

Was ist ein Geschäftsprozess?

Beispiele:

- Bearbeitung eines Schadensfalls in einer Versicherung
- Kreditüberprüfung in einer Bank
- Ausschreibung, Vergabe eines Bauprojekts in der Stadtverwaltung
- Reklamationsprozess in einem Versandhaus
- Wartung eines technischen Geräts in einem Kraftwerk
- Entwicklung von Software in einem Systemhaus
- Just-in-Time Logistik eines Automobilzulieferers
- ...

Was ist ein Geschäftsprozess?

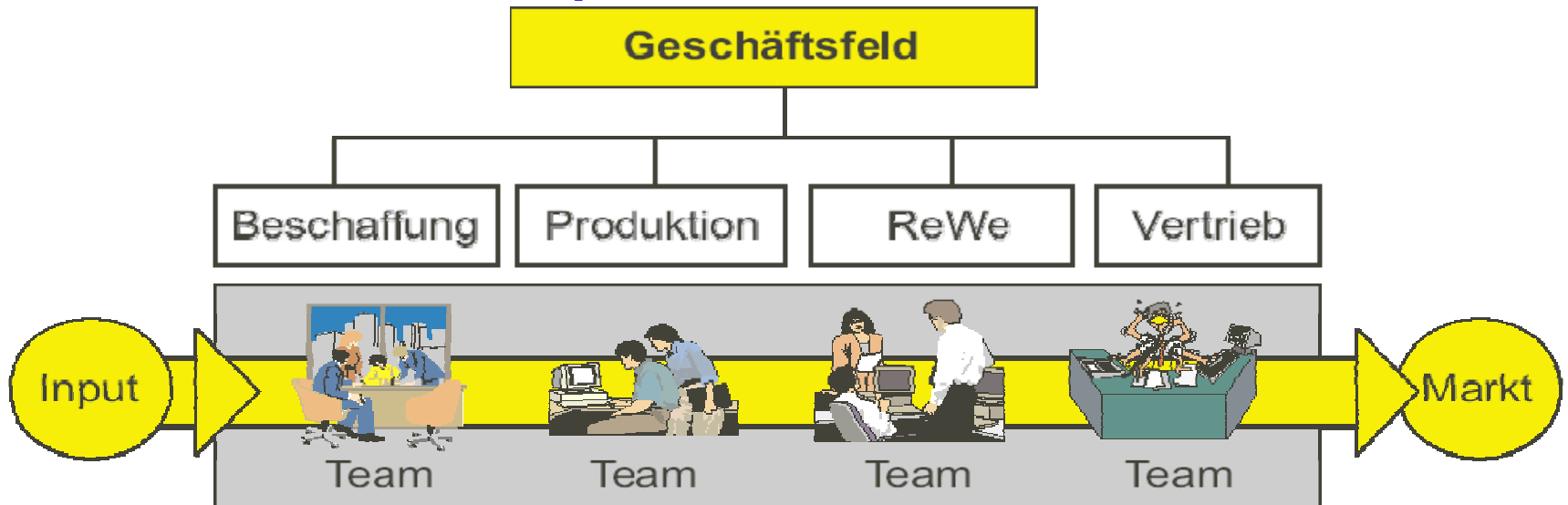
Ein Geschäftsprozess ist eine Folge von Aktivitäten, die

- in einem logischen Zusammenhang stehen,
- inhaltlich abgeschlossen sind und
- unter Zuhilfenahme von Ressourcen
- und eingehenden Informationen
- durch Menschen und/ oder Maschinen
- auf ein Unternehmensziel hin
ausgeführt werden.

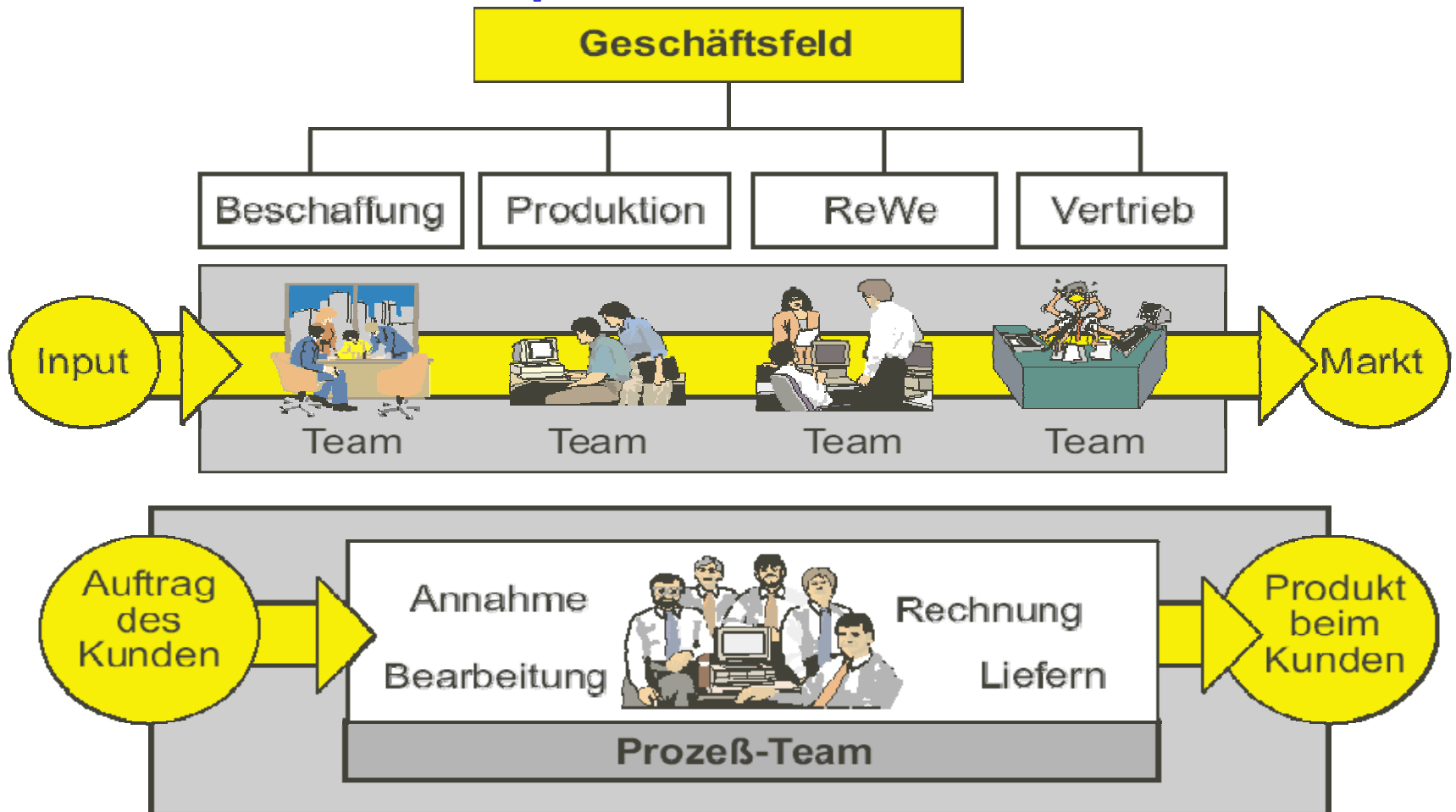
Beispiel Versandhaus



Beispiel Versandhaus



Beispiel Versandhaus



Eigenschaften eines Geschäftsprozesses

- Ein Geschäftsprozess
 - ist ein Vorgang in Wirtschaftseinheiten,
 - kann funktions-, hierarchie- und standort-übergreifend ablaufen.
- Ein Geschäftsprozess zeichnet sich aus durch:
 - einen definierten Anfang/ definiertes Ende,
 - erforderliche Eingaben (z.B. Kundenwünsche),
 - produzierte Ergebnisse (z.B. Befriedigung der Kundenwünsche).

Wozu Geschäftsprozesse?

Existieren unabhängig von unserer Wahrnehmung

Bewusste Wahrnehmung (= Modellierung) gestattet

- Analyse, Optimierung, Umgestaltung, Automatisierung
- Reaktion auf neue Wettbewerbssituation, rechtliche Rahmenbedingungen etc.
- Höhere Qualität (ISO 9000)
- Bessere Ausschöpfung der Ressourcen
- Größere Produktvielfalt

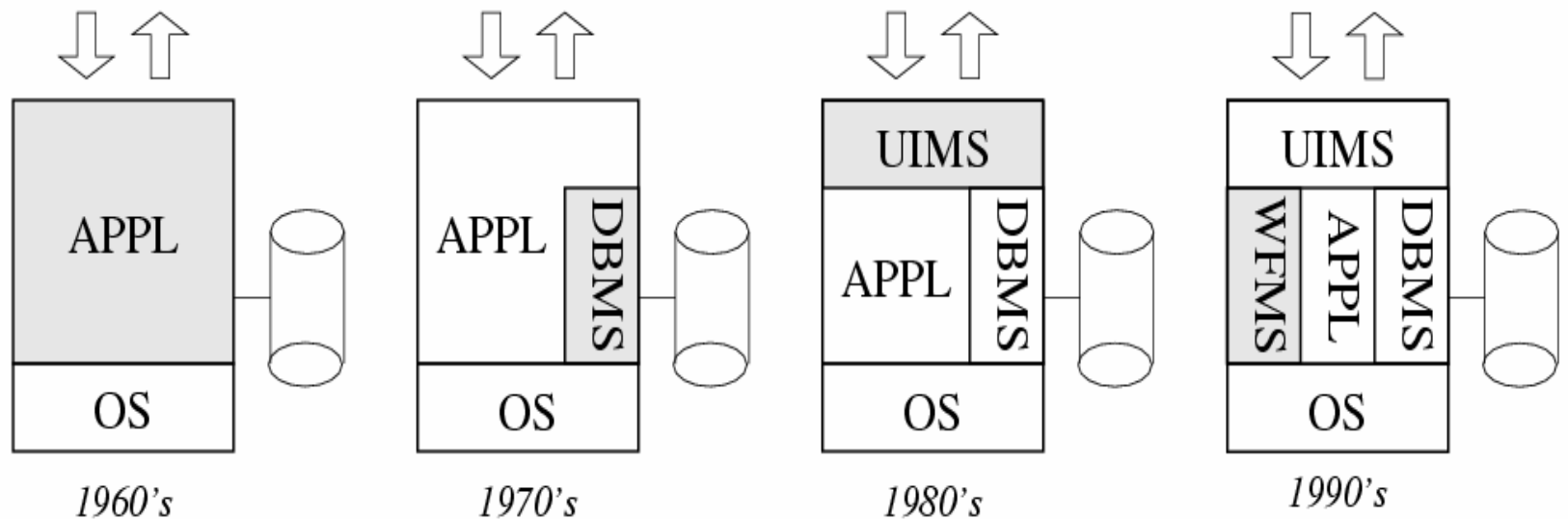
Was ist ein Workflow?

- Ein Workflow
 - ist eine z.T. automatisiert ablaufende Gesamtheit von Aktivitäten,
 - wird von einem Workflow-Management-System gesteuert/überwacht,
 - bezieht sich auf Teile eines Geschäftsprozesses,
 - besteht aus Vorgangsabschnitten (Sub-Workflows).
- Ein Workflow ist die **informationstechnische Realisierung** eines Geschäftsprozesses.

Unterschiede der Modellierung

- Geschäftsprozessmodellierung
 - Fokus:
 - Betriebliche Systeme/Strukturen
 - Ziele:
 - Analyse (Wirtschaftlichkeit)
 - Gestaltung (Dokumentation)
 - Optimierung (BPR, CPI)
 - Struktur:
 - Hierarchische Detaillierungs-stufen
- Workflow-Modellierung
 - Fokus:
 - Informationssysteme/Anwendungen
 - Ziele:
 - Realisierung (build time)
 - Ausführung (run time)
 - Optimierung (Performance)
 - Struktur:
 - Technische Modellhierarchie

Was ist Workflow-Management?



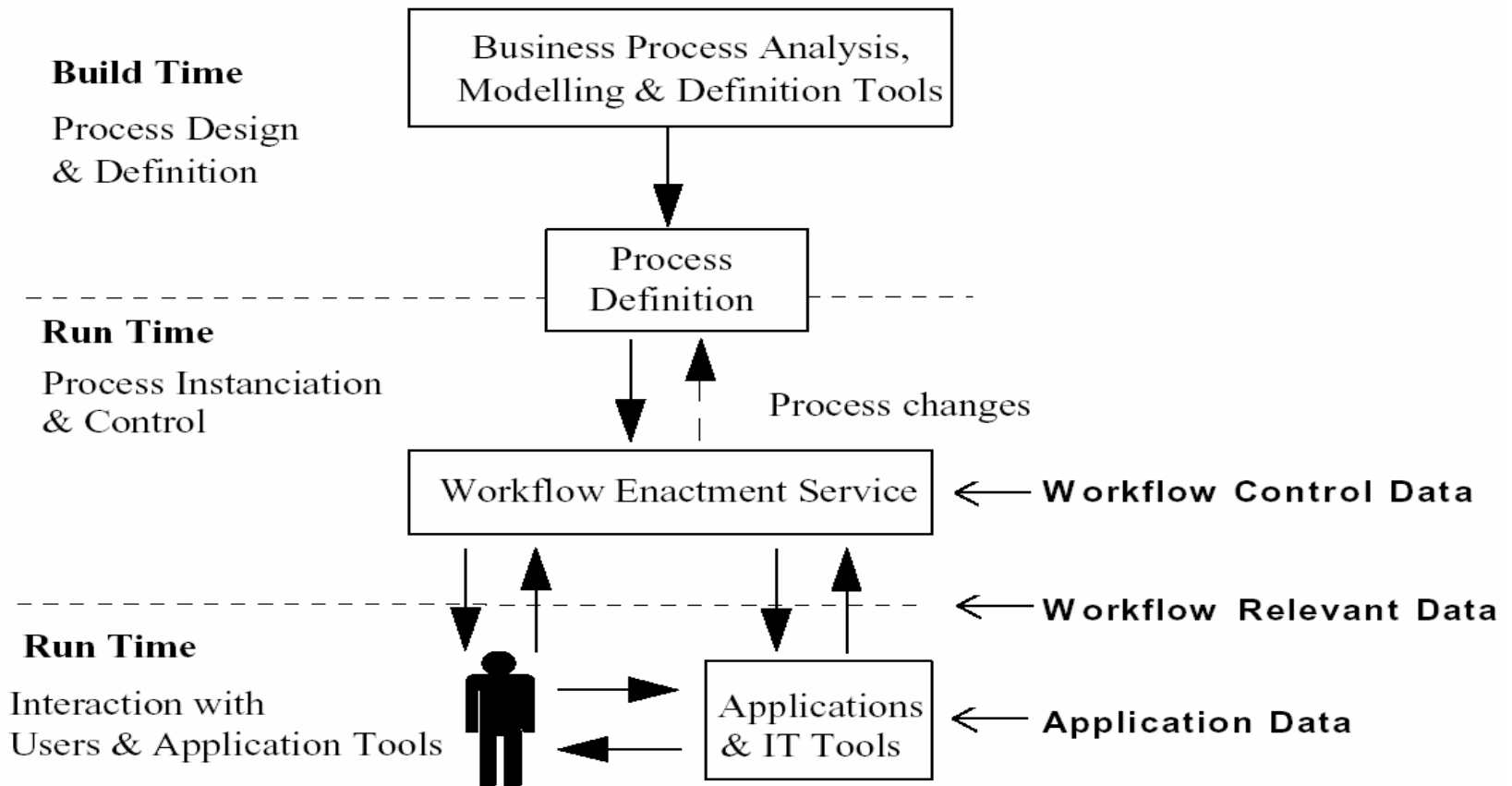
Was ist Workflow-Management?

- Workflow-Management ist die IT-basierte Unterstützung von Geschäftsprozessen. Dazu zählt:
 - Das Spezifizieren (Geschäftsprozessmodellierung)
 - Das Optimieren (Business Process Reengineering)
 - durch Simulation, Analyse und Reorganisation
 - Das Implementieren (Workflow-Modellierung)
 - Das Ausführen (Workflow-Management-System)
 - Die Qualitätssicherung (ISO 9000x)

Workflow-Management-System

- Ein Workflow-Management-System (WFMS) ist:
 - ein universelles Softwaresystem
 - zur Steuerung des Arbeitsflusses (Workflows)
 - nach den Vorgaben einer Spezifikation (Workflow-Modell).

Ausführung eines Workflows

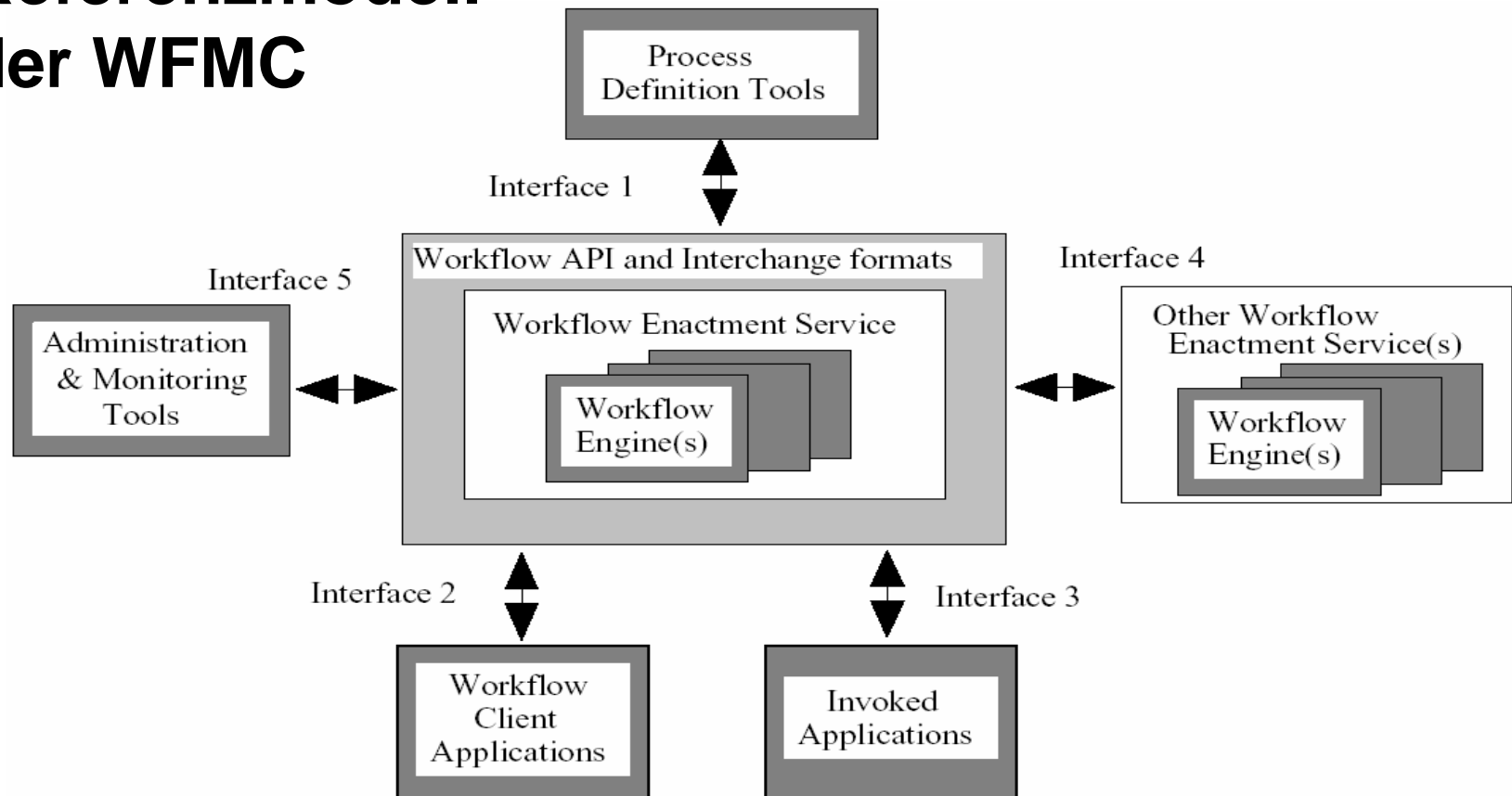


Workflow-Management-System

- Ein Workflow-Management-System (WFMS) unterstützt:
 - die Entwicklung (Modellierungskomponente),
 - die Ausführung (Laufzeitkomponente),
 - die Überwachung (Monitor-Komponente).

Architektur eines WFMS

Referenzmodell der WFMC



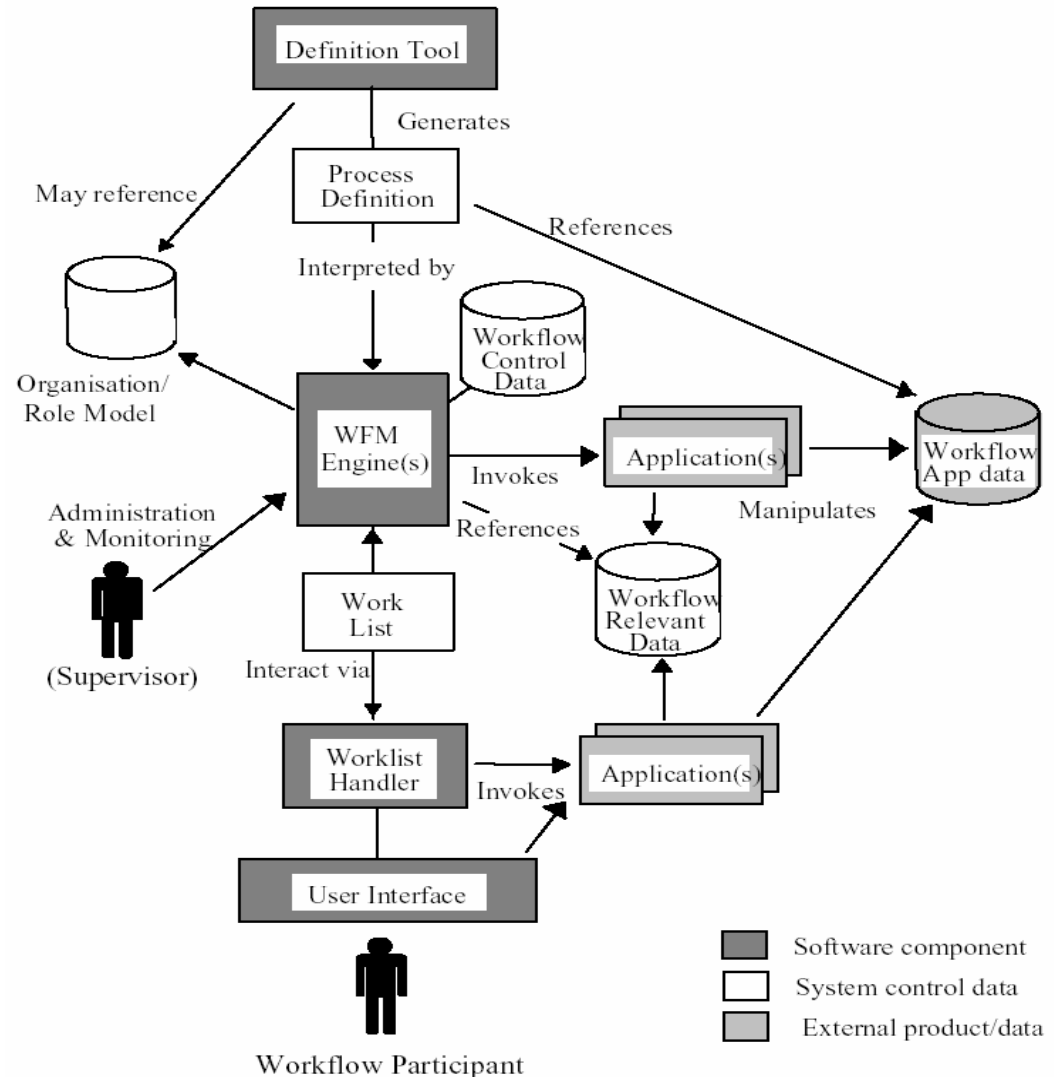
Workflow-Anwendung

- Eine Workflow-Anwendung ist:
 - eine implementierte und eingeführte Lösung
 - zur Steuerung des Arbeitsflusses (Workflows)
 - mit einem Workflow-Management-System.

Struktur einer Workflow-Anwendung

Eine WFA umfasst:

- das Workflow-Management-System,
- die Workflow-Modelle und Daten,
- die angebundenen Anwendungen.



Vorteile von Workflow-Management

- Automatisierung von Routineaufgaben
 - Qualitätssicherung der Prozesse
 - Nachweis der Protokolle
 - Möglichkeit der statistischen Auswertung
 - Strukturierung, Rationalisierung und Kontrolle der Abläufe
 - Synchronisation verteilter Prozesse
 - Motivation der Mitarbeiter
- Schneller! Besser! Billiger! Flexibler! ⇒ Erfolgreicher!

Nachteile von Workflow-Management

- Hardware-Kosten
 - Software-Kosten + Kosten für Tailoring
 - Schulungskosten
 - Wartungskosten + Kosten für Reorganisation
 - Medienbrüche
 - Probleme der Akzeptanz bzw. Motivation
 - Rechtliche Schwierigkeiten
 - Datenschutz
- Genaue Nutzen/Kosten-Analyse + Einbindung der MA

Was ist ein Service?

Beispiel: Geschäftsprozess „Online-Aktienhandel“ ist Zusammenspiel folgender Services:

- Wertpapierservice: nennt handelbare Papiere
- Marktdatendienst: Papier→aktueller Kurs
- Depotdienst: Daten zu Depot des Anwenders
- Orderdienst: Abwicklung von Kauf/Verkauf
- Archivdienst: Protokollierung der Abläufe

(Quelle: de.wikipedia.org „Serviceorientierte Architektur“)

Was ist ein Service?

Ein Service ist

- eine Komponente, die
- eine wohldefinierte Funktionalität
- über eine standardisierte Schnittstelle

anderen Services bzw. Anwendungen zur Verfügung stellt.

Wozu Services?

- Strukturierung organisationsübergreifender Zusammenarbeit
- Leichte Austauschbarkeit der Komponenten
- Leichte Aggregation vorhandener zu neuen Services („virtuelle Unternehmen“)
- Kapselung der Funktionalität
- Beherrschung heterogener Infrastrukturen

Übersicht Vorlesungsinhalt

1. Modellierungssprachen für Geschäftsprozesse
EPK, Aktivitätsdiagramme, Petrinetze
2. Analysemethoden für Petrinetze
Techniken für GP-spezifische Eigenschaften
3. Modellierung und Analyse quantitativer Aspekte
zeitbewertete, stochastische Petrinetze
4. Workflow-Mining
Auf Basis von Petrinetzen
5. Modellierung von Services
WSDL, BPEL, ..., Transformation in Petrinetze
6. Algorithmen für Services
Bedienungsanleitungen, Austauschbarkeit, ...

Teil 1

Modellierungssprachen für Geschäftsprozesse

Elemente eines Prozessmodells

- Aktivitäten, z.B.
 - Angebot erstellen
 - Rechnung ausstellen
 - Angebot akzeptieren
- Ereignisse, z.B.
 - Angebot eingetroffen
 - Kreditwürdigkeit bestätigt
- Kausale Abhängigkeiten zwischen diesen, z.B.
 - vor
 - nach
 - nebenläufig zu
 - alternativ zu

Eigenschaften eines Prozessmodells

Ereignisgesteuert (event driven)

im Gegensatz zu

- getaktet
- zeitgesteuert (time triggered)

Diskret

im Gegensatz zu

- kontinuierlich
- hybrid

Ressourcenorientiert (produzieren, konsumieren)

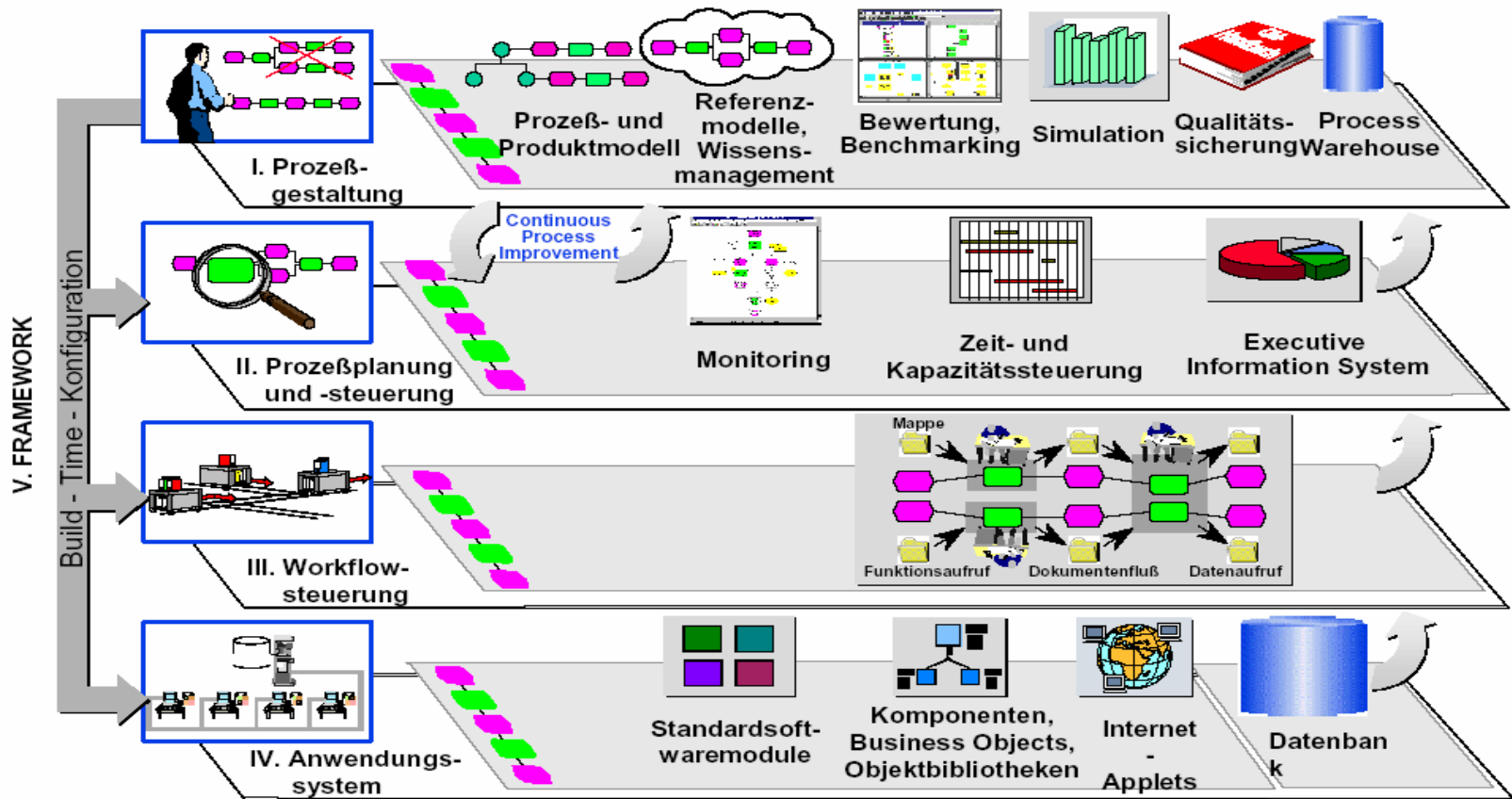
im Gegensatz zu

- Wertorientiert (lesen, schreiben)

Ereignisgesteuerte Prozessketten (EPK)

- Semiformal (ohne mathematisch präzise Semantik)
- Werkzeugunterstützt (ARIS Toolset, SAP Business workflow)
- In kleinen und mittleren Unternehmen weit verbreitet für
 - Business Process Reengineering
 - Activity based costing
 - Dokumentation (für ISO 9000)
 - Spezifikation und Steuerung von Workflows
- Prof. A.-W. Scheer: *ARIS - Modellierungsmethoden, Metamodelle, Anwendungen*. 3. Auflage, Springer, Berlin 1998

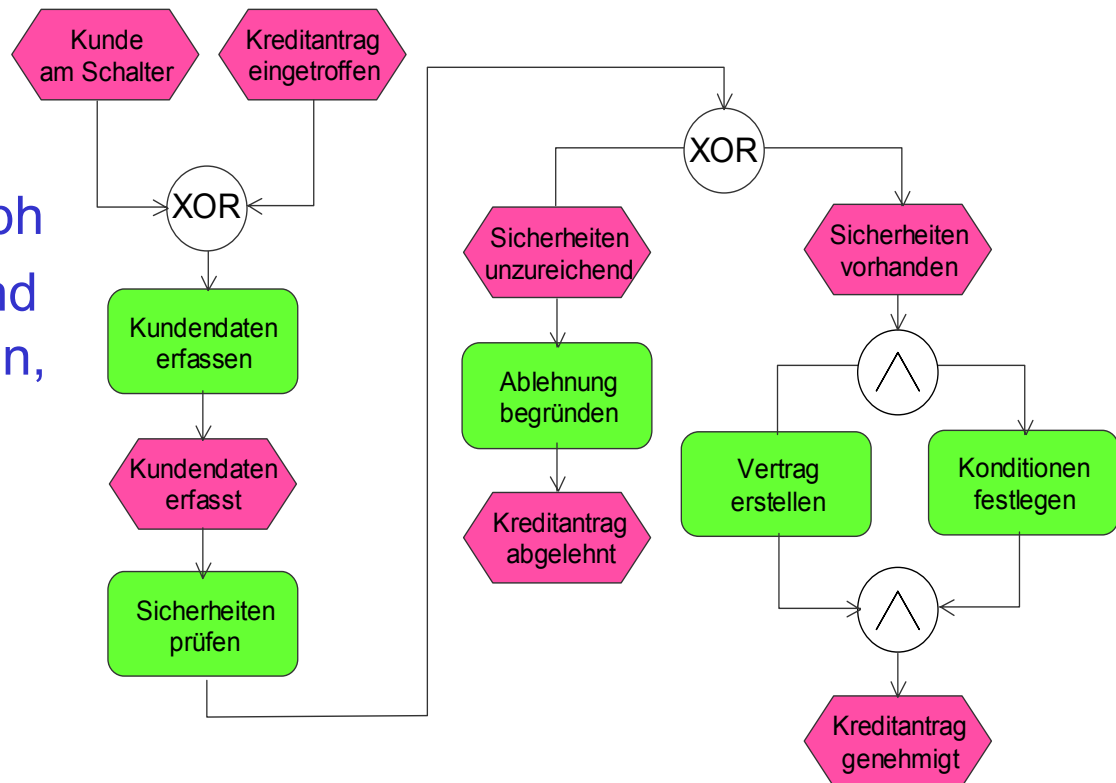
Geschäftsprozesse mit EPK's



Modellierung

Eine EPK

- ist ein gerichteter Graph
- besteht aus aktiven und passiven Komponenten,
- Knoten sind
 - Funktionen,
 - Ereignisse,
 - Konnektoren.
- Kanten beschreiben
 - Daten- und Kontrollfluss,
 - sowie Zuordnungen



Grundelemente

- Grundelemente:

- Ereignis



- Funktion



- Verknüpfung



- Kontrollfluss



- Erweiterungen

- Informationsobjekte

- Informationsfluss

- Organisationseinheiten

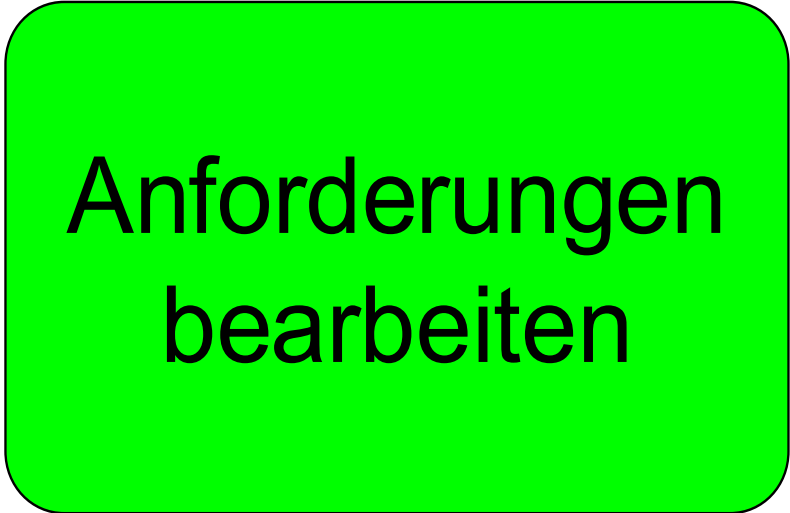
- Zuordnung

- Prozesswegweiser

Funktionen

Aktive Komponenten

- Zugriff auf Daten
 - Konsumieren,
 - Transformieren,
 - Transportieren,
 - Produzieren
- Besitzen Entscheidungskompetenz
- Verbrauchen Zeit



Anforderungen
bearbeiten

Attribute von Funktionen

- Basisattribute
 - Name, ID
 - Beschreibung/Definition
 - ...
- Zuordnung
 - intern/extern
 - ...
- Bearbeitungsart
 - automatisiert, manuell
 - ...
- Dauer
 - Bearbeitungszeit
 - Liegezeit
 - ...
- Kosten
 - Materialkosten
 - Personalkosten
 - ...

Ereignisse

Passive Komponente

- Repräsentiert Zustand eines Informationsobjekts
- löst Funktionen aus,
- ist Ergebnis von Funktionen,
- ist auf einen Zeitpunkt bezogen,
- beeinflusst weiteren Ablauf des Geschäftsprozesses



Attribute von Ereignissen

- Basisattribute
 - Name, ID
 - ...
- Ereignisherkunft
 - intern/extern
 - ...
- Ereignisart
 - automatisiert, manuell
 - ...
- Klassifikation
 - Trigger
 - Nebenbedingung
 - Zustand
 - ...
- Rolle
 - Start-/Zwischen-/Endereignis
 - Auslöse-/Bereitstellungseignis
 - ...

Arten von Ereignissen

- Erzeugung eines neuen Prozessobjekts (Create)
 - Beispiel: " Stammsatz ist angelegt "
- Finaler Status eines Prozessobjekts (Delete)
 - Beispiel: " Auftrag ist erledigt "
- Attributänderung eines Prozessobjekts (Update)
 - Beispiel: " Rechnung ist geprüft "
- Eintreffen eines bestimmten Zeitpunkts
 - Beispiel: " Mahntermin ist erreicht "
- Bestandsänderung, die einen Prozess(schritt) auslöst
 - Beispiel: " Kreditlimit ist überschritten "

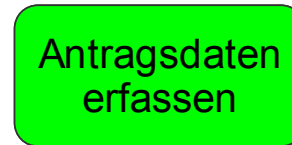
Abstraktions- vs. Ausprägungsebene

*Abstraktions-
ebene*

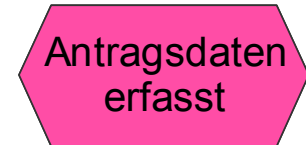
Ereignistyp



Funktionstyp



Ereignistyp



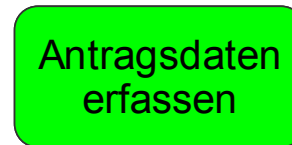
Abstraktions- vs. Ausprägungsebene

**Abstraktions-
ebene**

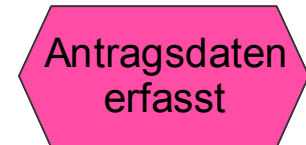
Ereignistyp



Funktionstyp



Ereignistyp



**Ausprägungs-
ebene**

Ereignis

Peter Becker aus
Klingenbach hat einen
Kreditantrag gestellt

Funktion

Name: Peter Becker
Ort: Klingenbach
Alter: 36
Kredit: 150.000
in Kundendatei
eintragen

Ereignis

Die Daten über
Peter Becker sind
in der Kundendatei
enthalten

Tanja Müller aus
Beeringen hat einen
Kreditantrag gestellt

Name: Tanja Müller
....
in Kundendatei
eintragen

Die Daten über
Tanja Müller sind in
der Kundendatei
enthalten

Prozessmodellierung betrachtet Abstraktionsebene

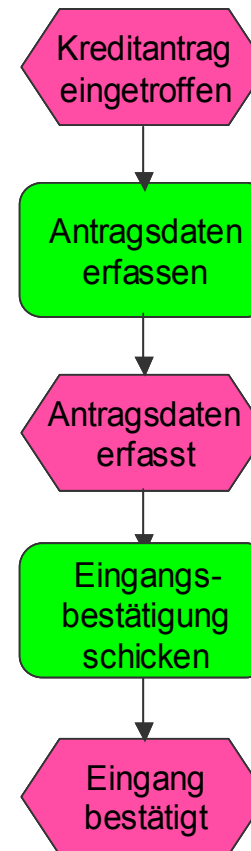
Gliederung

- Hintergrund ✓
- Modellierung ⇐
 - Grundelemente ✓
 - Kontrollstrukturen
 - Regeln der Modellierung
- Erweiterte Konzepte

Verknüpfungen: Sequenz

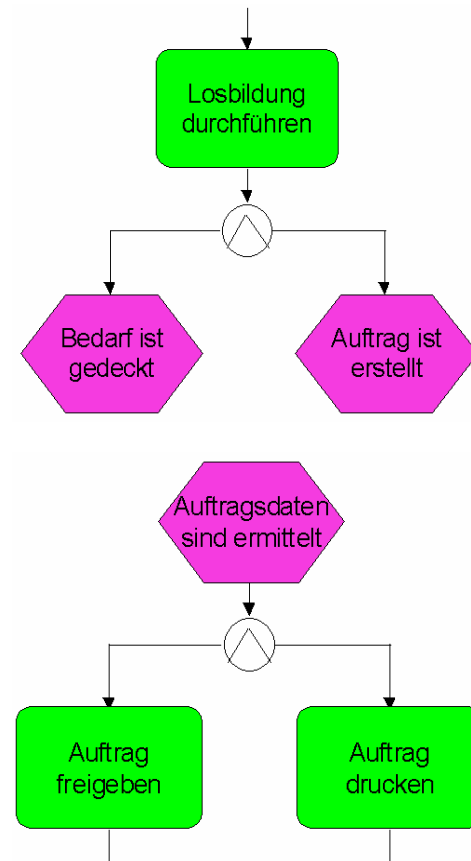
EPK sind bipartite
Graphen:

- Ereignisse lösen Funktionen aus.
- Funktionen erzeugen Ereignisse.



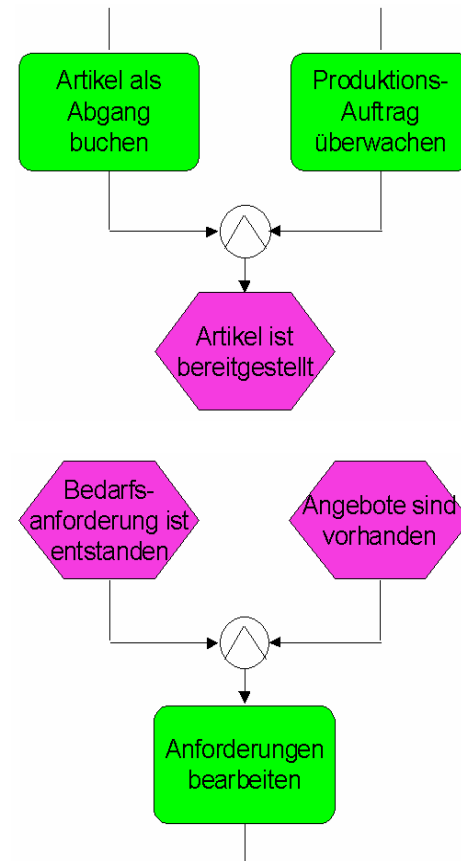
Verknüpfungen: AND-Split

- Nach der Funktion treten zwei Ereignisse auf
- Ein Ereignis ist Auslöser für zwei Funktionen



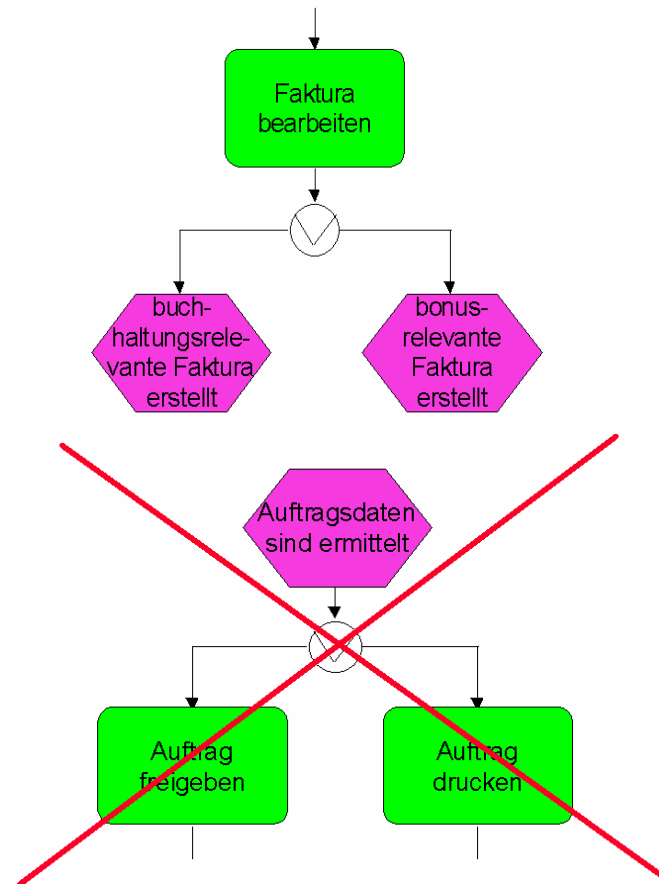
Verknüpfungen: AND-Join

- Das Ereignis tritt nach zwei Funktionen auf
- Zwei Ereignisse sind Auslöser für die Funktionen



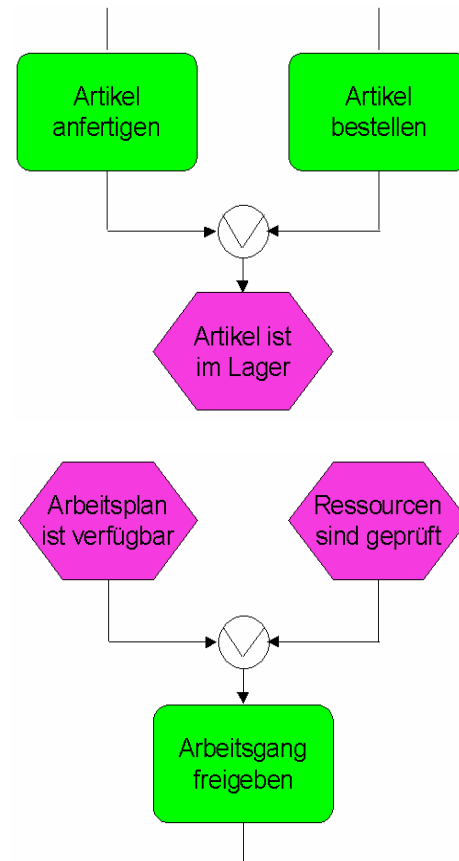
Verknüpfungen: OR-Split

- Nach der Funktion sind mehrere Fortsetzungen möglich
- Nicht erlaubt
 - Ereignisse haben keine Entscheidungskompetenz



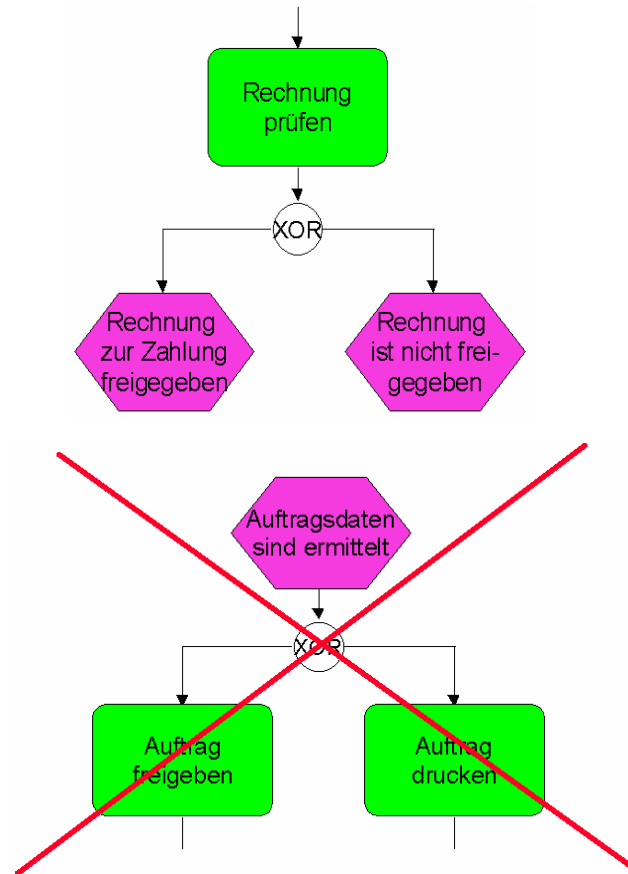
Verknüpfungen: OR-Join

- Das Ereignis tritt nach mindestens einer Funktion auf
- Mindestens ein Ereignis ist Auslöser für die Funktionen



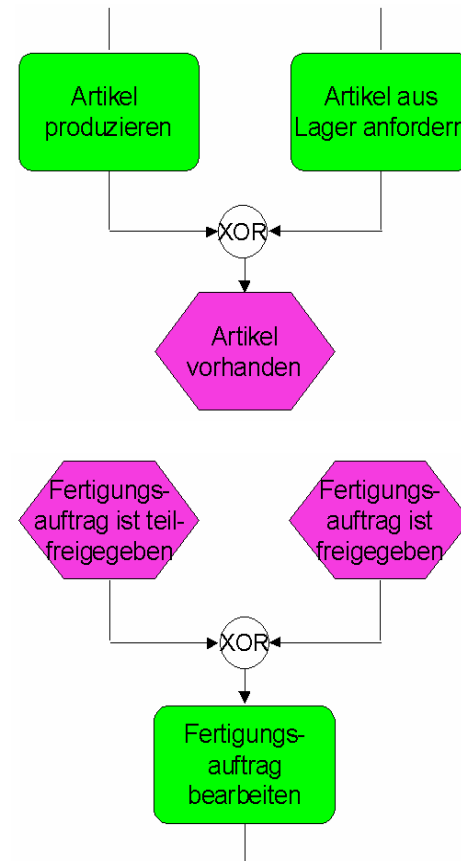
Verknüpfungen: XOR-Split

- Nach der Funktionen ist genau eine Fortsetzung möglich
- Nicht erlaubt
 - Analog zum OR-Split



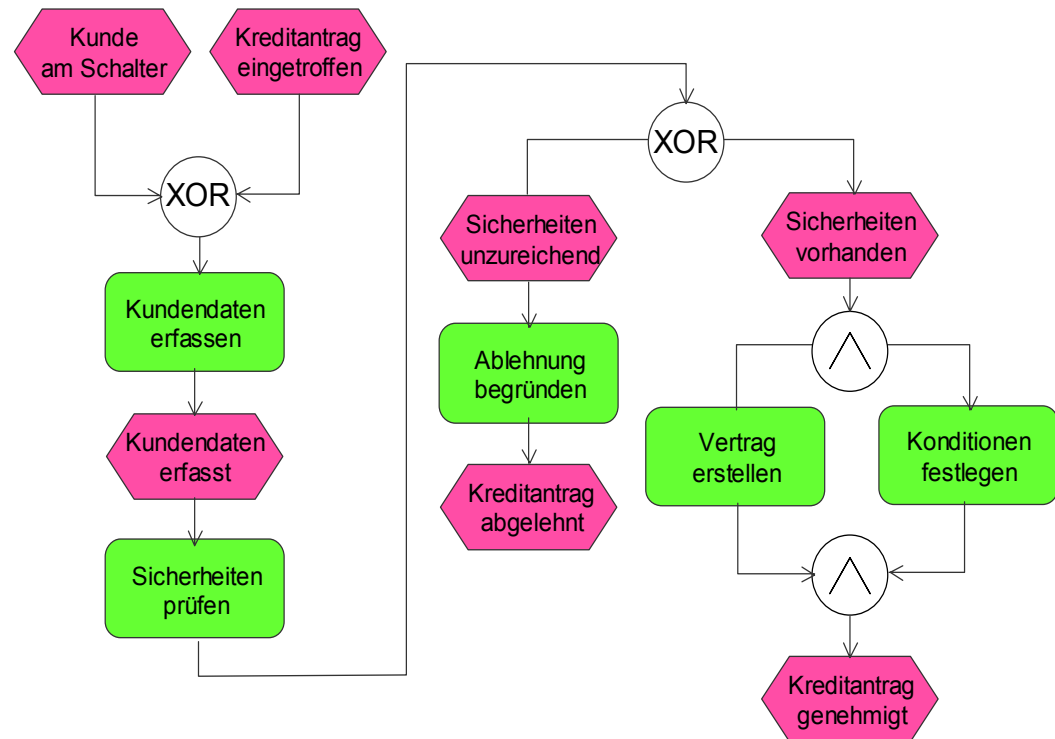
Verknüpfungen: XOR-Join

- Das Ereignis tritt nach genau einer der beiden Funktionen auf
- Genau ein Ereignis ist Auslöser der Funktionen



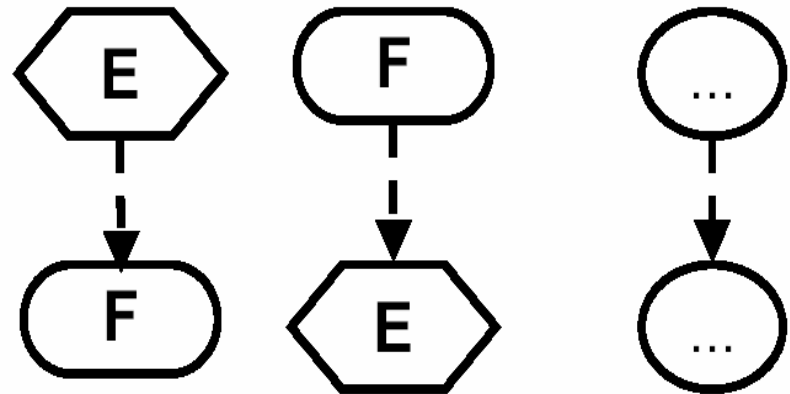
Regel 1

- Jede EPK beginnt und endet mit mindestens je einem Ereignis.
- **Ausnahme:** Verweis auf eine andere EPK.



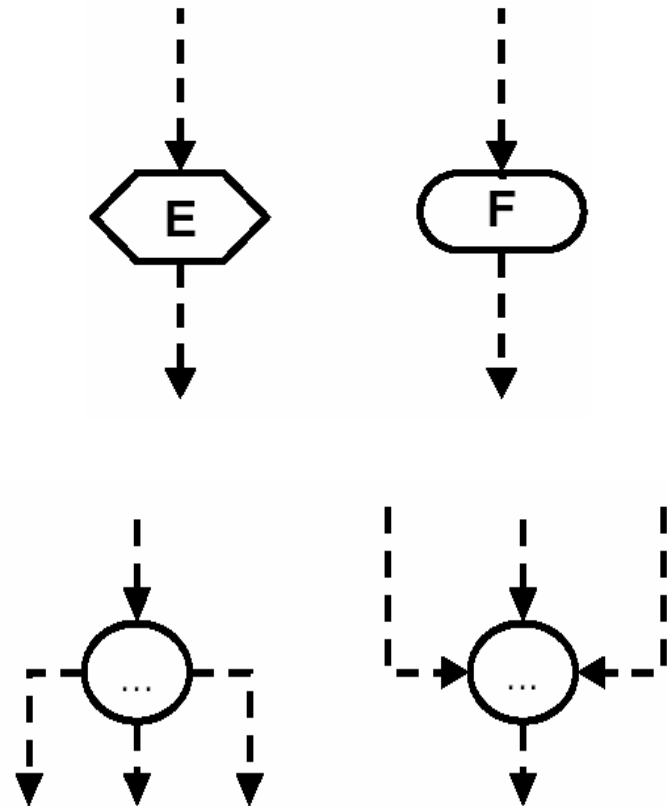
Regel 2

- Jede Kante verbindet zwei Knoten von jeweils unterschiedlichem Typ
- **Ausnahme:** Auf einen Konnektor darf ein Konnektor folgen.



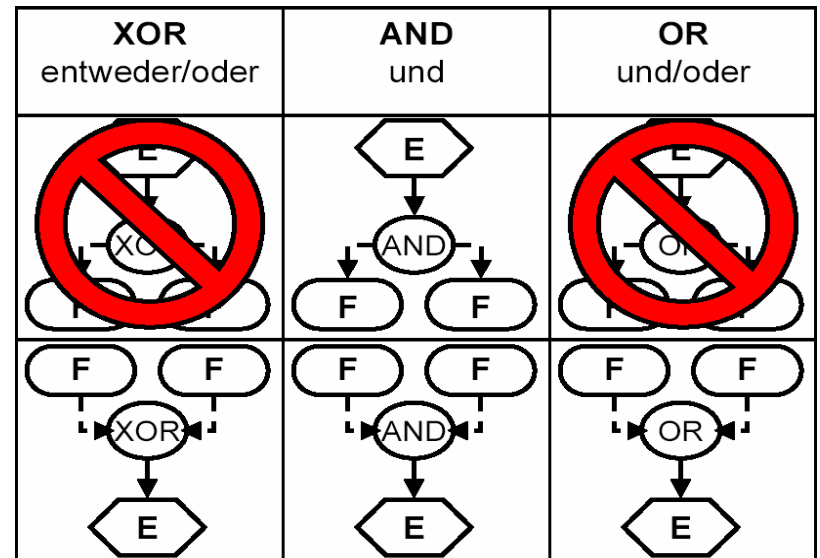
Regel 3

- Kontrollfluss verzweigt und vereinigt sich nur an Konnektoren.



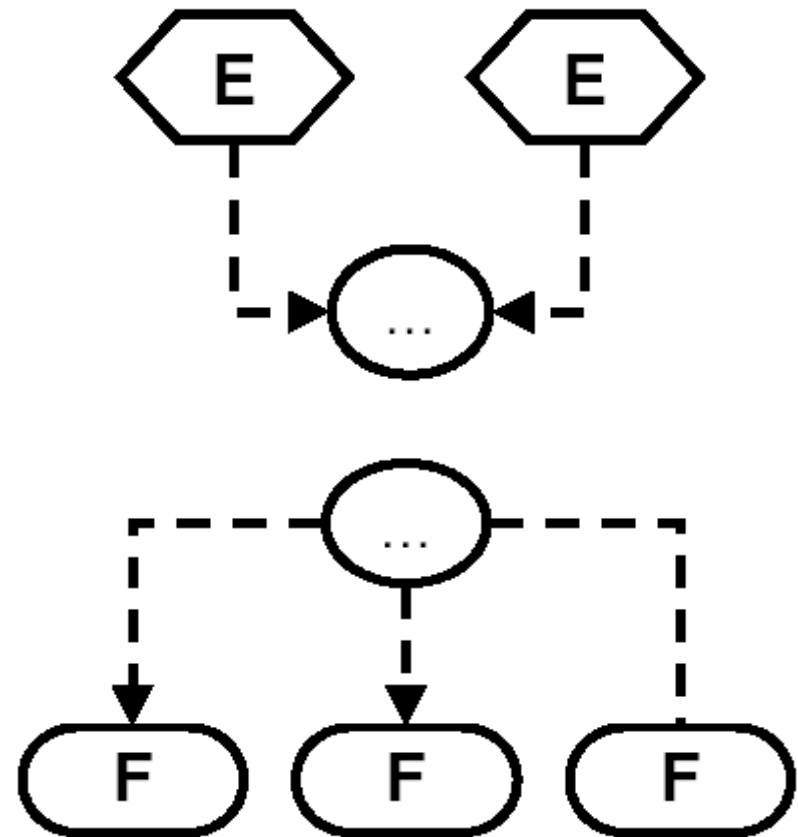
Regel 4

- Auf Ereignisse dürfen keine OR- bzw. XOR-Konnektoren folgen.



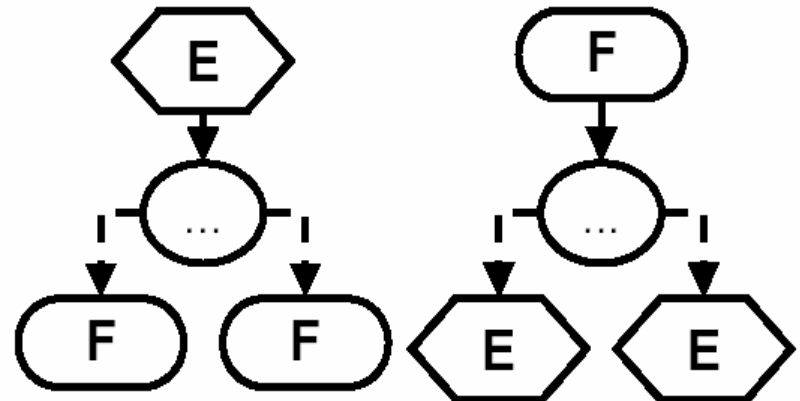
Regel 5

- Alle Ein-/ Ausgänge der Konnektoren sind vom gleichen Typ.



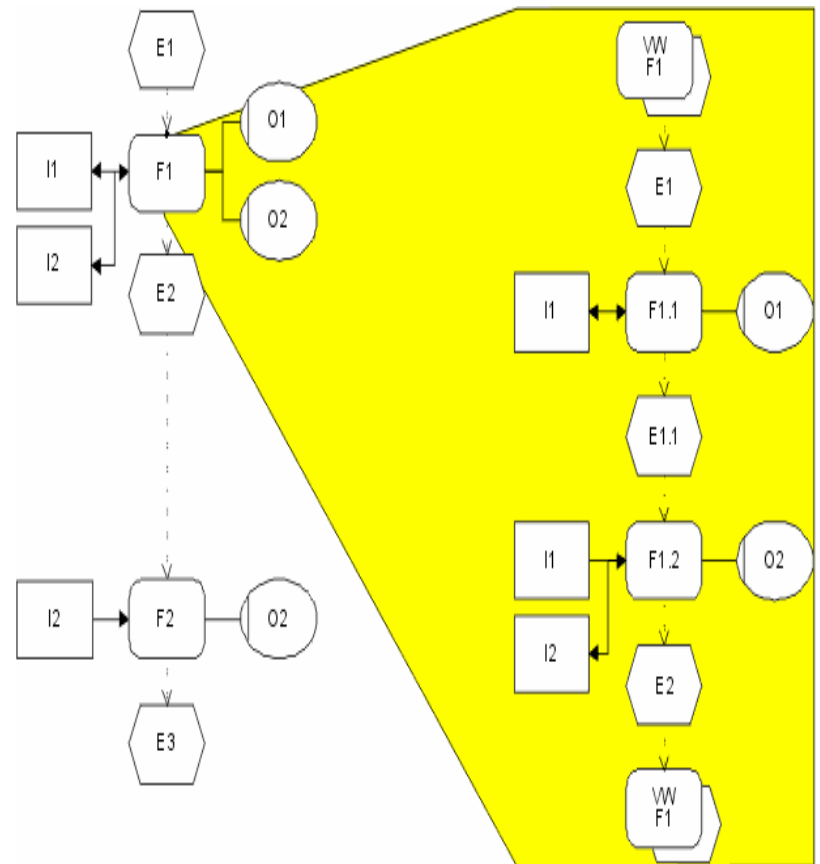
Regel 6

- Konnektoren verbinden stets Ereignisse mit Funktionen bzw. Funktionen mit Ereignissen.



Regel 7

- Eine Funktion kann durch eine weitere EPK verfeinert werden.

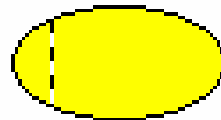


Erweiterte EPK (eEPK)

Prozeßweg-
weiser



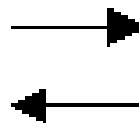
Organisations-
einheit



Informations-
objekt



Informations-
fluß



Zuordnung von
Systemorganisa-
tionseinheiten



Der *Prozeßwegweiser* zeigt die Verbindung von einem zu einem anderen Prozeß (Navigationshilfe).

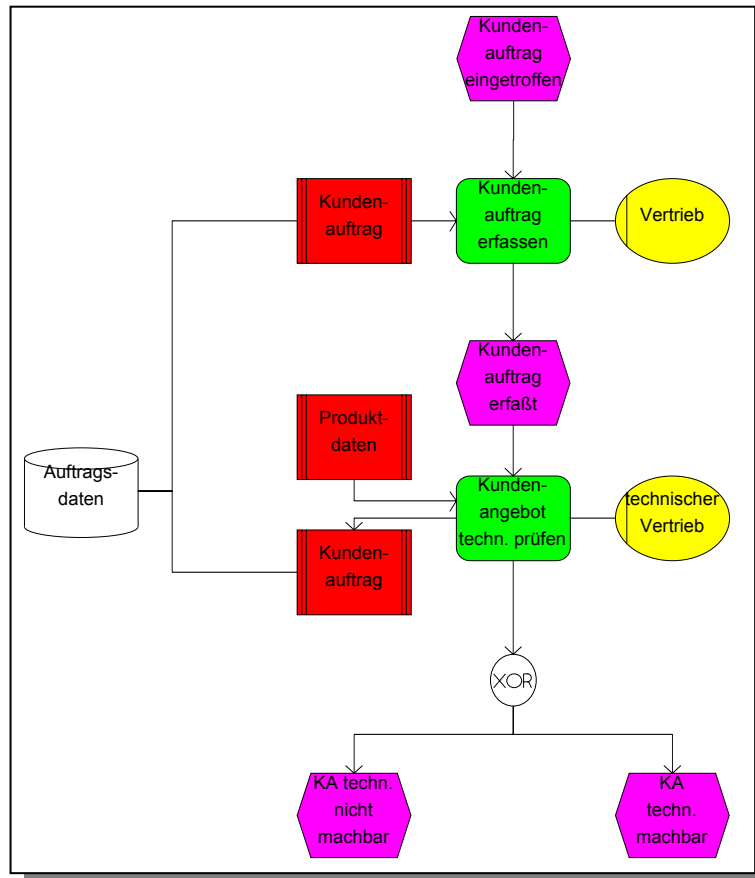
Die *organisatorische Einheit* beschreibt die Gliederungsstruktur eines Unternehmens.

Ein *Informationsobjekt* ist eine Abbildung eines Gegenstands der realen Welt (Business Object).

Der *Informationsfluß* beschreibt, ob von einer Funktion gelesen, geändert oder geschrieben wird.

Die *Zuordnung von Systemorganisationseinheiten* beschreibt, welche Einheit (Mitarbeiter) die Funktion bearbeitet.

Erweiterte EPK (eEPK)

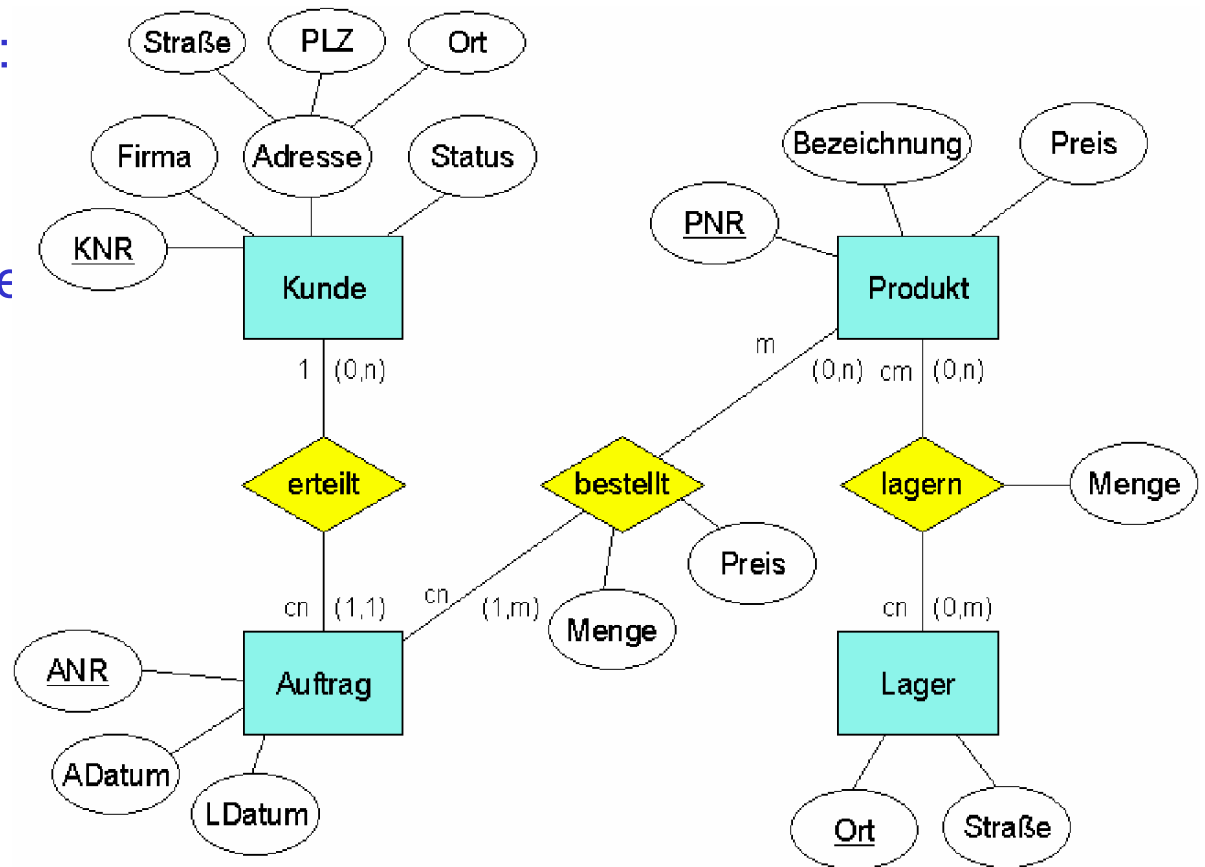


Erweiterung der EPK

- Input-Output-Beziehung von Daten
- Verbindung von Funktionen mit Organisationseinheiten
- Anwendungssysteme für automatisierte Funktionen
- Medien für Kommunikation, Speicherung usw. (z.B. Fax, Drucker, ...)
- materielle und immaterielle Leistungen, Ergebnisse

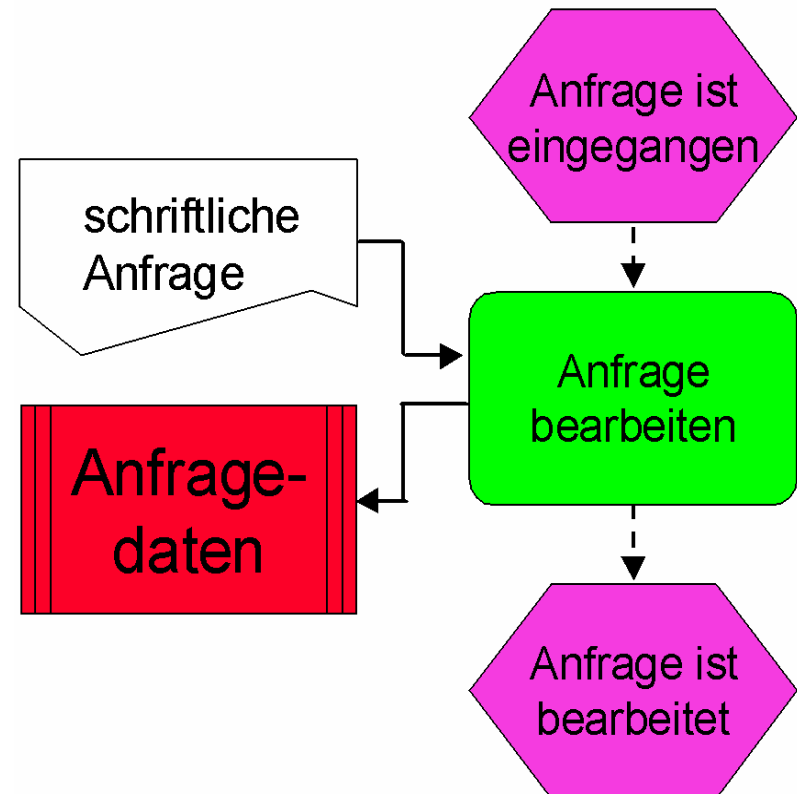
Datenmodellierung

- ER-Diagramm:
 - Objekte
 - Attribute
 - Beziehungen



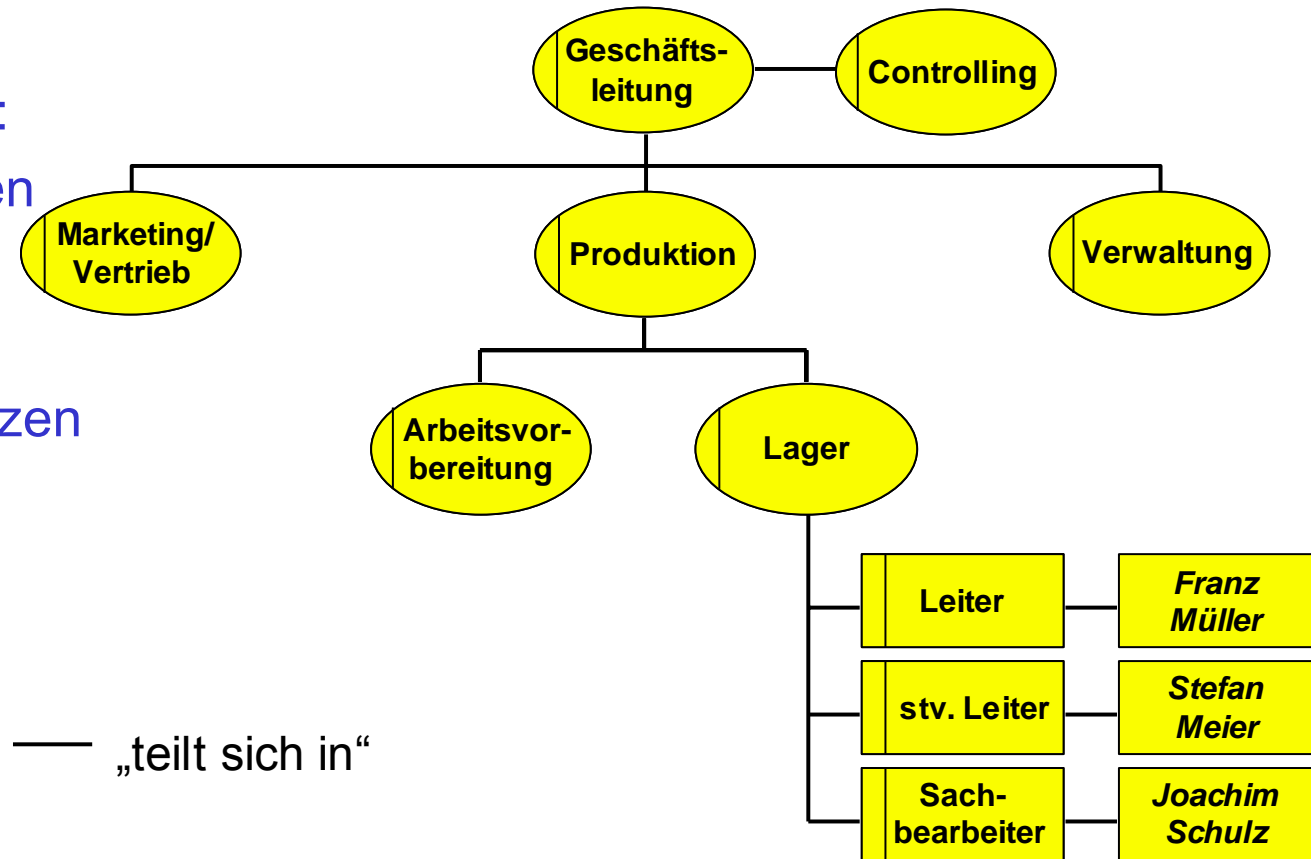
Integration von Daten

- Input- und Output-Parameter
- Datenfluss
 - Datencontainer
 - Dokumente



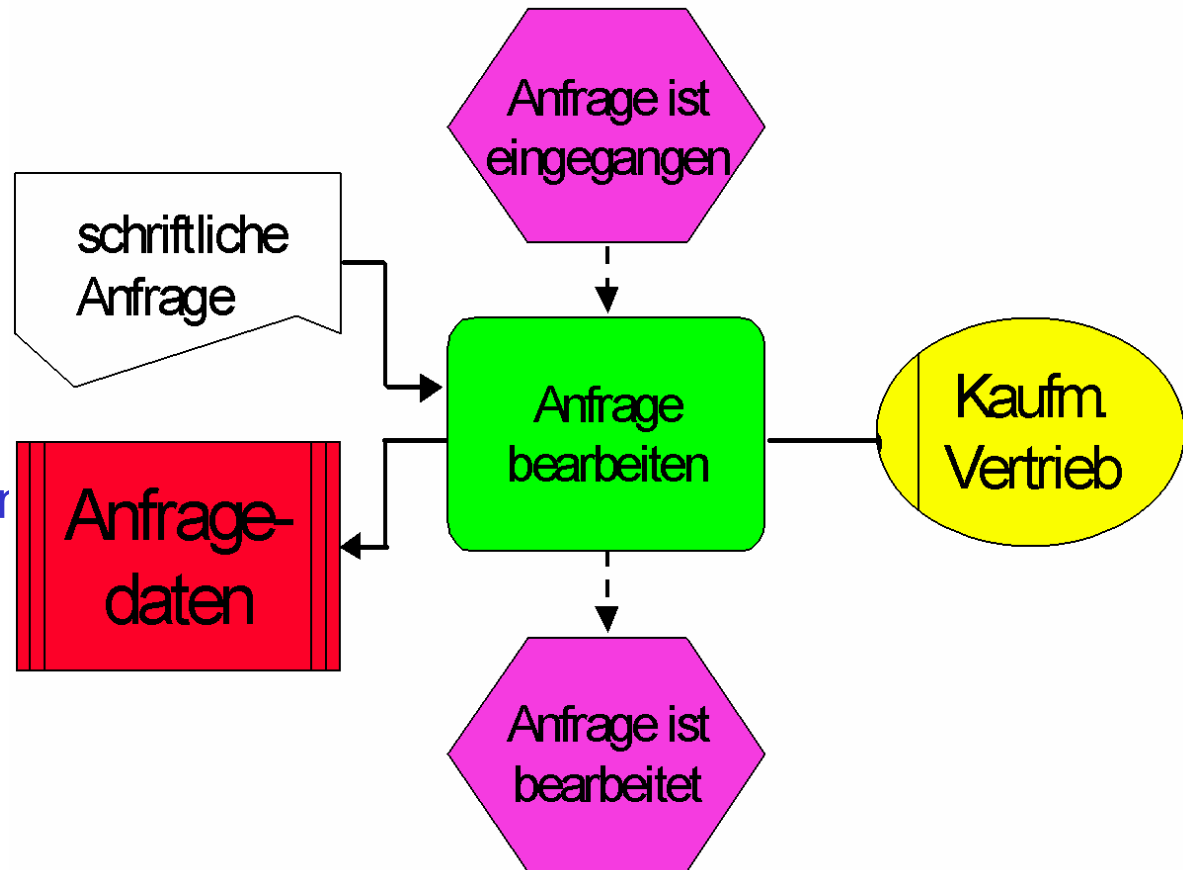
Organisationsstruktur

- Organigramm:
 - Abteilungen
 - Rollen
 - Personen
 - Kompetenzen

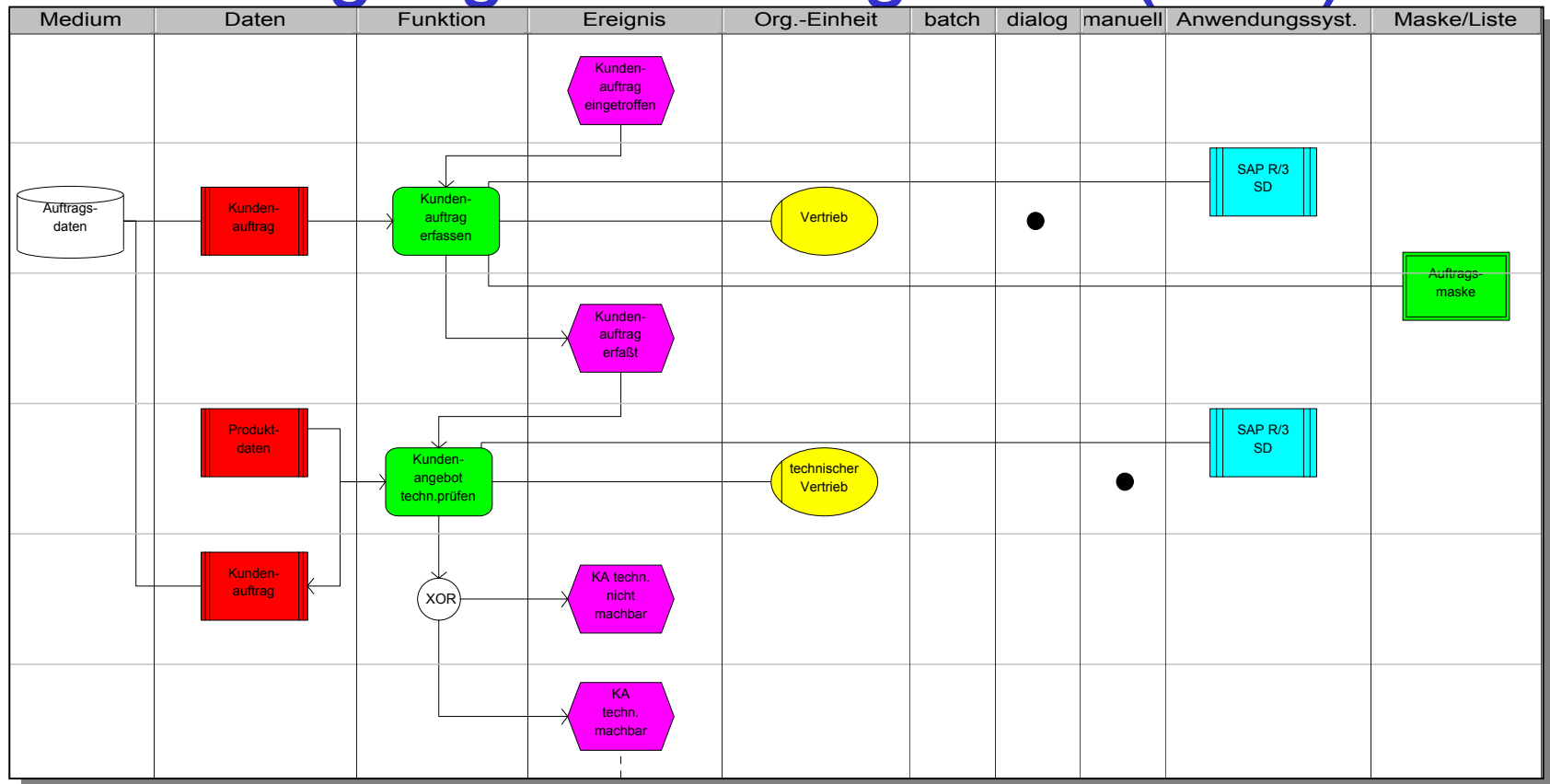


Integration der Org.-Struktur

- Zuordnung von
 - Rolle,
 - Kompetenz,
 - Berechtigung,
 - Verantwortung,
 - Organisationseinheit
- zu Funktionen.



