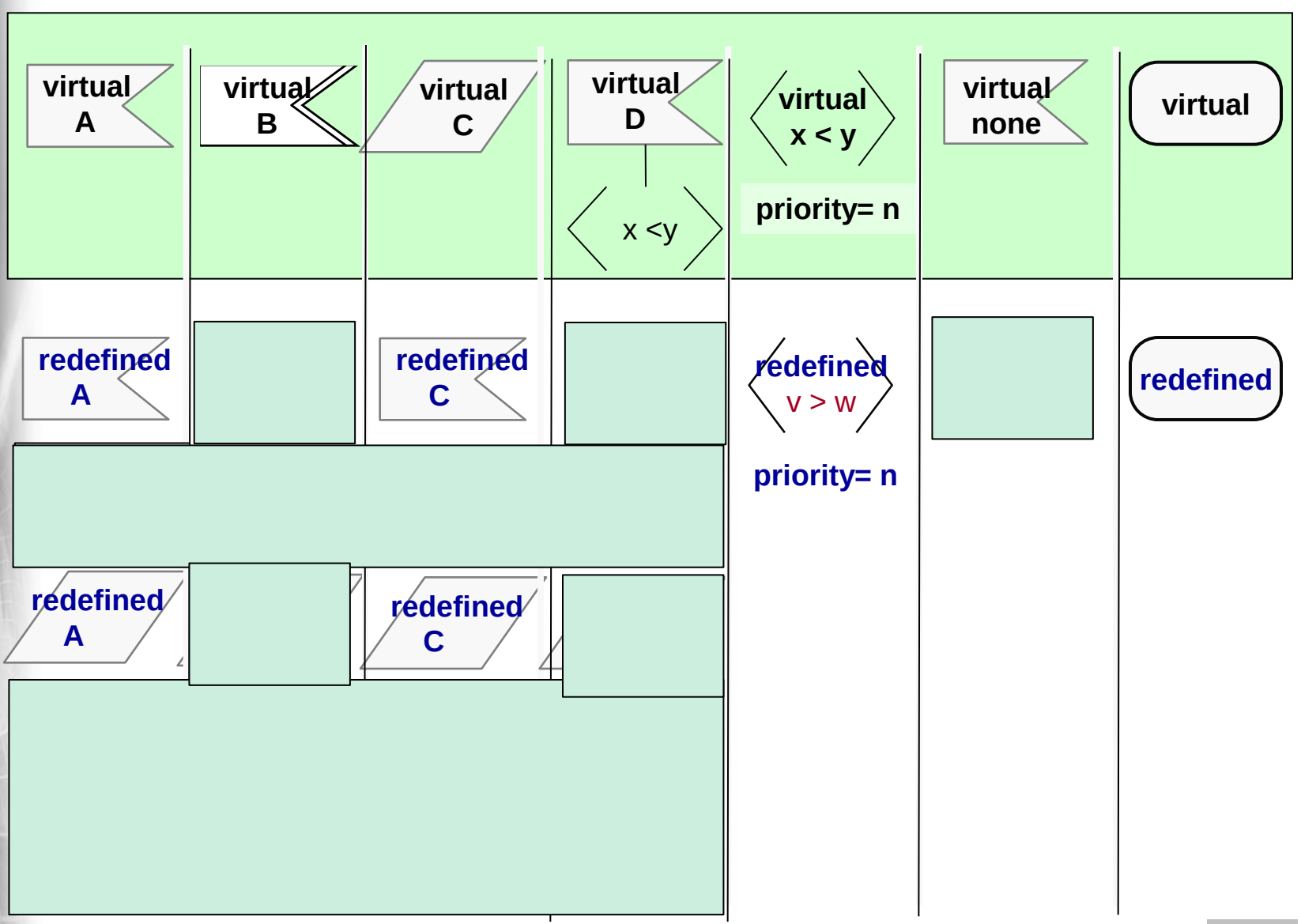
(Zustandsname, Signalname)

