

**Kapitel 2:**

**Formale Grundlagen von  
Prozessbeschreibungssprachen**

Prof. Dr. Peter Dadam  
Universität Ulm  
Institut für Datenbanken und Informationssysteme  
[www.uni-ulm.de/dbis](http://www.uni-ulm.de/dbis)  
[peter.dadam@uni-ulm.de](mailto:peter.dadam@uni-ulm.de)

## **2.0 Vorbemerkungen**

### **2.1 Korrektheitsaspekte von Prozessmodellen – eine Einführung**

### **2.2 Klassische Petri-Netze**

### **2.3 Höhere Petri-Netze**

### **2.4 Workflow-Netze**

### **2.5 Aktivitätennetze**

### **2.6 AristaFlow-Prozessmodell**

### **2.7 Andere Ansätze**

### **2.8 Ausdrucksmächtigkeit von Prozessbeschreibungssprachen**

### **2.9 Abschließende Bemerkungen**

### **2.10 Weiterführende Literatur**

## 2.0 Vorbemerkungen

---

- ❑ Prozessmodelle werden heute in der Regel graphisch dargestellt
- ❑ Die dort verwendeten Symbole und Anordnungsbeziehungen zwischen den Symbolen sollten formal präzise und eindeutig definiert sein, ansonsten
  - Gefahr von Missverständnissen in der zwischenmenschlichen Kommunikation
  - führt dies zu unterschiedlichen Implementierungen in Systemen (Modellierungswerkzeugen, Prozess-Management-Systemen, Analysetools, ...)
  - eingeschränkte (oder gar nicht vorhandene) Möglichkeiten zur formalen Analyse auf Modellierungsfehler
- ❑ Möglichkeit zur (mögl. umfassenden) formalen Analyse auf Modellierungsfehler insbesondere bei ausführbaren Prozessmodellen wichtig
  - sonst hoher Testaufwand vor dem Deployment (Analogie: Programmiersprachen mit „weak typing“)
  - Gefahr von Laufzeitfehlern, insbesondere bei komplexen Prozessen

## 2.0 Vorbemerkungen

---

### Lernziele dieses Kapitels:

- ❑ Einführung in unterschiedliche Formalismen mit exakt definierter „Schaltsemantik“ zur Beschreibung von Abläufen (Prozessen)
- ❑ Sensibilisierung für das Thema „Korrektheit“ von Prozessmodellen
- ❑ Einblicke in die Ausdrucksmächtigkeit des jeweiligen Ansatzes  
... sowie dessen Möglichkeiten zur Erkennung von Modellierungsfehlern

### Anmerkung:

Auf „Malwerkzeuge“ ohne formal exakt definierte Semantik, die man in der Praxis häufig für die frühen Phasen der Prozesserschaffung und -modellierung einsetzt, werden wir in späteren Kapiteln eingehen.

## 2.0 Vorbemerkungen

## 2.1 Korrektheitsaspekte von Prozessmodellen – eine Einführung

## 2.2 Klassische Petri-Netze

## 2.3 Höhere Petri-Netze

## 2.4 Workflow-Netze

## 2.5 Aktivitätennetze

## 2.6 AristaFlow-Prozessmodell

## 2.7 Andere Ansätze

## 2.8 Ausdrucksmächtigkeit von Prozessbeschreibungssprachen

## 2.9 Abschließende Bemerkungen

## 2.10 Weiterführende Literatur

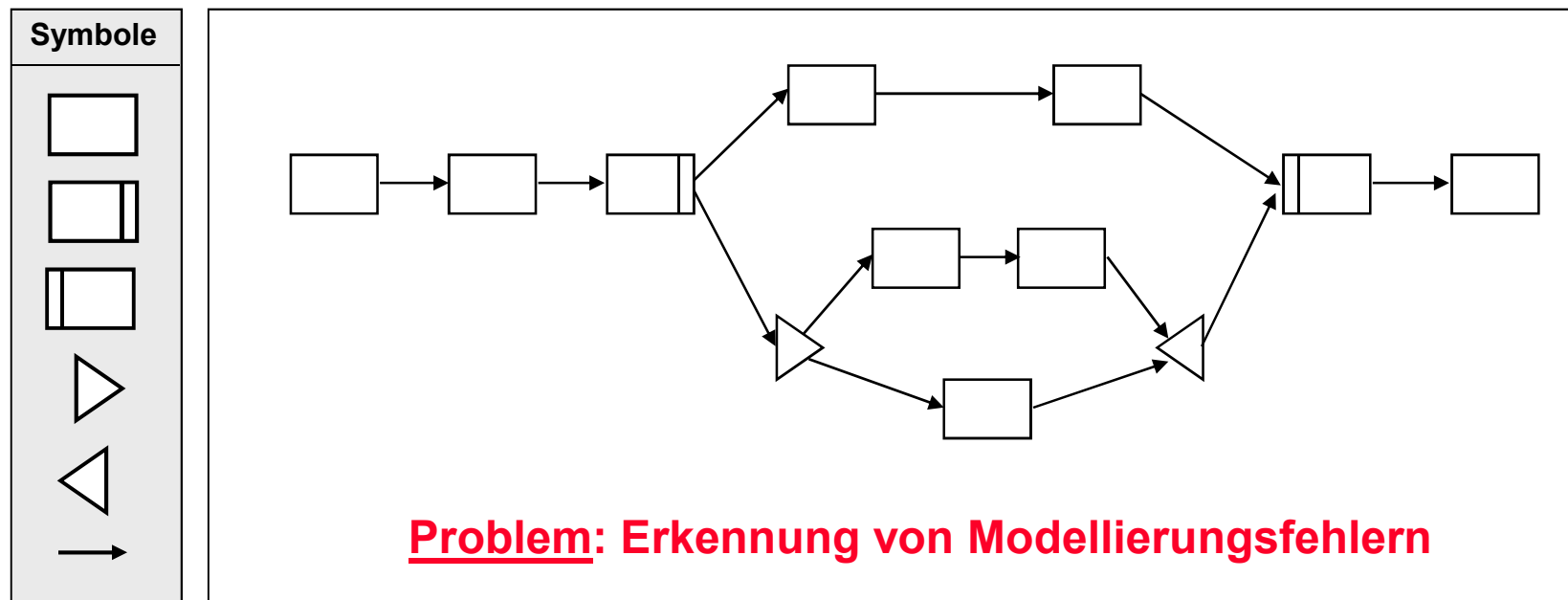
## 2.1 Korrektheitsaspekte von Prozessmodellen – eine Einführung

---

### „Korrektheit“ von Prozessmodellen – was ist das Problem?

#### □ Herausforderung 1: „Strukturelle Korrektheit“

Probleme allzu „freihändiger“ und „liberaler“ Modellierung

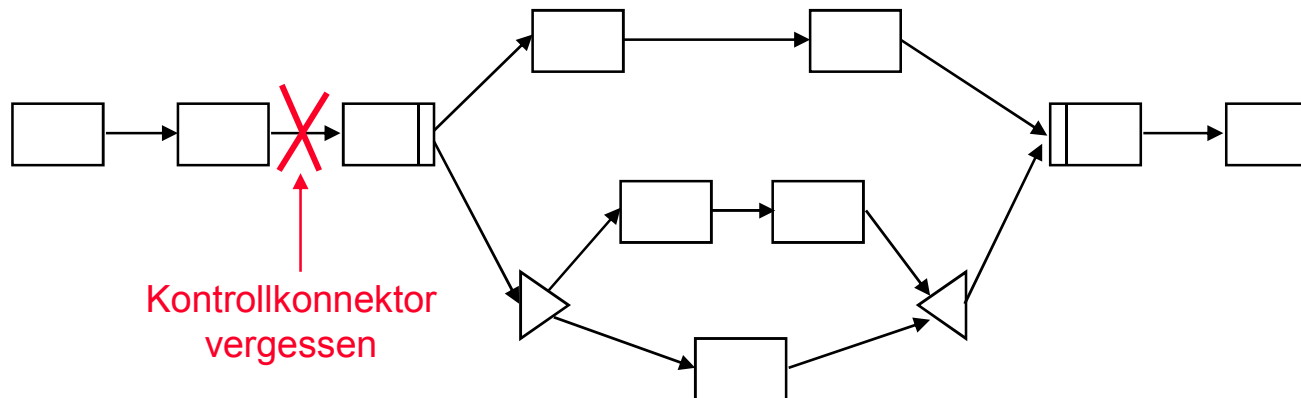


## 2.1 Korrektheitsaspekte von Prozessmodellen – eine Einführung

---

### □ Herausforderung 1: „Strukturelle Korrektheit“

Probleme allzu „freihändiger“ und „liberaler“ Modellierung



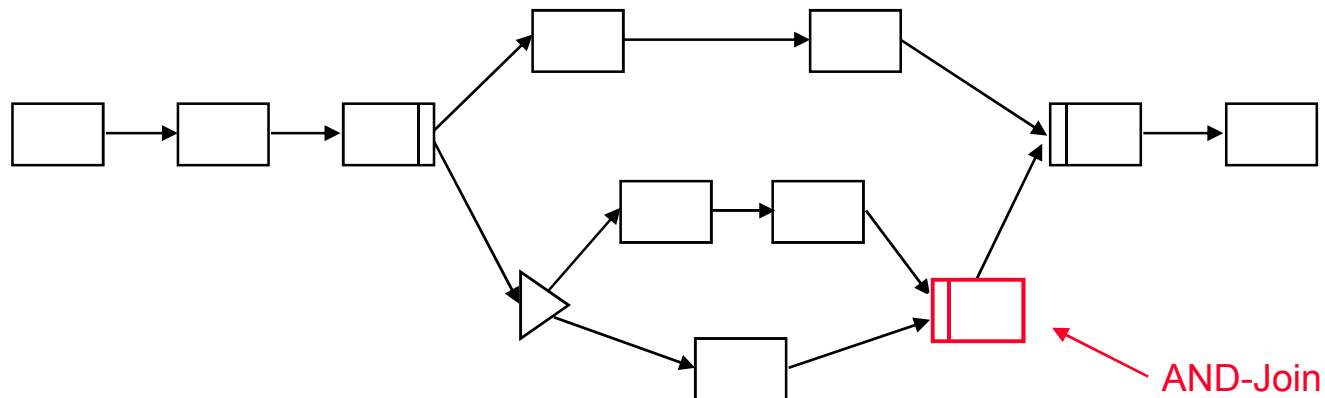
**Was passiert bei der Ausführung?** (Annahme: Prozess-Modell wird als „korrekt“ akzeptiert)

## 2.1 Korrektheitsaspekte von Prozessmodellen – eine Einführung

---

### □ Herausforderung 1: „Strukturelle Korrektheit“

Probleme allzu „freihändiger“ und „liberaler“ Modellierung



**Was passiert bei der Ausführung?** (Annahme: Prozess-Modell wird als „korrekt“ akzeptiert)

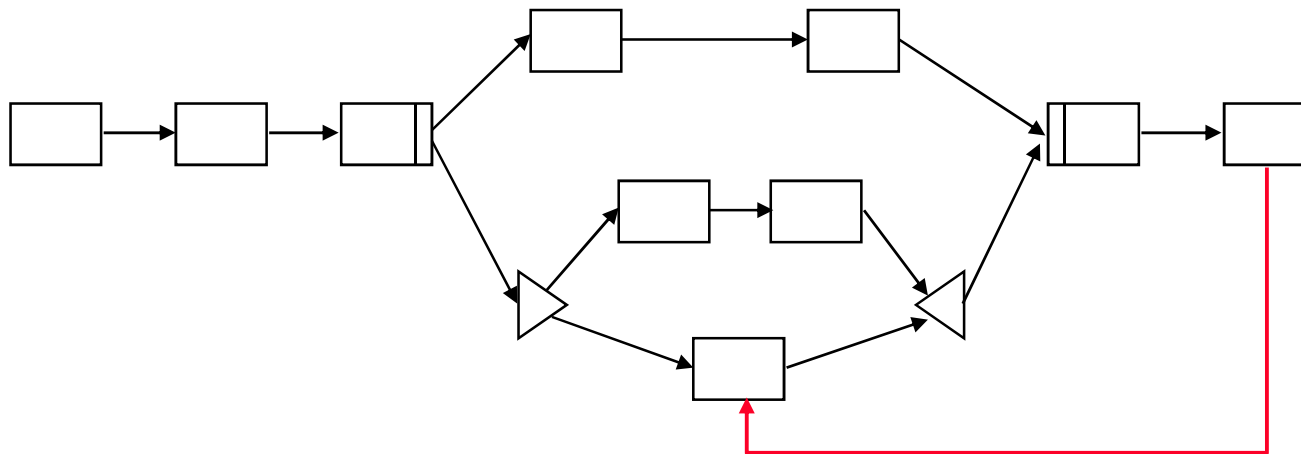


## 2.1 Korrektheitsaspekte von Prozessmodellen – eine Einführung

---

### □ Herausforderung 1: „Strukturelle Korrektheit“

Probleme allzu „freihändiger“ und „liberaler“ Modellierung

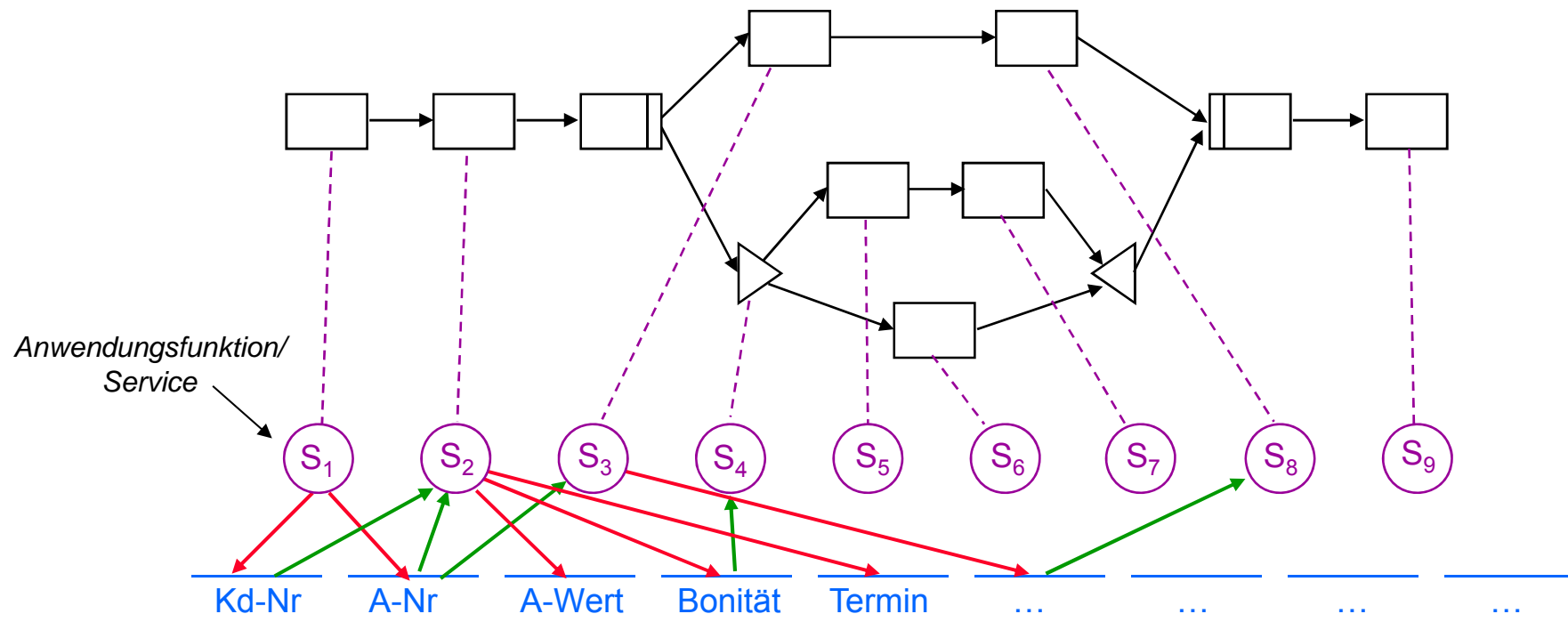


**Gewollte Schleife (Loop) oder versehentlicher Zyklus?**

**Rücksprung „mitten hinein“ gewollt?**

## 2.1 Korrektheitsaspekte von Prozessmodellen – eine Einführung

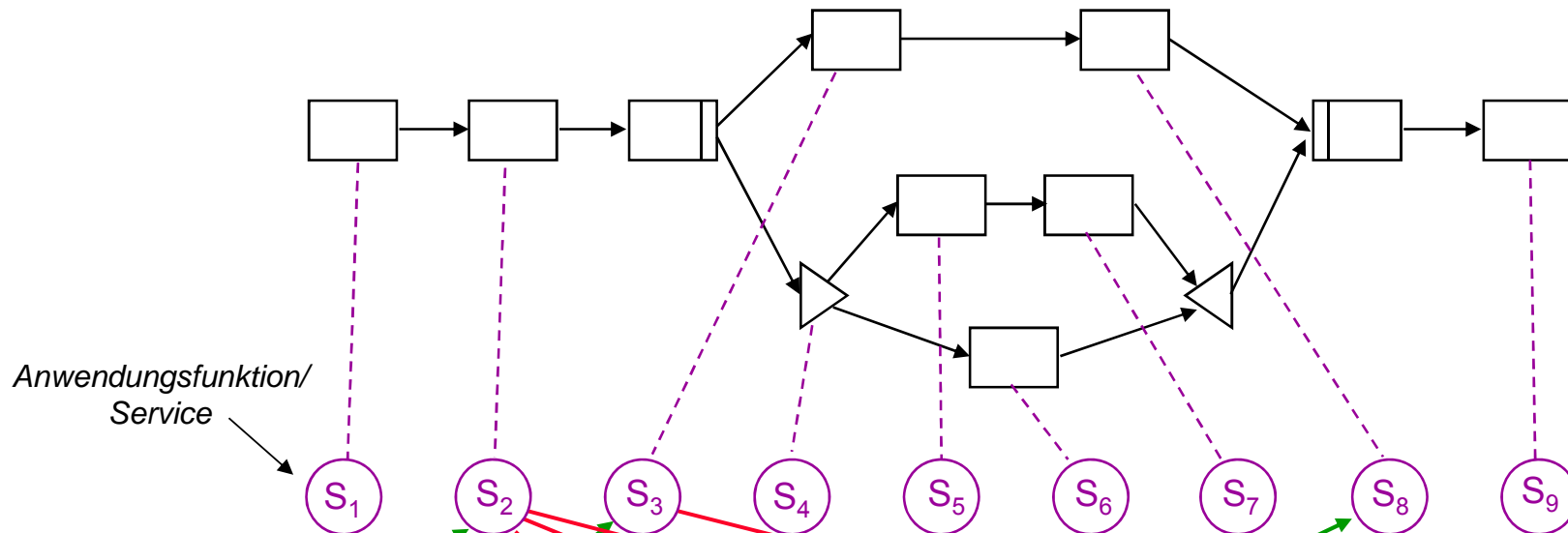
### □ Herausforderung 2: Korrektheit der Datenflüsse



## 2.1 Korrektheitsaspekte von Prozessmodellen – eine Einführung

### □ Herausforderung 2: Korrektheit der Datenflüsse

**Kontrollfluss darf nicht in Widerspruch zum Datenfluss stehen, sonst drohen Laufzeitfehler!**



**Wenn das PMS den Datenfluss ignoriert, drohen Laufzeitfehler!  
Ad-hoc-Abweichungen (siehe später) dann „at the user's risk“!**

## 2.1 Korrektheitsaspekte von Prozessmodellen – eine Einführung

---

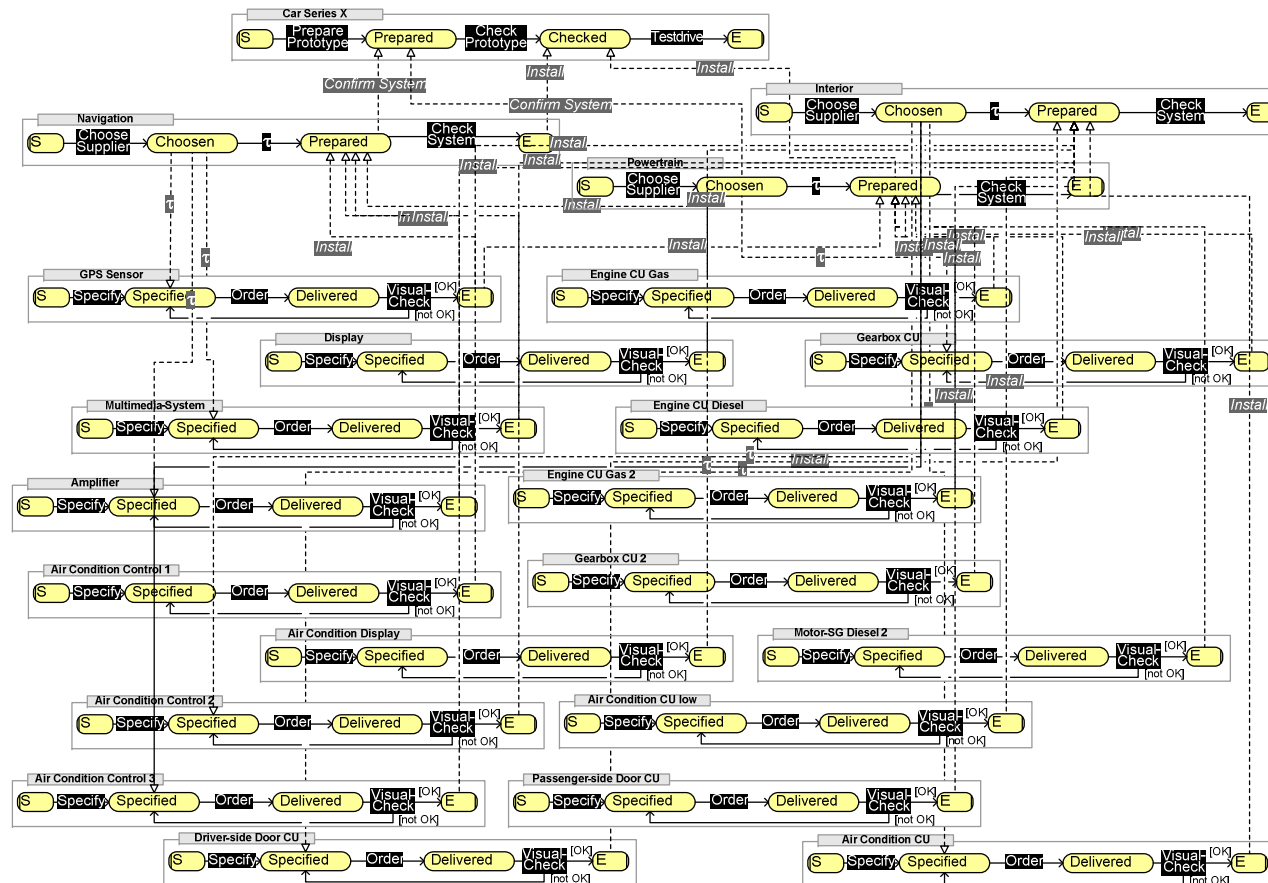
### □ Mögliche Modellierungsfehler in der Übersicht (Auswahl)

- Datenfluss in Widerspruch zu Kontrollfluss
- Unversorgte Aufrufparameter von Aktivitäten
- Deadlock (Prozess gerät in einen nicht geplanten „Endzustand“)
- Ungewünschte Zyklen
- Pfade, die nie durchlaufen werden können
- Teilspezifizierte XOR-Verweigungen
- Überlappende XOR-Prädikate
- ...