Vorgangskettendiagramm (VKD)



Nachteil: Prozesse mit vielen Verzweigungen und Schleifen sind aufgrund der Spaltenanordnung schlecht darstellbar

Zusammenfassung EPK

- Vorteile:
 - Hohe Anschaulichkeit
 - Gute Integration verschiedener Sichten
 - Realitätsgetreue und exakte Abbildung der Geschäftsprozesse
 - Gute Toolunterstützung (ARIS Toolset)
- Nachteile:
 - Statische Sicht auf Prozessstrukturen
 - Simulation nur mit Erweiterungen
 - Keine mathematische Fundierung
 - Keine Möglichkeiten zur Bewertung/Optimierung

Zusammenfassung

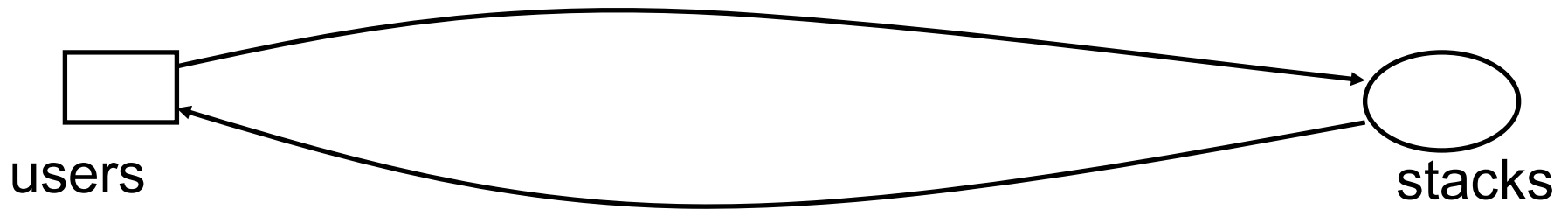
- Ereignisgesteuerte Prozessketten haben sich in der Praxis - zumindest im deutschsprachigen Raum - als Beschreibungsmittel für betriebliche Abläufe etabliert.
- EPK's sind nur informal eingeführt worden, es gibt keine exakte Semantikdefinition.
- Weitergehende Arbeiten versuchen die anschaulichen EPK's mit Analysemethoden (Simulation, Validierung, Verifikation) zu verbinden.

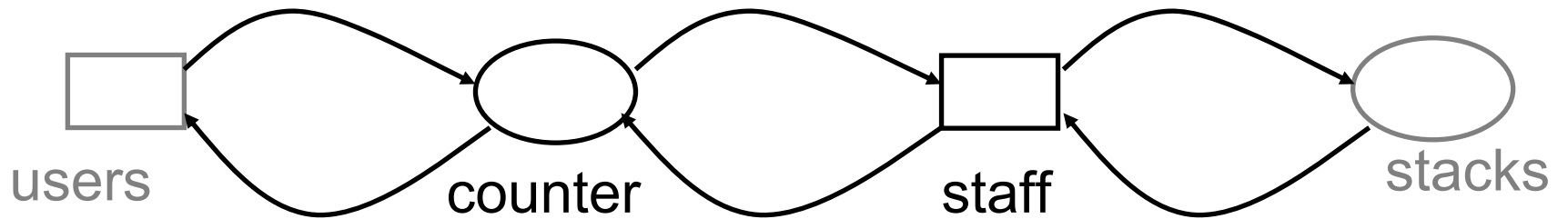
UML-Aktivitätsdiagramme

→ www.oose.de

Petri Netze

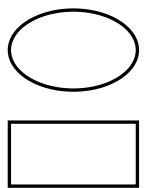
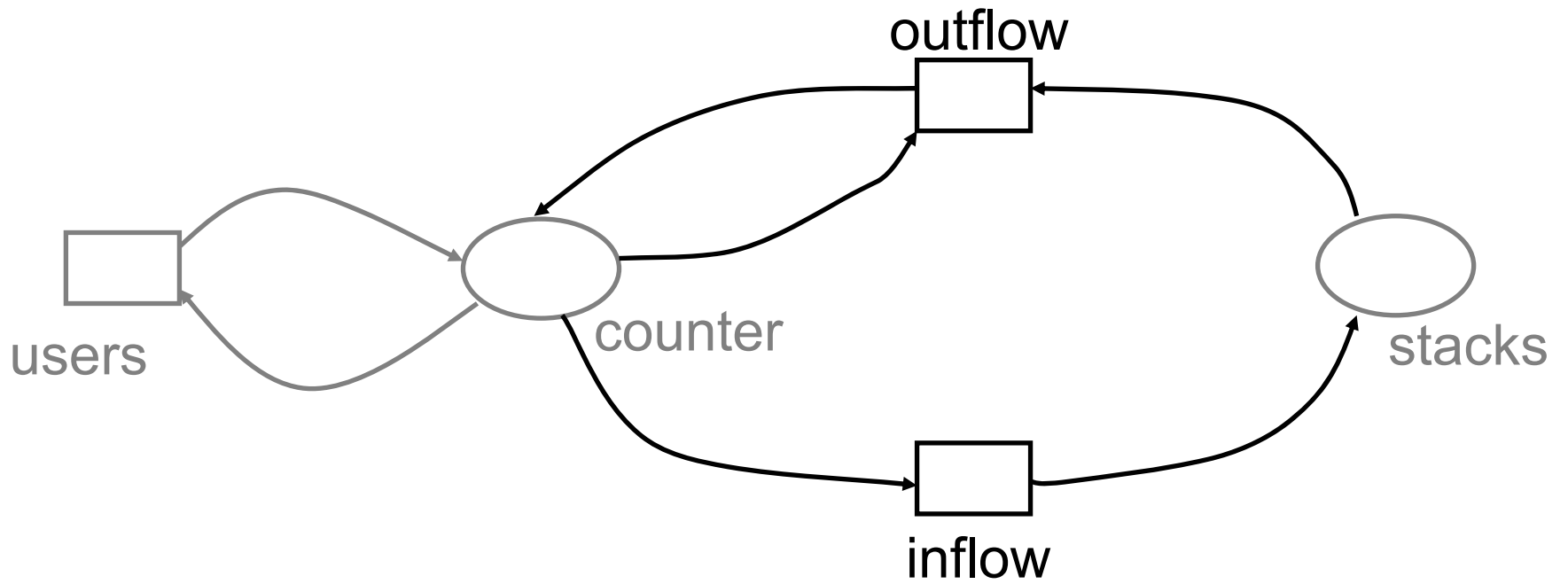
1. Beispiel: Bibliothek



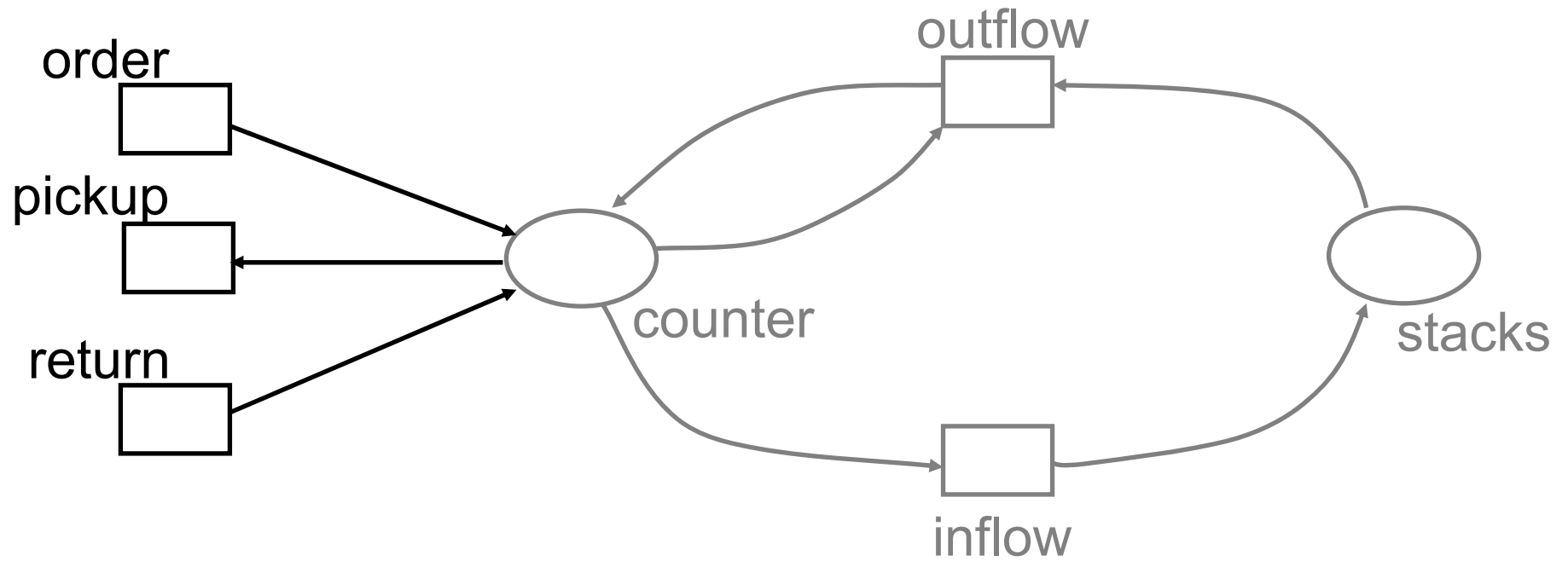


Pfeile: Direkte Verbindung; Zugriff

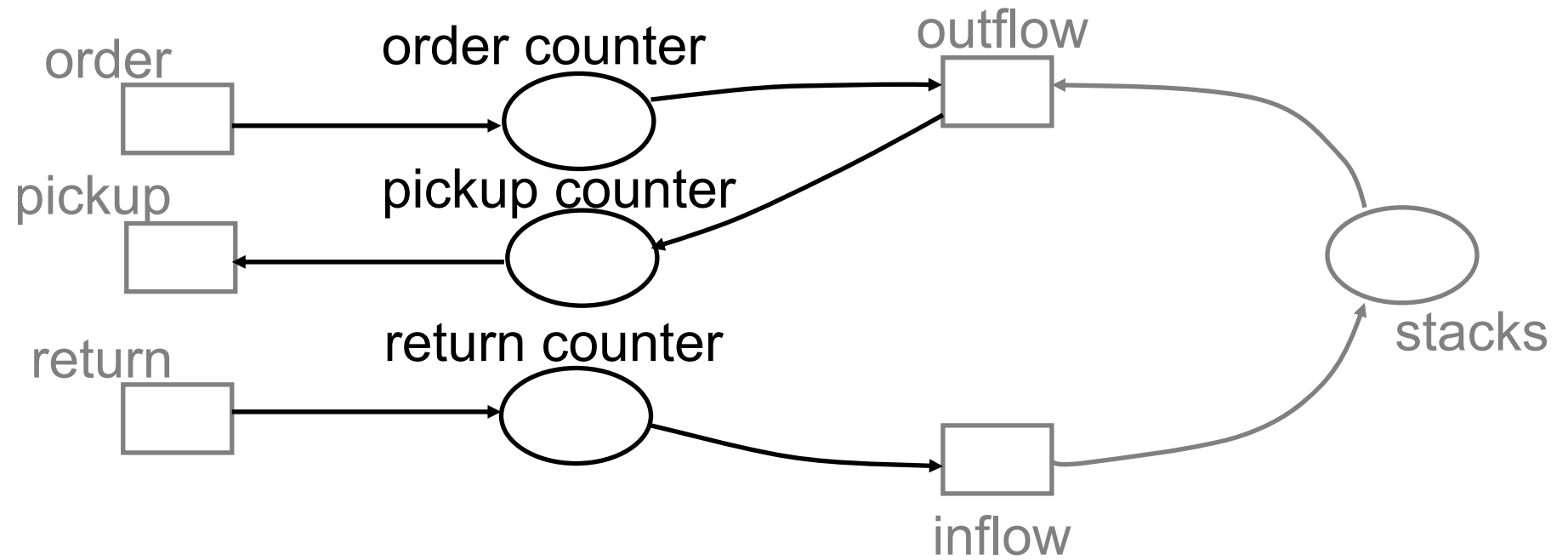
Also: User haben keinen direkten Zugriff zum Lager

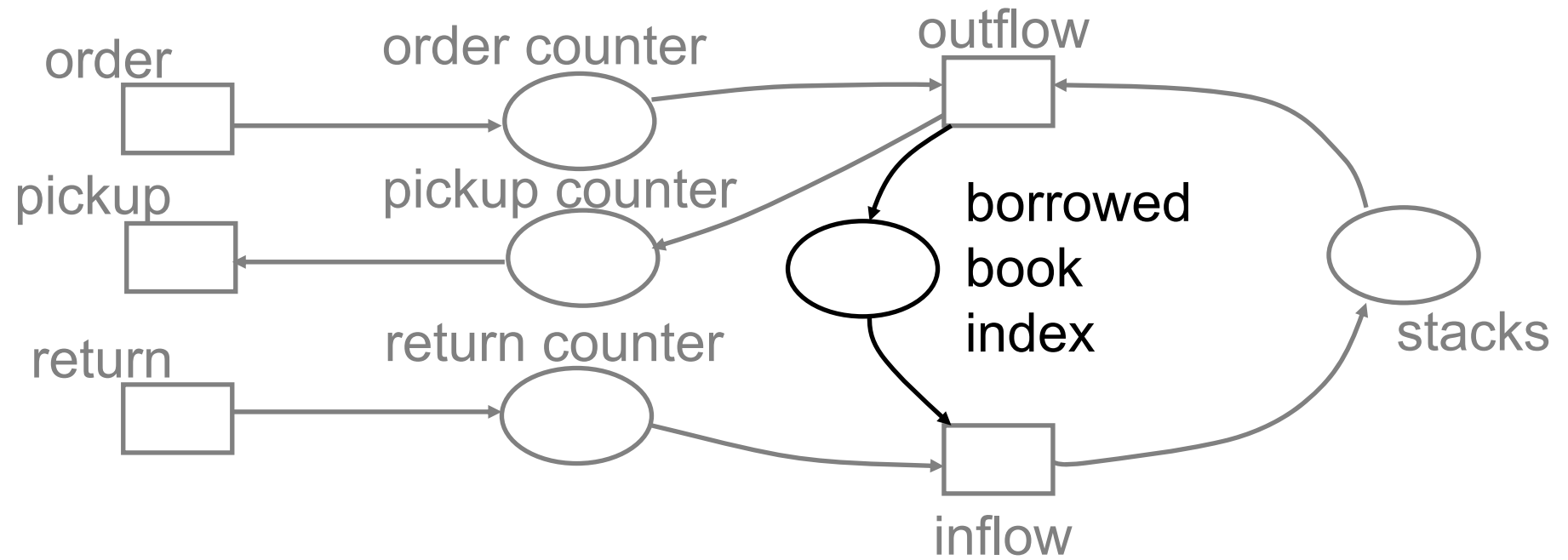


Passive Komponente
Aktive Komponente



Verschiedene Aktivitäten von Nutzern





2. Passive und Aktive Komponenten

Passive Komponente



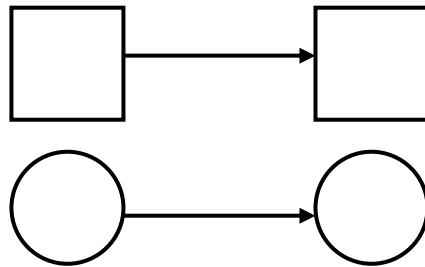
- Anwesendes (Dinge, Information, Bits...)

active component

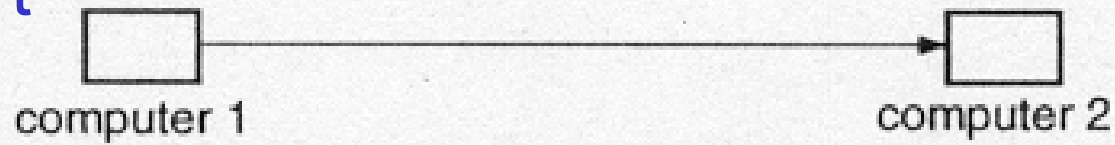


- Ändert passive Komponenten
- produziert Dinge
- konsumiert Dinge
- verschiebt Dinge

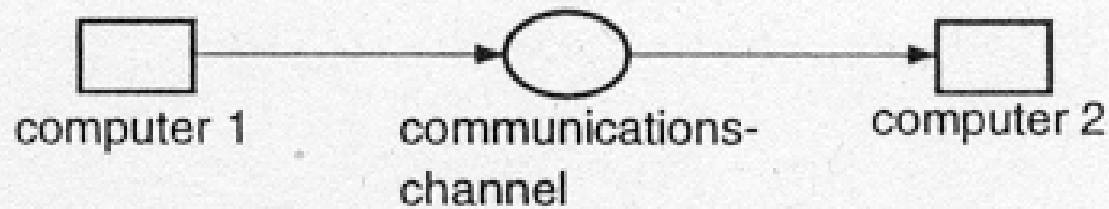
Nie
Nie

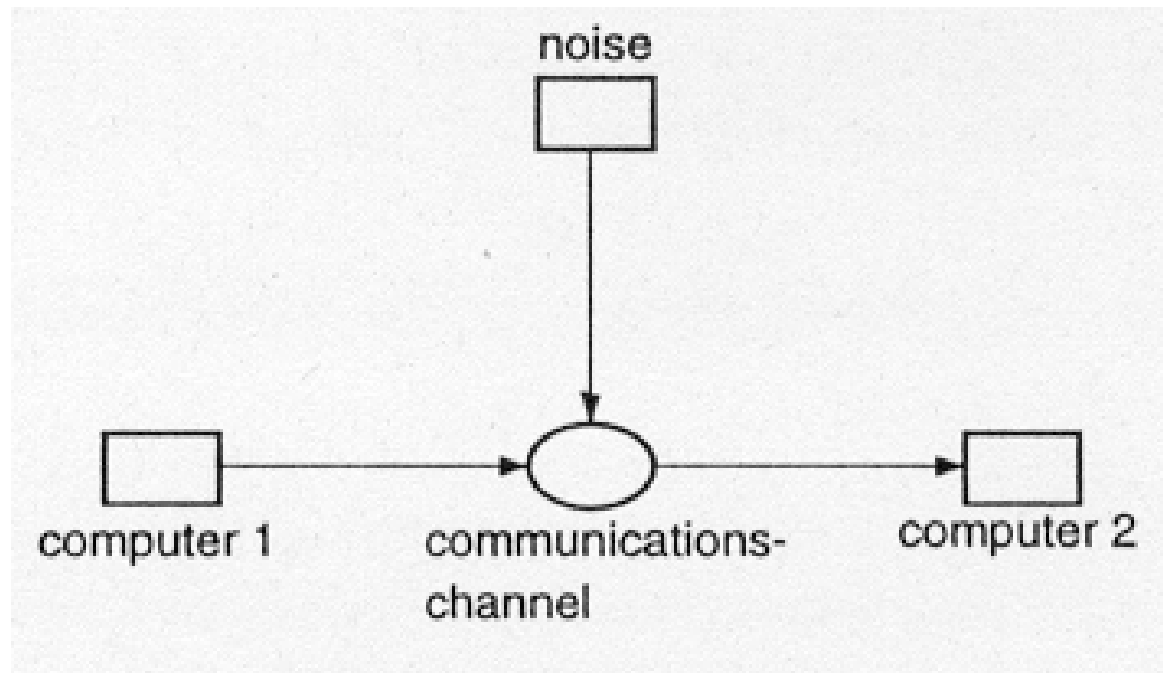


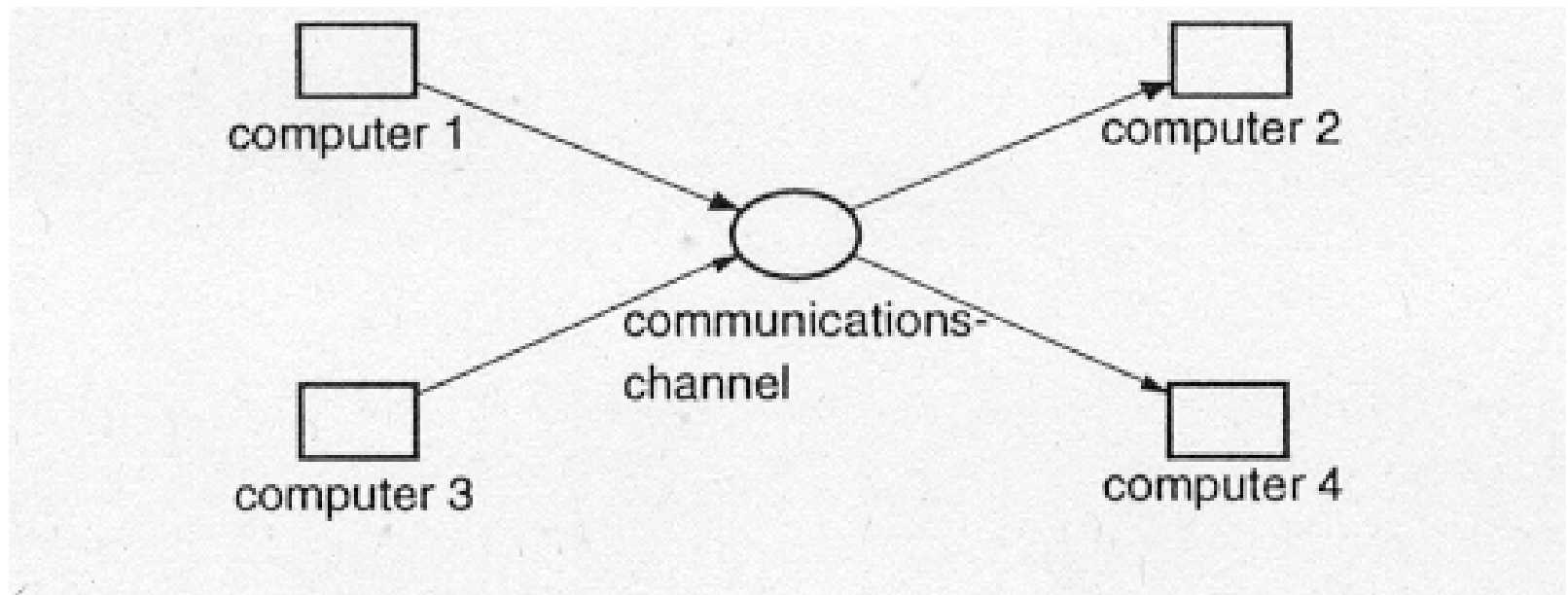
Don't



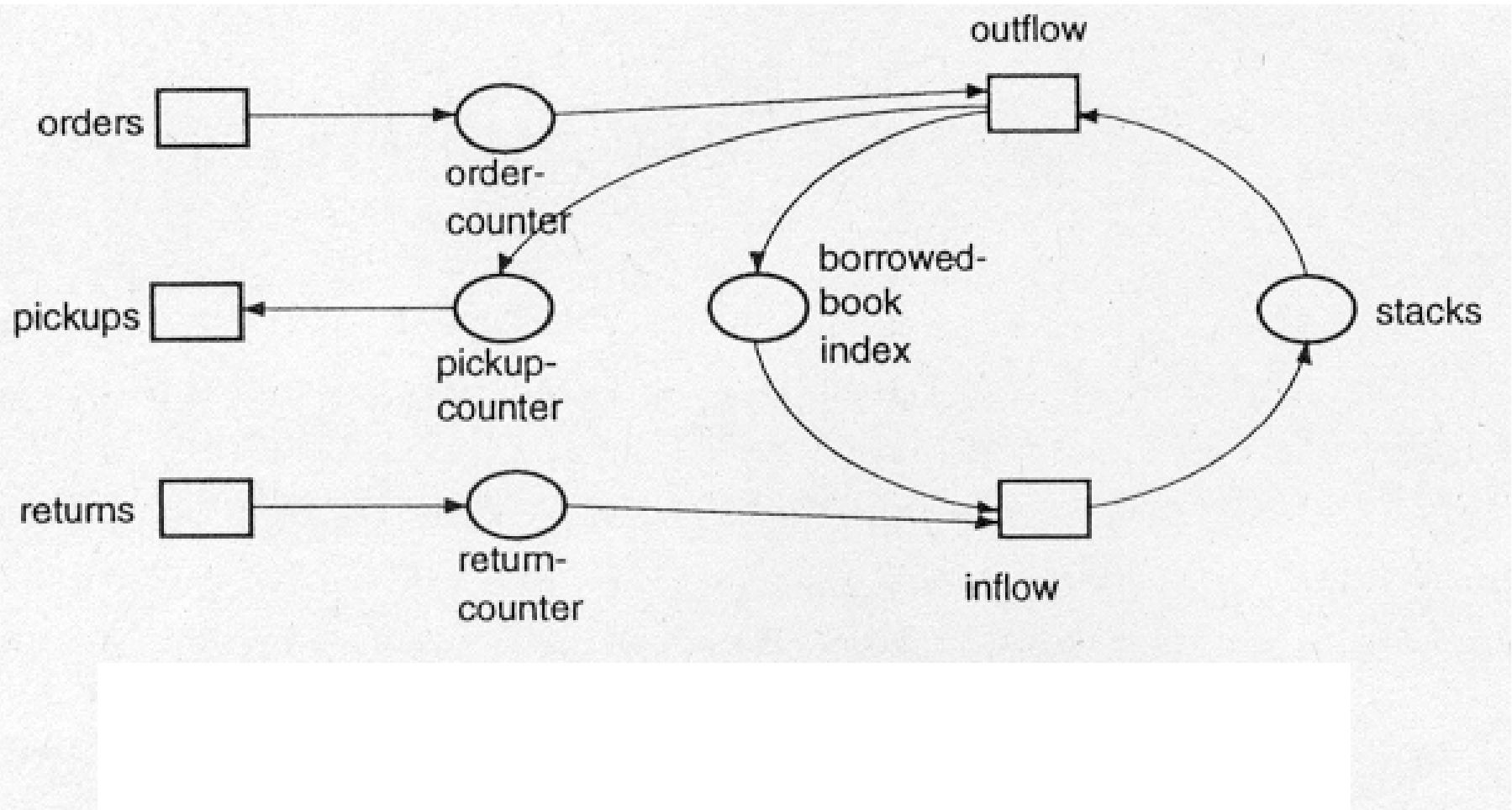
Do

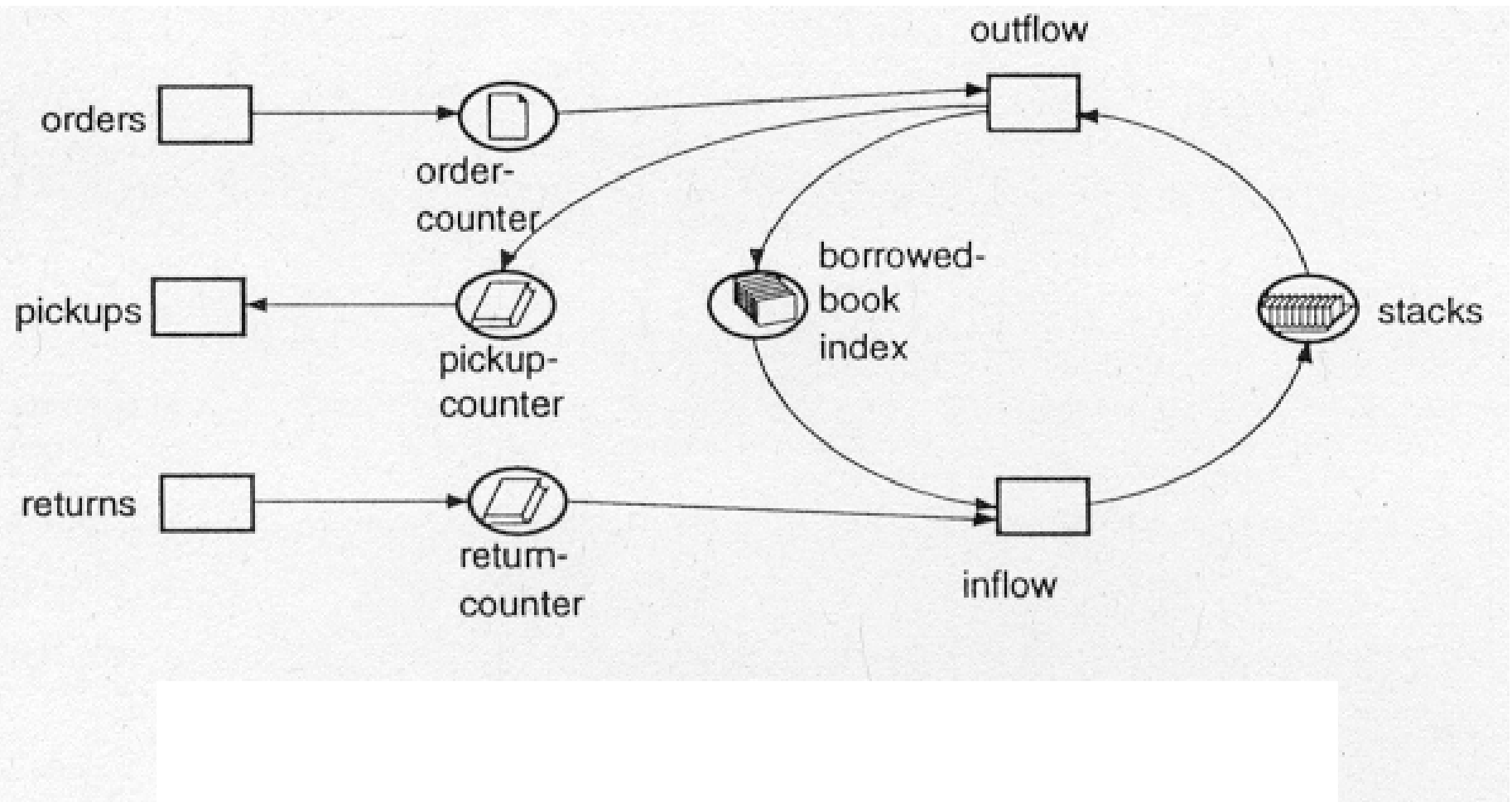


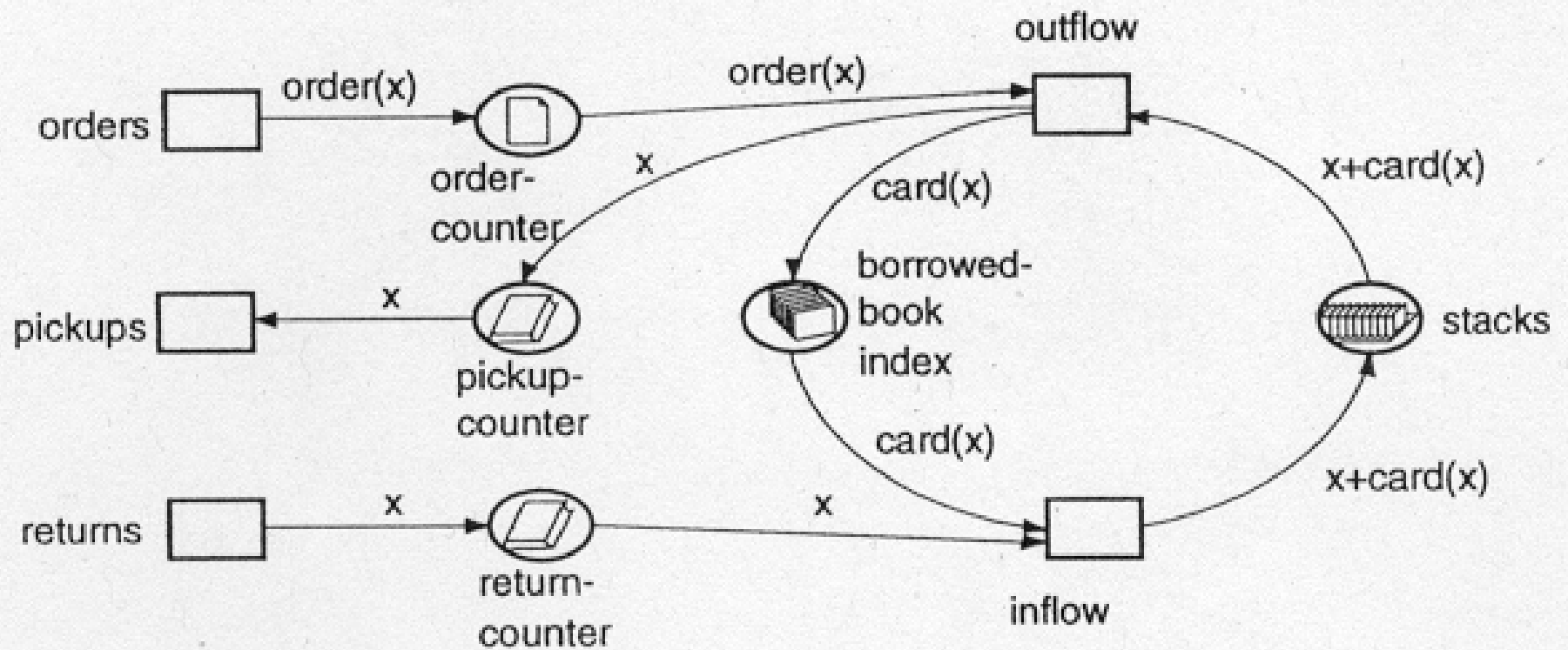




3. Dynamik







Definitionen

Petrinetz:

S – endliche Menge von *Stellen*

T – endliche Menge von *Transitionen*

F – Menge von *Bögen*

W – Bogen*vielfachheiten*

m_0 – Anfangs*markierung*

$$S \cap T = \emptyset$$

$$F \subseteq (S \times T) \cup (T \times S)$$

$$W: F \rightarrow \mathbb{N} \setminus \{0\}$$

$[S, T, F, W, m_0]$

Elemente von $S \cup T$ heißen *Knoten*

Markierung:

Verteilung von Marken auf Stellen

$$m: S \rightarrow \mathbb{N} \cup \{0\}$$

Definitionen

Vorbereich eines Knoten x :

$$\bullet x = \{ y \mid [y, x] \in F \}$$

Nachbereich eines Knoten x :

$$x \bullet = \{ y \mid [x, y] \in F \}$$

Transition t ist **aktiviert** (hat Konzession) in Markierung m :

$$\text{Für alle } s \in \bullet x : W([s, t]) \leq m(s)$$

Transition t **schaltet/feuert** in m und führt zu m' :

t ist aktiviert in m und für alle s :

$$m'(s) = m(s) - W([s, t]) + W([t, s])$$

$$\begin{array}{ccc} \text{(Annahme: } W([x, y]) = 0 \text{ für } [x, y] \notin F) & \xrightarrow{t} & \\ m [t > m' & m & m' \end{array}$$

Definitionen

Erreichbarkeit

... mit Transitionssequenz w

- $m \xrightarrow{\varepsilon} m$
- Wenn $m \xrightarrow{w} m_1$ und $m_1 \xrightarrow{t} m'$ so $m \xrightarrow{wt} m'$

... beliebig

$m \xrightarrow{*} m'$ falls es ein w gibt mit $m \xrightarrow{w} m'$

Menge der von m erreichbaren Markierungen

$$R_N(m) = \{m' \mid m \xrightarrow{*} m'\}$$

Erreichbarkeitsgraph des Netzes $[S, T, F, W, m_0]$:

Gerichteter, beschrifteter Graph $[V, E]$

$$V = R_N(m_0)$$

$$[m, t, m'] \in E \text{ gdw. } m \xrightarrow{t} m'$$

Beispiel

World of Petri nets

<http://www.daimi.au.dk/PetriNets>

Dort:

... /Introductions/aalst/

Beispiele für Netze, Schalten, Erreichbarkeitsgraphen

Erste Erkenntnis: Monotonie des Schaltens

Satz:

$m[w > m']$ und $m_1 \geq m \rightarrow$ es gibt ein $m_1' \geq m'$ mit $m_1[w > m_1']$

Beweis: (Induktion über T^*)

A) $m[\varepsilon > m]$ und $m_1[\varepsilon > m_1]$

Vor) $m[w > m']$ und $m_1 \geq m \rightarrow$ es gibt ein $m_1' \geq m'$ mit $m_1[w > m_1']$

Beh) $m[wt > m'']$ und $m_1 \geq m \rightarrow$ es gibt ein $m_1'' \geq m''$ mit $m_1[wt > m_1'']$

Beweis:

Sei $m[wt > m'']$ und $m_1 \geq m$. D.h., es gibt ein m' mit $m[w > m']$ und $m'[t > m'']$, also $m'' = m' - W(.,t) + W(t,.)$

Nach Ivor. ex. $m_1' \geq m'$ mit $m_1[w > m_1']$. Außerdem ist $m' \geq W(.,t)$, also erst recht $m_1' \geq W(.,t)$. Demnach ist t bei m_1' aktiviert. Schließlich ist mit $m_1'' := m_1' - W(.,t) + W(t,.)$

1. $m_1'' \geq m''$ und
2. $m_1'[t > m_1'']$, also auch $m_1[wt > m_1'']$.

q.e.d.

Eigenschaften

Erreichbarkeit einer Markierung m' von m
 $m \xrightarrow{*} m'$

Transition t **tot** bei m
bei keinem m' in $R_N(m)$ ist t aktiviert

Transition t **lebendig** bei m
von jedem m' in $R_N(m)$ ist ein m'' erreichbar, wo t aktiviert ist

Stelle s **k-beschränkt** bei m
Bei allen m' in $R_N(m)$ ist $m'(s) \leq k$ **sicher** = 1-beschränkt

Markierung m ist **Home-state**
 m ist von jedem m' aus $R_N(m_0)$ erreichbar

Eigenschaften

N verklemmungsfrei

in jedem m aus $R_N(m_0)$ ist mindestens eine Transition aktiviert

N lebendig

alle Transitionen in N lebendig

N reversibel

von jedem m aus $R_N(m_0)$ ist m_0 erreichbar

N k-beschränkt

jede Stelle in N ist k-beschränkt

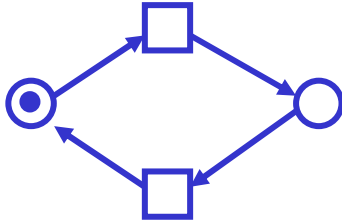
N beschränkt

es gibt ein k so, dass N k-beschränkt

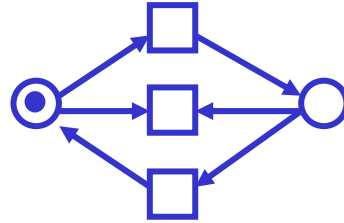
Beispiele

L, V

B, R



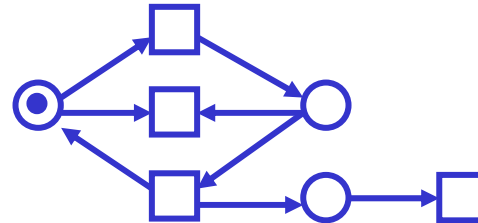
$\neg L, V$



$\neg L, \neg V$



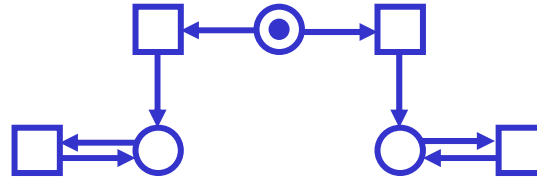
$\neg B, R$



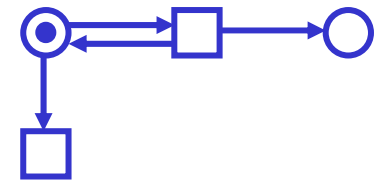
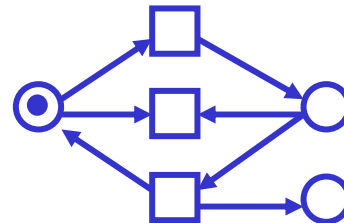
Geht nicht

$B, \neg R$

Geht, nichttrivial



$\neg B, \neg R$



Einige Beziehungen

N lebendig, $|T| \geq 1 \rightarrow$ N verklemmungsfrei

N reversibel \rightarrow jede Transition ist bei m_0 entweder tot oder lebendig

Beweis: Sei t nicht tot bei m_0 . Z.z. t ist lebendig.

t nicht tot bei m_0 heißt: es gibt m^* : $m_0 \xrightarrow{*} m^*$ und t aktiviert bei m^* .

t lebendig heißt: für jedes m gibt es ein m' : $m \xrightarrow{*} m'$ und t aktiviert bei m' .

Sei m erreichbar. Wegen Reversibilität: $m \xrightarrow{*} m_0$, und nach

Voraussetzung $m_0 \xrightarrow{*} m^*$, also $m \xrightarrow{*} m^*$,

also gilt Behauptung mit $m' = m^*$.

Weitere Beziehungen

N reversibel gdw. m_0 ist Home State

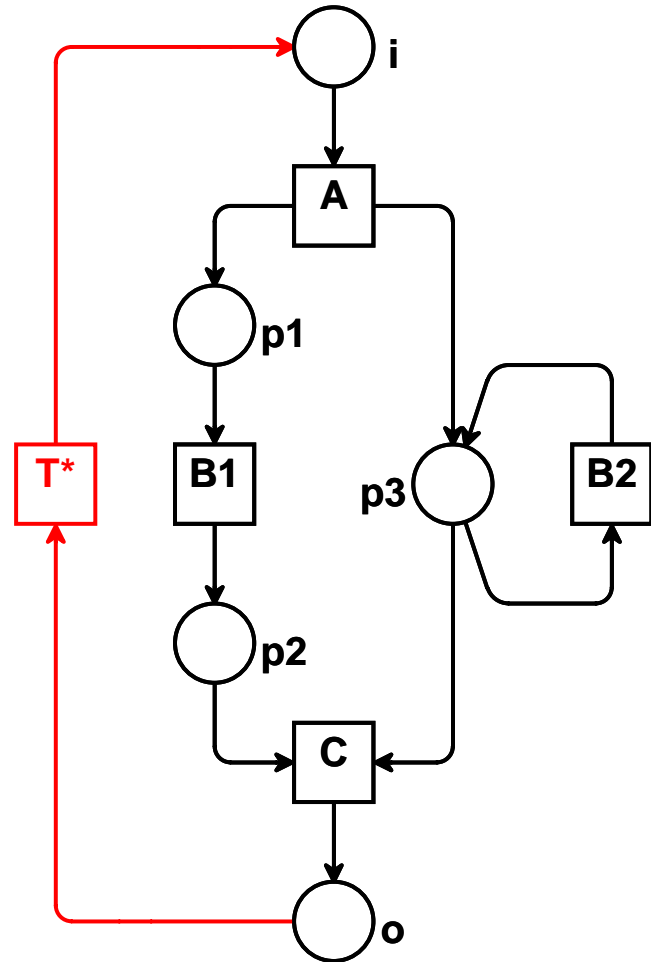
N reversibel gdw. alle erreichbaren Zustände sind Home States

N beschränkt gdw. $R_N(m_0)$ endlich.

Workflow-Netze

Ein Workflow-Netz

- ist ein Petrinetz $N = (S, T, F)$ (d.h. ein S/T-Netz),
- hat genau einen Anfang mit leerem Vorbereich ($i \in P, \bullet i = \emptyset$),
- hat genau ein Ende mit leerem Nachbereich ($o \in P, o \bullet = \emptyset$)
- und, ergänzt um Transition t^* , stark zusammenhängend.



Wichtige Eigenschaft für Workflownetze

- Soundness
- Ideen:
 - Jeder angefangene Vorgang kann beendet werden
 - Bei Ende eines Vorgangs alles aufgeräumt
 - Jede Aktivität möglich

Soundness formal:

- Für alle m , erreichbar von $[i]$ gilt: Von m ist $[o]$ erreichbar
- Für alle m mit $m(o) > 0$: $m(p) = 0$ für alle $p \neq o$
- Für alle t ex. m , von i erreichbar, bei dem t aktiviert ist
- Verbindung zu klassischen PN-Eigenschaften:
 - N ist sound gdw. zugrundeliegendes Petrinetz ist lebendig und beschränkt.

Eigenschaften und Erreichbarkeitsgraph

Dazu muss man wissen:

Stark zusammenhängende Komponenten:

Sei $[V, E]$ gerichteter Graph.

Knoten v, v' sind stark zusammenhängend ($v \sim v'$), falls $v \rightarrow^* v'$ und $v' \rightarrow^* v$.

\sim ist Äquivalenzrelation. Klassen heißen **SZK**.

Eine SZK heißt terminal (TSZK), falls von ihr aus keine andere SZK erreichbar ist.

Eigenschaften und Erreichbarkeitsgraph

Sei N Petrinetz und $[R_N(m_0), E]$ sein Erreichbarkeitsgraph.

- N beschränkt gdw. $R_N(m_0)$ endlich
- m erreichbar gdw. $m \in R_N(m_0)$
- N verklemmungsfrei gdw. jeder Knoten hat Nachfolger
- N reversibel gdw. $R_N(m_0)$ ist (T)SZK
- t lebendig gdw. t kommt in jeder terminalen SZK vor.
- N hat Home States gdw. N hat genau eine TSZK.

Beschränktheit

Wenn $m_0 [^*> m [^*> m'$ und $m' > m$, dann ist N unbeschränkt.

Umkehrung gilt auch:

Wenn N unbeschränkt ist, dann gibt es m und m' mit $m_0 [^*> m [^*> m'$ und $m' > m$.

$m' > m$ heißt: für alle s $m'(s) \geq m(s)$ und für mindestens ein s
 $m'(s) > m(s)$.

Beweis

Haben Graph, der

- von m_0 zusammenhängend ist,
- unendlich viele Knoten hat (N unbeschränkt)
- aber jeder Knoten endlich viele Nachfolger hat (T endlich)

In einem solchen Graph gibt es einen unendlichen, wiederholungsfreien Weg. („Königs Graphenlemma“)

Beweisidee: wenigstens von einem der Nachfolger müssen unendlich viele andere Knoten erreichbar sein, diesen Nachfolger wählen wir und gehen weiter.

Beweis

Haben also unendliche Markierungsfolge mit lauter verschiedenen Markierungen.

Behauptung: in einer solchen Folge gibt es eine schwach monoton wachsende Teilfolge

Beweis:

In jeder unendlichen Zahlenfolge gibt es eine schwach monoton wachsende Teilfolge:

Entweder ein Wert kommt unendlich oft vor \rightarrow nehmen den

Oder alle Werte kommen nur endlich oft vor \rightarrow nach endlich vielen Auswahlen stehen immer noch unendlich viele weitere Werte (also auch größere als bisher verbraucht) zur Verfügung

Beweis

Sei $S = \{s_1 \dots s_n\}$

Wählen nun aus der unendlichen Markierungsfolge
erst eine unendliche Teilfolge, die auf s_1 schwach
monoton

wächst, dann daraus eine, die auf s_2 wächst, usw. bis
 s_n .

→ unendliche Teilfolge (m_1, m_2, \dots) , die auf allen Stellen
schwach monoton wächst.

→ $m_0 [^*> m_1 [^*> m_2$, $m_1 \leq m_2$ und $m_1 \neq m_2$

→ q.e.d.

ω -Markierungen

$\mu : S \rightarrow \mathbb{N} \cup \{\omega\}$ heißt ω -Markierung

Fassen ω -Markierungen auf als Grenzwert einer Folge von (erreichbaren) Markierungen

$(m_i)_{i \in \mathbb{N}}$ konvergiert gegen μ , falls für jedes k ein n existiert so dass für alle $j > n$ und alle $s \in S$ gilt:

$m_j(s) = \mu(s)$, falls $\mu(s) \neq \omega$

$m_j(s) > k$, falls $\mu(s) = \omega$

→ Auf den „endlichen“ Stellen ist Folge ab einer bestimmten Stelle konstant

Überdeckbarkeitsgraph

Ziel: endlicher Graph (auch für unbeschränkte Netze),
mit

- Knoten sind ω -Markierungen
- Zu jeder konvergenten Folge von erreichbaren Markierungen ist der Grenzwert im Graph überdeckt

Def: μ_1 überdeckt μ_2 , falls $\mu_1 \geq \mu_2$

Dabei: $\omega \geq k$

$$\omega + k = \omega$$

$$\omega - k = \omega, \text{ falls } k \neq \omega$$

$$\omega - \omega \text{ undefiniert}$$

Sei $\Omega_\mu = \{s \mid \mu(s) = \omega\}$

Konstruktion Überdeckbarkeitsgraph

VAR C: SET OF ω -Markierung

Cover(\emptyset, m_0);

Cover(M, μ):

 C = C \cup { μ };

 FOR ALL t, t aktiviert in μ DO

$\mu' = \mu - W(.,t) + W(t,.)$

 IF exists μ^* in M: $\Omega\mu^* = \Omega\mu'$, $\mu^* < \mu'$ THEN

 FOR ALL s, $\mu^*(s) < \mu'(s)$ DO $\mu'(s) = \omega$ END

 END

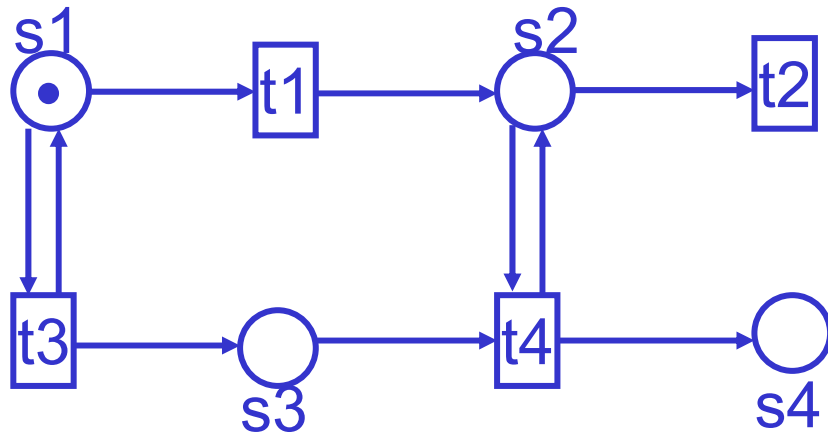
 IF $\mu' \notin C$ THEN

 Cover(M \cup { μ }, μ');

 END

END

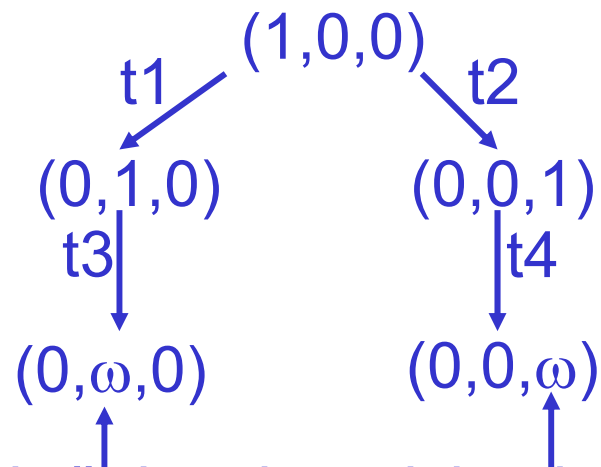
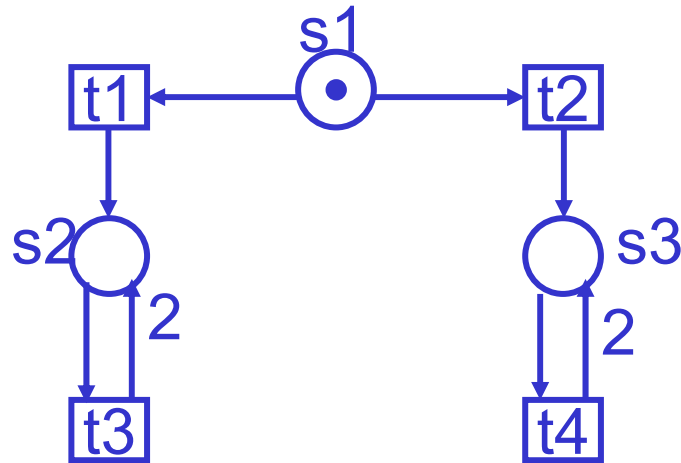
Beispiel



$$\begin{array}{l}
 (1, 0, 0, 0) \xrightarrow{t_1} (0, 1, 0, 0) \xrightarrow{t_2} (0, 0, 0, 0) \\
 \quad \quad \quad \downarrow t_3 \\
 t_3 \curvearrowright (1, 0, \omega, 0) \xrightarrow{t_1} (0, 1, \omega, 0) \xrightarrow{t_2} (0, 0, \omega, 0) \\
 \quad \quad \quad \quad \quad \quad \downarrow t_4 \\
 \quad \quad \quad t_4 \curvearrowright (0, 1, \omega, \omega) \xrightarrow{t_2} (0, 0, \omega, \omega) \\
 \quad \quad \quad \quad \quad \quad \uparrow \quad \uparrow
 \end{array}$$

Simultan unbeschränkt: $t_3^{2k} t_1 t_4^k$

Noch ein Beispiel



Beide unbeschränkt, aber nicht simultan

2. Aussage

Satz: Zu jeder erreichbaren Markierung m gibt es im Überdeckbarkeitsgraphen eine Markierung μ mit $\mu \geq m$.

Beweis (Induktion über T^*)

A) $m_0 [\varepsilon > m$, also $m = m_0$, m_0 ist enthalten im Überdeckbarkeitsgraph.

Vor: Zu m mit $m_0[w > m$ gibt es ein μ mit $\mu \geq m$.

Beh: Zu m' mit $m_0[w > m[t > m'$ gibt es ein μ' mit $\mu' \geq m'$.

Beweis: t aktiviert in m , also erst recht in μ .

$\mu - W(.,t) + W(t,.) (= \mu^*) \geq m'$, weil $m[t > m'$ und $\mu \geq m$.

t -Nachfolger von μ ist gleich μ^*

oder entsteht daraus durch Einfügen neuer ω

→ dieser Knoten überdeckt m' . q.e.d.

Folgerung: Jede schaltbare Sequenz kann man auch im Ü-Graph ablaufen

3. Aussage

Satz: Für jedes Netz ist der Überdeckbarkeitsgraph endlich.

Beweis: können sonst wieder monotone Teilsequenz auszeichnen

→ überdeckende Markierungen

→ neue ω

aber: neue ω können, da S endlich, nur endlich oft dazukommen und verschwinden nie.

4. Aussage

Satz: Zu jeder konvergenten Folge (einer, die einen Grenzwert besitzt) erreichbarer Markierungen gibt es einen Knoten im Überdeckbarkeitsgraph, der den Grenzwert überdeckt.

Beweis

Jedes Folgenglied ist überdeckt, insgesamt können aber nur endlich viele Knoten an den Überdeckungen beteiligt sein

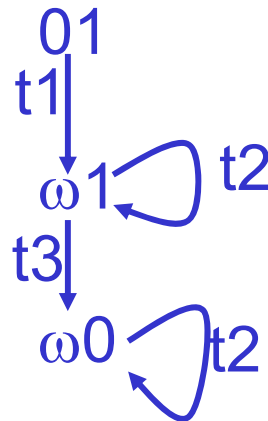
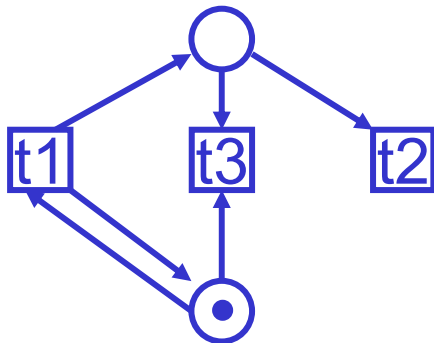
- Ein Knoten überdeckt unendlich viele Folgenglieder
- Dieser Knoten überdeckt wg. Konvergenz ab irgendeinem Folgenglied alle weiteren.
- Dieser Knoten überdeckt auch den Grenzwert.

Überdeckbarkeitsgraph und Eigenschaften

- Wenn m erreichbar, so gibt es ein μ im \ddot{U} -graph mit $\mu \geq m$
- t genau dann nicht tot, wenn t im \ddot{U} -graph aktiviert.
- s genau dann beschränkt, wenn kein Knoten μ mit $\mu(s) = \omega$ im \ddot{U} -Graph existiert
- Zu jeder beschränkten Stelle s ist $\text{MAX}\{\mu(s) \mid \mu \text{ im } \ddot{U}\text{-graph}\}$ das kleinste k , so dass s k -beschränkt ist.
- $Q \subseteq S$ genau dann simultan unbeschränkt, falls ein Knoten im \ddot{U} -graph existiert, bei dem alle Knoten aus Q den Wert ω haben
- Wenn \ddot{U} -Graph eine TSZK enthält, in der eine Transition t nirgends aktiviert ist, so ist t nicht lebendig
- Ist N beschränkt, so stimmen \ddot{U} -Graph und Erreichbarkeitsgraph überein.

Umkehrungen gelten nicht

- Obwohl im Ü-graph alle Knoten Nachfolger haben, ist N nicht verklemmungsfrei
- Obwohl t2 in jeder TSZK, ist t2 nicht lebendig



Übungsaufgabe: Gegenbeispiel für Umkehrung der Reversibilitätsaussage, Home states

Die Stubborn-Set-Methode

(eine Variante von Partial Order Reduction)

Ziel: Erreichbarkeitsgraph unter Bewahrung
vorgegebener Eigenschaften verkleinern

Mittel: Lokalität der Transitionen

Beziehungen zwischen Transitionen

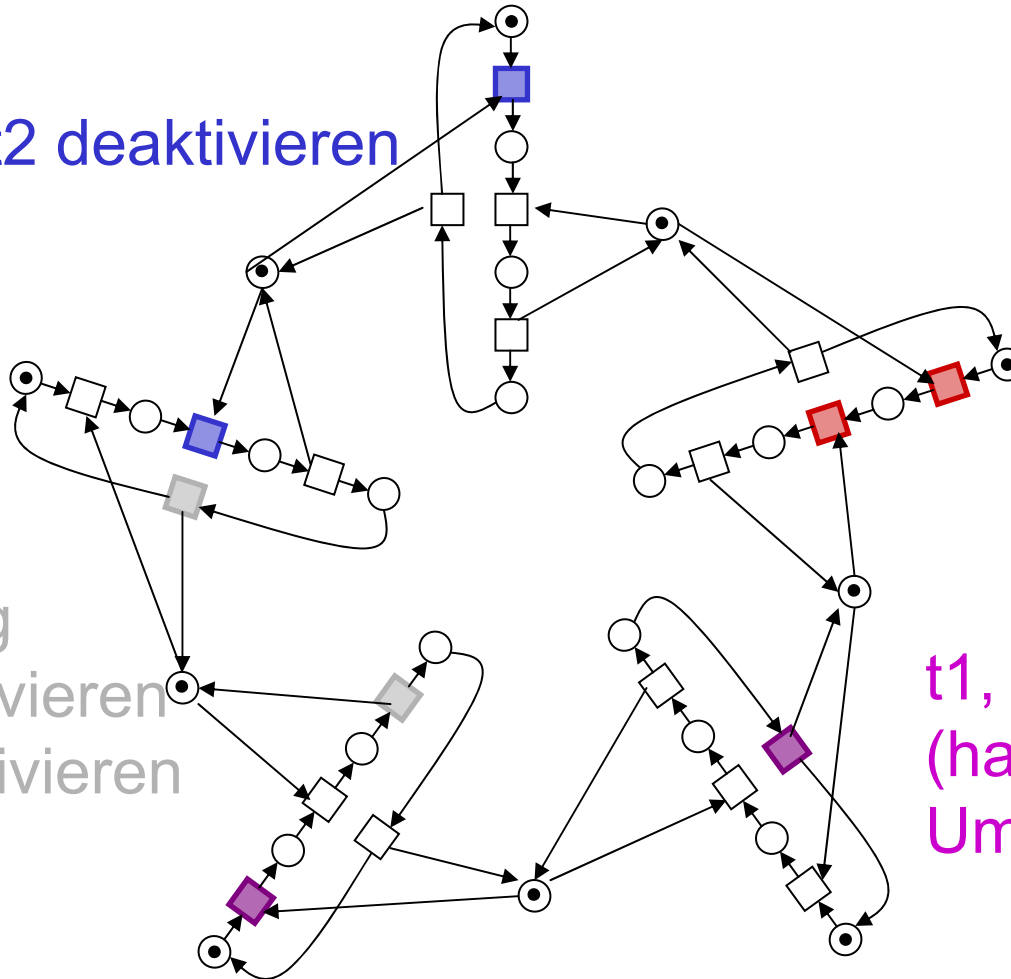
Konflikt

t1 kann t2 deaktivieren

t1 kann t2
aktivieren

t1,t2 sind
nebenläufig
(weder aktivieren
noch deaktivieren
sie sich)

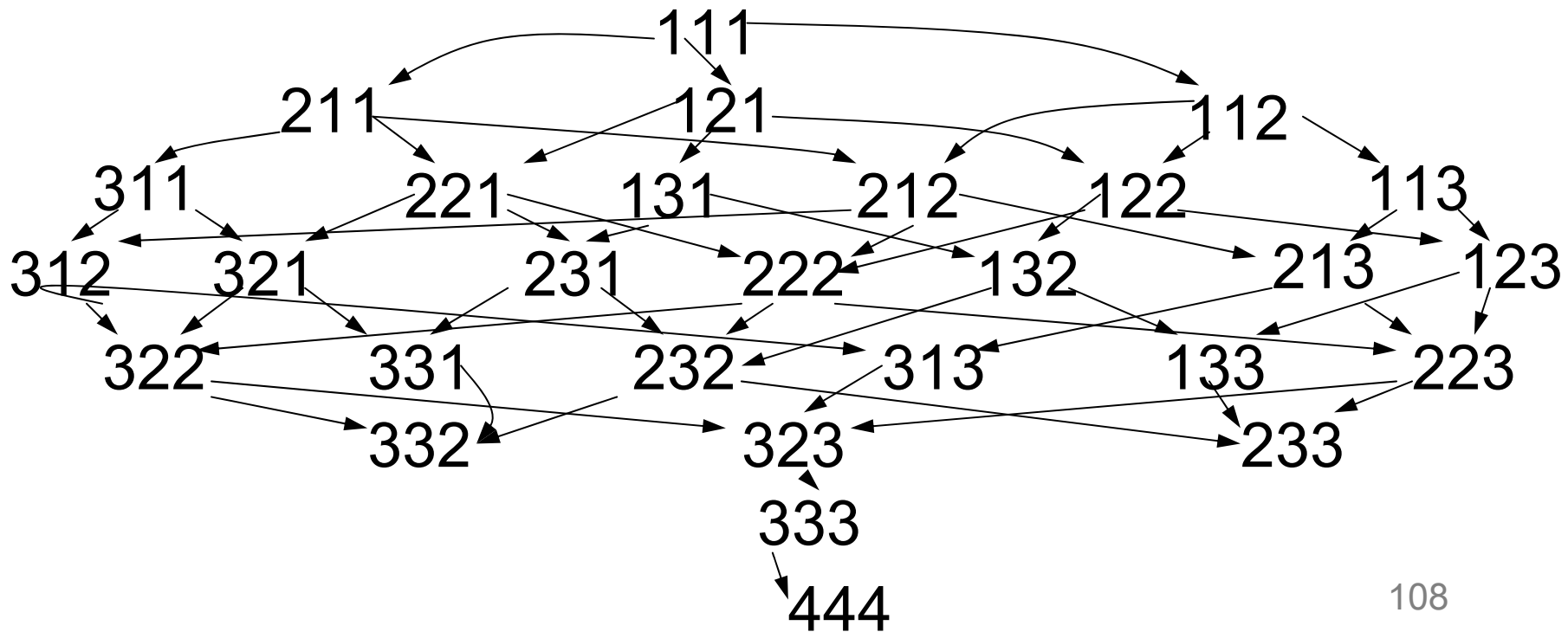
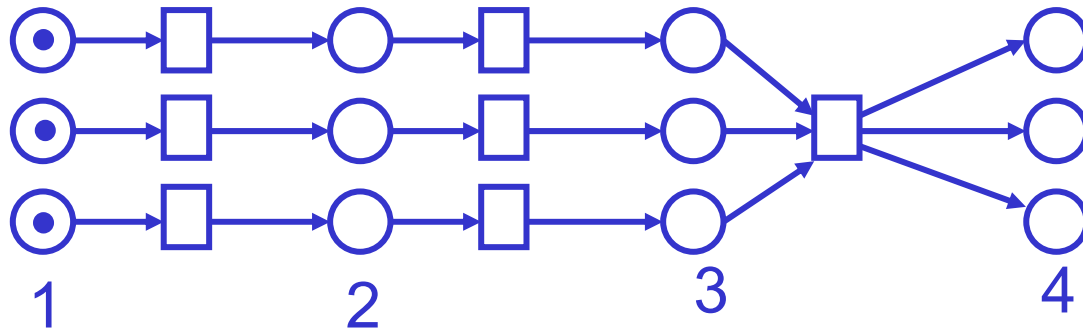
t1, t2 sind unabhängig
(haben disjunkte
Umgebungen)



Nebenläufigkeit

- unabhängige Transitionen sind immer nebenläufig
- Hintereinanderausführung nebenläufiger Transitionen führt zum gleichen Zustand $m - W(.,t1) - W(.,t2) + W(t1,.) + W(t2,.)$, unabhängig von Reihenfolge
- aber: Zwischenzustände sind meist verschieden:
 $m - W(.,t1) + W(t1,.)$ $m - W(.,t2) + W(t2,.)$

Beispiel



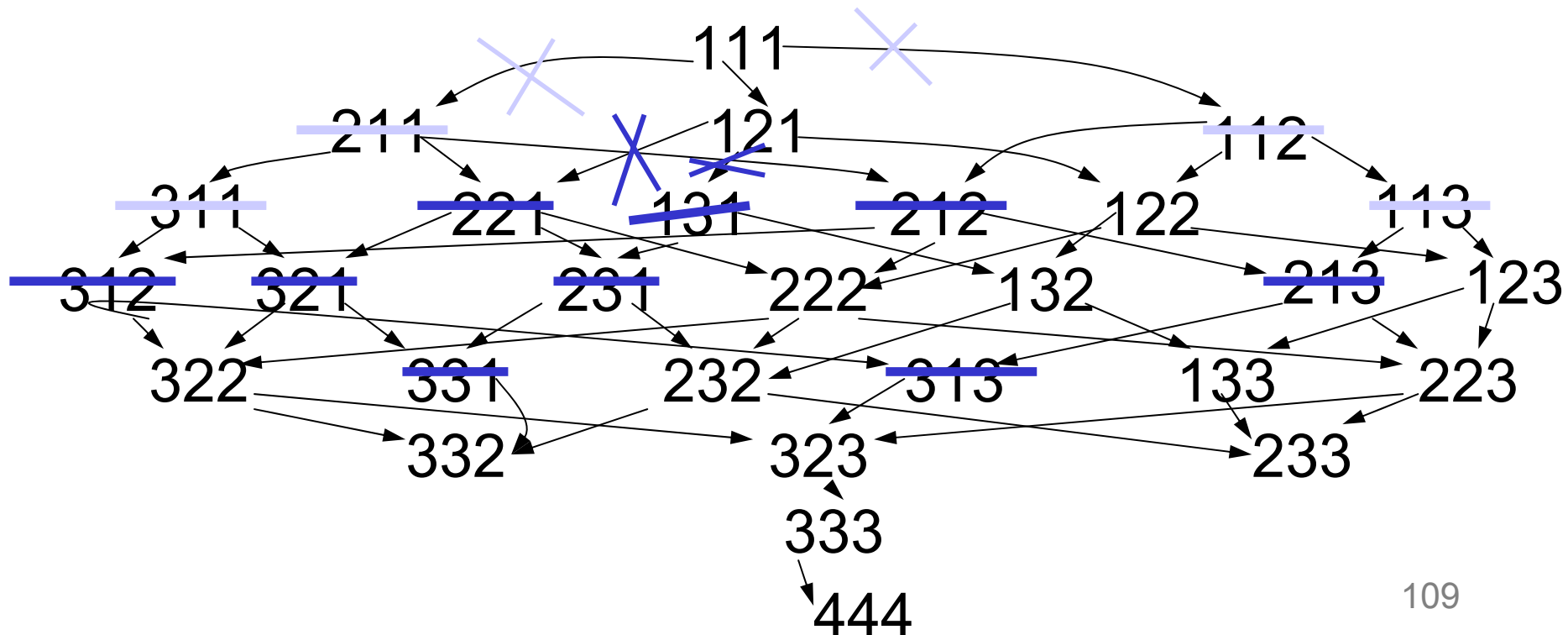
Idee von Stubborn sets

Sei m Markierung.

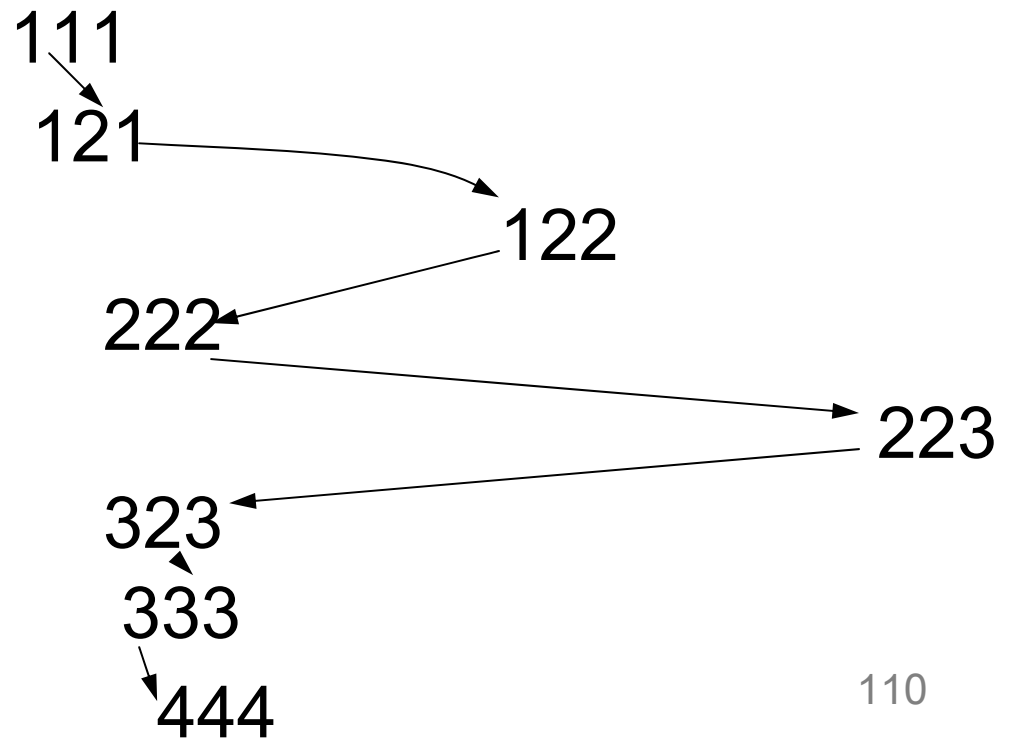
$\text{stubborn}(m)$ ist eine (wenn möglich) nichtleere Teilmenge von Transitionen

→ Reduziertes Transitionssystem:

verfolge bei m nur die aktivierten Aktionen in $\text{stubborn}(m)$



Reduziertes Transitionssystem



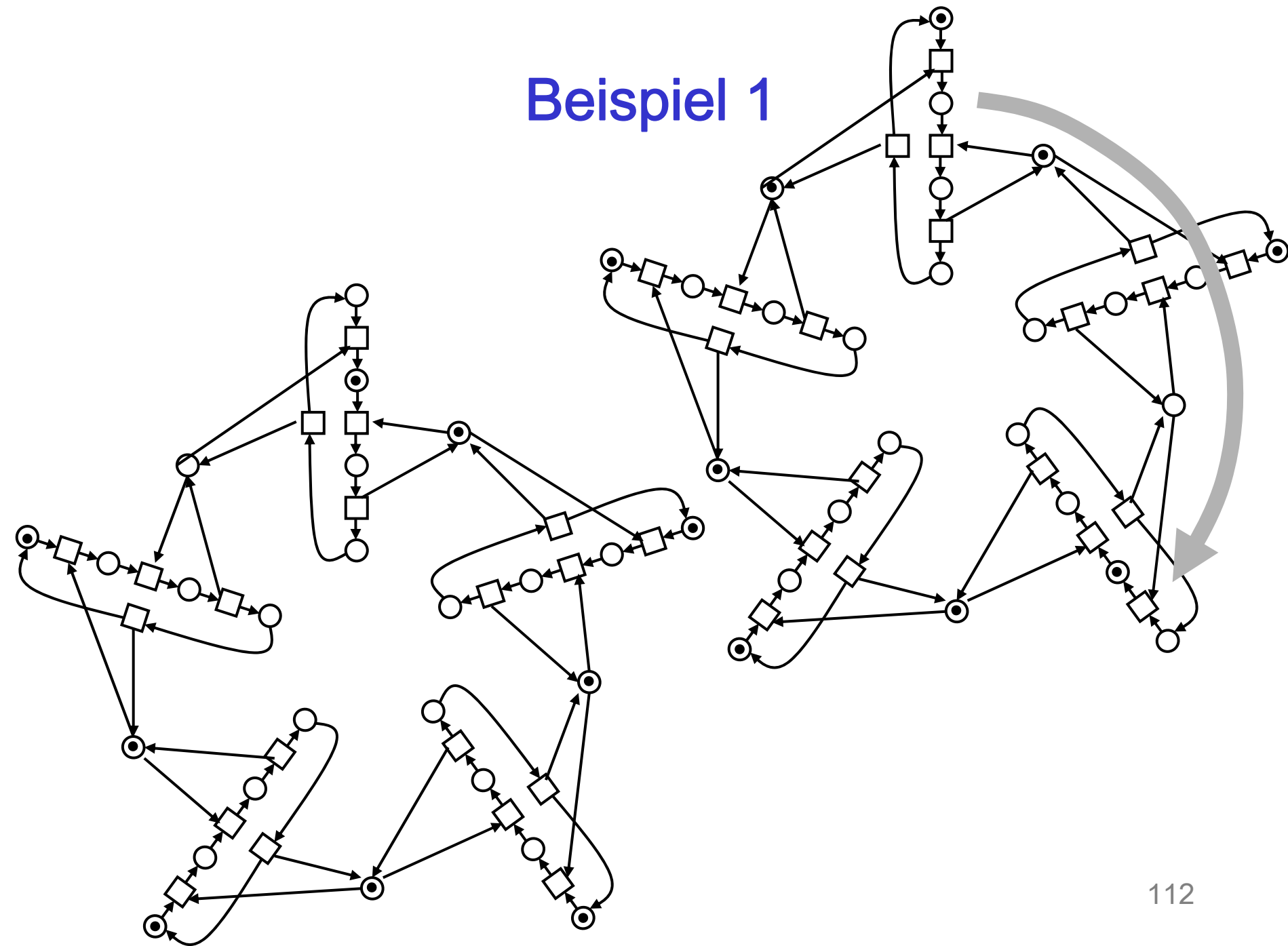
Reduktion durch Symmetrie

Grundgedanke: symmetrisch strukturierte Systeme haben symmetrisches Verhalten

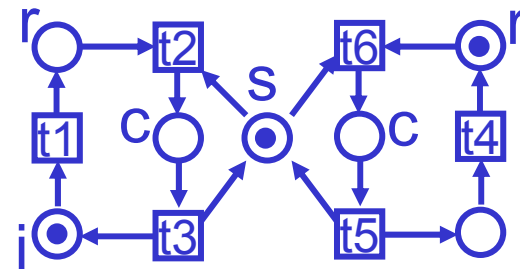
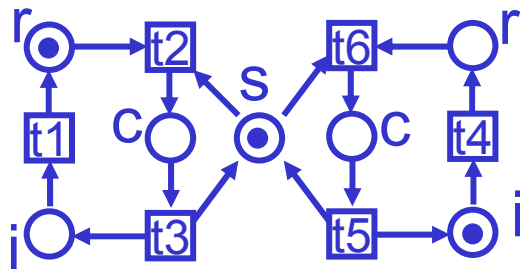
Wenn Verhalten bei m bekannt und m' symmetrisch zu m , braucht Verhalten bei m' nicht mehr untersucht werden

technisch: Äquivalenzrelation; Quotienten-Transitionssystem

Beispiel 1



Beispiel 2



Netzreduktion

- Lokale, eigenschaftserhaltende Ersetzungsregeln
- Eigenschaften hier: Lebendigkeit, Beschränktheit
- Ziel: kleineres Netz, kleinerer Erreichbarkeitsgraph

Eigenschaftserhaltung:

N erfüllt Eigenschaft gdw. N' erfüllt Eigenschaft

oder die Eigenschaft kann für N sicher entschieden werden

Verabredungen:

- Alle Bogenvielfachheiten 1
- Altes Netz N , neues Netz N'
- gestrichenes rot, eingefügtes grün, bleibendes blau

Erste Regel: Transitionen ohne Vorplatz

Voraussetzung: Es gibt ein t mit $\bullet t = \emptyset$

Anwendung: Streiche t und alle Stellen in $t\bullet$

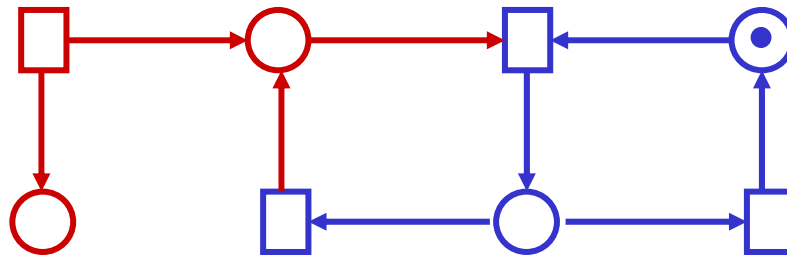
Aussagen:

- t lebendig

-Alle Stellen in $t\bullet$ unbeschränkt

-Alle anderen Stellen beschränkt in N' gdw. beschränkt in N

-Alle anderen Transitionen in N' lebendig gdw. in N



Beweis

Satz 1: Wenn $m [w > m' \text{ in } N$, so $m_{|S'} [w-t > m'_{|S'} \text{ in } N'$
($w-t = w$, wo alle t gestrichen sind)

Beweis: Die Token, die t in N produziert, werden in N' nicht mehr benötigt.

Satz 2: Wenn $m [w > m' \text{ in } N'$, so $m^* [t^{\text{length}(w)} w > m^{*'} \text{ in } N$,
wobei $m^*_{|S} = m$ und $m^{*'}_{|S'} = m'$

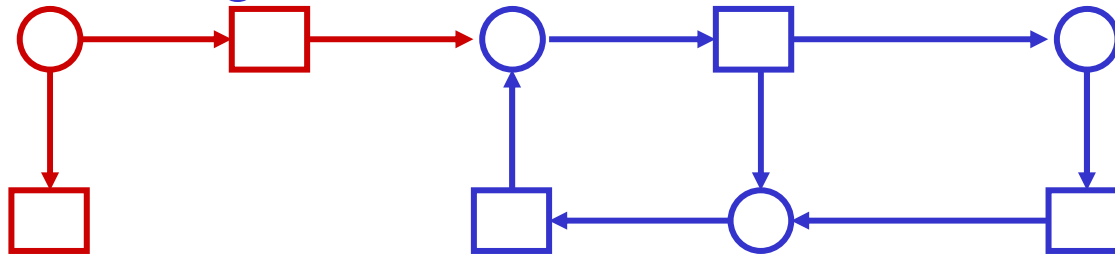
Beweis: $t^{\text{length}(w)}$ produziert ausreichend Token für die nachfolgenden Transitionen

Folgerung: Bewahrung von Lebendigkeit und Beschränktheit

Zweite Regel: Stellen ohne Vortransition

Voraussetzung: Es gibt ein s mit $m(s) = 0$ und $\bullet s = \emptyset$

Anwendung: Streiche s und $s \bullet$



Aussagen

-s beschränkt

-Alle Nachtransitionen nicht lebendig (sogar tot)

-Alle anderen Stellen beschränkt in N gdw. in N'

-Alle anderen Transitionen lebendig in N gdw. in N'

Beweis

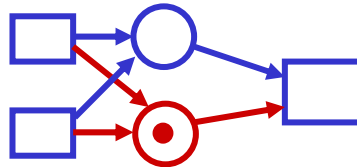
Da die gestrichenen Transitionen tot sind, sind die Erreichbarkeitsgraphen von N und N' isomorph

Bem: Bedingung $m(s) = 0$ ist wichtig!

Dritte Regel: Parallele Knoten

Voraussetzung: Es gibt Knoten x, y mit $\bullet x = \bullet y$ und $x \bullet = y \bullet$
(„ x, y parallel“)

Anwendung: Falls x, y Stellen mit $m(x) < m(y)$ sind, streiche y , sonst streiche x .



Aussagen

- Transition x, y lebendig in N gdw. y lebendig in N'
- Stelle x, y beschränkt in N gdw. x (bzw. y) beschränkt in N'
- Alle anderen Knoten ... wie üblich.

Beweis

Wenn gestrichene Stelle nicht ausreichend Marken hat,
so die verbliebene erst recht nicht.