1-deutige Zuordnung

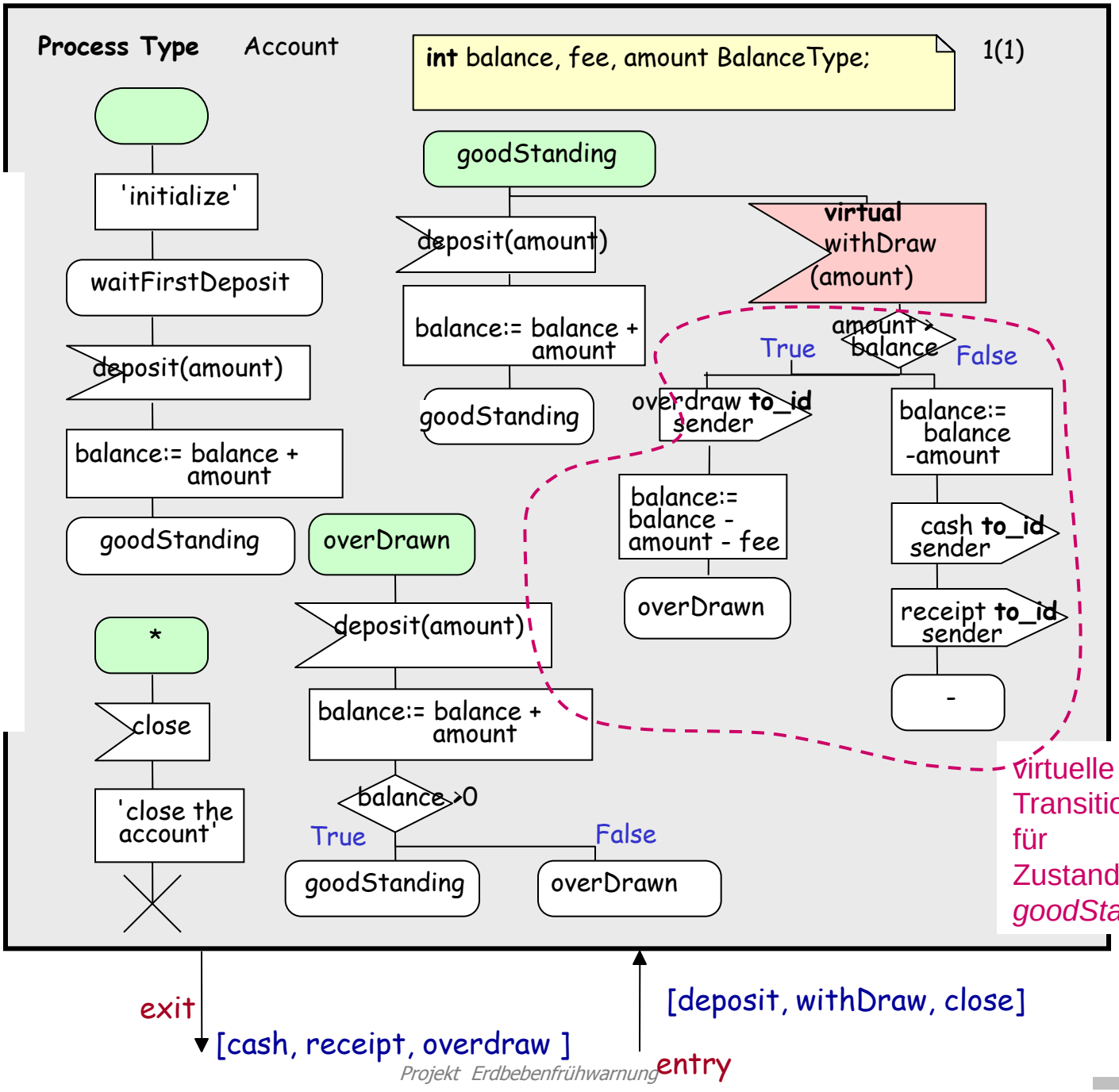
- kann in einer Ableitung von X im Zustand S zu einem beliebigen Trigger der folgenden Art umfunktioniert werden
  - **input redefined** *sig*
  - **save redefined** *sig*
  - ~~**priority input redefined** *sig*~~
  - ~~**input redefined** *sig* **provided** <expr-2>~~