## 2.1 Korrektheitsaspekte von Prozessmodellen – eine Einführung

### □ „Korrektheit“ von Prozessen – ist das tatsächlich ein Problem?

- Bei kleinen, überschaubaren Prozessmodellen mag das Fehlen von systemseitigen Prüfungen ja noch kein großes Problem sein
- wird aber bei größeren und komplexeren Prozessmodellen schnell zum Alptraum



# Inhalt

---

## 2.0 Vorbemerkungen

## 2.1 Korrektheitsaspekte von Prozessmodellen – eine Einführung

## 2.2 Klassische Petri-Netze

## 2.3 Höhere Petri-Netze

## 2.4 Workflow-Netze

## 2.5 Aktivitätennetze

## 2.6 AristaFlow-Prozessmodell

## 2.7 Andere Ansätze

## 2.8 Ausdrucksmächtigkeit von Prozessbeschreibungssprachen

## 2.9 Abschließende Bemerkungen

## 2.10 Weiterführende Literatur

## 2.2 Klassische Petri-Netze

---

### Inhalt

2.2.1 Hintergrund

2.2.2 Statische Elemente

2.2.3 Dynamische Elemente

2.2.4 Ausführungsverhalten von Bedingungs-/Ereignisnetzen (B/E-Netzen)

2.2.5 Stellen-/Transitionsnetze (S/T-Netze)

2.2.6 Formale Analysen

## 2.2 Klassische Petri-Netze

---

### 2.2.1 Hintergrund

- ❑ 1962 in der Dissertation von Carl Adam Petri auf Basis der Graphentheorie entwickelt
- ❑ Ziel: Modellierung, Analyse und Simulation von dynamischen Systemen mit nebenläufigen und nichtdeterministischen Vorgängen
  - kommunikative Zusammenhängen zwischen Systemkomponenten
  - kausallogische Zusammenhänge zwischen Systemkomponenten
  - parallele Prozesse
  - Handlungsabläufe und Informationsflüsse
- ❑ In der Grundform allerdings eher „low-level“ Konzepte, deshalb in der Grundform i.d.R. nicht direkt einsetzbar
  - Inspirierte aber viele Untersuchungen über formale Eigenschaften von (Teilklassen von) Petri-Netzen, z.B. Verklemmungsfreiheit
  - diente als Ausgangsbasis vieler Weiterentwicklungen ↓ „Höhere“ Petri-Netze
- ❑ Die Workflowbeschreibungssprachen und/oder Ausführungsmodelle vieler Prozess-Management-Systeme basieren im Kern auf den Petri-Netz-Konzepten



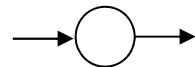
## 2.2 Klassische Petri-Netze

---

### 2.2.2 Statische Elemente

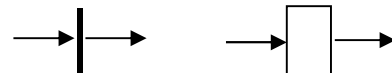
- Beschreibungselemente (werden im Folgenden verwendet)

- **Stellen**, im Folgenden dargestellt durch Kreise,



beschreiben **Zustände** oder **Bedingungen** und stehen für die momentane Lage eines Systems bzw. den Stand eines Prozesses

- **Transitionen**, im Folgenden dargestellt durch senkrechte Striche oder Rechtecke,



symbolisieren **Ereignisse** (bzw. Aktionen) und bewirken den **Übergang** in einen neuen Zustand

- **Kanten**, sie verbinden Stellen mit Transitionen (und umgekehrt)

## 2.2 Klassische Petri-Netze

---

- Ein Petri-Netz beschreibt immer gewisse statische Beziehungen zwischen den Komponenten eines Systems, wobei **Stellen** *passive Komponenten* (Bedingungen, Zustände) und **Transitionen** *aktive Komponenten* (Ereignisse, Aktionen) darstellen.
- Beispiel für Zustände und mögliche Ereignisse

Zustand	Ereignis
ledig	Trauung
Ampel ist rot	Umschalten Ampelfarbe
Lager ist leer	Lagerzugang
Zug steht	Zug fährt los
Zug fährt	Zug bremst

## 2.2 Klassische Petri-Netze

---

### □ Definition 2-1: Petri-Netz

Ein **Petri-Netz** ist ein Tripel  $PN = (P, T, F)$  mit

$$P \cap T = \emptyset$$

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

Hierbei repräsentiert ***P*** die **Menge der Stellen** (*places*), ***T*** die **Menge der Transitionen** (*transitions*) und ***F*** beschreibt die **Menge der gerichteten Kanten (Pfeile)**, d.h. die **Flussrelation** (*flow relation*) zwischen *S* und *T*.

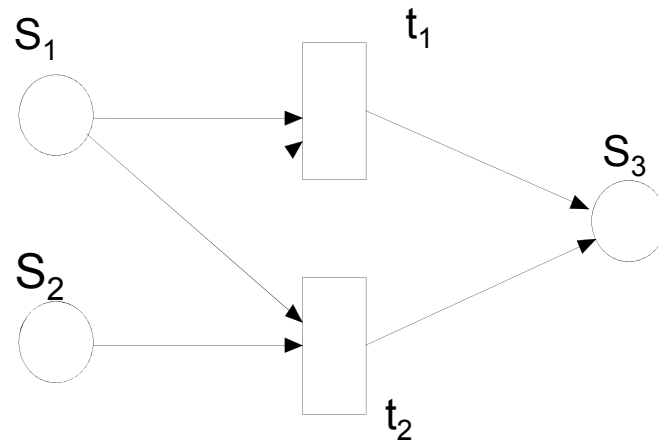
### □ Anmerkungen :

- Es werden in den einfachen, im Folgenden betrachteten Petri-Netzen nur Stellen mit Transitionen und umgekehrt verbunden, nicht jedoch Stellen mit Stellen oder Transitionen mit Transitionen.
- Diese Petri-Netze gehören somit in die Klasse der *bipartiten* Graphen.

## 2.2 Klassische Petri-Netze

---

□ Beispiel:



P =

T =

F =


## 2.2 Klassische Petri-Netze

---

### □ Definition 2-2: Vorbereich, Nachbereich

Es gelte wieder  $N = (P, T, F)$  und sei  $K := S \cup T$ . Die **Menge der Vorgänger-Knoten**  $P(x)$  eines Knotens  $x \in K$  bezeichnet man als den **Vorbereich** von  $x$ . Schreibweise im Folgenden:  $\bullet x$ . Formal:  $\bullet x := \{ y \in K \mid (y, x) \in F \}$ .

Die **Menge der Nachfolger-Knoten**  $S(x)$  eines Knotens  $x$  bezeichnet man als den **Nachbereich** von  $x$ . Schreibweise:  $x\bullet$ . Formal:  $x\bullet := \{ y \in K \mid (x, y) \in F \}$ .

### □ Mit anderen Worten

Die Stellen, von denen eine Kante zu einer Transition  $t$  führt, bilden den Vorbereich  $\bullet t$  von  $t$  und die Stellen, zu denen von  $t$  eine Kante hinführt, den Nachbereich  $t\bullet$  von  $t$ .

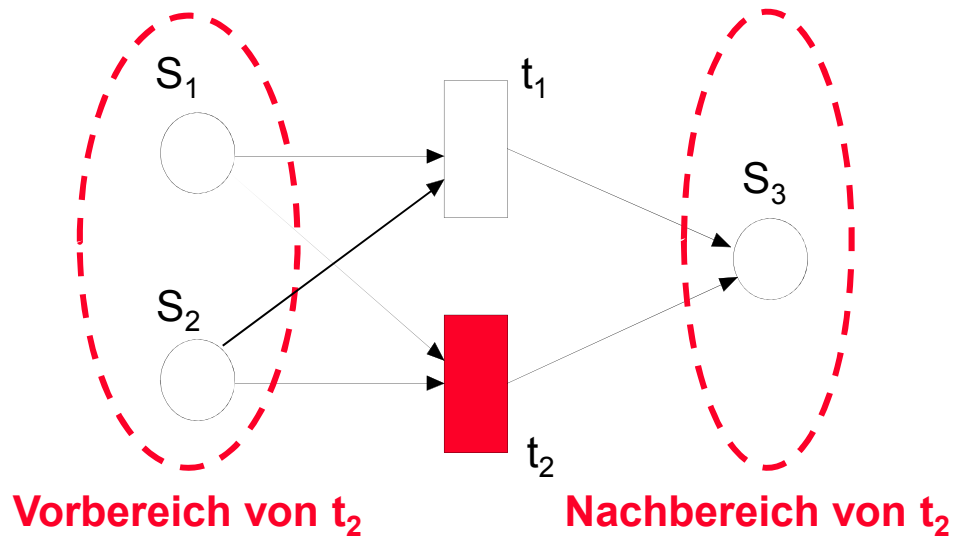
Entsprechend gilt:

Die Transitionen, von denen eine Kante zu einer Stelle  $s$  führt, bilden den Vorbereich  $\bullet s$  von  $s$  und die Transitionen, zu denen von  $s$  eine Kante hinführt, den Nachbereich  $s\bullet$  von  $s$ .

## 2.2 Klassische Petri-Netze

---

□ Beispiel:



P =	
T =	
F =	

## 2.2 Klassische Petri-Netze

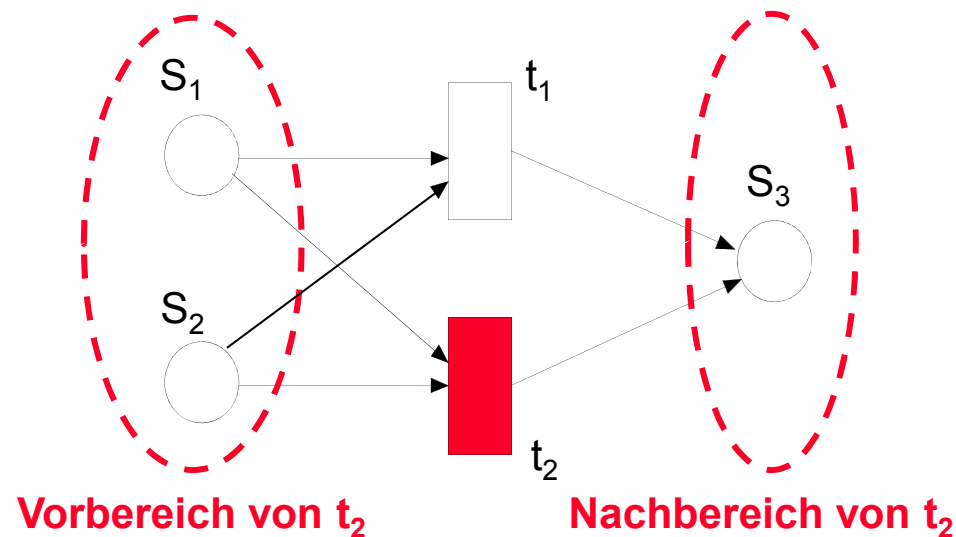
---

### □ Formaler Zusammenhang zwischen Zuständen und Ereignissen sowie Vorbedingungen und Nachbedingungen

- Jedes Ereignis setzt eine exakt definierbare Menge realisierter Zustände (Vorbedingungen) und/oder eine exakt definierte Menge von erreichbaren Zuständen (Nachbedingungen) voraus.

... oder anders ausgedrückt

- Jeder Zustand wird durch mindestens ein Ereignis aufgehoben (beendet) und/oder durch mindestens ein Ereignis eingeleitet (realisiert).



## 2.2 Klassische Petri-Netze

---

### □ Dies bedeutet bzw. daraus folgt:

- Zustände bzw. Ereignisse sind im Rahmen der Petri-Netz-Theorie stets durch ihre unmittelbare, *lokale Umgebung* festgelegt bzw. beschreibbar.
- Die Tatsache, dass ein auslösender Zustand selbst wieder durch ein Ereignis realisiert wurde, ist für die Beschreibung des nachfolgenden Ereignisses nicht bedeutsam.
- Dieser Sachverhalt erlaubt u.a. die **modulare Modellierung** von Systemen  
D.h. die Zustandsknoten können selbst wiederum Ergebnis oder Auslöser von Ereignissen sein (siehe später)

### □ Sehr häufig anzutreffende Petri-Netz-Konvention

**Ein Zustand soll nicht gleichzeitig Vor- und Nachbedingung für eine Transition (Ereignis) sein.**

Der Sinn dieser Konvention wird im nächsten Abschnitt („Dynamische“ Elemente der Petri-Netz-Theorie) verständlich.

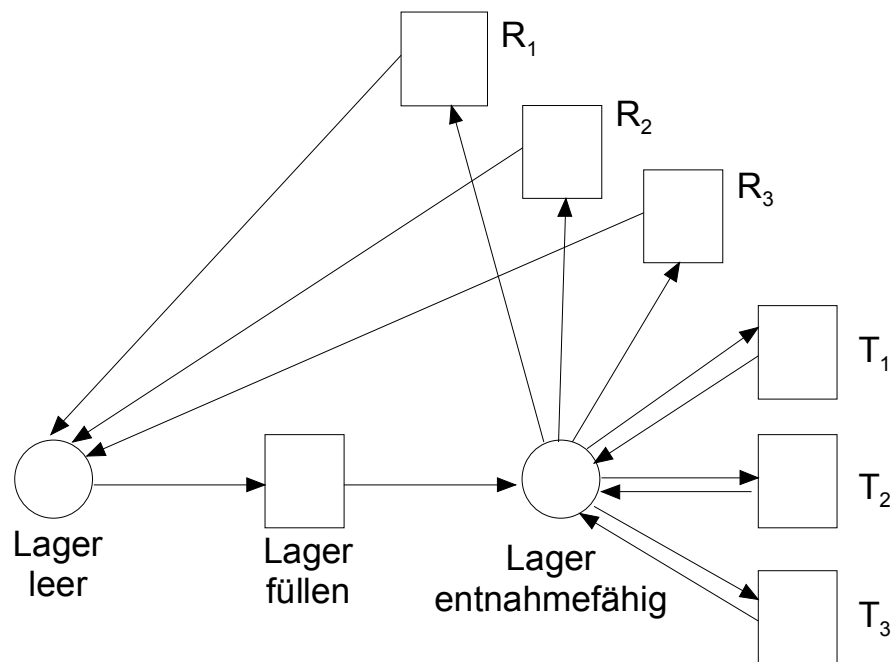


## 2.2 Klassische Petri-Netze

### □ Beispiel:

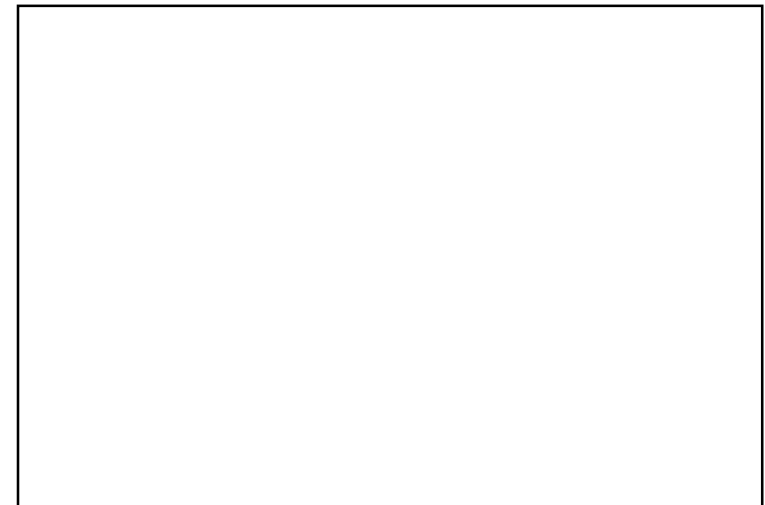
Drei Verbraucher haben Zugang zu einem Lager. Sie können Teile des Lagerbestandes entnehmen oder das Lager auch ganz räumen. Bei Teilentnahme können weitere Entnahmen stattfinden (exogene Bedingung). Bei Räumung muss das Lager durch den Beschaffer in einer Sonderaktion (aber zeitlos!) aufgefüllt werden.

Bezeichne im Folgenden  $T_i$  bzw.  $R_i$  eine Teilentnahme bzw. Räumung durch Verbraucher  $i$  ( $i = 1,2,3$ )



Diese Darstellung erfüllt nicht diese Petri-Netz-Konvention.

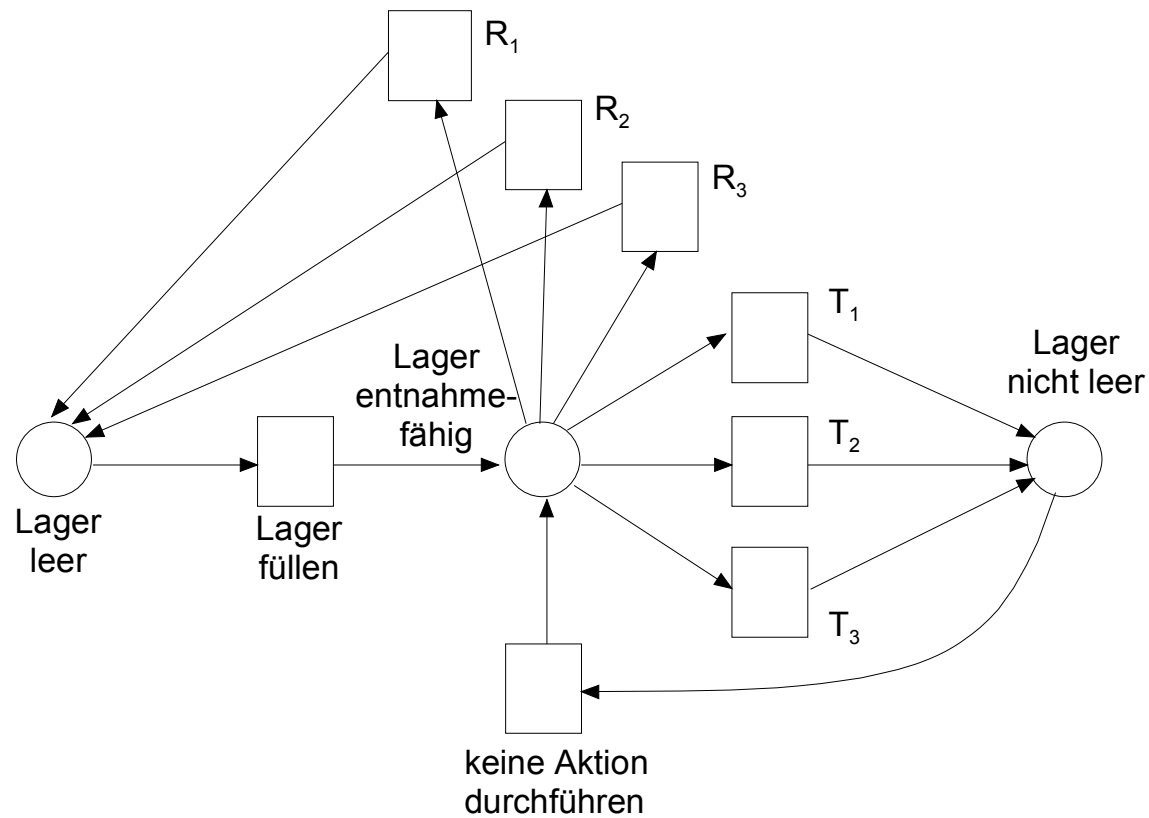
Begründung:



## 2.2 Klassische Petri-Netze

### □ Beispiel (Forts.)

(Mögliche) Korrekte Lösung:



## 2.2 Klassische Petri-Netze

---

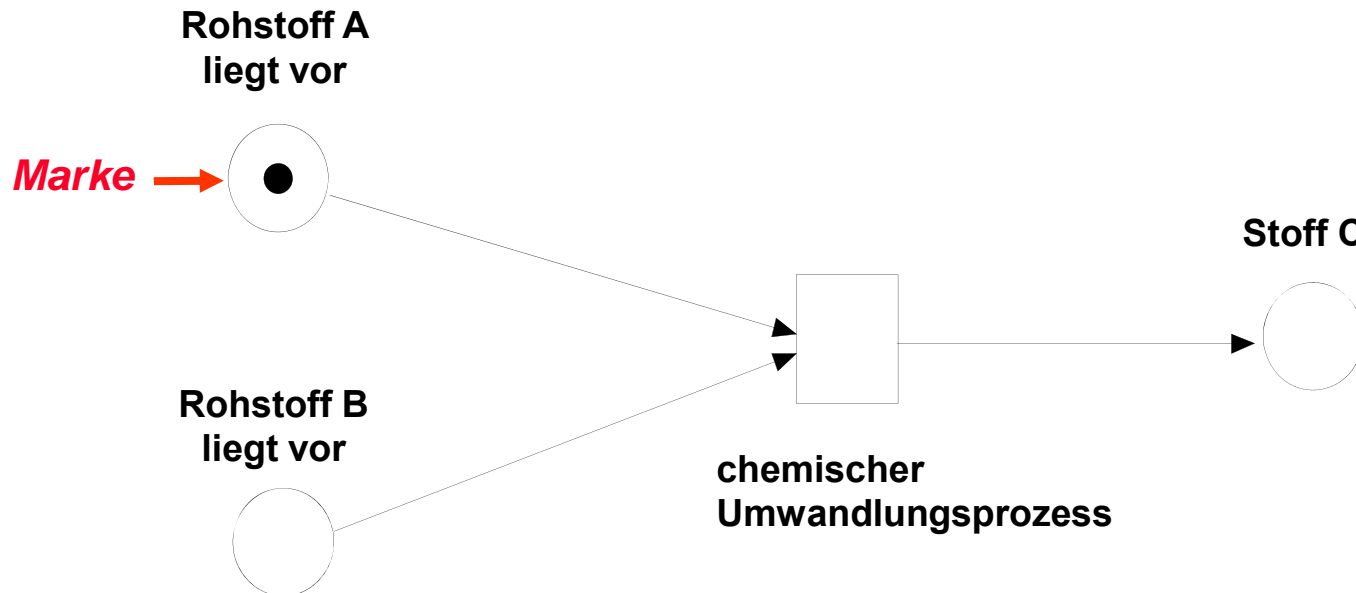
### 2.2.3 Dynamische Elemente

- ❑ Dynamische Elemente: **Marken (Token)**
- ❑ Grundlage für die Beschreibung des Zustands und des (Ausführungs-)Verhaltens von Petri-Netzen
- ❑ Einfachster Fall: **Bedingungs-/Ereignis-Netz (B/E-Netz)**
  - Interpretation von Stellen und Transitionen:
    - Stellen = Bedingungen
    - Transitionen = Ereignisse
  - Jede Stelle kann eine Marke (Token) tragen
    - Ist eine Stelle s markiert, so ist die durch s beschriebene Bedingung erfüllt
    - Nicht markierte Stellen entsprechen ungültigen Bedingungen
- ❑ Die Zuordnung von Marken zu Stellen eines Netzes heißt **Markierung**

## 2.2 Klassische Petri-Netze

---

□ Beispiel:



- Sind für ein Ereignis nicht alle notwendigen Bedingungen (Voraussetzungen) erfüllt, so kann dieses Ereignis aufgrund der kausallogischen Zusammenhänge nicht stattfinden.
- Bezogen auf das obige Beispiel heißt das, dass der chemische Umwandlungsprozess (in dieser Situation) nicht stattfinden kann

## 2.2 Klassische Petri-Netze

### □ Definition 2-3:

#### **Schalten eines Ereignisses, aktiviertes Ereignis, Schaltregel, Transitionsregel**

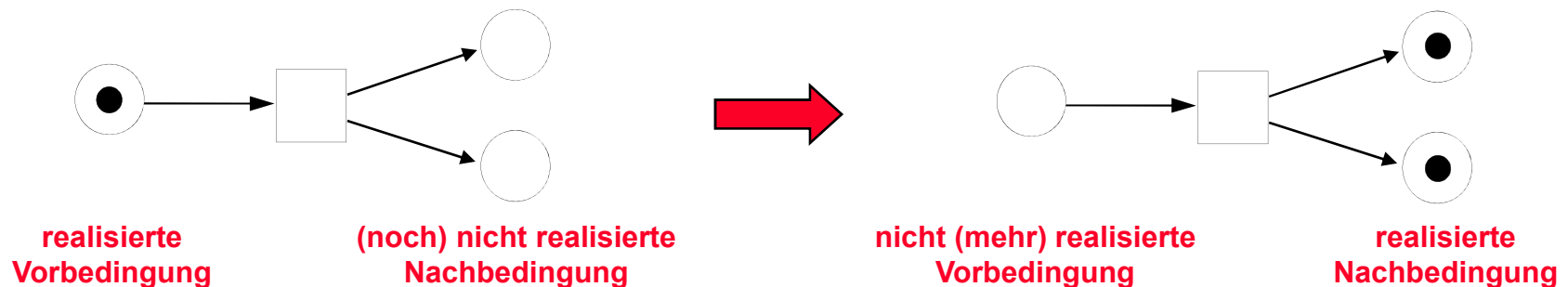
Ein **Ereignis** kann in der Petri-Netz-Theorie nur dann stattfinden („**schalten**“), wenn es **aktiviert** ist.

Ein Ereignis heißt (in einem B/E-Netz) aktiviert, wenn

- alle seine Eingangszustände mit Marken belegt sind und
- alle seine Ausgangszustände markenfrei sind.

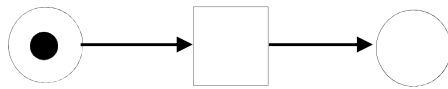
**Schaltregel** (auch **Transitionsregel** genannt): Findet ein Ereignis (ein „Schaltvorgang“) statt, so werden seine Marken von seinen Eingangszuständen entfernt und seine Ausgangszustände mit je einer Marke belegt. (Sie gelten nun als realisiert bzw. wahr.)

### □ **Anwendung der Schaltregel**

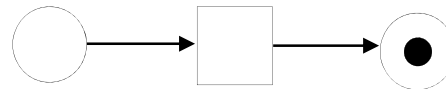


## 2.2 Klassische Petri-Netze

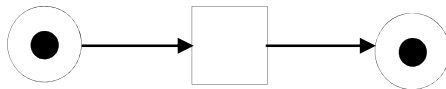
□ Welche der folgenden Transitionen können schalten?



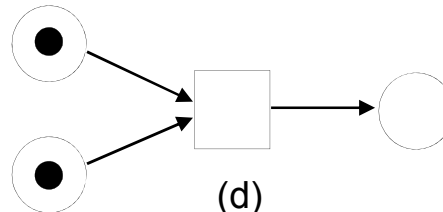
(a)



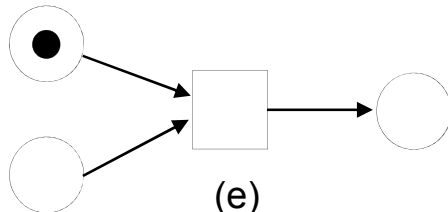
(b)



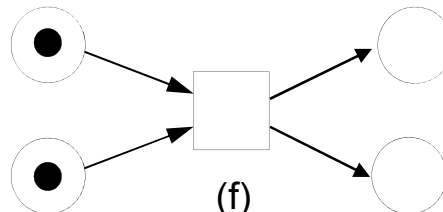
(c)



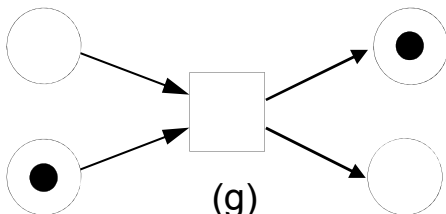
(d)



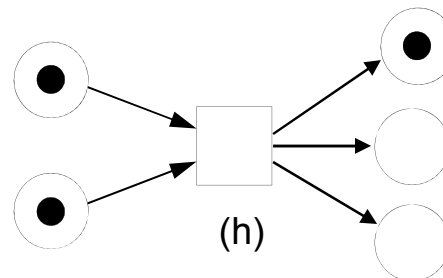
(e)



(f)



(g)



(h)

a)

b)

c)

d)

e)

f)

g)

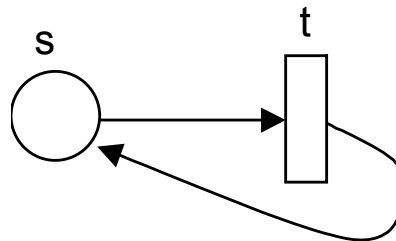
h)

## 2.2 Klassische Petri-Netze

---

### □ Wichtige Anmerkungen:

- Definition 2-3 besagt nur, unter welchen Voraussetzungen ein Schaltvorgang ein Schaltvorgang stattfinden kann und was dieser bewirkt.
- Es wird jedoch nichts darüber ausgesagt, wann dieser Schaltvorgang stattfindet.
- D.h. eine Transition muss nicht sofort schalten, wenn ihre Vor- und Nachbedingungen erfüllt sind.
- Falls also zu einem Zeitpunkt mehrere Transitionen schaltbereit sind, so heißt dies nicht, dass diese auch alle gleichzeitig schalten müssen.
- Definition 2-3 impliziert auch, dass (und warum) B/E-Netze („Ein-Marken-Netze“) schlingenfrei sein müssen.



Beispiel für Schlinge

## 2.2 Klassische Petri-Netze

---

### □ Definition 2-4: Markierung, Markierungsfunktion, Fall

Formal werden Markierungen eines B/E-Netzes durch Abbildungen

**M**: **S**  $\rightarrow$  { **TRUE**, **FALSE** } dargestellt.

**M** heißt dann *Markierungsfunktion*.

- **M(s) = TRUE**: Stelle s trägt eine Marke,  
d. h. die durch s beschriebene Bedingung ist erfüllt
- **M(s) = FALSE**: Stelle s trägt keine Marke,  
d. h. die durch s beschriebene Bedingung ist nicht erfüllt

Für eine **Markierung M** ist die Menge aller markierten Stellen gegeben durch  
**c** := { **s**  $\in$  **S** | **M(s) = TRUE** }. **c** heißt **Fall (case)**.



## 2.2 Klassische Petri-Netze

---

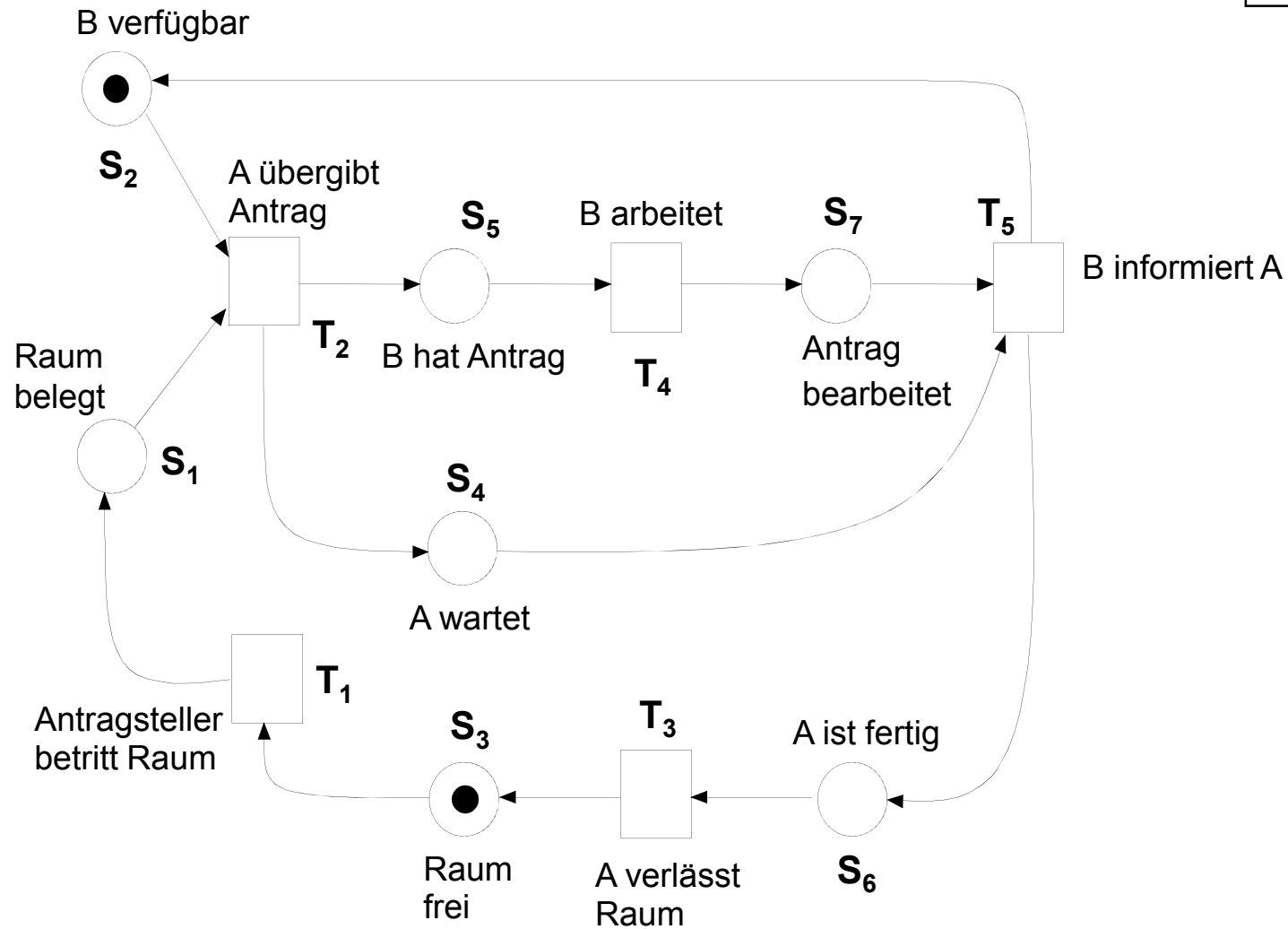
### 2.2.4 Ausführungsverhalten von Bedingungs-/Ereignisnetzen (B/E-Netzen)

#### □ Beispiel:

- Der Schalterraum einer Behörde ist mit einem Beamten B besetzt, der für die Bearbeitung eines Antrages zuständig ist.
- Die Bearbeitung erfolgt in Anwesenheit des Antragstellers A.
- Aus Datenschutzgründen darf stets nur ein Antragsteller den Raum betreten.

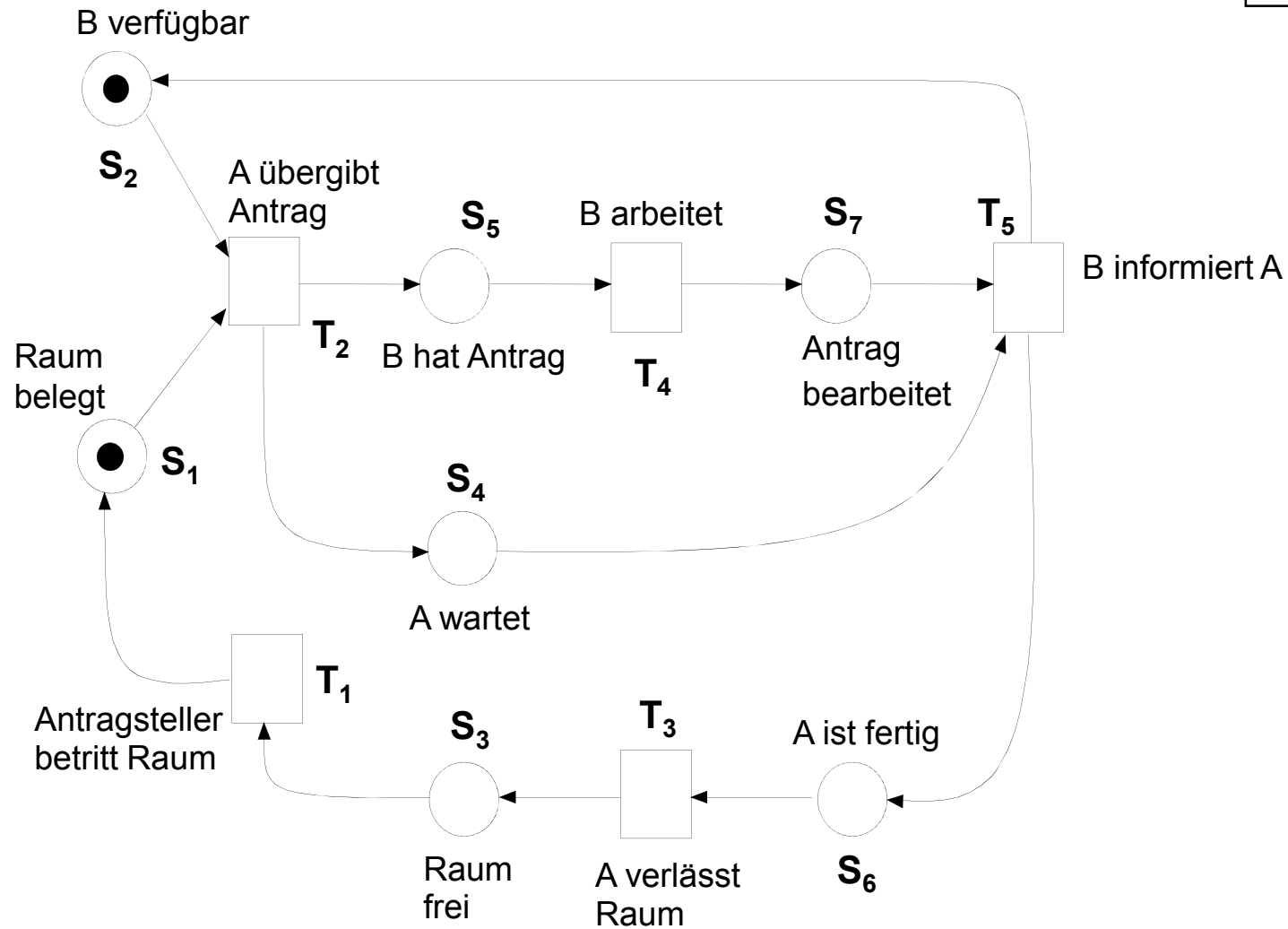
## 2.2 Klassische Petri-Netze

Beispiel (Forts.)

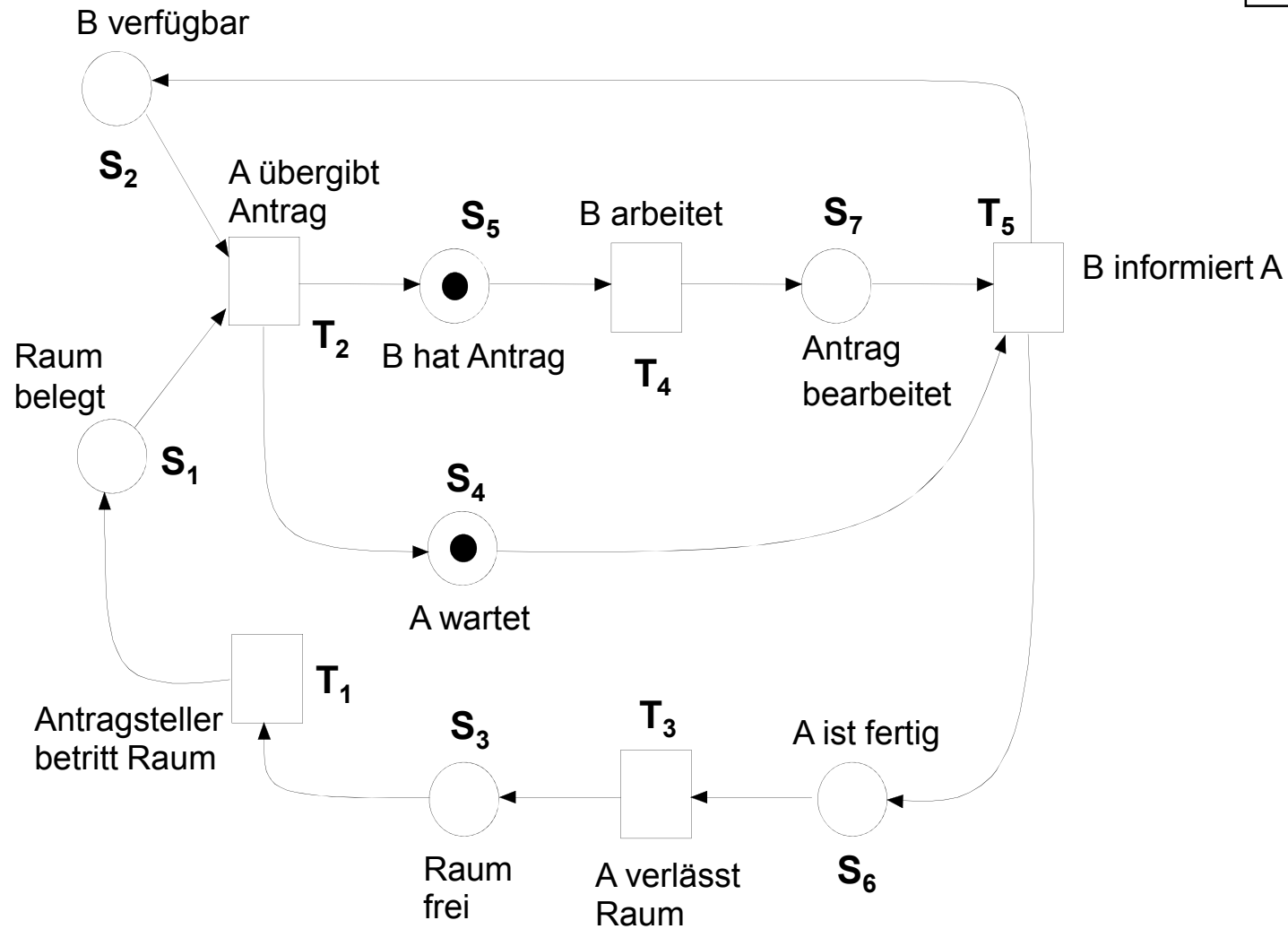


## 2.2 Klassische Petri-Netze

Beispiel (Forts.)