In jeder Schaltsequenz können Transitionen x, y beliebig
gegenseitig ausgetauscht werden

Vierte Regel: Äquivalente Stellen

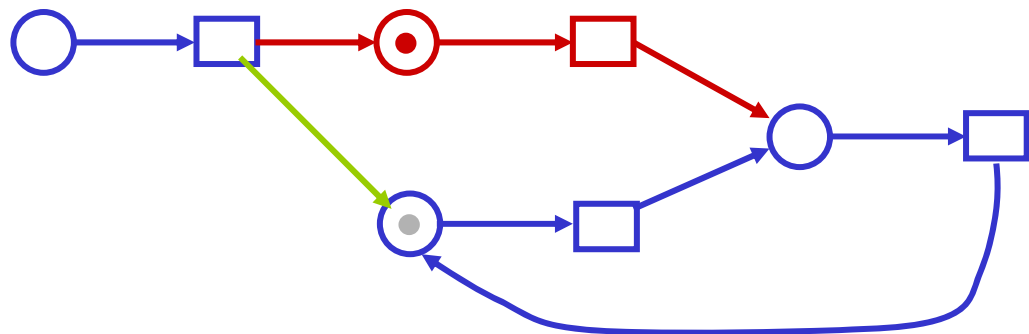
Voraussetzung: Es gibt Stellen s_1, s_2 mit $s_1 \bullet = \{t_1\}$,
 $s_2 \bullet = \{t_2\}$

$$t_1 \bullet = t_2 \bullet, \bullet t_1 \setminus \{s_1\} = \bullet t_2 \setminus \{s_2\}$$

Anwendung: Verbinde alle Vortransitionen von t_2 mit t_1 ,
(bei schon existierenden Bögen: Vielfachheit erhöhen)

Erhöhe $m_0(s_1)$ um $m_0(s_2)$

Streiche s_2, t_2



Aussagen

Wenn s_1 unbeschränkt in N , so s_2 unbeschränkt in N'

Wenn s_1, s_2 beschränkt in N , so s_1 beschränkt in N'

Wenn t_1 oder t_2 nicht lebendig in N , so sind
Vortransitionen von s_1 nicht lebendig in N'

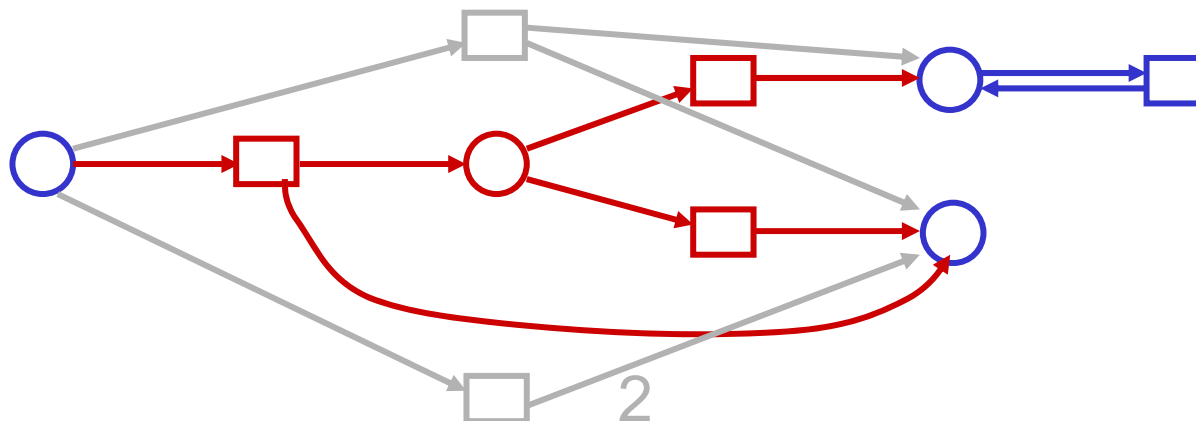
Wenn t_2 lebendig in N , so t_1 lebendig in N'

Fünfte Regel: Zusammenfassung einer Sequenz über einzelne Stelle

Voraussetzung: Es gibt ein s mit $s\bullet \neq \emptyset$, $\bullet s \neq \emptyset$, $s\bullet \cap \bullet s = \emptyset$,
 $s\bullet\bullet \neq \emptyset$, $\bullet(s\bullet) = \{s\}$, $m_0(s) = 0$

Anwendung: $S' = S \setminus \{s\}$ $T' = (T \setminus (s\bullet \cup \bullet s)) \cup (s\bullet \times \bullet s)$

mit $\bullet[t,u] = \bullet t$ und $[t,u]\bullet = t\bullet \cup u\bullet$ (ggf. Vielfachheit erhöhen!)



Aussagen, Beweis

Wenn s unbeschränkt in N , so ist ein Nachplatz einer neuen Transition in N' unbeschränkt

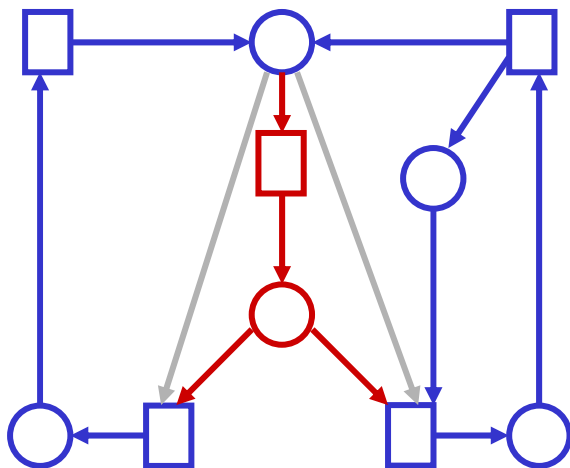
Alte Transitionen lebendig gdw. die zugehörigen neuen

Beweisidee: Sequenz $\dots t \dots u \dots$ kann zu Sequenz $\dots tu \dots$ transformiert werden (Argument wie bei Stubborn sets: u zu keiner Transition in Konflikt!)

Sechste Regel: Zusammenfassen einer Sequenz mit einzelner Starttransition

Voraussetzung: Es gibt eine Stelle s mit $\bullet s = \{t\}$, $t \bullet = \{s\}$,
 $s \bullet \neq \emptyset$, $s \bullet \cap \bullet s = \emptyset$, $m_0(s) = 0$, $(\bullet t) \bullet = \{t\}$

Anwendung: Streiche s, t , Leite alle Bögen zu t zu allen Nachtransitionen von s um (ggf. Vielfachheit erhöhen)



Aussagen: Übung

Beweisidee:

t nicht im Konflikt zu anderen
Transitionen \rightarrow Schalten von t
kann verzögert werden bis zum
Schalten einer Nachtransition
von s

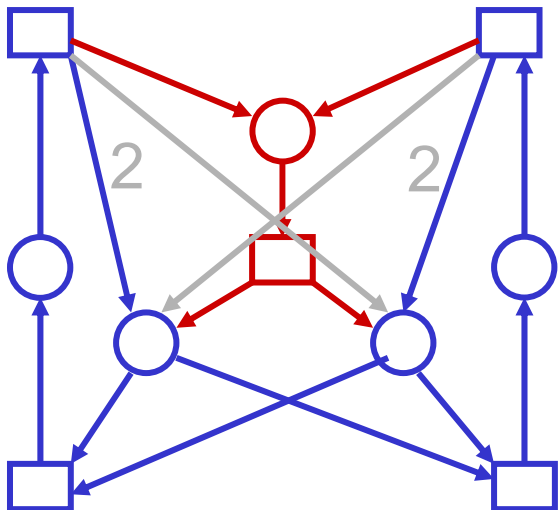
Letzte Regel: Sequenz mit einzelner Endtransition

Voraussetzung: Es gibt ein t mit $\bullet t = \{s\}$, $\bullet s \neq \emptyset$,
 $t \bullet \neq \emptyset$, $s \notin t \bullet$,

$$m_0(s) = 0$$

Anwendung: Addiere Effekt von t zu den Vortransitionen von s , Streiche s, t

Aussagen, Beweis:
Übung



Zusammenfassung Reduktion

Regeln verkleinern Netz und oft auch Zustandsraum

Schönes Ziel wäre: Regeln, mit denen jedes Netz zu einem von endlich vielen Netzen reduziert werden kann.

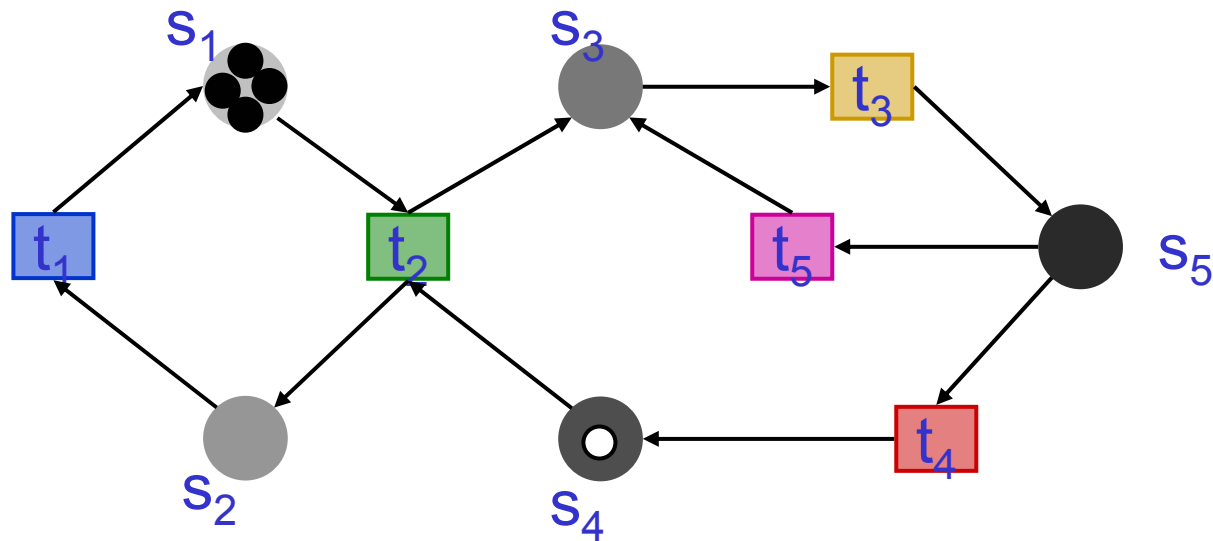
bislang nicht erreicht.

Dipl.-Thema: Implementation und Weiterentwicklung der Regeln, Erarbeitung eines Konzeptes, um Zeugenpfade aus N' in Zeugenpfade für N umzuwandeln.

Strukturanalyse

- Ausnutzung linearer Algebra
- Studium spezieller Netzklassen
- Ausnutzung topologischer Strukturen

Markierungen, Transitionen als Vektor

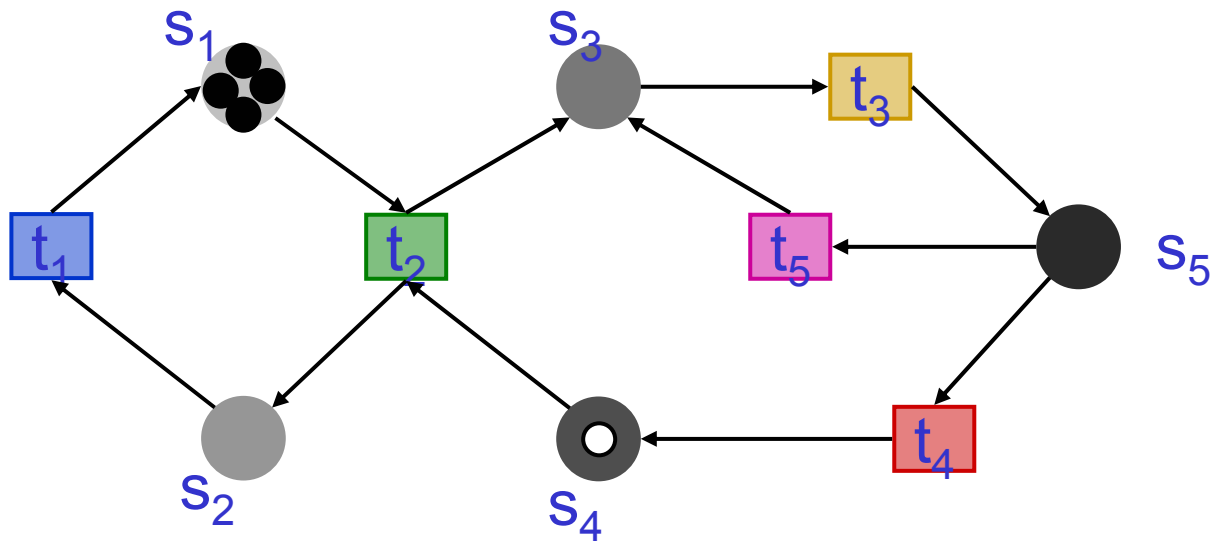


$$\underline{m}_0: (4, 0, 0, 1, 0)$$

$$\underline{t}_2 = (-1, 1, 1, -1, 0)$$

Wenn $m_0 \xrightarrow{\underline{t}_2} m_1$ so $\underline{m}_0 + \underline{t}_2 = \underline{m}_1 = (3, 1, 1, 0, 0)$

Inzidenzmatrix



$$C = \begin{pmatrix} 1 & -1 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 1 \\ 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & -1 & -1 \end{pmatrix}$$

Die Zustandsgleichung

Wenn $m_0 \xrightarrow{t_2 t_3 t_5 t_1 t_3} m$ so

$$\underline{m}_0 + \underline{t}_2 + \underline{t}_3 + \underline{t}_5 + \underline{t}_1 + \underline{t}_3 = \underline{m}$$

$$\underline{m}_0 + (1 \cdot \underline{t}_1) + (1 \cdot \underline{t}_2) + (2 \cdot \underline{t}_3) + (0 \cdot \underline{t}_4) + (1 \cdot \underline{t}_5) = \underline{m}$$

$$\underline{m}_0 + C \cdot (1, 1, 2, 0, 1) = \underline{m}$$

Parikh-Vektor von

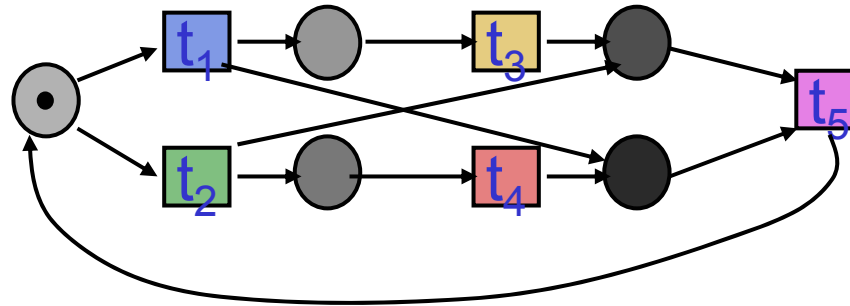
$t_2 t_3 t_5 t_1 t_3$

→ Wenn $m_0 \xrightarrow{w} m$ so $\underline{m}_0 + C \cdot \text{Parikh}(w) = \underline{m}$

→ Markierung m ist höchstens dann erreichbar, wenn

$C \cdot \underline{x} = (\underline{m} - \underline{m}_0)$ eine Lösung für nat. \underline{x} hat

Beispiel



erreichbare Mark.: passende Lösungen der Zst-gleichung

$(1, 0, 0, 0, 0)$	$(0, 0, 0, 0, 0),$	$(1, 0, 1, 0, 1), \dots$
$(0, 1, 0, 0, 1)$	$(1, 0, 0, 0, 0), \dots$	
$(0, 0, 1, 1, 0)$	$(0, 1, 0, 0, 0), \dots$	
$(0, 0, 0, 1, 1)$	$(1, 0, 1, 0, 0),$	$(0, 1, 0, 1, 0), \dots$

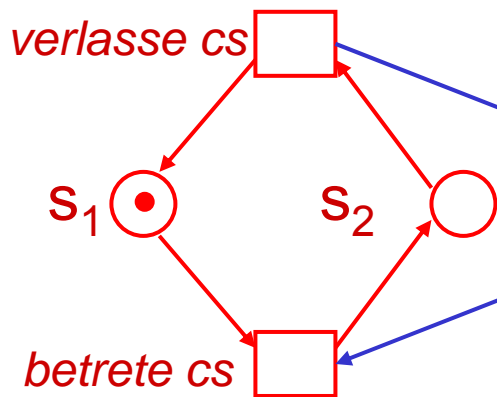
nicht erreichbar,
 $(0, 1, 1, 0, 0)$

hat trotzdem Lösung! ☹️
 $(1, 1, 0, 0, 1), \dots$

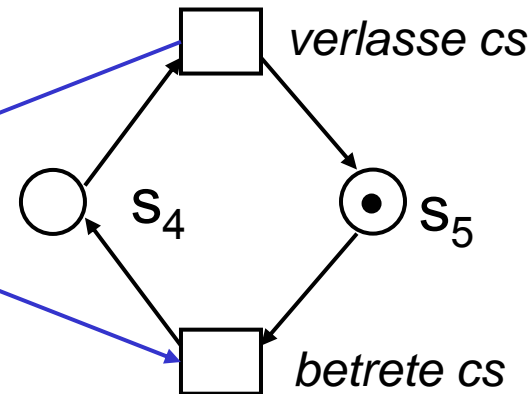
Stellen-Invarianten

Bsp.: gegenseitiger Ausschluss

Prozess 1



Prozess 2



mutex: $m \text{ erreichbar} \rightarrow m(s_2) + m(s_4) \leq 1$

mit Stellen-Invariante

Beweis: 1. $m(s_2) + m(s_3) + m(s_4) = 1$ initial wahr
 2. $m(s_2) + m(s_3) + m(s_4) = 1$ ist stabil
 3. $m(s_2) + m(s_3) + m(s_4) = 1 \rightarrow m(s_2) + m(s_4) \leq 1$

Stelleninvariante i

Def. 1: Für alle t , $\sum_{[s,t] \in F} W([s,t]) \cdot i(s) = \sum_{[t,s] \in F} W([t,s]) \cdot i(s)$

Def. 2: für alle t , $\underline{i} \cdot \underline{t} = 0$

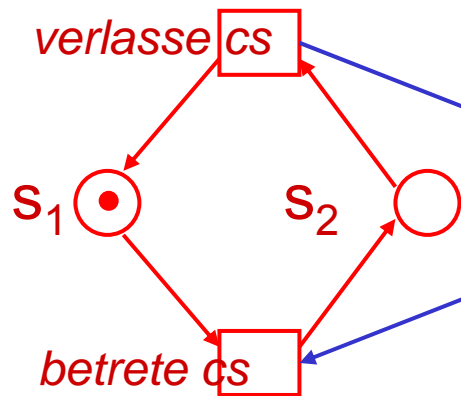
Def. 3: $\underline{i} \cdot C = (0, \dots, 0)$

Wenn m erreichbar von m_0 , so $\underline{i} \cdot \underline{m} = \underline{i} \cdot \underline{m_0}$

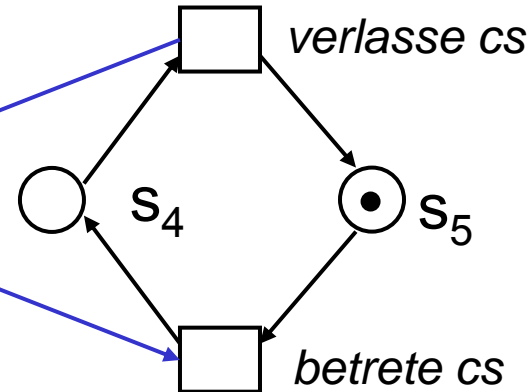
Beweis: $m_0 \xrightarrow{w} m \rightarrow \underline{m_0} + C \cdot \text{Parikh}(w) = \underline{m}$
 $\rightarrow \underline{i} \cdot \underline{m_0} + \underline{i} \cdot C \cdot \text{Parikh}(w) = \underline{i} \cdot \underline{m}$
 $\quad \quad \quad (= 0 \quad \quad \quad)$
 $\rightarrow \underline{i} \cdot m_0 = \underline{i} \cdot m$

Stellen-Invarianten

Prozess 1



Prozess 2



$(0, 1, 1, 1, 0)$ ist Stellen-Invariante

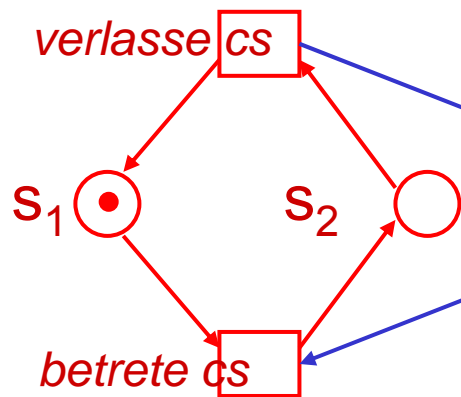
$$\rightarrow \underline{i} \cdot \underline{m}_0 = 1 = \underline{i} \cdot \underline{m} = m(s_2) + m(s_3) + m(s_4)$$

für alle erreichbaren \underline{m}

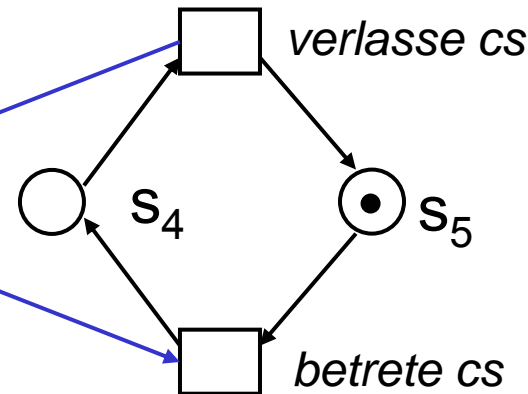
$$\rightarrow m(s_2) + m(s_3) + m(s_4) = 1 \text{ ist stabil.}$$

Weitere Stellen-Invarianten

Prozess 1



Prozess 2



(0 , 1 , 1 , 1 , 0) gegenseitiger Ausschluss

(0 , 1 , 1 , 0 , -1) $m(s_2) + m(s_3) = m(s_5)$
 → Wenn s_2 markiert, so s_5 markiert

(1 , 1 , 0 , 0 , 0) $m(s_1) + m(s_2) = 1$
 → $m(s_1) \leq 1, m(s_2) \leq 1, s_1, s_2$ beschränkt

Lebendigkeit

Für alle Petrinetze,

-lebendig

-ohne isolierte Stellen

→

für alle Stellen-Invarianten \underline{i}

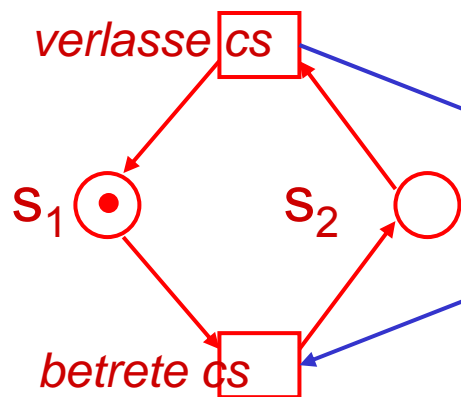
- ohne neg. Komponenten

- mit mind. einer pos. Komp. s

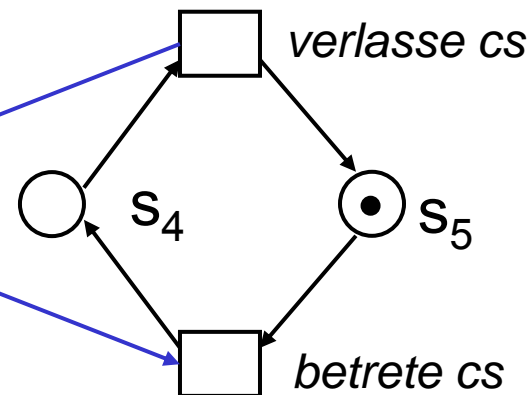
: $\underline{i} \cdot \underline{m}_0 > 0$

(sonst wären Transitionen verbunden mit s tot)

Prozess 1



Prozess 2



(0 , 1 , 1 , 1 , 0)

(1 , 1 , 0 , 0 , 0)

(0 , 0 , 0 , 1 , 1)

Beschränktheit

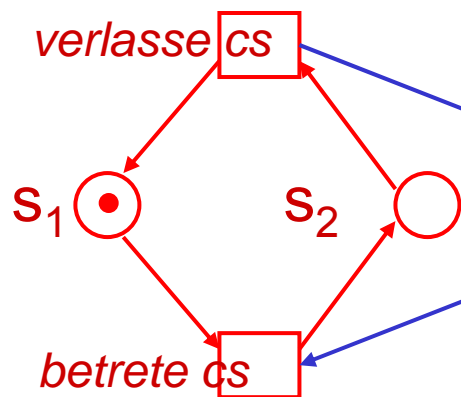
Wenn Stellen-Invariante \underline{i} ex.

$-i(s) > 0 \quad \rightarrow \quad s$ ist beschränkt

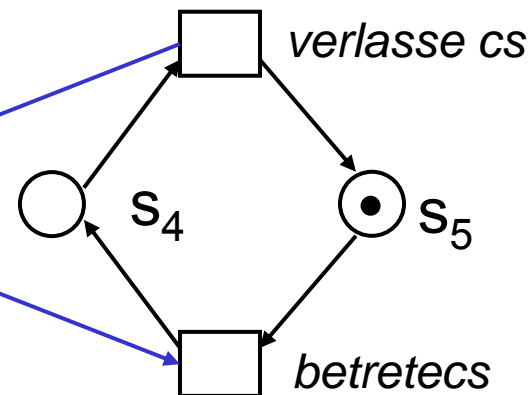
$-i(s') \geq 0$, für alle s'

Beweis: m erreichbar $\rightarrow \underline{i} \cdot \underline{m} = \underline{i} \cdot \underline{m}_0$
 $\rightarrow i(s) \cdot m(s) \leq \underline{i} \cdot \underline{m} = \underline{i} \cdot \underline{m}_0$
 $\rightarrow m(s) \leq \underline{i} \cdot \underline{m}_0 / i(s)$

Prozess 1



Prozess 2



(1, 2, 1, 2, 1)

Beschränktheit

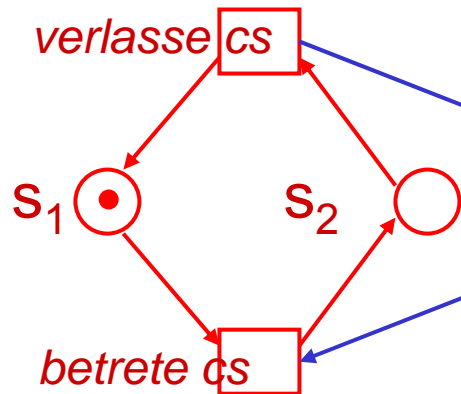
Satz: Zu jedem ganzzahligen Vektor δ gibt es genau dann eine Markierung m , von der die Markierung $m+\delta$ erreichbar ist, wenn $Cx=\delta$ eine nichtnegative Lösung hat.

Folgerung: Zu einem Netz gibt es genau dann eine Anfangsmarkierung, bei der N unbeschränkt ist, wenn $Cx>0$, $x>0$ lösbar ist.

Transitions-Invarianten

= Lösungen von $C \cdot \underline{y} = (0, \dots, 0)$

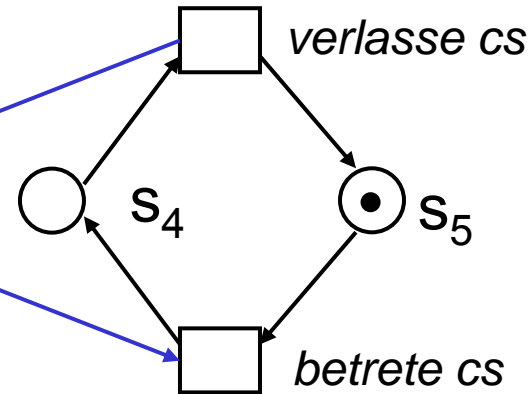
Prozess 1



$(1, 1, 0, 0)$

$(0, 0, 1, 1)$

Prozess 2



$(2, 2, 1, 1)$

$m_0 \xrightarrow{w} m \quad \rightarrow \quad m_0 = m \text{ gdw Parikh}(w) \text{ ist Transitions-Invariante}$

Lebendigkeit, Beschränktheit

N

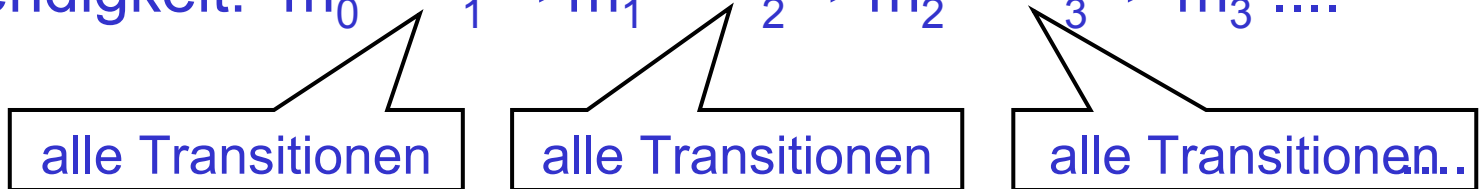
- lebendig
- beschränkt

→

Ex. Transitions-Invariante j
- für alle t , $j(t) > 0$

Beweis:

Wg. Lebendigkeit: $m_0 \xrightarrow{w_1} m_1 \xrightarrow{w_2} m_2 \xrightarrow{w_3} m_3 \dots$



Wg. Beschränktheit: für irgendein $i < j$: $m_i = m_j$

$$\rightarrow m_i \xrightarrow{w_{i+1} \dots w_j} m_j = m_i$$

→ $\text{Parikh}(w_{i+1} \dots w_j)$ ist Transitions-Invariante.

Berechnung von Invarianten

$$\underline{x}C=\underline{0}, \quad Cy=\underline{0}$$

Ganzzahlige Invarianten: normale Eliminationsverfahren

- ganzzahlige homogene Systeme haben immer rationales, also auch ganzzahliges Erzeugendensystem

Positive Invarianten:

... sind Lösungen eines Ungleichungssystems

$$xC=0, \quad x \geq 0 \quad \text{bzw.} \quad Cy=0, \quad y \geq 0$$

→ Methoden der linearen Optimierung

Struktureigenschaften

Weiter:

- Spezielle Netzklassen
- Topologische Eigenschaften

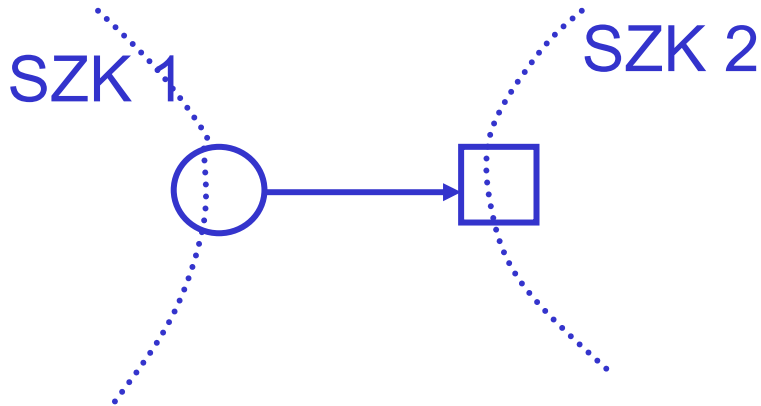
Vereinbarung: N zusammenhängend, alle
Vielfachheiten 1

(nicht zusammenhängende Komponenten können
isoliert betrachtet werden)

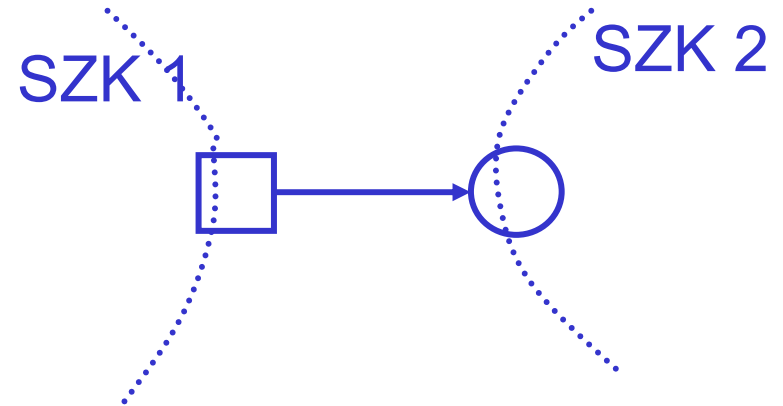
Damit schon mal ein Resultat:

Jedes lebendige und beschränkte Netz ist stark
zusammenhängend

Beweis



t lebendig \rightarrow s muss immer wieder Marken produzieren.
Von SZK 2 gehen aber keine Marken zurück nach SZK 1
 \rightarrow Statt t zu schalten, können wir die Marken auf s liegen lassen \rightarrow s unbeschränkt



t lebendig, keine Rückkopplung mit SZK 1
 \rightarrow s unbeschränkt

Zustandsmaschinen

Def.: Ein Petrinetz, wo jede Transition genau einen Vor- und genau einen Nachplatz hat.

Wesentliche Eigenschaft: Markenzahl bleibt konstant

→ $(1, \dots, 1)$ ist S-Invariante.

→ Jede Zustandsmaschine ist beschränkt

$(1, \dots, 1)$ ist sogar die einzige S-Invariante (bis auf Faktor):

Vor- und Nachplätze müssen jeweils gleiches Gewicht haben, und Netz ist nach Vor. zusammenhängend 145

Lebendigkeit in Zustandsmaschinen

Satz: Eine Zustandsmaschine ist lebendig gdw. sie stark zusammenhängend ist und $m_0 > \underline{0}$ ist

Beweis

„ \rightarrow “ Weil lebendig, und sowieso beschränkt \rightarrow stark zus.

„ \leftarrow “ Sei t Transition und m Markierung. Wg. Invariante $(1, \dots, 1)$ ist $m > \underline{0}$. Sei s bei m markiert. Wg starkem Zusammenhang gibt es einen Weg von s nach t .

Die Folge der Transition auf diesem Weg ist realisierbar,

weil die Stellen auf dem Weg jeweils einzige Vor- und Nachplätze der Transition sind.

Synchronisationsgraphen

Def.: Jede Stelle hat genau eine Vor- und genau eine Nachtransition

Wesentliche Eigenschaften:

- keine Konflikte
- Einzige Transitionsinvariante (bis auf Faktor):
 $(1, \dots, 1)$

Reden jetzt viel über Kreise:

= Folge x_1, \dots, x_n von verschiedenen Knoten, wo für alle $i < n$ $[x_i, x_{i+1}]$ in F , sowie $[x_n, x_1]$ in F .

Synchronisationsgraphen und Kreise

Satz: In jedem Kreis eines Synchronisationsgraphen bleibt die Markenzahl konstant

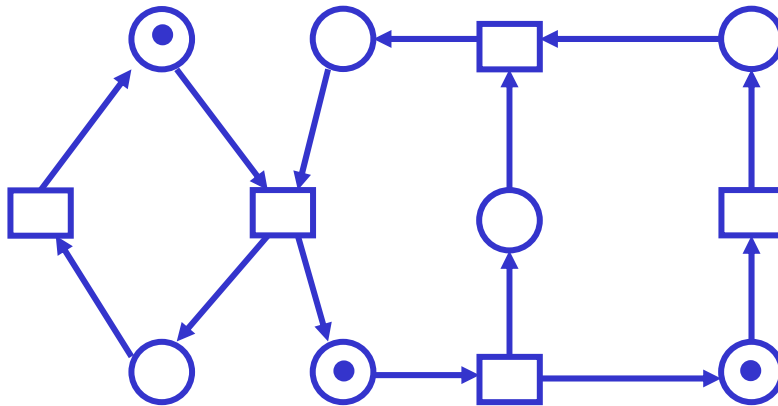
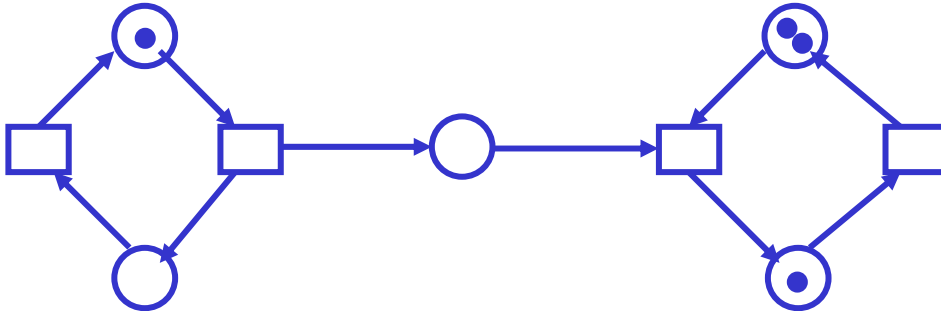
Beweis:

Sei t eine Transition mit einer Stelle des Kreises in ihrer Umgebung. Dann ist sie einzige Vor- oder Nachtransition dieser Stellen, also selbst Teil des Kreises, also hat sie

sowohl genau einen Vor- als auch genau einen Nachplatz auf dem Kreis

→ sie konsumiert genau eine Marke und produziert genau eine Marke auf dem Kreis

Beispiele



Lebendigkeit

Satz: Ein Synchronisationsgraph ist lebendig gdw. jeder Kreis initial markiert ist.

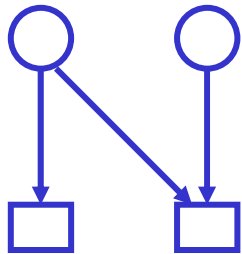
Beschränktheit

Satz: Wenn Synchronisationsgraph N stark zusammenhängend ist, so ist N beschränkt

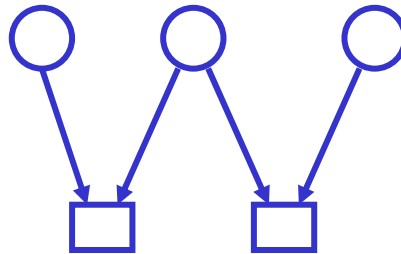
Satz: Wenn Sy.-Graph N lebendig ist, so ist N beschränkt gdw. jede Stelle auf einem Kreis liegt.

Free Choice Netze

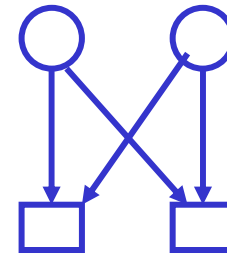
Def: Wenn Transitionen Vorplätze teilen, teilen sie alle Vorplätze.



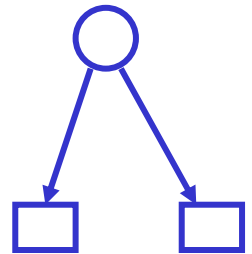
nicht FC



nicht FC



FC



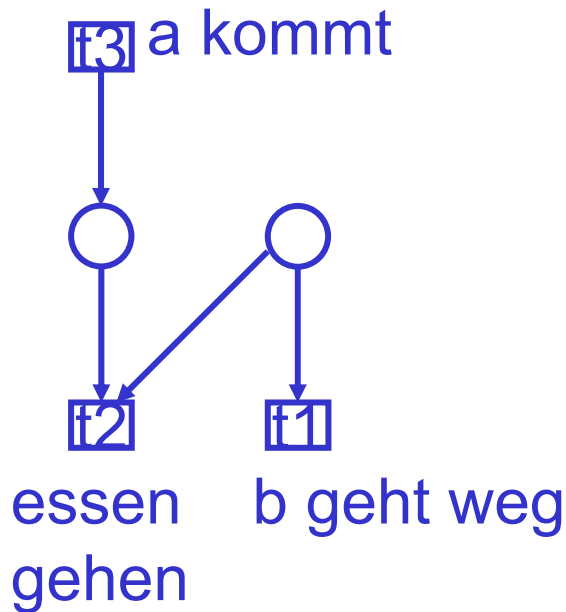
FC

Satz: Jeder Synchronisationsgraph ist ein FC-Netz.

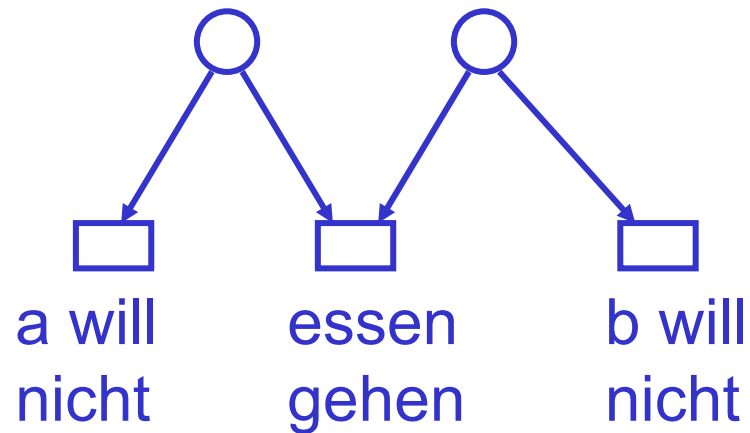
Konfusion

Def.: Eine Situation, wo die Frage, ob t1 und t2 in Konflikt stehen, von einem zu t1 nebenläufigen t3 abhängen

→ FC-Netze sind konfusionsfrei



N-Konfusion
(aktivierende Konfusion)



M-Konfusion
(deaktivierende Konfusion)

Konfliktcluster

Def.: Enthält einen Netzknoten x und

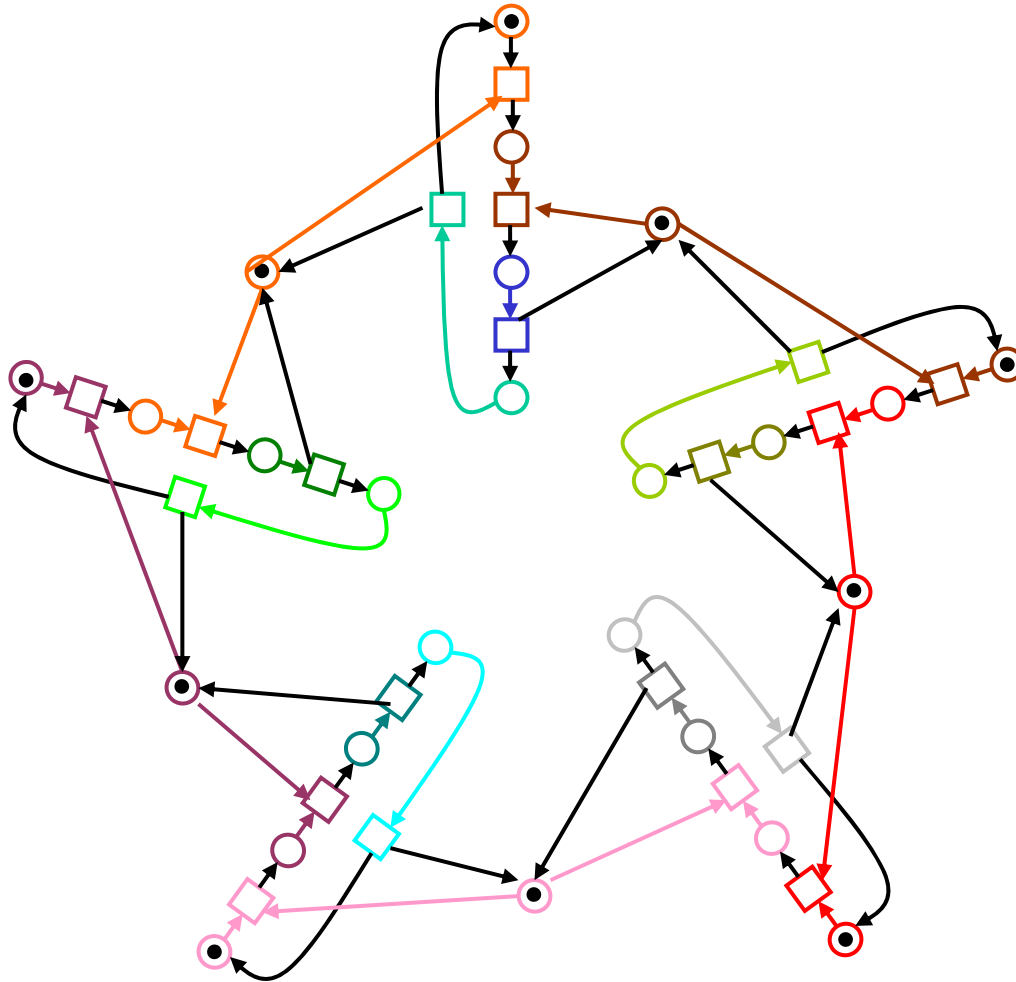
- zu jeder Stelle alle Nachtransitionen
- zu jeder Transition alle Vorplätze.

Satz: Die Menge aller Konfliktcluster ist eine Zerlegung von

$$S \cup T.$$

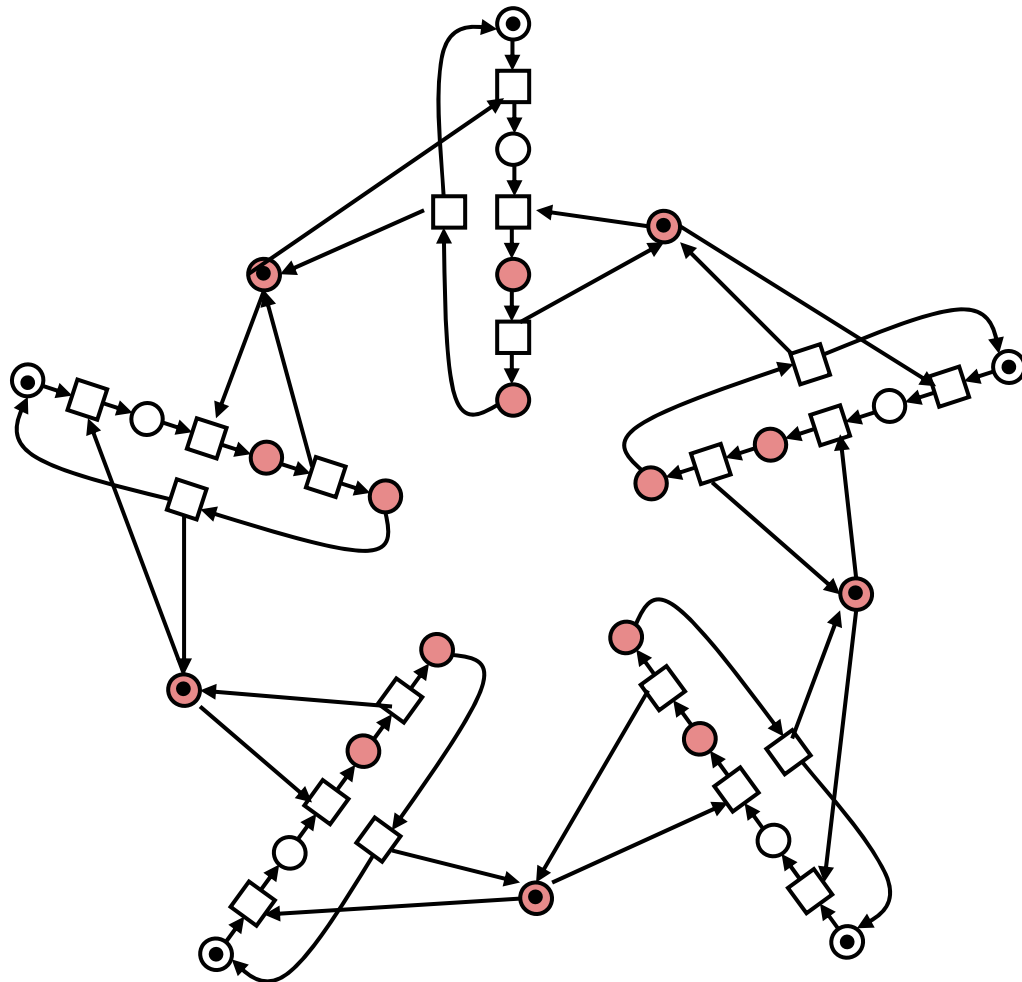
In FC-Netzen ist zu einem Cluster C immer $S_C \times T_C \subseteq F$
→ Ist eine Transition eines Clusters aktiviert, sind es alle.

Konfliktcluster



Siphons (strukturelle Deadlocks)

Def.: $\emptyset \neq D \subseteq S$ heißt Siphon falls $\bullet D \subseteq D \bullet$.



Wichtigste Eigenschaft:

Einmal leer – immer leer.

Beweis: Ist D leer, so ist keine Transition aktivierbar, die Marken nach D schaffen kann.

Siphons und Verklemmungen

Satz: In jeder toten Markierung eines (beliebigen!) Netzes gibt es einen unmarkierten Siphon.

Beweis:

Sei m tote Markierung und $D = \{s \mid m(s) = 0\}$

Da bei m keine Transition aktiviert ist, gilt $D \bullet = T$.

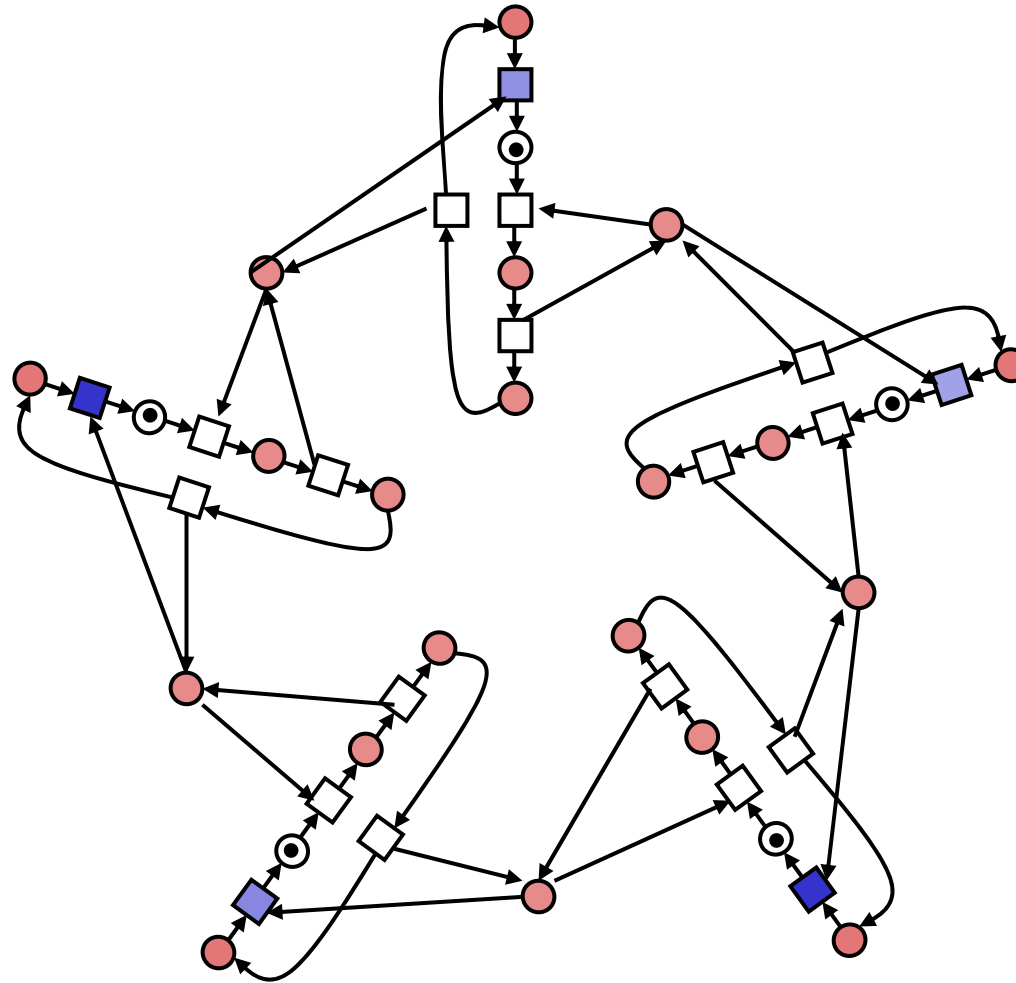
Damit ist $\bullet D \subseteq D \bullet$.

Folgerung: Jedes Netz ohne Siphons ist verklemmungsfrei.



Hilft aber nicht: In jedem stark zusammenhängenden Netz ist S Siphon.

Beispiel



Fallen

Def: $Q \subseteq S$ heißt Falle, wenn $Q \bullet \subseteq \bullet Q$.

Wichtigste Eigenschaft: Einmal nicht leer, immer nicht leer.

Folgerung: Jedes Netz, in dem jeder Siphon eine initial markierte Falle enthält, ist verklemmungsfrei.

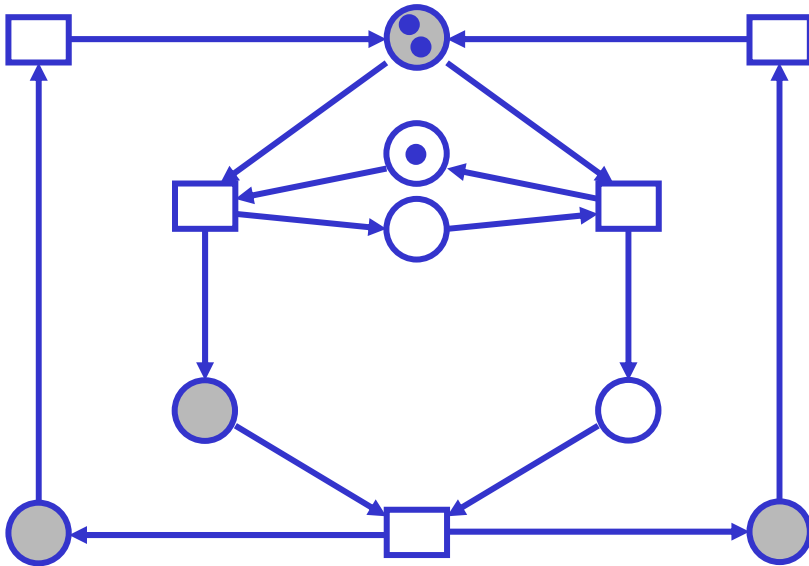
Wissenswertes über Siphons und Fallen

- Vereinigung von Siphons (Fallen) ist Siphon (Falle)
- Jeder Siphon enthält eindeutig bestimmte max. Falle (ggf. leer)
- N verklemmungsfrei wenn die max. Falle in jedem min. Siphon initial markiert ist.
- Kreis in Synchronisationsgraph ist Siphon und Falle
- Zu jeder nichtneg. S -Invariante i ist $\text{supp}(i)$ Siphon und Falle

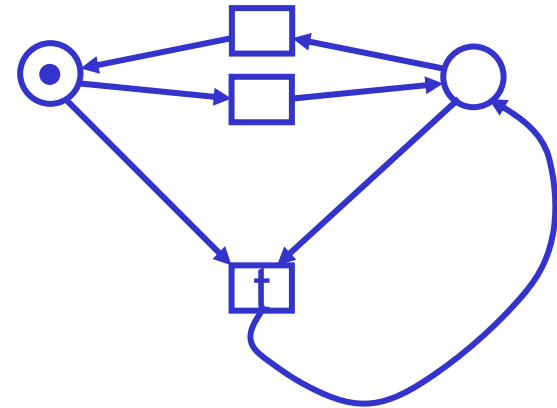
Commoners Theorem

Satz: Ein FC-Netz ist lebendig gdw jeder Siphon eine initial markierte Falle enthält.

Gegenbeispiele



lebendig, aber max. Falle
im markierten Siphon ist
leer, also nicht markiert.



einzigster Siphon (S) ist
markierte Falle, t aber
nicht lebendig

Letztes Resultat im Kapitel Strukturelle Analyse

Das Rang-Theorem:

Zu einem FC-Netz N gibt es eine Anfangsmarkierung, bei der N lebendig und beschränkt ist gdw.

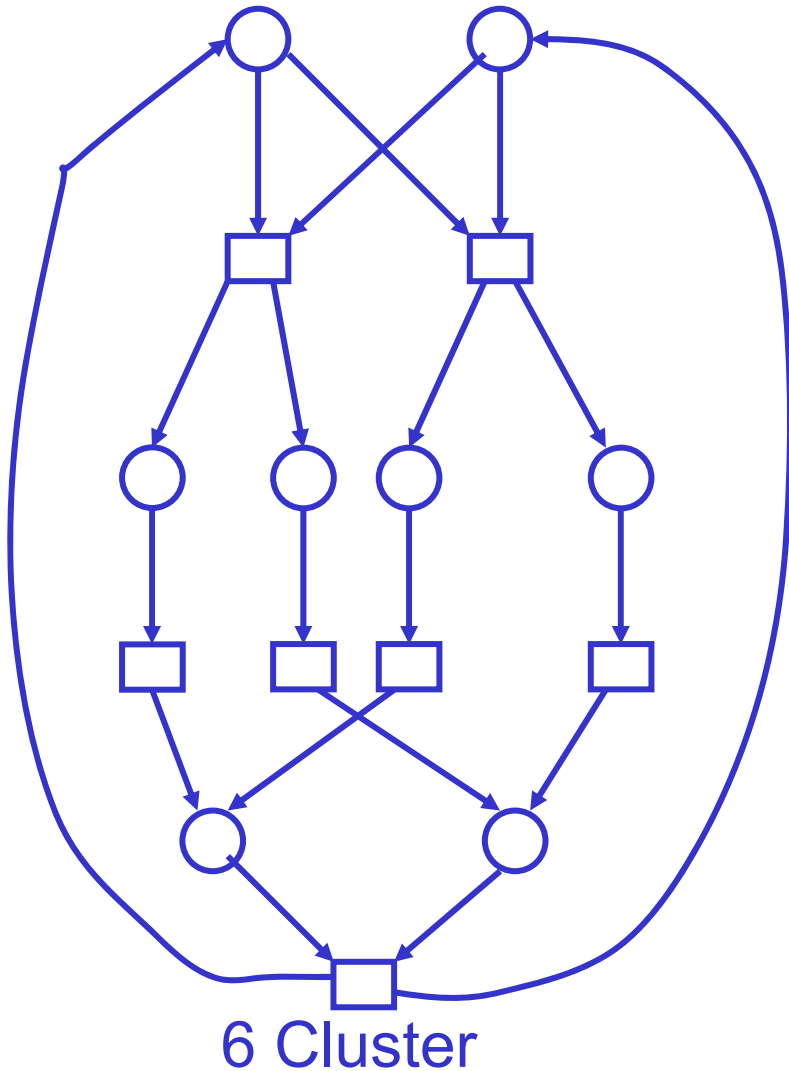
- (a) N ist stark zusammenhängend, hat mind. eine Stelle, eine Transition
- (b) N hat eine strikt positive S-Invariante
- (c) N hat eine strikt positive T-Invariante
- (d) $\text{Rang der Inzidenzmatrix} = \text{Anzahl der Konfliktcluster} - 1$.

→ In Polynomialzeit entscheidbar!

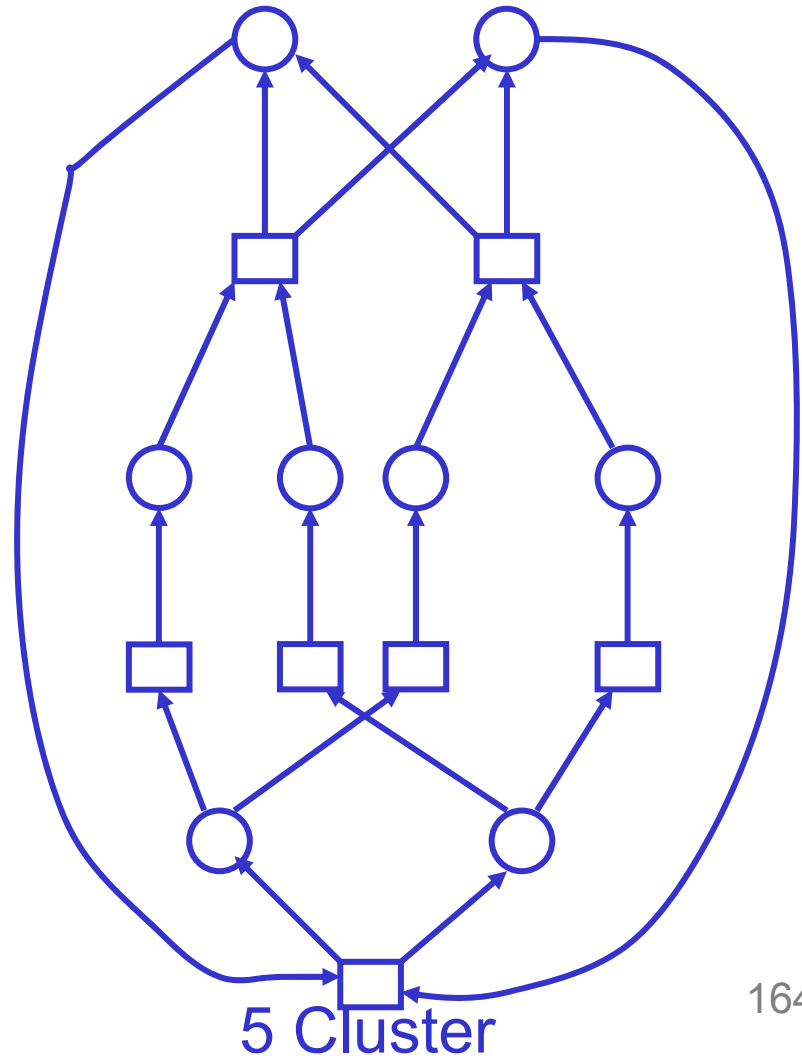
Beispiel

rang= 5 S-Inv.(1,1,1,1,1,1,1,1) T-Inv (1,1,1,1,1,1,2)

strukturell L&B



nicht strukturell L&B



Zusammenfassung Strukturtheorie

1. Lineare Algebra

- Zustandsgleichung \rightarrow notw. Kriterium für Erreichbarkeit
- S-Invarianten \rightarrow Erreichbarkeit (notw.)
- Beschränktheit (hinr.)
- Lebendigkeit (notw.)

Für alle Petrinetze,		für alle Stellen-Invarianten \underline{i}	
-lebendig	\rightarrow	- ohne neg. Komponenten	: $\underline{i} \cdot \underline{m}_0 > 0$
-ohne isolierte Stellen		- mit mind. einer pos. Komp. s	

- T-Invarianten \rightarrow L&B (notw.)
- Rangtheorem \rightarrow L&B in Free-Choice Netzen (hinr+notw.)

Zusammenfassung Strukturtheorie

2. Netzklassen

- Zustandsmaschinen
- Synchronisationsgraphen
- Free-Choice Netze

3. Netzstrukturen

- Siphons
- Fallen
- S-Komponenten
- T-Komponenten

Strukturelle Resultate: Beschränktheit

Allgemein: hinr.: Existenz strikt pos. S-Invariante

hinr.: Überdeckbarkeit mit S-Komponenten

Zustandsmaschinen: immer beschränkt

leb. Synchronisationsgraphen: notw.+hinr.: Überdeckbarkeit mit Kreisen

leb. FC-Netze: notw.+hinr.: Existenz strikt pos. S-Invariante
notw.+hinr.: Überdeckbarkeit mit S-Komponenten

Strukturelle Resultate: Lebendigkeit

Zustandsmaschinen: notw.+hinr.: starker
Zusammenhang

Synchronisationsgraphen: notw.+hinr.: alle Kreise
markiert

FC-Netze: notw.+hinr.: jeder Siphon enthält markierte
Falle

strukturelle Resultate: L&B

Allgemein: notw.: Existenz strikt pos. T-Invariante
notw.: starker Zusammenhang

Zustandsmaschinen: notw.+hinr.: starker
Zusammenhang

Synchronisationsgraphen: notw.+hinr.: von markierten
Kreisen überdeckt

FC-Netze: notw: von S-Komponenten überdeckt
notw.: von T-Komponenten überdeckt
notw.+hinr.: Rangtheorem

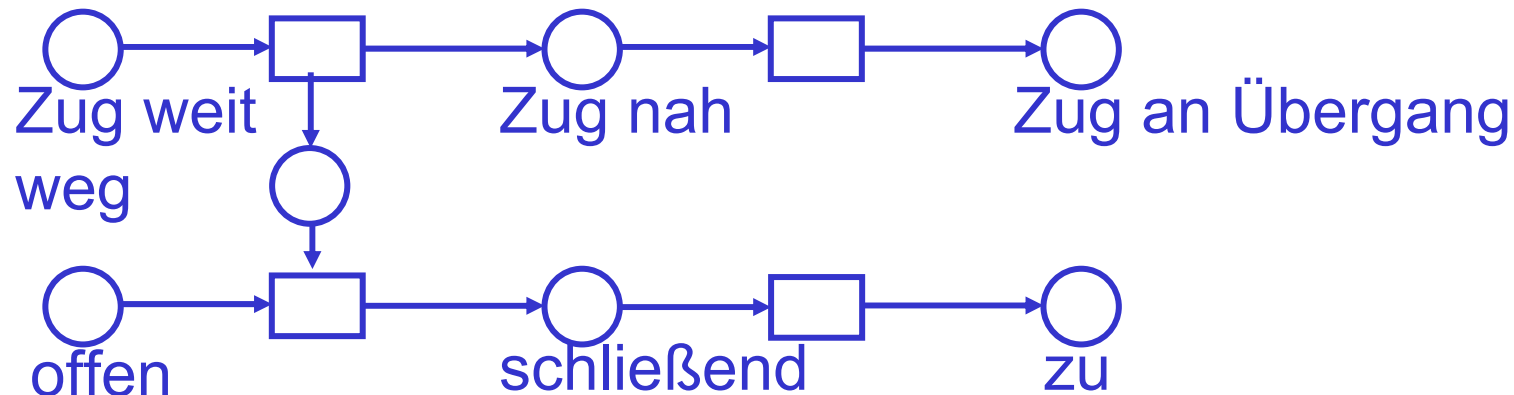
Modellierung und Analyse quantitative Aspekte

Zeitnetze

Ziel: Echtzeitverhalten

Echtzeit; Typisches Beispiel Bahnübergang:

Zug passiert Sensor → Schranke schliesst sich



Viele Zeitkonzepte

Uhren an Marken: neue Marken unreif, reifen (ggf. verfallen)

Uhren an Stellen: so ähnlich

Uhren an Bögen: Bögen nur in einem Zeitfenster durchlässig

Uhren an Transitionen: Schaltdauer oder Schaltfenster ab
Zeitpunkt der Aktivierung

Immer: gewisser Schaltzwang („Sofortschaltregel“)

Netze mit Schaltdauer

$[S, T, F, W, d, z_0]$ $d: T \rightarrow \text{Nat} \setminus \{0\}$ $z_0: S \rightarrow \text{Nat}, T \rightarrow \text{Nat} \cup \{\#\}$
 $z_0(t) = \#$

$z(s)$ – Markierung

$z(t) = \#$ schaltet nicht

$z(t) = k$ schaltet noch k Einheiten

Übergang = Schalten + 1 Tick Zeitverlauf:

1. Für alle t mit $z(t) = 0$, produziere die Marken für t und setze $z(t)$ auf $\#$
2. Wähle eine max. Menge U von Transitionen t mit $z(t) = \#$, die gemeinsam aktiviert sind.
3. Entferne die durch U konsumierten Marken.
4. Setze $z(t)$ auf $d(t)$ für t aus U
5. Subtrahiere 1 von allen $z(t) \neq \#$

Schaltregeln

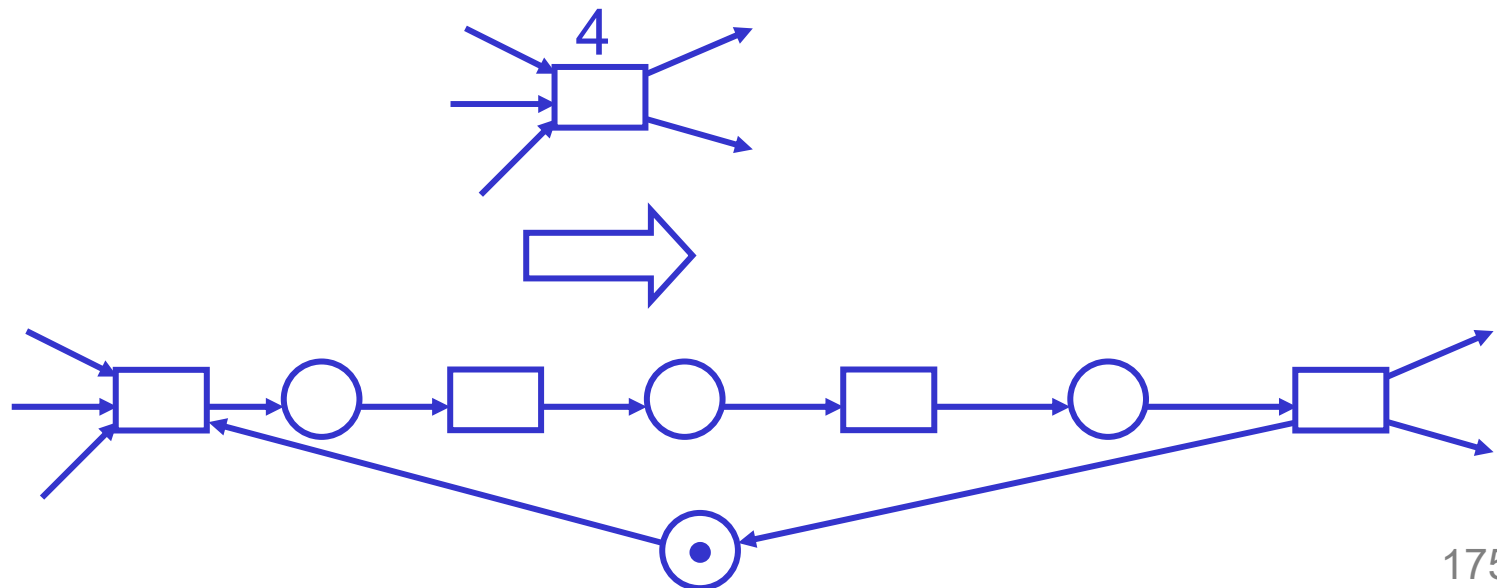
Hatten bei Petrinetzen: Einzelschaltregel: $m[t > t']$

Auch möglich: Schritt-Regel: Schalte immer eine Menge gemeinsam aktivierter Transitionen (gemeinsam aktiviert = genug Marken für alle da)
→ gleiche Menge erreichbarer Markierungen, mehr Bögen im Erreichbarkeitsgraph

Max-Schritt-Regel: Schalte immer Mengen von Transitionen, die maximal sind bzgl. Inklusion.

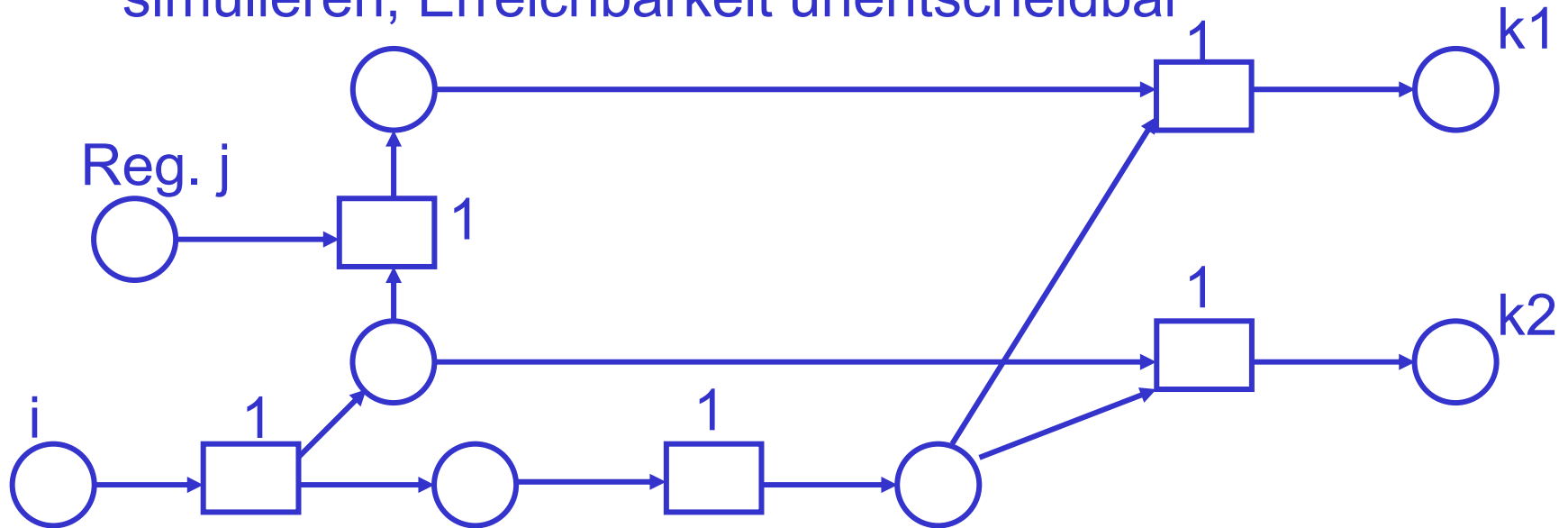
Schaltdauer und Max-Schritt-Regel

- Netz, bei dem alle Schaltdauern 1 sind, ist äq. zum zugrundeliegenden Netz bei Max-Schritt-Schaltregel.
- Schaltdauernnetz kann zu äq. Petrinetz unter Max-Schritt-Regel umgeformt werden



Schaltdauer und Entscheidbarkeit

Netze mit Schaltdauer können Zählerautomaten simulieren, Erreichbarkeit unentscheidbar



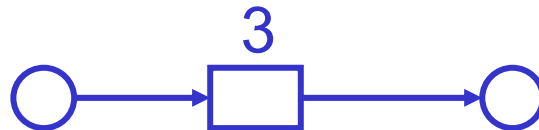
T&DEC

Beschränkte Schaltdauernetze

Beschränkte Schaltdauernetze haben endlichen Zustandsgraph

Wenn zugrundeliegendes PN beschränkt, so auch das ZeitdauerNetz (Umkehrung gilt nicht)

Erreichbarkeit, Invarianten etc.: Problem sind Marken im „Bauch“ der Transitionen



Aber: Jede im Dauernetz erreichbare Markierung ist im zugrundeliegenden Petrinetz überdeckbar.

Schaltdauer und Free Choice

Satz: jedes lebendige FC-Netz ist bei jeder Schaltdauerzuordnung lebendig.

Zeigen: N mit einer Schaltdauer nicht lebendig \rightarrow leerer Siphon (also Originalnetz nicht lebendig)

Wählen z so, dass bei z jede Transition und jede Stelle entweder tot oder lebendig ist.

$D :=$ Menge der toten Stellen bei z

Sei $t \in \bullet D \rightarrow t$ tot. $\rightarrow t$ muss toten Vorplatz haben, da Marken von den Vorplätzen nur von den Transitionen

konsumiert werden können, die gleichzeitig mit t aktiviert sind.

Also: alle Vorplätze lebendig $\rightarrow t$ aktivierbar, Wid.

Also: $t \in D\bullet$, also ex. leerer Siphon, im Originalnetz realisierbar
 $\rightarrow N$ nicht lebendig

Intervallnetze

$[S, T, F, W, \text{eft}, \text{lft}, z_0]$ $\text{eft}: T \rightarrow \text{Nat}$
 $\text{lft}: T \rightarrow \text{Nat} \cup \{\infty\}$ $\text{eft}(t) \leq \text{lft}(t)$

$z(s)$ – Markierung

$z(t) \in [0, \text{lft}(t)] \cup \{\#\}$ $z_0(t) = 0$, falls t aktiviert, $\#$ sonst.

Verhalten

(A) Schalten einer Transition

- Vor.: t aktiviert, $z(t) \in [\text{eft}(t), \text{lft}(t)]$

→ schalte t ,

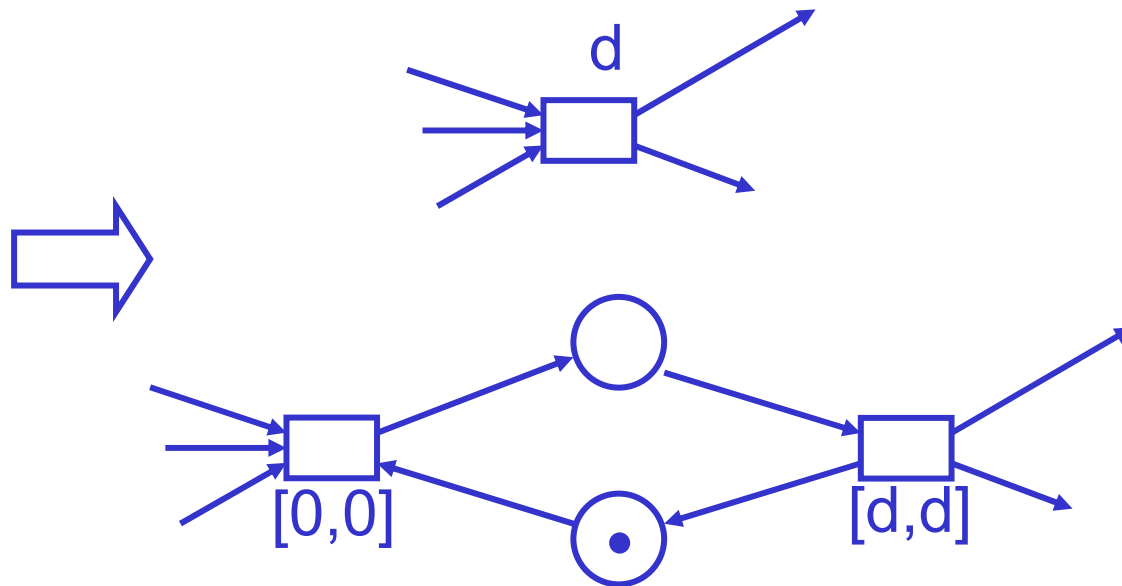
für alle $t' \in t \bullet \bullet \cup (\bullet t) \bullet$: $z(t') := 0$, falls t' aktiviert,
 $z(t') = \#$, sonst.

(B) Zeitverlauf (reelles(!!!) $\tau > 0$)

- Vor.: für kein t ist $z(t) + \tau > \text{lft}(t)$

→ für alle t mit $z(t) \neq \#$: $z(t) := z(t) + \tau$.

Intervallnetze können Dauernetze simulieren



also: Erreichbarkeitsproblem etc. unentscheidbar

Erreichbarkeitsanalyse für beschränkte Intervallnetze

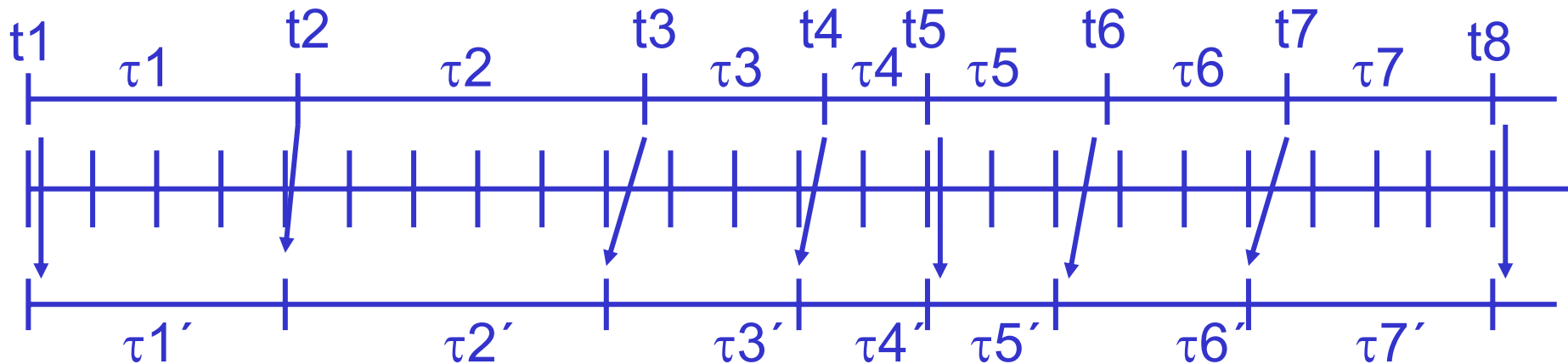
2 Ansätze:

1. Integrierer Erreichbarkeitsgraph
2. Zeitzone

Integrier Erreichbarkeitsgraph

Betrachten nur Zeitverläufe mit ganzzahligen τ .

Rechtfertigung: Zu jeder Sequenz, die bei ganzzahliger Uhrenstellung beginnt und bei ganzzahliger Uhrenstellung endet, gibt es eine, die nur über ganzzahlige Zwischenzustände verläuft.



$$z(t) = \sum_{i=j}^k \tau_i$$

$$z'(t) = \sum_{i=j}^k \tau_i'$$

Falls $z(t)$ ganzzahlig,
so $z(t) = z'(t)$, sonst
 $\lfloor z(t) \rfloor \leq z'(t) \leq \lceil z(t) \rceil$

Also: $z(t) \in [\text{eft}(t), \text{lft}(t)]$
gdw. $z'(t) \in [\text{eft}(t), \text{lft}(t)]$

Folgerungen

- Qualitative Eigenschaften können am integren Graph abgelesen werden.
- Minima, Maxima (kürzeste/längste Wege) liegen auf ganzzahligen Zeiten
- Wenn N beschränkt und alle $l_{ft} \neq \infty$, so ist integrer Graph endlich

Zonengraph

Idee: Ein Zustand repräsentiert eine Markierung m und viele Uhrenstellungen U , durch Ungleichungen abgegrenzt

Übergang von $[m, U]$ = Schalten einer Transition t und beliebiger Zeitverlauf, beginnend bei jeder Uhrenstellung aus U , wo t aktiviert ist.

Bedingung: Die Menge der Uhrenstellungen, bei der t aktiviert ist, ist nicht leer.

Neue Markierung – klar

Neue Uhrenstellungen - interessant

Neue Uhrenstellungen

Geg. Ungleichungssystem E , das U beschreibt.

Ungleichungen haben Form $k_1 \leq z(t) \leq k_2$ und

$k_1 \leq z(t) - z(t') \leq k_2$, für aktivierte t, t' .

(k_2 kann ∞ sein), oder $z(t) = \#$, für inaktive t .

t aktiviert heißt $z(t) \geq \text{eft}(t) \rightarrow$ neue Ungleichung.