(Zustandsname, Signalname)

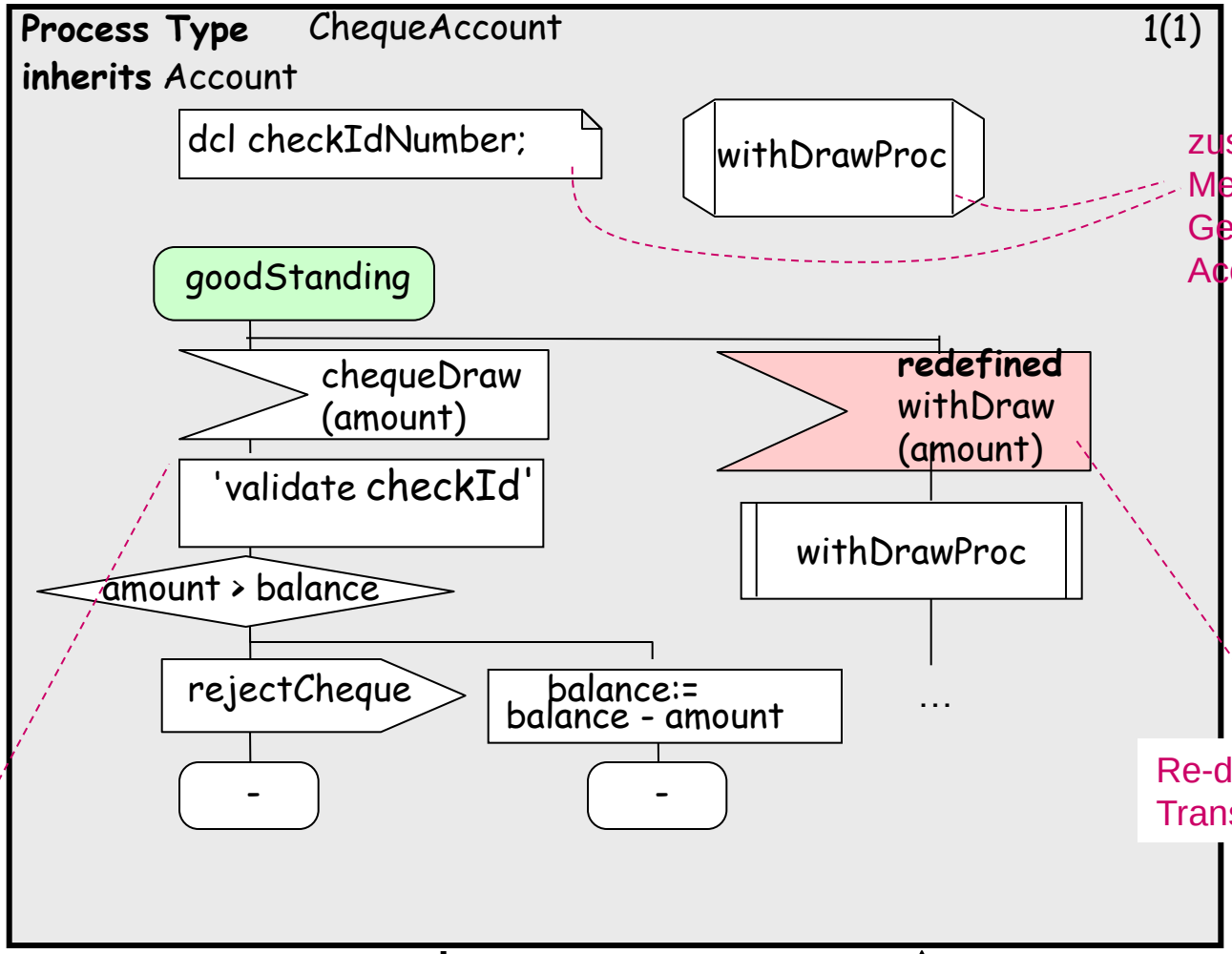
Redefinition von Transitionen: Zusammenfassung