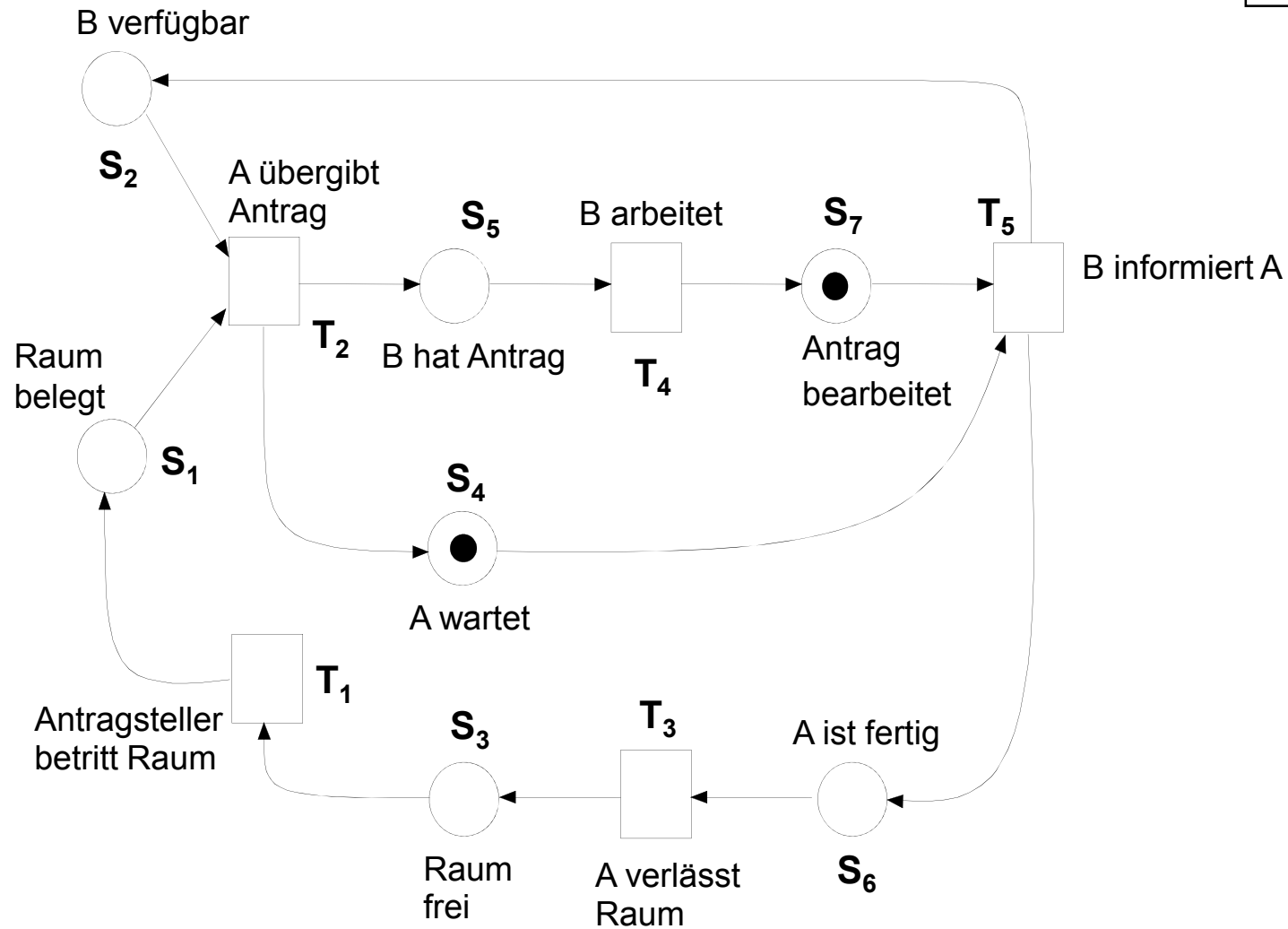


#### Beispiel (Forts.)



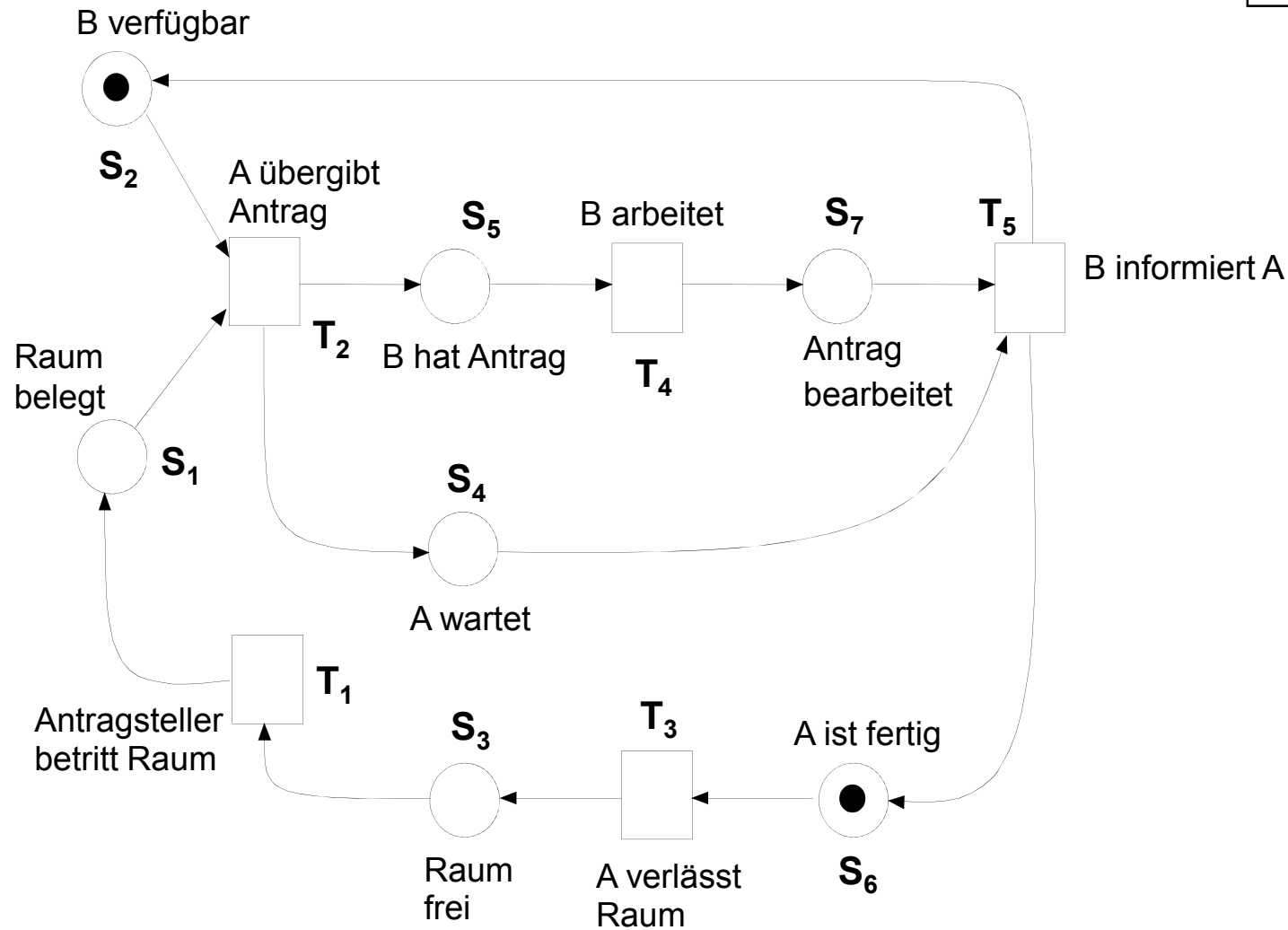
## 2.2 Klassische Petri-Netze

Beispiel (Forts.)



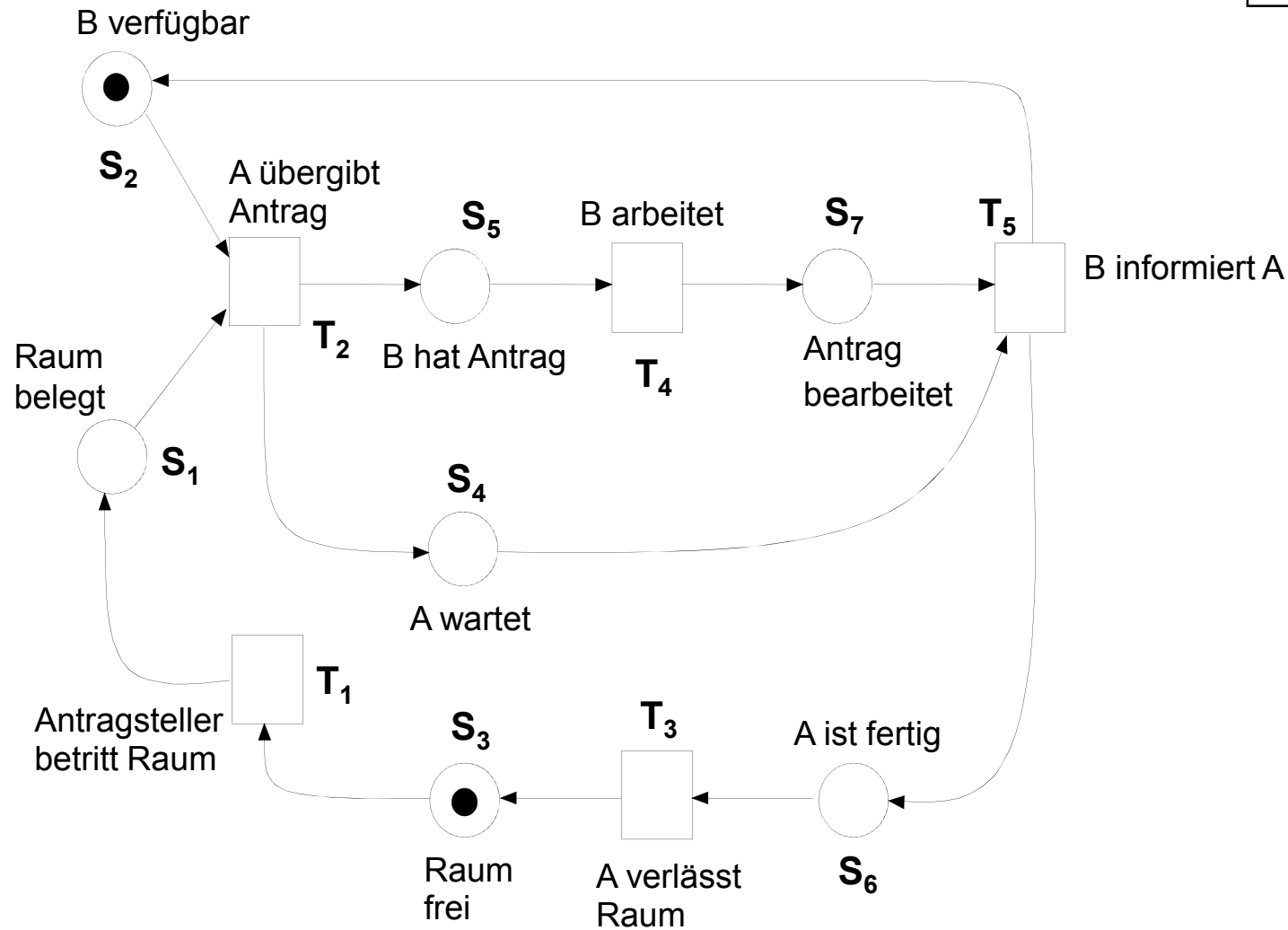
## 2.2 Klassische Petri-Netze

Beispiel (Forts.)



## 2.2 Klassische Petri-Netze

Beispiel (Forts.)




## 2.2 Klassische Petri-Netze

---

### □ Beispiel (Forts.)

**Erreichbare Markierungen und Schaltverhalten:**

Zeit	Stellen							Schaltbereite Transitionen				
	S1	S2	S3	S4	S5	S6	S7	T1	T2	T3	T4	T5
0												
1												
2												
3												
4												
5												





## 2.2 Klassische Petri-Netze

---

### □ Deterministische und nicht-deterministische Systeme

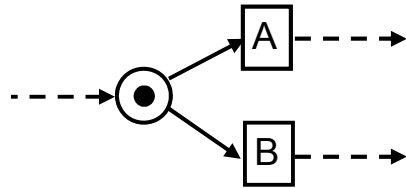
- Im vorherigen Schalterraum-Beispiel war der konkrete Ablauf fest vorbestimmt, die einzige „Unsicherheit“ war, zu welchem Zeitpunkt eine aktivierte Transition schaltet.  
Wir hatten es hier somit mit einem **deterministischen System** zu tun.
- Oft ist man jedoch an Systemen interessiert, die verschiedene Ausführungsalternativen aufweisen und wo man im System nicht regeln will bzw. kann, welche Alternative gewählt wird.  
In solchen Fällen haben wir es mit einem **nicht-deterministischen System** zu tun.
- Anwendungs-Beispiele:
  - Bei einer XOR-Entscheidung in einem Workflow hängt es von den Daten ab, welcher Ausführungspfad genommen wird. Die Entscheidungsregel ist im System aber nicht modelliert.
  - Ein Glückspielautomat „würfelt“. – Je nach Ergebnis wird ein Gewinn ausgewiesen oder nicht.
  - Fußgängerampeln an einer Kreuzung: Die Knöpfe zum Anfordern von „Grün“ können zu beliebigen Zeiten und in beliebigen Konstellationen betätigt werden.

## 2.2 Klassische Petri-Netze

### □ Deterministische und nicht-deterministische Systeme (Forts.)

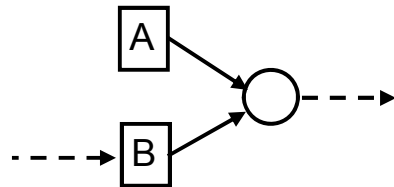
#### ○ Petri-Netz-Beispiele:

- XOR-Entscheidungen, ohne im System hinterlegte „Schaltregel“



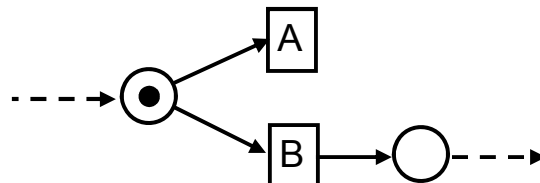
**Nicht-Determinismus:**

- Transitionen, die (aus Systemsicht) „spontan“ Marken ins System „einfüttern“



**Nicht-Determinismus:**

- Transitionen, die (aus Systemsicht) „spontan“ Marken „konsumieren“

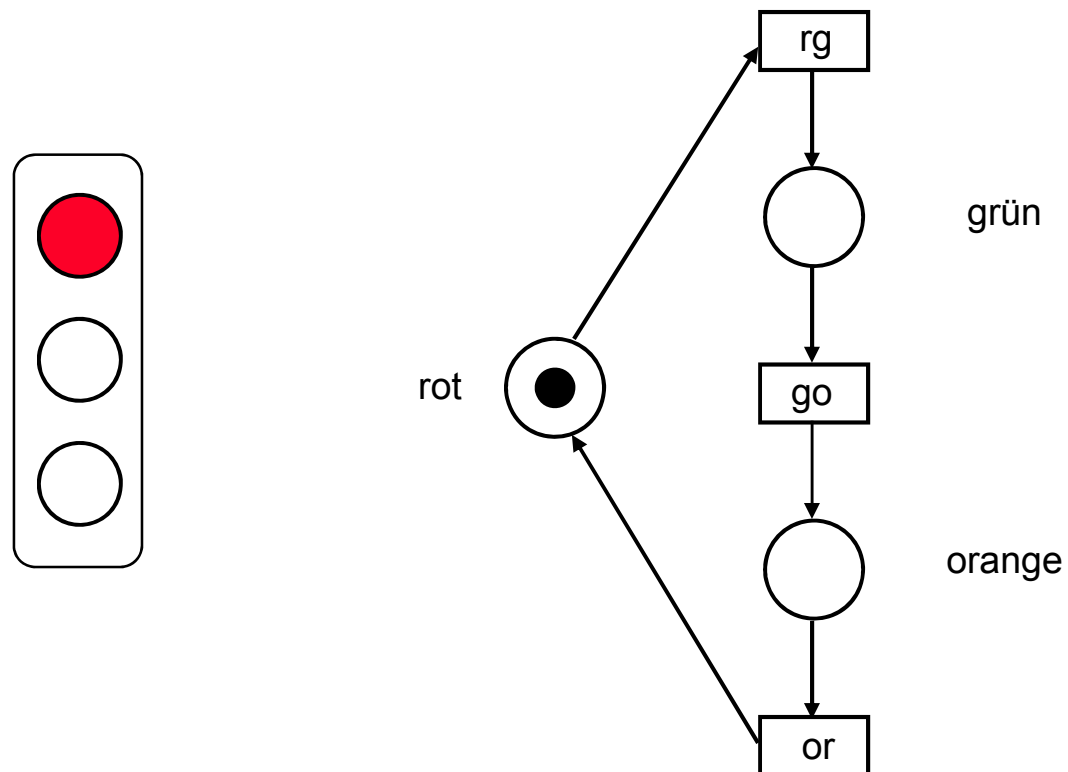


**Nicht-Determinismus:**

## 2.2 Klassische Petri-Netze

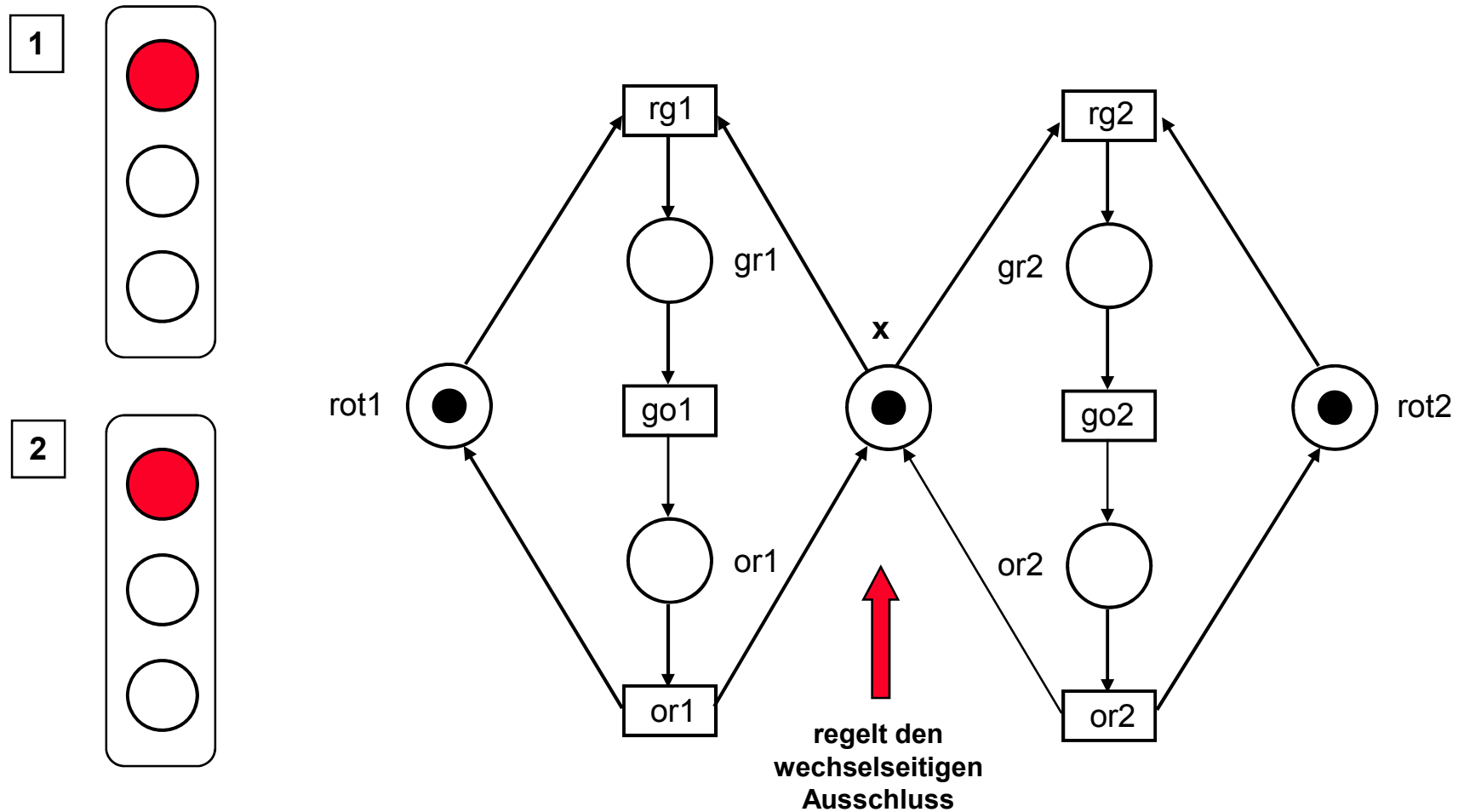
---

- **Aufgabe:** Modellierung des Verhaltens einer einzelnen Ampel



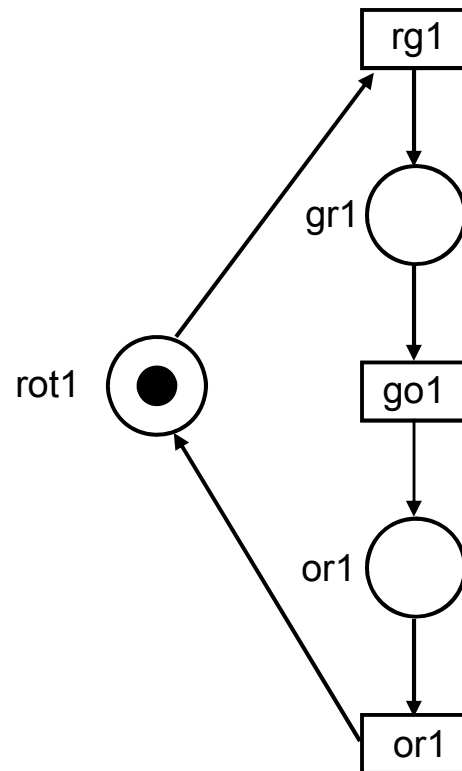
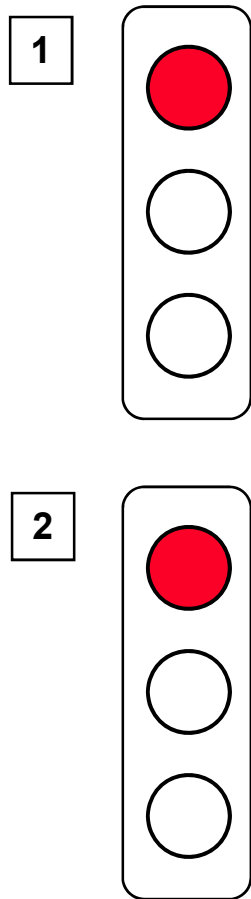
## 2.2 Klassische Petri-Netze

- **Aufgabe:** Modellierung des Verhaltens zweier konkurrierender Ampeln

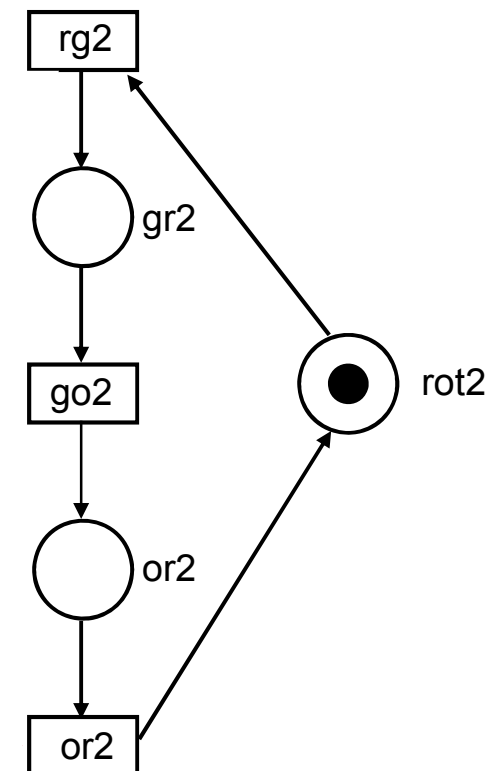


## 2.2 Klassische Petri-Netze

### □ Aufgabe: Alternierendes Schalten der beiden Ampeln



?