Wenn neues Ungleichungssystem lösbar, gibt es also Uhrenstellungen in U , wo t schalten kann.

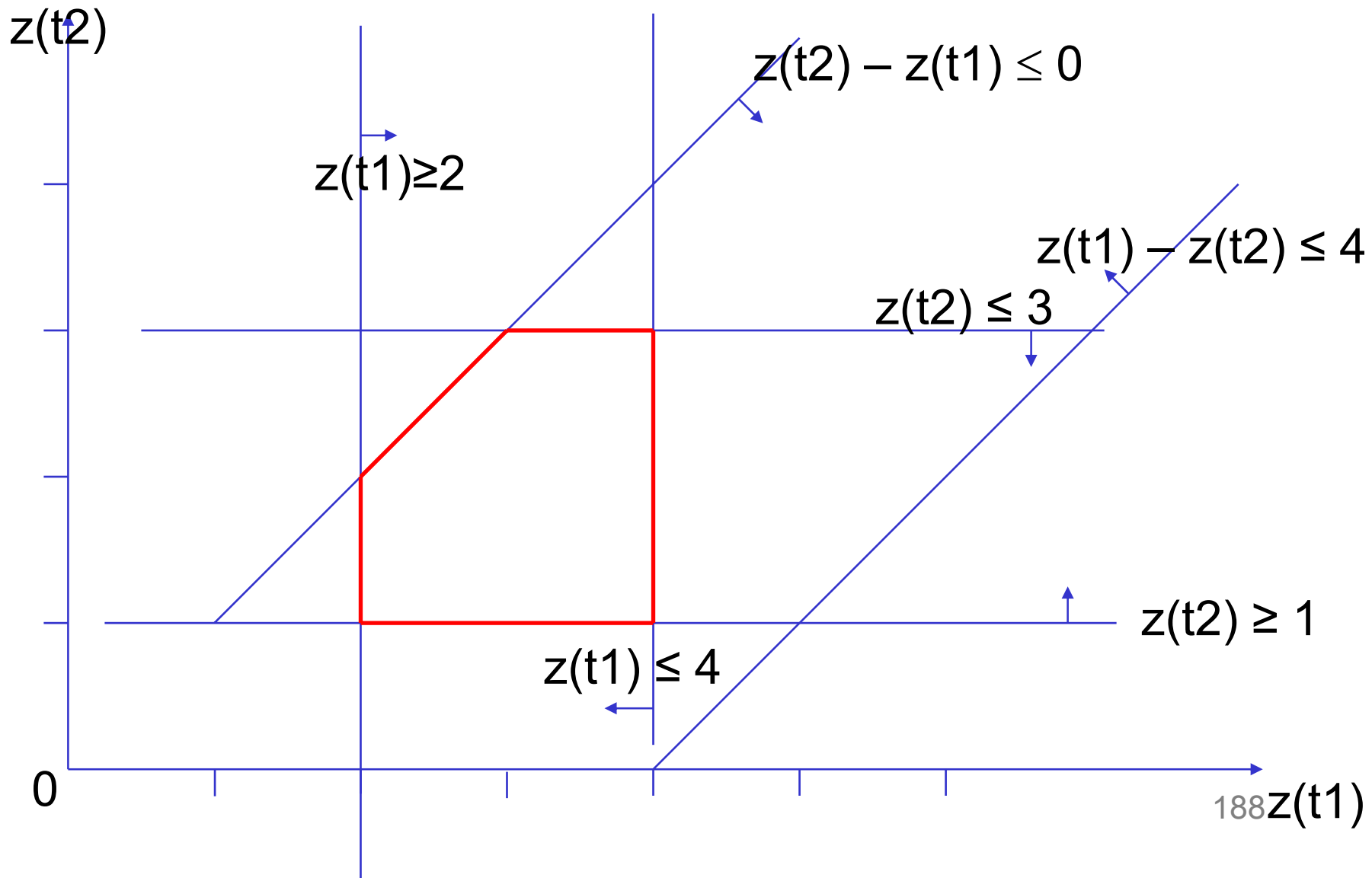
Neue Uhrenstellungen

Haben: Ungleichungssystem U_1 , das alle Uhrenstellungen beschreibt, bei denen t schalten kann.

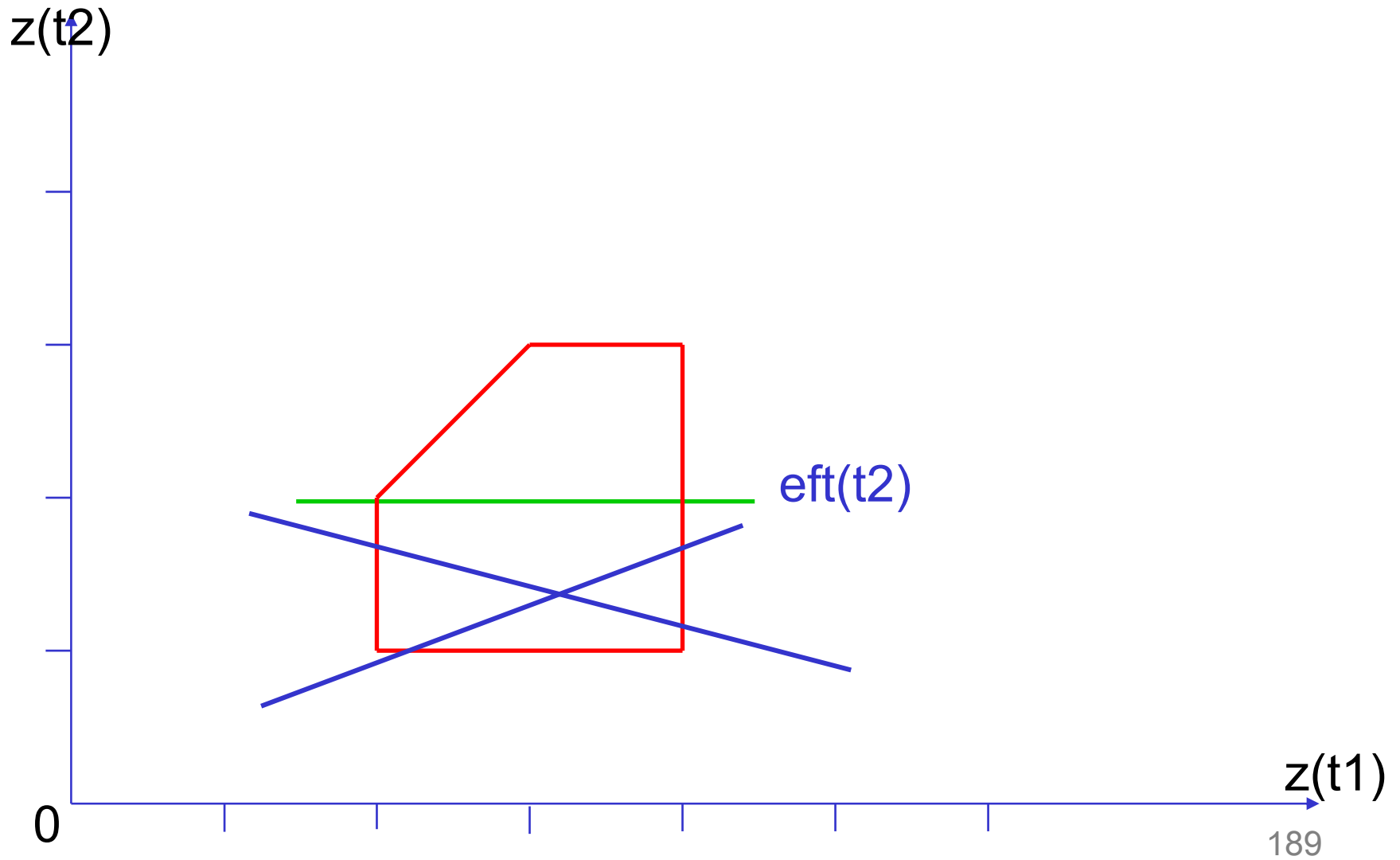
Nun: Effekt des Schaltens und Zeitverlaufs

Ersetze für alle frisch aktivierten Transitionen t die Ungleichung $k_1 \leq z(t) \leq k_2$ durch $0 \leq z(t) \leq lft(t)$ und alle Ungleichungen $k_1 \leq z(t) - z(t') \leq k_2$ durch diejenige Ungleichung $k_1^* \leq z(t') - z(t) \leq k_2^*$, für die $k_1^* \leq z(t') \leq k_2^*$ in U' vorkommt (analog $z(t) - z(t')$)
→ neues Ungleichungssystem

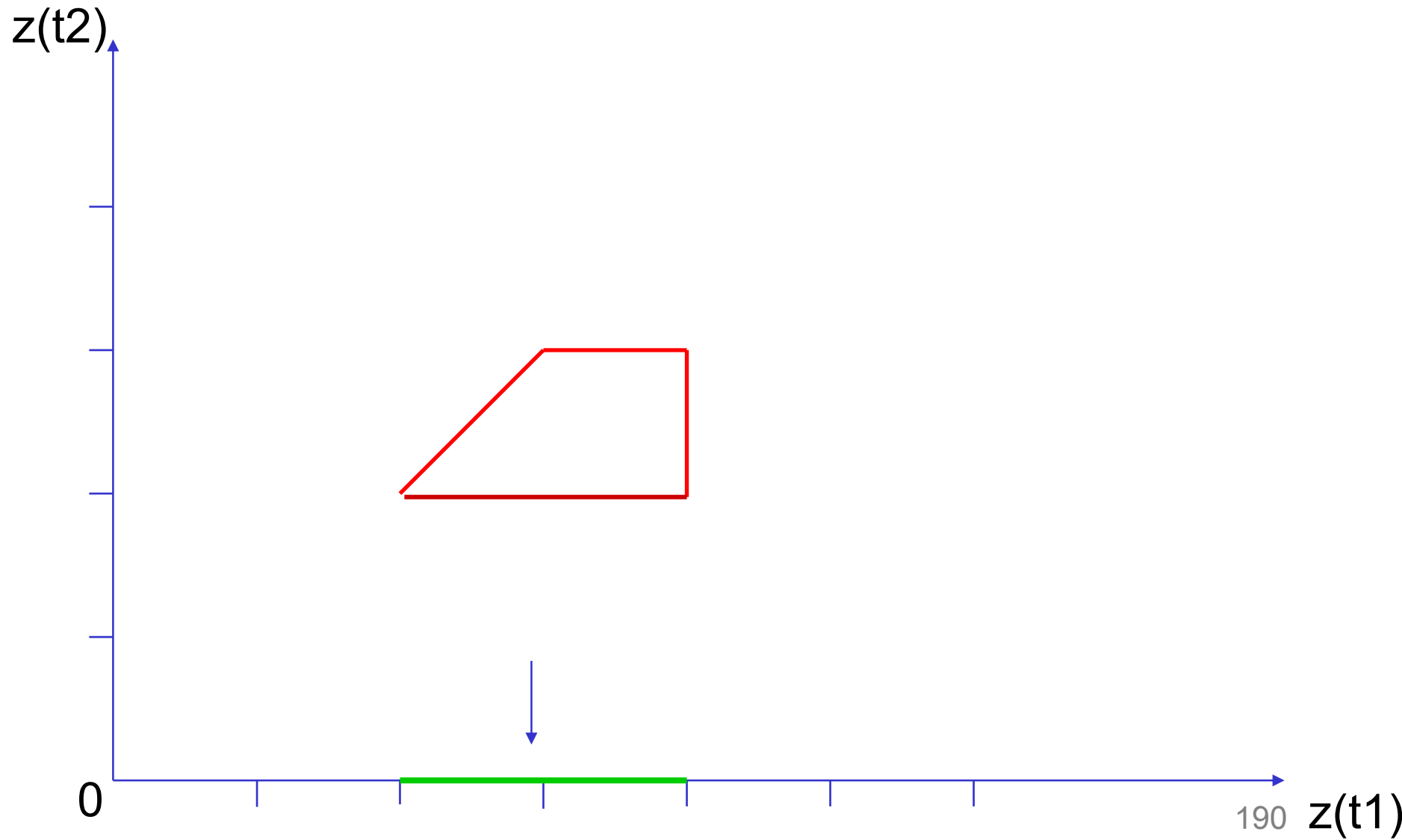
Zonengraph



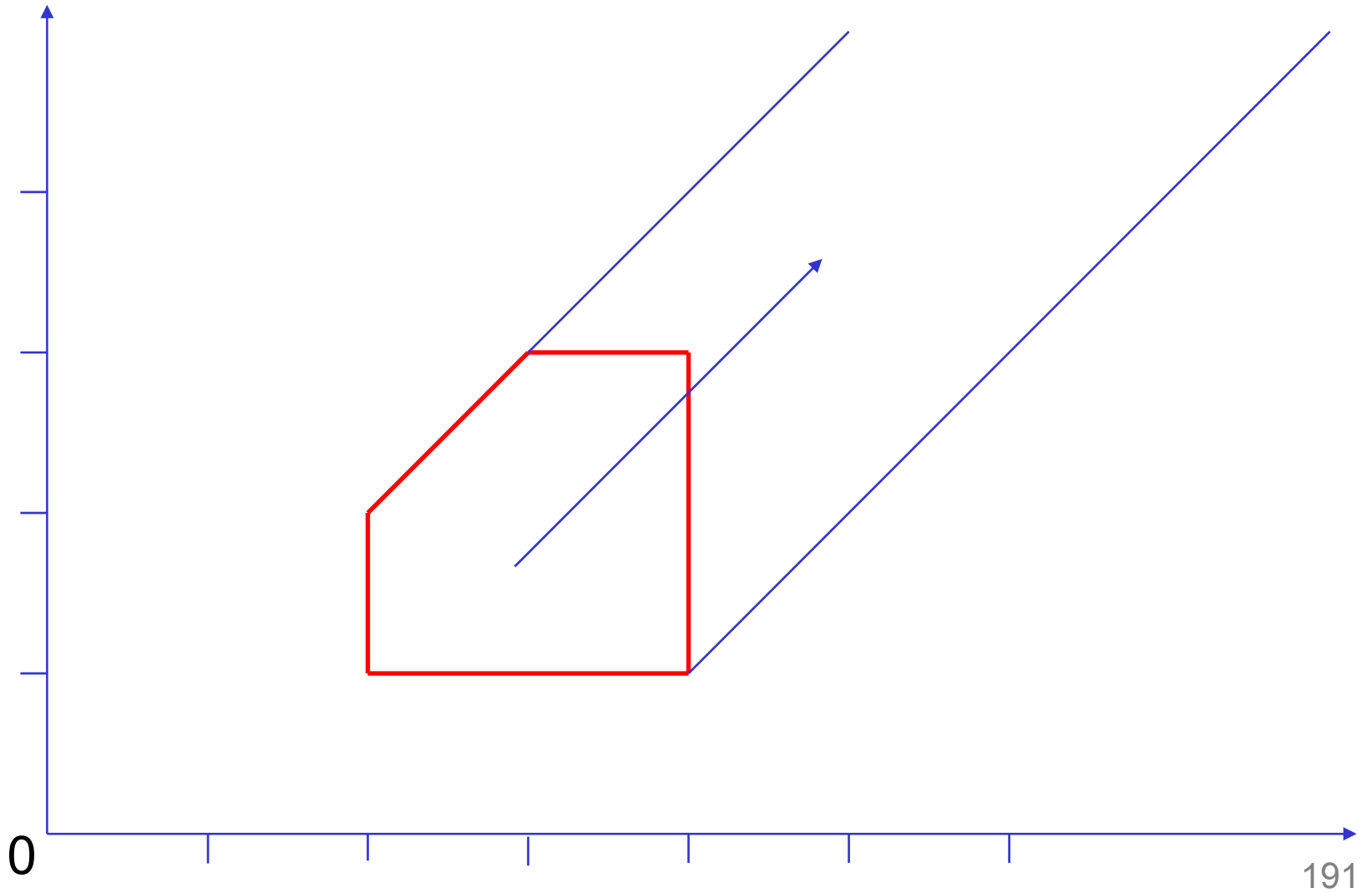
1. Aktivierung von t



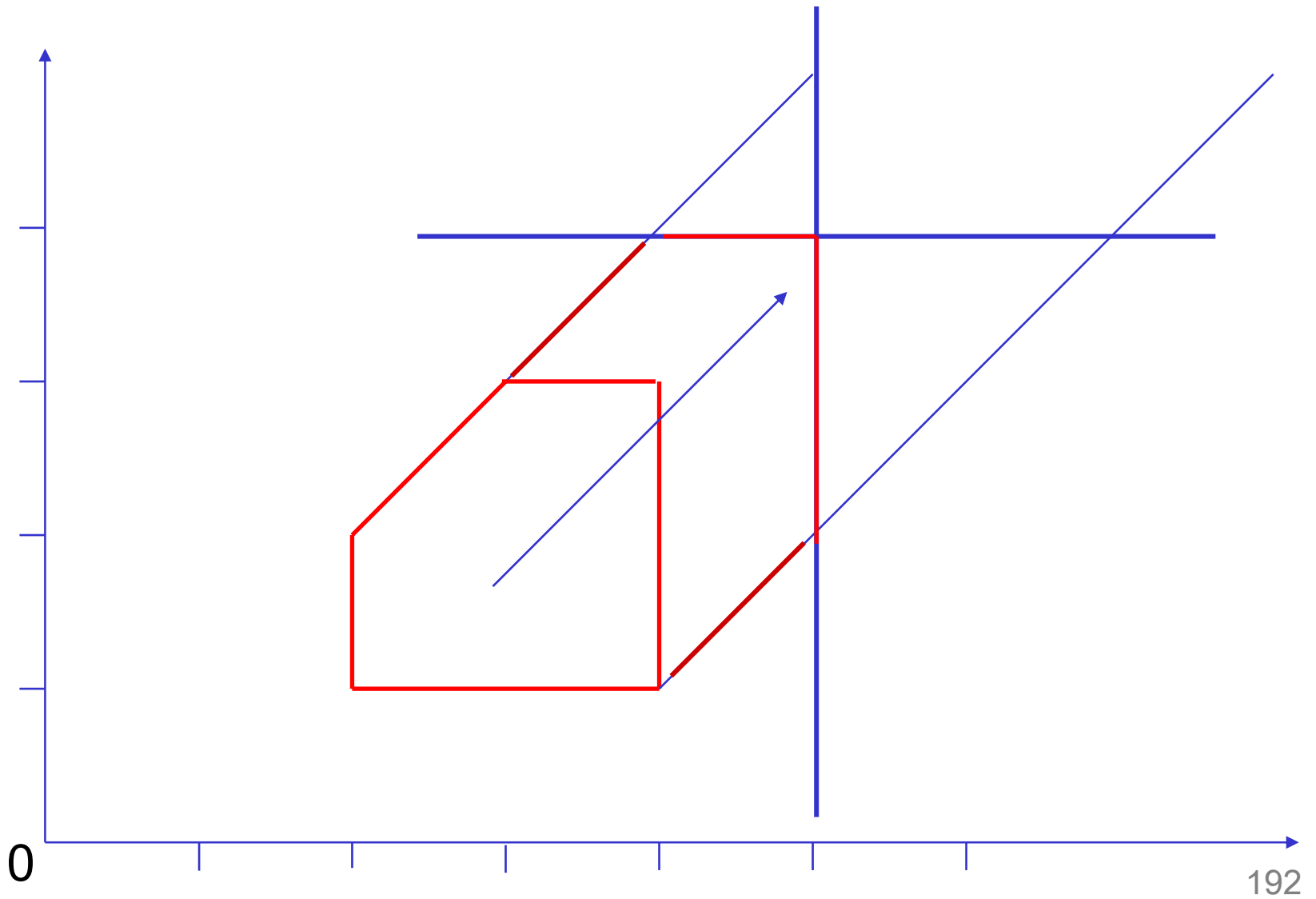
Uhr auf 0 stellen



Zeitverlauf



Abgrenzung gegen lfts



Implementation von Zonen

bisher: Zone = Konjunktion von Constraints

Problem: Eine Zone hat viele Darstellungen

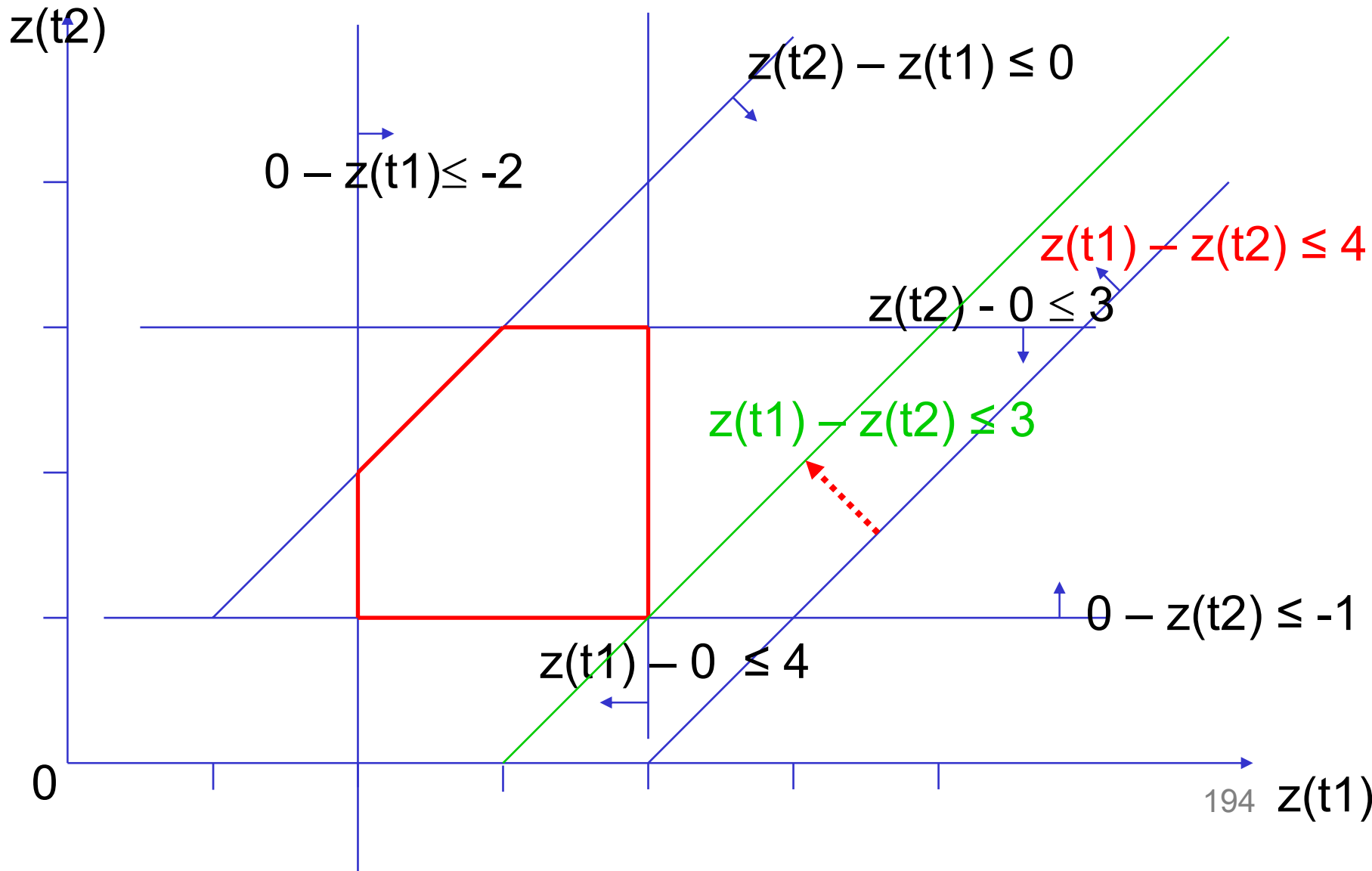
Lösung: Normalform

=

- für jedes $t, t' \in T \cup \{0\}$ genau ein Constraint der Form $z(t1) - z(t2) \leq k$
- jeweils das kleinstmögliche k

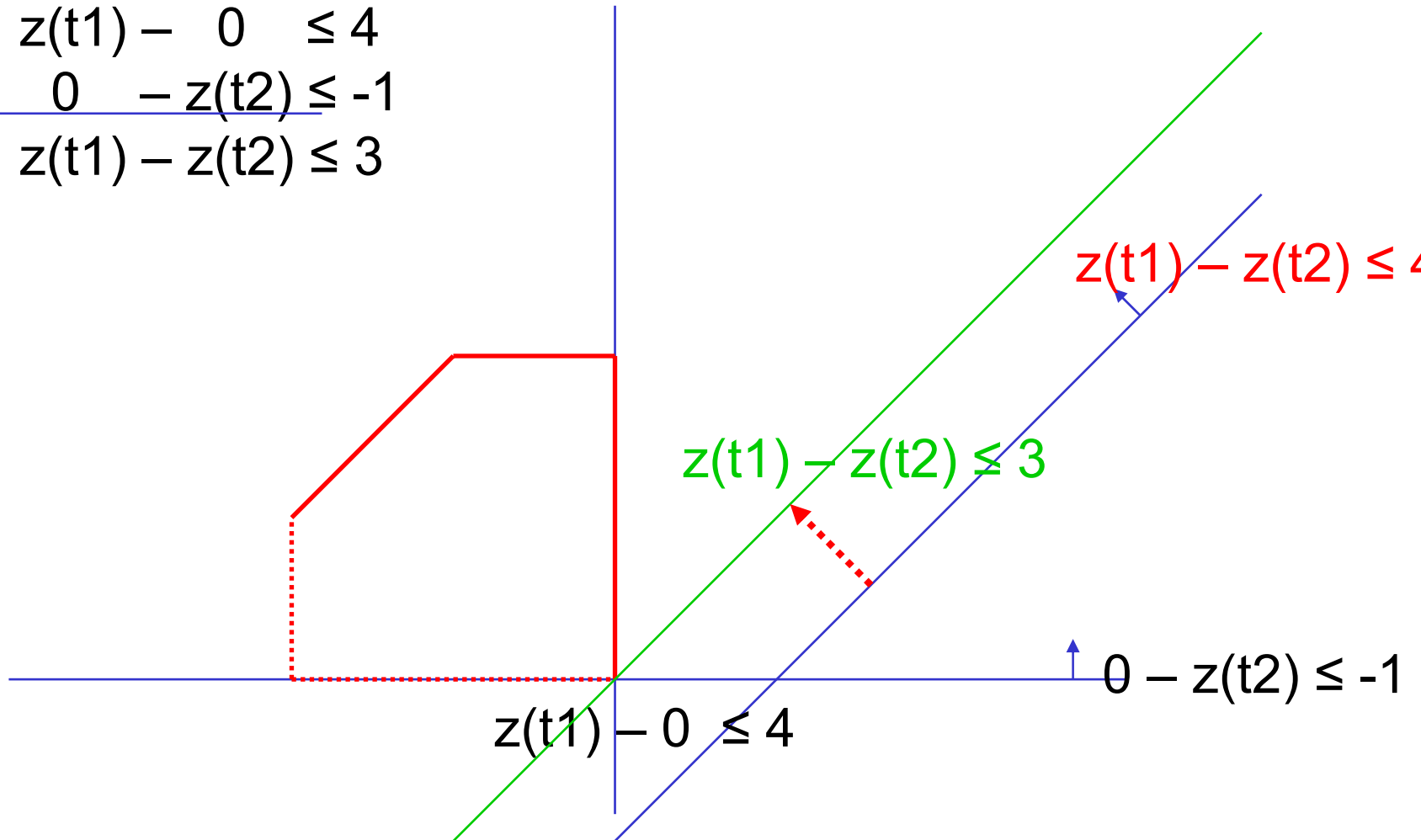
Normalform ist eindeutig!

geometrische Veranschaulichung



Berechnung engerer Constraints

$$\begin{array}{rcl} z(t1) - 0 & \leq & 4 \\ + & 0 & - z(t2) \leq -1 \\ \hline z(t1) - z(t2) & \leq & 3 \end{array}$$



Normalisierung

Algorithmus:

1. Wandle Constraints $z(t) \leq/\geq k$ in Constraints $z(t) - 0 \leq k$ und/oder $0 - z(t) \leq -k$ um
2. Für jedes t, t' aus $T \cup \{0\}$ behalte das engste Constraint bzw. füge $c_i - c_j \leq \infty$ ein
3. Wiederhole, bis nix Neues:
Für jedes $z(t_1) - z(t_2) \leq k_1$, $z(t_2) - z(t_3) \leq k_2$,
füge $z(t_1) - z(t_3) \leq k_1 + k_2$ ein, falls es enger ist als das vorhandene Constraint für $z(t_1) - z(t_3)$;

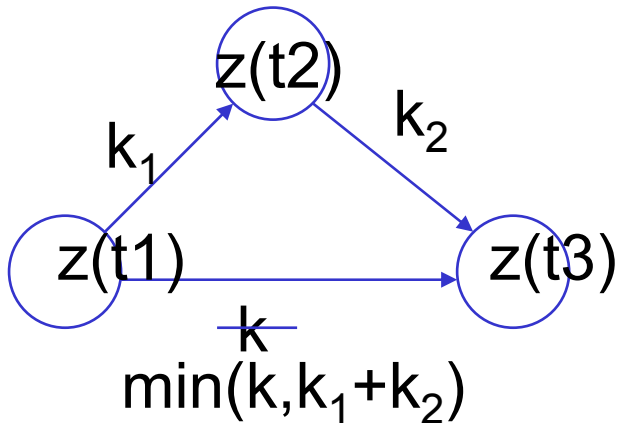
Algorithmus führt zu Normalform

Shortest Path Closure

= Graphalgorithmus für Normalisierung

geg: Graph G mit "Kosten" für jede Kante

ges: Graph G' , wo an jeder Kante $[a,b]$ die Kosten des billigsten Weges von a nach b in G stehen



→ Wert von $z(t_i) - z(t_j)$

Algorithmus von Floyd & Warshall
 $O(n^3)$

Stochastische Petrinetze

Schalten einer Transition ist Ereignis, das stochastischen Gesetzmäßigkeiten folgt

- Aufenthaltswahrscheinlichkeiten für Markierungen
- Durchsatz von Transitionen
- Durchschnittliche Markenzahl auf Stelle
- ...

Crashkurs Stochastik

Zufallsvariable = Variable mit ungewissem Wert, aus bekanntem Wertebereich

Experiment = Bestimmung des Wertes einer Zufallsvariable

Ereignis = Menge von Werten aus dem Wertebereich

$P(A)$ = Wahrscheinlichkeit, dass Ereignis A bei einem Experiment eintritt

Beispiel: Zufallsvariable = Augenzahl eines Würfels

Experiment = einmal würfeln

Ereignisse: - Eine Sechs!

- Augenzahl ungerade

$P(\text{Sechs}) = 1/6$ $P(\text{ungerade}) = 1/2$

A,B ausschließend, falls $A \cap B = \emptyset$.

$A_1 \dots A_k$ erschöpfend, falls $A_1 \cup \dots \cup A_k = \text{Wertebereich}$

Crashkurs Stochastik

Bedingte Wahrscheinlichkeit $P(A|B)$:

Wahrscheinlichkeit, dass A eintritt, falls vom (gleichen) Experiment schon bekannt ist, dass B eintrat

Beispiel: $P(\text{sechs}) = 1/6$ $P(\text{sechs}|\text{gerade}) = 1/3$

Satz: $P(A|B) = P(A \cap B) / P(B)$

A,B stochastisch unabhängig, falls $P(A \cap B) = P(A)P(B)$.

Wenn A,B stoch. unabh., dann $P(A|B) = P(A)$

Cashkurs Stochastik

Diskrete Zufallsvariablen = Wertebereich abzählbar

Beispiel: geometrische Verteilung

Wertebereich: Nat

Parameter $p < 1$

$$P(x=n) = (1-p)^{n-1}p$$

z.B.: Wie oft muss ich bis zur ersten 6 würfeln? $p = 1/6$

Eigenschaft: Gedächtnislosigkeit

$$P(x=n+m \mid x \geq m) = P(x=n)$$

Crashkurs Stochastik

Kontinuierliche Zufallsvariablen x

Wertebereich: reelle Zahlen

x heißt stetig, falls ex. f mit $F_x(a) = P(x < a) = \int_{-\infty}^a f(x) dx$.

($\rightarrow \int_{-\infty}^{\infty} f(x) dx = 1$)

f = Dichte

Beispiel: neg. Exponentialverteilung

$f(x) = \lambda e^{-\lambda x}$ für $x \geq 0$ und $f(x) = 0$ für $x < 0$

Crashkurs Stochastik

Eigenschaften der neg. Exponentialverteilung

Erwartungswert: $1/\lambda$

Gedächtnislosigkeit: $P(x > a + b \mid x > b) = P(x > a)$

Crashkurs Stochastik

Stochastischer Prozess

= Familie $\{x(t) \mid t \in T\}$ von Zufallsvariablen

$T = \text{Nat} \rightarrow$ diskrete Zeit

$T = [0, \infty) \rightarrow$ dichte Zeit

Wertebereich der Zufallsvariablen diskret „Kette“

Bei uns: Wertebereich = erreichbare Markierungen

homogener Markov-Prozess = gedächtnisloser stoch. Prozess:

$$\begin{aligned} P(x(t) = k \mid x(t_1) = k_1, \dots, x(t_n) = k_n) \quad (t_1 < \dots < t_n < t) \\ = P(x(t) = k \mid x(t_n) = k_n) \end{aligned}$$

Markov-Ketten

(Homogene) Markov-Kette = gedächtnisloser stoch. Prozess mit diskretem Wertebereich:

- $P(x(t) = k \mid x(t_1)=k_1, \dots, x(t_n)=k_n) \ (t_1 < \dots < t_n < t)$
 $= P(x(t) = k \mid x(t_n)=k_n)$
- $P(x(t+s) = k \mid x(t_n+s) = k_n) = P(x(t) = k \mid x(t_n)=k_n)$

Übergangswahrscheinlichkeiten

diskrete Markov-Ketten:

$$q_{ij}(s) = P(x(n+s) = j \mid x(n) = i) \rightarrow \text{Matrix } Q(s)$$

$$\text{Satz: } q_{ij}(s) = \sum_k q_{ik}(m) q_{kj}(s-m) \text{ für } m < s$$

$$\text{Also: } Q(s) = Q(m)Q(s-m)$$

$$\rightarrow Q(s) = Q(1)^s$$

Aufenthaltswahrscheinlichkeiten

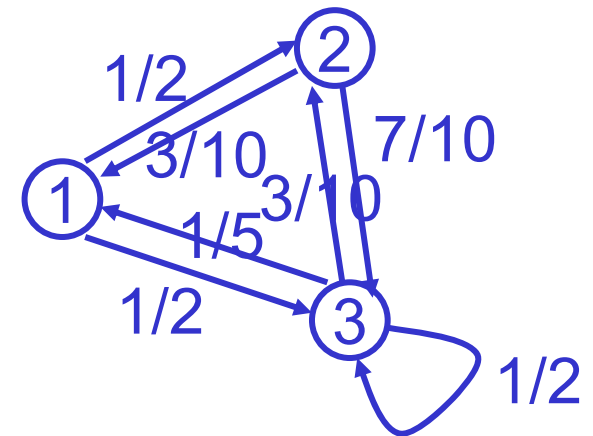
$$\pi_j(n) = P(x(n) = j)$$

$$\pi_j(n) = \sum_i \pi_i(0) q_{ij}(n)$$

$$\pi(n) = \pi(0) Q(1)^n$$

Beispiel

$$Q=Q(1) = \begin{pmatrix} 0 & 0.5 & 0.5 \\ 0.3 & 0 & 0.7 \\ 0.2 & 0.3 & 0.5 \end{pmatrix}$$



$$\pi(0) = (1, 0, 0) \quad \pi(1) = \pi(0)Q = (0, .5, .5) \quad \pi(2) = \pi(1)Q = (.25, .15, .6)$$

Stationäre Aufenthaltswahrscheinlichkeiten

Wahrscheinlichkeitsverteilung π für den Aufenthalt in den einzelnen Zuständen des Wertebereichs heißt stationär, falls

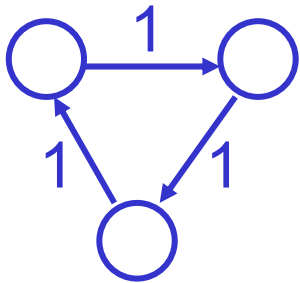
$$\pi Q = \pi \quad \text{und} \quad \sum_i \pi_i = 1$$

Falls Wertebereich stark zusammenhängend, endlich, aperiodisch

so ex. eindeutig bestimmte stationäre Verteilung und ist gleich $\lim_{n \rightarrow \infty} \pi(0) Q^n$, also $\lim_{n \rightarrow \infty} \pi(n)$ für beliebige Startverteilung $\pi(0)$. „Das eingeschwungene Verhalten“

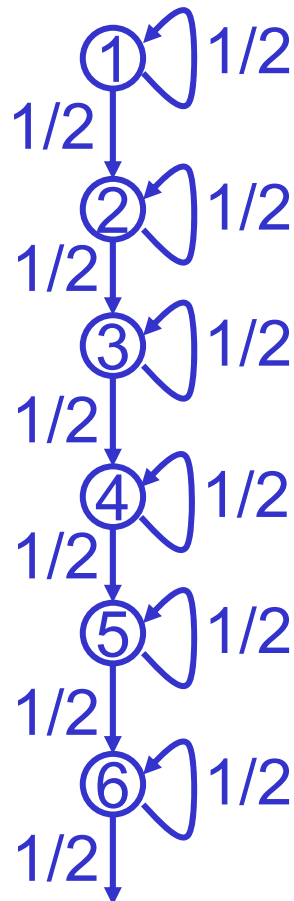
Gegenbeispiele

periodisch

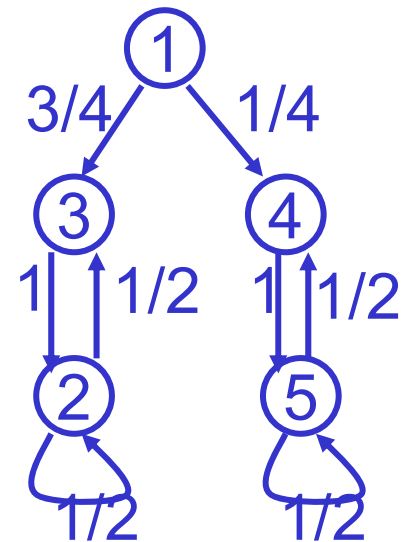


keine stat. Verteilung

unendlich



nicht stark zus.



$(0, 2/3, 1/3, 0, 0)$

$(0, 0, 0, 1/3, 2/3)$

mehrere stat. Verteilungen,
abh. von $\pi(0)$

Jetzt zusätzlich: diskrete Aufenthaltsdauern

τ_{ij} : Aufenthaltsdauer in i , falls nächster Zustand j ist

Folgt gedächtnisloser Wahrscheinlichkeitsverteilung
 $s_{ij}(m) = P(\tau_{ij}=m) = (1-a)^{m-1}a$ für einen Parameter a

Wartezeit in i : τ_i , unabhängig vom nächsten Zustand

$$P(\tau_i=m) = \sum_j q_{ij} s_{ij}(m)$$

Zeitlicher Prozessverlauf

Q beschreibt Schritte, sagt noch nicht über Zeit

$\phi_{ij}(n)$ = Wahrscheinlichkeit, dass Prozess nach n Zeittakten in j , falls jetzt in i (egal, wie lange schon, gedächtnislos!)

Zwei Möglichkeiten:

- (1) $i=j$ und i wird nicht vor Zeit n verlassen
- (2) i wird verlassen, j später betreten

Verweilwahrscheinlichkeit (1)

$$\delta_{ij} = 1, \text{ falls } i=j, \quad \delta_{ij}=0, \text{ sonst}$$

Verweilwahrscheinlichkeit

$$= \delta_{ij} P(\tau_i > n)$$

$$= \delta_{ij} \left(1 - \sum_{m=0}^n \sum_j q_{ij} s_{ij}(m) \right)$$

$$=: W_{ij}(n)$$

Übergangswahrscheinlichkeit (2)

Sei $\Phi = (\phi_{ij})$

i wird zu irgendeiner Zeit $m \leq n$ das erste Mal verlassen,
zu irgendeinem Zustand k .

$$\rightarrow \sum_{m=0}^n \sum_k q_{ik} P(\tau_{ik}=m) \phi_{kj}(n-m)$$

$$= \sum_{m=0}^n \sum_k q_{ik} s_{ik}(m) \phi_{kj}(n-m)$$

$$= \sum_{m=0}^n (Q \bullet S(m)) \Phi(n-m) \quad \bullet: \text{komponentenweise Mult.}$$

$$(1)+(2) \rightarrow \Phi(n) = W(n) + \sum_{m=0}^n (Q \bullet S(m)) \Phi(n-m)$$

(rekursiv berechenbar)

Jetzt: kontinuierliche Zeiten

Analog zu diskreten Zeiten, nur:

statt geometrischer Verteilung

neg. exp. Verteilung

1-Schritt-
Übergangs-

Infinitesimale

Übergangsmatrix Q

wahrscheinlichkeiten

$$\lim_{\varepsilon \rightarrow 0} P(x(t+\varepsilon)=j | x(t)=i)$$

$$\pi(n) = \pi(0) Q^n$$

$$\pi(t) = \pi(0) e^{\int_0^t Q(u) du}$$

eingeschwungen:

$$\pi Q = \pi$$

$$\pi Q = 0$$

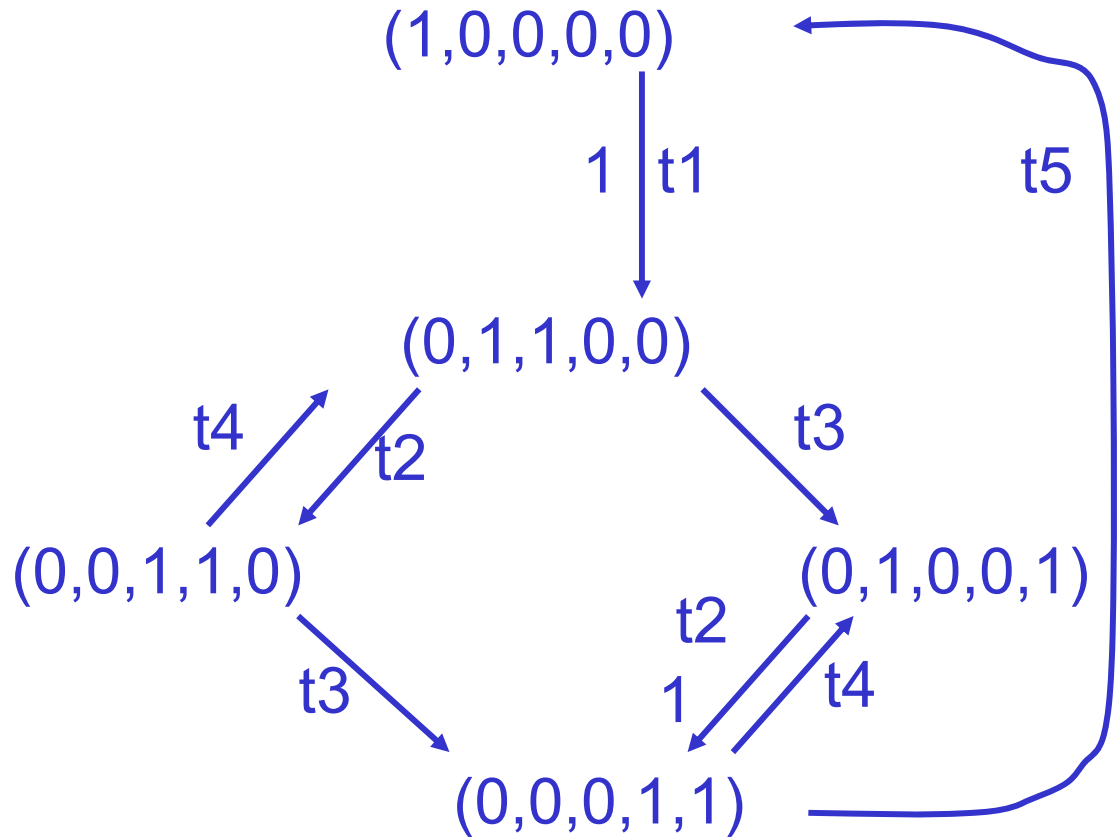
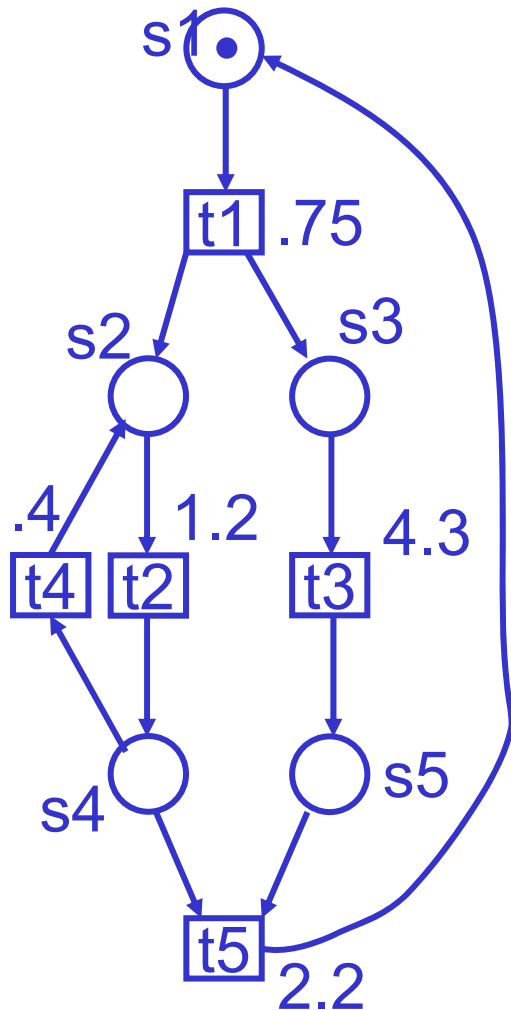
Stochastische Petrinetze

$[S, T, F, W, m_0, \lambda]$ $\lambda: T \rightarrow \text{Real}$ $\lambda(t)$ = Schaltrate von t
= Parameter einer
neg.exp. Verteilung,
die die Schaltver-
zögerung von t
beschreibt

Motivation für neg.exp. Verteilung

1. gedächtnislos
2. Markov-Theorie anwendbar
3. viele Verteilungen mittels neg.exp. Verteilung approximierbar

Beispiel



Schaltregel

In Markierung m , wählt jede Transition t_i eine Schaltverzögerung x_{ti} entsprechend ihrer neg. exp. Verteilung

Die Transition mit der niedrigsten Verzögerung schaltet, ihre Verzögerung bestimmt die Aufenthaltszeit in m

Übersetzung in Markov-Prozess

Wertebereich = Menge der erreichbaren Markierungen

Seien $t_1 \dots t_n$ aktiviert in m

$P(t_i \text{ hat geringste Schaltverzögerung}) =$

$$P(x_{t_i} < x_{t_1} \ \& \ \dots \ \& \ x_{t_i} < x_{t_n}) = \lambda(t_i) / (\lambda(t_1) + \dots + \lambda(t_n))$$

mittlere Verweildauer in m : $1 / (\lambda(t_1) + \dots + \lambda(t_n))$

Verweildauer neg. exp.-verteilt

→ haben Markov-Prozess mit kontinuierlicher Zeit

→ Aufenthaltswahrscheinlichkeiten, Durchsatz

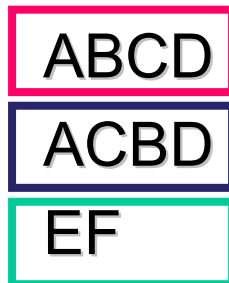
Process Mining

Anliegen

- Modelle entstehen oft losgelöst von den tatsächlichen Prozessen
- Ereignisprotokolle (event logs) stehen vielfach als Abbild der tatsächlichen Abläufe zur Verfügung
- Idee: Aus den Logs automatisch die zugrundeliegenden Prozesse rekonstruieren
- Ziele
 - realistischere Prozesse
 - Vergleich mit den „ausgedachten“ Prozessen

Logs

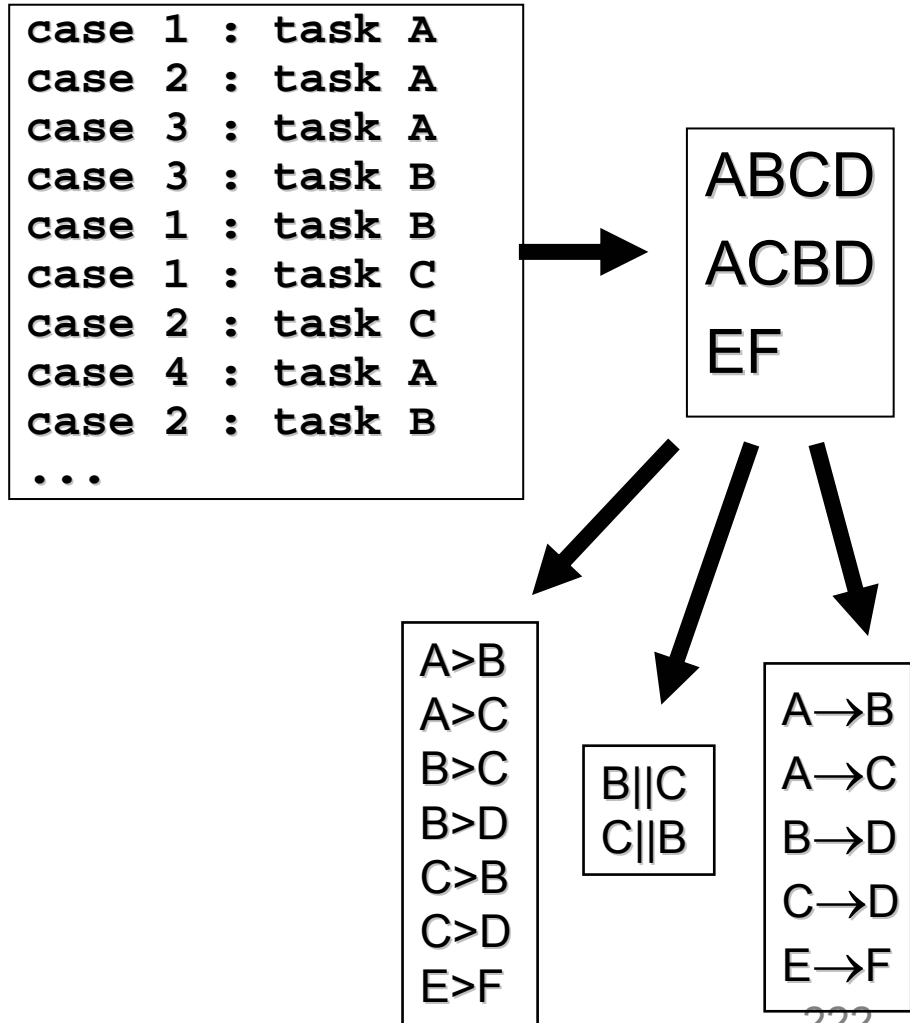
- Vor: kein Rauschen
- Log enthält normalerweise: case id und task id's
- zusätzlich oft: Ereignistyp, Zeit, Ressourcen, Daten
- Im Beispiel: 4 Sequenzen



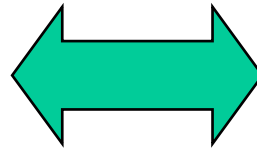
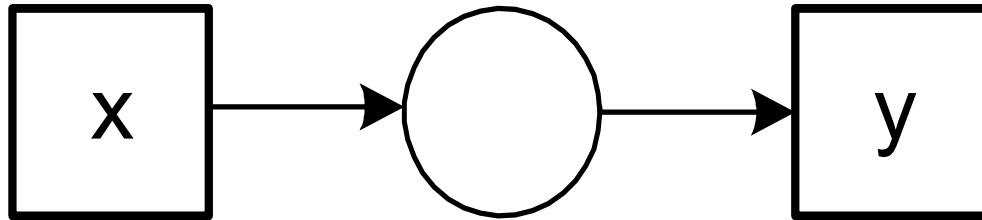
case 1	:	task A	←
case 2	:	task A	←
case 3	:	task A	←
case 3	:	task B	←
case 1	:	task B	←
case 1	:	task C	←
case 2	:	task C	←
case 4	:	task A	←
case 2	:	task B	←
case 2	:	task D	←
case 5	:	task E	←
case 4	:	task C	←
case 1	:	task D	←
case 3	:	task C	←
case 3	:	task D	←
case 4	:	task B	←
case 5	:	task F	←
case 4	:	task D	←

Erster Algorithmus: α – Relationen $>, \rightarrow, ||, \#$

- **Direkte Folge:** $x > y$ gdw x in einem Fall direkt von y gefolgt.
- **Kausal:** $x \rightarrow y$ gdw $x > y$ und nicht $y > x$.
- **Parallel:** $x || y$ gdw $x > y$ und $y > x$
- **Unbezogen:** $x \# y$ gdw weder $x > y$ noch $y > x$.

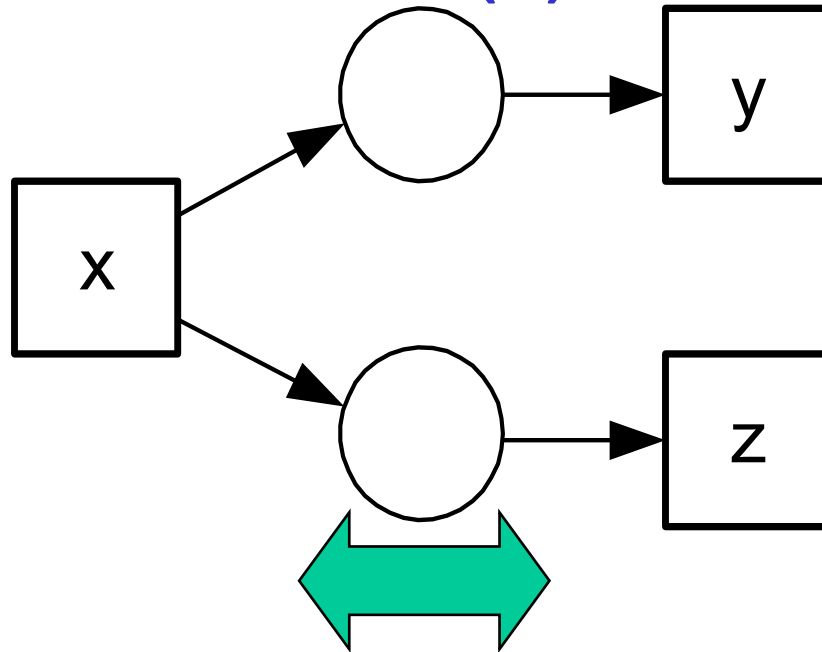


Idee (1)



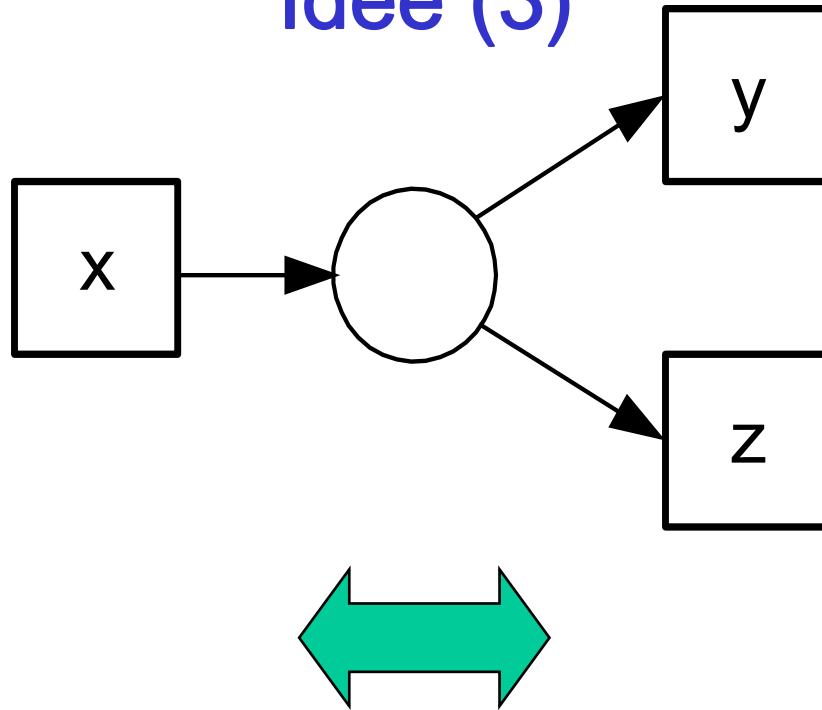
$x \rightarrow y$

Idee (2)



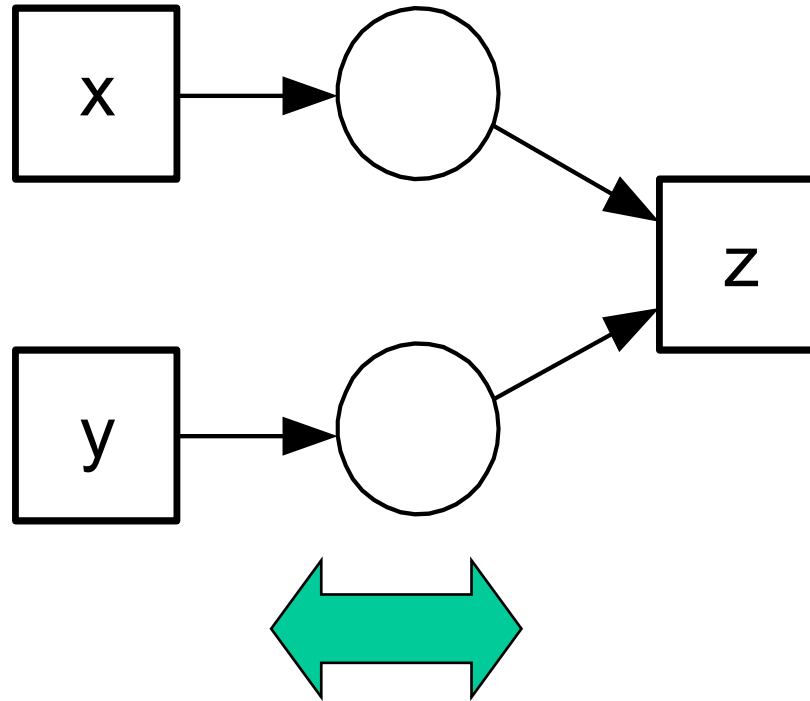
$x \rightarrow y$, $x \rightarrow z$, und $y \parallel z$

Idee (3)



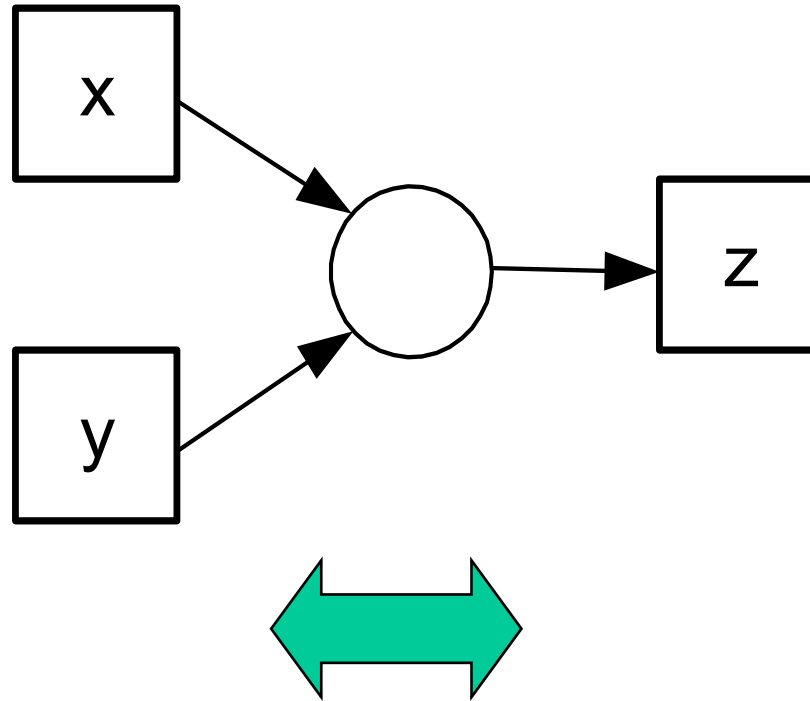
$x \rightarrow y$, $x \rightarrow z$, und $y \# z$

Idee (4)



$x \rightarrow z$, $y \rightarrow z$, und $x \parallel y$

Idee (5)



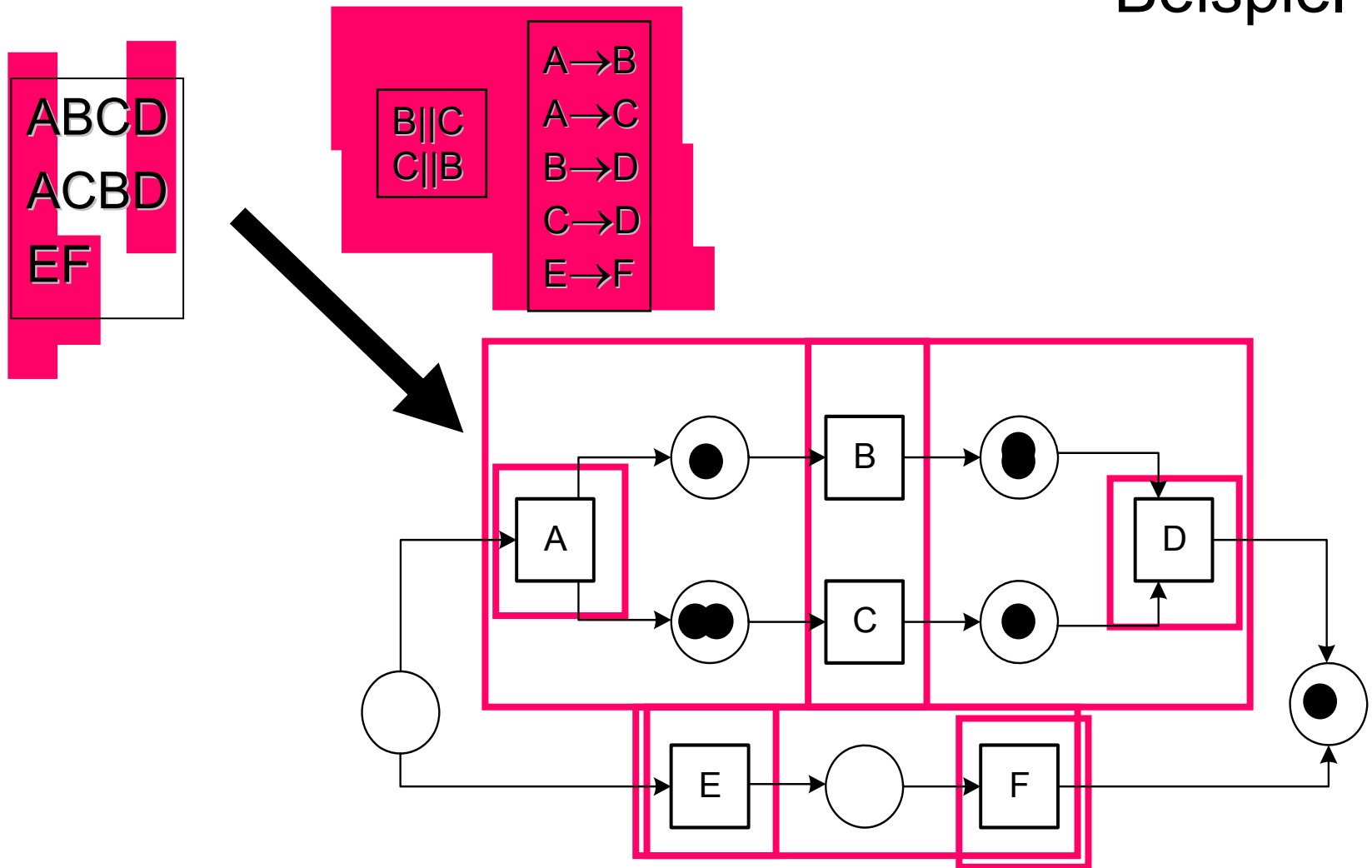
$x \rightarrow z$, $y \rightarrow z$, und $x \# y$

α -Algorithmus

Sei W log über T . $\alpha(W)$ ist wie folgt definiert.

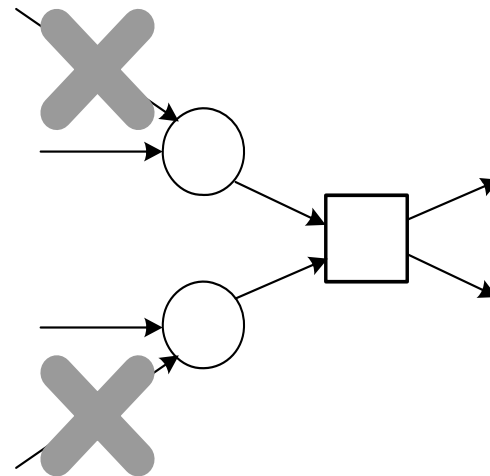
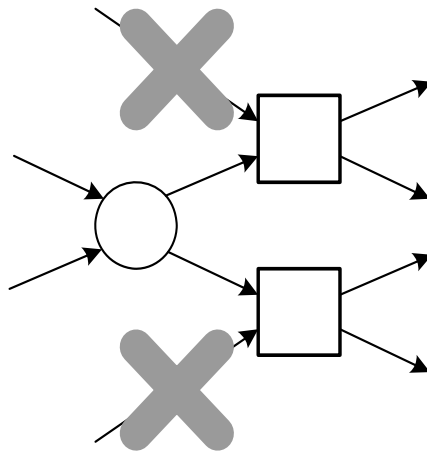
1. $T_W = \{ t \in T \mid \exists_{\sigma \in W} t \in \sigma \},$
2. $T_I = \{ t \in T \mid \exists_{\sigma \in W} t = \text{first}(\sigma) \},$
3. $T_O = \{ t \in T \mid \exists_{\sigma \in W} t = \text{last}(\sigma) \},$
4. $X_W = \{ (A, B) \mid A \subseteq T_W \wedge B \subseteq T_W \wedge \forall_{a \in A} \forall_{b \in B} a \rightarrow_W b \wedge \forall_{a_1, a_2 \in A} a_1 \#_W a_2 \wedge \forall_{b_1, b_2 \in B} b_1 \#_W b_2 \},$
5. $Y_W = \{ (A, B) \in X \mid \forall_{(A', B') \in X} A \subseteq A' \wedge B \subseteq B' \Rightarrow (A, B) = (A', B') \},$
6. $P_W = \{ p_{(A, B)} \mid (A, B) \in Y_W \} \cup \{ i_W, o_W \},$
7. $F_W = \{ (a, p_{(A, B)}) \mid (A, B) \in Y_W \wedge a \in A \} \cup \{ (p_{(A, B)}, b) \mid (A, B) \in Y_W \wedge b \in B \} \cup \{ (i_W, t) \mid t \in T_I \} \cup \{ (t, o_W) \mid t \in T_O \}, \text{ and}$
8. $\alpha(W) = (P_W, T_W, F_W).$

Beispiel



Bewertung

- Falls Log vollständig ist bzgl. “>”, kann eine Klasse von FC-Netzen korrekt erkannt werden
- *Structured Workflow Nets* (SWF-nets)
 - keine impliziten Stellen
 - kein Vorkommen von:

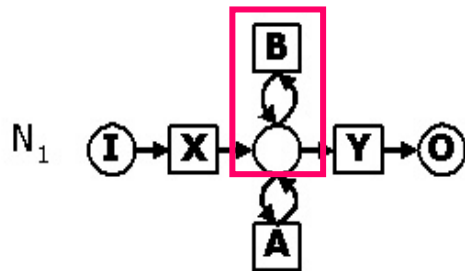


Beschränkungen

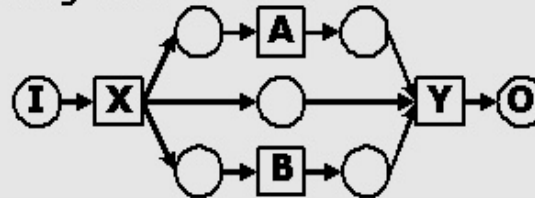
- Problematic constructs
 - Short loops
 - Synchronization of OR-join places
 - Duplicate Tasks
 - Implicit Places
 - Non-free Choice

Kurze Schleifen

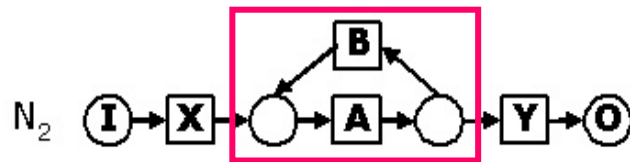
$B > B$ und nicht $B > B$ impliziert $B \rightarrow B$ (unmöglich!)



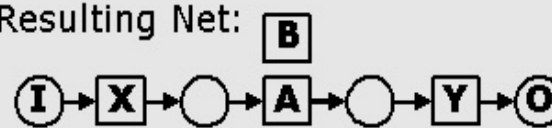
Resulting Net:



Länge 1



Resulting Net:



Länge 2

$A > B$ und $B > A$ impliziert $A || B$ und $B || A$ anstelle $A \rightarrow B$ and $B \rightarrow A$

Mining auf der Basis der Regionentheorie

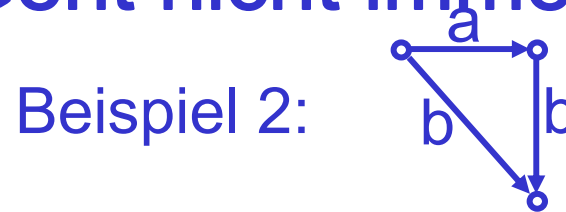
- Regionentheorie konstruiert ein Petrinetz aus einem Transitionssystem („seinem Erreichbarkeitsgraph“)
- Anwendung im Mining: Logs in irgendein Transitionssystem übersetzen (z.B. Baum), Netz konstruieren, ggf. iterativ verbessern oder approximieren

Problemstellung

- geg: gerichteter, endlicher, von einem Knoten v_0 zusammenhängender, an den Bögen beschrifteter Graph $G = [V, E, b]$ mit Beschriftungen aus einer Menge T .
- ges: Petrinetz, dessen Transitionsmenge T ist und dessen Erreichbarkeitsgraph isomorph zu G ist.
(jetzt insbesondere: Bogen e mit $e(b) = t$ wird realisiert durch Schalten von t im Petrinetz).

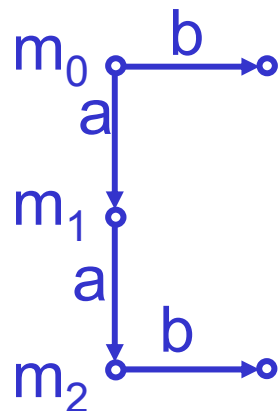
1. Frage: geht das immer?

Antwort: Geht nicht immer



a ändert Markierung nicht, also sind Quelle und Ziel von a nicht verschieden.

Beispiel 3



Sei s nicht ausreichend markiert für b in m_1 .

- (1) $m_0(s) \geq W([s, b])$
- (2) $W([s, b]) > m_0(s) + \underline{a}(s)$
- (3) $m_0(s) + 2 \underline{a}(s) \geq W([s, b])$

$$1+2 \rightarrow \underline{a}(s) < 0$$

$$2+3 \rightarrow \underline{a}(s) > 0 \quad \text{Wid.!}$$

geht nicht immer...

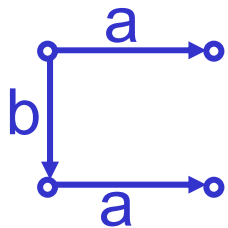
.... mit sicheren Netzen:



Beispiel:



... sicheren, schleifenfreien Netzen:



a entzieht b Aktivierung, also gibt es geteilten Vorplatz s. Dann kann aber a nicht nach b, weil dabei 2 Marken von s abzuziehen wären.

Jetzt zur Lösung

Einschränkung (zur Vereinfachung):

suchen sicheres Petrinetz, das geg. Graph realisiert.

Varianten: mit und ohne Schleifen.

Ansatz: müssen im Graph Stellen wiederfinden, d.h.,
die

Zustandsmengen, wo eine Stelle markiert bzw.
unmarkiert

ist.

Vorteil von *sicher*. Stelle unmarkiert =
Komplementärstelle

markiert (oBdA: Netz voll durchkomplementiert).

→ suchen Zustandsmengen, bei denen Stelle markiert
ist.

Stellen im Erreichbarkeitsgraph

Sei G Erreichbarkeitsgraph eines sicheren Petrinetzes,
 s eine Stelle und $M = \{m \mid m \text{ Knoten in } G, m(s) = 1\}$

Sei $m [t > m'$.

Wenn $t \in \bullet s, t \in s \bullet$, so $m \in M, m' \in M$

Wenn $t \in \bullet s, t \notin s \bullet$, so $m \notin M, m' \in M$

Wenn $t \notin \bullet s, t \in s \bullet$, so $m \in M, m' \notin M$

Wenn $t \notin \bullet s, t \notin s \bullet$, so $m \in M, m' \in M$
oder $m \notin M, m' \notin M$

Regionen

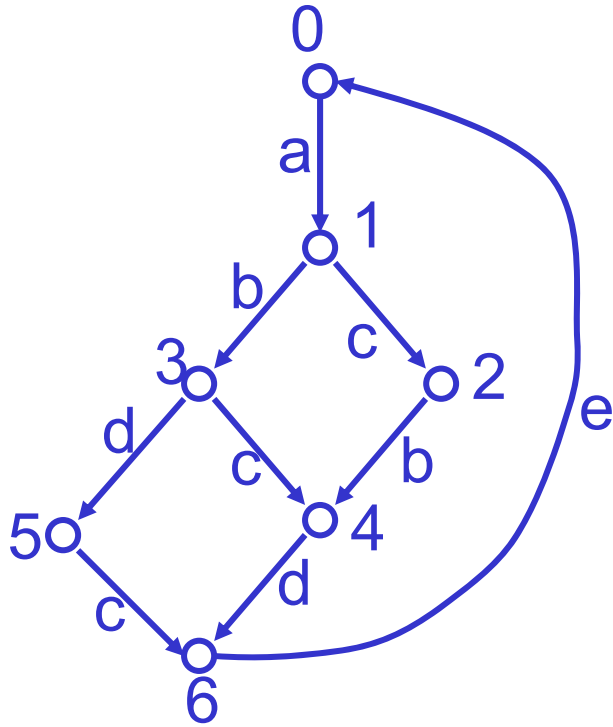
Sei R Knotenmenge. Bogen $[v, v']$

- betritt R , falls $v \notin R, v' \in R$
- verlässt R , falls $v \in R, v' \notin R$

Knotenmenge R heißt Region, falls für alle t :

- wenn ein mit t beschrifteter Bogen R betritt, betreten alle mit t beschrifteten Bögen R
- wenn ein mit t beschrifteter Bogen R verlässt, verlassen alle mit t beschrifteten Bögen R

Beispiel



\emptyset

12

34

56

135

246

1234

1256

3456

123456

0

012

034

056

0135

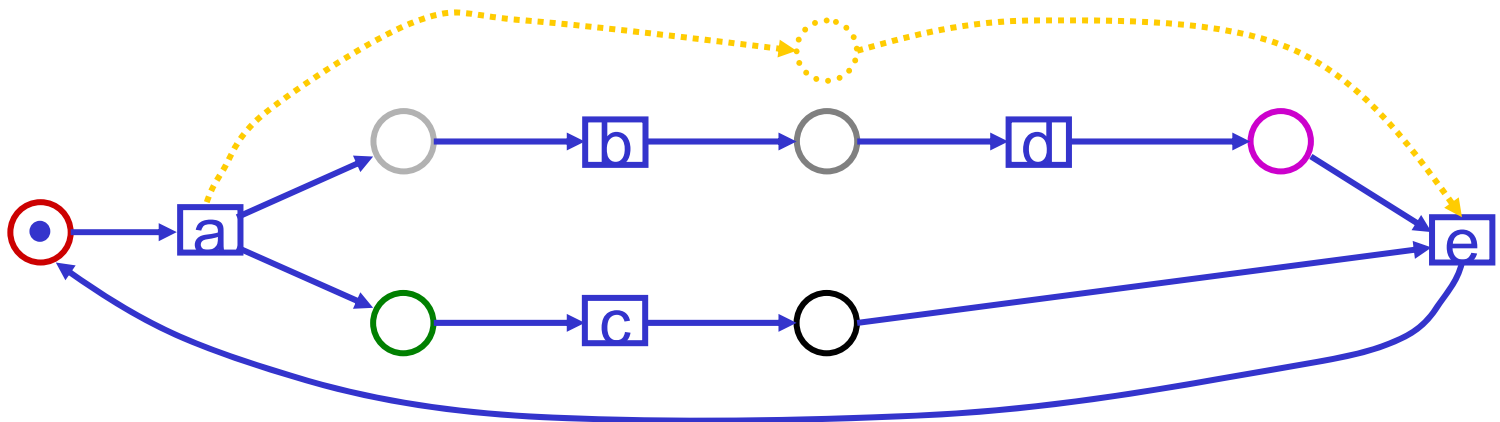
0246

01234

01256

03456

0123456



Das Regionennetz

Sei $G = [V, E, b]$ gerichteter Graph, R Menge der Regionen.

Dann ist $N_G = [R, T, F, m_0]$ mit

$[R, t] \in F$ gdw. t verlässt R

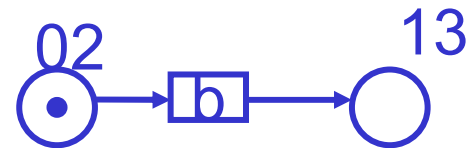
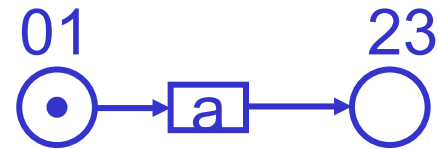
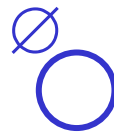
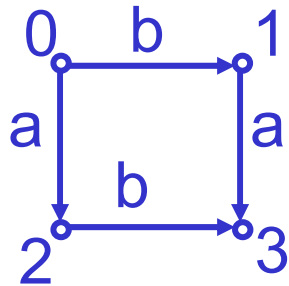
$[t, R] \in F$ gdw. t betritt R

$m_0(R) = 1$, falls $v_0 \in R$,

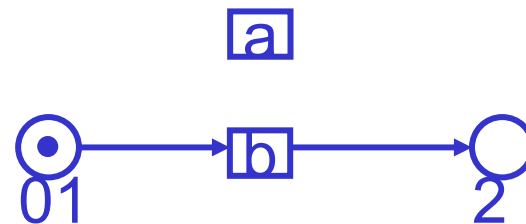
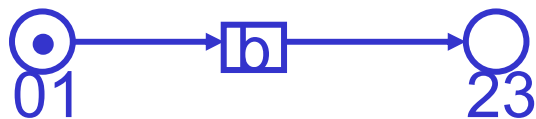
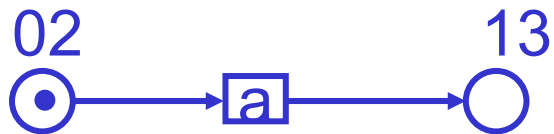
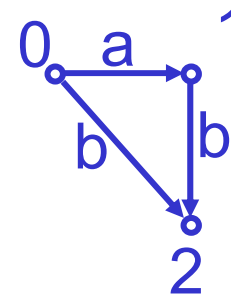
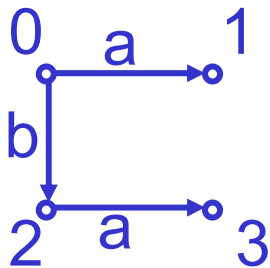
$= 0$, sonst

das Regionennetz zu G .

Bsp:



Weitere Beispiele



Satz

Wenn G isomorph zum Erreichbarkeitsgraph eines sicheren, schleifenfreien Petrinetzes ist, dann ist G isomorph (\sim) zum Erreichbarkeitsgraph von N_G .

Zwischenfazit

Prozedur:

Geg.: Graph G .

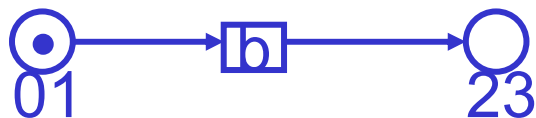
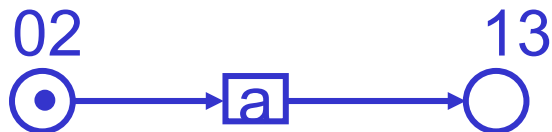
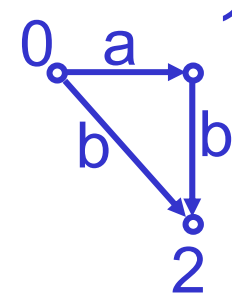
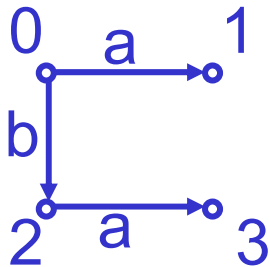
1. Berechne Regionen
2. Bilde Regionennetz
3. Berechne dessen Erreichbarkeitsgraph
4. Vergleiche mit G

isomorph \rightarrow Ergebnis

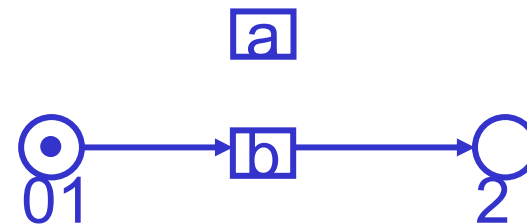
nicht isomorph \rightarrow Es gibt kein sicheres schleifenfreies
Netz

Wollen verstehen, wann, und warum es nicht geht...