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 - fee
overDrawn
balance:= balance -amount
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



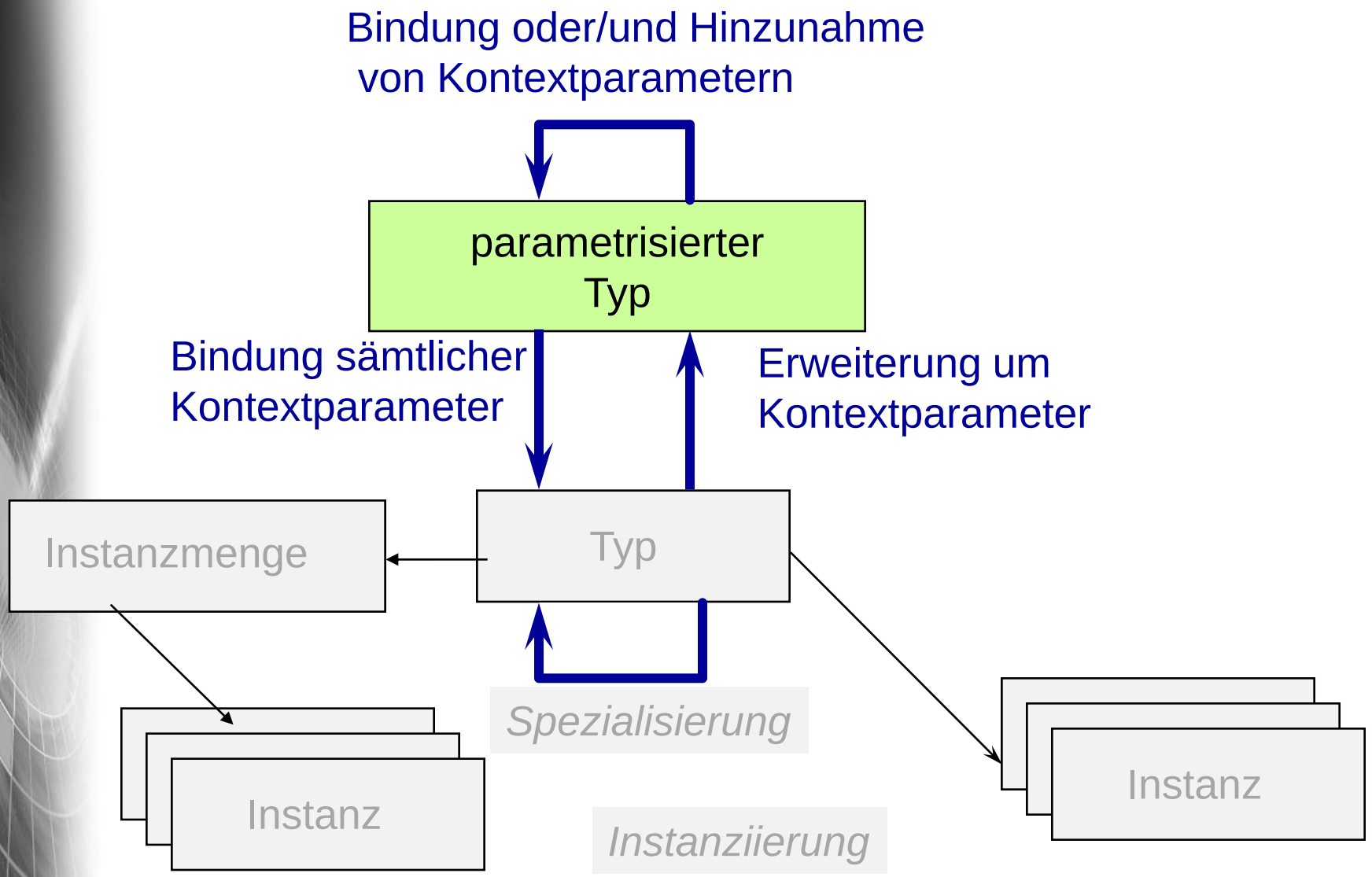
zusätzliche Merkmale Gegenüber Account

Re-definierte Transition

zusätzliche Transition

zusätzliche Signale für geerbte Gates

Typ- Relationen



Spezialisierung von Typen

… als Einfachvererbung von

- Systemtypen
- Blocktypen
- Prozesstypen
- Servicetypen
- Prozeduren
- Signalen
- Datentypen

Konkretisierung von Typschablonen

- … als Einfachvererbung von
  - Systemtypen mit Kontextparametern
  - Blocktypen mit Kontextparametern
  - Prozesstypen mit Kontextparametern
  - Servicetypen mit Kontextparametern
  - Prozeduren mit Kontextparametern
  - Signalen mit Kontextparametern
  - Datentypen mit Kontextparametern

bei Bindung sämtlicher Kontextparameter durch konkrete Typen

Gates mit Typ-Constraints (Wdh.)

- Gates müssen für
  - System-,
  - Block-,
  - Process- und
  - Servicetypen

  definiert werden

- Gate- Definition
  - Name,
  - Signallisten (je Richtung eine),
  - optionale Angabe einer Typ-Bedingung für den Partner-Block (oder Partnerböcke) des Gates: „endpoint constraint“

    der Partner muss von einem Basistyp sein

    Beispiel:

    gate g **in with** sigX,(listY) **atleast** bank ;

Virtuelle Typen

- Block-, Process- und Service-Typen sowie Prozeduren können als **virtuelle Typen** vereinbart werden

- durch ihre Definition und Anwendung **in umfassenden Containertypen** als
  - Instanzenbildung virtueller Blocktypen in Systemtypen
  - Instanzenbildung virtueller Prozesstypen in Blocktypen
  - Instanzenbildung virtueller Servicetypen in Prozesstypen
  - Aufruf virtueller Prozeduren in Prozess- und Servicetypen oder in Prozeduren

  werden **Konfigurationen** (als kommunizierende Block Prozess- und Service-Instanzen oder Prozeduraufrufketten) definiert

- In **Ableitungen dieser Containertypen** können bei Beibehaltung bzw. Erweiterung dieser Konfigurationen die virtuellen Typdefinitionen der Basis-Container **redefiniert** werden

Beschränkungen virtueller Typen

- Der Mindesttyp einer virtuellen Redefinition kann per atleast- Klausel definiert werden

  Die aktuelle Definition sichert diese durch Vererbung des Typs aus der atleast-Angabe