## 2.2 Klassische Petri-Netze

---

### 2.2.5 Stellen-/Transitionsnetze (S/T-Netze)

□ **Definition 2-5: Stellen-/Transitionsnetze (S/T-Netze)**

Ein 6-Tupel  $Y = (S, T, F, K, W, M_0)$  heißt **Stellen-Transitions-Netz** bzw. **S/T-Netz**, falls

$(S, T, F)$  ein Petri-Netz ist,

$K : S \rightarrow \mathbb{N} \cup \{\infty\}$  die **Kapazitäten** der Stellen (evtl. unbeschränkt),

$W : F \rightarrow \mathbb{N}$  die **Kantengewichte** der Kanten und

$M_0 : S \rightarrow \mathbb{N}_0$  die **Anfangsmarkierung** angibt, wobei  $\forall s \in S : M_0(s) \leq K(s)$ .

Anmerkung:

Manchmal unterscheidet man zwischen einem **Netz ohne Anfangsmarkierung** und einem solchen **mit Anfangsmarkierung** und spricht dann nur im ersten Fall von einem **S/T-Netz** und im zweiten Fall von einem **S/T-System**.

## 2.2 Klassische Petri-Netze

---

### □ Was ist anders als bei den B/E-Netzen?

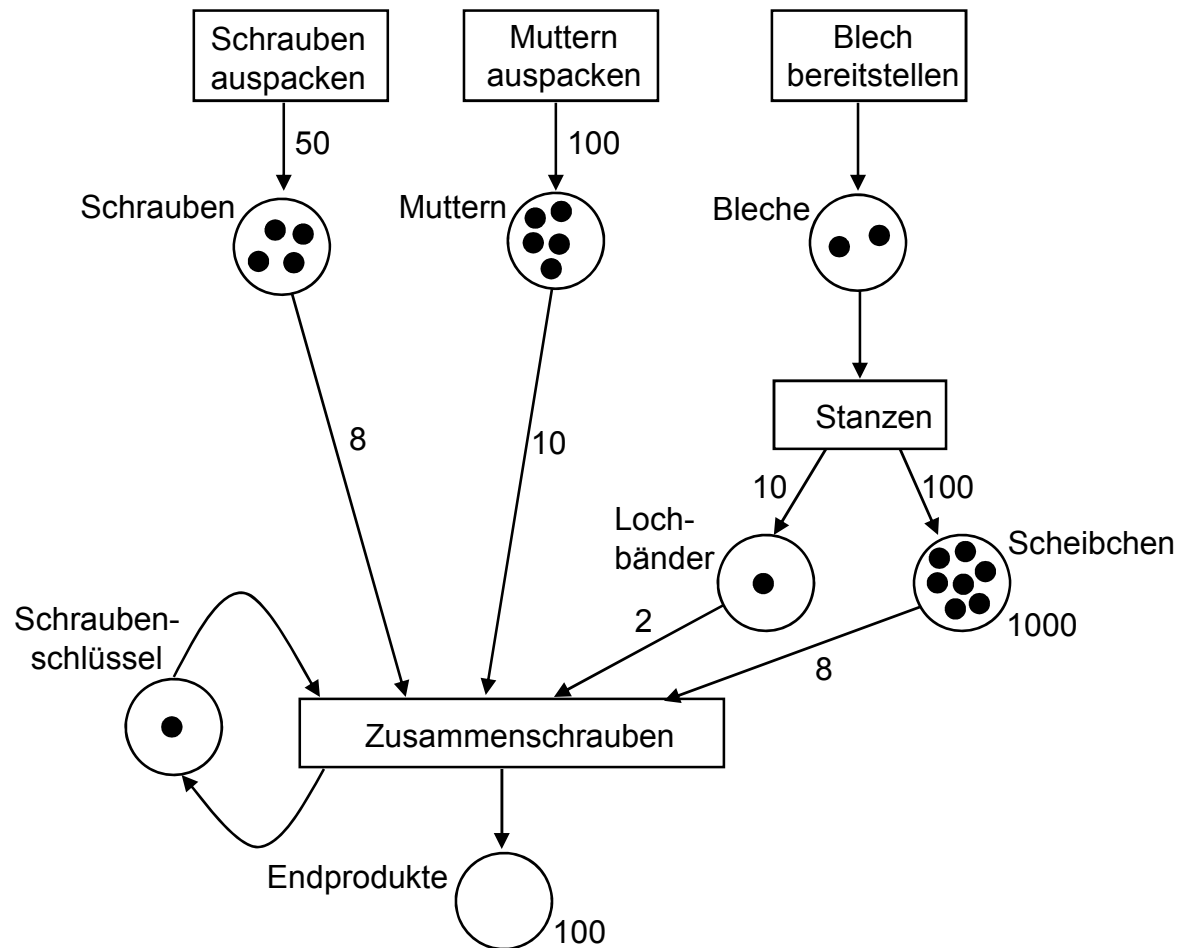
- Ergänzung des Netzes um eine **Kantengewichtsfunktion  $W$** , die jeder Kante eine natürliche Zahl als Gewicht zuordnet.
- Die Stellen  $s$  haben in einer Situation (Markierung  $M$ ) als Wert eine natürliche Zahl von Marken  $M(s)$ , die durch eine **Kapazität  $K(s)$**  nach oben beschränkt ist.
- **Stellen ohne Kapazitätsangabe** haben implizit die **Kapazität  $\infty$**  und **Kanten ohne Gewichtsangabe** haben implizit das **Kantengewicht 1**.

### □ ... oder anders ausgedrückt

- Stellen können jetzt mehrere (per Default sogar unendlich viele) Marken aufnehmen } wird durch die **Kapazität** einer Stelle festgelegt
- Es können beim Schalten einer Transition
  - von einer Stelle im Vorbereich evtl. mehrere Marken abgezogen werden
  - auf einer Stelle im Nachbereich evtl. mehrere Marken abgelegt werden} wird durch die **Kantengewichte** festgelegt

## 2.2 Klassische Petri-Netze

### □ Beispiel 1



### Erläuterungen

Beim Stanzen eines Bleches

Beim Auspacken von Muttern

Die Kapazität der Stelle Muttern

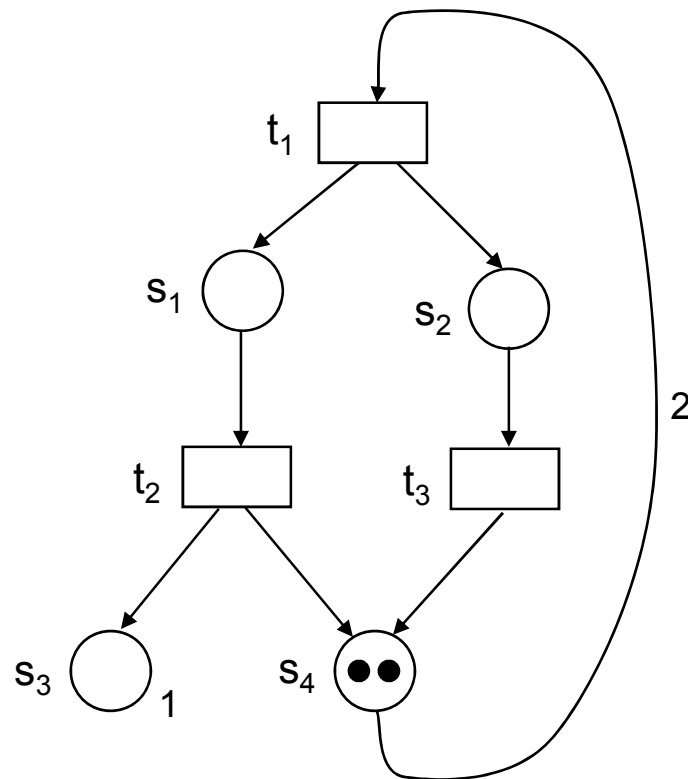
Die Kapazität der Stelle Scheibchen

Für ein Eindprodukt



## 2.2 Klassische Petri-Netze

### □ Beispiel 2



$$S = \{s_1, s_2, s_3, s_4\}$$

$$T = \{t_1, t_2, t_3\}$$

$$F = \{(t_1, s_1), (t_1, s_2), (s_1, t_2), (s_2, t_3), (t_2, s_3), (t_2, s_4), (t_3, s_4), (s_4, t_1)\}$$

$$K = \{(s_1, \infty), (s_2, \infty), (s_3, 1), (s_4, \infty)\}$$

$$W = \{((t_1, s_1), 1), ((t_1, s_2), 1), ((s_1, t_2), 1), ((s_2, t_3), 1), ((t_2, s_3), 1), ((t_2, s_4), 1), ((t_3, s_4), 1), ((s_4, t_1), 2)\}$$

$$M_0 = \{(s_1, 0), (s_2, 0), (s_3, 0), (s_4, 2)\}$$

## 2.2 Klassische Petri-Netze

---

### □ Anmerkungen

- S/T-Netze sind erheblich vielseitiger einsetzbar als B/E-Netze
- B/E-Netze sind ein Spezialfall der S/T-Netze

Quizfrage: Wie macht man ein S/T-Netz verhaltensmäßig zu einem B/E-Netz?





- Die Stellen können alles Mögliche repräsentieren, wie etwa
  - zur Bearbeitung anstehende Dokumente
  - Verfügbarkeit von Ressourcen (ja/nein)
  - der verfügbare Ressourcen-Umfang
- Die Kantengewichte können ebenfalls verschiedene Bedeutungen haben, wie etwa
  - Belegung einer Ressource (ja/nein)
  - Ressourcen-Verbrauch
  - Auffüllen von Ressourcen

## 2.2 Klassische Petri-Netze

□ **Definition 2-6: Aktivierte Transition unter M,  
Schalten einer Transition von M nach M', Folgemarkierung**

Eine Transition  $t \in T$  heißt **aktiviert unter M**, geschrieben  $M[t\rangle$ , wenn

1.  $\forall s \in \bullet t : M(s) \geq W(s, t)$  
2.  $\forall s \in t \bullet : M(s) + W(t, s) \leq K(s)$  

Wir sagen **t schaltet von M nach M'** und schreiben  $M[t\rangle M'$ , wenn t unter M aktiviert ist und M' aus M durch Entnahme von Marken aus den Eingangsstellen und Ablage von Marken auf die Ausgangsstellen gemäß den Kantengewichten entsteht:

$$M'(s) = \begin{cases} M(s) - W(s, t), & \text{falls } s \in \bullet t \setminus t \bullet \\ M(s) + W(s, t), & \text{falls } s \in t \bullet \setminus \bullet t \\ M(s) - W(s, t) + W(t, s), & \text{falls } s \in t \bullet \cap \bullet t \\ M(s), & \text{sonst} \end{cases}$$

M' heißt dann **(unmittelbare) Folgemarkierung** von M unter t.

Anmerkung: Die Aktivierungsbedingungen und die vorstehende Definition der Folgemarkierung bezeichnet man auch als die **Schaltregel** des betrachteten S/T-Netzes.

## 2.2 Klassische Petri-Netze

---

### □ Anmerkungen

- Die Bedingungen für die Aktivierbarkeit sorgen dafür, dass beim Schalten
  - einerseits genügend Marken entsprechend den Kantengewichten vorhanden sind, so dass die Markierung keiner Stelle unter Null sinkt (siehe ⌚ in Definition 2-6), und
  - andererseits die Kapazität an keiner Stelle überschritten wird (siehe ⌚ in Definition 2-6).
- Auch nach einer Schaltung erhält man wieder eine legale Markierung  $M'$  mit  $\forall s \in S : 0 \leq M'(s) \leq K(s)$

## 2.2 Klassische Petri-Netze

---

### 2.2.6 Formale Analysen

#### □ Typische Fragestellungen

- Zeigt das modellierte Petri-Netz das gewünschte Verhalten?
- In welche Zustände kann das Netz gelangen?
- Gibt es erreichbare Zustände, in denen keine Transition mehr schaltbereit ist (d.h. können Deadlocks auftreten)?
- Müssen (unbeschränkte) Stellen mehr Marken aufnehmen als gewünscht?
- ...

#### □ **Definition 2-7: Erreichbarer Zustand (bzw. erreichbare Markierung)**

Von der aktuellen Markierung des Netzes aus über eine Schaltfolge aktivierter Transitionen erreichbare Folgemarkierung.

## 2.2 Klassische Petri-Netze

---

### □ Definition 2-8: Erreichbarkeitsmenge, Erreichbarkeitsgraph

Sei  $M_0$  eine Ausgangsmarkierung, dann bezeichnet  $[M_0\rangle$  die Menge aller möglichen Markierungen, die – ausgehend von  $M_0$  – unter Verwendung der Schaltregeln erreicht werden können.

Man bezeichnet  $[M_0\rangle$  daher auch als **Erreichbarkeitsmenge** des S/T-Netzes.

Der **Erreichbarkeitsgraph**  $G(Y)$  eines S/T-Netzes  $Y = (\vec{N}, M_0)$  beschreibt, auf welchem Weg (d.h. mittels welcher Transitionen) eine bestimmte Markierung  $M_i \in [M_0\rangle$  erreicht werden kann:

$$G(Y) := \{ (M_1, t, M_2) \in [M_0\rangle \times T \times [M_0\rangle \mid M_1[t\rangle M_2 \}.$$

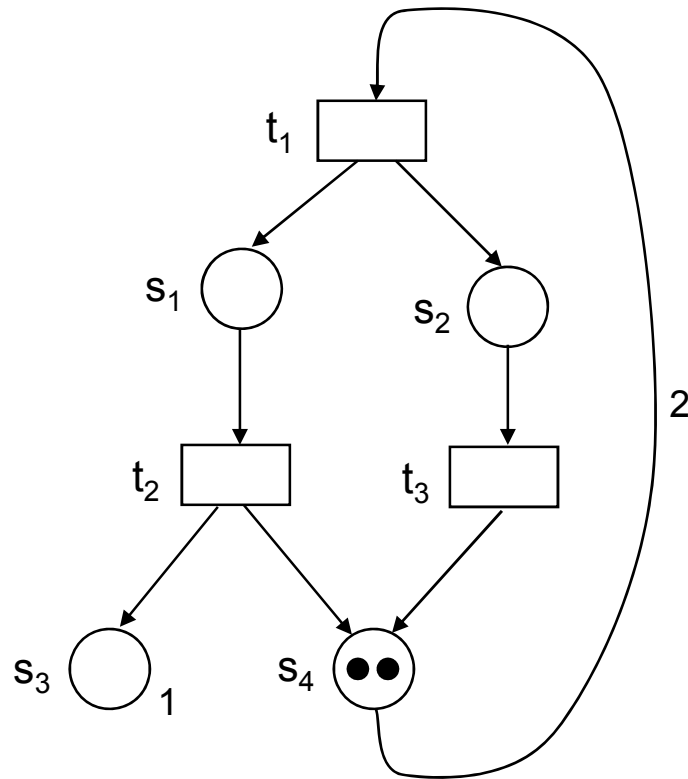
### Anmerkung:

Bestimmung der Erreichbarkeitsmenge für viele Fragestellungen relevant, wie z.B.

- kann eine bestimmte – z.B. unerwünschte – Markierungssituation eintreten?
- kann es umgekehrt sein, dass eine bestimmte Markierung nie erreicht wird?
- usw.

## 2.2 Klassische Petri-Netze

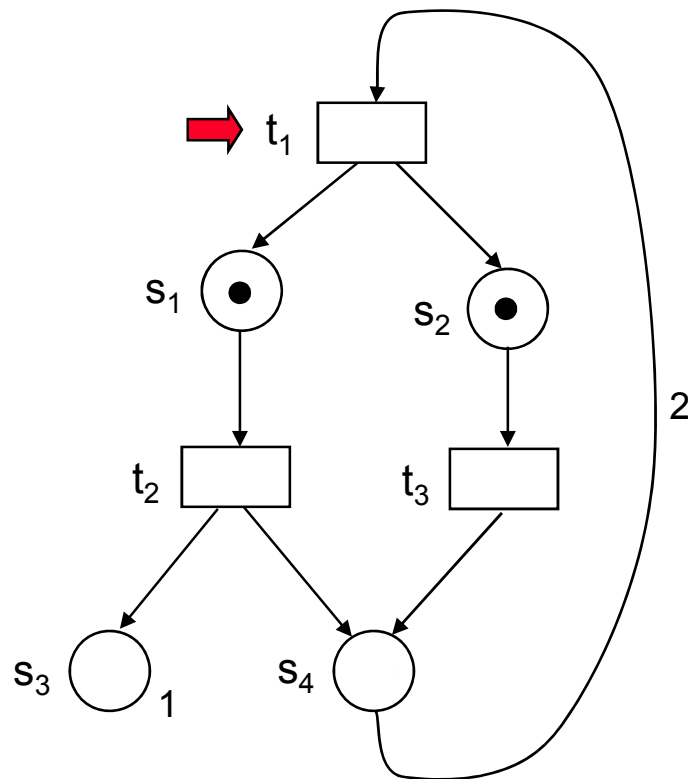
### □ Erreichbarkeitsanalyse



Nr.	$S_1$	$S_2$	$S_3$	$S_4$	aktivierte Transitionen
$M_0$	0	0	0	2	$t_1 \rightarrow M_1$
$M_1$					

## 2.2 Klassische Petri-Netze

### □ Erreichbarkeitsanalyse (Forts.)

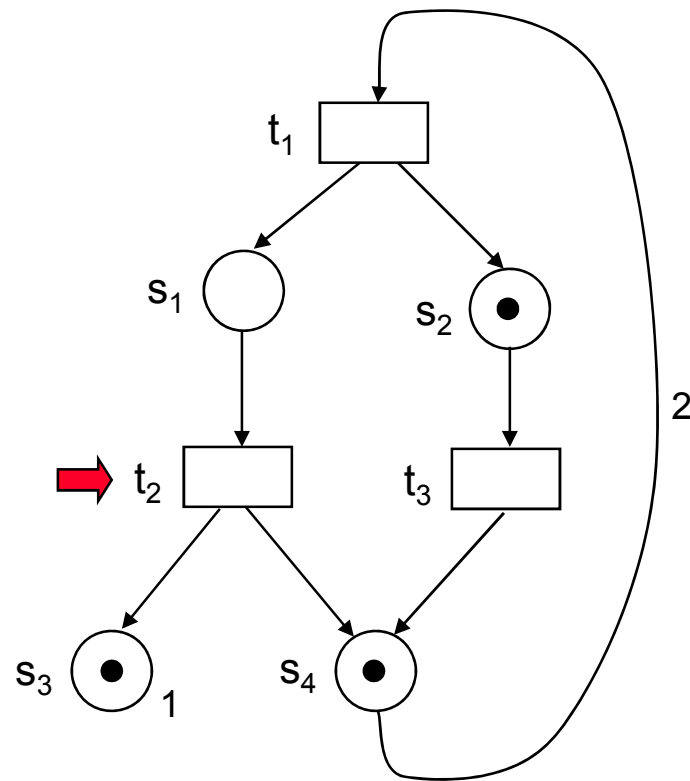


Nr.	$S_1$	$S_2$	$S_3$	$S_4$	aktivierte Transitionen
$M_0$	0	0	0	2	$t_1 \rightarrow M_1$ ←
$M_1$	1	1	0	0	$t_2 \rightarrow M_2$ $t_3 \rightarrow M_3$
$M_2$					
$M_3$					



## 2.2 Klassische Petri-Netze

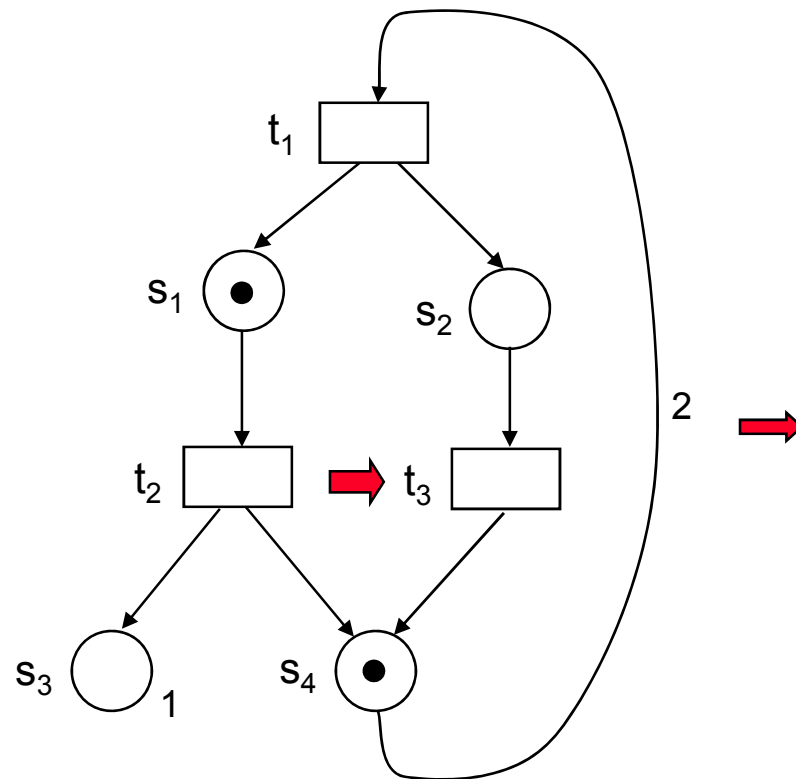
### □ Erreichbarkeitsanalyse (Forts.)