Beispiele, wo es nicht ging



Vorbedingung von b
nicht exakt widerspiegelt



ϕ kein Isomorphismus

So geht es:

Damit ϕ Isomorphismus wird:

Zustandsseparation:

Zu je zwei Knoten v, v' gibt es eine Region R mit
 $v \in R, v' \notin R$

Damit Vorbedingungen exakt wiedergespiegelt werden:

Ereignisseparation:

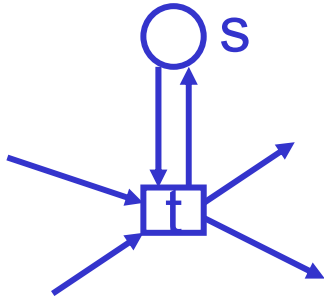
Zu jedem t ist

$$\bigcap_{t \text{ verlässt } R} R = \{v \mid \text{ex. } v' \text{ mit } [v, v'] \in E, b([v, v']) = t\}$$

Charakterisierung synthetisierbarer Graphen

Satz: Zu einem Graph G gibt es genau dann ein sicheres schleifenfreies Netz N mit $EG_N \sim G$, wenn die Eigenschaften Zustands- und Ereignisseparation in G erfüllt sind.

Der Fall mit Schleifen



Wie verhalten sich t und R_s ?

Alle t -Bögen haben Quelle und Ziel in R_s !

Def.: t heißt intern zu Region R , falls für alle $[v, v']$ mit $b([v, v']) = t$ gilt: $v, v' \in R$.

→ neue Konstruktion:

Regionen wie bisher. Regionennetz wie bisher, aber zusätzlich:
Für jede Region R , zu der t intern ist, ziehe Schleife zwischen t und R .

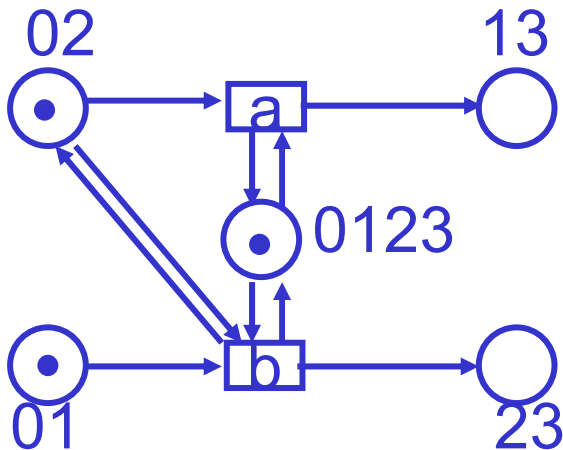
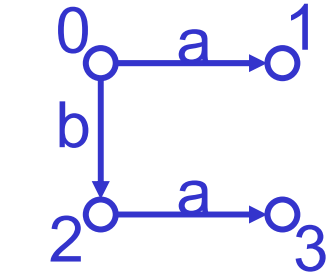
Beispiel

Warum klappt es?

Durch die Schleifen ändert sich das Kriterium der Ereignis-separation:

$$\bigcap_t \text{verlässt } R \text{ oder ist intern zu } R \quad R$$

$$= \{v \mid \text{ex. } v' \text{ mit } [v, v'] \in E, b([v, v']) = t\}$$



Hurra, es klappt.

Minimale Regionen

Ziel: Nicht so viele sinnlose Stellen einbauen

Region R ist minimal, falls keine nichtleere Teilmenge von R Region ist.

Satz: Jede nichtleere Region ist disjunkte Vereinigung minimaler Regionen.

Folgt aus: Wenn $R_1 \subseteq R_2$ Regionen sind, so auch $R_2 \setminus R_1$.

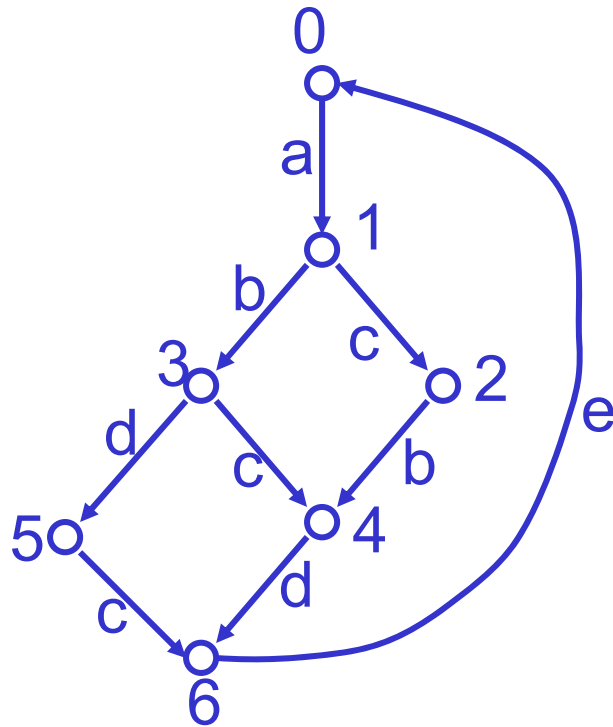
Nichtminimale Regionen können weggelassen werden

Sei R nicht minimal im Vorbereich von t . Also: t verlässt R .

Dann gibt es auch eine min. Region $R' \subseteq R$, die von t verlassen wird. Wenn R' markiert, dann auch R markiert,

weil $R' \subseteq R$. Also ist R redundant.

Beispiel



\emptyset

0

12

012

34

034

56

056

135

0135

246

0246

1234

01234

1256

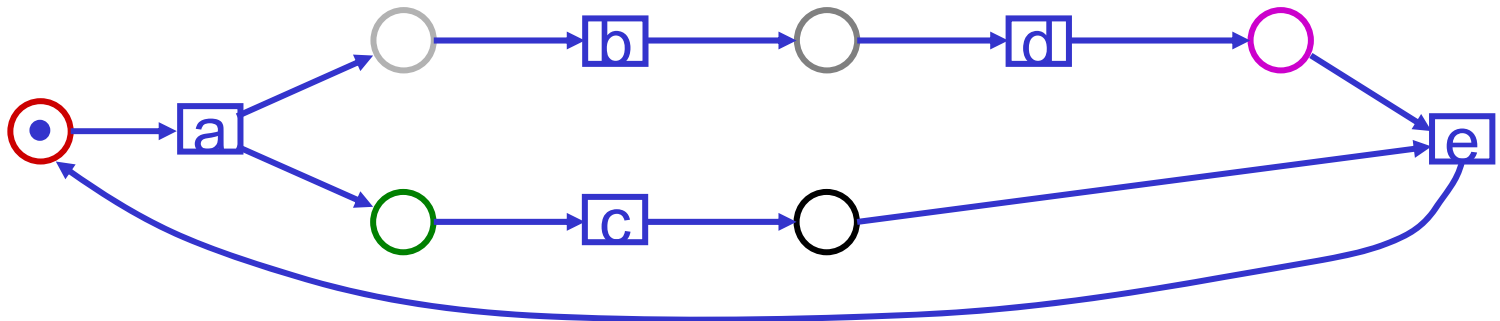
01256

3456

03456

123456

0123456



Anwendung im Mining: z.B. Approximation

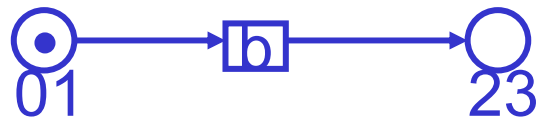
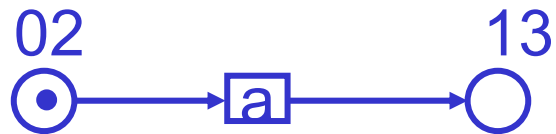
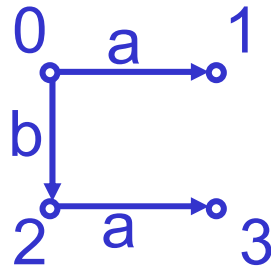
- Problem der Regionentheorie: Logs sind normalerweise unvollständig, synthetisiertes Netz spiegelt aber EXAKT das gegebene Transitionssystem wider.

→ sollten approximieren

Idee: Netz vereinfachen, dabei keine Sequenzen deaktivieren (ggf., aber welche aktivieren)

Mittel: Kriterium der Ereignisseparation weglassen.

Beispiel



Sprachbasierte Regionentheorie

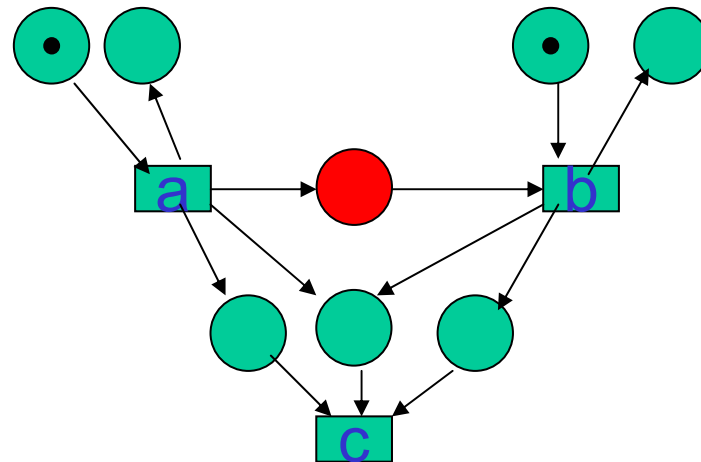
- Geg.: Sprache
- Ges.: Petrinetz, das diese Sprache realisiert
- Idee:
 - Transitionen gegeben (vorkommende Buchstaben)
 - Plätze unterteilbar in gute und schlechte
 - gut: Elemente der Sprache werden nicht behindert
 - schlecht: Elemente der Sprache werden behindert
 - Nimm einfach ALLE guten Plätze mit jeweils eindeutig bestimmbarer minimaler Anfangsmarkierung

Beispiel

- geg: abc,bac

gute Plätze (Auswahl)

schlechte Plätze (Auswahl)



Problem: zu viele Plätze (mit Vielfachheiten: unendlich viele)

Lösung: 1. Lasse Plätze weg, von denen klar ist, dass sie nicht gebraucht werden

2. Lasse weitere Plätze weg (Überapproximation)


Flexibilisierung mit ADEPT

Hintergrund


Stationsarbeitsplatz - Stationsarzt
 Datei Hilfe

Stationsübersicht


Zimmer 57




Fr. Schiebe
frakt. Abrasio
Aus OP zurück




Fr. Meyer
Interruptio
In OP




Fr. Heinrich
IVF
In OP bringen




Zimmer 58



Fr. Apfel
frakt. Abrasio





Maßnahme anordnen
 OP-Plan
 Umschalten zu Ambulanz
 Notfallaufnahme

Anstehende Aufgaben.

Patientin	Tätigkeit	Bemerkung	Ausführen	Aktualisieren
Fr. Schiebe	OP-Bericht erstellen			
Fr. Schiebe	Klinische Entlassung			
Fr. Förster	Überprüfung der Befunde	ängstliche Patientin		
Fr. Hansen	Überprüfung der Befunde			

Einbestellungsplan GYN IV

Freitag, 18. Juli 1997

Patientin	Diagnose-OP	Tel.Nr.
Fr. Mayer (Abbruch)	Interruptio	07561-22456
Fr. Dampf (Überprüfung der Befunde)	cg/op Hysteroskopie	
Fr. Heinrich (Abbruch in Ausführung)	IVF	
Fr. Apfel (Stationäre Aufnahme)	frakt. Abrasio	07551-22456
Fr. Frontze (Überprüfung der Befunde)	IVF	0731 3051
Fr. Schiebe (Rückmeldung)	frakt. Abrasio	

Tag Woche Monat

← →

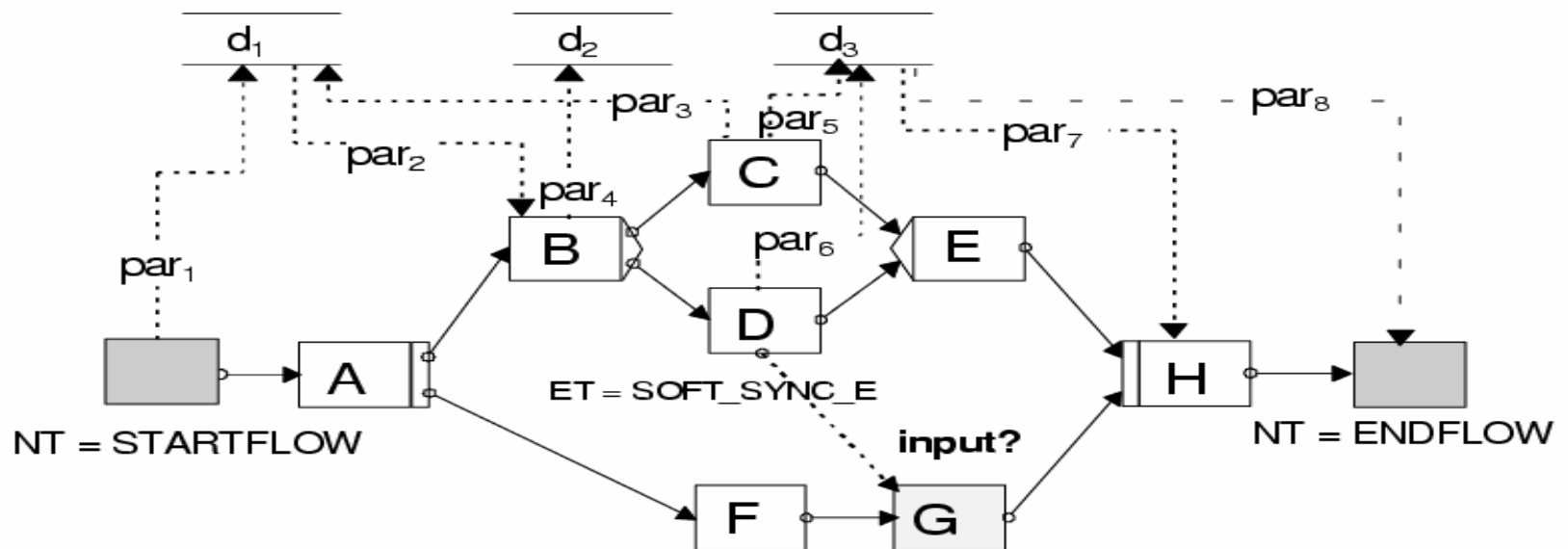
neuer Termin Termin verschieben

Termin sperren Termin löschen Patientendaten

ADEPT Prozessmodell

Prozess als gerichteter, strukturierter Graph, d.h.

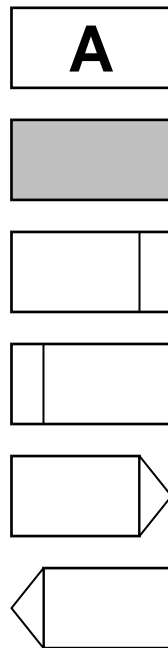
- typisierte Knoten (Aktivitäten, Datenobjekte, Strukturelemente)
- typisierte Knoten (Kontrol- & Datenfluss, Synchronisation, Loop,



Grundelemente

Knoten:

- Aktivität
- Strukturelement
- AND-Split
- AND-Join
- OR-Split
- OR-Join









Strukturelemente:

- StartFlow- & EndFlow-Knoten
- StartLoop- & EndLoop-Knoten
- Failure- & Restart-Knoten

Grundelemente

Kanten:

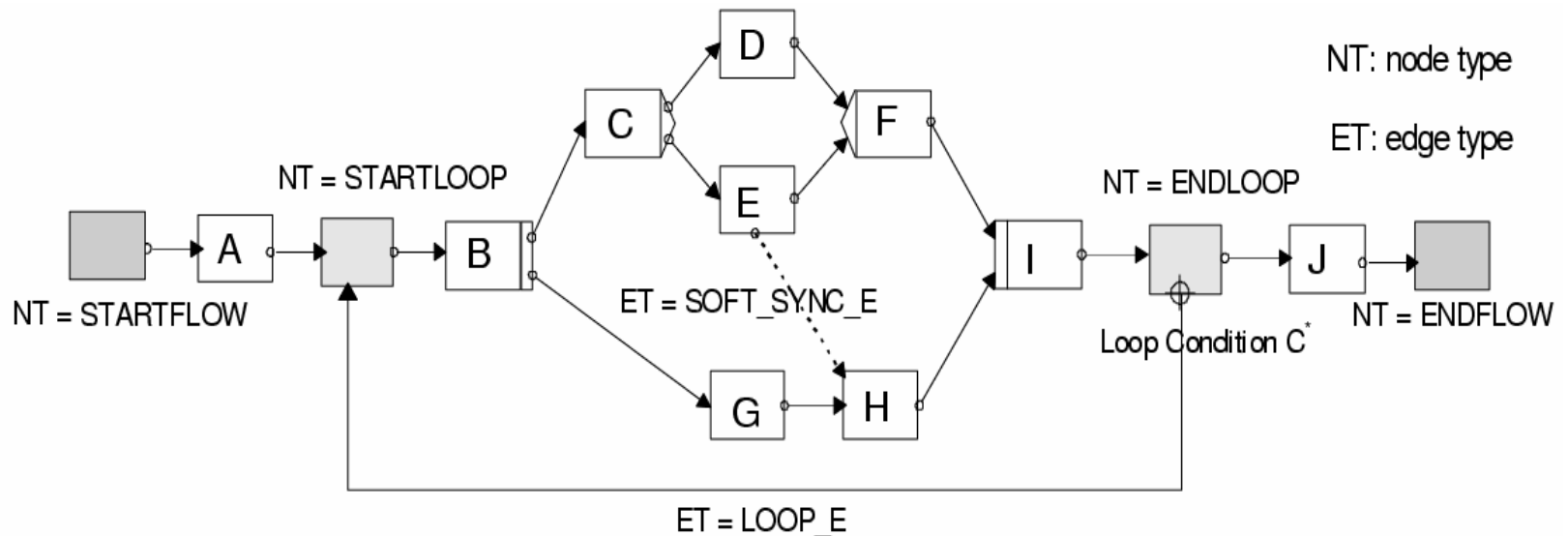
- Kontrollfluss 
- LOOP 
- Fehler 
- Synchronisation 

- Datenfluss 

Kantentypen:

- Control_E
- Loop_E
- Failure_E
- Strict_Sync_E
- Soft_Sync_E
- Data_E

Start und Ende

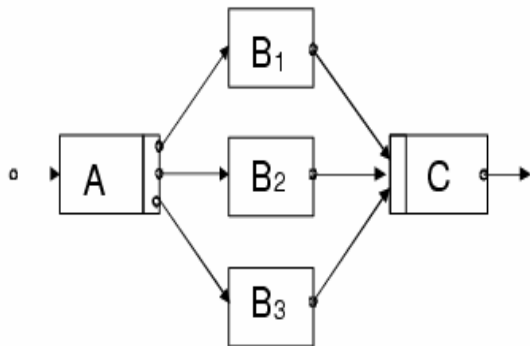
- Jeder Prozess hat genau einen Start- und genau einen End-Knoten
- Jeder Knoten liegt auf einem Pfad zwischen diesen beiden



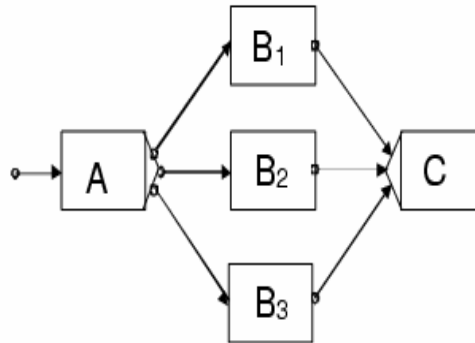
Verknüpfung von Aktivitäten

- Sequentielle Ausführung wird durch Kontrollflusskanten abgebildet
- Verzweigung nur mit Split und Join
- Verwendung dreier Grundmuster

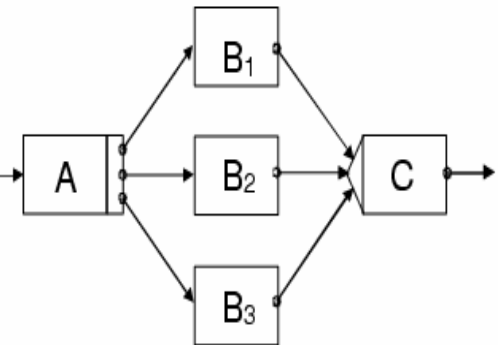
Symmetrische Parallelisierung



Symmetrische Alternative

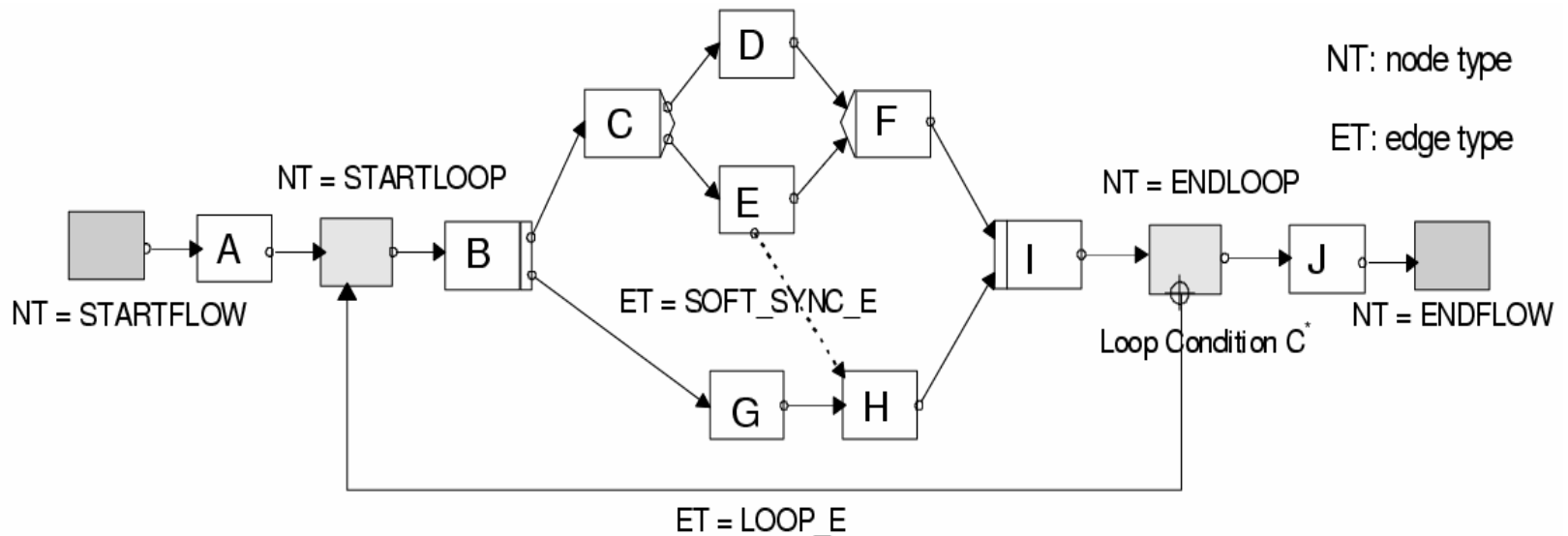


Parallelisierung mit Auswahl



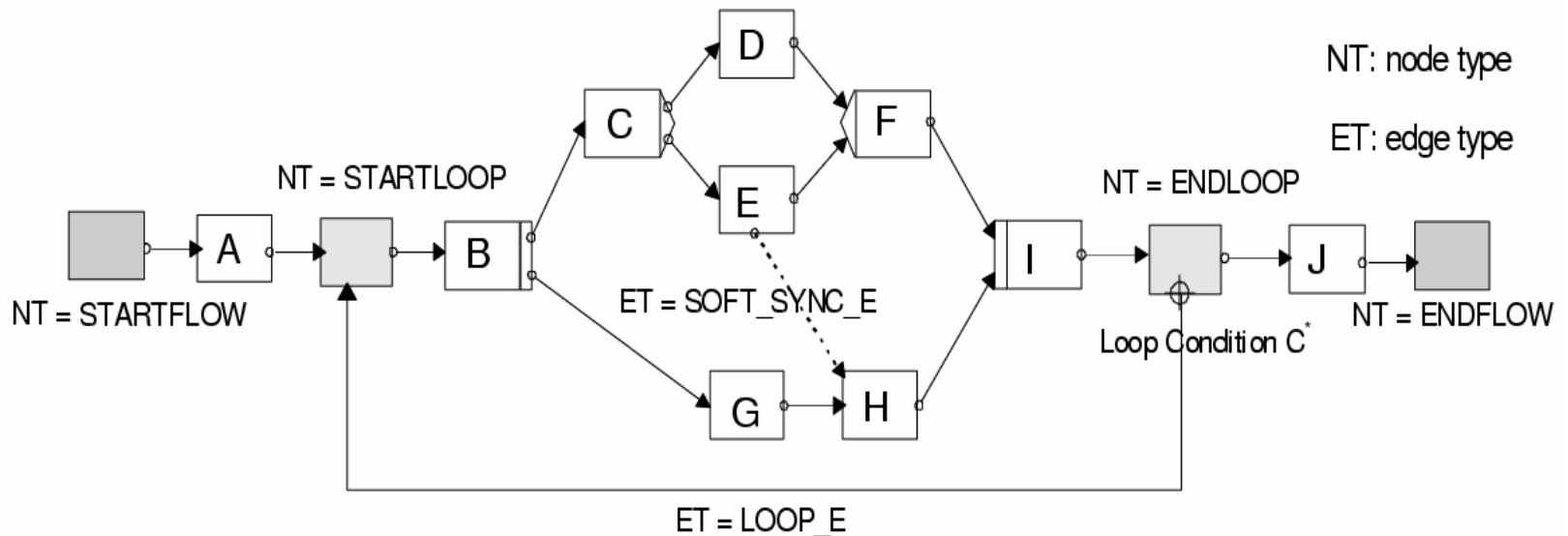
Schleifen

- Kontrollflussskanten zyklensfrei
- LOOP zwischen StartLoop- & EndLoop-Knoten
- LOOP umfasst abgeschlossenen Block



Synchronisation

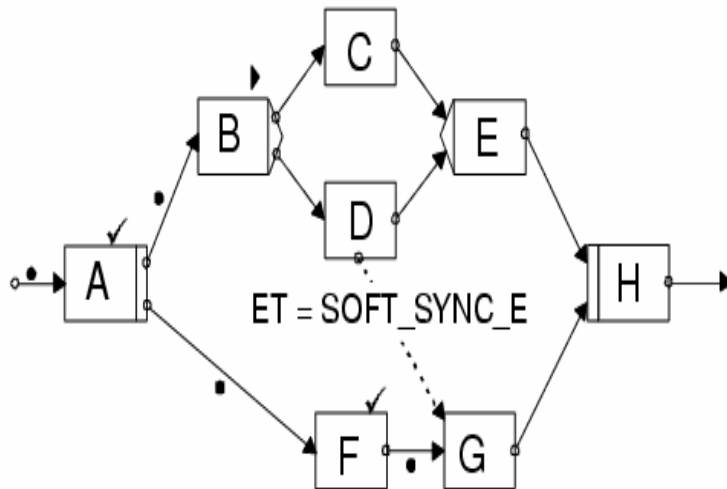
- Synchronisation nur zwischen parallelen Aktivitäten
- Synchronisation darf LOOP nicht verlassen



Die Dynamik

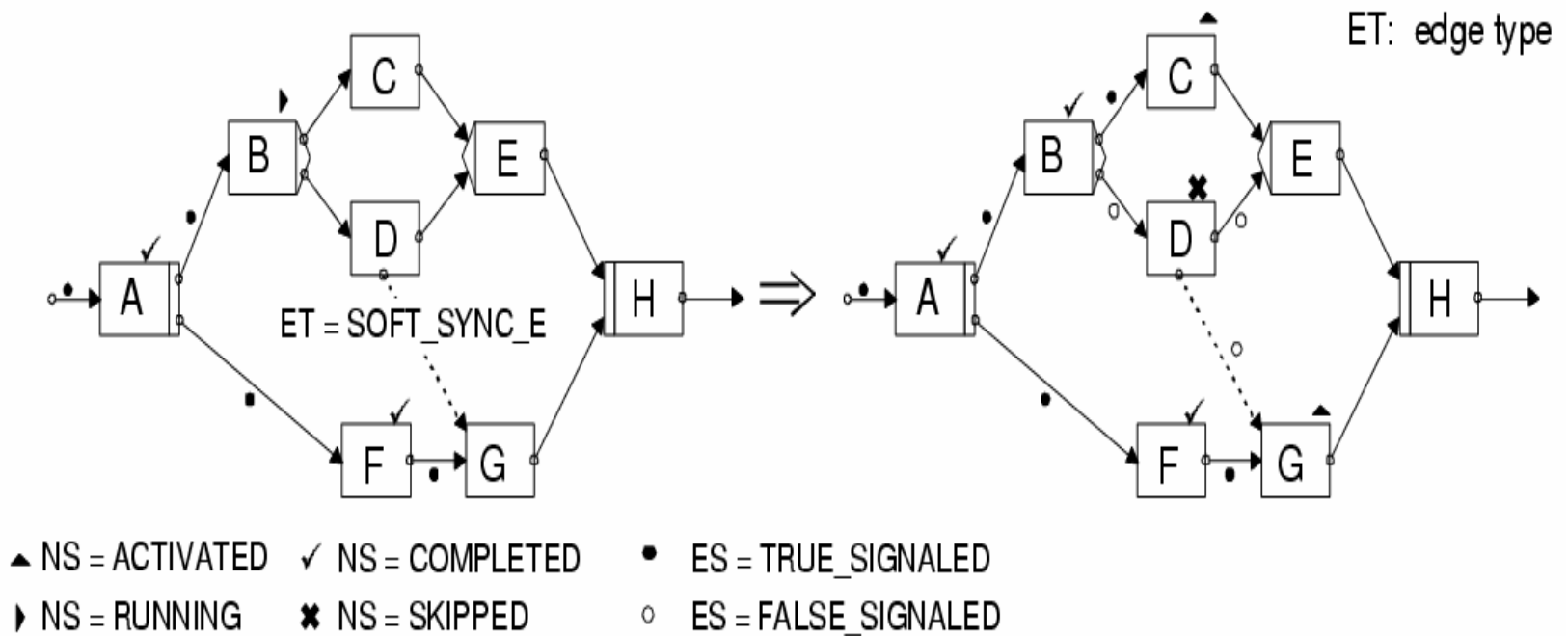
- Zustände eine Aktion
 - NotActivated
 - Activated
 - Running
 - Completed
 - Failed
 - Skipped
- Zustände eine Kante
 - NotSignaled
 - TrueSignaled
 - FalseSignaled

Auswerten einer Sync-Kante

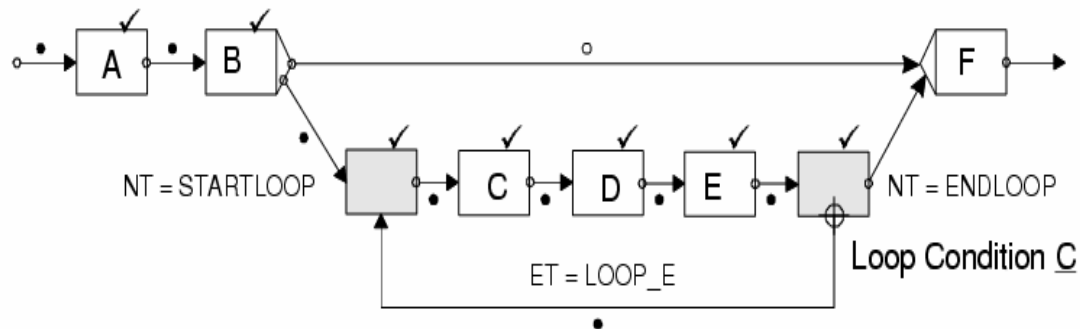


- | | | |
|------------------|------------------|---------------------|
| ▲ NS = ACTIVATED | ✓ NS = COMPLETED | • ES = TRUE_SIGNED |
| ▶ NS = RUNNING | ✕ NS = SKIPPED | ○ ES = FALSE_SIGNED |

Auswerten einer Sync-Kante



Rücksetzen eines LOOP-Blocks



▲ NS = ACTIVATED

✓ NS = COMPLETED

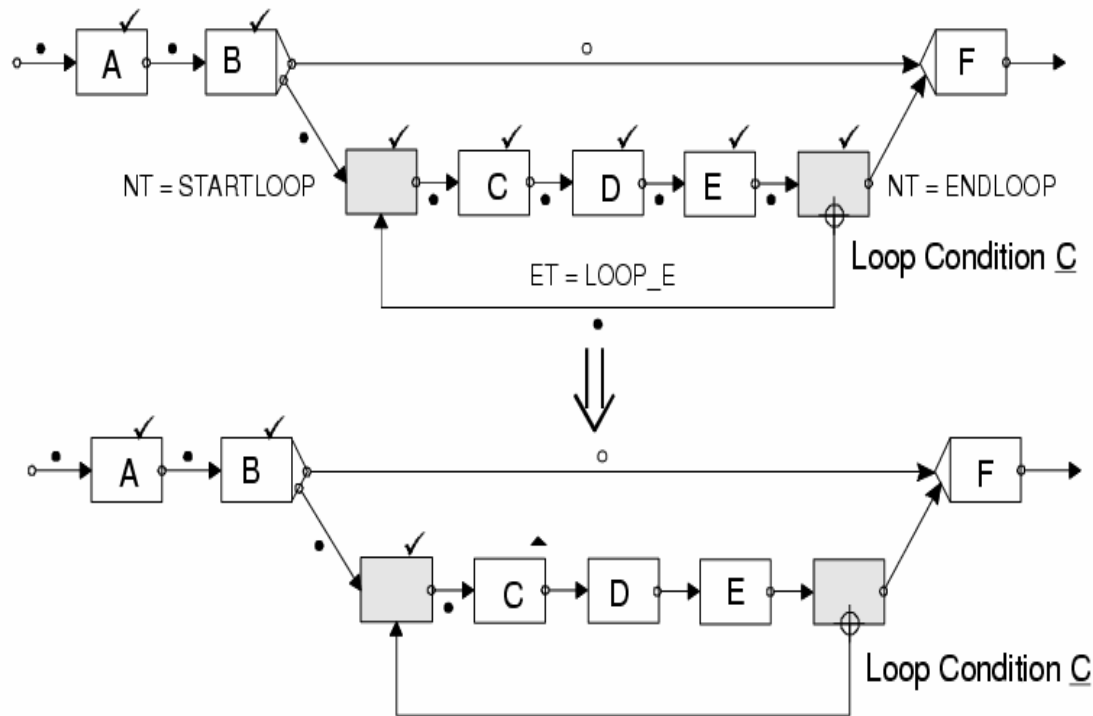
• ES = TRUE_SIGNED

○ ES = FALSE_SIGNED

NT : node type

ET : edge type

Rücksetzen eines LOOP-Blocks



- ▲ NS = ACTIVATED
- ✓ NS = COMPLETED
- ES = TRUE_SIGNED
- ES = FALSE_SIGNED

NT : node type

ET : edge type

Flexibilisierung

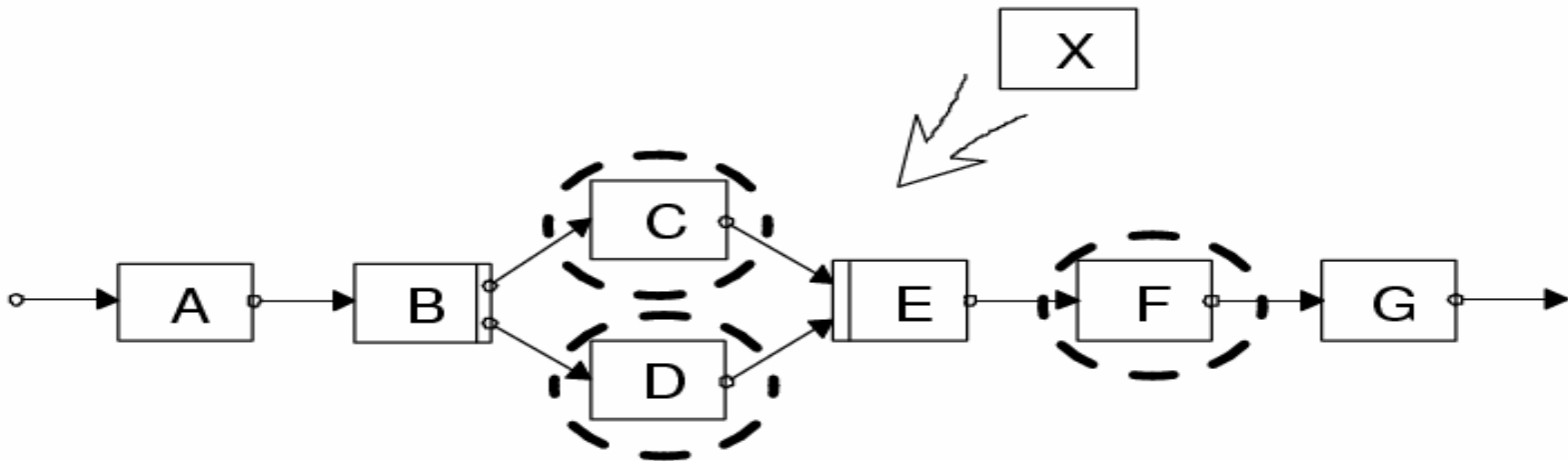
- Reale Prozesse definieren das Verhalten.
 - Regelfall ist die Ausnahme.
 - Modifikation durch EDV-Laien.
- Alle Operationen überführen das Prozessmodell in syntaktisch korrekten und legalen Zustand.

Flexibilisierung

- Strukturelle Änderungen am Workflow
 - Einfügen von Aktionen
 - Löschen von Aktionen
 - ...
- Dynamische Änderungen am Workflow
 - Auslassen von Aktionen
 - Vorziehen von Aktionen
 - ...
- Satz der Operationen minimal & vollständig

Einfügen von Aktivitäten

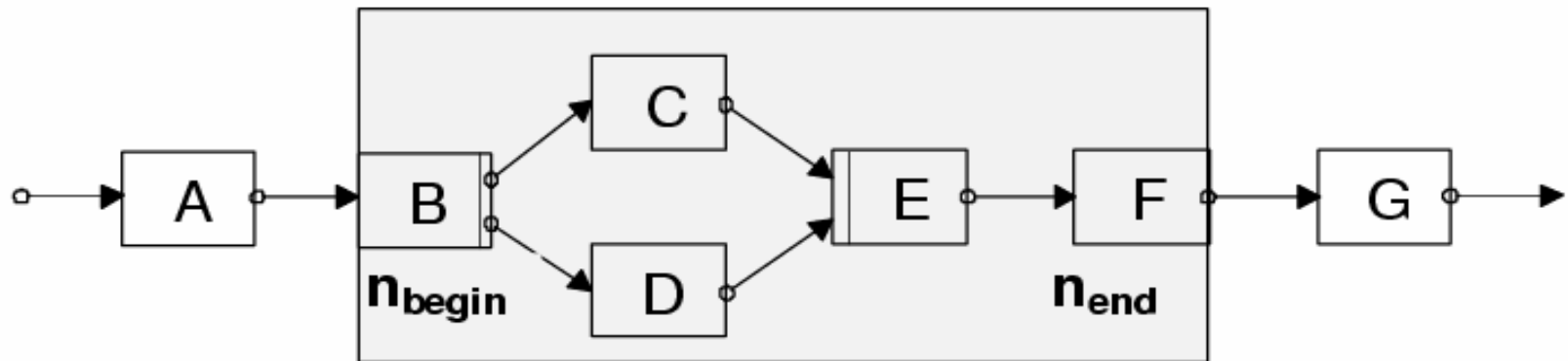
- Beispiel: Insert X between {C, D} and {F}



- $M_{\text{before}} = \{C, D\}$, $M_{\text{after}} = \{F\}$

Einfügen von Aktivitäten

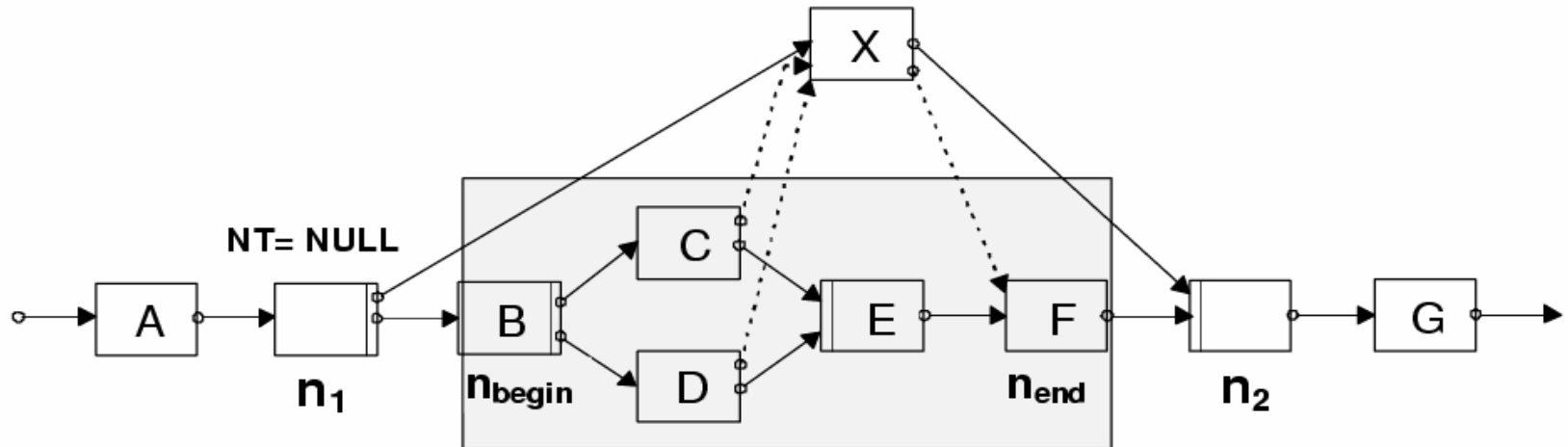
- Beispiel: Insert X between {C, D} and {F}



1. Finde den minimalen Block, der C, D und F enthält.

Einfügen von Aktivitäten

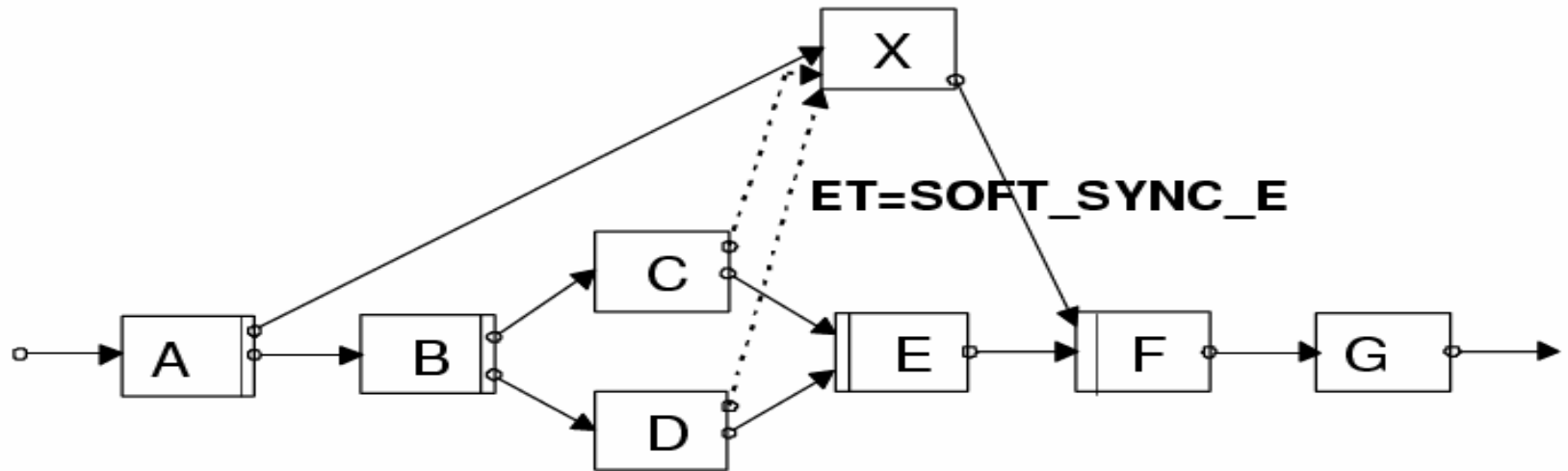
- Beispiel: Insert X between {C, D} and {F}



- Füge Hilfsknoten vor und nach dem Block ein.
Füge X zwischen den Hilfsknoten ein.
Synchronisiere X mit C, D und F.

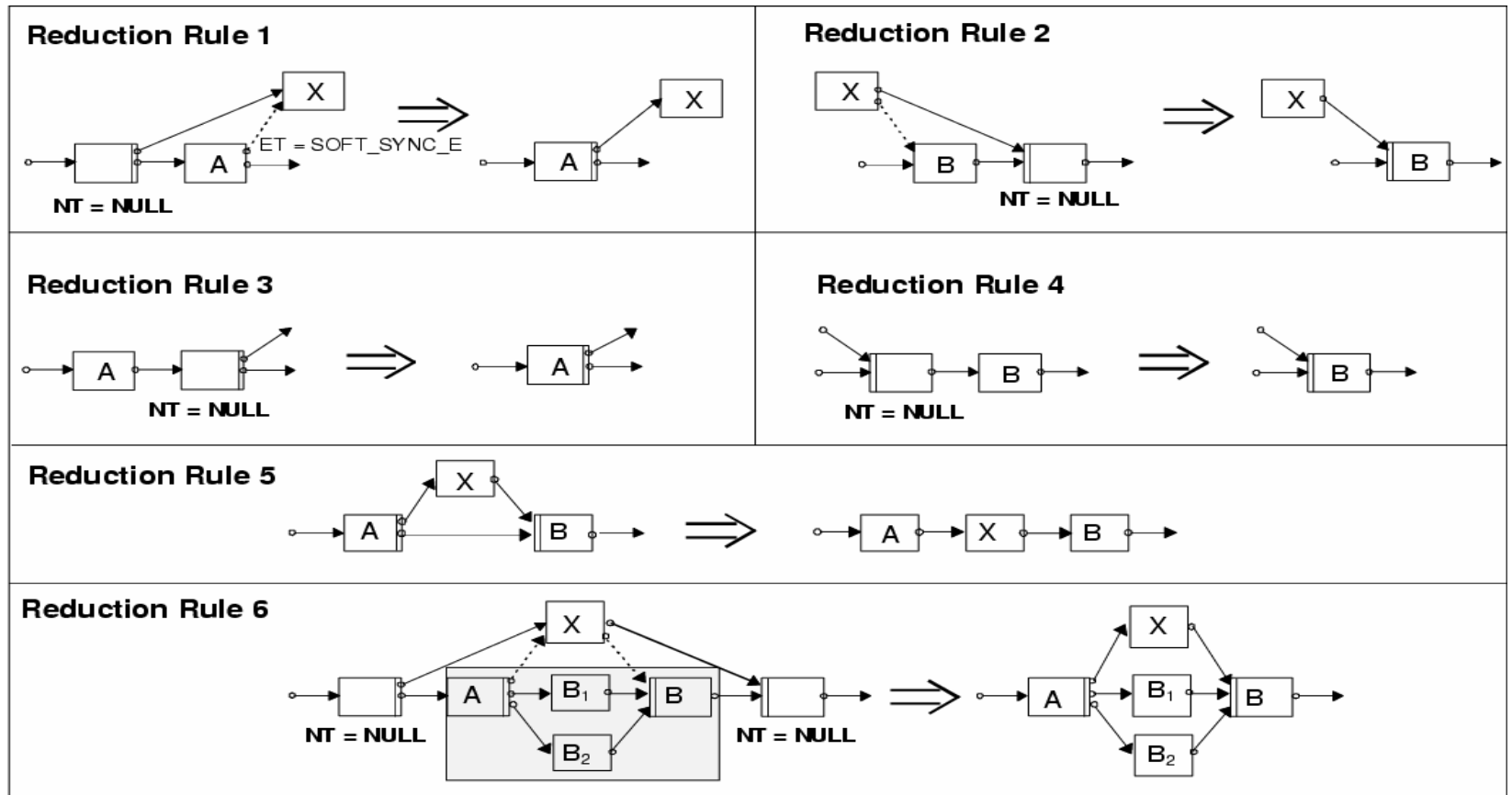
Einfügen von Aktivitäten

- Beispiel: Insert X between {C, D} and {F}

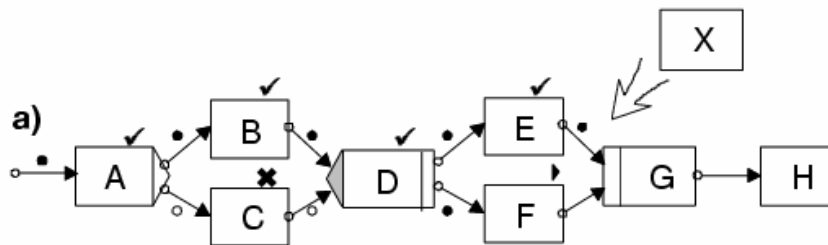


3. Vereinfache den Kontrollflussgraphen.

Regeln der Vereinfachung



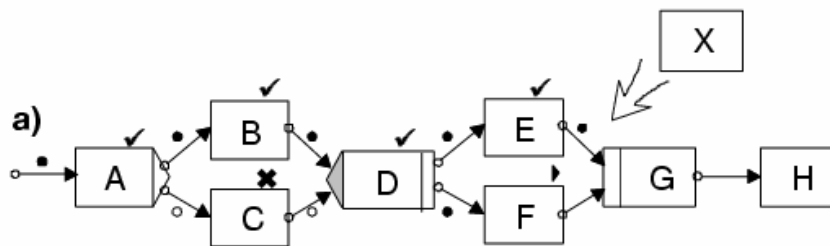
Einfügen von Aktivitäten - „on the fly“



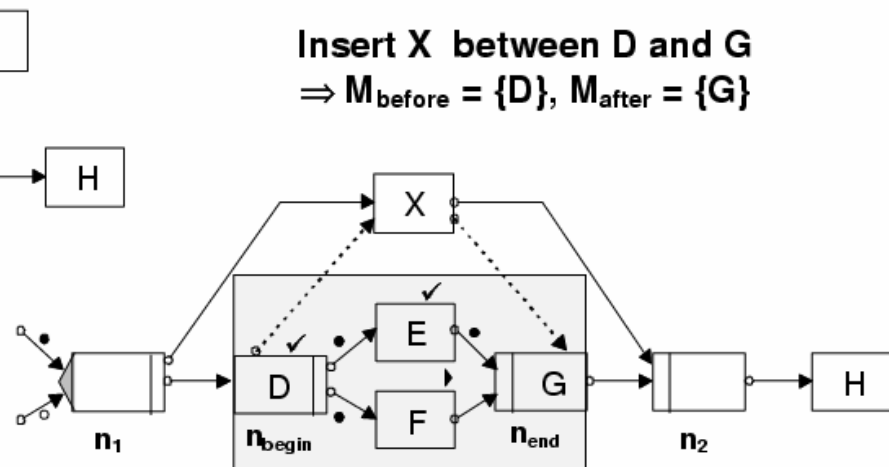
Insert X between D and G
 $\Rightarrow M_{\text{before}} = \{D\}, M_{\text{after}} = \{G\}$

- | | |
|------------------|---------------------|
| ▲ NS = ACTIVATED | • ES = TRUE_SIGNED |
| ▶ NS = RUNNING | ○ ES = FALSE_SIGNED |
| ✓ NS = COMPLETED | |
| ✗ NS = SKIPPED | |

Einfügen von Aktivitäten - „on the fly“



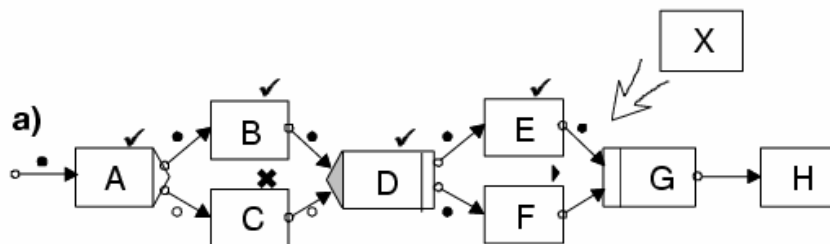
- b)
1. Find the minimal block containing D and G
 2. Insert n_1 and n_2 before / after the block. Insert X as a new branch between n_1 and n_2 , and synchronize it with D and G.



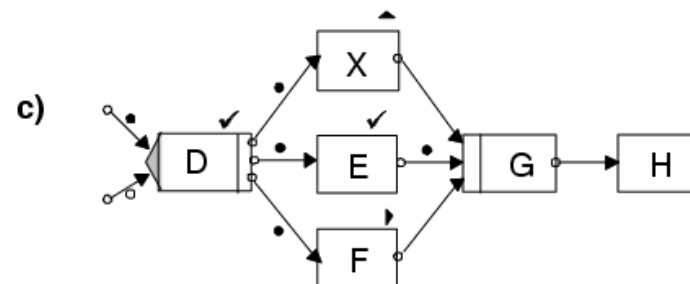
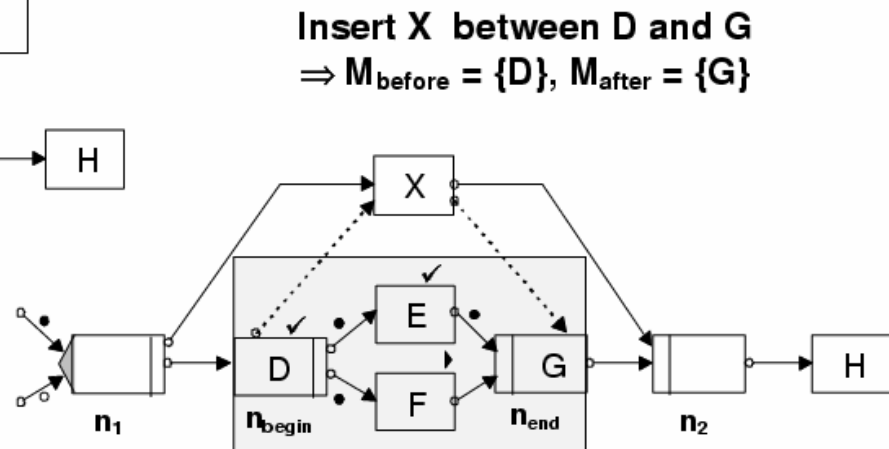
- ▲ NS = ACTIVATED
- ▶ NS = RUNNING
- ✓ NS = COMPLETED
- ✗ NS = SKIPPED

- ES = TRUE_SIGNED
- ES = FALSE_SIGNED

Einfügen von Aktivitäten - „on the fly“



- b)
1. Find the minimal block containing D and G
 2. Insert n_1 and n_2 before / after the block. Insert X as a new branch between n_1 and n_2 , and synchronize it with D and G.



3. Apply reduction rule 6 and reevaluate the workflow's state

▲ NS = ACTIVATED
 ► NS = RUNNING
 ✓ NS = COMPLETED
 ✗ NS = SKIPPED

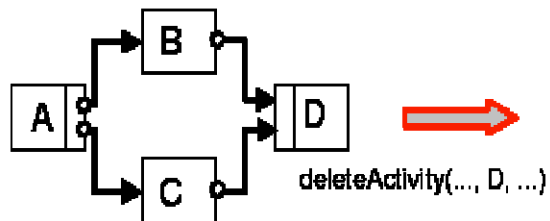
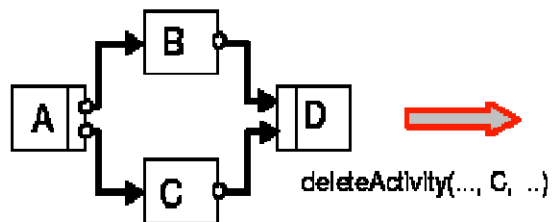
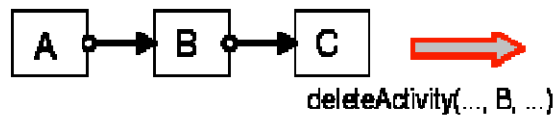
• ES = TRUE_SIGNED
 ○ ES = FALSE_SIGNED

Einfügen von Aktivitäten - „on the fly“

- Syntaktische Korrektheit und legaler Zustand
 - nachfolgende Aktionen im Zustand NotActivated / Activated
- Einfügen hat Vorrang vor Ausführung
- Nach Einfügen wird Systemzustand re-evaluiert
- Datenkonsistenz muss beachtet werden
 - Änderungsmanagement notwendig
 - Fehlende Inputs bereitstellen

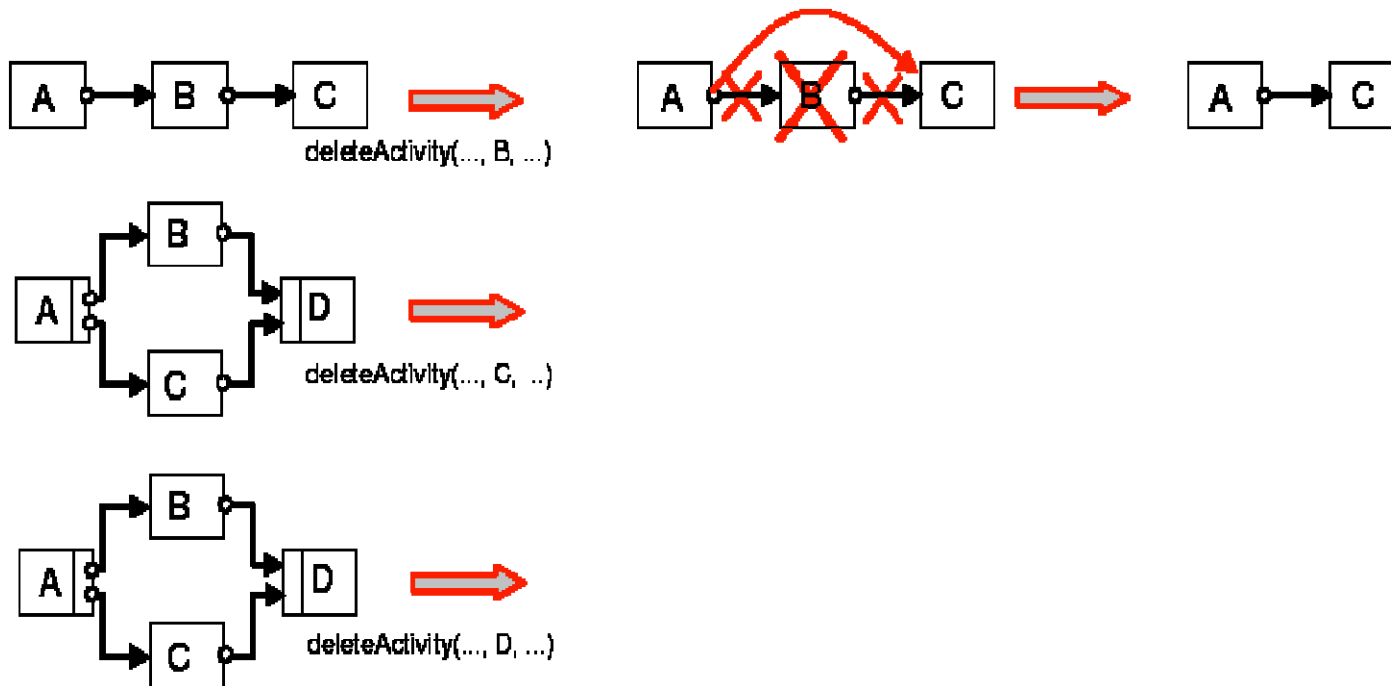
Löschen von Aktivitäten

- Änderung des Kontrollflusses



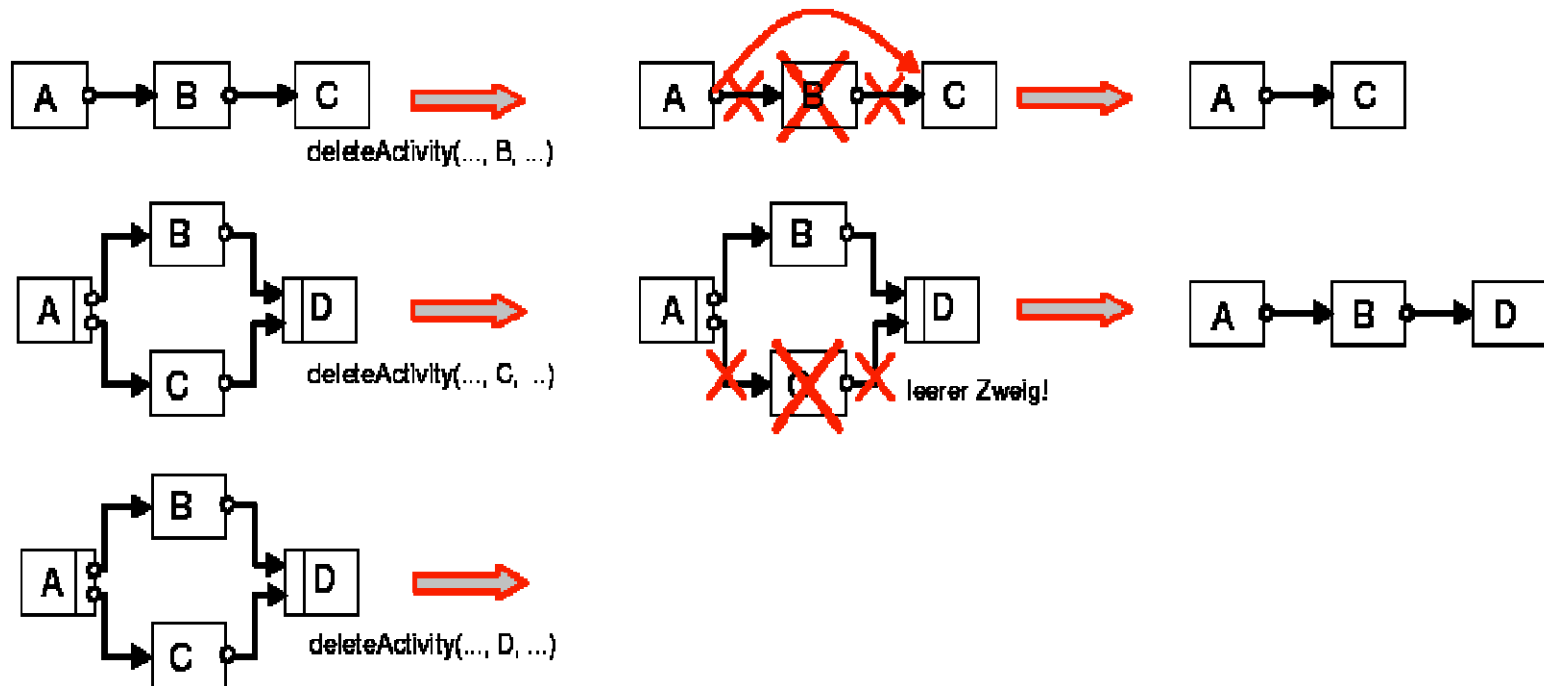
Löschen von Aktivitäten

- Änderung des Kontrollflusses



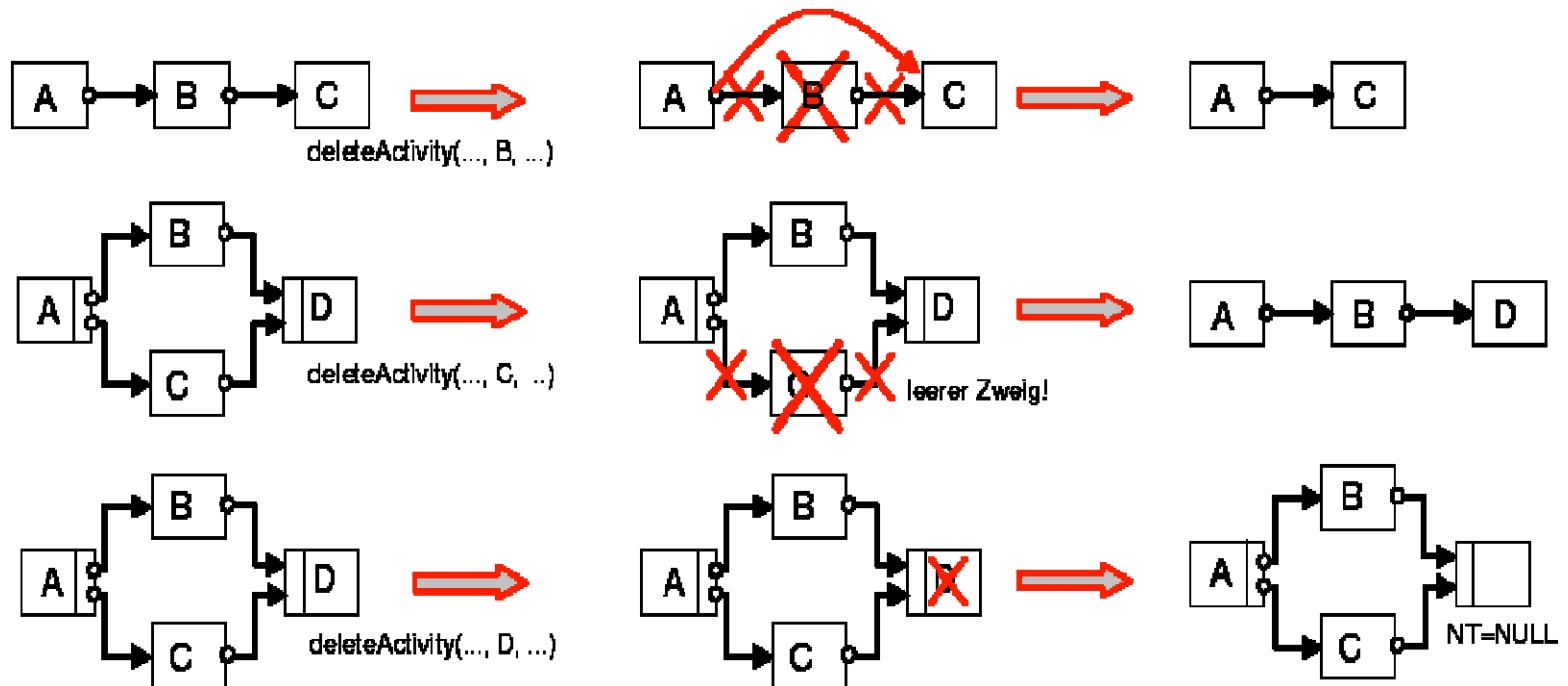
Löschen von Aktivitäten

- Änderung des Kontrollflusses



Löschen von Aktivitäten

- Änderung des Kontrollflusses

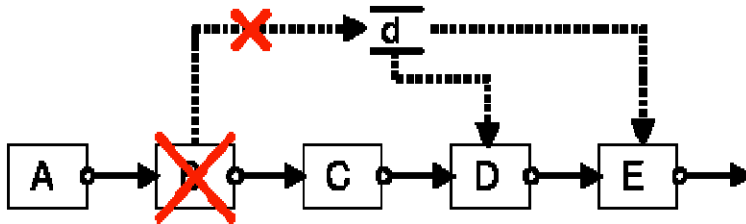


Löschen von Aktivitäten - „on the fly“

- Legaler Zustand beachten
 - Löschen im Zustand NotActivated / Activated
- Aktivitäten bleibt als NullTask übrig
 - keine Änderung der Struktur
- Möglichkeiten bei fehlenden Daten:
 - alle abhängigen Aufgaben löschen
 - Einfügen einer die Daten erzeugenden Aktivitäten
 - Angabe einer anderen Quelle (z.B. Dialog-Maske)
 - Verbot der Löschung

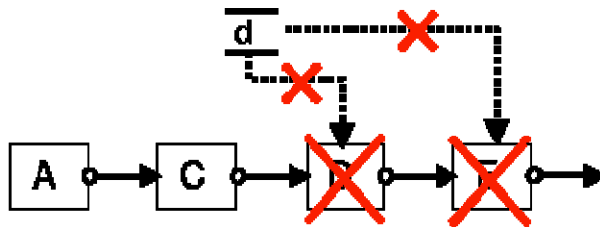
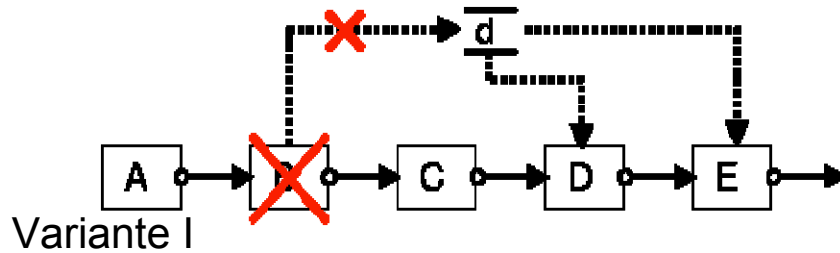
Löschen von Aktivitäten - „on the fly“

- Änderung des Datenflusses



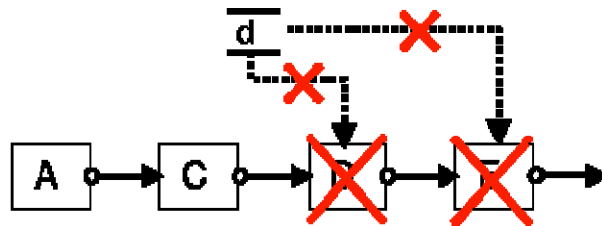
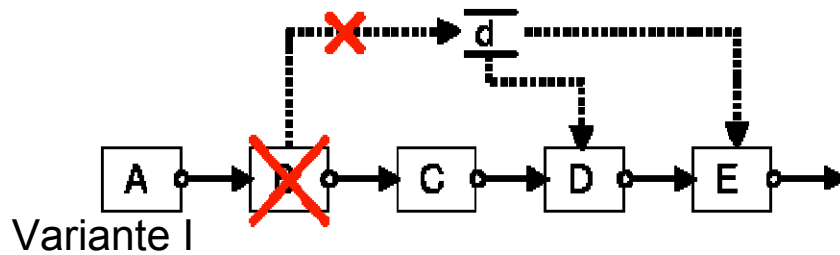
Löschen von Aktivitäten - „on the fly“

- Änderung des Datenflusses

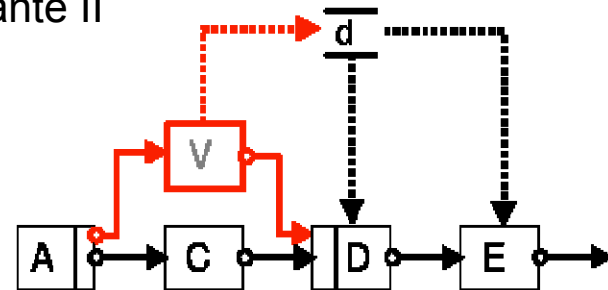


Löschen von Aktivitäten - „on the fly“

- Änderung des Datenflusses

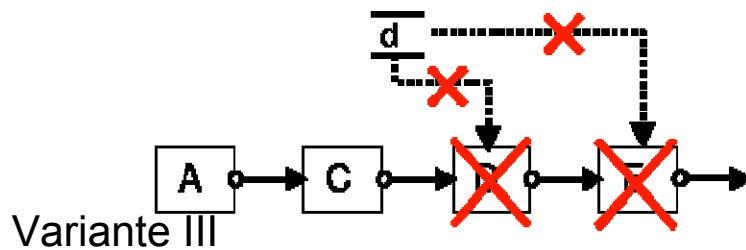
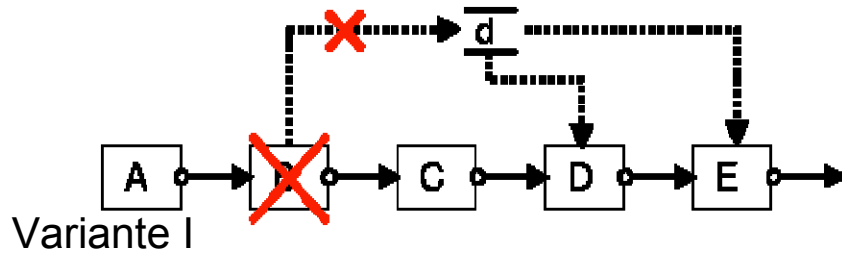


Variante II

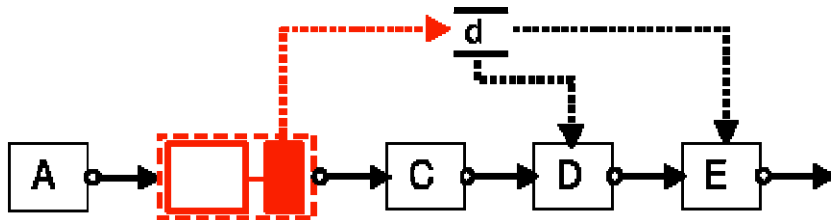
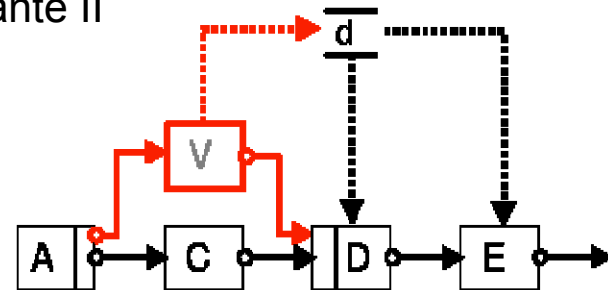


Löschen von Aktivitäten - „on the fly“

- Änderung des Datenflusses

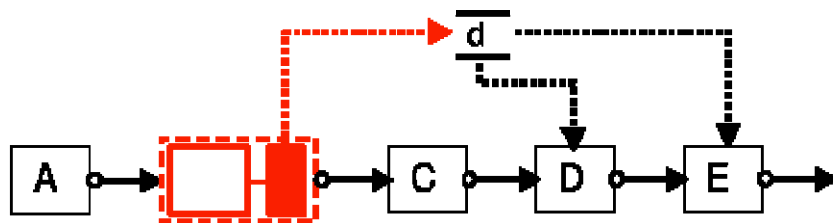
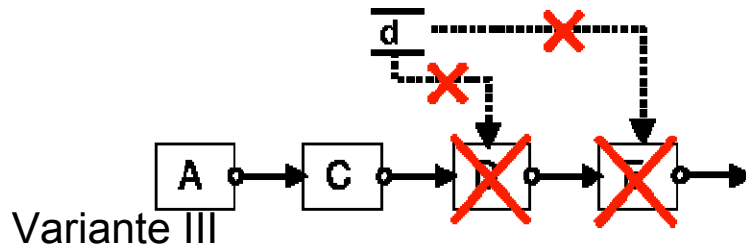
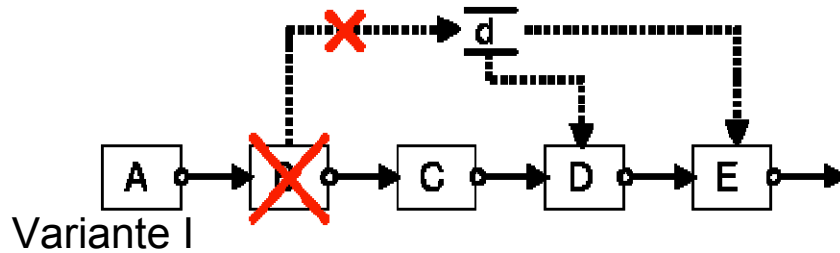


Variante II

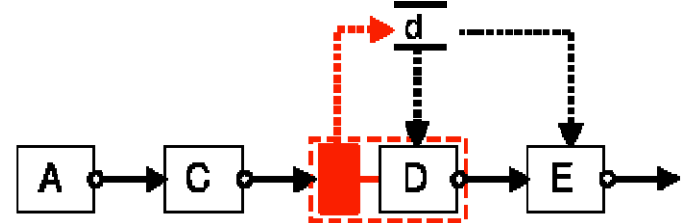
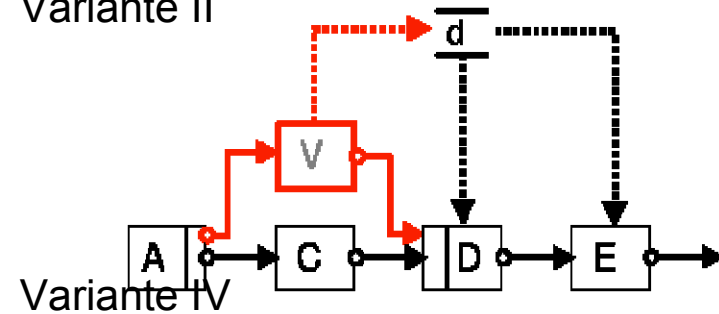


Löschen von Aktivitäten - „on the fly“

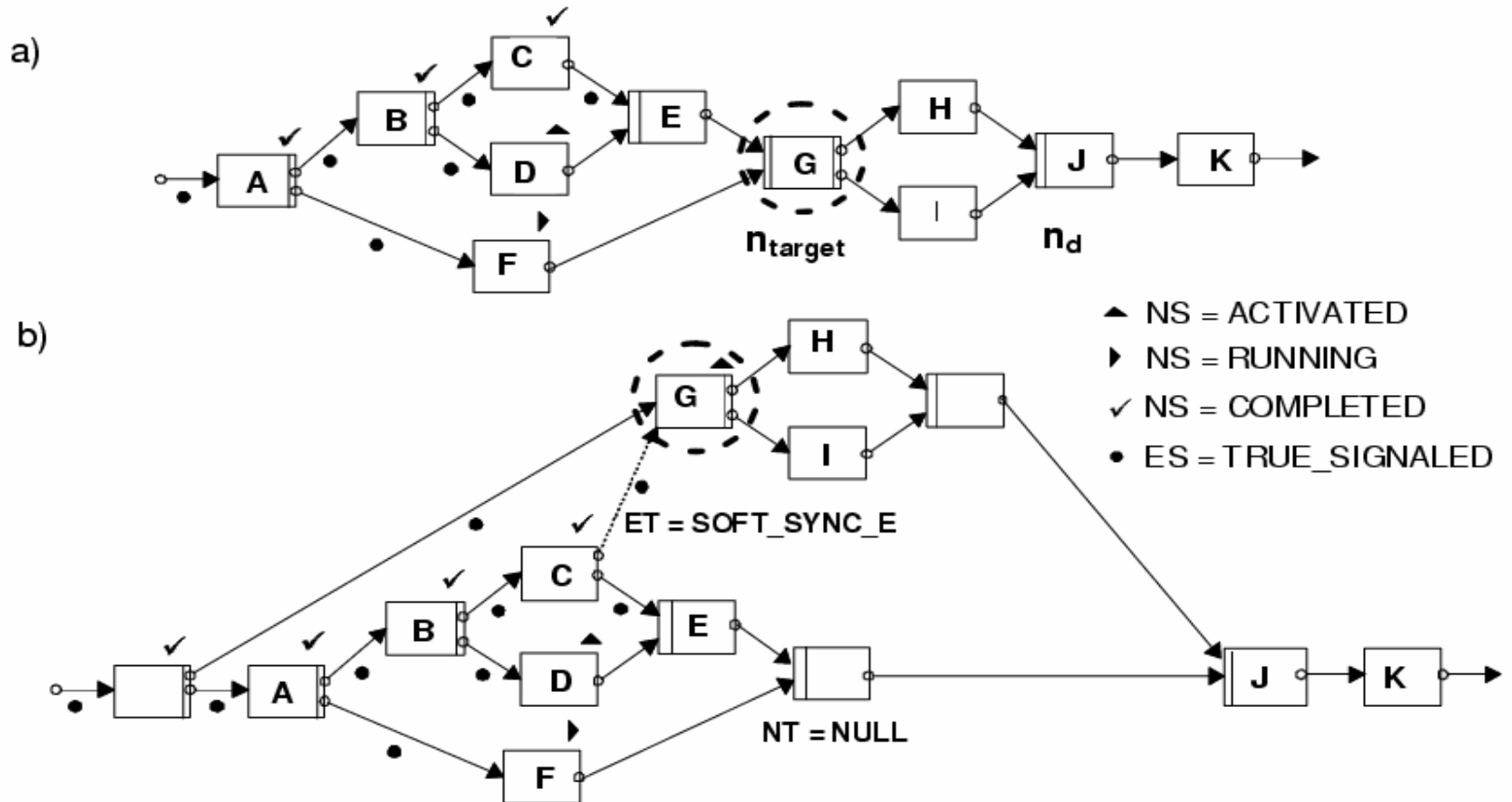
- Änderung des Datenflusses



Variante II



Umordnen von Aktivitäten



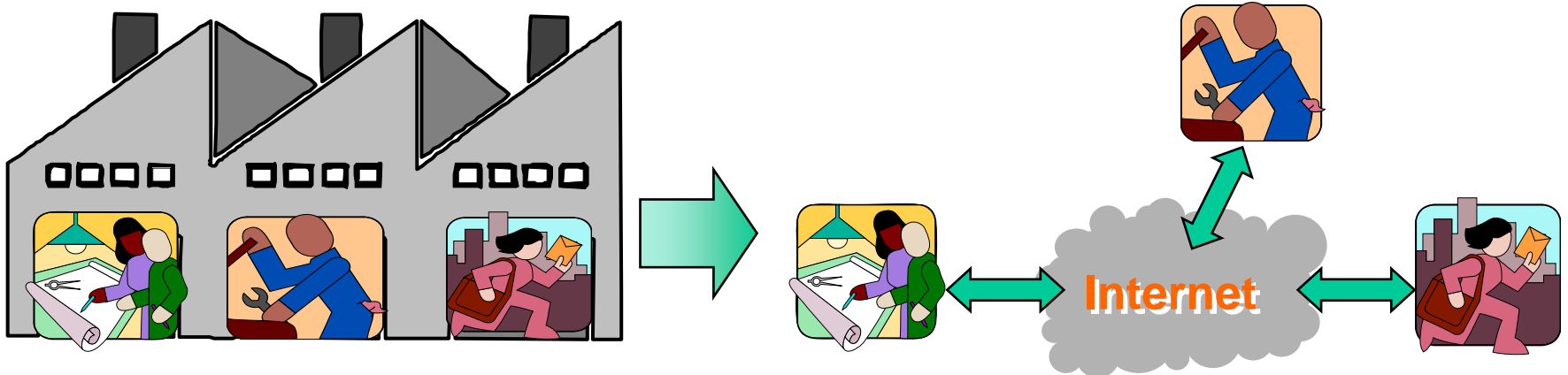
Temporäre vs. permanente Änderungen

- Permanente Änderung gilt für Prozessmodell.
- Temporäre Änderung gilt für Prozessinstanz.
- Änderungsmanagement notwendig
 - Regelung der Zugriffsrechten
 - Protokollierung der Änderungen (Logging)
- Rücksetzung von temporären Änderungen möglich
- Permanente Änderungen dürfen nicht von temporären Änderungen abhängen.