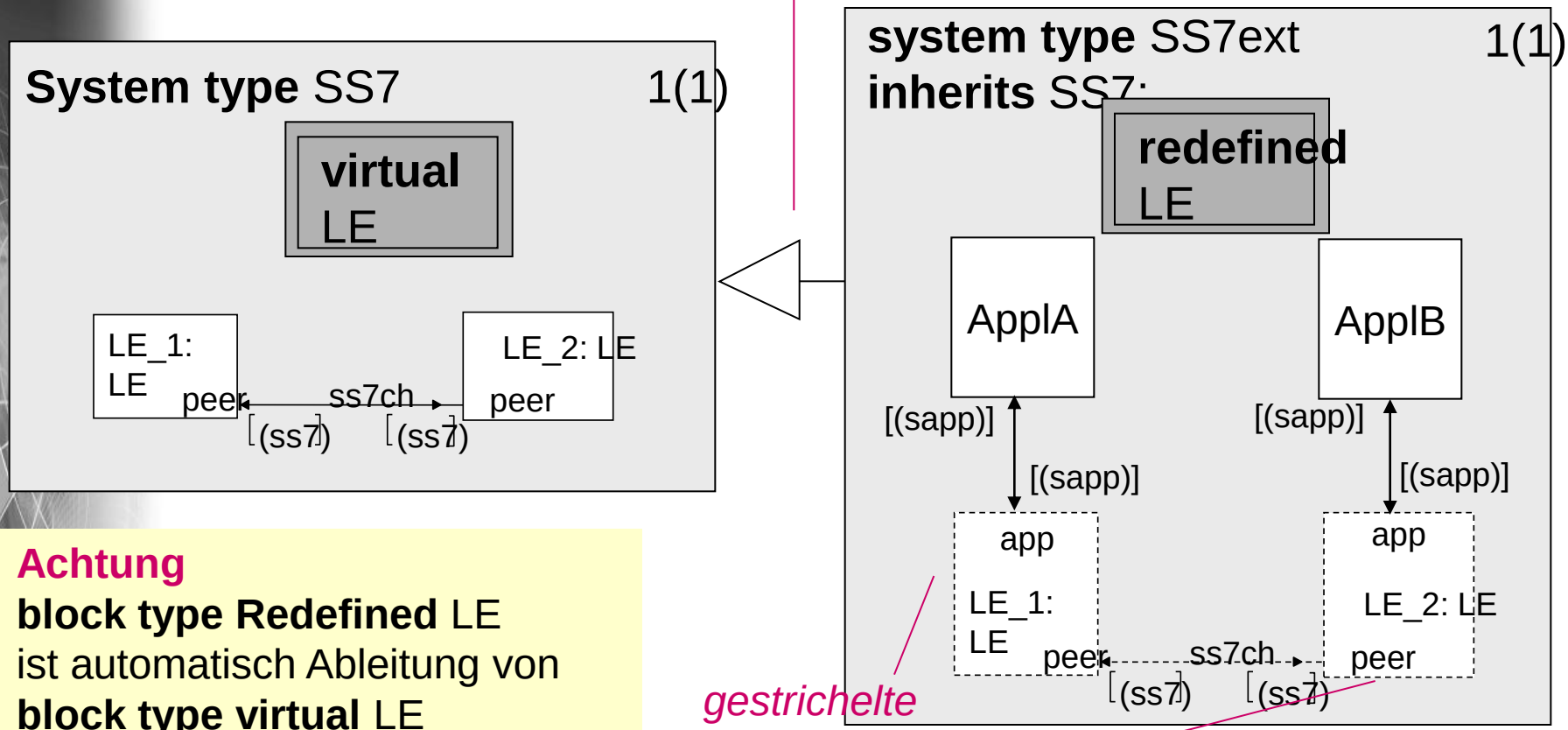
  Die Redefinition muss sich dann „nur“ auf den Basistyp (atleast- Angabe) beziehen und nicht auf die Spezialisierung der virtuellen Definition

- Wird auf die atleast- Angabe bei der virtuellen Definition verzichtet, gibt es eine implizites Bedingung:

  Die Redefinition ist eine Ableitung der virtuellen Definition

Virtuelle Block-Typen

Vererbung der definierenden Containertypen: hier als Systemtyp

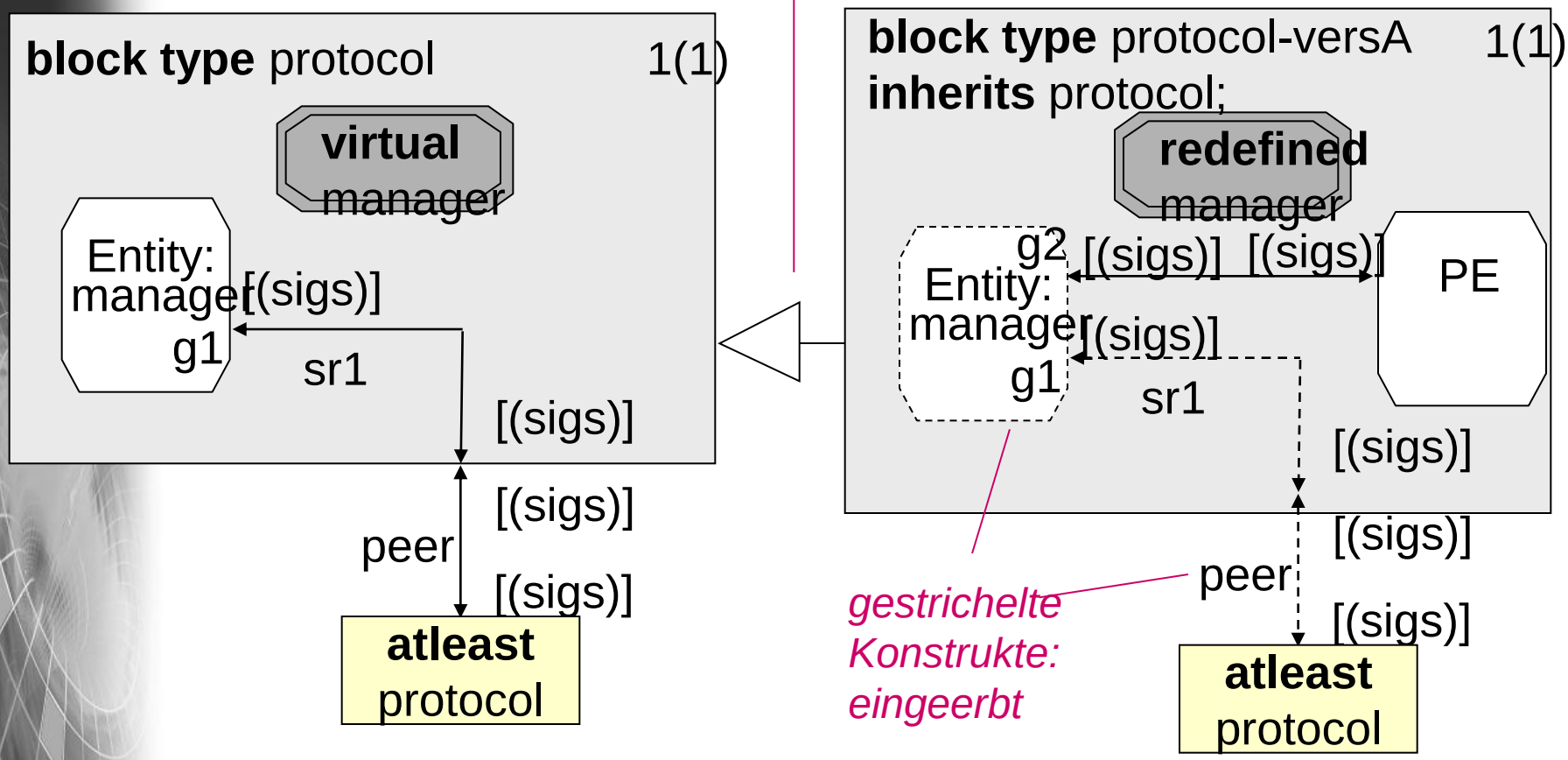


Achtung
block type Redefined LE
ist automatisch Ableitung von
block type virtual LE