Nr.	$S_1$	$S_2$	$S_3$	$S_4$	aktivierte Transitionen
$M_0$	0	0	0	2	$t_1 \rightarrow M_1$
$M_1$	1	1	0	0	$t_2 \rightarrow M_2$ ← $t_3 \rightarrow M_3$
$M_2$	0	1	1	1	$t_3 \rightarrow M_4$
$M_3$					
$M_4$					

## 2.2 Klassische Petri-Netze

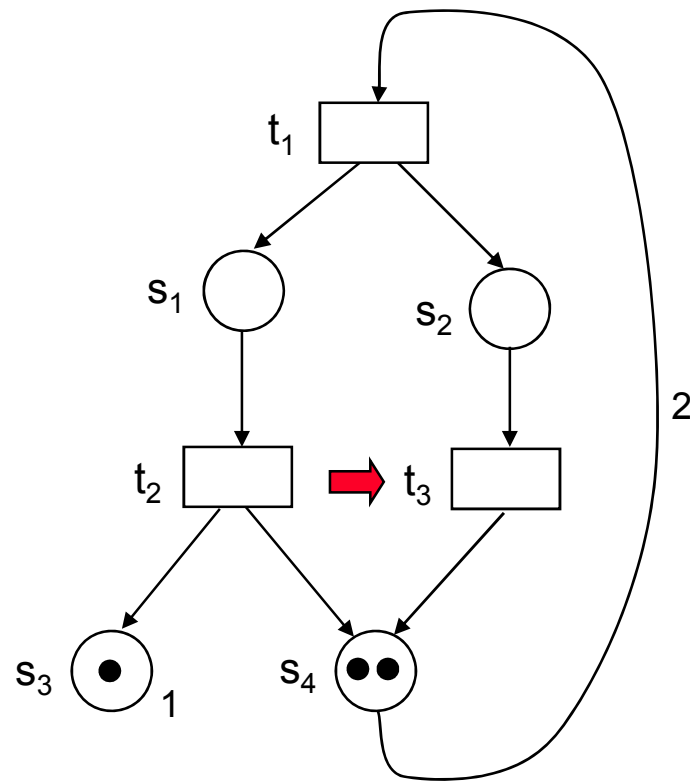
### □ Erreichbarkeitsanalyse (Forts.)



Nr.	$S_1$	$S_2$	$S_3$	$S_4$	aktivierte Transitionen
$M_0$	0	0	0	2	$t_1 \rightarrow M_1$
$M_1$	1	1	0	0	$t_2 \rightarrow M_2$ $t_3 \rightarrow M_3$ ←
$M_2$	0	1	1	1	$t_3 \rightarrow M_4$
$M_3$	1	0	0	1	$t_2 \rightarrow M_5$
$M_4$					
$M_5$					

## 2.2 Klassische Petri-Netze

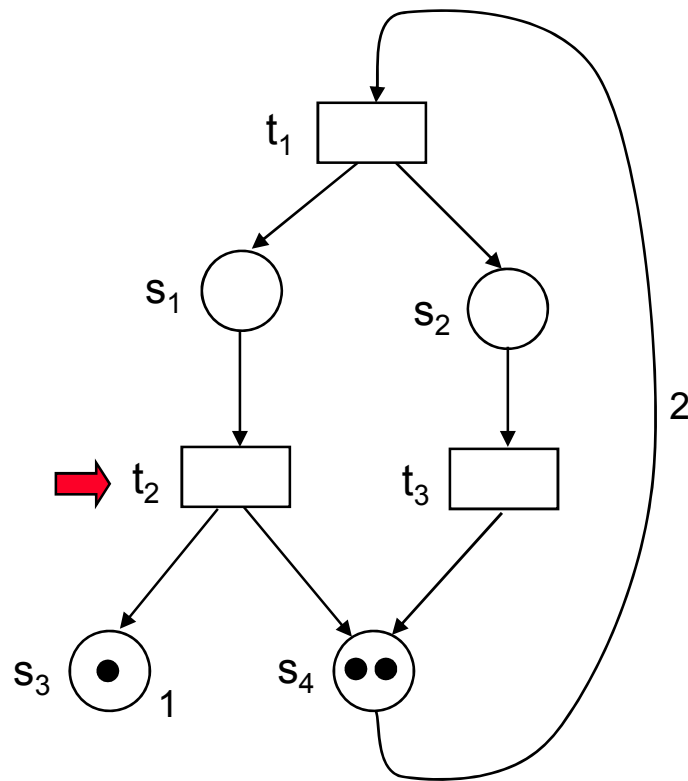
### □ Erreichbarkeitsanalyse (Forts.)



Nr.	$S_1$	$S_2$	$S_3$	$S_4$	aktivierte Transitionen
$M_0$	0	0	0	2	$t_1 \rightarrow M_1$
$M_1$	1	1	0	0	$t_2 \rightarrow M_2$ $t_3 \rightarrow M_3$
$M_2$	0	1	1	1	$t_3 \rightarrow M_4$ ←
$M_3$	1	0	0	1	$t_2 \rightarrow M_5$
$M_4$	0	0	1	2	$t_1 \rightarrow M_6$
$M_5$					
$M_6$					

## 2.2 Klassische Petri-Netze

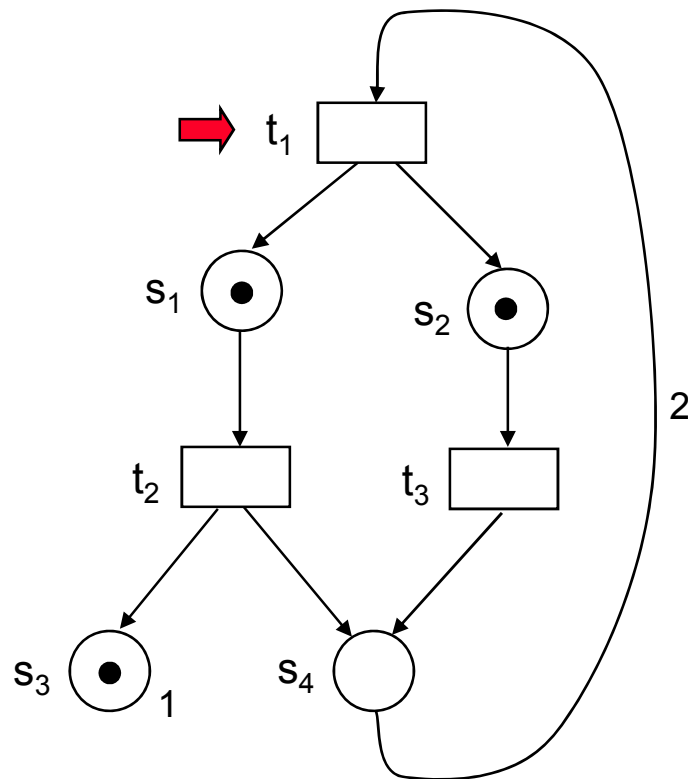
### □ Erreichbarkeitsanalyse (Forts.)



Nr.	$S_1$	$S_2$	$S_3$	$S_4$	aktivierte Transitionen
$M_0$	0	0	0	2	$t_1 \rightarrow M_1$
$M_1$	1	1	0	0	$t_2 \rightarrow M_2$ $t_3 \rightarrow M_3$
$M_2$	0	1	1	1	$t_3 \rightarrow M_4$
$M_3$	1	0	0	1	$t_2 \rightarrow M_5 \leftarrow$
$M_4$	0	0	1	2	$t_1 \rightarrow M_6$
$M_5$	0	0	1	2	$= M_4!$
$M_6$					

## 2.2 Klassische Petri-Netze

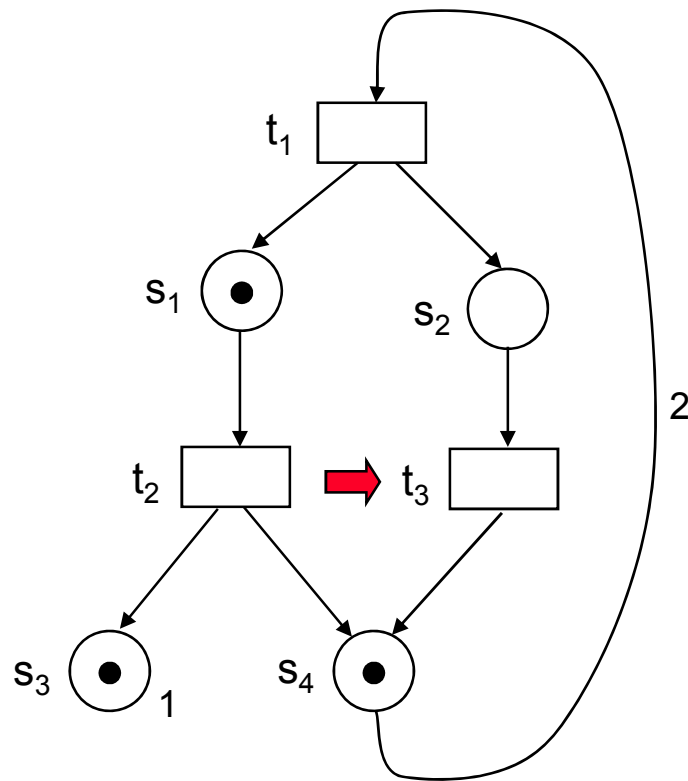
### □ Erreichbarkeitsanalyse (Forts.)



Nr.	$S_1$	$S_2$	$S_3$	$S_4$	aktivierte Transitionen
$M_0$	0	0	0	2	$t_1 \rightarrow M_1$
$M_1$	1	1	0	0	$t_2 \rightarrow M_2$ $t_3 \rightarrow M_3$
$M_2$	0	1	1	1	$t_3 \rightarrow M_4$
$M_3$	1	0	0	1	$t_2 \rightarrow M_5$
$M_4$	0	0	1	2	$t_1 \rightarrow M_6$ ←
$M_5$	0	0	1	2	$= M_4!$
$M_6$	1	1	1	0	$t_3 \rightarrow M_7$
$M_7$					

## 2.2 Klassische Petri-Netze

### □ Erreichbarkeitsanalyse (Forts.)

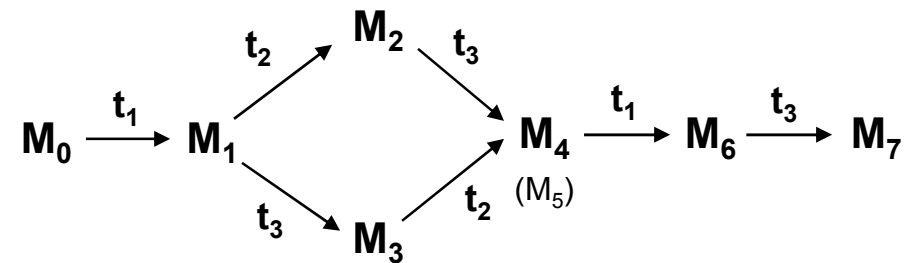


Nr.	$S_1$	$S_2$	$S_3$	$S_4$	aktivierte Transitionen
$M_0$	0	0	0	2	$t_1 \rightarrow M_1$
$M_1$	1	1	0	0	$t_2 \rightarrow M_2$ $t_3 \rightarrow M_3$
$M_2$	0	1	1	1	$t_3 \rightarrow M_4$
$M_3$	1	0	0	1	$t_2 \rightarrow M_5$
$M_4$	0	0	1	2	$t_1 \rightarrow M_6$
$M_5$	0	0	1	2	$= M_4!$
$M_6$	1	1	1	0	$t_3 \rightarrow M_7$ ←
$M_7$	1	0	1	1	--- (Endzustand)

## 2.2 Klassische Petri-Netze

### □ Erreichbarkeitsanalyse (Forts.)

Nr.	S <sub>1</sub>	S <sub>2</sub>	S <sub>3</sub>	S <sub>4</sub>	Transitionen
M <sub>0</sub>	0	0	0	2	t <sub>1</sub> → M <sub>1</sub>
M <sub>1</sub>	1	1	0	0	t <sub>2</sub> → M <sub>2</sub> t <sub>3</sub> → M <sub>3</sub>
M <sub>2</sub>	0	1	1	1	t <sub>3</sub> → M <sub>4</sub>
M <sub>3</sub>	1	0	0	1	t <sub>2</sub> → M <sub>5</sub>
M <sub>4</sub>	0	0	1	2	t <sub>1</sub> → M <sub>6</sub>
M <sub>5</sub>	0	0	1	2	= M <sub>4</sub> !
M <sub>6</sub>	1	1	1	0	t <sub>3</sub> → M <sub>7</sub>
M <sub>7</sub>	1	0	1	1	---



Erreichbarkeitsgraph

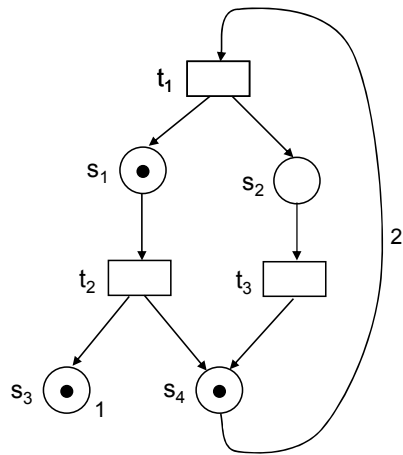
## 2.2 Klassische Petri-Netze

---

### □ Algorithmus zur Bestimmung der Erreichbarkeitsmenge

```
Erreicht1 := { M0 }; Durchlaufen1 := Erledigt1 := ∅; i := 1; /* Initialisierungen */
while Erreichti ≠ Erledigti do
  begin
    wähle ein M ∈ Erreichti \ Erledigti;
    if ∃ □ t, M' mit M[t⟩ M' und (M, t, M') ∉ Durchlaufeni then
      begin
        Durchlaufeni+1 := Durchlaufeni ∪ { (M, t, M') };
        Erreichti+1 := Erreichti ∪ { M' };
        Erledigti+1 := Erledigti
      end
    else
      begin
        Erreichti+1 := Erreichti;
        Durchlaufeni+1 := Durchlaufeni;
        Erledigti+1 := Erledigti ∪ { M }
      end
    i := i + 1;
  end;
```





## Beispiel

i	gewähltes M	Durchlaufen	Erreicht	Erledigt
	(Initialisierung)	$\emptyset$	$(0, 0, 0, 2)$	$\emptyset$
1	$(0, 0, 0, 2)$	$(0, 0, 0, 2) \cup t_1 (1, 1, 0, 0)$	$(1, 1, 0, 0)$	—
2	$(0, 0, 0, 2)$	—	—	$(0, 0, 0, 2)$
3	$(1, 1, 0, 0)$	$(1, 1, 0, 0) \cup t_2 (0, 1, 1, 1)$	$(0, 1, 1, 1)$	—
4	$(1, 1, 0, 0)$	$(1, 1, 0, 0) \cup t_3 (1, 0, 0, 1)$	$(1, 0, 0, 1)$	—
5	$(1, 1, 0, 0)$	—	—	$(1, 1, 0, 0)$
6	$(0, 1, 1, 1)$	$(0, 1, 1, 1) \cup t_3 (0, 0, 1, 2)$	$(0, 0, 1, 2)$	—
7	$(0, 1, 1, 1)$	—	—	$(0, 1, 1, 1)$
8	$(1, 0, 0, 1)$	$(1, 0, 0, 1) \cup t_2 (0, 0, 1, 2)$	—	—
9	$(1, 0, 0, 1)$	—	—	$(1, 0, 0, 1)$
10	$(0, 0, 1, 2)$	$(0, 0, 1, 2) \cup t_1 (1, 1, 1, 0)$	$(1, 1, 1, 0)$	—
11	$(0, 0, 1, 2)$	—	—	$(0, 0, 1, 2)$
12	$(1, 1, 1, 0)$	$(1, 1, 1, 0) \cup t_3 (1, 0, 1, 1)$	$(1, 0, 1, 1)$	—
13	$(1, 1, 1, 0)$	—	—	$(1, 1, 1, 0)$
14	$(1, 0, 1, 1)$	—	—	$(1, 0, 1, 1)$
Erreicht <sub>14</sub> = Erledigt <sub>14</sub> ↓ fertig!				

## 2.2 Klassische Petri-Netze

---

□ **Definition 2-9:** aktivierbar, deadlockfrei, schwach lebendig, (stark) lebendig, tot

Eine Transition  $t$  eines S/T-Systems  $Y = (\vec{N}, M_0)$  heißt **aktivierbar**, wenn sie unter mindestens einer Folgemarkierung aktivierbar ist:

$$\exists M_1 \in [M_0] : M_1[t].$$

Ein S/T-System  $Y = (S, T, F, K, W, M_0)$  heißt **deadlockfrei** oder **schwach lebendig**, wenn unter jeder erreichbaren Markierung mindestens eine Transition aktiviert ist:

$$\forall M_1 \in [M_0] : \exists t \in T : M_1[t].$$

Es heißt **lebendig** oder **stark lebendig**, wenn aus jeder erreichbaren Markierung jede Transition aktivierbar ist:

$$\forall M_1 \in [M_0] : \forall t \in T : \exists M_2 \in [M_1] : M_2[t].$$

Es heißt **tot**, wenn keine Transition aktiviert ist:

$$\forall t \in T : \neg M_0[t].$$

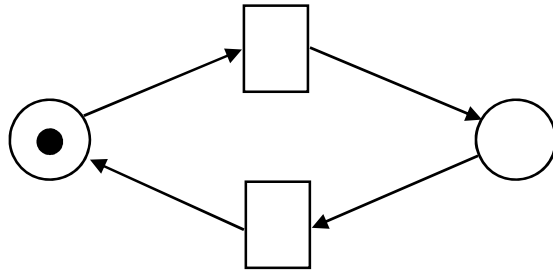
Hinweis:

Man beachte, dass *tot* in der obigen Definition nicht das Gegenteil von *lebendig* ist.