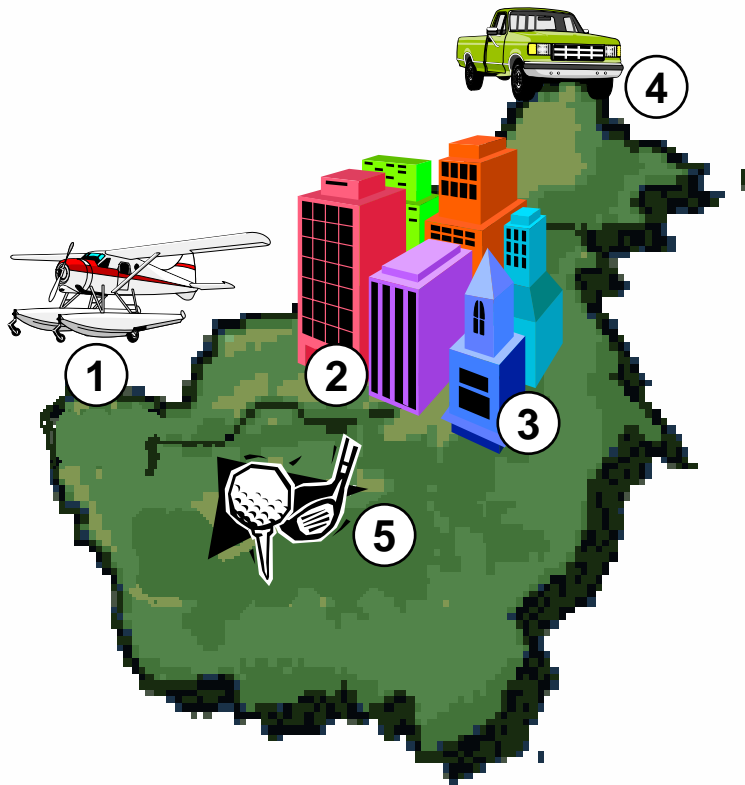
Modelle für (Web) Services

Verteilte Geschäftsprozesse

- Dezentralisierung
- Modularisierung
- Standardisierung



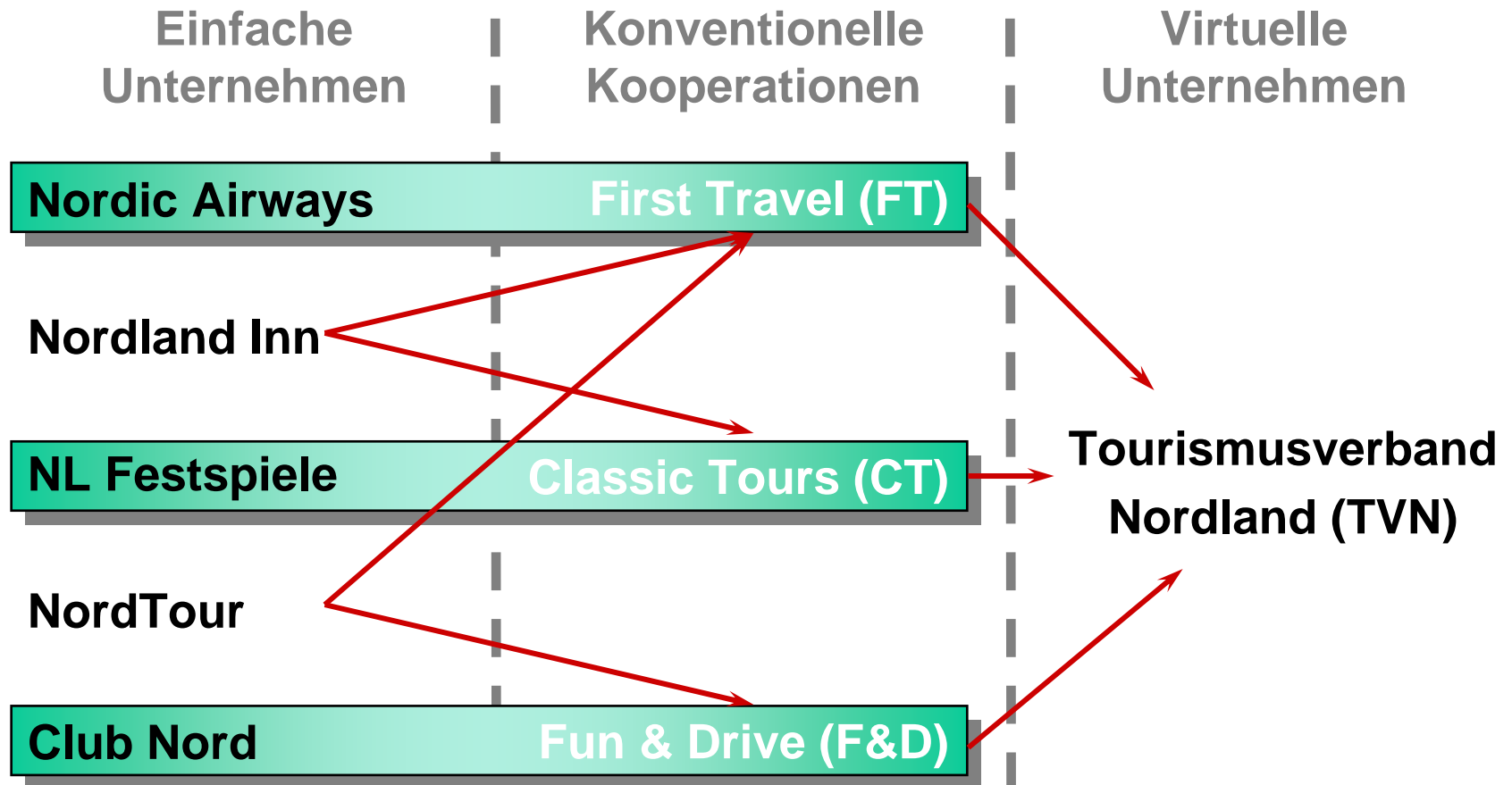
Virtuelle Unternehmen



Firmen in Nordland:

- (1) Nordic Airways
- (2) Nordland Inn Hotel
- (3) Nordland Festspiele
- (4) NordTour - Car rental
- (5) Club Nord - Golf Club

Virtuelle Unternehmen



Definition

Ein *Virtuelles Unternehmen* ist eine zeitlich befristete, zwischenbetriebliche Kooperation mehrerer, rechtlich unabhängiger Unternehmen in einem informationstechnisch unterstützten Netzwerk, das:

- innerhalb kürzester Zeit für einen bestimmten Auftrag entsteht,
- dem Kunden durch die Integration der Kernkompetenzen individualisierte Produkte und Dienstleistungen erstellt und
- ohne physische und juristische Körperschaften auskommt.

Ein Virtuelles Unternehmen tritt Dritten gegenüber wie ein einheitliches Unternehmen auf.

Der Web-Service-Ansatz

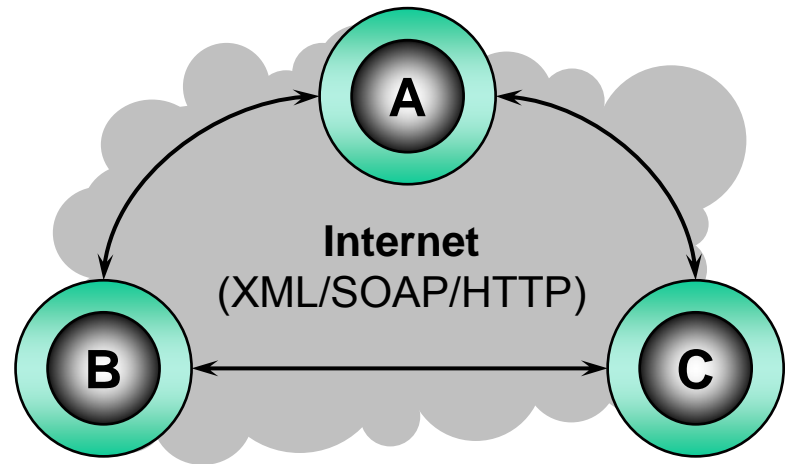
Ein **Web Service** ist ein

- inhaltlich abgeschlossenes,
- beliebig komplexes und
- selbsterklärendes

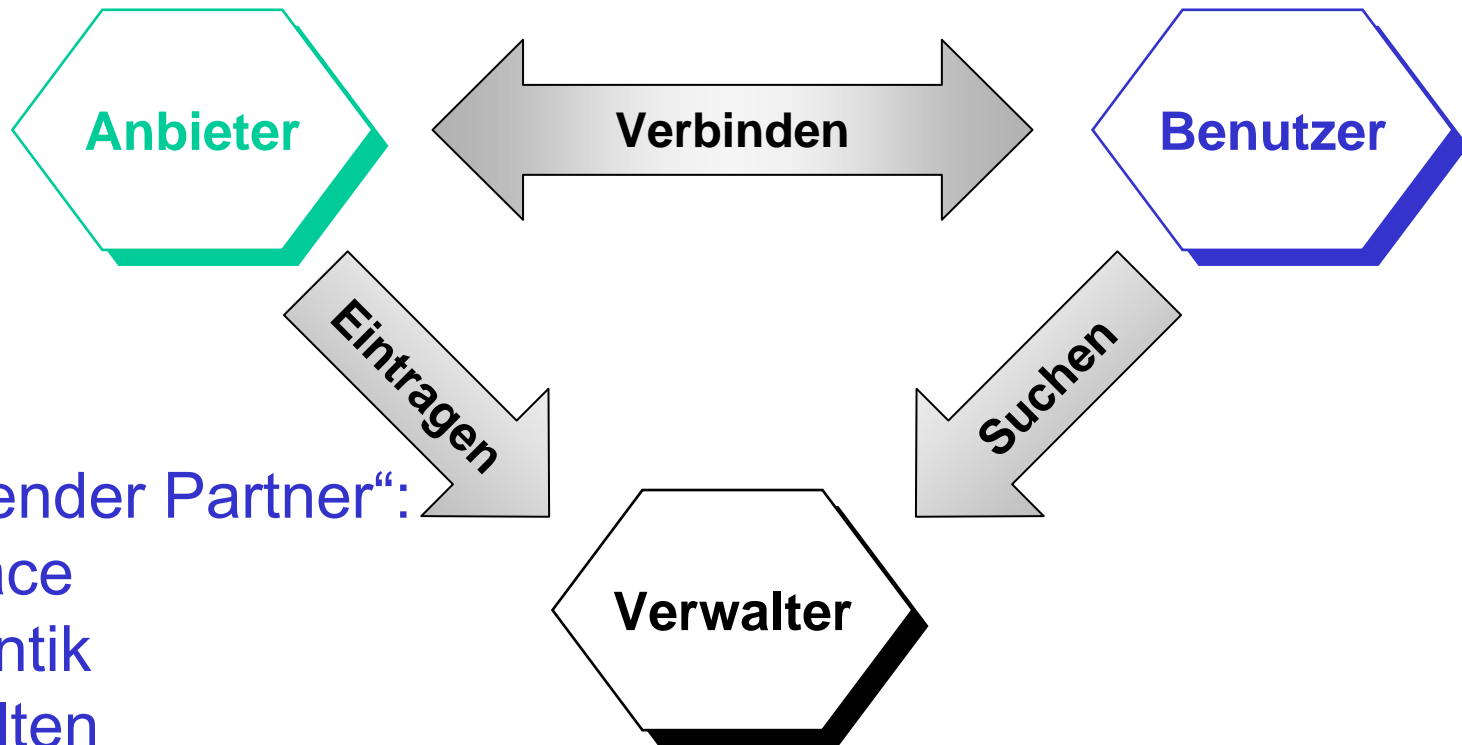
Anwendungsmodul im Internet.

Ein Web Service

- kapselt einen **lokalen** Geschäftsprozess,
- kommuniziert über eine **standardisierte** Schnittstelle.



Paradigma: Web Services



„passender Partner“:

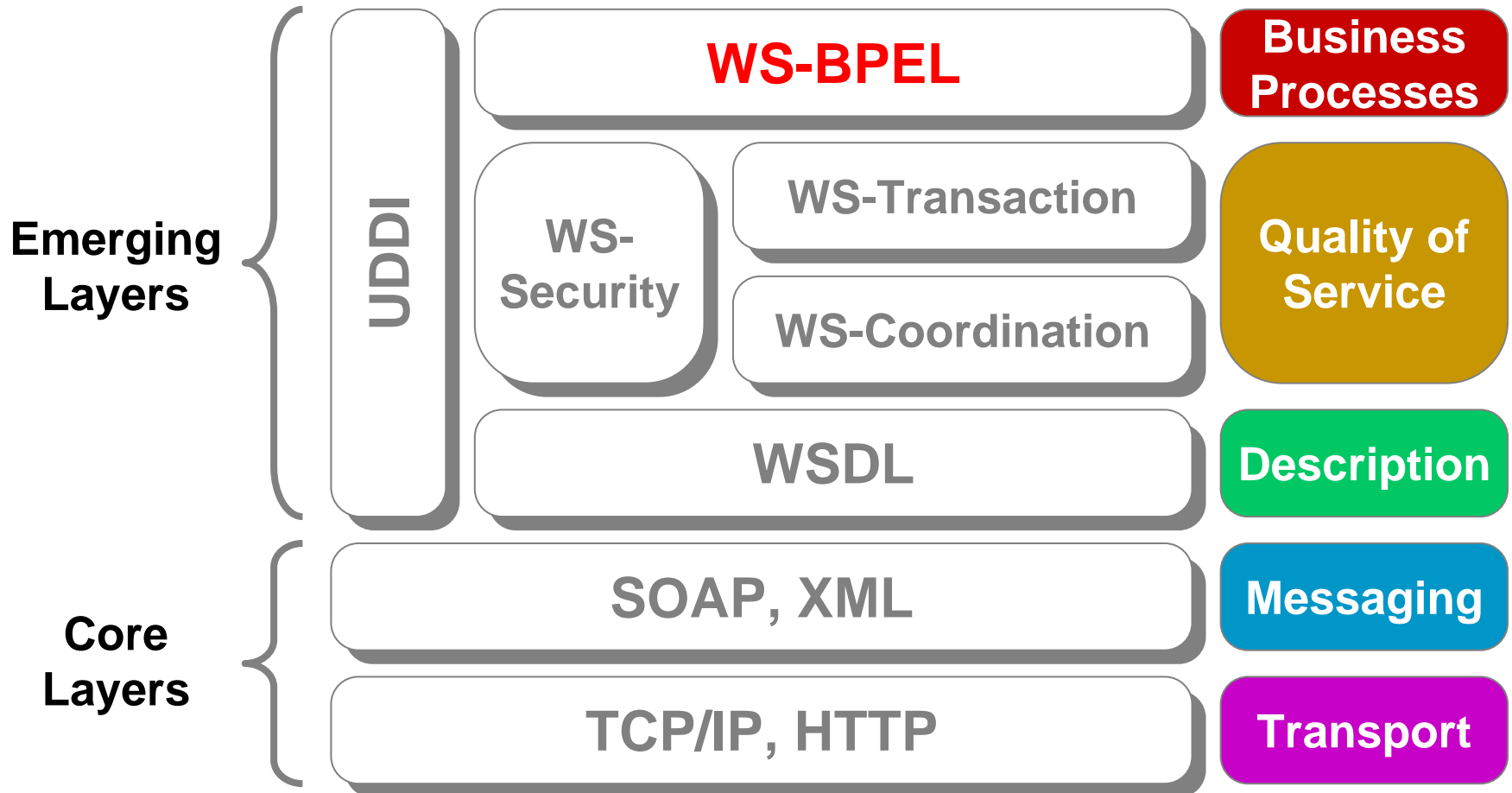
Interface

Semantik

Verhalten

Nichtfunktionale Eigenschaften

Web Service Technology Stack



BPEL-Einführung

Was ist BPEL?

- Business Process Execution Language for Web Services (früher: BPEL4WS, jetzt: WS-BPEL)
- „BPEL ist eine XML-basierte Sprache zur Beschreibung von Geschäftsprozessen, deren einzelne Aktivitäten durch Webservices implementiert sind.

Weiterhin kann mit BPEL ein Webservice selbst beschrieben werden.“



WIKIPEDIA
Die freie Enzyklopädie

Webservice

ist BPEL?

Kontroll-
Fluss

neuer
Webservice

```
<process>
<variables>
  <variable name="request" />
  <variable name="answer" />
</variables>
<sequence>
  <flow>
    <receive partnerLink="A" />
    <invoke partnerLink="B" />
  </flow>
  <invoke partnerLink="C" />
</sequence>
</process>
```

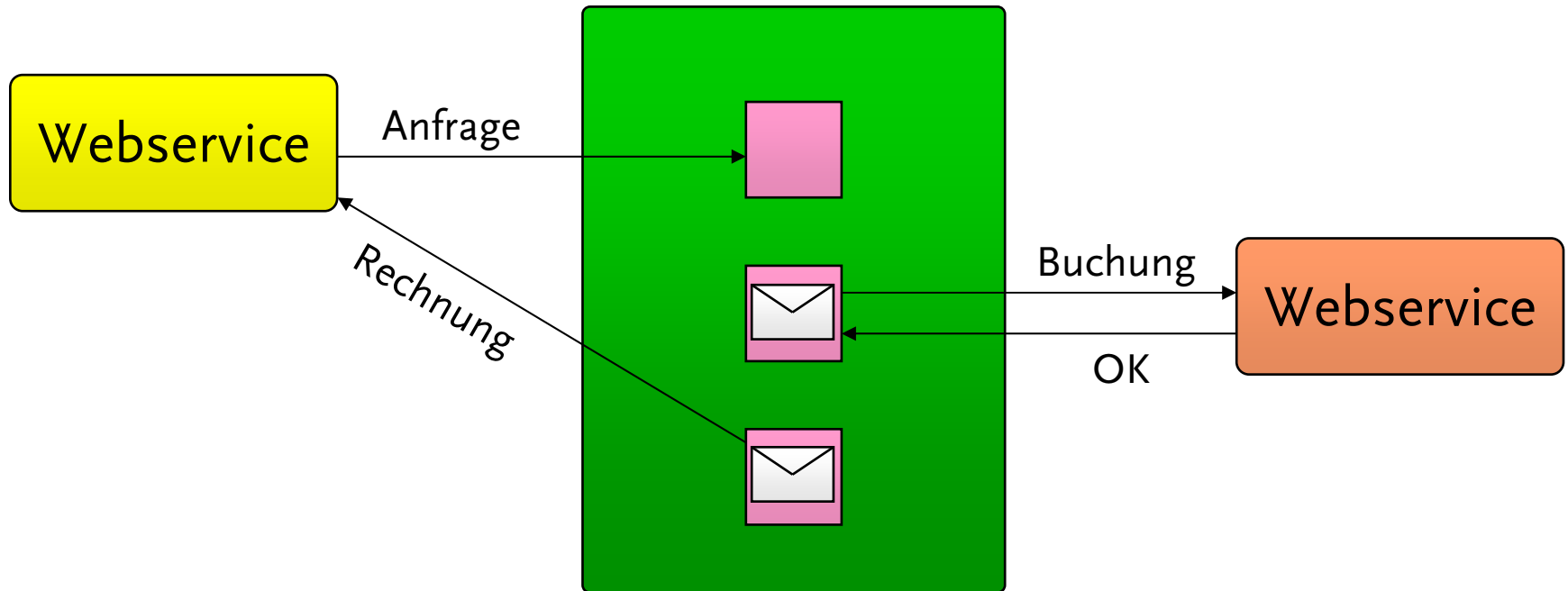
Webservice

„Programmieren im Großen“

„Service-Orchestrierung“

Webservice

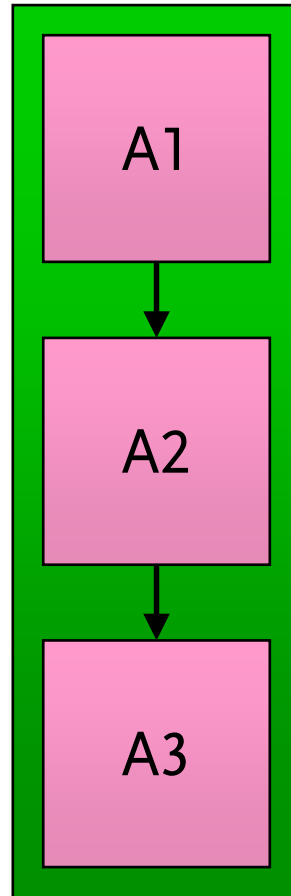
Aktivitäten: Kommunikation



- Receive: Anfragen annehmen
- Invoke: Webservices aufrufen
- Reply: Anfragen beantworten

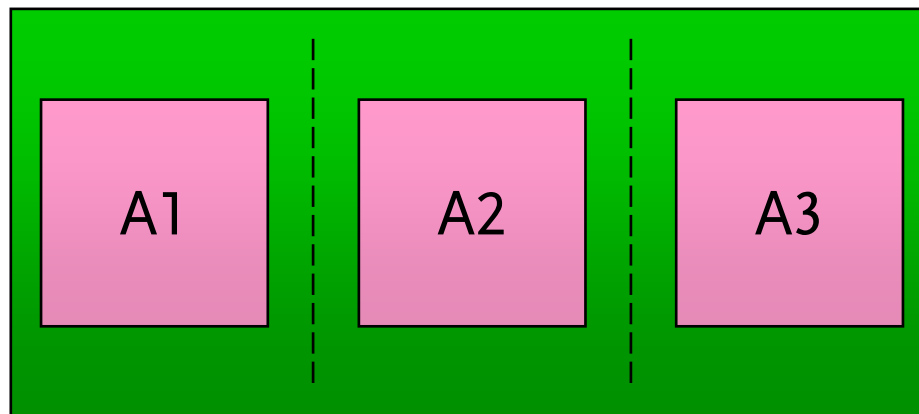
Strukturierte Aktivitäten

- Sequence: sequentielle Ausführung



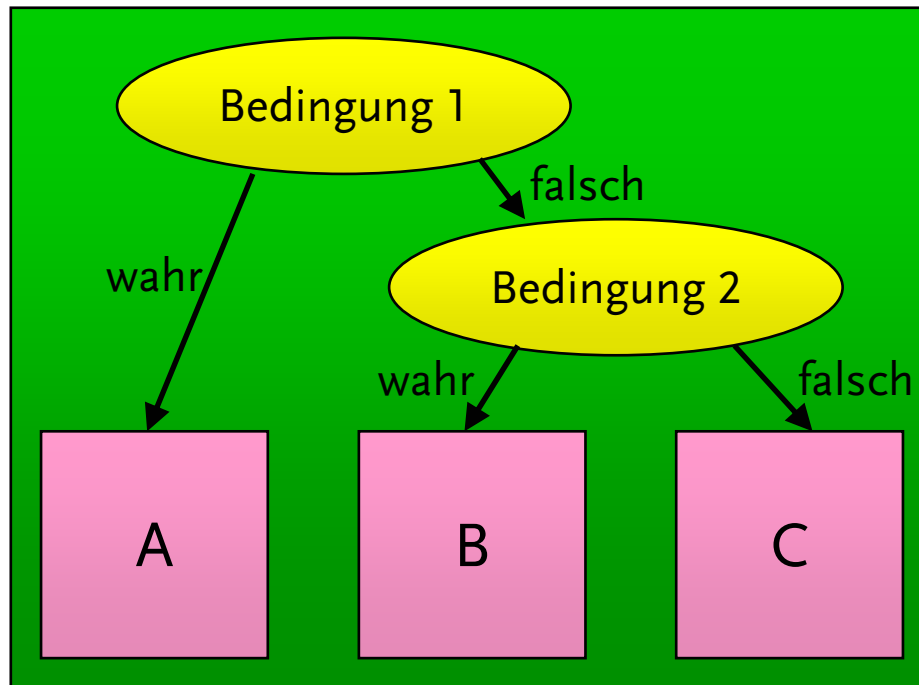
Strukturierte Aktivitäten

- Flow: nebenläufige Ausführung



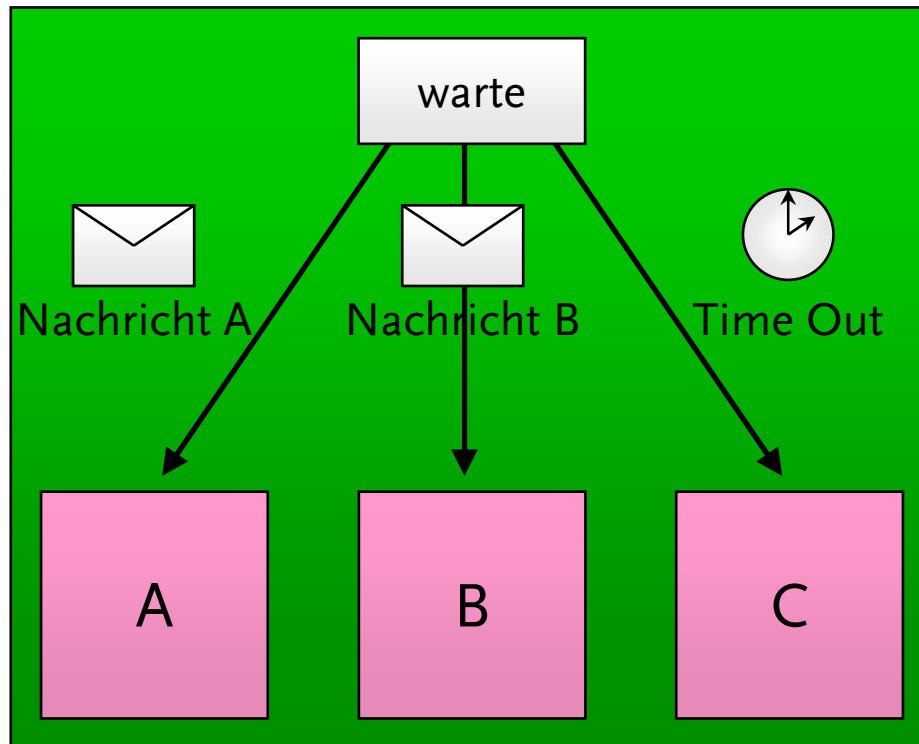
Strukturierte Aktivitäten

- Switch: bedingte Ausführung (Daten)



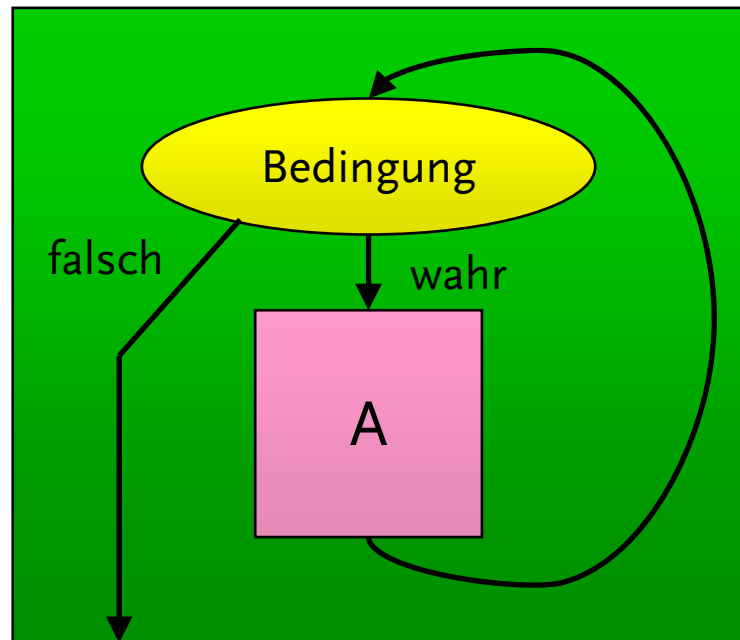
Strukturierte Aktivitäten

- Pick: bedingte Ausführung (Nachrichten, Zeit)



Strukturierte Aktivitäten

- While: wiederholte Ausführung



Nachrichten und Daten

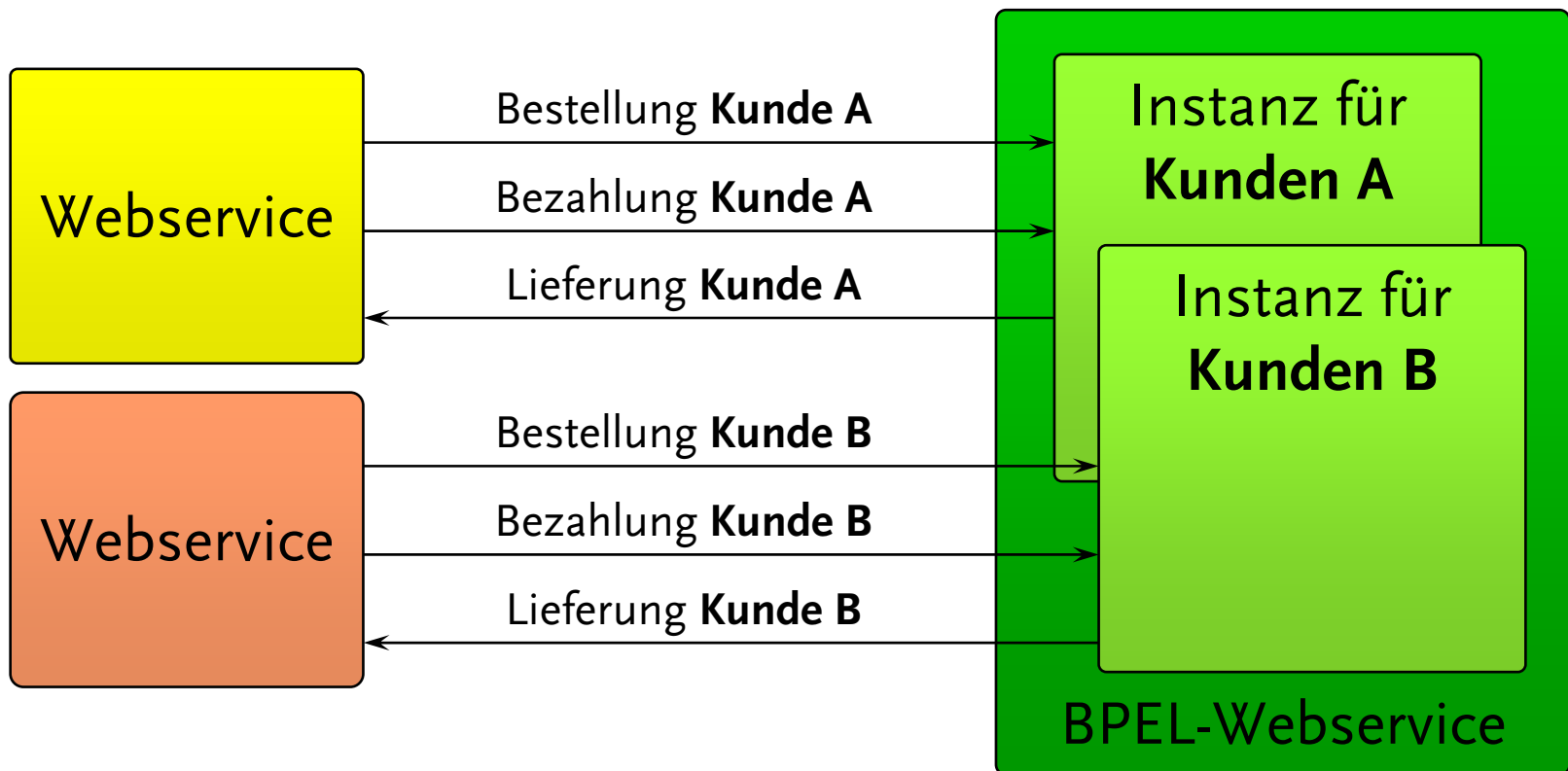
- Einheitliches Datenmodell

- Alles ist XML:
 - Nachrichten
 - Daten
 - Vergleiche

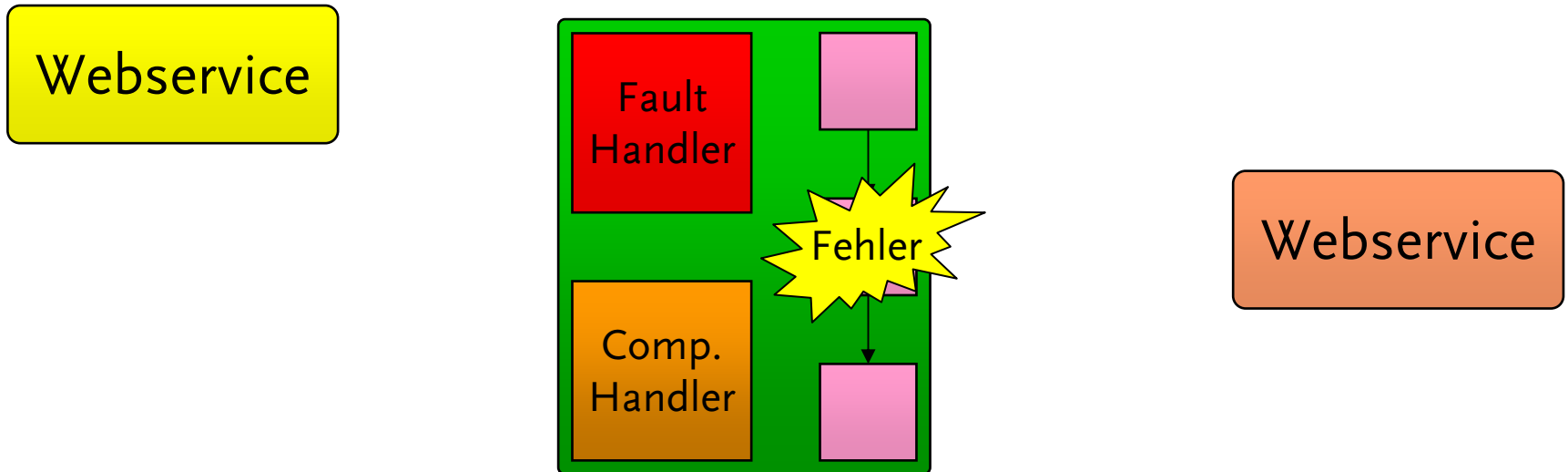


Instanzen

- BPEL-Datei als „Bauplan“
- Correlation Sets als „Schlüsselwort“



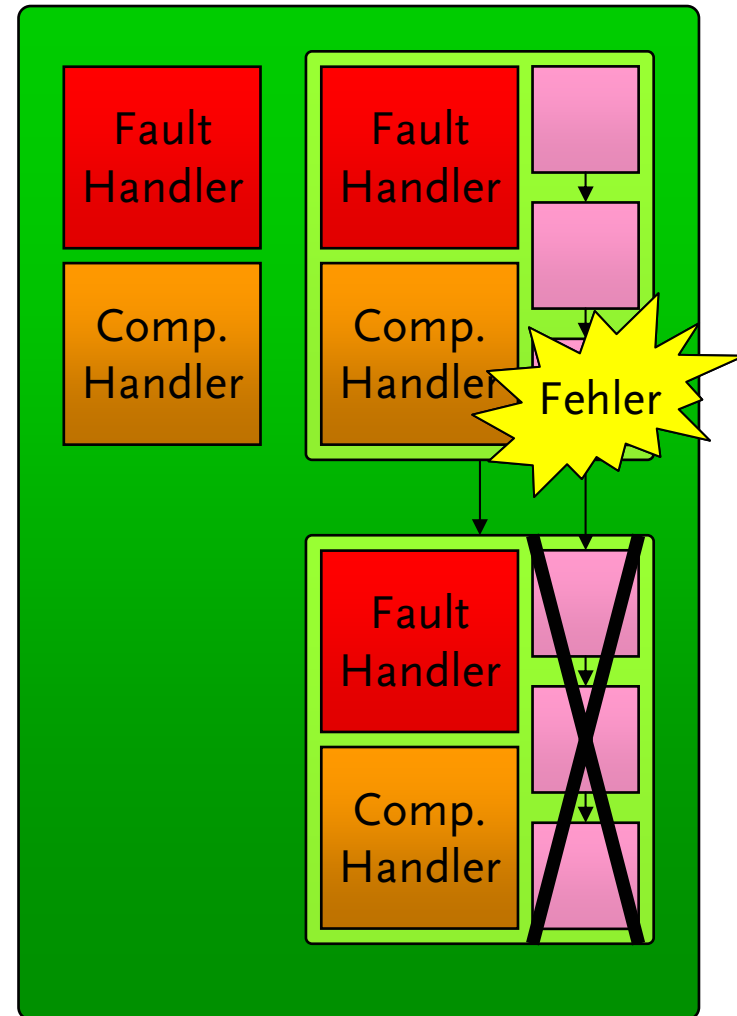
Fehler



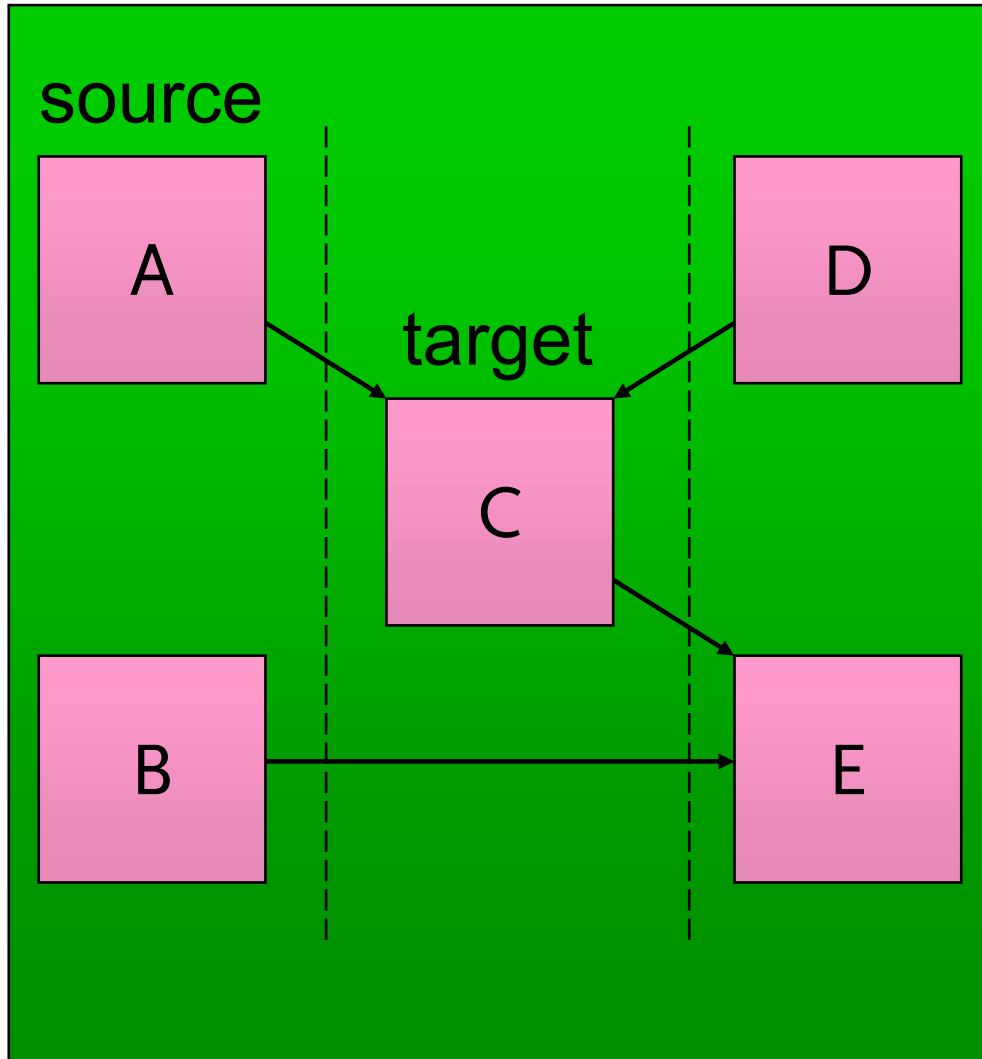
- Fehlerbehandlung mittels Fault Handler
- Webservices = „long running transactions“
- Kompensation mittels Compensation Handler

Kapselung

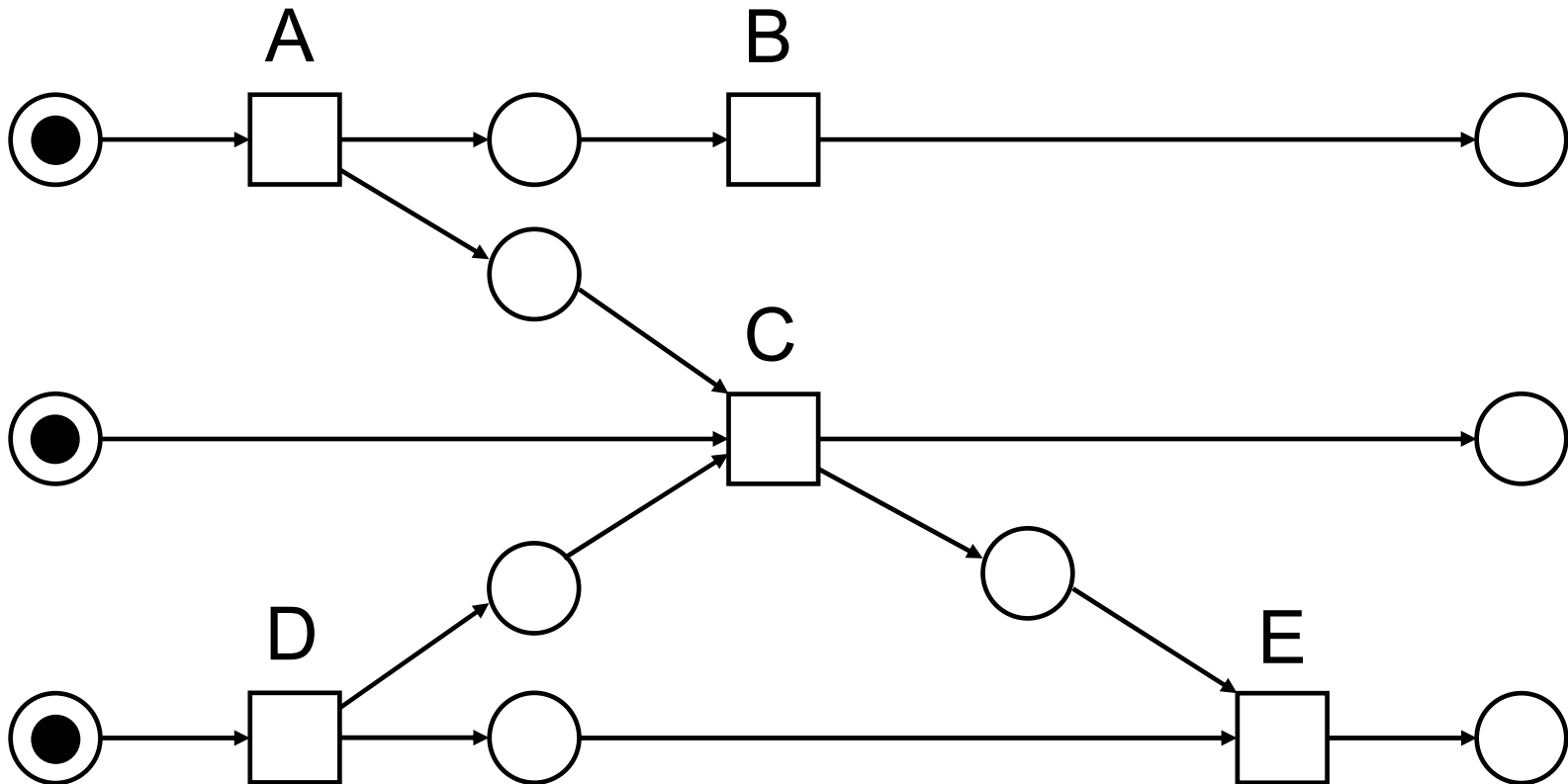
- Fehlerbehandlung
- Kompensation
- Namensräume



Links

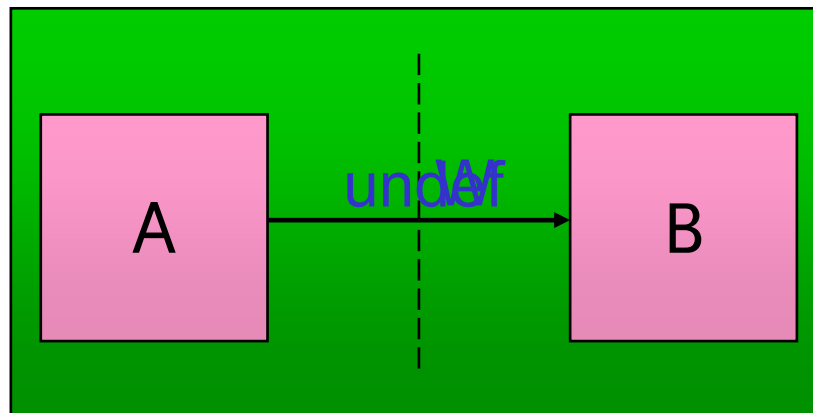


... nur ein halbgeordneter Ablauf

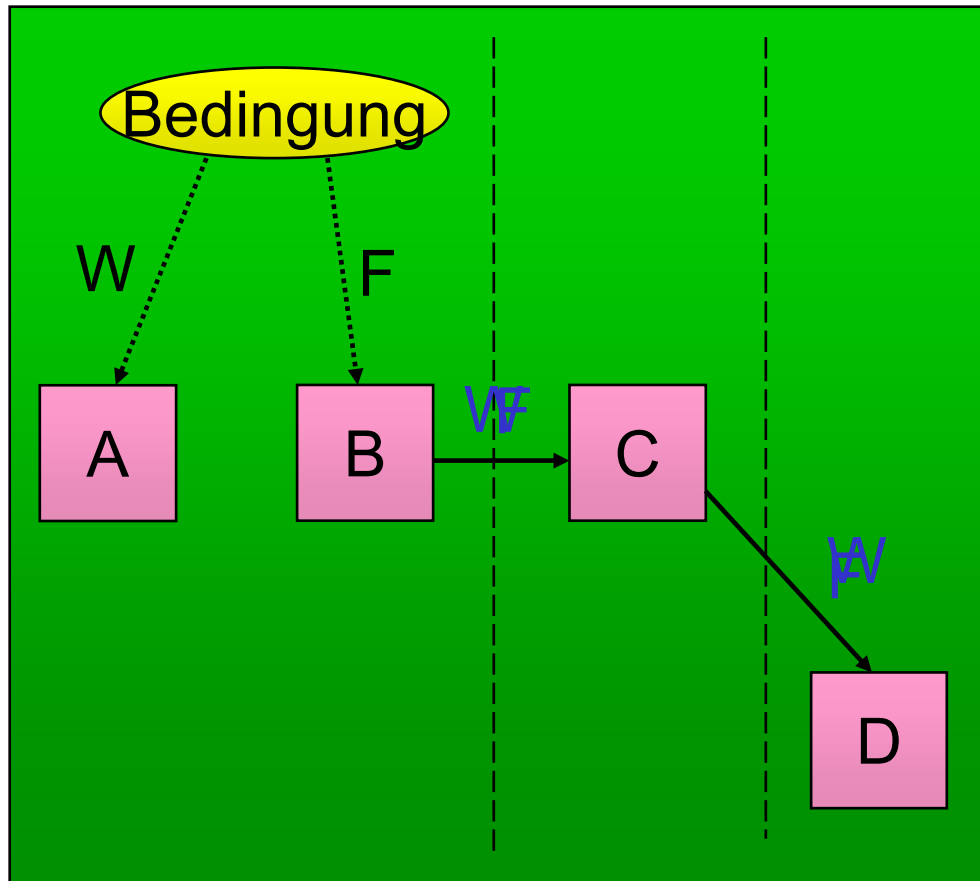


Etwas detaillierter ...

- Link verbindet eine source-Aktivität mit einer target-Aktivität
- $\text{Status} := \text{Links} \rightarrow \{W, F, \text{undef}\}$
 - Status jedes Link zu Prozessbeginn undef
 - A abgearbeitet $\rightarrow \text{Status}(AB) = W$
 - B aktiviert und Status eines Link wahr \rightarrow starte B

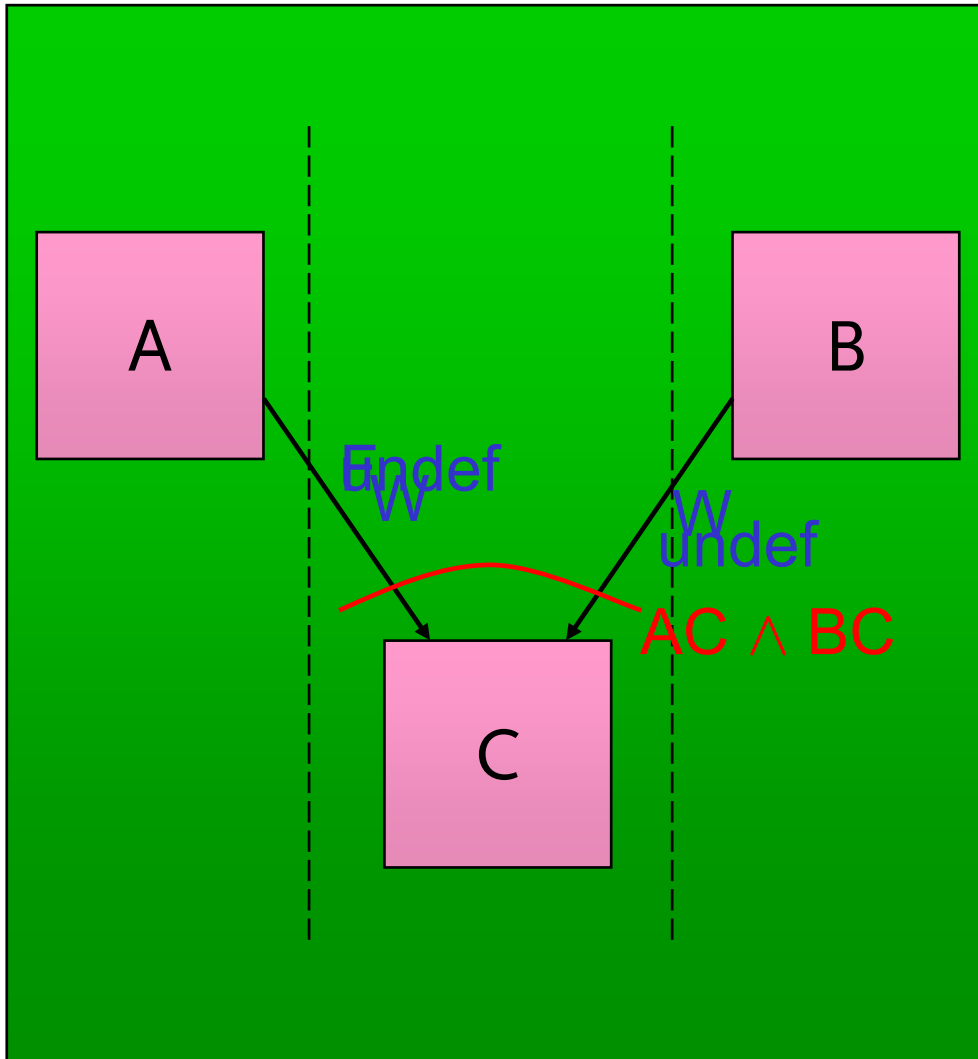


Dead-Path-Elimination (DPE)



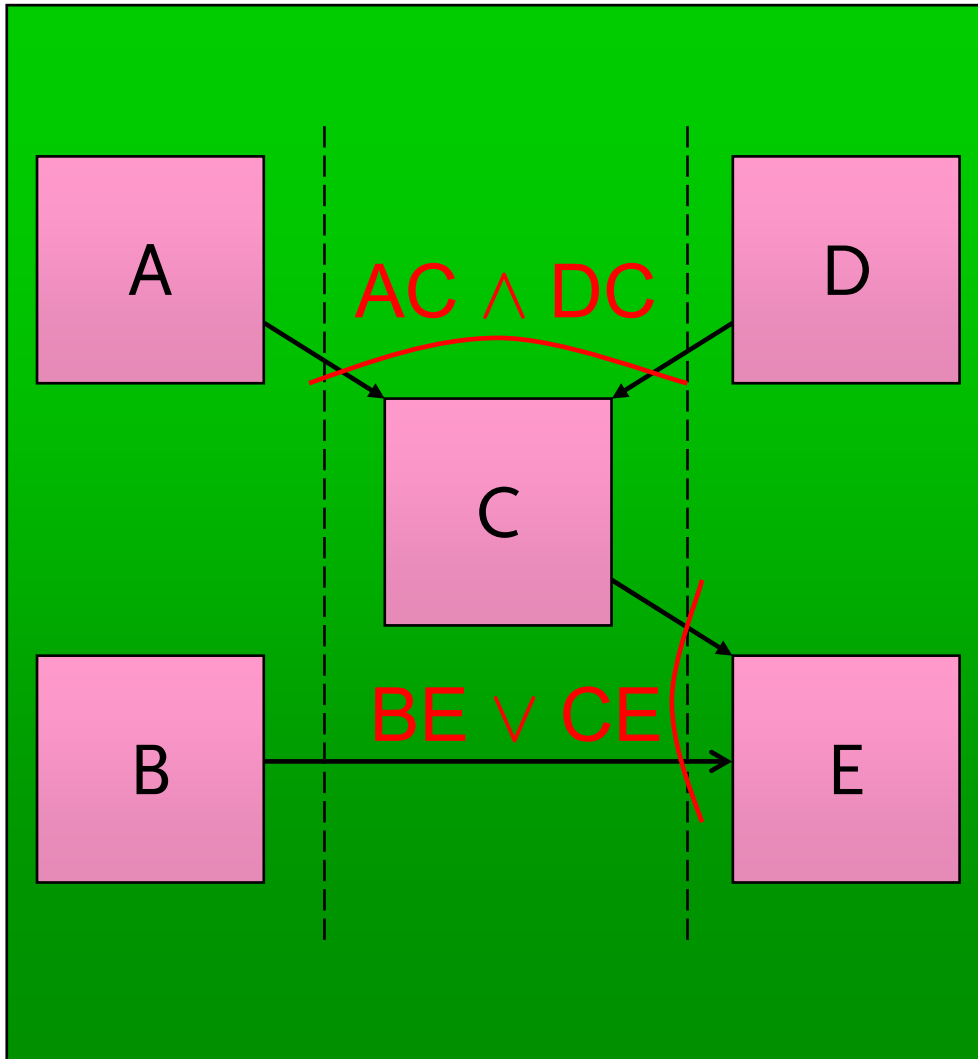
- Verklemmung, da $\text{Status(BC)} = \text{undef}$
- Lösung: $\text{Status(BC)} = F$
- C wird übersprungen
- $\text{Status(CD)} = F$
- DPE: Falsch zügig propagieren und so „tote Pfade“ auf der Engine zu eliminieren
- Auslöser sind switch und pick

Mehrere eingehende Links



- C startet, wenn
 - **Status** aller eingehenden Links definiert ist
 - **join condition** wahr ist
- **join condition** ist bool. Ausdruck
wahr: starte C
falsch: überspringe C
- **Status(Link)** ist Wert der transition condition der source-Aktivität

Anwendung von DPE



- Sei $\text{Status}(AC) = F$
- $AC \wedge DC = F$
- C wird übersprungen, setzt aber $\text{Status}(CE) = F$
- wenn $\text{Status}(BE) = W$, startet E

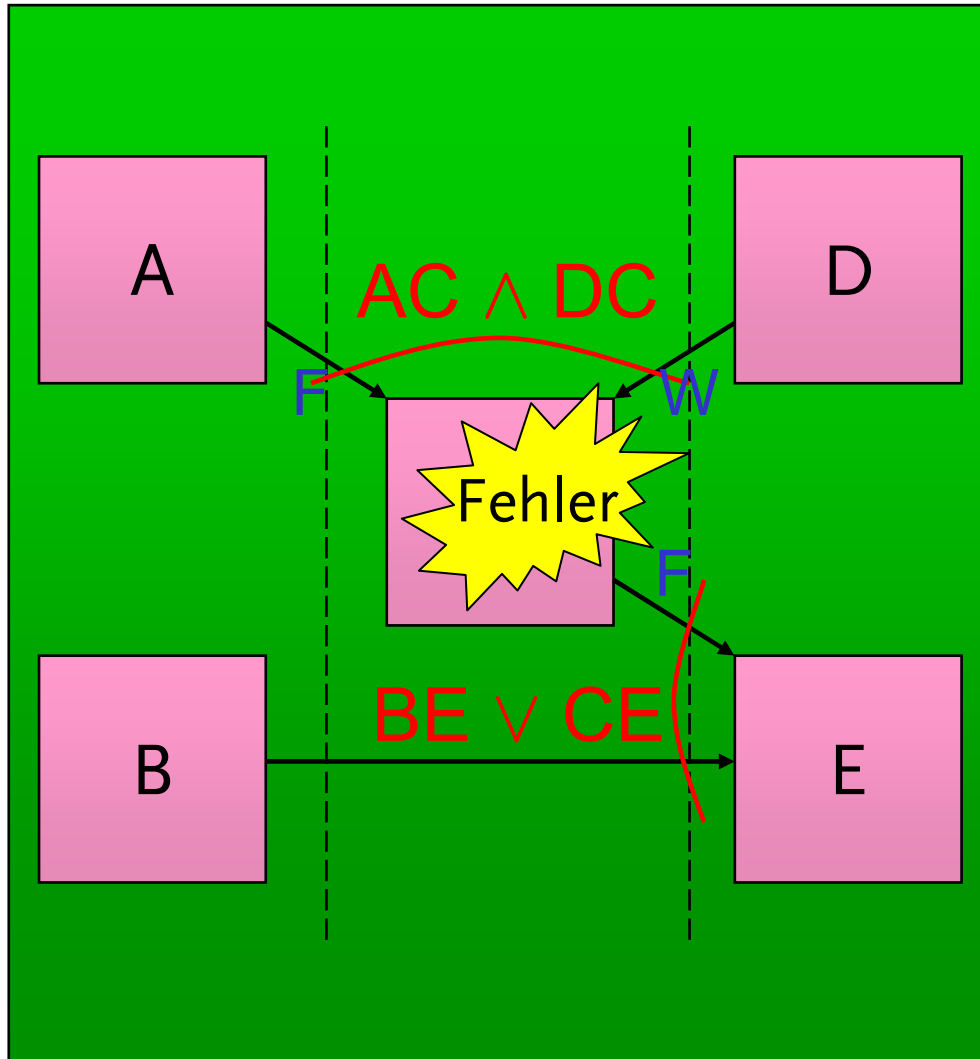
Bis jetzt

- Aktivität (source) abarbeiten
- Link-Status setzen (Wert der transition cond.)
- **join condition** auswerten
 - wertet Status aller eingehenden Links aus
 - z.B.: m aus n, 1 aus n (Standard), n aus n, ...
 - Wahr : target-Aktivität startet
 - Falsch : target-Aktivität überspringen, DPE
suppressJoinFailure = "yes"

Alles? Leider nein ...

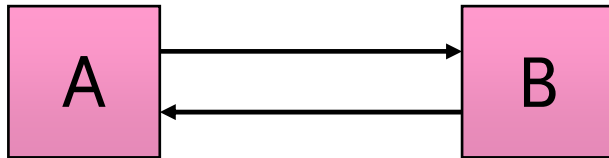
- Falsch : target-Aktivität *wirft Fehler*, DPE
suppressJoinFailure = "no"

SuppressJoinFailure = “no”

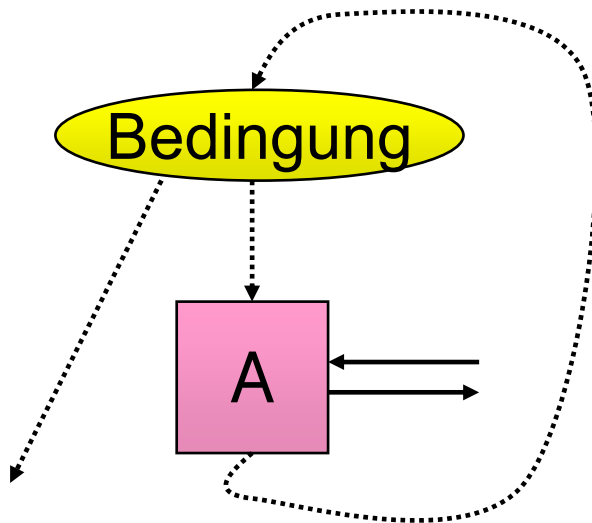


- Sei $\text{Status}(AC) = F$
- $AC \wedge DC = F$
- C wirft Fehler, setzt aber $\text{Status}(CE) = F$
- Fehlerbehandlung beendet gekapselten Bereich

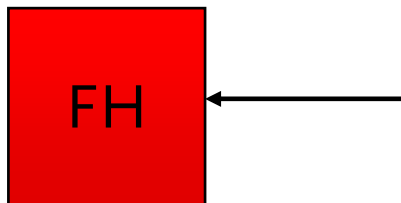
Einschränkungen für Links



Keine Zyklen



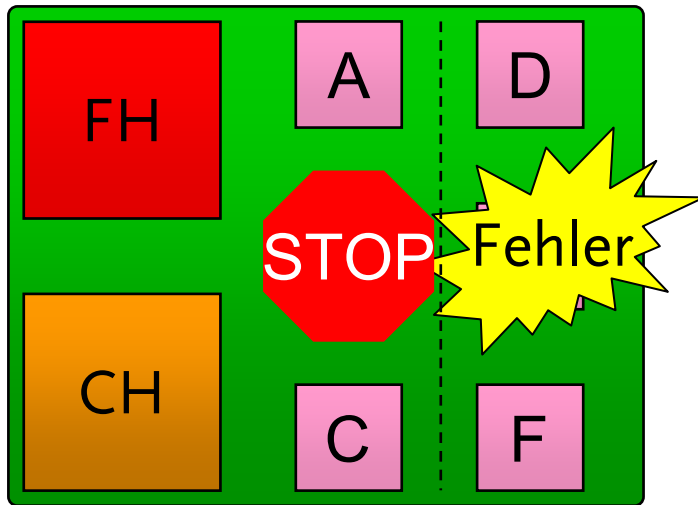
Grenzen des while nicht übertreten



FH ist nie target-Aktivität eines Link

Fehlerbehandlung

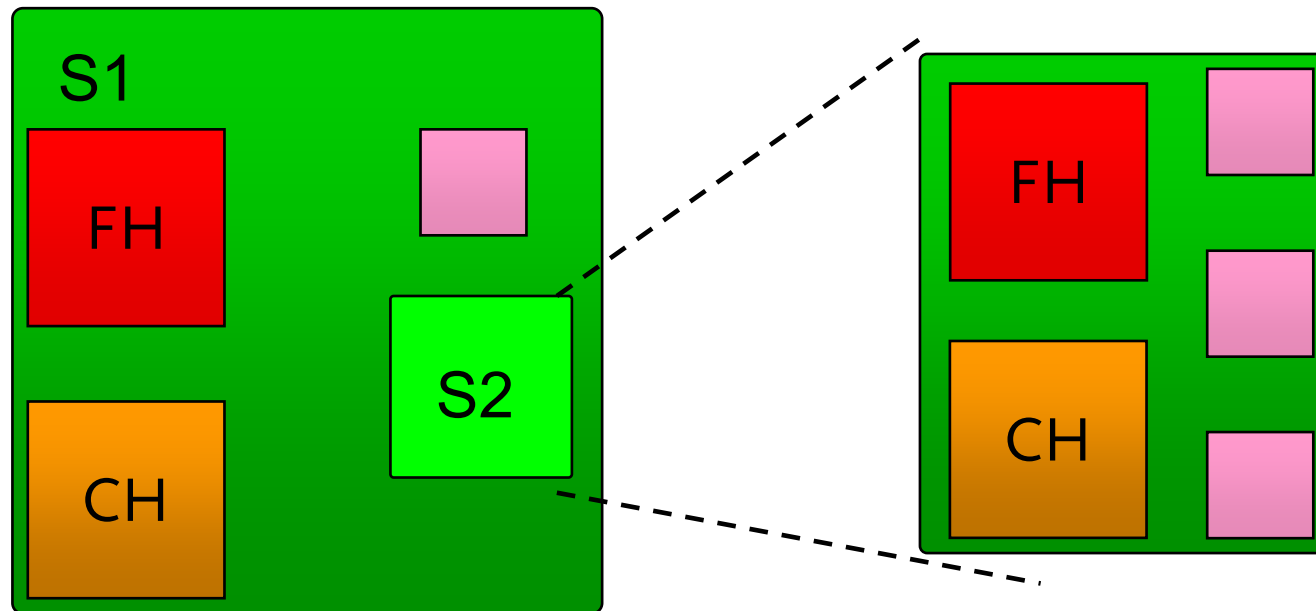
Sichtbarkeitsbereich



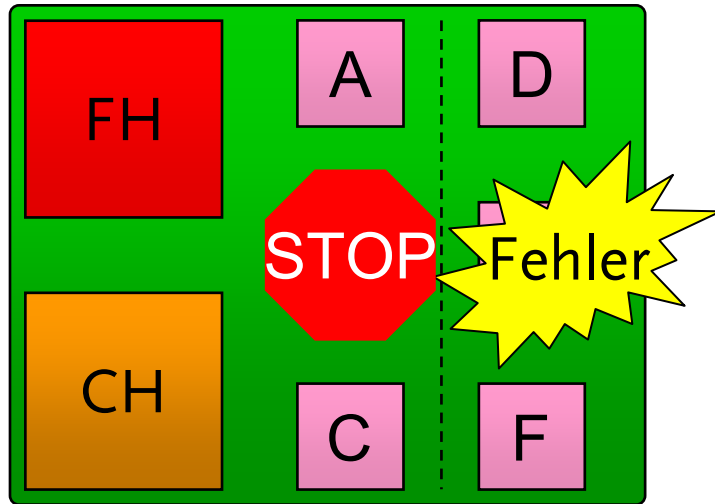
1. Beende alle Aktivitäten, die im Sichtbarkeits-bereich von E sind
2. FH behandelt den Fehler

Konventionen

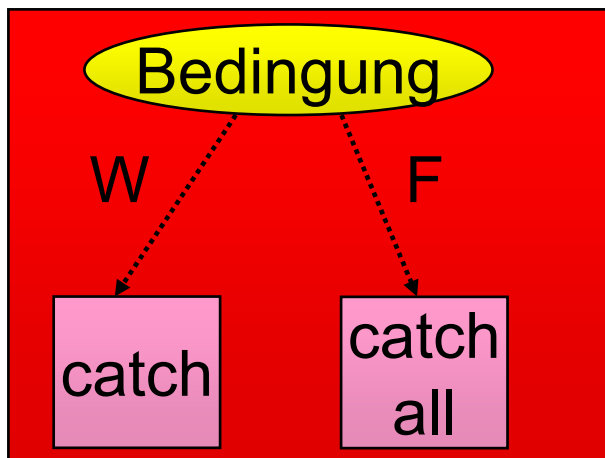
- Sichtbarkeitsbereich = Scope
- S1 ist *Elternscope* von S2
- S2 ist *Kindscope* von S1



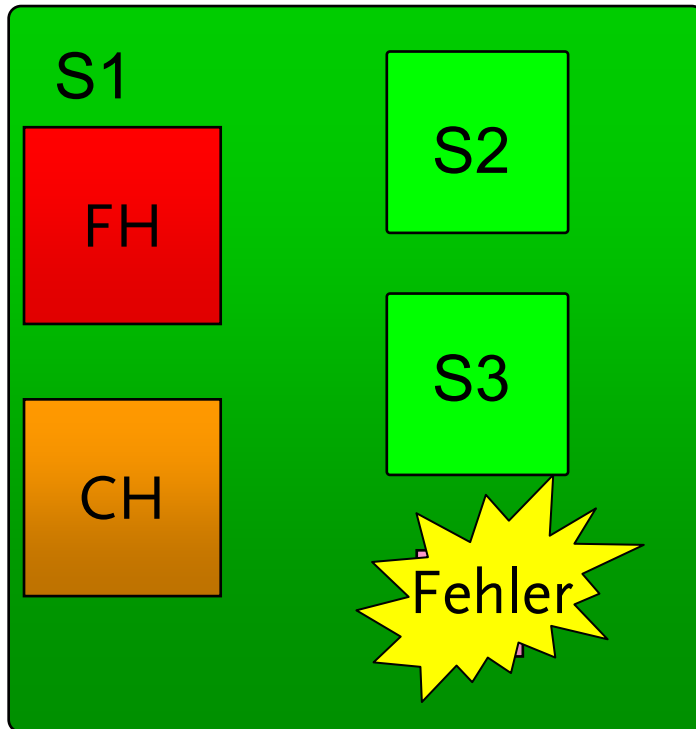
Nutzerdefinierte Fehlerbehandlung



- Fehlertyp über Namen, Fehlerdaten identifizierbar
- catch – behandelt ausgewählte Fehler
- catchAll – behandelt jeden Fehler
- nach Fehlerbehandlung wird Scope beendet
- kein passendes catch bzw. kein catchAll
→ Standard-FH wird ausgeführt

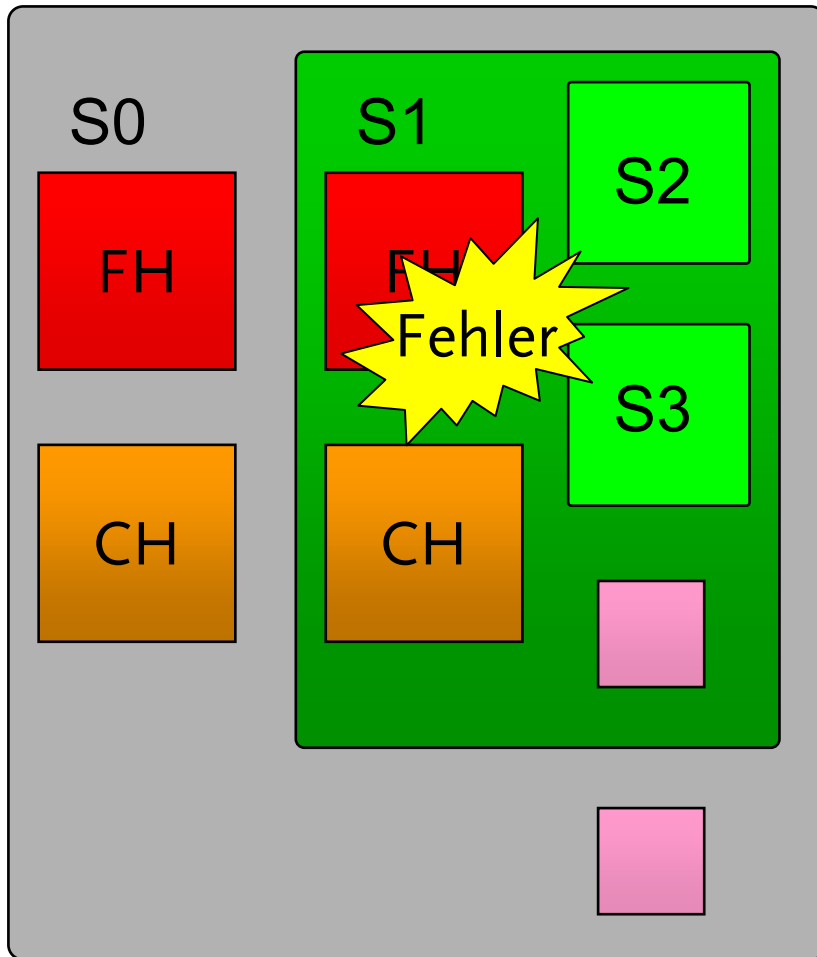


Standard-Fehlerbehandlung



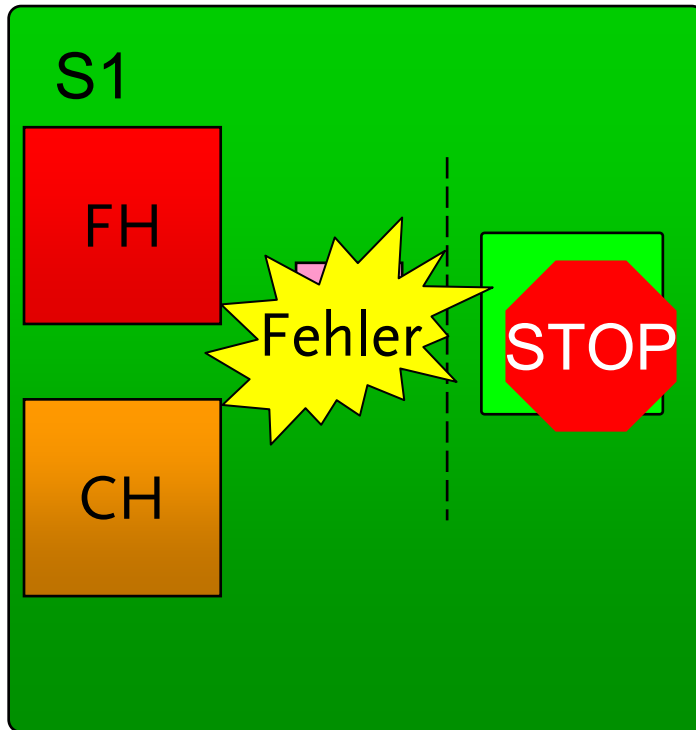
- Unabhängig vom Fehlertyp
- Rufe alle CH der Kindscores in umgekehrter Reihenfolge ihrer Abarbeitung auf
- Hier: CH(S3), CH(S2)
- S1 ist äußerster Scope
→ Ja: Beende Prozessinstanz

Standard-Fehlerbehandlung (Forts.)