gestrichelte Konstrukte: eingeerbt

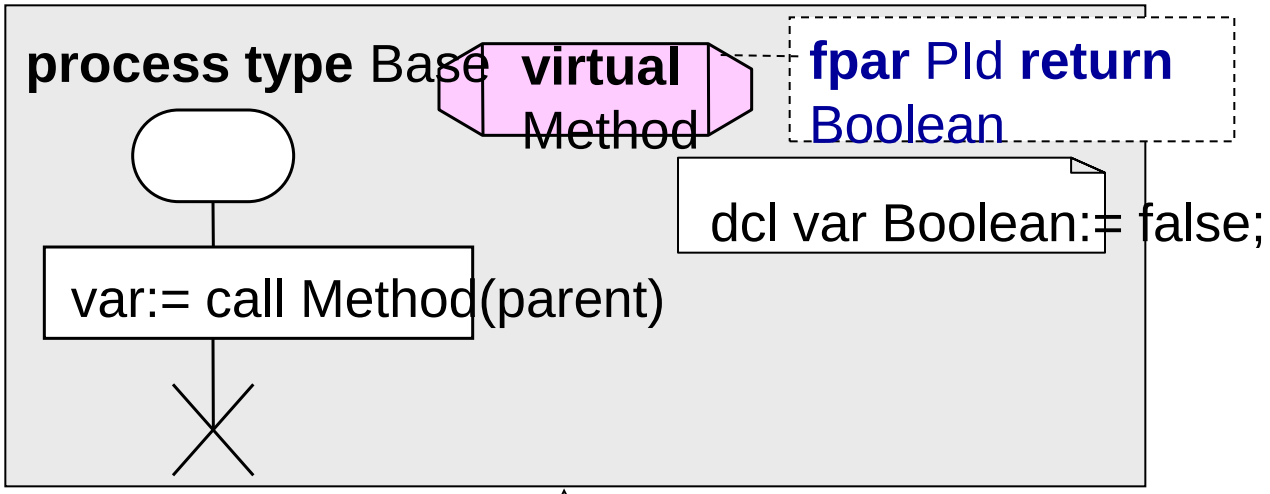
Virtuelle Process-Typen

Vererbung der definierenden Containertypen: hier als Blocktyp

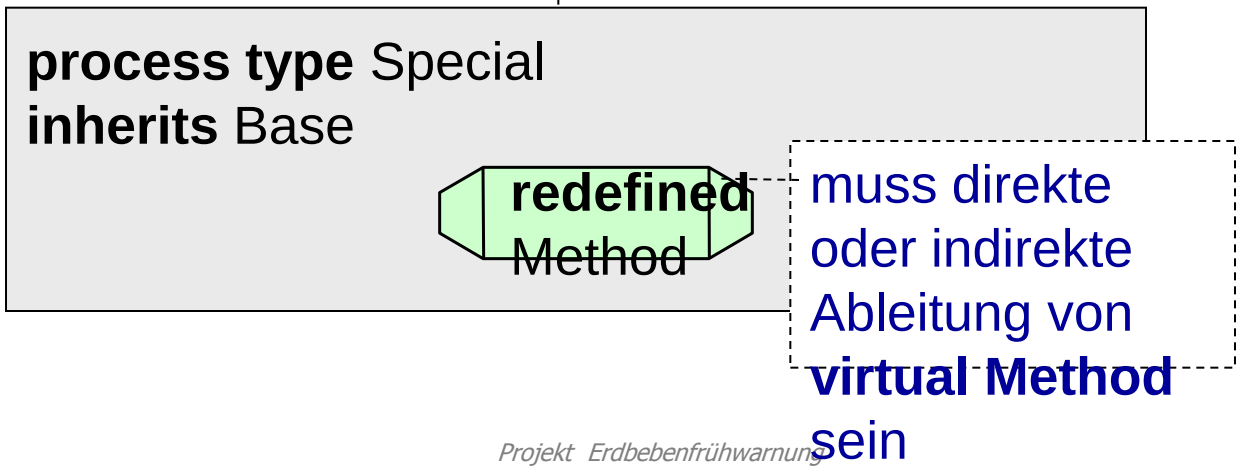


Achtung
process type redefined manager ist automatisch Ableitung von **process type virtual** manager

Virtuelle Prozeduren



Vererbung der definierenden Containertypen: hier als Prozesstyp



Virtuelle Prozeduren