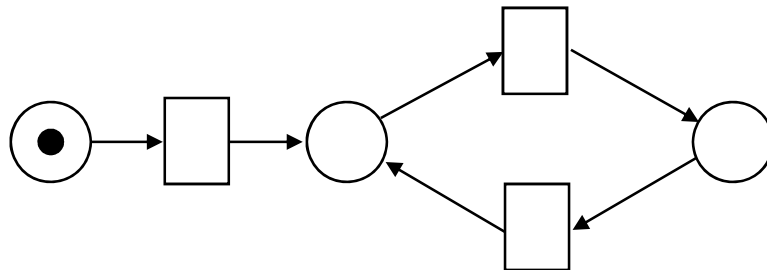
## 2.2 Klassische Petri-Netze

---

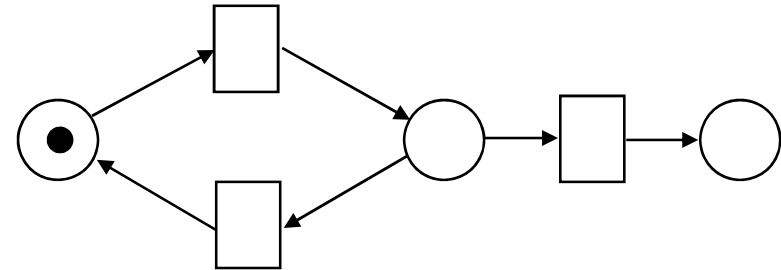
- Beispiel für ein stark lebendiges S/T-Netz:



- Haben diese S/T-Netze die Eigenschaft schwach lebendig zu sein?



a) 



b) 

# Inhalt

---

## 2.0 Vorbemerkungen

## 2.1 Korrektheitsaspekte von Prozessmodellen – eine Einführung

## 2.2 Klassische Petri-Netze

## 2.3 Höhere Petri-Netze

## 2.4 Workflow-Netze

## 2.5 Aktivitätennetze

## 2.6 AristaFlow-Prozessmodell

## 2.7 Andere Ansätze

## 2.8 Ausdrucksmächtigkeit von Prozessbeschreibungssprachen

## 2.9 Abschließende Bemerkungen

## 2.10 Weiterführende Literatur

## 2.3 Höhere Petri-Netze

---

2.3.1 Motivation

2.3.2 Gefärbte Petri-Netze (coloured Petri Nets)

2.3.4 Prädikaten-/Transitionsnetze

## 2.3 Höhere Petri-Netze

---

### 2.3.1 Motivation

#### □ Bisher

- Nur Betrachtung von Netzen mit anonymen (d.h. nicht unterscheidbaren) Marken
- Für Modellierung komplexer Systeme und Abläufe nicht ausreichend

#### □ Höhere Petri-Netze

- Den Marken werden Werte zugeordnet, so dass sie sich unterscheiden und man Prädikate für das Schaltverhalten von Transitionen definieren kann
- Dadurch komplexere Schaltregeln realisierbar

#### □ Viele Netzvarianten

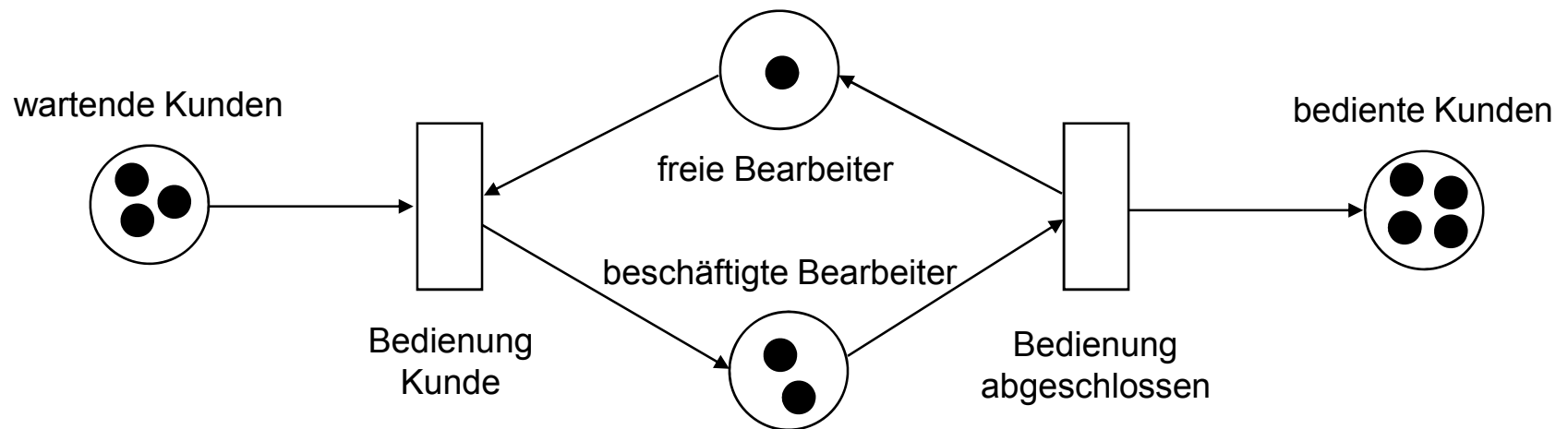
- Prädikat-Transitionsnetze
- Produktnetze
- Coloured Nets
- Relationennetze
- Zeitbehaftete Netze, etc

#### □ Im Folgenden: Skizzierung der Grundidee höherer Petri-Netze anhand von Beispielen

## 2.3 Höhere Petri-Netze

---

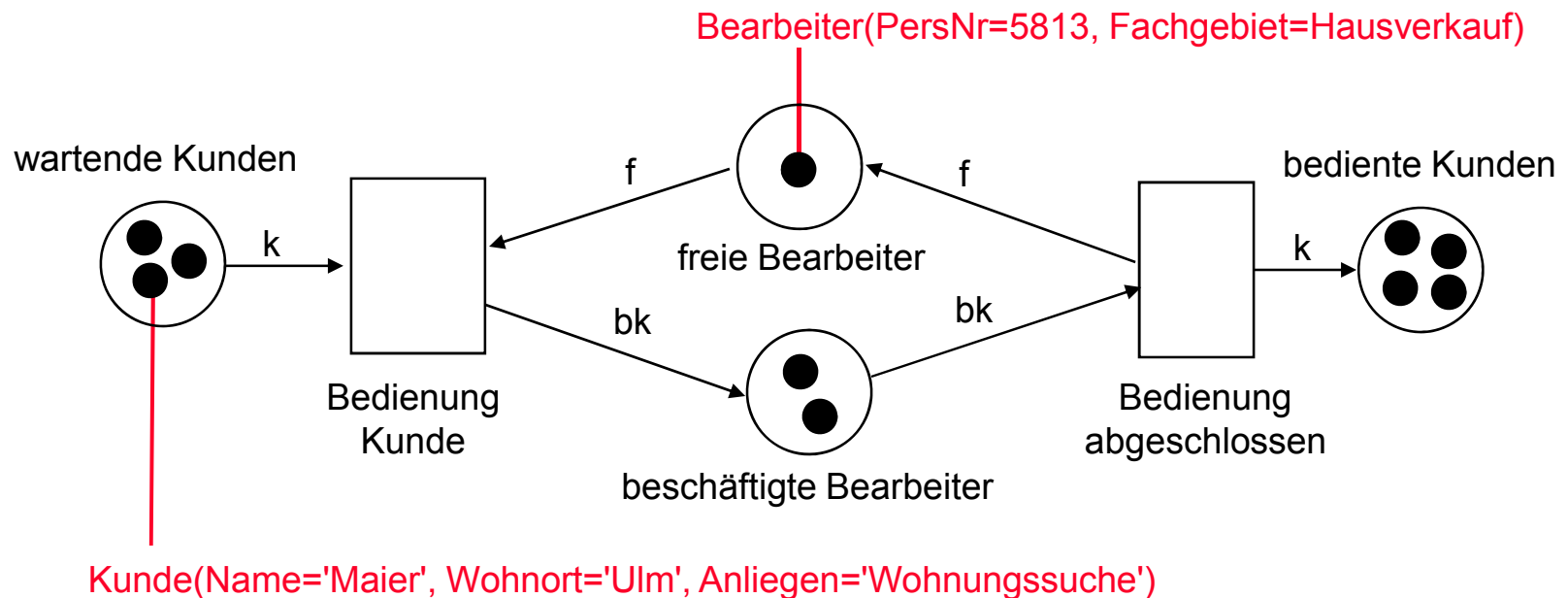
- Klassische Petri-Netze: **Nicht unterscheidbare Marken**



## 2.3 Höhere Petri-Netze

### 2.3.2 Gefärbte Petri-Netze (Coloured Petri Nets)

- Erweiterung um „Farben“ (Attribute/Werte)
- Jede Marke besitzt einen Wert und ist deshalb von anderen Marken unterscheidbar
- Außerdem: Typisierte Marken (= Records/Tupel mit Attributnamen und -Typen)

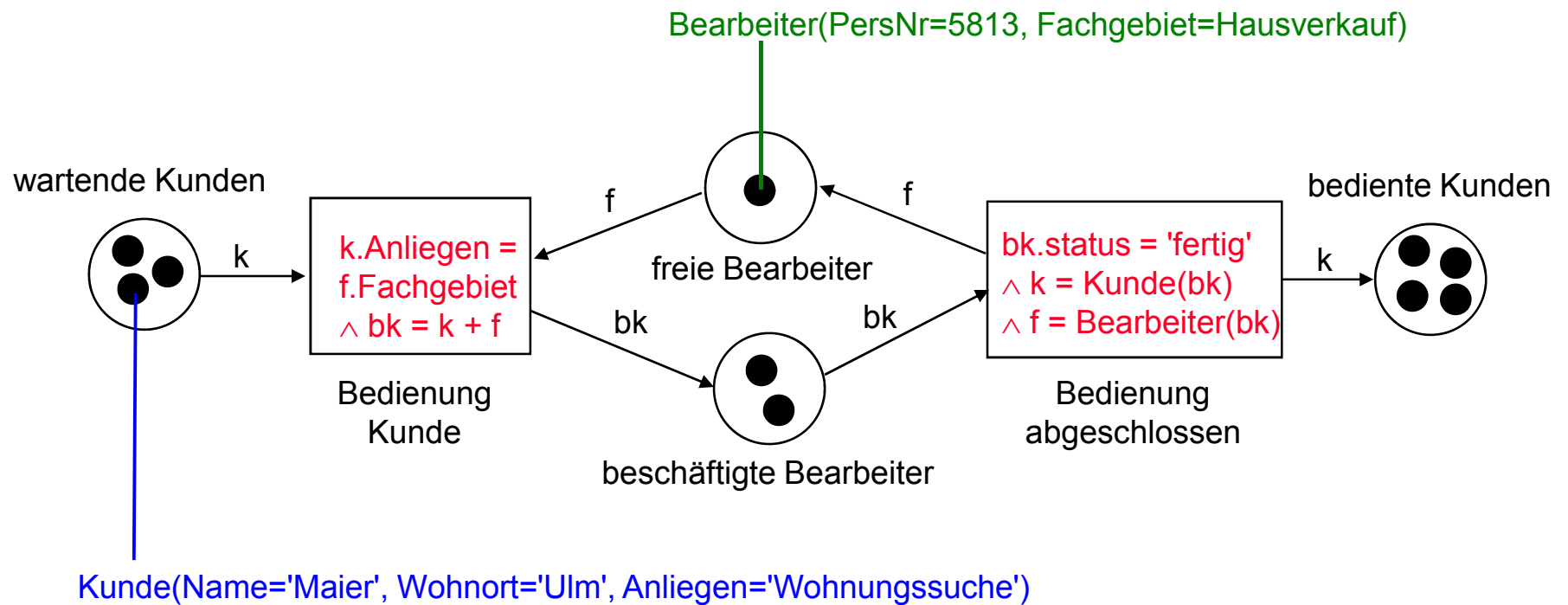




## 2.3 Höhere Petri-Netze

### 2.3.4 Prädikaten-/Transitionsnetze

- Weitere Erweiterung:  
Bedingungen bzgl. der Werte von zu konsumierenden oder zu produzierenden Marken



# Inhalt

---

## 2.0 Vorbemerkungen

## 2.1 Korrektheitsaspekte von Prozessmodellen – eine Einführung

## 2.2 Klassische Petri-Netze

## 2.3 Höhere Petri-Netze

## 2.4 Workflow-Netze

## 2.5 Aktivitätennetze

## 2.6 AristaFlow-Prozessmodell

## 2.7 Andere Ansätze

## 2.8 Ausdrucksmächtigkeit von Prozessbeschreibungssprachen

## 2.9 Abschließende Bemerkungen

## 2.10 Weiterführende Literatur

## 2.4 Workflow-Netze

---

### □ Warum Workflow-Netze (WF-Netze)?

- (Höhere) Petri-Netze im Prinzip geeignet, um (Geschäfts-)Prozesse zu modellieren
- Unterscheidbare Marken erlauben verschiedene „Prozessinstanzen“ im selben Netz zu repräsentieren
- Petri-Netze bieten vielfältige Analysemöglichkeiten
- Aber auch: Sehr viele Freiheitsgrade für die Modellierung, damit verbunden
  - ein hoher Analyseaufwand
  - viele Modellierungsfehler nicht durch Analysen erkennbar, da strukturell zulässig

## 2.4 Workflow-Netze

---

### □ Workflow-Netze (WF-Netze) ♣

- Erweiterung der klassischen Petri-Netze um zusätzliche Konzepte und Notationen
- Strukturelle Einschränkungen, um die Analysierbarkeit und Verständlichkeit zu erhöhen
- Wesentliche Eigenschaften:
  - WF-Netze haben stets genau einen Start- und genau einen Endknoten
  - Alle Stellen und Transitionen liegen auf einem Pfad vom Start- und Endknoten
  - Basieren auf gefärbten Petri-Netzen, d.h. die Marken sind unterscheidbar; sie repräsentieren Prozessinstanzen und enthalten Anwendungsdaten sowie den Prozess-Instanz-Identifizier
  - WF-Netze können hierarchisch strukturiert werden; d.h. eine Transition kann ein einfacher Prozessschritt sein oder ein komplexer Prozessschritt, der durch ein anderes, dediziertes WF-Netz realisiert wird.

♣ Zum Teil angelehnt an M. Weske: Business Process Management – Concepts, Languages, Architectures. Springer-Verlag, 2007

## 2.4 Workflow-Netze

---

### □ Definition 2-10: Workflow-Netz

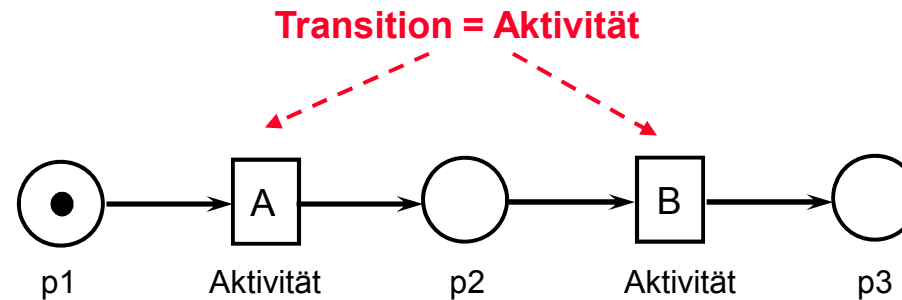
Ein Petri-Netz  $PN = (P, T, F)$  ist ein **Workflow-Netz** genau dann, wenn die folgenden Bedingungen gelten:

- Es gibt genau eine Stelle  $i \in P$  (*Startknoten* genannt), die keine einmündende Kante hat, d.h.  $\bullet i = \emptyset$
- Es gibt genau eine Stelle  $o \in P$  (*Endknoten* genannt), die keine ausgehende Kante hat, d.h.  $o \bullet = \emptyset$
- Jede Stelle und jede Transition liegt auf einem Pfad vom Start- zum Endknoten

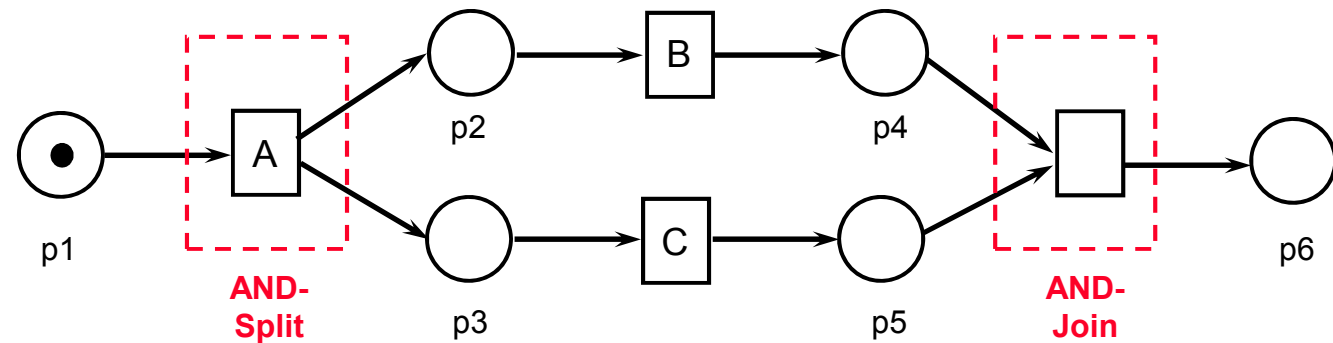
## 2.4 Workflow-Netze

### □ Basis-Kontrollflussmodellierung

#### ○ Sequenz



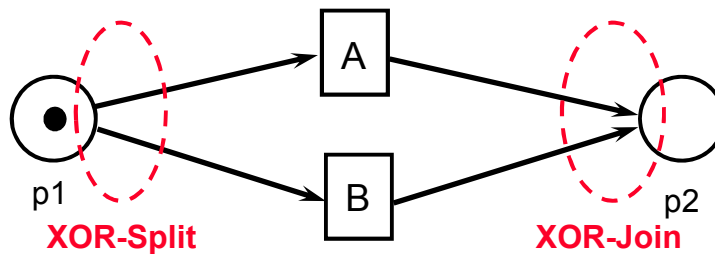
#### ○ Parallelität



## 2.4 Workflow-Netze

### □ Basis-Kontrollflussmodellierung (Forts.)

#### ○ Alternative Pfade



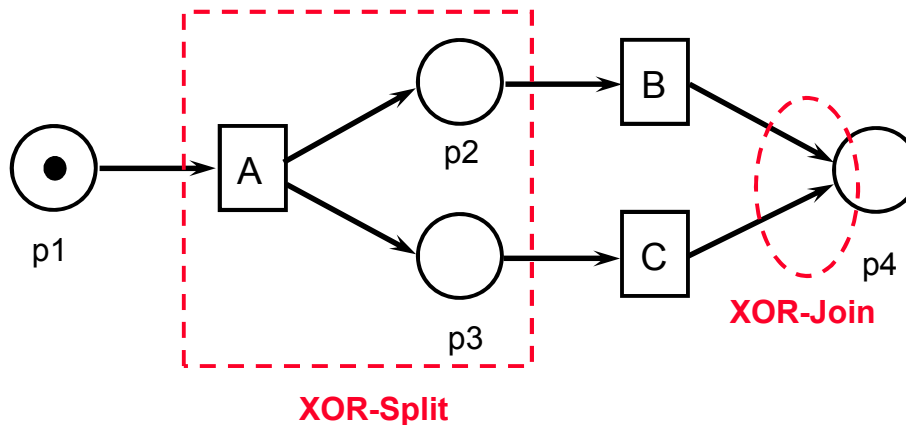
#### Impliziter XOR-Split

Aktivitäten A und B werden beide aktiviert (aber nicht gestartet!).

Wenn dann eine davon gestartet wird, wird die andere „abgewählt“.

↓ „deferred choice“

Beispiel (siehe später)



#### Expliziter XOR-Split

Die in A hinterlegte Entscheidungsregel entscheidet, ob die Marke entweder in p2 oder p3 gelegt wird.

#### Achtung:

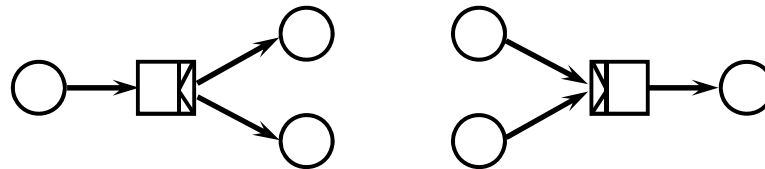
**Dies entspricht nicht mehr dem Schaltverhalten von Transitionen in Standard-Petri-Netzen!**

## 2.4 Workflow-Netze

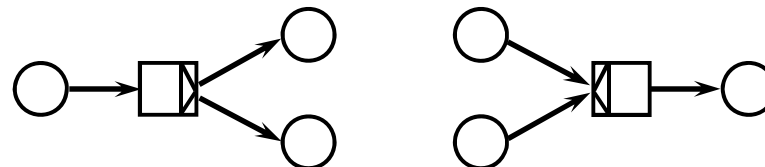
### □ Erweiterte Kontrollflussmodellierung

- Einführung **verschiedener Symbole für Transitionen**, um **unterschiedliches Schaltverhalten** explizit beschreiben zu können

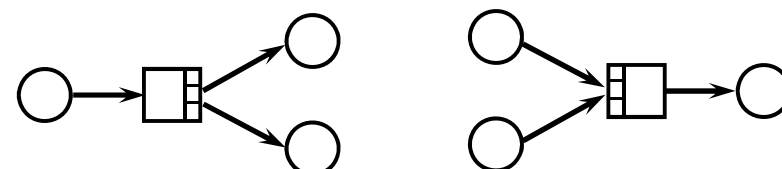
#### ■ AND-Split und AND-Join



#### ■ XOR-Split und XOR-Join



#### ■ Entscheidungsregelbasierter Split (XOR, OR, AND) + entsprechender Join



Diese ergänzenden Symbole sind nur ein zusätzliches „Angebot“.

Der Workflow-Netz-Formalismus schreibt nicht vor, wie diese im Zusammenspiel zu verwenden sind, etwa dass ein expliziter AND-Split mit einem expliziten AND-Join abgeschlossen werden muss.

Es liegt in der Verantwortung des Prozessmodellierers, diese Symbole korrekt anzuwenden.



## 2.4 Workflow-Netze

---

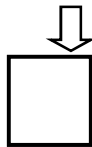
### □ Erweiterte Kontrollflussmodellierung (Forts.)

- Einführung **ergänzender „Trigger“-Symbole für Transitionen**, um zwischen verschiedenen Aktivierungsarten unterscheiden zu können



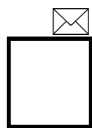
#### Automatischer Trigger

↓ Aktivität startet automatisch



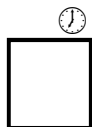
#### Benutzer-Trigger

↓ Aktivität wird von einem Benutzer gestartet



#### Externer Trigger

↓ Aktivität wartet auf ein externes Ereignis um zu starten



#### Zeit-Trigger

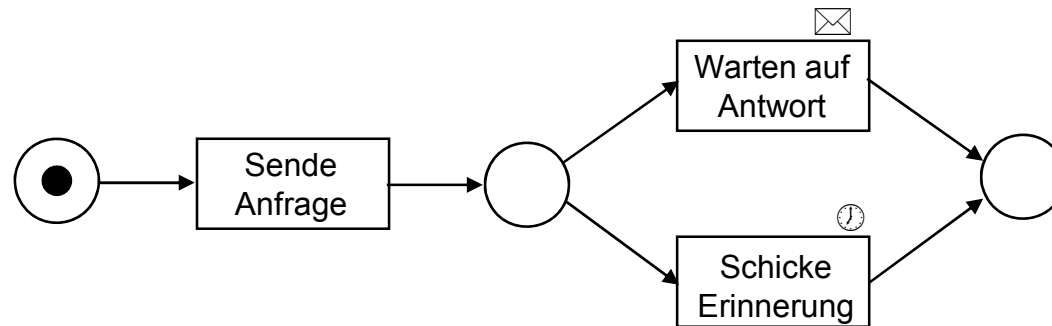
↓ Aktivität wird vom PMS gestartet, wenn die Wartezeit abgelaufen ist

## 2.4 Workflow-Netze

---

### □ Modellierungs-Beispiele

- Eine Anfrage soll versandt werden. Geht die Antwort nicht innerhalb einer vorgegebenen Zeit ein, soll eine Erinnerung versandt werden.