- Unabhängig vom Fehlertyp
 - Rufe alle CH der Kindsopes in umgekehrter Reihenfolge ihrer Abarbeitung auf
 - Hier: CH(S3), CH(S2)
 - S1 ist äußerster scope
- Ja: Beende Prozessinstanz
- Nein: Reiche Fehler an FH des Elternscope

Beenden eines Scope



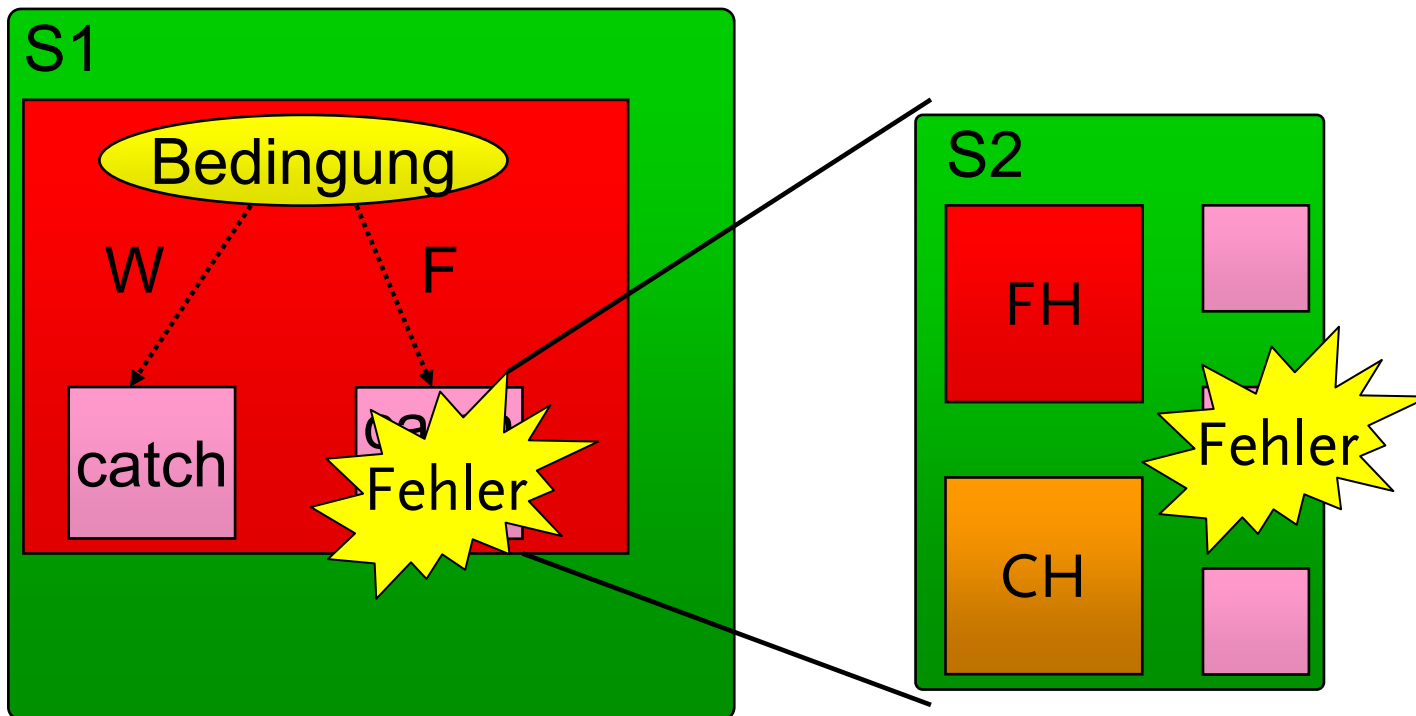
- A wirft Fehler
- S2 muss beendet werden
- forcedTermination-FH in S2 wird aktiviert
- Verhalten analog zu Standard-FH
 - Aktivitäten beenden
 - Kompensieren

Fehler bei der Fehlerbehandlung

- Fehler beim Kompensieren (Standard-FH)
→ später
- Aktivität im catch wirft Fehler (nutzerdef. FH)
→ 2 Fälle

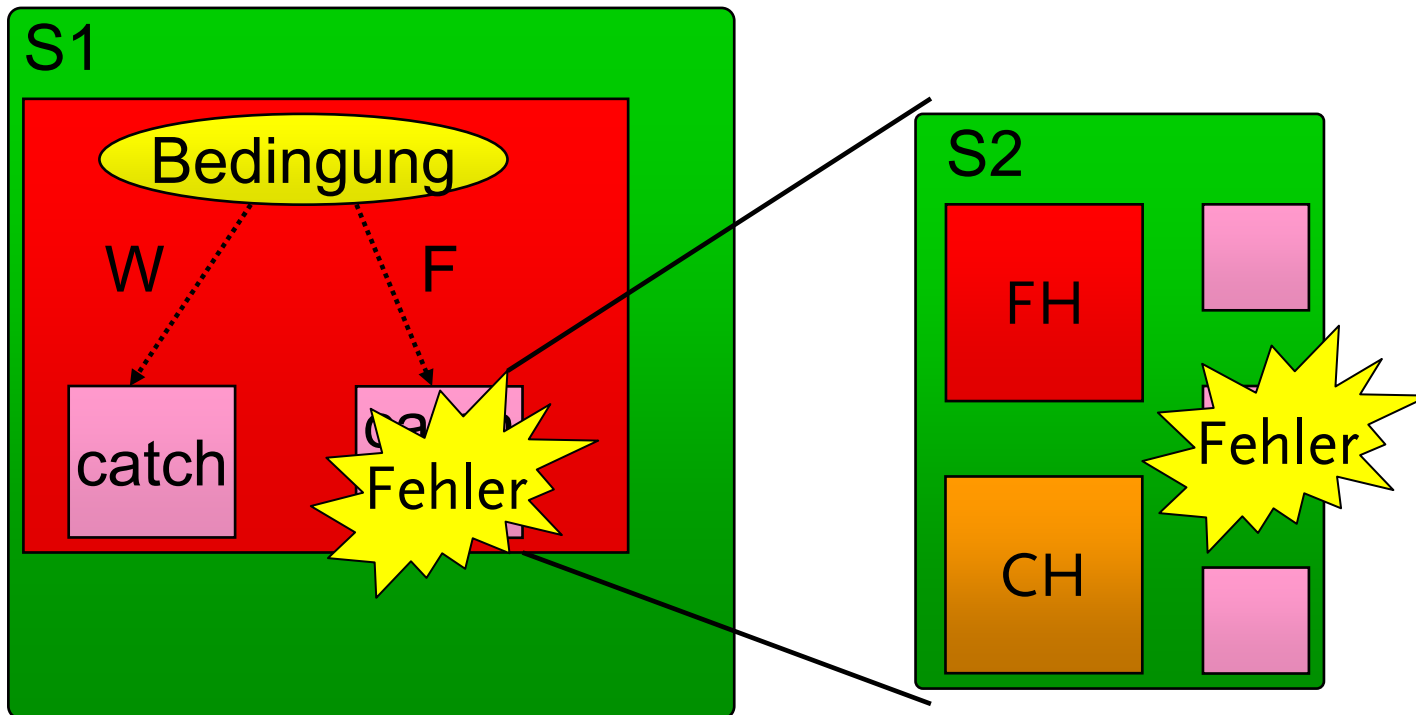
Fehler im catch (Fall 1a)

- FH von S2 behandelt den Fehler
- Fehler in S1 nicht sichtbar



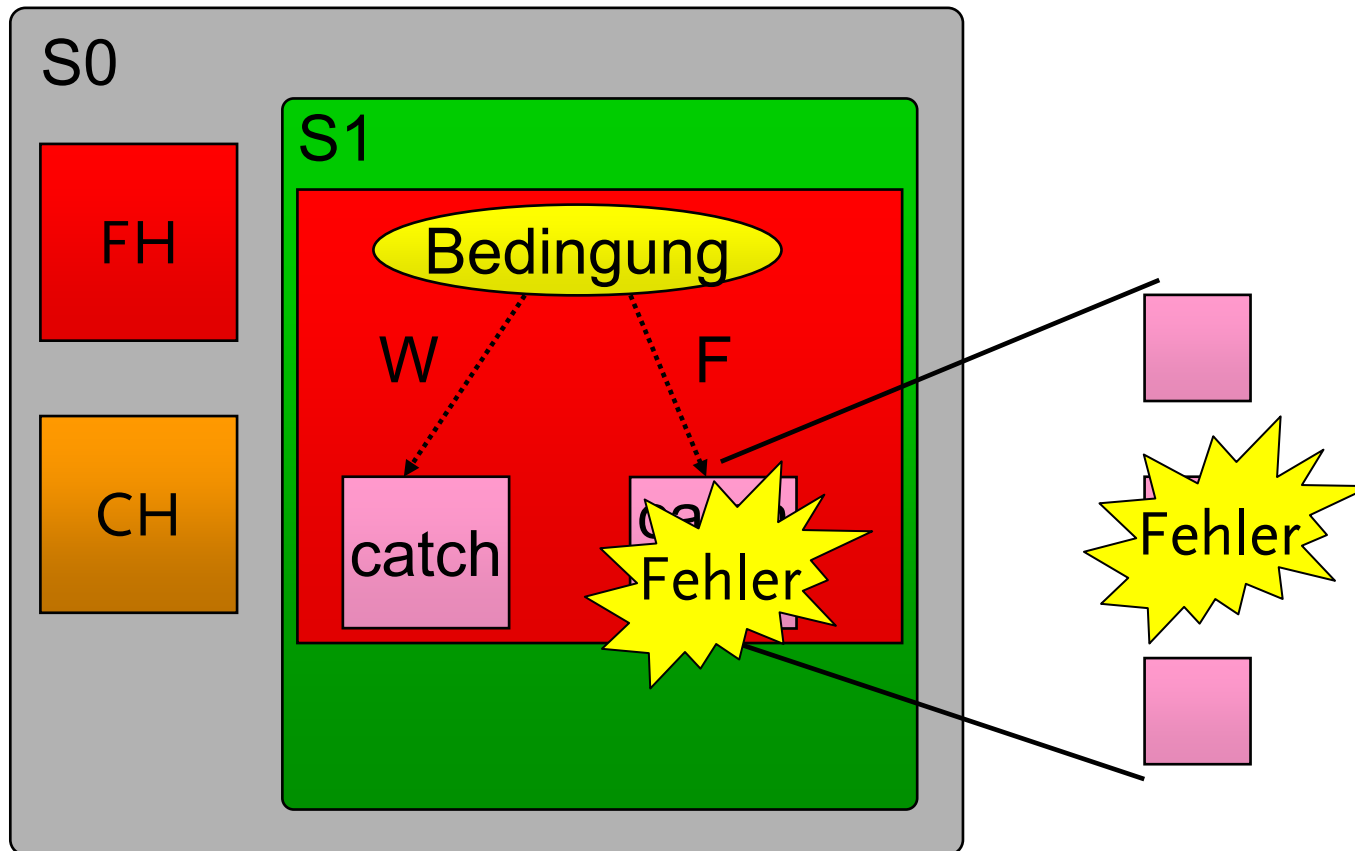
Fehler im catch (Fall 1b)

- FH von S2 behandelt den Fehler *nicht*
- Beende Prozessinstanz



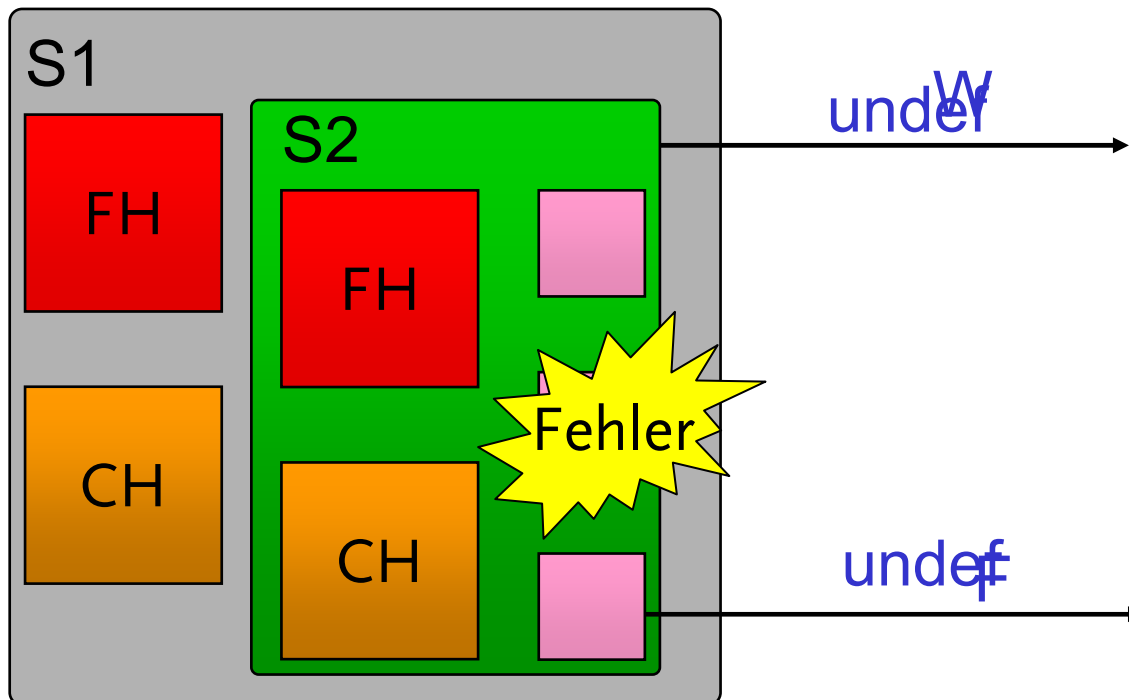
Fehler im catch (Fall 2)

- Reiche Fehler an FH von S0



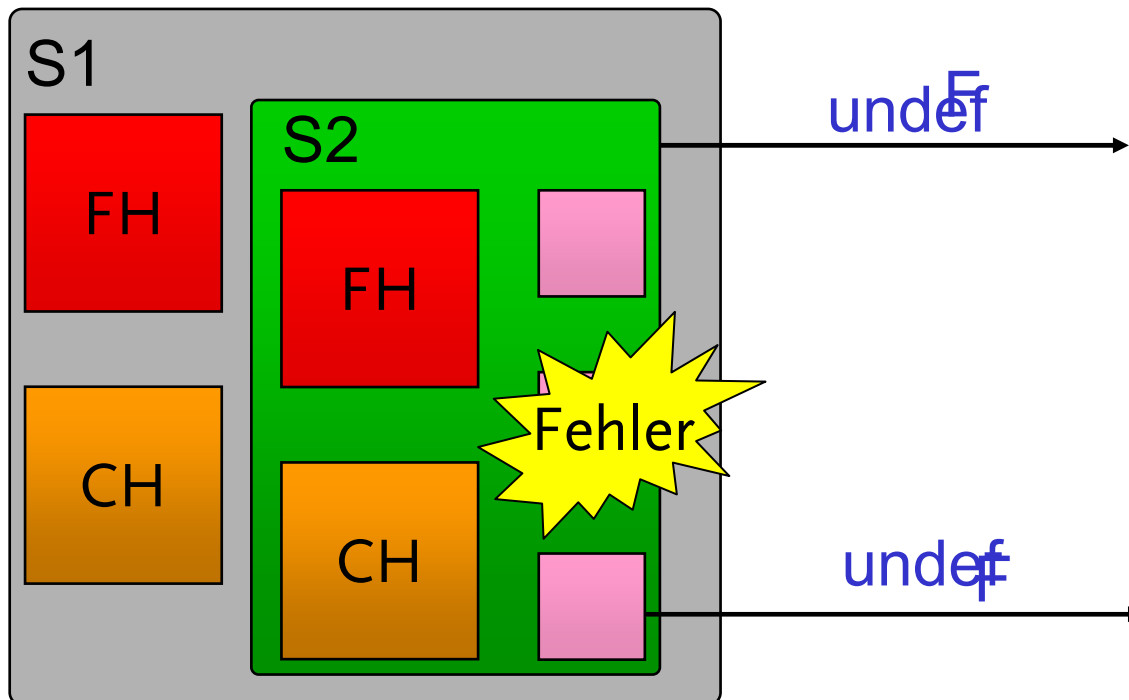
Links und Fehlerbehandlung

- Fehlerbehandlung erfolgreich
 - Fehler nicht an FH von S1 weiterreichen
 - Setze Status aller ausgehenden Links von S2 auf W
 - Setze Status aller ausgehenden Links aller nicht abgearbeiteten Aktivitäten von S2 auf F



Links und Fehlerbehandlung

- Fehlerbehandlung *nicht* erfolgreich
 - Fehler an FH von S1 weiterreichen
 - Setze Status aller ausgehenden Links von S2 auf F
 - Setze Status aller ausgehenden Links aller nicht abgearbeiteten Aktivitäten von S2 auf F



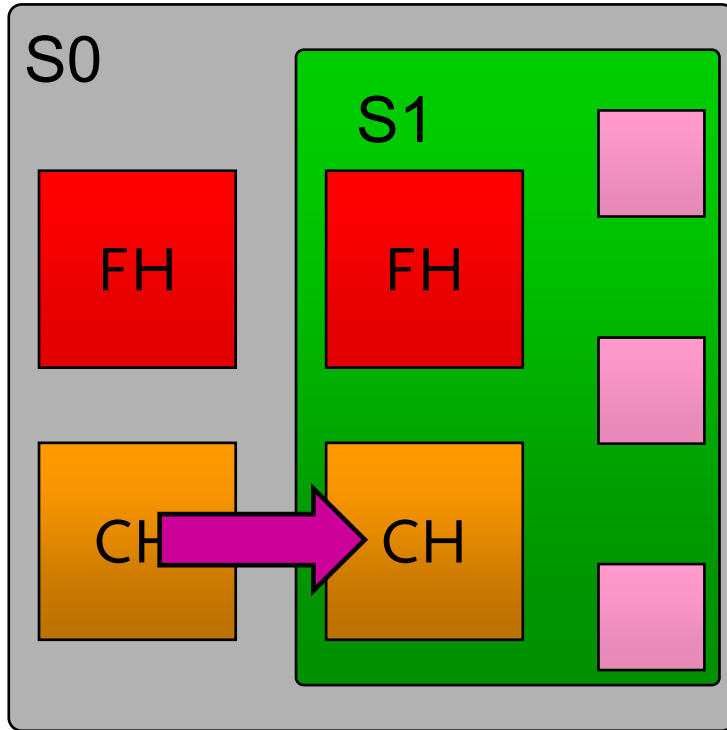
Kompensation von Scopes

- Compensation Handler (CH)
 - in v1.1 nur „Hülle“
 - Effekte ausgeführter Aktivitäten rückgängig machen
 - Sieht „Schnappschuss“ aller geänderten Variablen des Scope

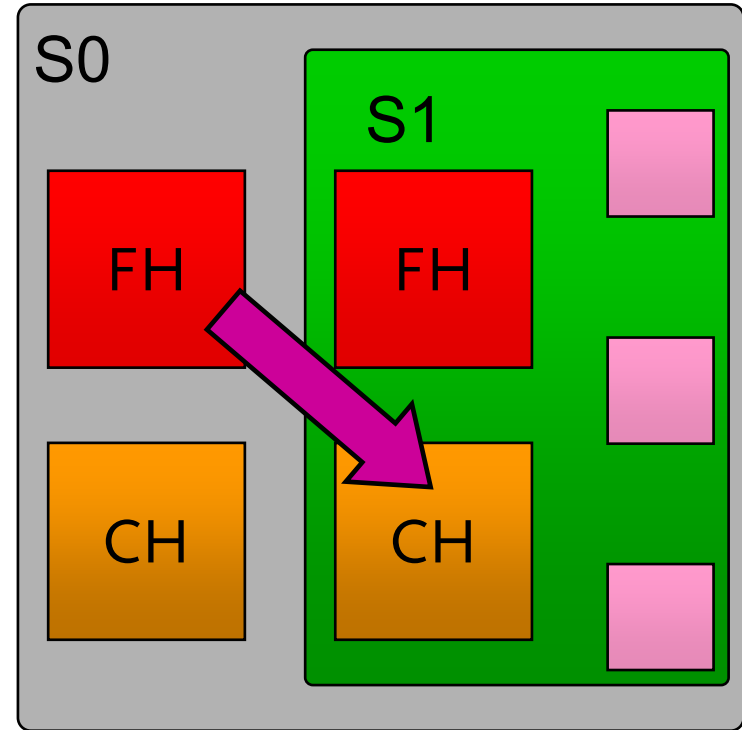
1. Wann wird kompensiert?

- Umgebender Scope vollständig und fehlerfrei abgearbeitet
 - andernfalls macht CH nichts
- FH des Scope war nicht aktiv
- Scope wird nur einmal kompensiert

2. Wie wird der CH aufgerufen?



- CH des Elternscope



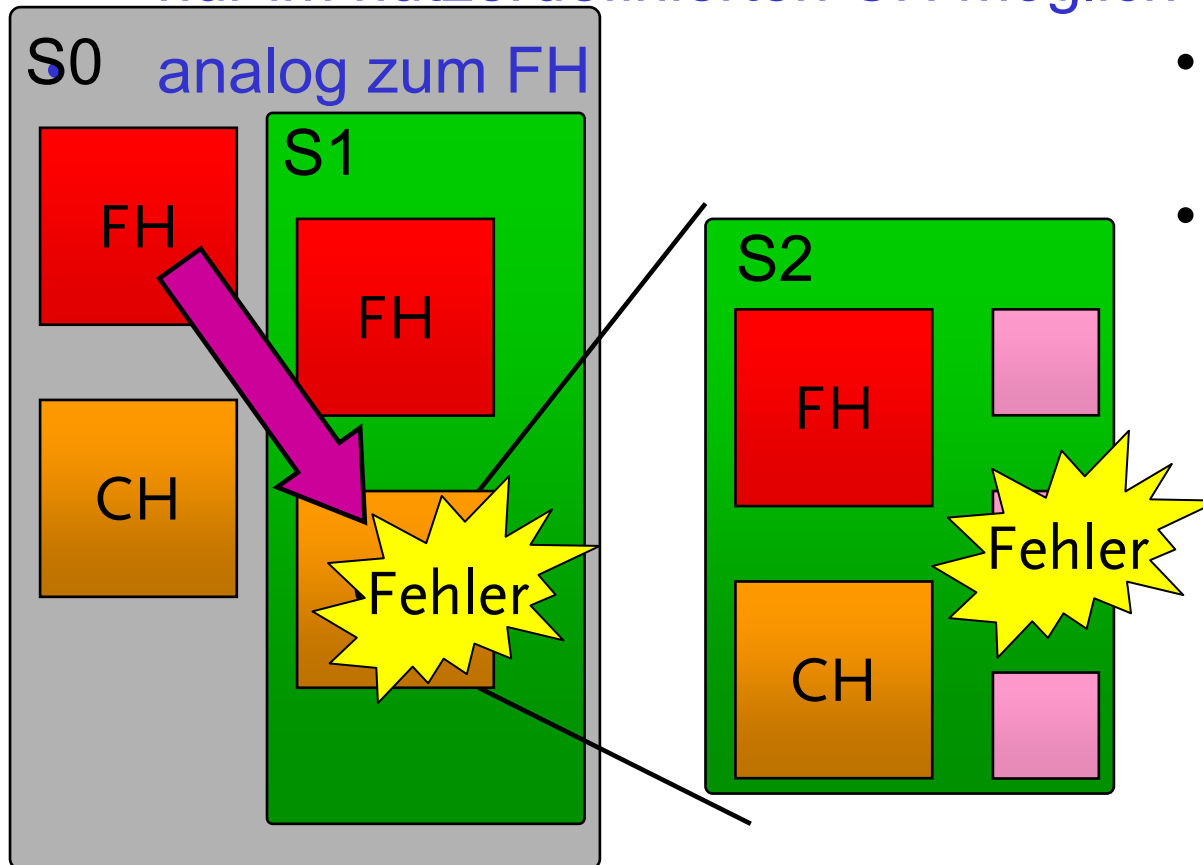
- Standard-FH des Elternscope
- nutzerdefinierter FH des Elternscope

3. Arten der Kompensation?

- Standard-CH
 - analog zur Standard-Fehlerbehandlung
 - rufe alle CH der Kindscope in umgekehrter Reihenfolge ihrer Abarbeitung auf
- Nutzerdefinierte Kompensation
 - spezifiziert Aktivität
 - Möglichkeit CH von Kindscope direkt aufzurufen (Aktivität compensate)

4. Fehler beim Kompensieren? (1)

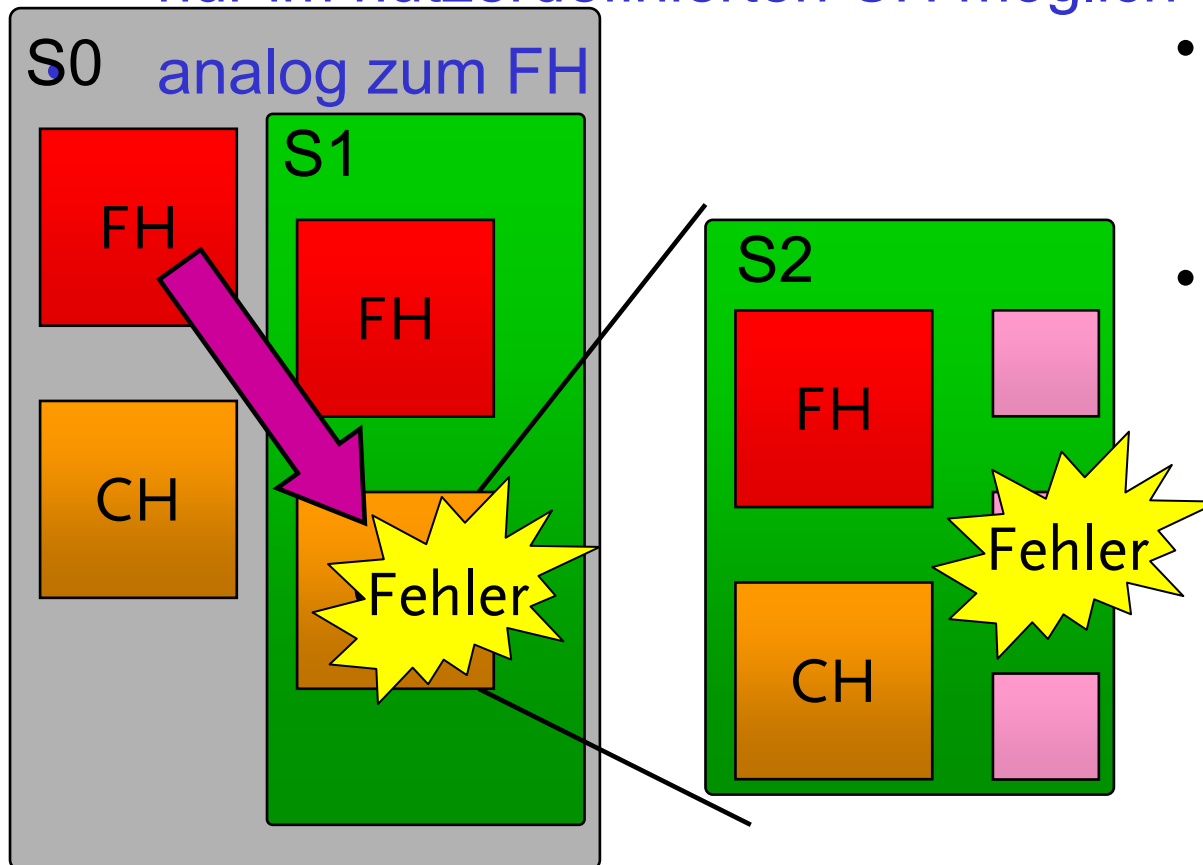
- nur im nutzerdefinierten CH möglich



- FH von S2 behandelt Fehler
- Kontrollfluss geht in FH von S0 weiter

4. Fehler beim Kompensieren? (2)

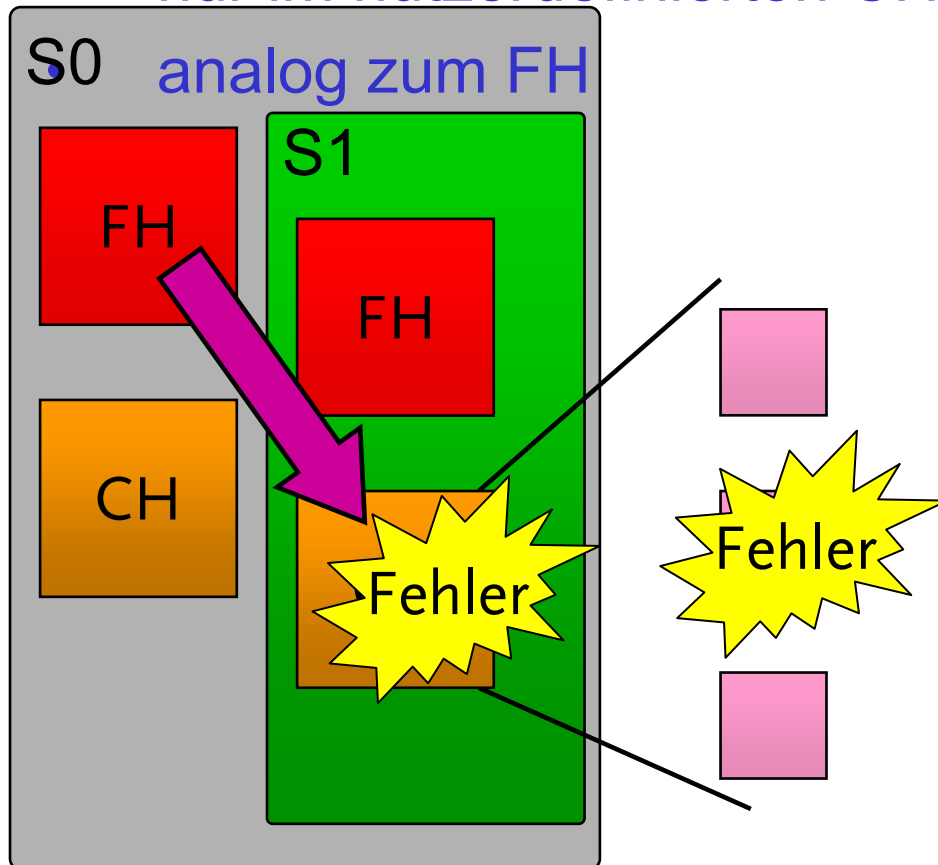
- nur im nutzerdefinierten CH möglich



- FH von S2 behandelt Fehler *nicht*
- Fehler an FH von S0 gereicht

4. Fehler beim Kompensieren? (3)

- nur im nutzerdefinierten CH möglich

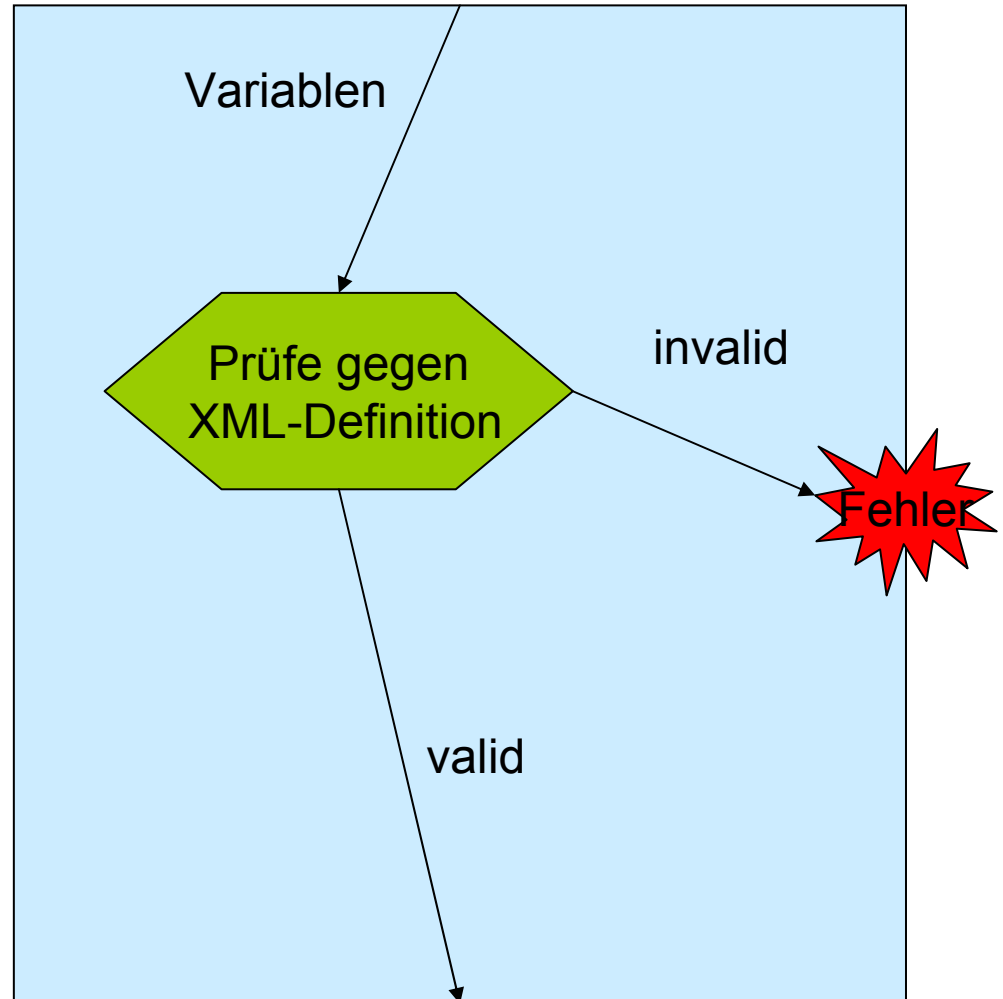


- Fehler an FH von S0 gereicht

WS-BPEL v2.0

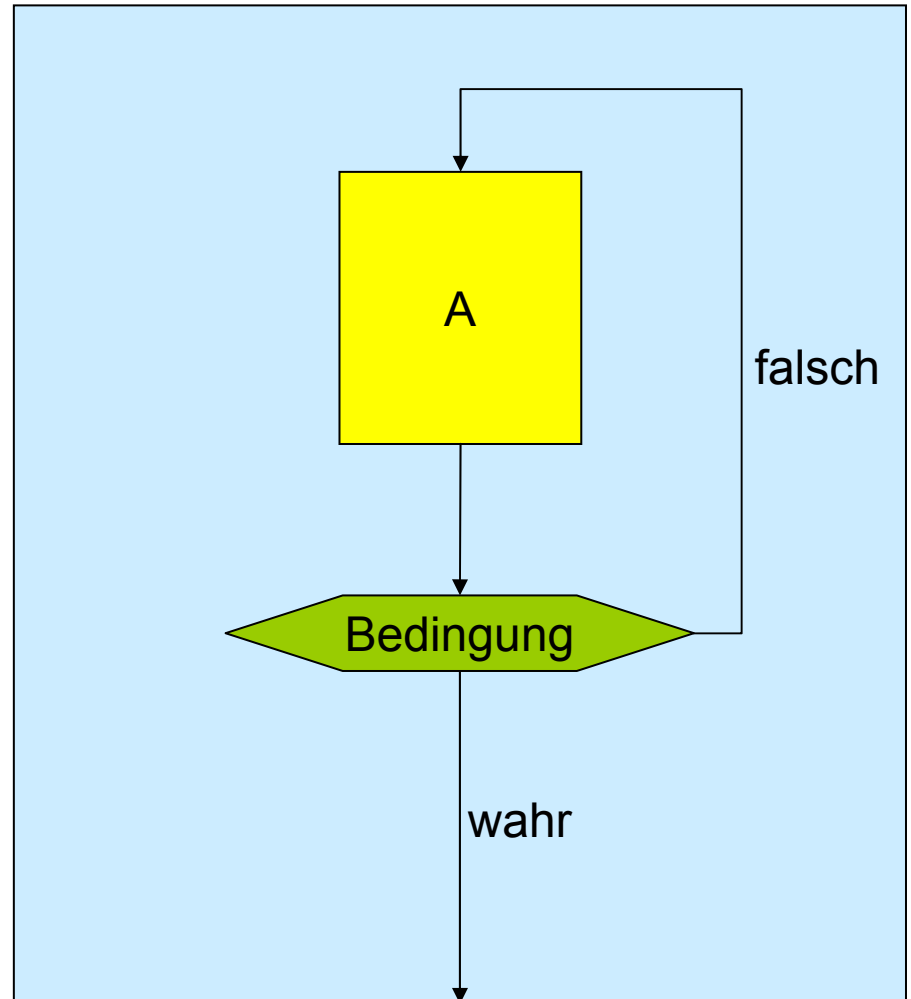
Neue Aktivitäten

- validate



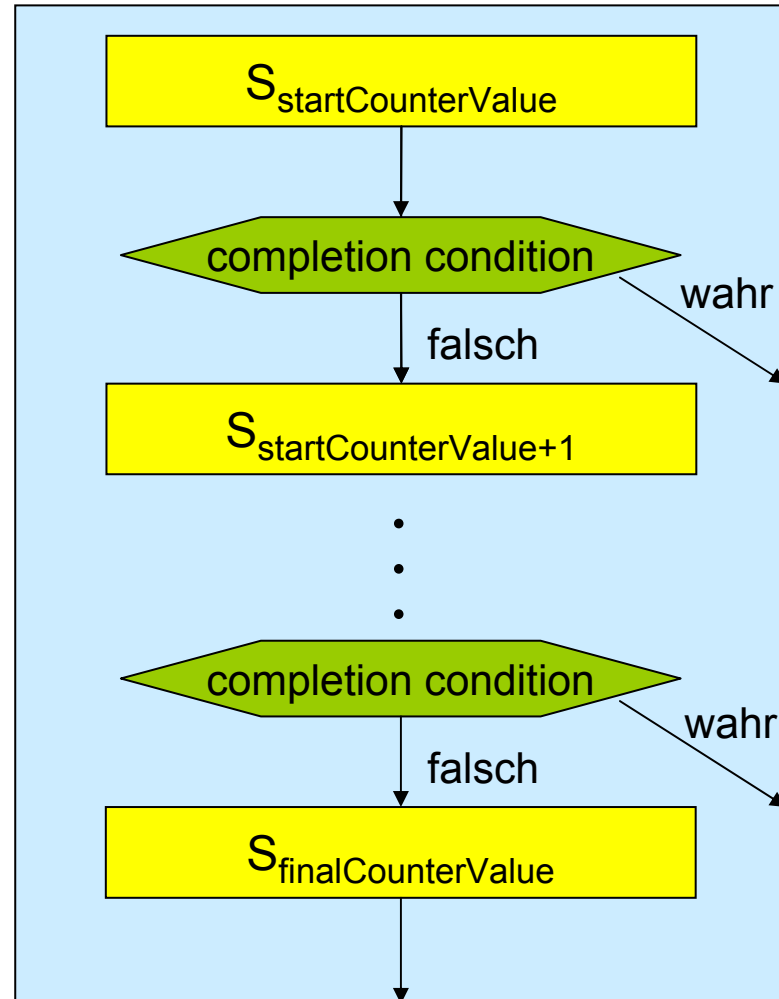
Neue Aktivitäten

- validate
- repeatUntil



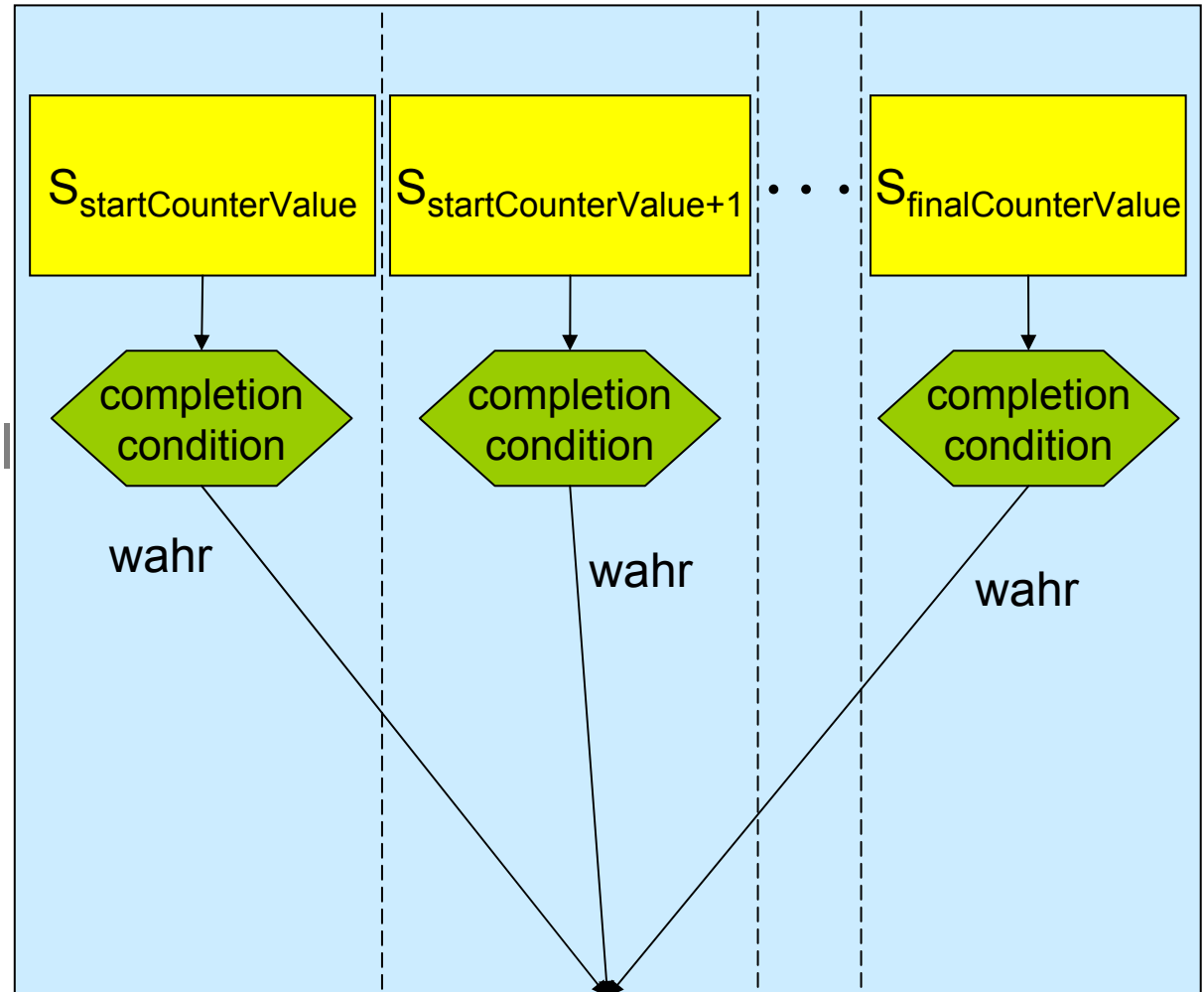
Neue Aktivitäten

- validate
- repeatUntil
- **forEach**
 - sequentiell



Neue Aktivitäten

- validate
- repeatUntil
- **forEach**
 - sequentiell
 - **parallel**



Neue Aktivitäten

- validate
- repeatUntil
- forEach
 - sequentiell
 - parallel
- rethrow

Executable vs. Abstract Process

- Ausführbare Prozesse:
 - konkrete Implementation von Geschäftsprozessen
 - ein ausführbarer BPEL-Prozess beschreibt einen Geschäftsprozess
- Abstrakte Prozesse:
 - abstrakte Sicht auf Geschäftsprozesse
 - beschreiben Klassen von BPEL-Prozessen
 - *Abstract Process = Common Base + Profile*

Common Base abstrakter Prozesse

- beschreibt syntaktisches Universum abstrakter Prozesse:
 - WS-BPEL XML Schema ist verbindlich
 - alle Konstrukte aus ausführbaren Prozessen sind erlaubt
 - plus zusätzliche explizite und implizite Konstrukte zum Verstecken von Informationen
(*opaque language extensions, omission*)
 - „createInstance“-Aktivität muss nicht spezifiziert sein
 - *abstractProcessProfile*-Attribut muss gesetzt sein

Opaque-Konstrukte

- *<opaqueActivity>*:
expliziter Platzhalter für ausgelassene Aktivitäten
- *Opaque Expressions*:
z.B. Verstecken der Entscheidungslogik

```
<if ... >  
  <condition opaque=„yes“/>  
  <then> ... </then>  
</if>
```

- *Opaque Assignments*:
Zuweisen versteckter Werte (bzw. Nicht-Determinismus)

```
<from opaque=„yes“/>
```

- *Opaque Attributes*: `attribut=„##BPELOpaque“`

Omission-Shortcut

- Alternativ zur Verwendung von *opaque* können Elemente eines BPEL-Prozesses weggelassen werden, wenn:
 - sie syntaktisch erforderlich sind
 - und keinen default-Wert besitzen, z.B.

```
<validate />
```

- Äquivalent zu:

```
<validate variables=„##BPELopaque“ />
```


Abstract Process Profile

- ermöglicht Interpretation des abstrakten Prozesses
- definiert, wie der abstrakte Prozess zu einem ausführbaren Prozess vervollständigt werden kann:
 - Dürfen Aktivitäten hinzugefügt werden?
(z.B. Erweiterung eines abstrakten Kommunikationsschemas)
 - Wenn ja, welche?
(z.B. Verwendung von Links verbieten)
 - Oder dürfen nur *Opaque Token* ersetzt werden?
(z.B. in Template-Profilen)
 - Welche *Opaque Token* sind überhaupt zulässig?
(z.B. joinConditions mit opaque=„yes“ verbieten)

WS-BPEL Standard Profile

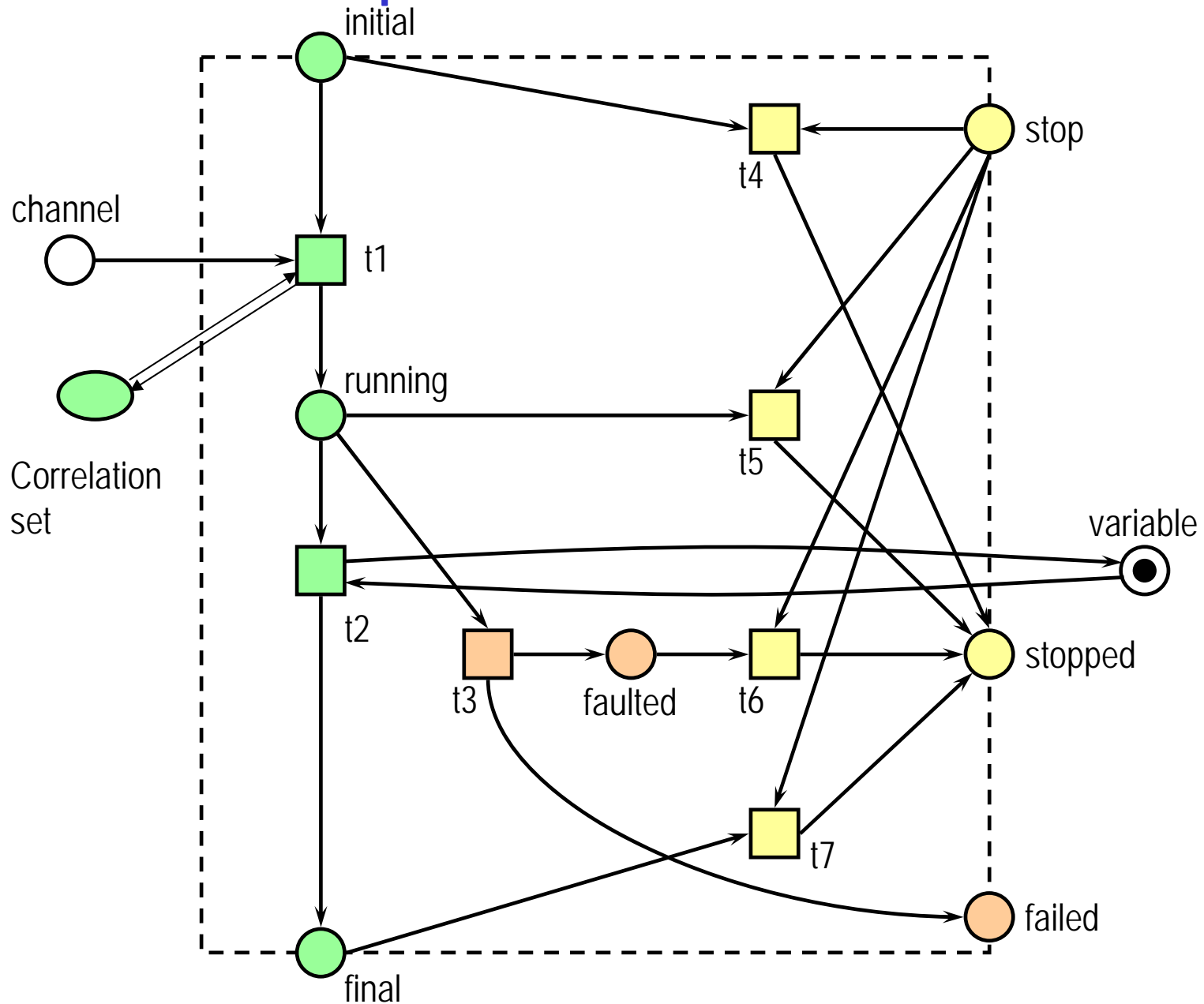
- Abstract Process Profile for Observable Behavior:
 - Modellierung des Interaktionsverhaltens eines Prozesses
 - Abstrahiert von Datenmanipulationen (Nicht-Determinismus)
 - Nur Elemente die sich auf Daten beziehen dürfen *opaque* sein
(z.B. *assign*, *variable-Attribute*, nicht jedoch z.B. *joinCondition*)
- Abstract Process Template Profile:
 - Abstrahiert von Details der Ausführung (z.B. *endpoints*)
 - Nur explizite Opacity erlaubt (*kein Omission-Shortcut*)
 - Executable Prozess darf (fast) keine Aktivitäten hinzufügen
 - Alle Startaktivitäten müssen ausgezeichnet sein
(*createInstance*)
 - *<documentation>*-Tags für Erläuterungen des Designers

Übersetzung von BPEL in Petrinetze

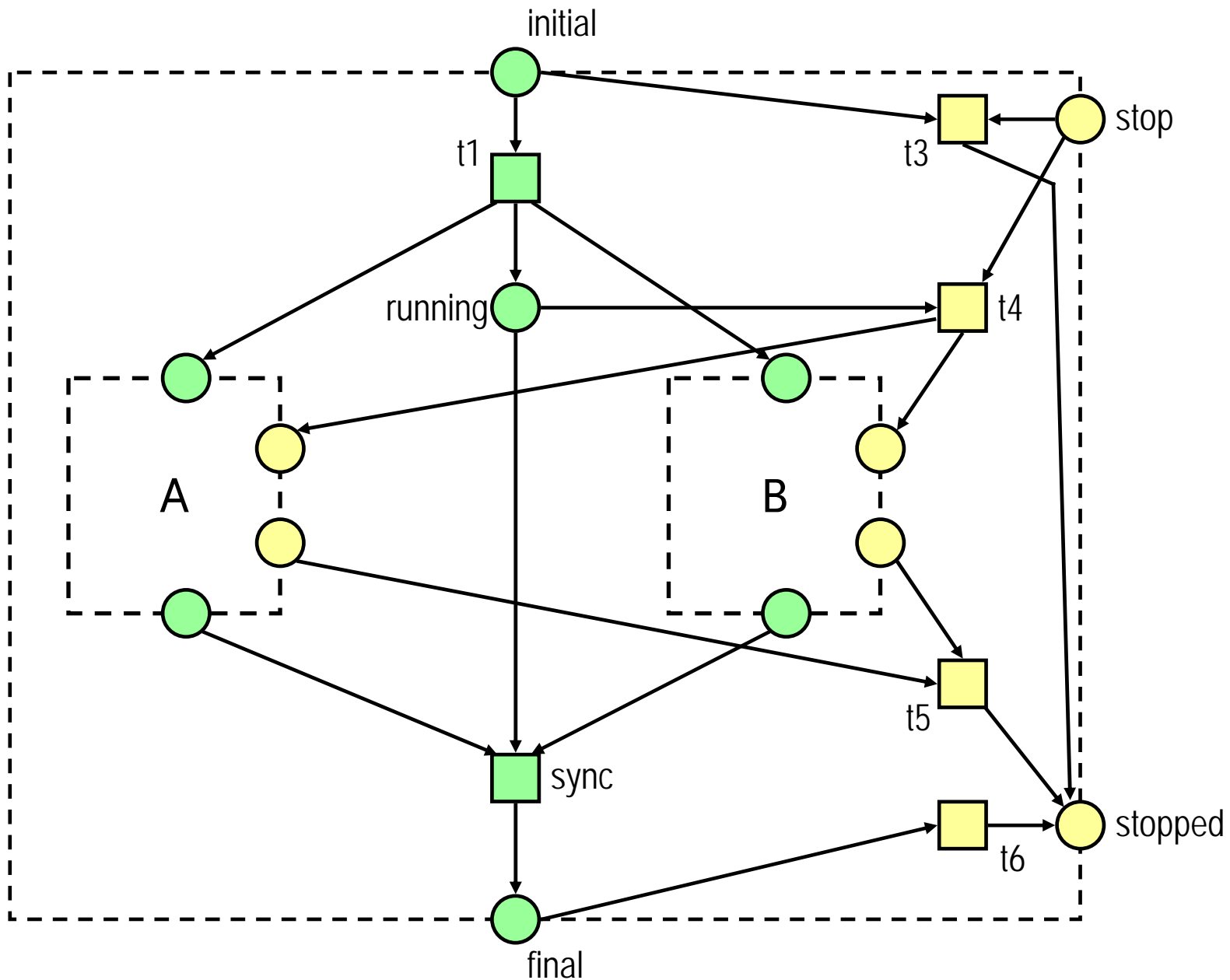
Ziele:

1. Formale Semantik geben (Schwächen in der Spezifikation)
2. Anwendung formaler Methoden ermöglichen

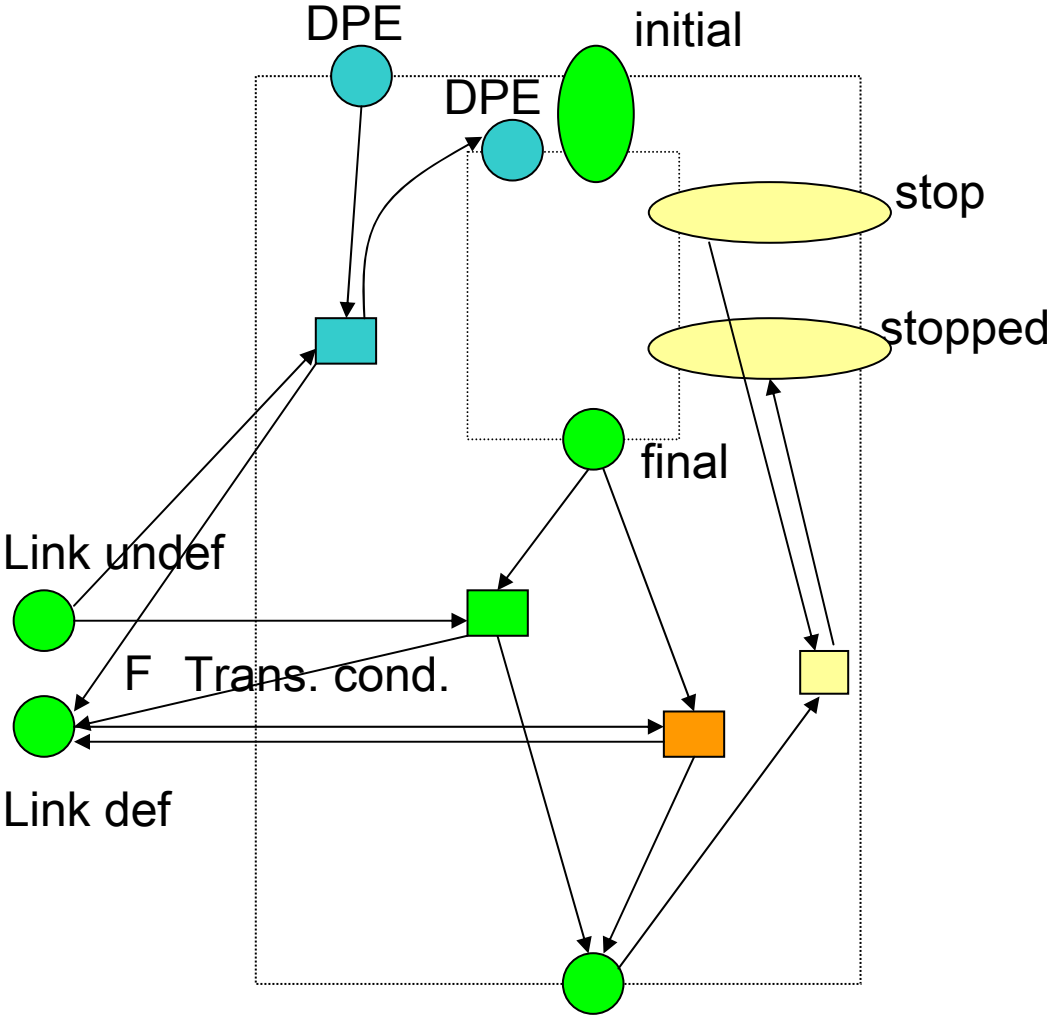
Beispiel einer Basisaktivität: Receive



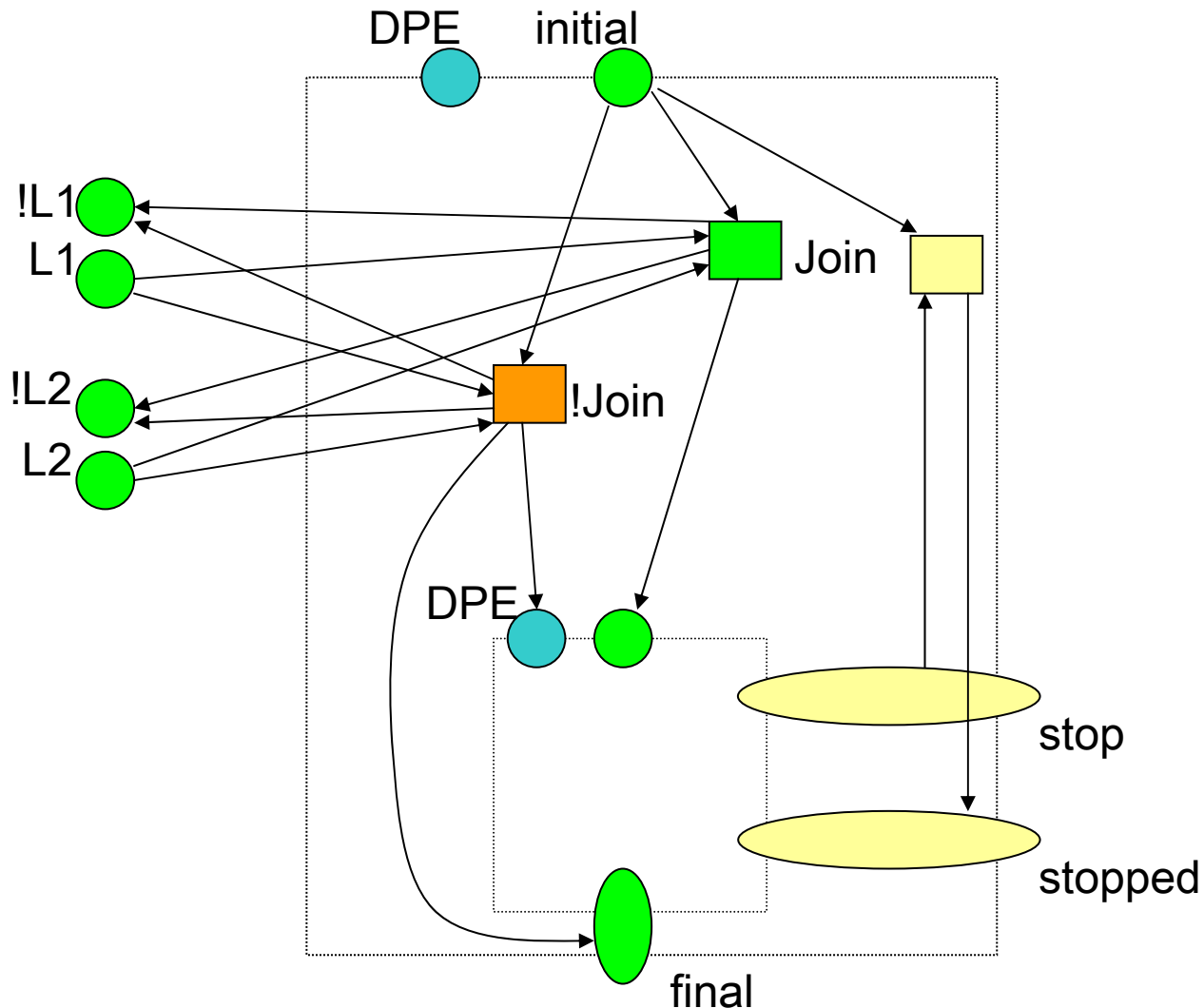
Semantik einer strukturierten Aktivität: Flow



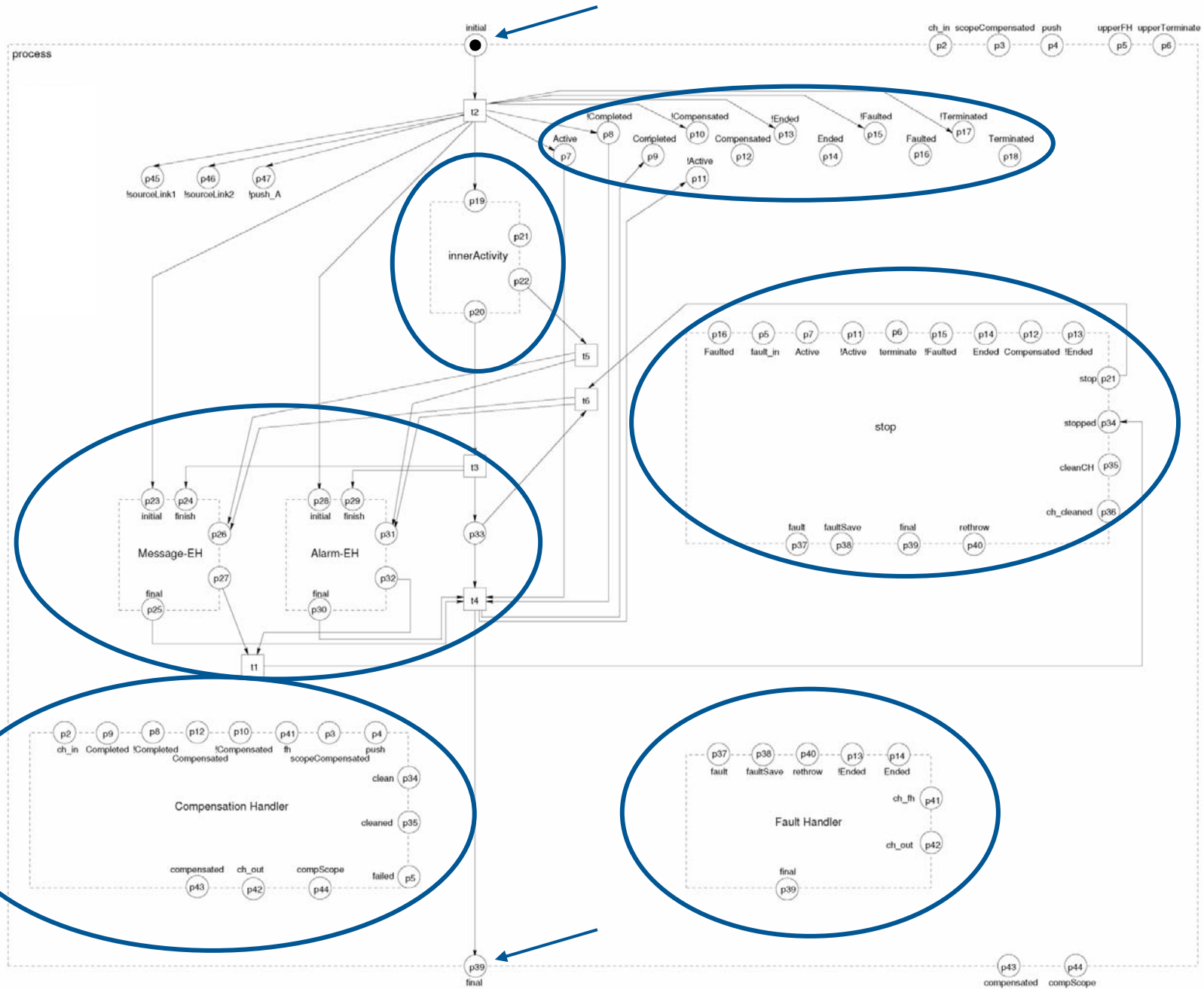
Links (Quelle)



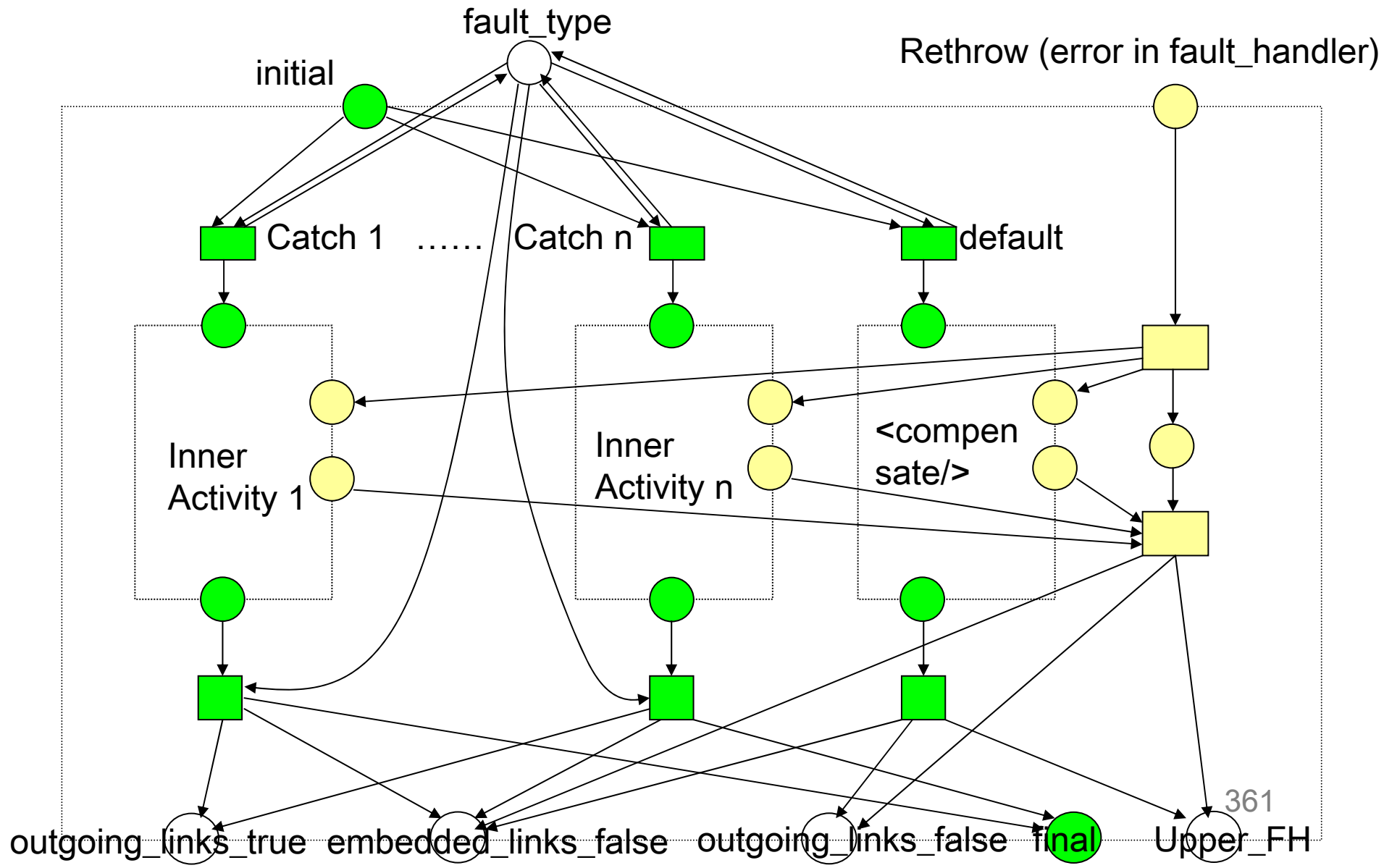
Links (Ziel, join failure suppressed)



Scope



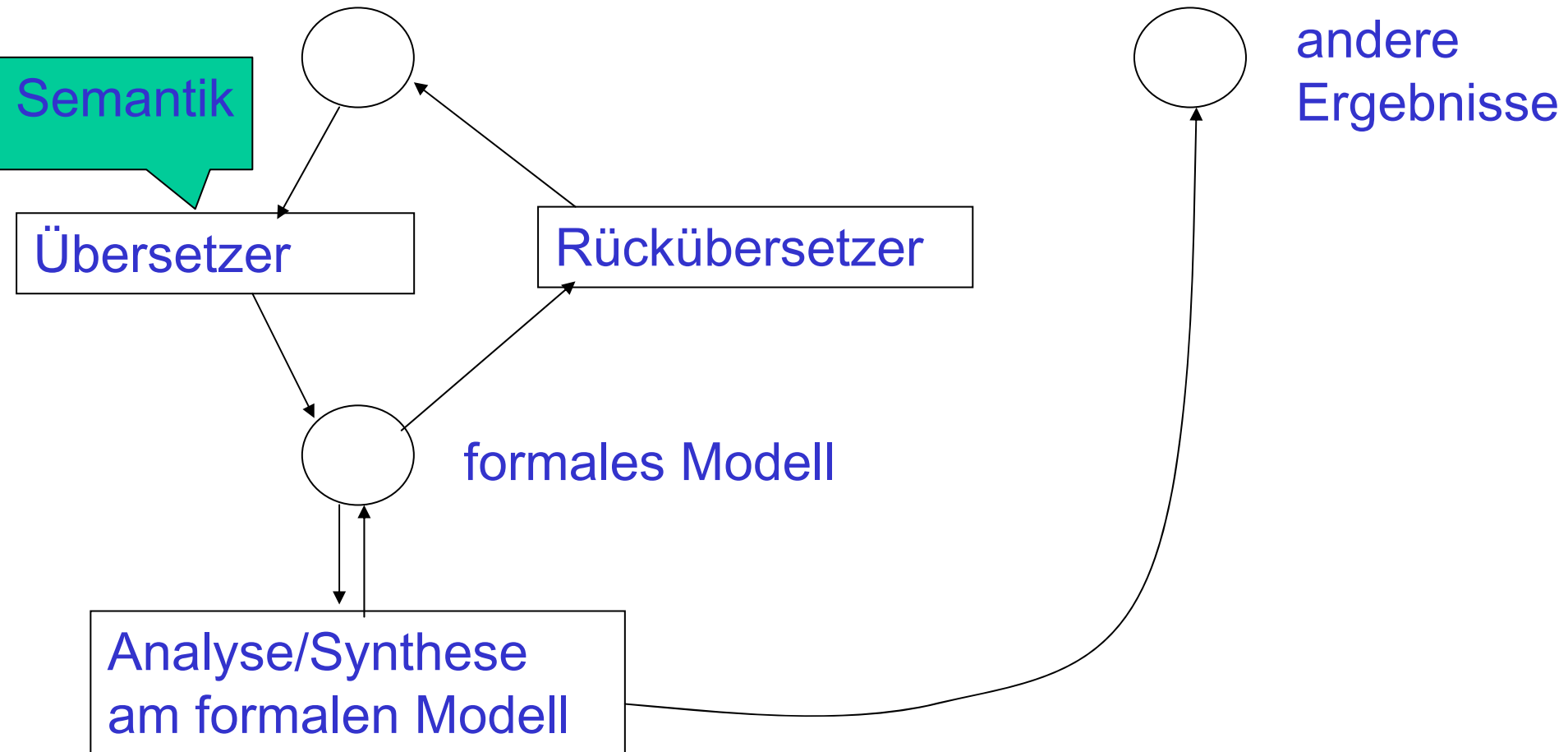
Default Fault Handler



Interessante Fragestellungen an das Verhalten von Services

Generelles Bild

BPEL-Prozess



Passende Services

- (1) Interfaces match (Typing, Semantic web, Standards)
- (2) Kontrollfluss matches (keine Deadlocks, Livelocks,...)
- (3) Semantik matches (Semantic Web, Standards)
- (4) Quality of Service matches

Problem 1: Compatibility



Geg: Services P, R

Ist $P+R$ deadlock-frei bzw. livelock-frei?

Problem 2: Controllability

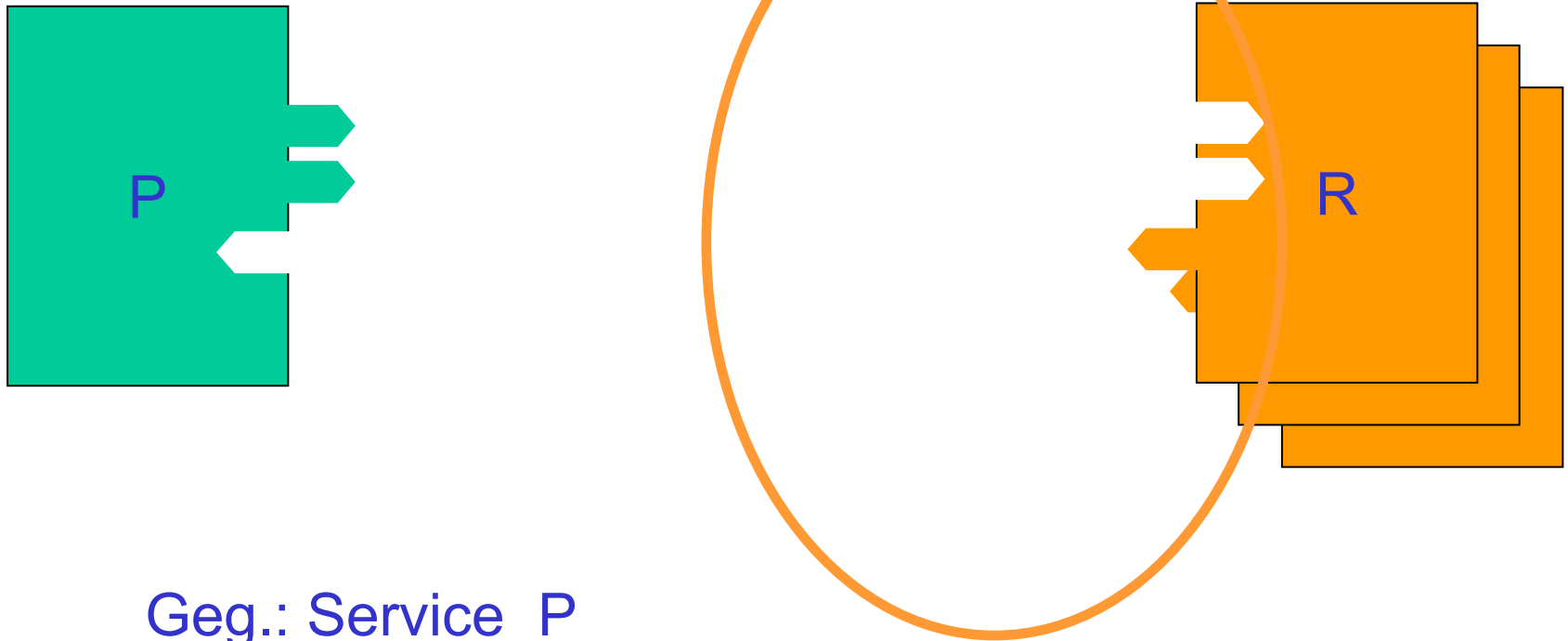


Geg.: Service P

Ex. einn R so dass $P+R$ deadlock-frei bzw. livelock-frei?

Eigentlicher Wert: Partnersynthese

Problem 3: Characterization



Charakterisiere alle R, für die P+R deadlock-frei bzw. livelock-frei!

Wertvolle Zutat: **Synthesierter Partner**

Eigentlicher Wert: **Bedienungsanleitung**

Problem 4: Exchangeability



Geg.: Services P, P'

Ist jeder Partner von P auch ein Partner von P' ?

Wertvolles Werkzeug: **Bedienungsanleitung**

Problem 5: Runtime Exchangeability

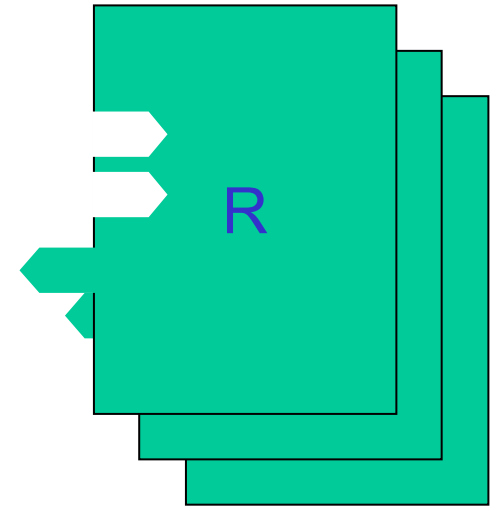
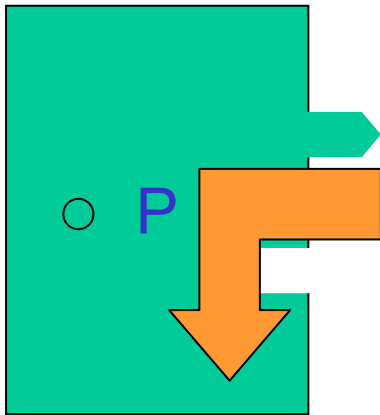


Geg.: Services P, P'

Ex. \rightarrow so dass P austauschbar ist mit $P+P'+\rightarrow$

Wertvolles Werkzeug: Bedienungsanleitung

Problem 6: Migration



Geg.: Service P und Zustand

Wie lange müsse Nachrichten an alte URI weitergeleitet werden an neue URI?

Wertvolles Werkzeug: Petrinetz-Theorie

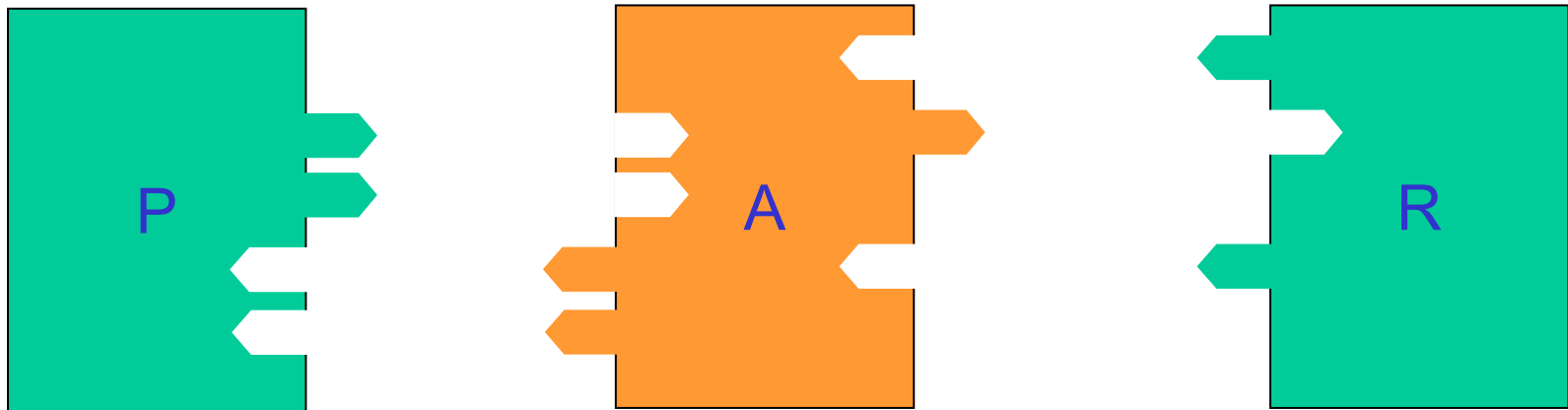
Problem 7: Public view



Geg.. service P

Finde ein kanonisches P' mit selben Partnern

Problem 8: Mediation/Adaptation



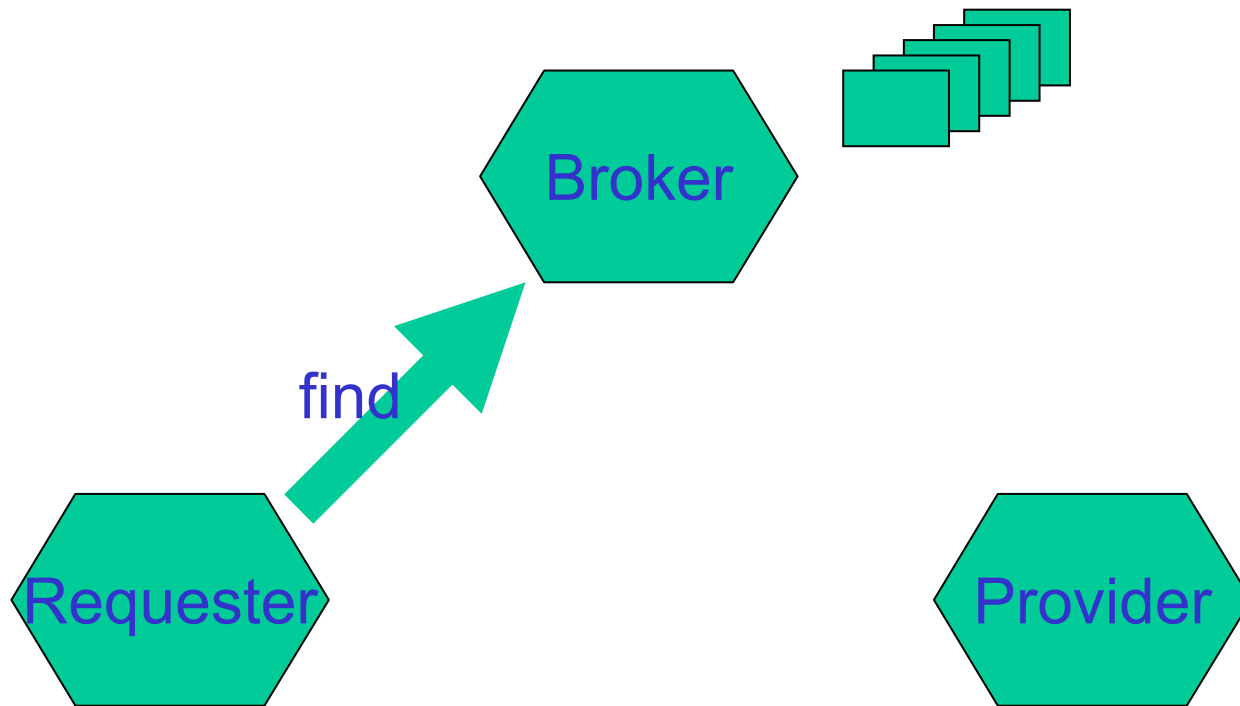
Geg.: Services P, R

Ex. ein Service A so, dass
P+A+R deadlock-frei bzw. livelock-frei

Wertvolles Werkzeug: Partnersynthese

Eigentlicher Wert: Adapter A

Problem 9: Discovery

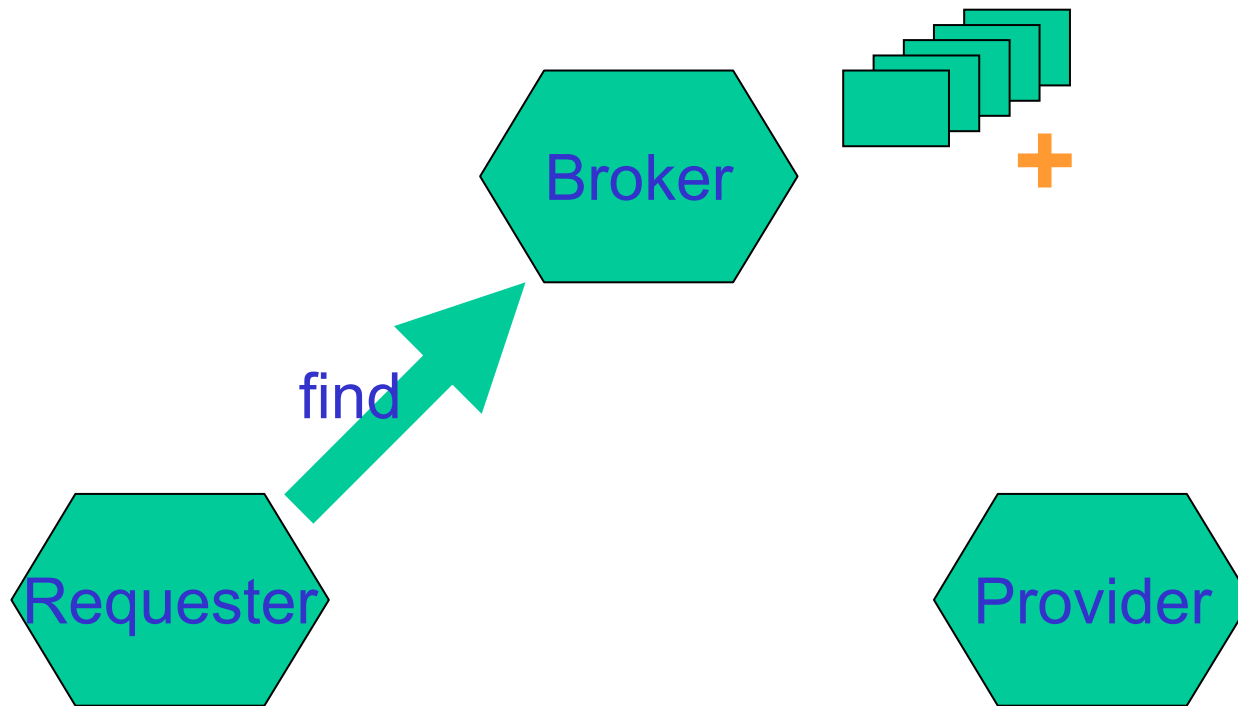


Geg.: Repository **S**, Service R

Enthält **S** einen zu R kompatiblen Service?

Wertvolle Werkzeuge: Bedienungsanleitung, Public View

Problem 10: Composition



Geg.: Repository **S**, service R

Enthält **S** services, die zu einem mit R kompatiblen komponiert werden können?

Wertvolles Werkzeug: **Mediation**

Variationen für viele Probleme

Sicherzustellende Eigenschaft

- Deadlock freedom+ livelock freedom
- Eventual termination
- ...

Kommunikationsmodell

- Asynchron
- Synchron (oder gemischt)
- Abhängigkeiten (leeres Formular vs. ausgefülltes Formular)

Zusätzliche Spezifikationen

- Verbiete/erzwinge Aktionen/Nachrichten

Variationen für viele Probleme

Natur der Umgebung

- Zentral
- Dezentral
- Autonom

Andere Aspekte

- Transactionen
- Policies
- Fault Handling
- Compensation
- ...

Problem 1: Compatibility

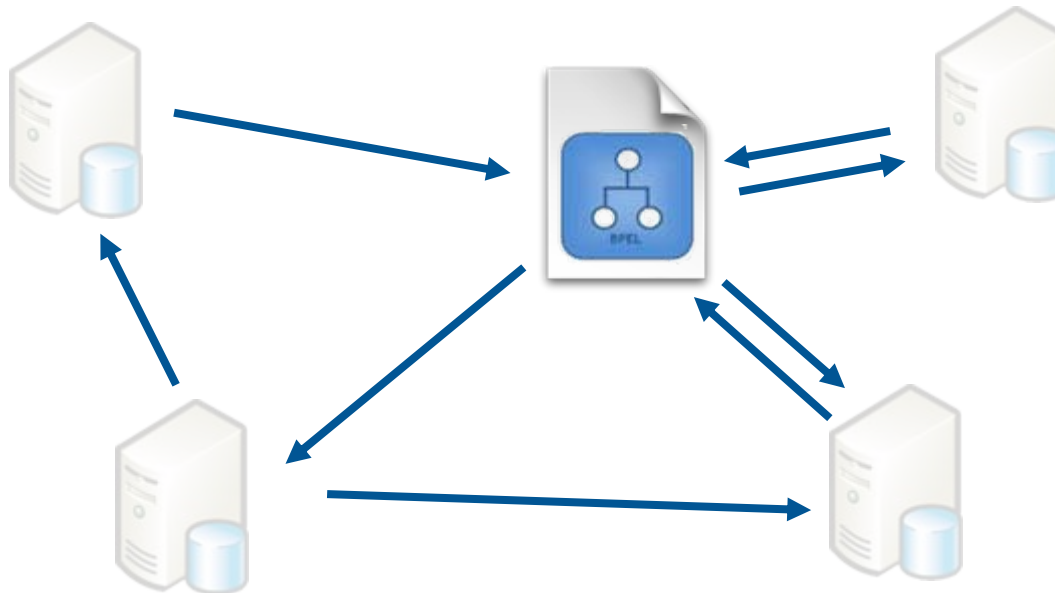


Geg: Services P, R

Ist $P+R$ deadlock-frei bzw. livelock-frei?

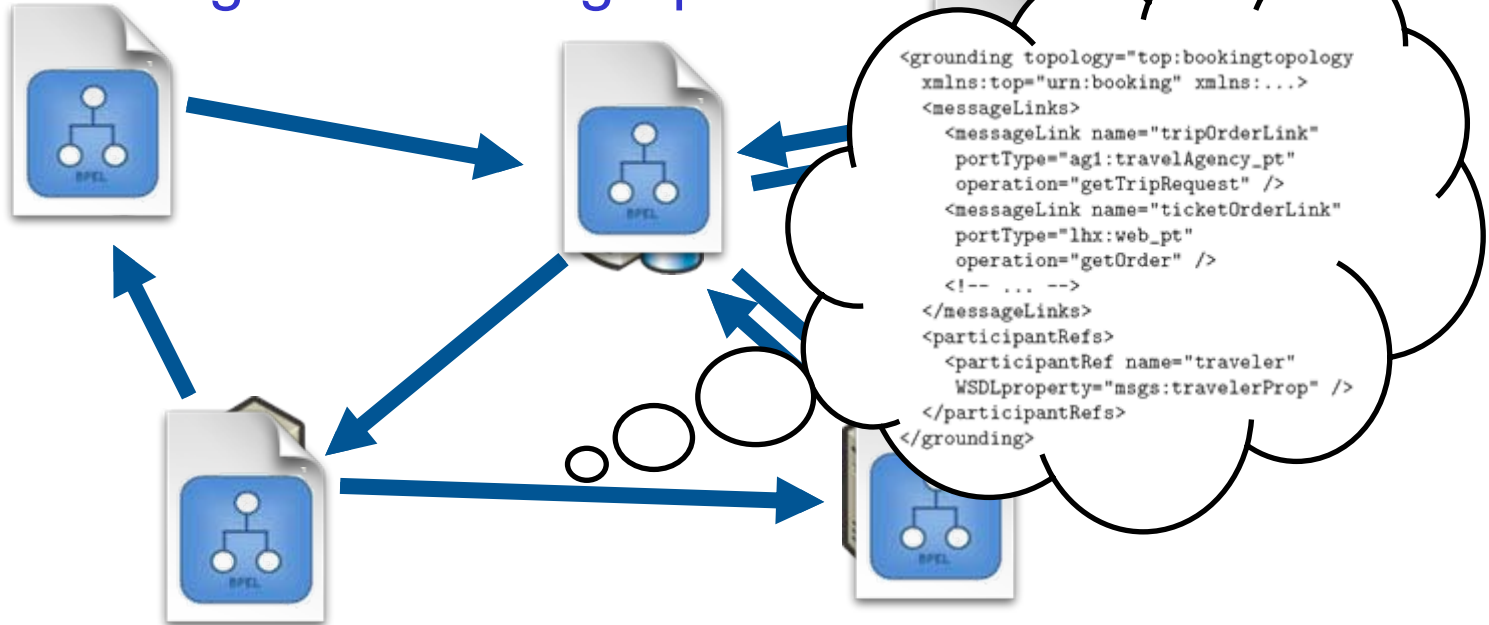
BPEL

- beschreibt EINE Service -Orchestration
- ... oder einen EINZELNEN Service
- Verhalten der gerufenen Services' NICHT spezifiziert



BPEL4Chor

- Erweiterung um Choreographien



behaviors

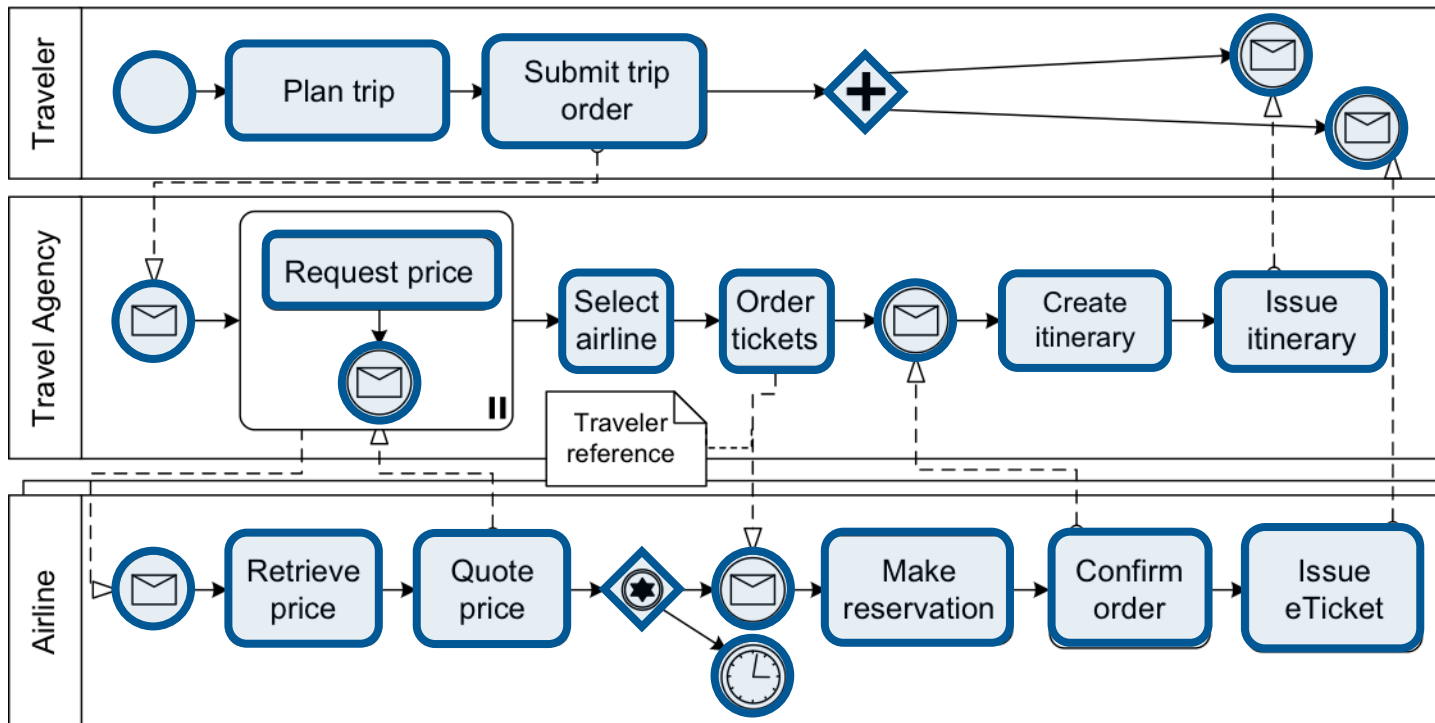


topology



grounding

Beispiel

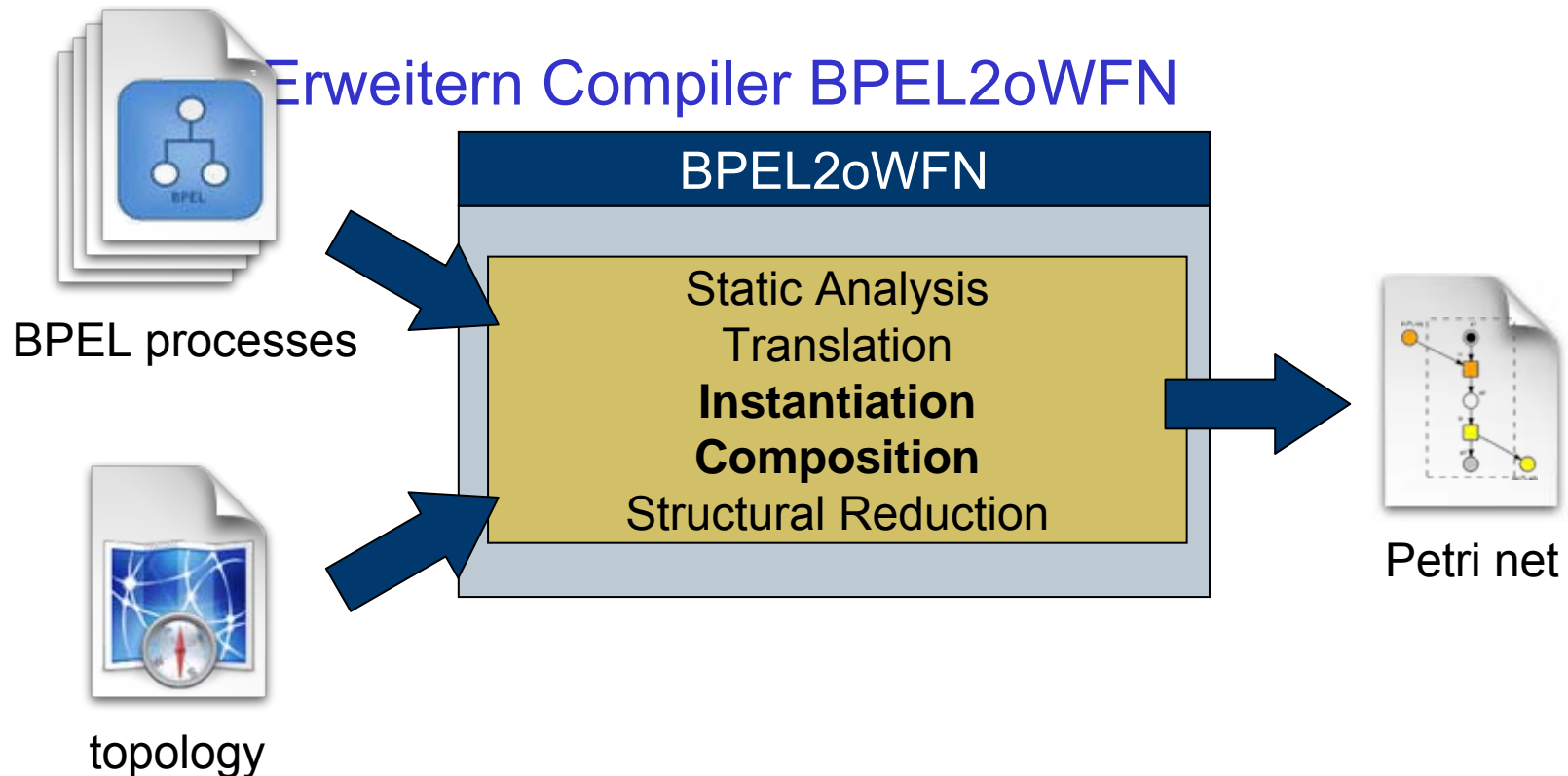


- Ein Reisender, ein Büro, mehrere Fluglinien

Was analysieren?

- “Klassische” Eigenschaften:
 - deadlock-freedom, livelock-freedom, no dead activities (a.k.a. Soundness)
- Nachrichten:
 - Kann jemals mehr als eine Nachricht auf einem Kanal liegen?
 - Min/Max Nachrichten bis zum Ende
- Verhalten:
 - Wird Request immer beantwortet?
 - Kann Teilnehmer Aktivität erzwingen/ausschließen?

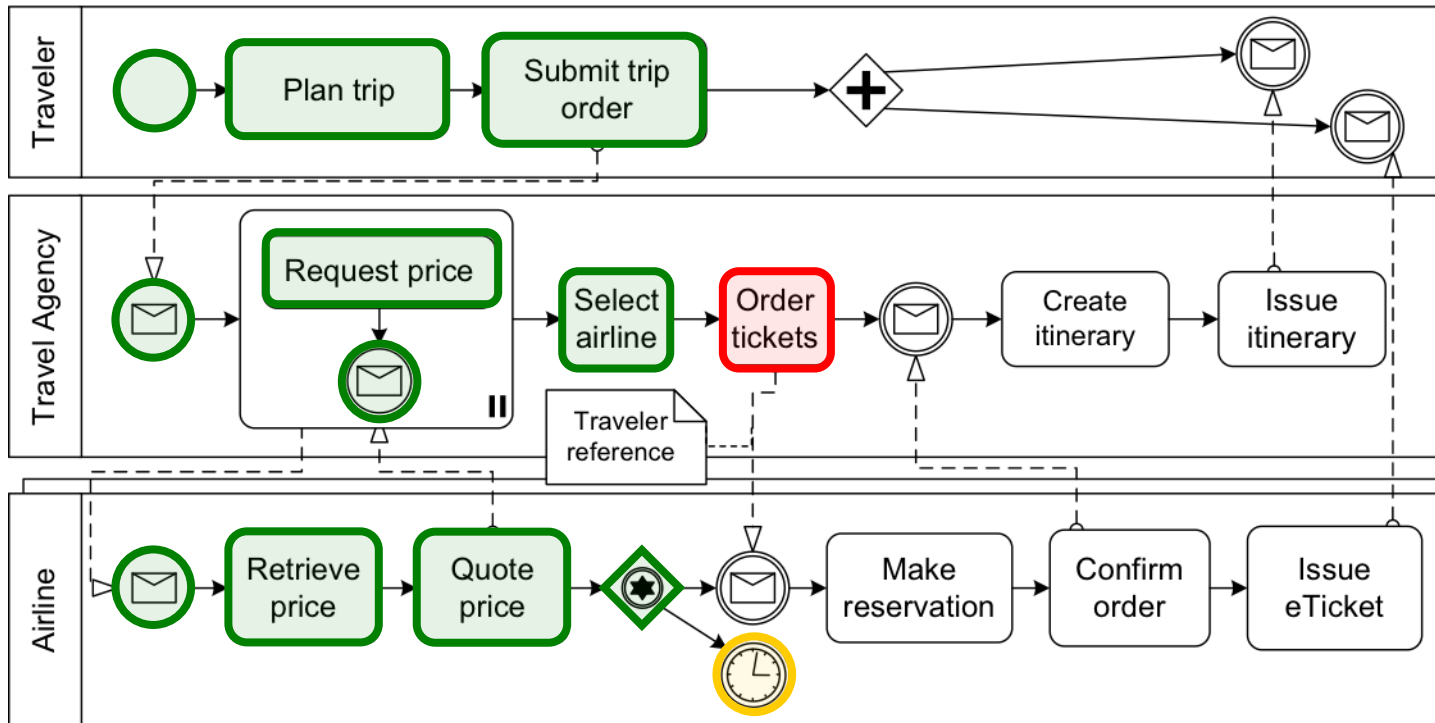
Übersetzen BPEL4Chor → Petrinetze



- um Instanziierung und Komposition

Result

- Choreography can **deadlock!**



- Jeder Teilnehmer für sich ist korrekt (controllable, sound...)
- deadlock subtil!

Case Study

airline instances

	1	5	10	100	1000
places	20	63	113	1013	10013
transitions	10	41	76	706	7006
states ①	14	3483	9806583	exponential growth	
states ②	14	561	378096		
states ③	11	86	261	18061	1752867
states ④	11	30	50	410	4010

- ① complete/unreduced
- ② symmetry reduction
- ③ partial order reduction
- ④ symmetry reduction and partial order reduction
- ⑤ out of memory (>2 GB)

linear
growth 😊

Problem 2: Controllability



Geg.: Service P

Ex. einn R so dass $P+R$ deadlock-frei bzw. livelock-frei?

Eigentlicher Wert: Partnersynthese

Operating Guidelines for Finite-State Service

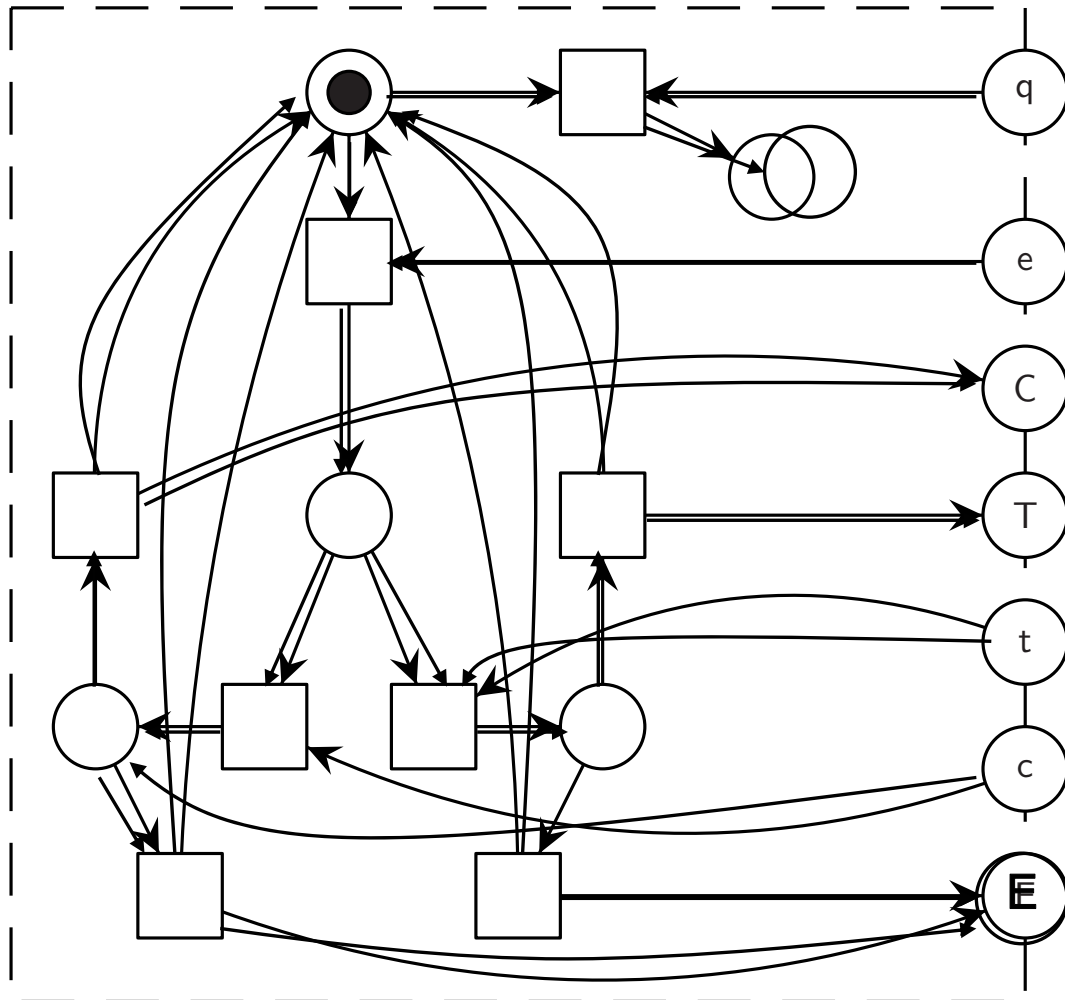
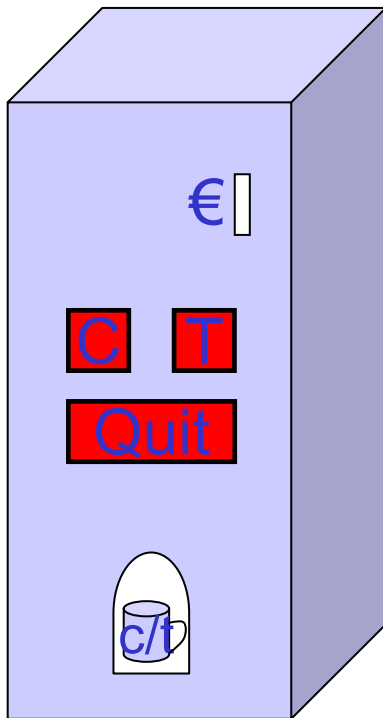
Karsten Wolf

Institut für Informatik
Universität Rostock

Niels Lohmann, Peter Massuthe
Institut für Informatik
Humboldt-Universität zu Berlin

Service

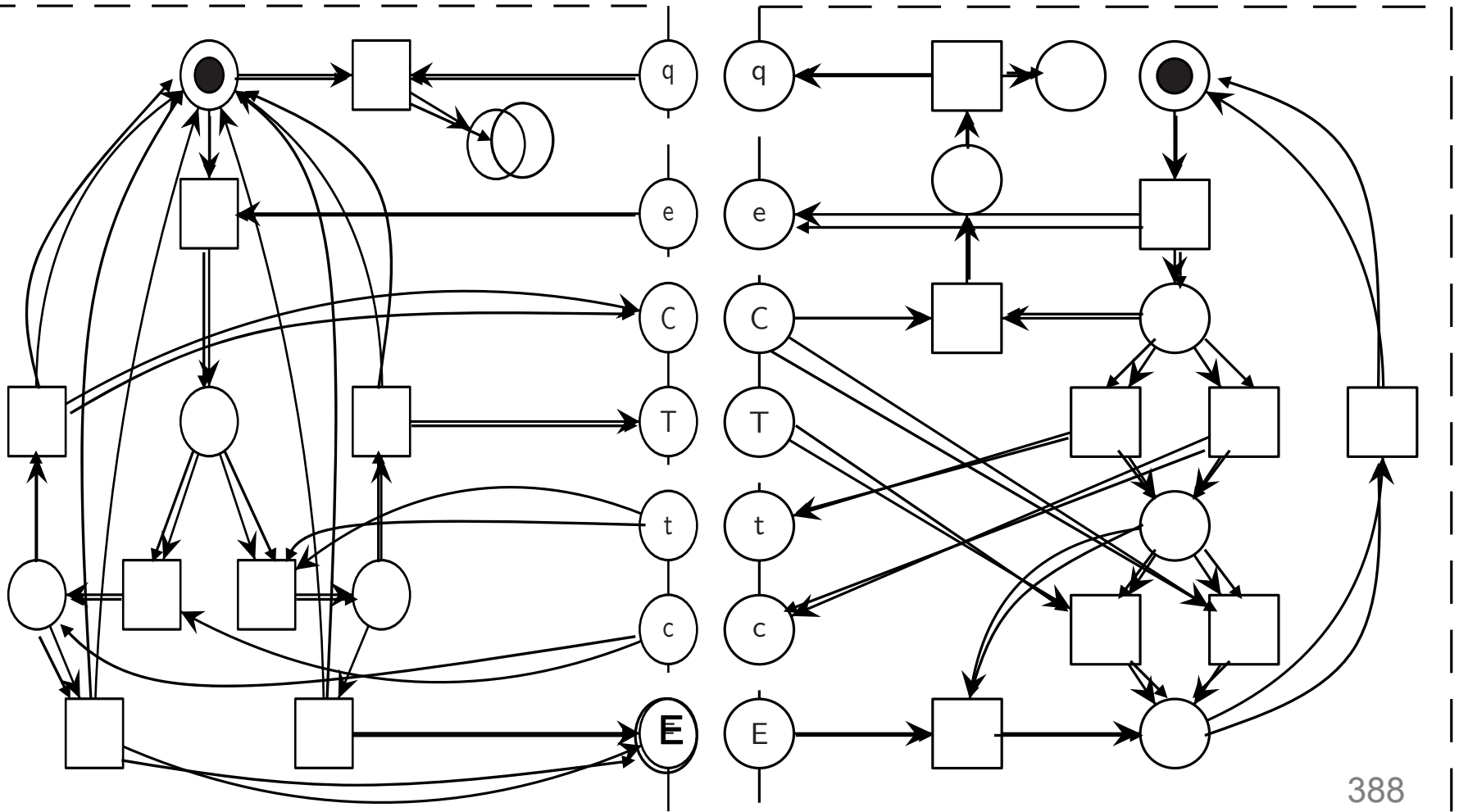
Getränkeautomat



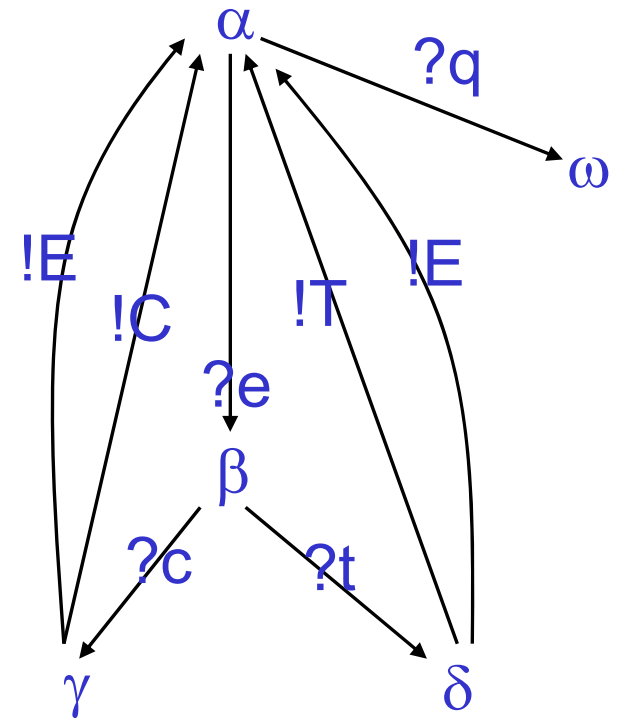
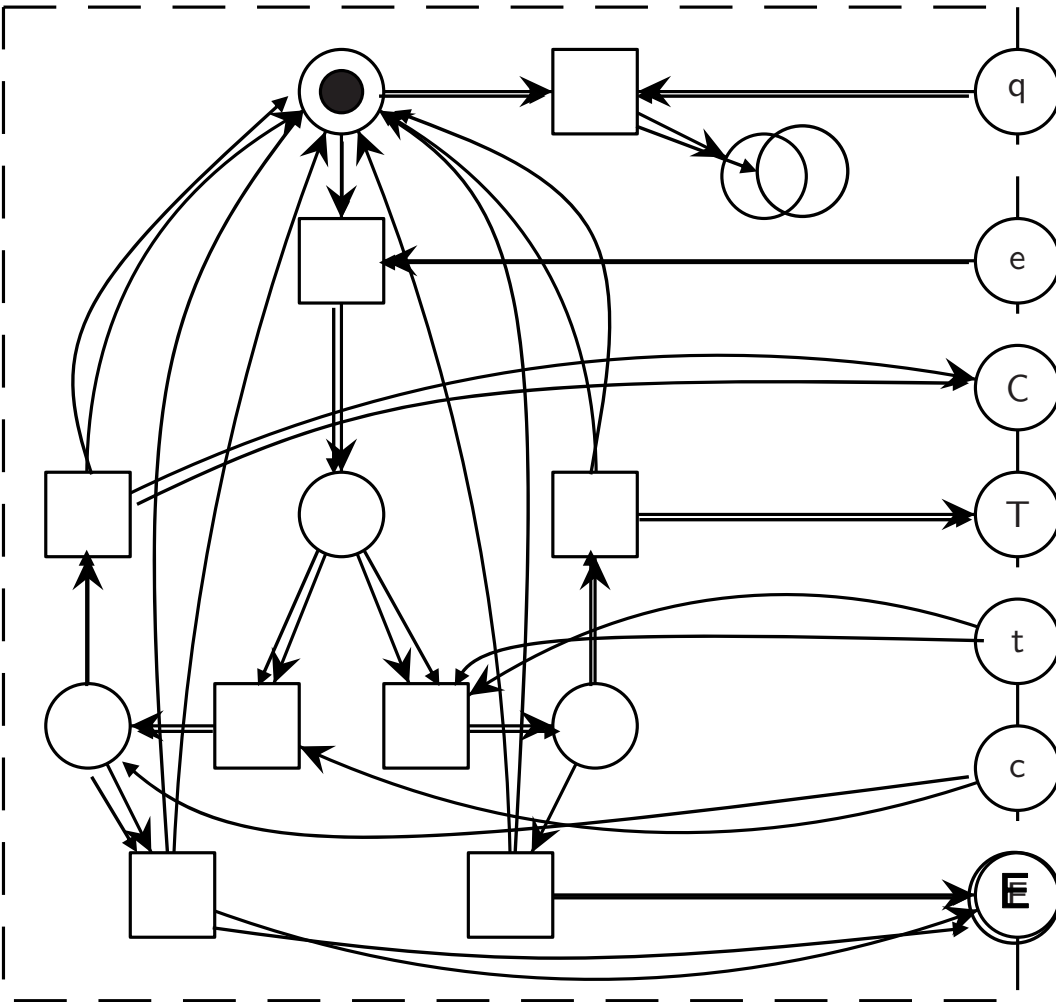
open workflow net

Service

Kunde

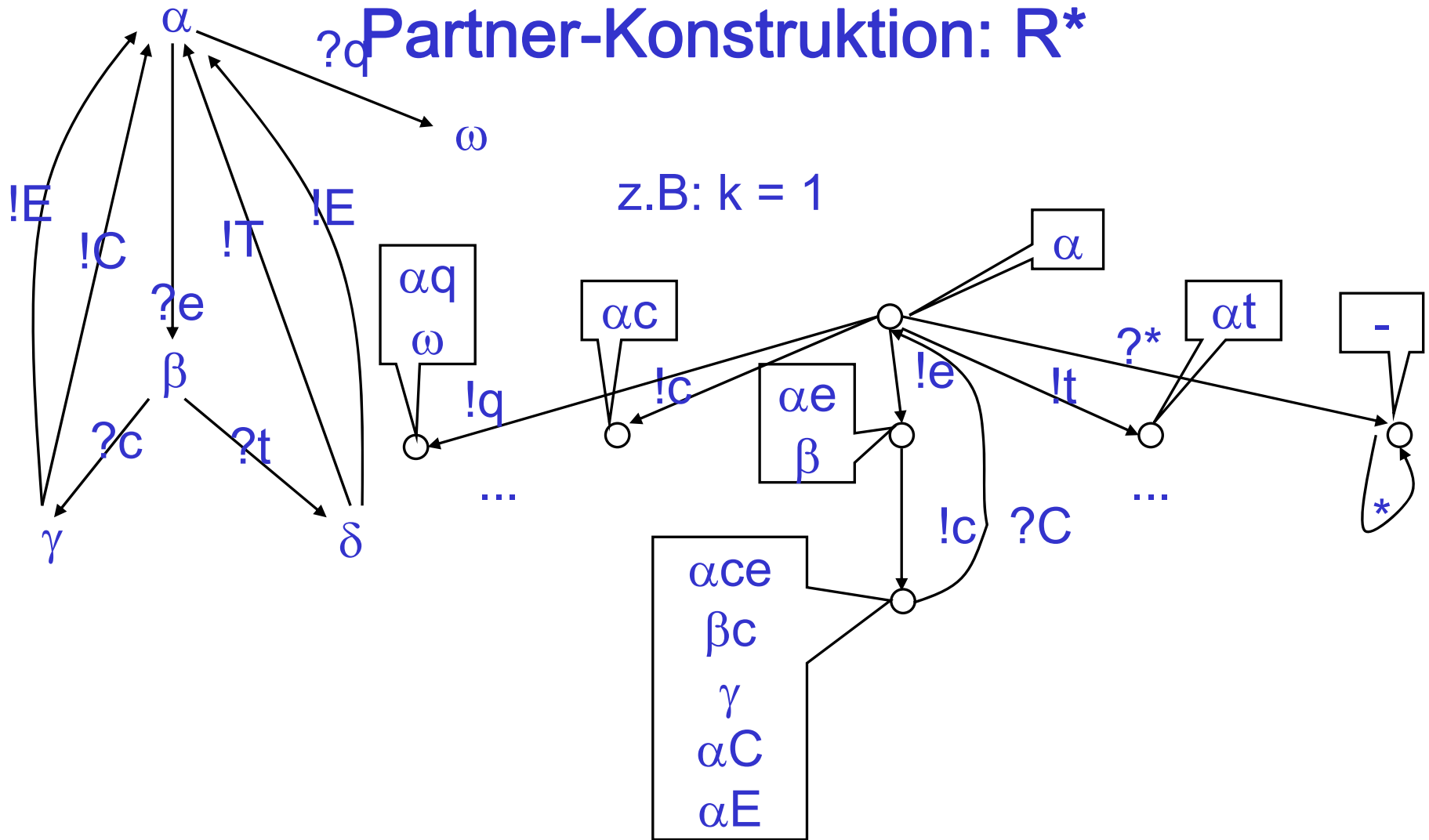


Verhalten eines einzelnen Service



Service automaton

Partner-Konstruktion: R^*

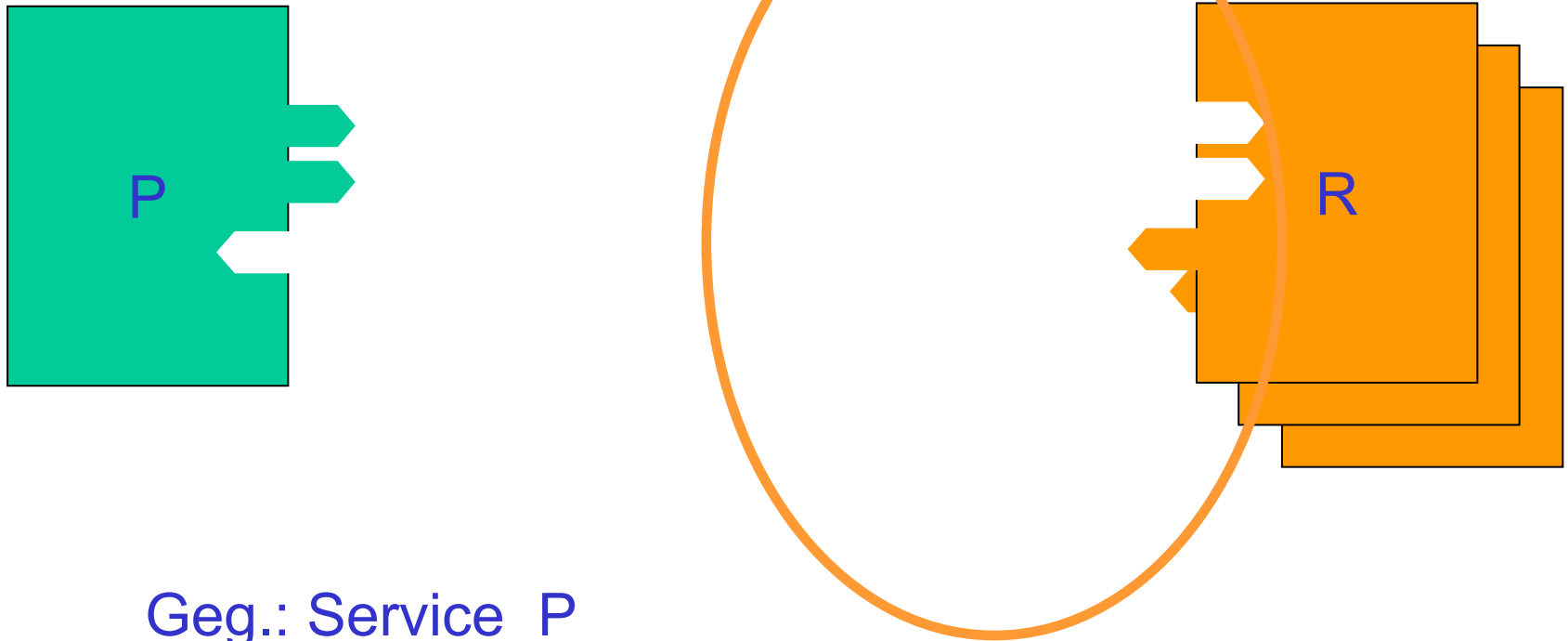


Konstruktion endlich wg. k -limit

Kanonizität von R^*

- Jeder deadlock-freie und k -limitierte Partner hat eine Simulationsrelation zu R^* (alles, was der andere kann, kann R^* auch)

Problem 3: Characterization



Charakterisiere alle R, für die P+R deadlock-frei bzw. livelock-frei!

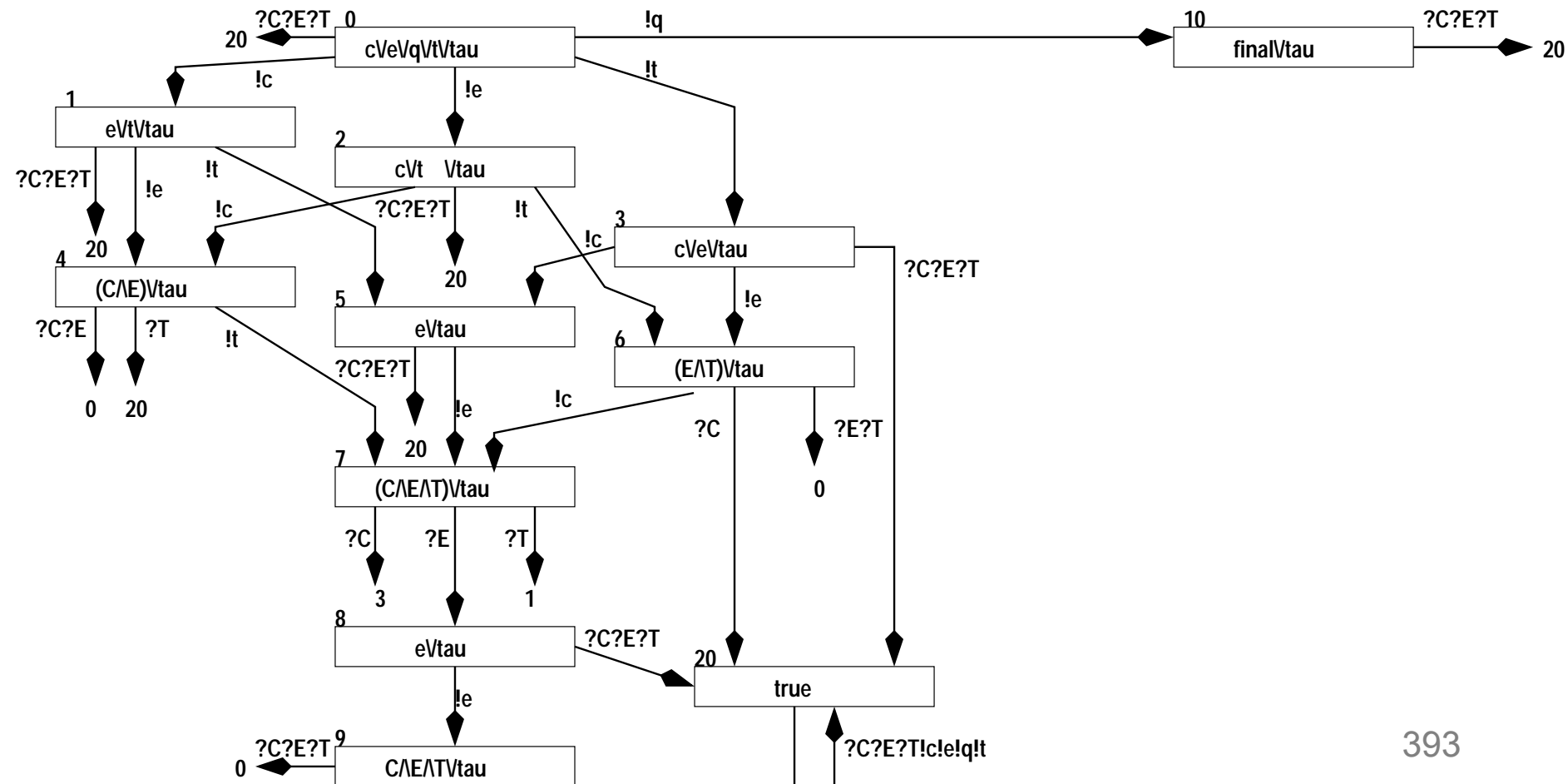
Wertvolle Zutat: **Synthesierter Partner**

Eigentlicher Wert: **Bedienungsanleitung**

Bedienungsanleitung

= R^* + Annotationen

Annotation = Boolesche Kodierung von „no deadlocks here“



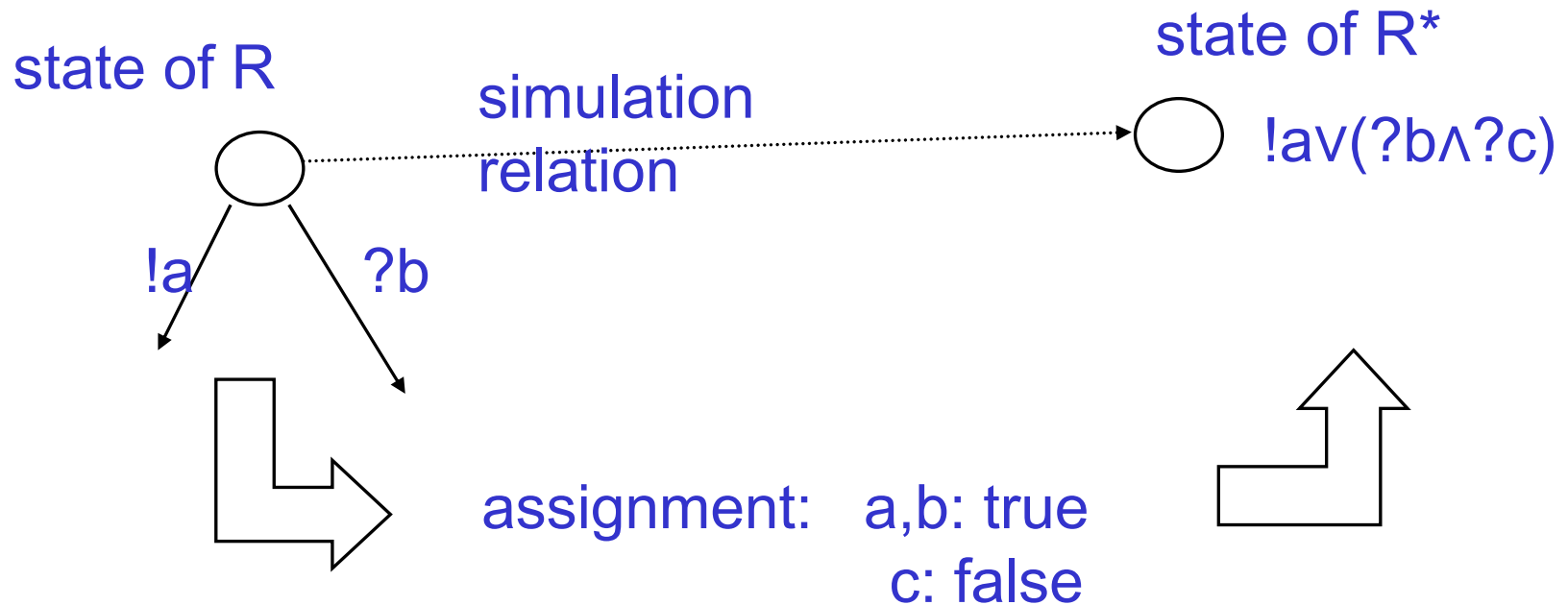
Results

Tool: FIONA

Service	P	T	States in OG	Time (s)
Purchase Order	38	23	168	0
Loan Approval	48	35	7	0
Olive Oil Ordering	21	15	14	0
Help Desk Service	33	28	8	2
Travel Service	517	534	320	7
Database Service	871	851	54	7583
Identity card service	149	114	280	216
Registration office	187	148	7	0
SMTP	206	215	362	200

Matching

= Simulationsrelation to R^* und Erfüllung der Annotationen



Problem 4: Exchangeability

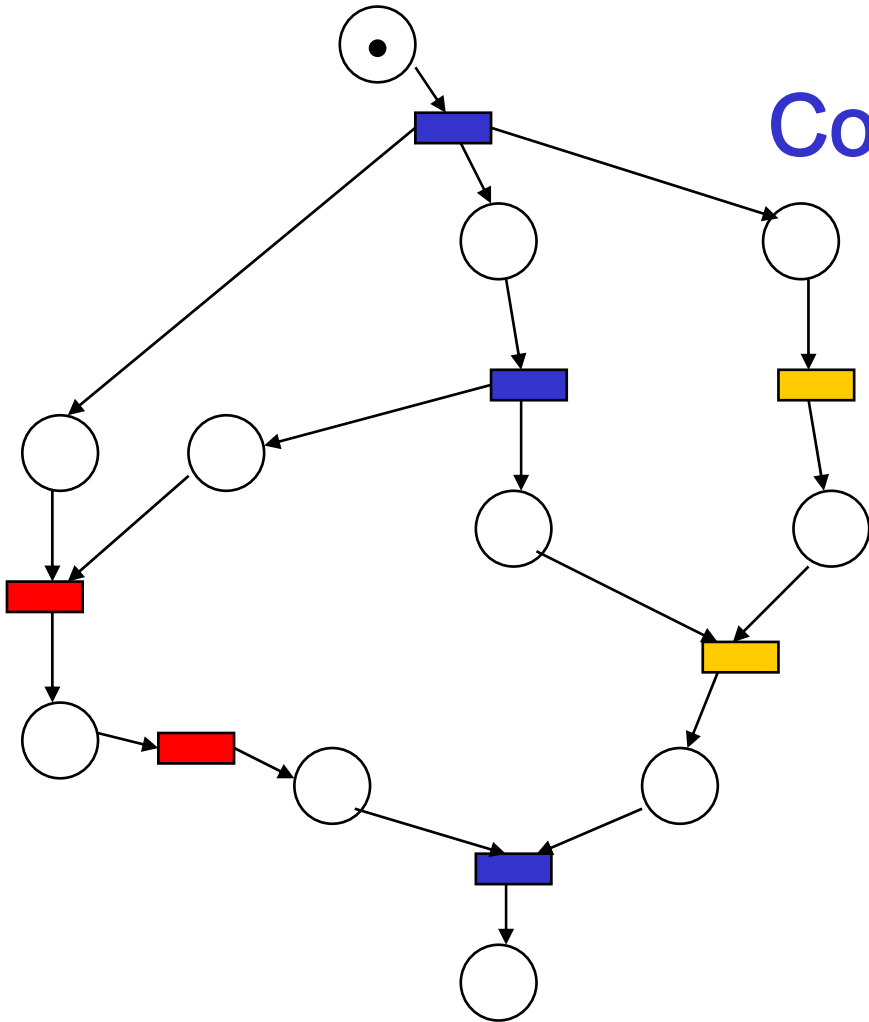


Geg.: Services P, P'

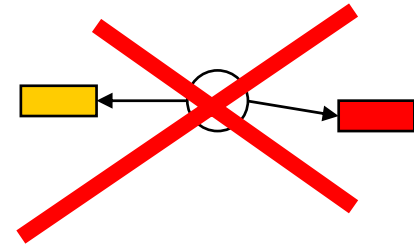
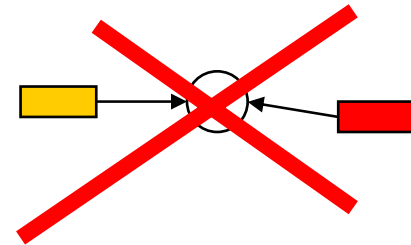
Ist jeder Partner von P auch ein Partner von P' ?

Wertvolles Werkzeug: **Bedienungsanleitung**

Contract

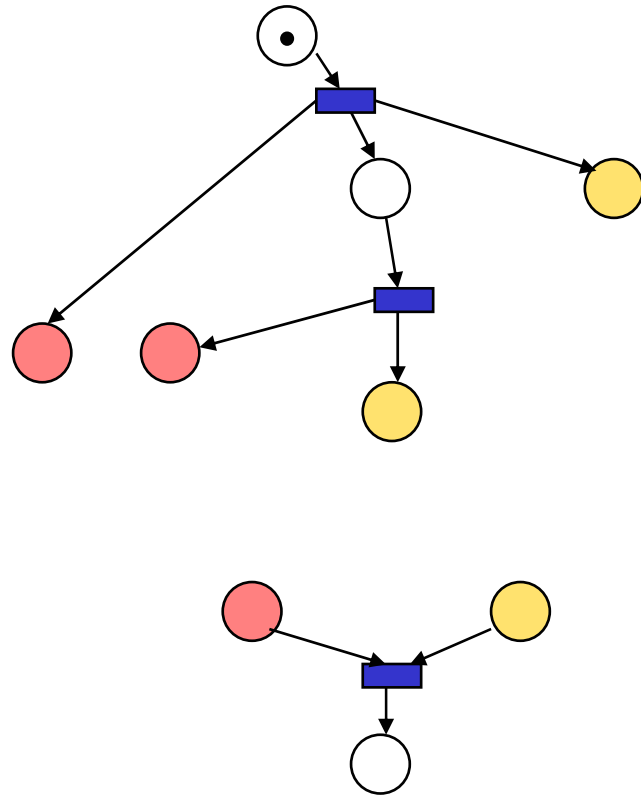


-oWFN
-weakly sound
-Leeres Interface

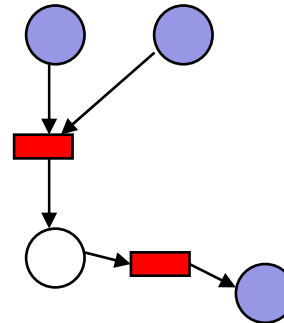


Limit Ltd.
Corpus Corp.
Park Bank

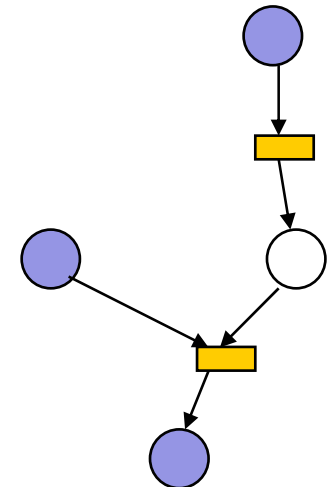
Contract → Public View



Corpus Corp.

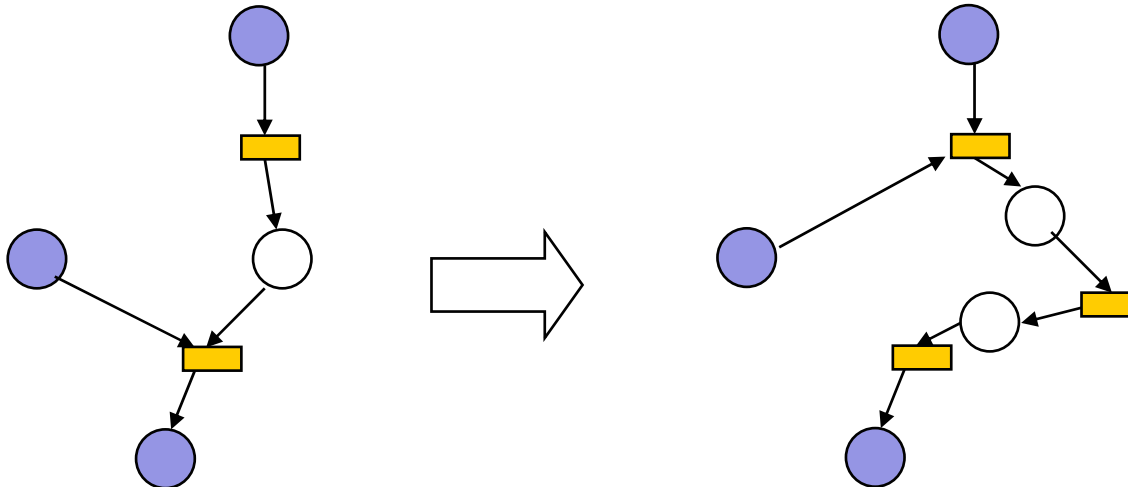


Limit Ltd.



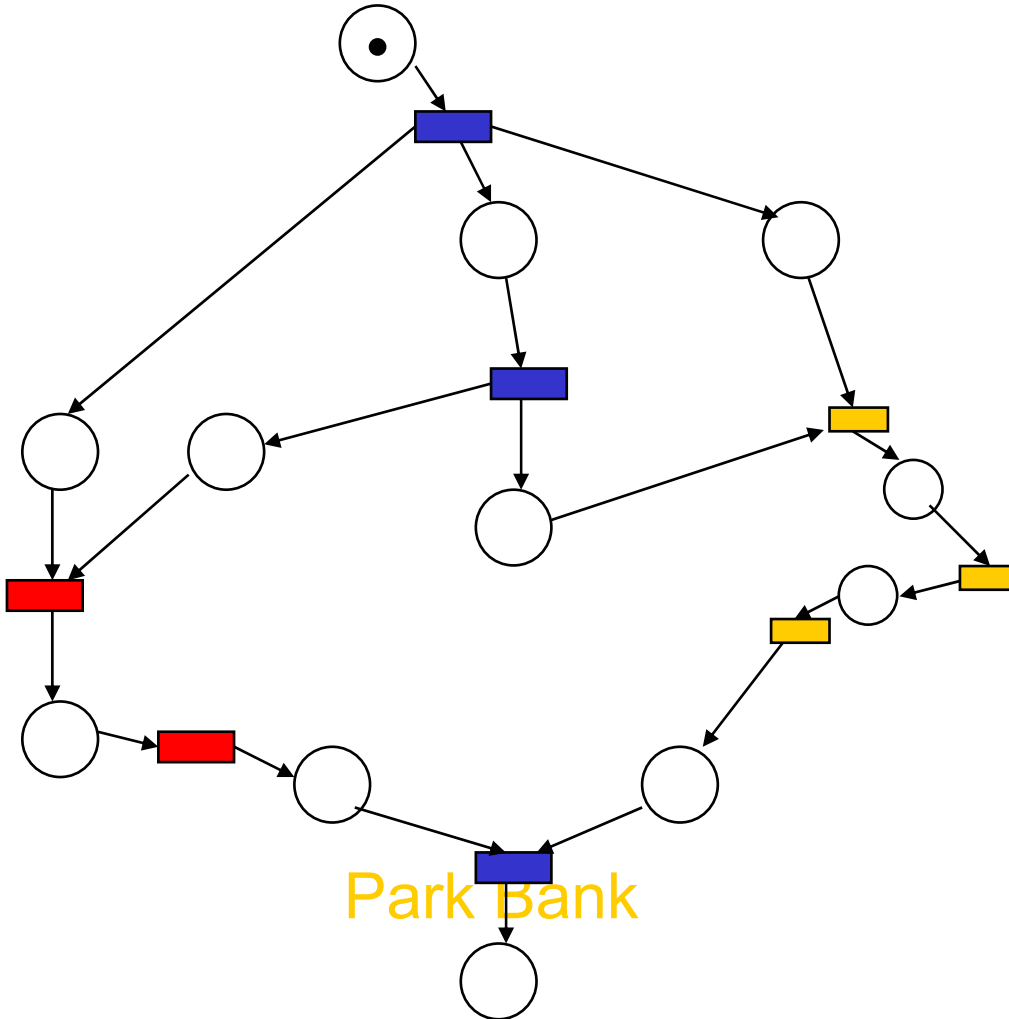
Park Bank

Public view → Private view



Park Bank

Private views \rightarrow Tatsächliches System



Ziel: System
weakly sound
(= weakly terminating),

Kriterien lokal zu jeder
Transformation
public \rightarrow private

Lösung

Kriterium: $\text{Strat}(\text{Public view}) \subseteq \text{Strat}(\text{Private View})$

Beweis: Wissen: $\text{Pu}_1 \oplus \text{Pu}_2 \oplus \dots \oplus \text{Pu}_n$ ist weakly sound

→ $\text{Pu}_2 \oplus \dots \oplus \text{Pu}_n \in \text{Strat}(\text{Pu}_1)$

→ $\text{Pu}_2 \oplus \dots \oplus \text{Pu}_n \in \text{Strat}(\text{Pr}_1)$

→ $\text{Pr}_1 \oplus \text{Pu}_2 \oplus \dots \oplus \text{Pu}_n$ ist weakly sound

→ $\text{Pr}_1 \oplus \text{Pu}_3 \oplus \dots \oplus \text{Pu}_n \in \text{Strat}(\text{Pu}_2)$

→ $\text{Pr}_1 \oplus \text{Pu}_3 \oplus \dots \oplus \text{Pu}_n \in \text{Strat}(\text{Pr}_2)$

→ $\text{Pr}_1 \oplus \text{Pr}_2 \oplus \text{Pu}_3 \oplus \dots \oplus \text{Pu}_n$ ist weakly sound

→ ...

→ $\text{Pr}_1 \oplus \text{Pr}_2 \oplus \dots \oplus \text{Pr}_n$ ist weakly sound

Implementation

$\text{Strat}(\text{Public view}) \subseteq \text{Strat}(\text{Private View})$

gdw $\text{OG}(\text{Public View})$ kann so in $\text{OG}(\text{Private View})$ eingebettet werden, dass Annotationen sich gegenseitig implizieren

Problem 7: Public view

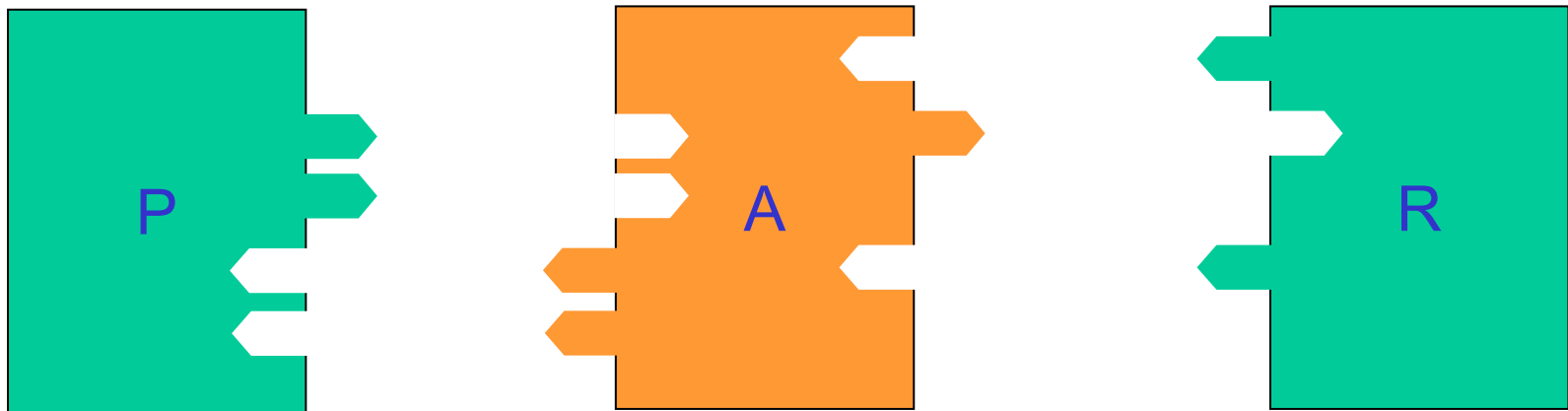


Geg.. service P

Finde ein kanonisches P' mit selben Partnern

Lösungsvorschlag: Rückübersetzung aus Bedienungsanleitung

Problem 8: Mediation/Adaptation



Geg.: Services P, R

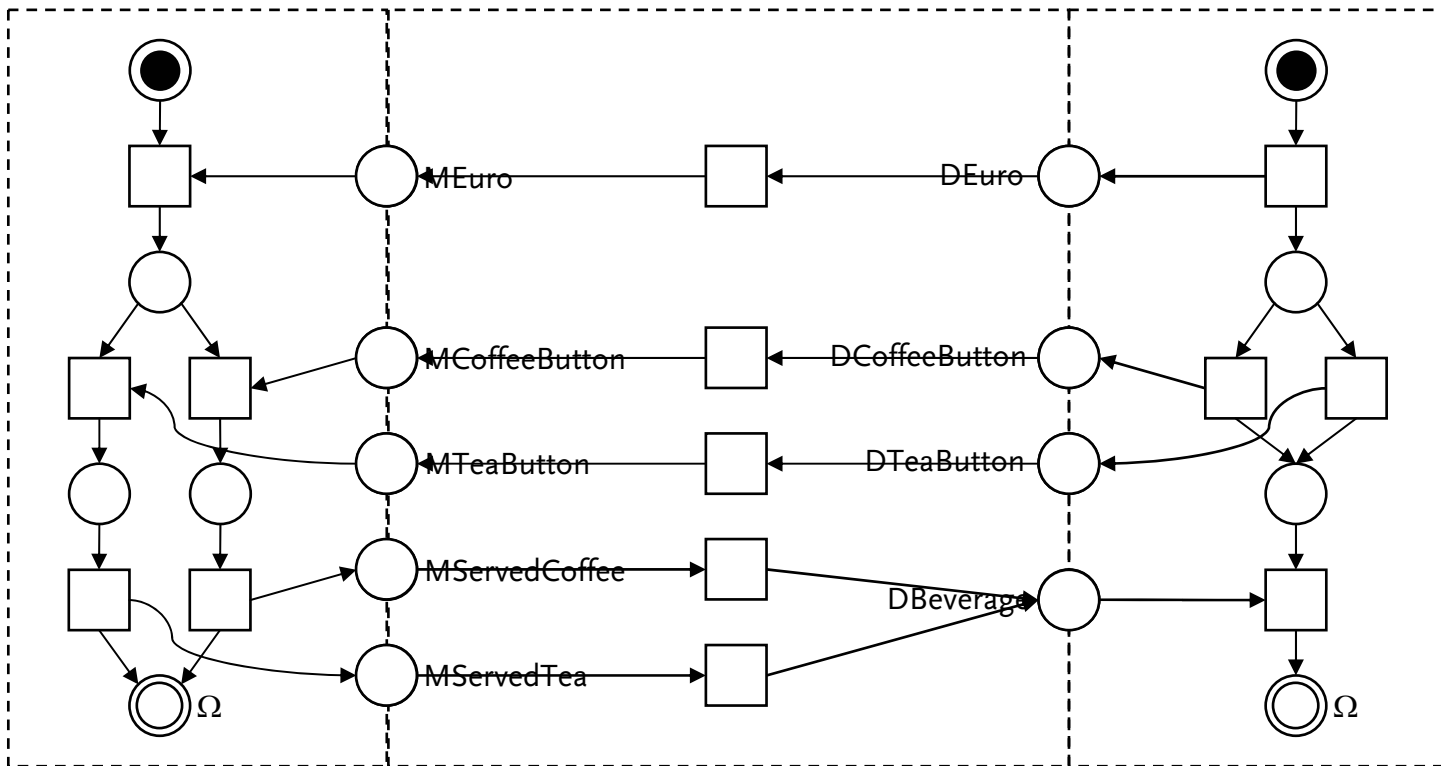
Ex. ein Service A so, dass
P+A+R deadlock-frei bzw. livelock-frei

Wertvolles Werkzeug: Partnersynthese

Eigentlicher Wert: Adapter A

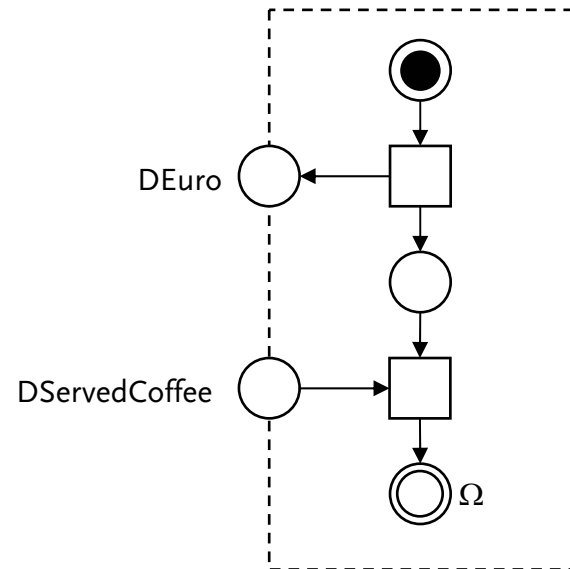
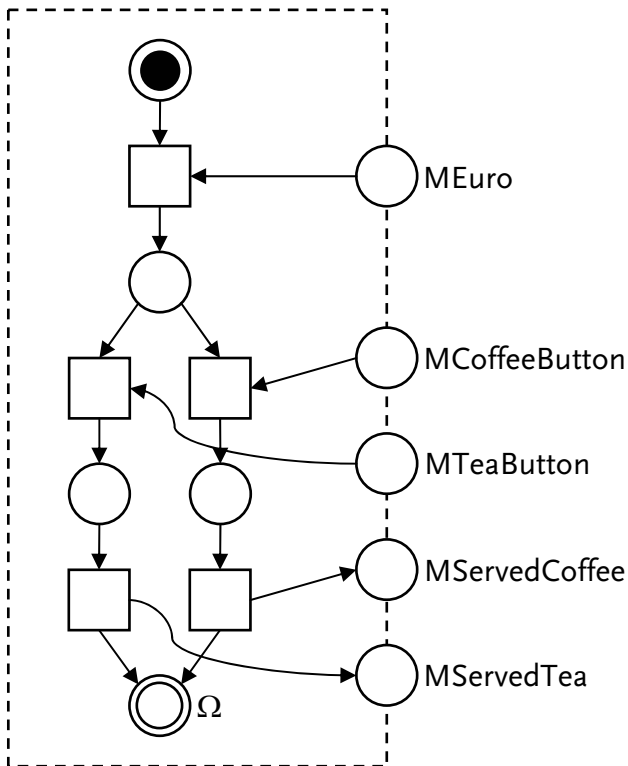
Wozu Adapter?

- Zur Überbrückung semantischer Unstimmigkeiten



Wozu Adapter?

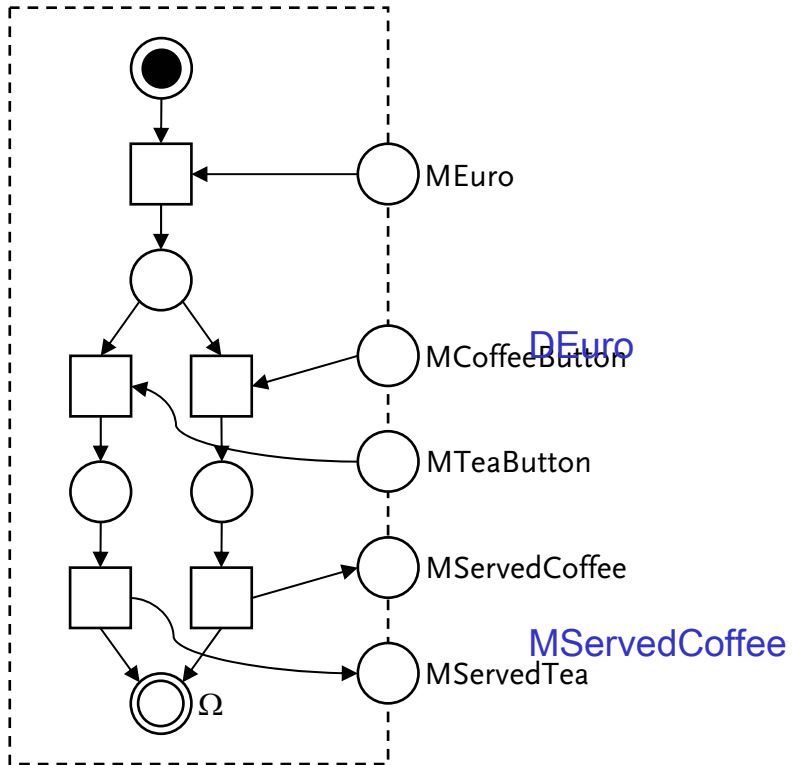
- Zur Überbrückung von Verhaltensunstimmigkeiten



Spezifikation elementarer Aktivitäten (SEA)

create a	$\rightarrow a$
copy a	$a \rightarrow a,a$
delete a	$a \rightarrow$
transform a into b	$a \rightarrow b$
split a into b,c,d	$a \rightarrow b,c,d$
merge a,b,c into d	$a,b,c \rightarrow d$
recombine a,b,c to d,e,f	$a,b,c \rightarrow d,e,f$

Beispiel

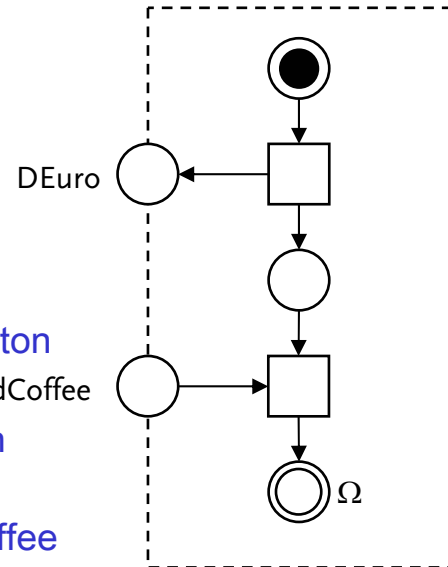


→ MEuro

→ MCoffeeButton

→ MTeaButton

→ DServedCoffee



Adaptersynthese I - Respektiere SEA

DEuro

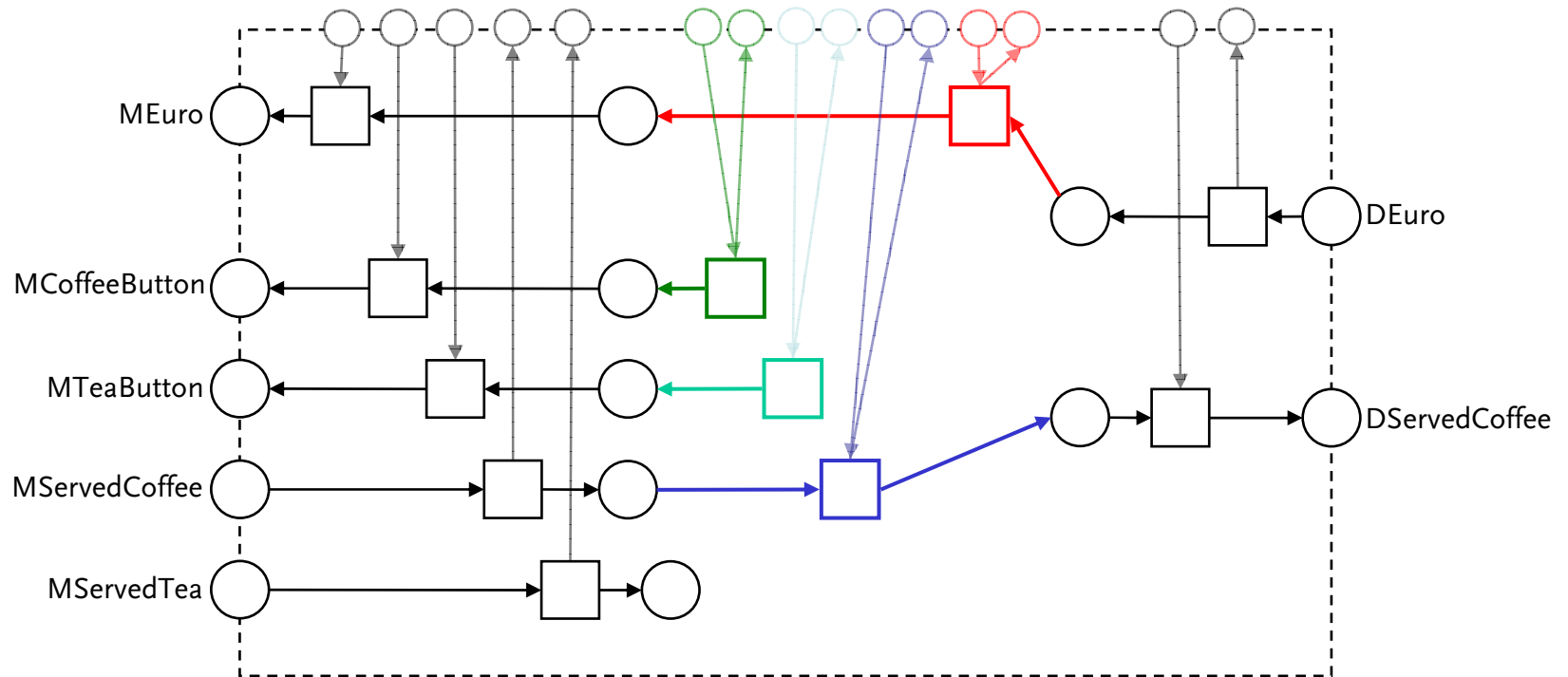
→ MEuro

→ MCoffeeButton

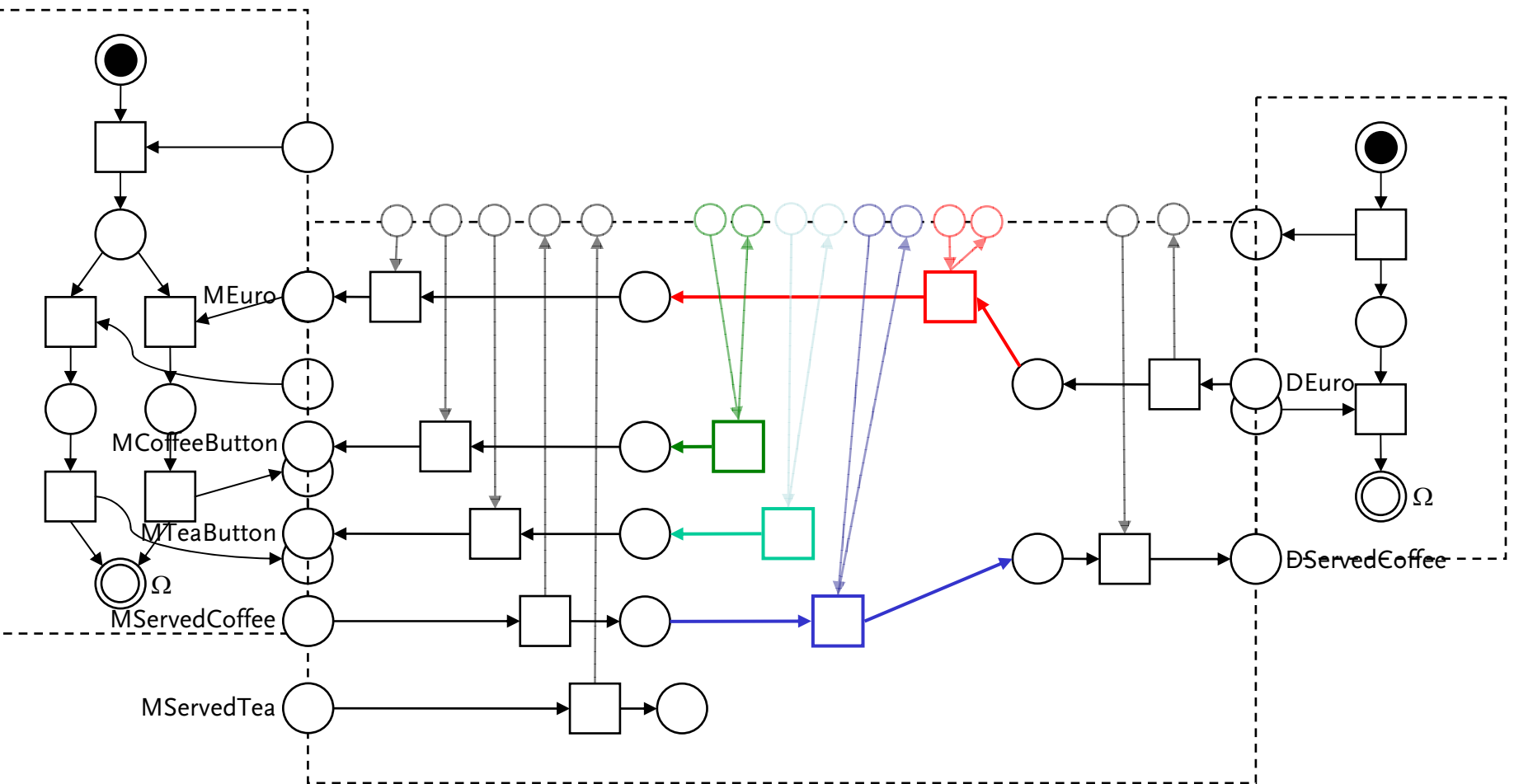
→ MTeaButton

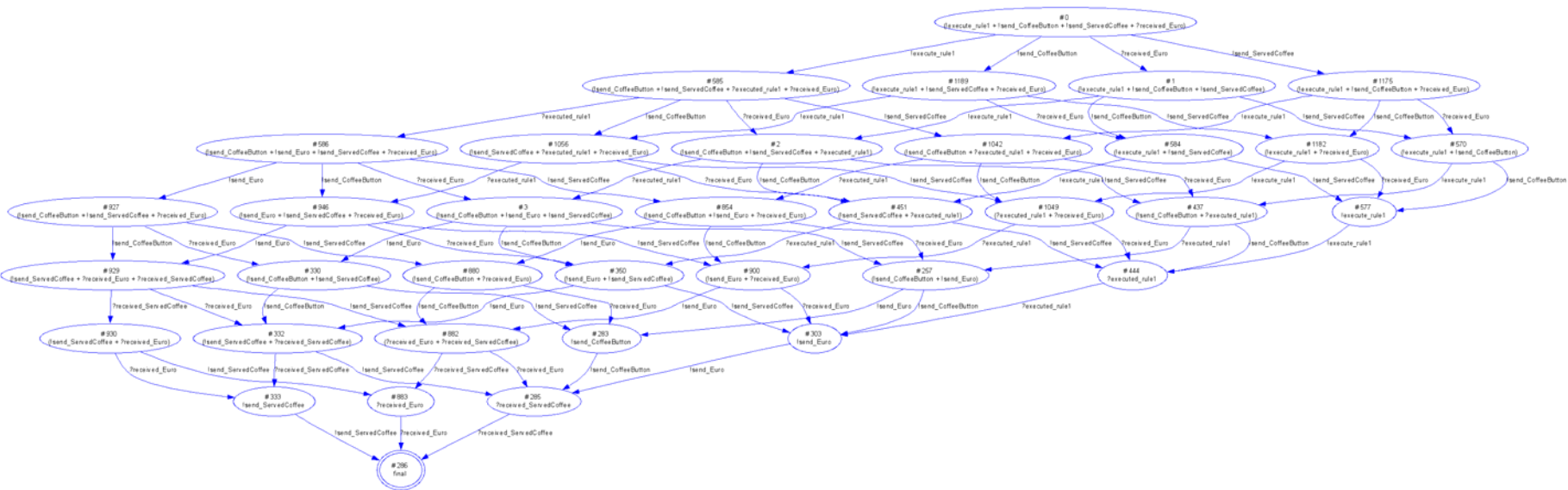
MServedCoffee

→ DServedCoffee

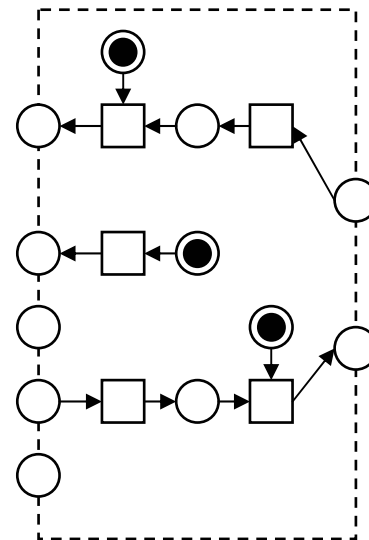
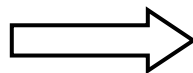
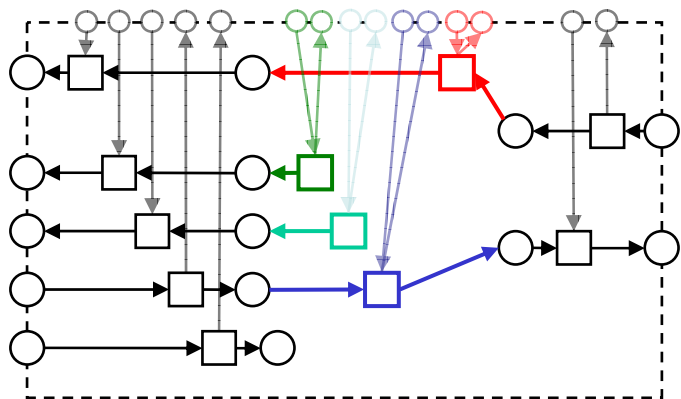


Adaptersynthese II: Partnersynthese

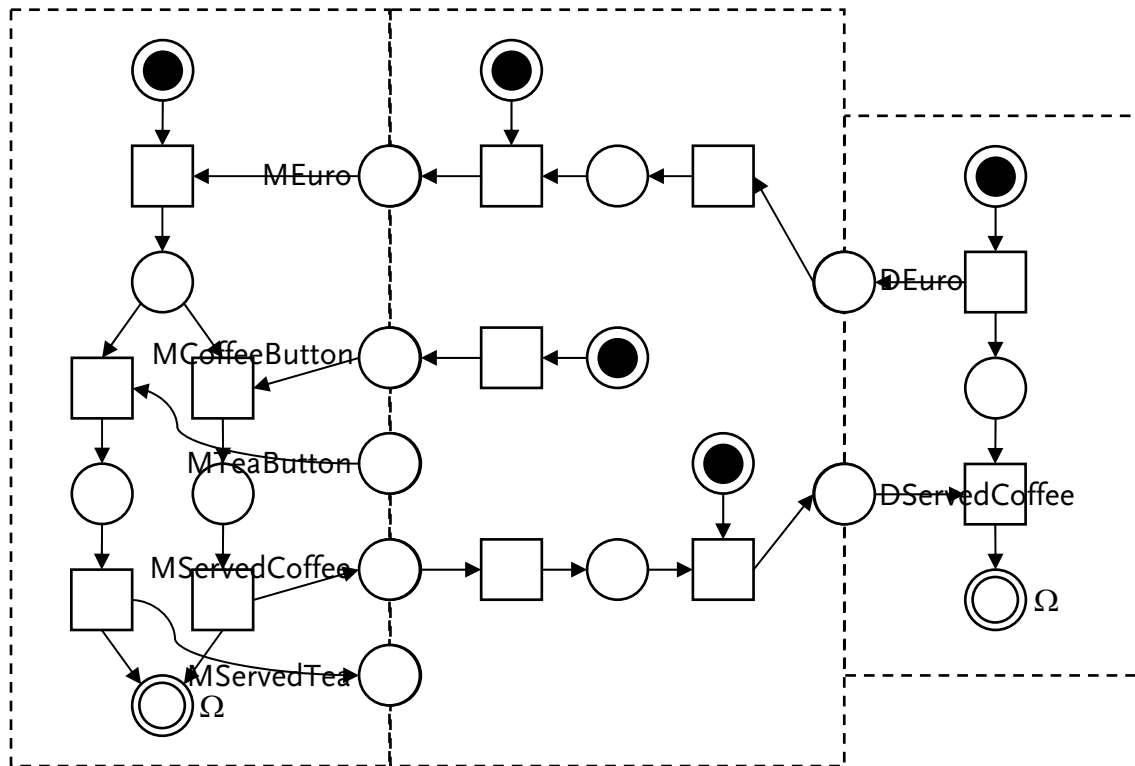




OG of myCoffee_myCustomer_adapter.owin (parameters: -m1)



Adaptersynthese– Resultat



Pros und Cons

- ✓ Synthese eines liberalsten Adapters
- ✓ können semantische Werkzeuge über SEA importieren
- ✓ Kein Partner → Niemand sonst wird einen finden
- ✓ flexibel, erweiterbar
- ✗ Viele Regeln, große Interfaces → Laufzeit noch zu groß

Zusammenfassung

- Teil 1: Geschäftsprozesse (geschlossene Systeme)
- Teil 2: Services (offene Systeme)
- Modellierung mit prozessorientierten Sprachen
 - typische Konstrukte: Aktivitäten, Sequenz, Parallele Ausführung, Splits & Joins (AND,OR,XOR)
 - Ressourcen spielen wichtige Rolle
 - semiformal bis formal

Zusammenfassung

- Modellierung gestattet
 - (semi)automatische Ausführung
 - Analyse
 - Dokumentation
 - Verbesserung und Veränderung
 - Synthese
- auf fundierter Grundlage

Zusammenfassung

- Analyse
 - qualitativ: z.B. Soundness (Geschäftsprozesse) oder Controllability (Services)
 - Methoden: Zustandsraumbasiert oder strukturell
 - quantitativ (Dauern, Zeitfenster oder stochastisch)

Zusammenfassung

- Synthese
 - Process Mining
 - Partner-, Adaptersynthese

Zusammenfassung

- Fazit: (z.T.) theoretisch fundierte Methoden produktiv einsetzbar in praktisch relevantem Kontext