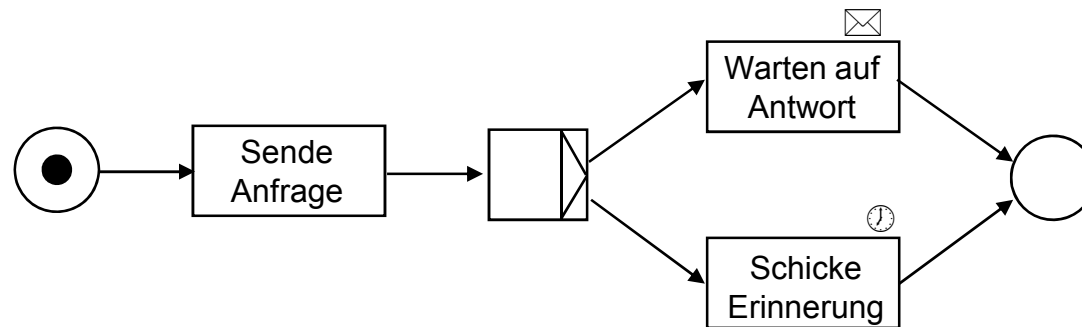
Quizfrage: Was geschieht, wenn die Antwort vor Ablauf des Timers eintrifft?  
Wird die Erinnerung trotzdem verschickt?

## 2.4 Workflow-Netze

---

### □ Modellierungs-Beispiele (Forts.)

- Wie vorher, aber nun mit explizitem XOR modelliert



Quizfrage: Ist diese Lösung äquivalent zur vorherigen und führt zum selben Ergebnis?

## 2.4 Workflow-Netze

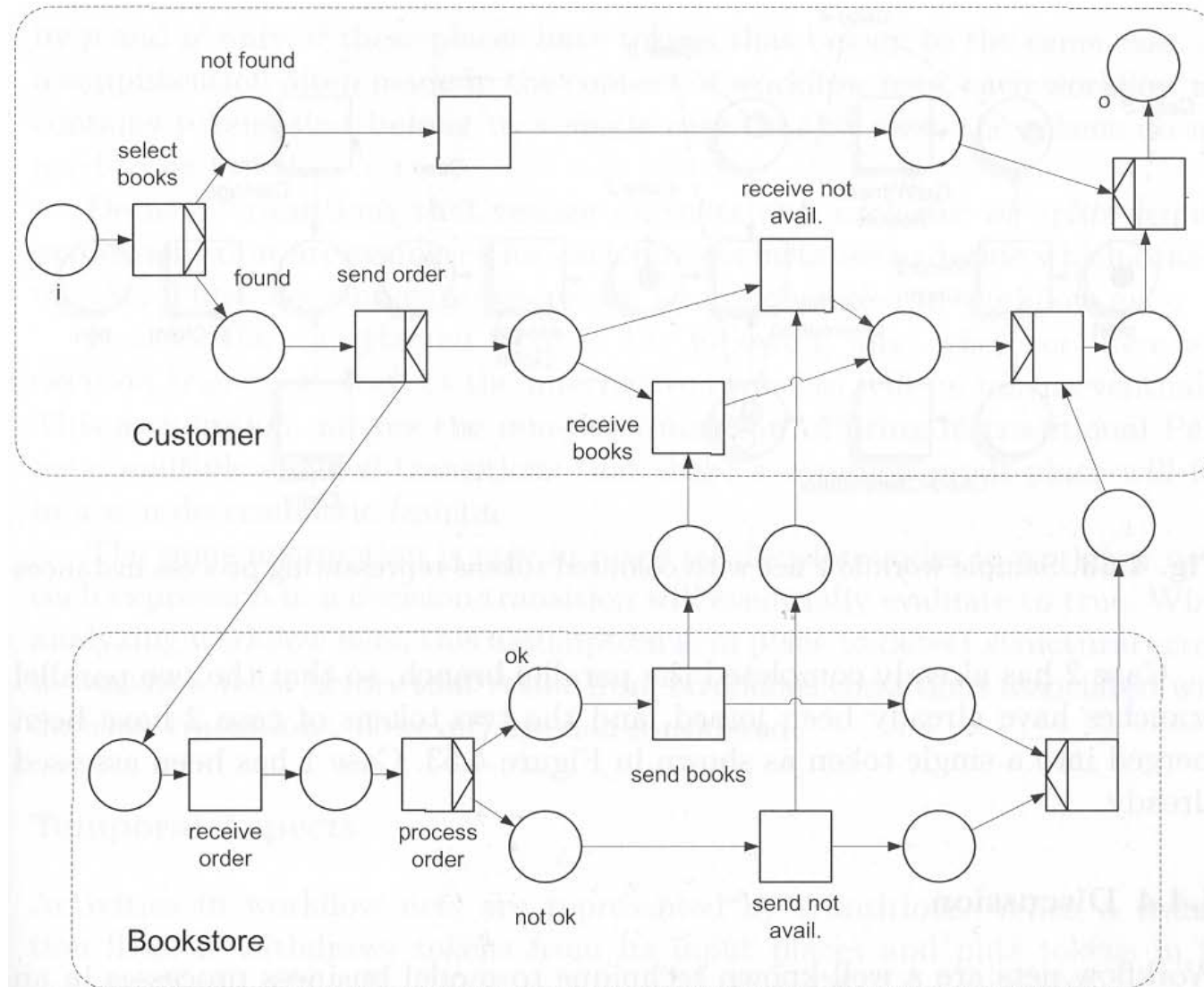
---

### □ Modellierungs-Beispiele (Forts.)

#### ○ Ein (stark vereinfachtes) Buchhandelsbeispiel, gleichzeitig Beispiel für Prozess, der mehrere Parteien involviert

- Ein Kunde durchsucht mit Hilfe eines Abfragesystems den Online-Katalog des Buchhändlers und wählt ggf. daraus Bücher aus.
- Wurden Bücher ausgewählt, erstellt das Abfragesystem einen Versandauftrag an den Buchhändler. Der Kunde wartet derweilen.
- Nach Eingang des Versandauftrags beim Buchhändler wird der Auftrag bearbeitet.
- Sind alle gewünschten Bücher vorhanden, werden diese versandt, das Abfragesystem entsprechend benachrichtigt und der Vorgang damit abgeschlossen.
- Sind die gewünschten Bücher nicht vorhanden, so wird das Abfragesystem informiert, dass eine Lieferung nicht möglich ist und der Vorgang abgeschlossen.

## 2.4 Workflow-Netze



## 2.4 Workflow-Netze

---

### □ Definition 2-11: Soundness eines Workflow-Netzes

Ein Workflow-Netz heißt **sound**, wenn die folgenden Eigenschaften erfüllt sind:

Sei  $m_i$  die Eingabemarkierung, d.h. lediglich die Eingabestelle  $i$  ist markiert. Sei  $m_o$  die Endemarkierung, d.h. lediglich die Ausgabestelle  $o$  ist markiert.

- Von jeder aus  $m_i$  erreichbaren Markierung  $m$  ist  $m_o$  erreichbar.
- $m_o$  ist die einzige von  $m_i$  aus erreichbare Markierung, in der die Ausgabestelle  $o$  markiert ist.
- Das Workflow-Netz enthält keine toten Transitionen.

## 2.4 Workflow-Netze

---

### □ Bewertung

#### ○ Allgemein

- Petri-Netze sehr wichtige Basis-Theorie für die Beschreibung und Analyse nebenläufiger Systeme
- Bis heute sehr lebendiges Forschungsgebiet mit breitem Spektrum an Arbeiten
- Sehr viele Weiterentwicklungen für verschiedene Fragestellungen

#### ○ Positiv

- Sehr gute Theoriebasis
- Vielfältige Analysemöglichkeiten (wir haben hier nur einen kleinen Ausschnitt behandelt)
- Wenig Beschränkungen: Modellierung (fast) beliebig strukturierter Prozesse möglich

#### ○ Negativ

- Kontrollfluss = Datenfluss (die Marken tragen die Information)
- Hohe Freiheitsgrade in der Modellierung führen leicht zu schwer verständlichen Modellen
- Auch sind Analysen (z.B. Erreichbarkeit, Deadlockfreiheit, ...) i.d.R. sehr aufwendig
- Nur sehr eingeschränkt tauglich für Systeme, die Ad-hoc-Abweichungen in Verbindung mit Korrektheit unterstützen wollen

# Inhalt

---

## 2.0 Vorbemerkungen

## 2.1 Korrektheitsaspekte von Prozessmodellen – eine Einführung

## 2.2 Klassische Petri-Netze

## 2.3 Höhere Petri-Netze

## 2.4 Workflow-Netze

## 2.5 Aktivitätennetze

## 2.6 AristaFlow-Prozessmodell

## 2.7 Andere Ansätze

## 2.8 Ausdrucksmächtigkeit von Prozessbeschreibungssprachen

## 2.9 Abschließende Bemerkungen

## 2.10 Weiterführende Literatur



## 2.5 Aktivitätennetze

---

### □ Merkmale

- Getrennte Modellierung von Kontroll- und Datenfluss
- Beschreibungselemente zur Festlegung des Kontrollflusses:
  - Kontrollkonnektoren
  - Transitionsbedingungen für Kontrollkonnektoren
  - Start-/ Endbedingungen für Aktivitäten
- Beschreibungselemente zur Festlegung des Datenflusses
  - Datenkonnektoren
  - Abbildung (Mapping) von Ausgabeparametern einer Aktivität auf Eingabeparameter von im Kontrollfluss nachfolgenden Aktivitäten (via Datenfeldern in Datencontainern)
- Konzepte zur hierarchischen Strukturierung von Aktivitätennetzen (im Folgenden nicht weiter betrachtet)
- Formal definierte Ausführungssemantik
- Verwendet u.a. in IBM WebSphere MQSeries Workflow® (ehemals: IBM FlowMark®)
- **Im Folgenden Beschränkung auf Kontroll- und Datenflussaspekte**

## 2.5 Aktivitätennetze

---

### □ Definition 2-12: Aktivitätennetz

Ein **Aktivitätennetz** ist ein 3-Tupel  $AN = (A, K, D)$  wobei

$A = \{ A_1, A_2, \dots, A_n \}$  die Menge der **Aktivitäten(knoten)**

$K \subseteq A \times A$  den **Kontrollfluss**

$D \subseteq A \times A$  den **Datenfluss**

repräsentiert und außerdem gilt, dass

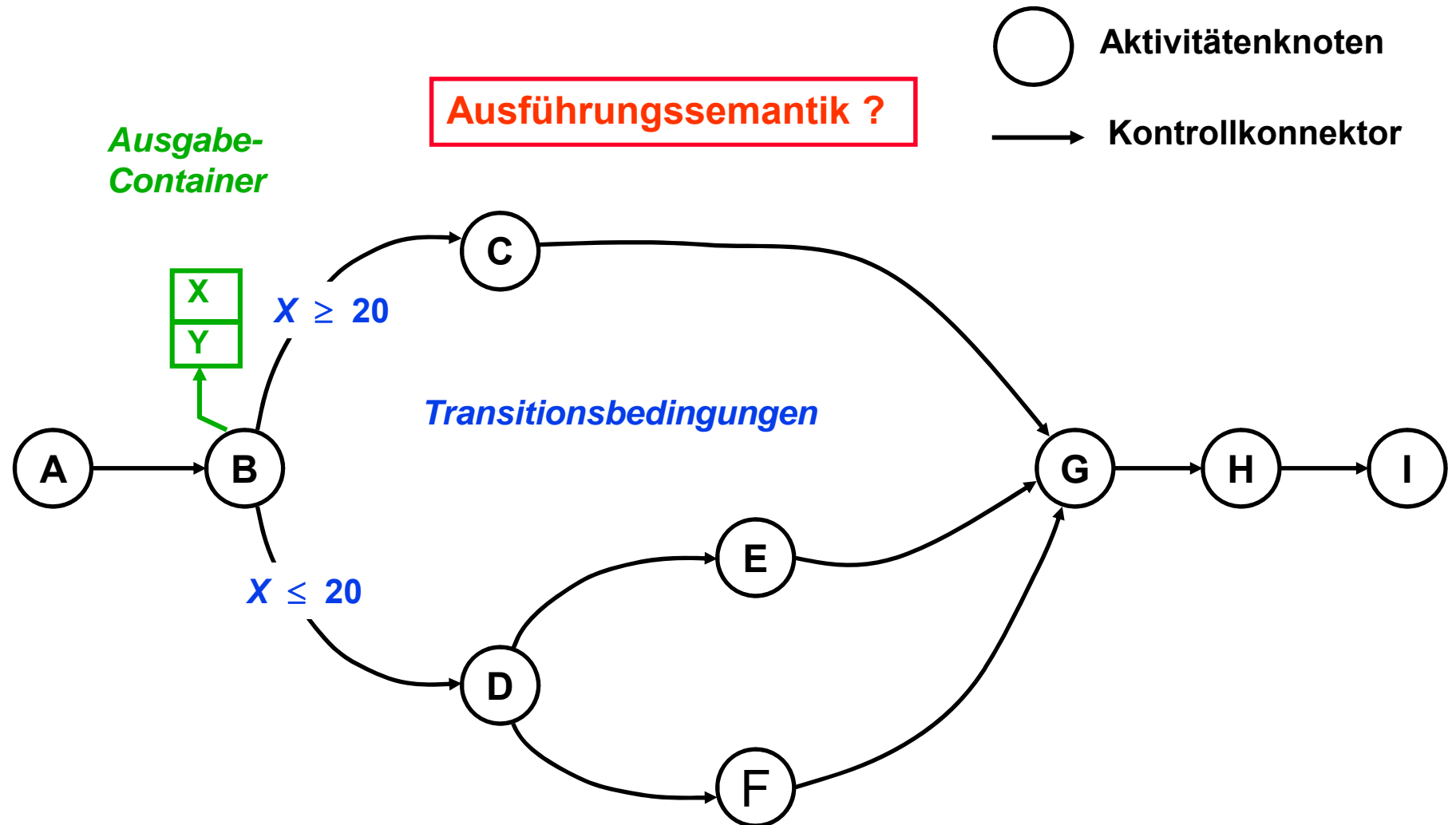
- $K \cup D$  zyklensfrei ist
- einer **Kontrollflusskante** eine **Transitionsbedingung** zugewiesen werden kann
- einem **Aktivitätenknoten** eine **Join-Bedingung** zugeordnet werden kann

### □ Anmerkungen und Erläuterungen:

- Aus der Zyklensfreiheit folgt, dass es in einem AN stets mind. eine Start- und End-Aktivität gibt.
- Durch die Zuordnung einer Transitionsbedingung  $t \in T$  zu einer Kontrollflusskante lassen sich bedingte Ausführungen modellieren.
- Eine **Join-Bedingung** legt fest, ob nur eine oder alle eingehenden Kontrollkonnektoren mit TRUE bewertet sein müssen, damit die Aktivität ausführbar ist.
- Der konkrete Datenfluss wird über Ein- und Ausgabe-Datencontainer realisiert (siehe Beispiel).

## 2.5 Aktivitätennetze

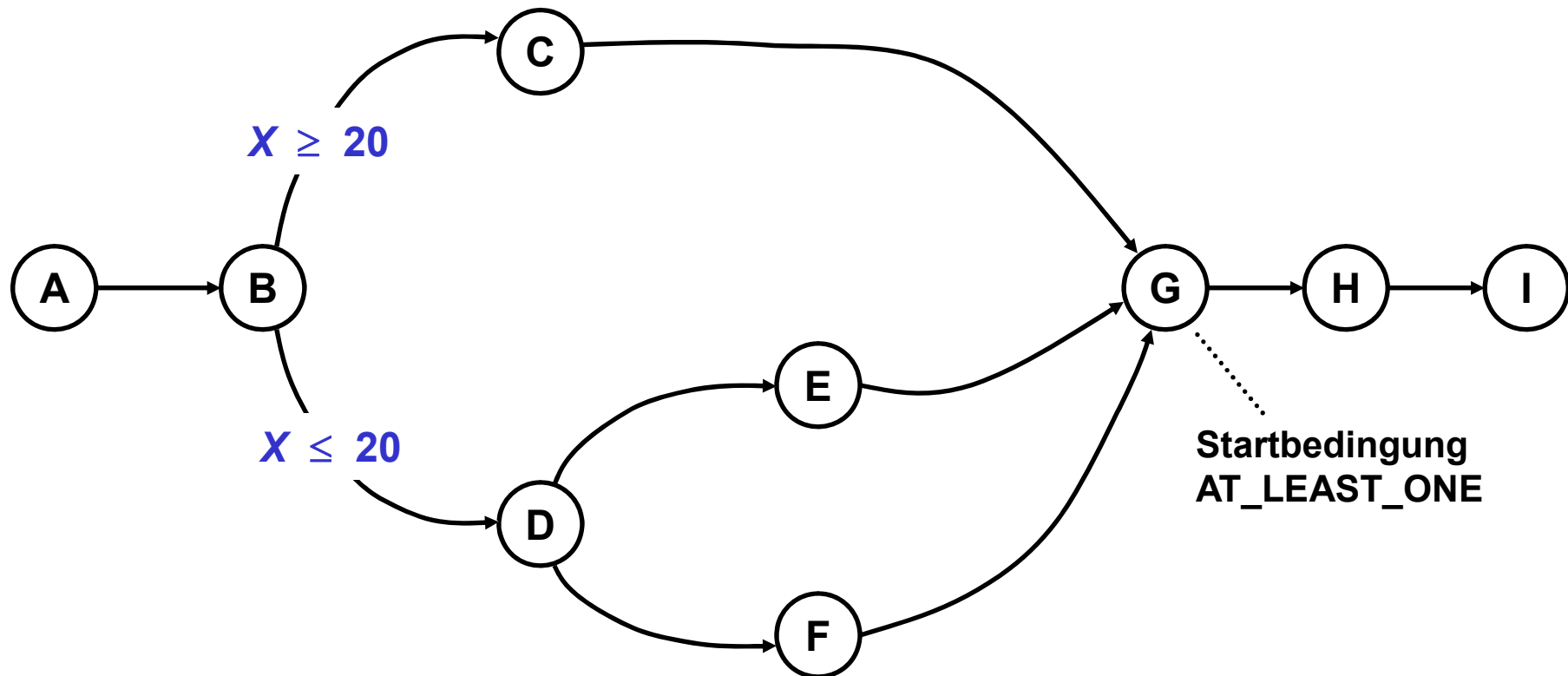
### □ Kontrollflussmodellierung



## 2.5 Aktivitätennetze

### □ Kontrollflussmodellierung (Forts.)

**Beispiel 1**



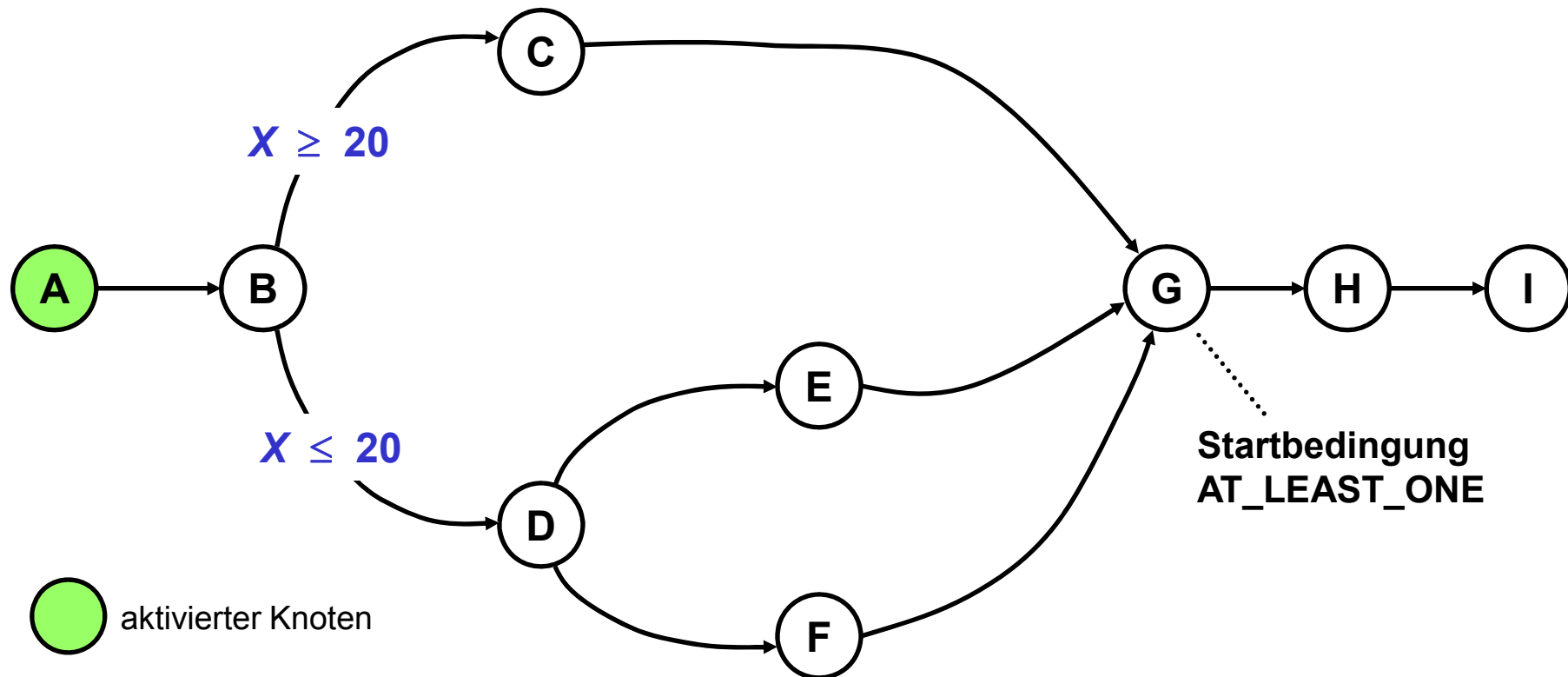
Startbedingung  
AT\_LEAST\_ONE

**Ausführung startet  
beim Knoten A**

## 2.5 Aktivitätennetze

### □ Kontrollflussmodellierung (Forts.)

**Beispiel 1**

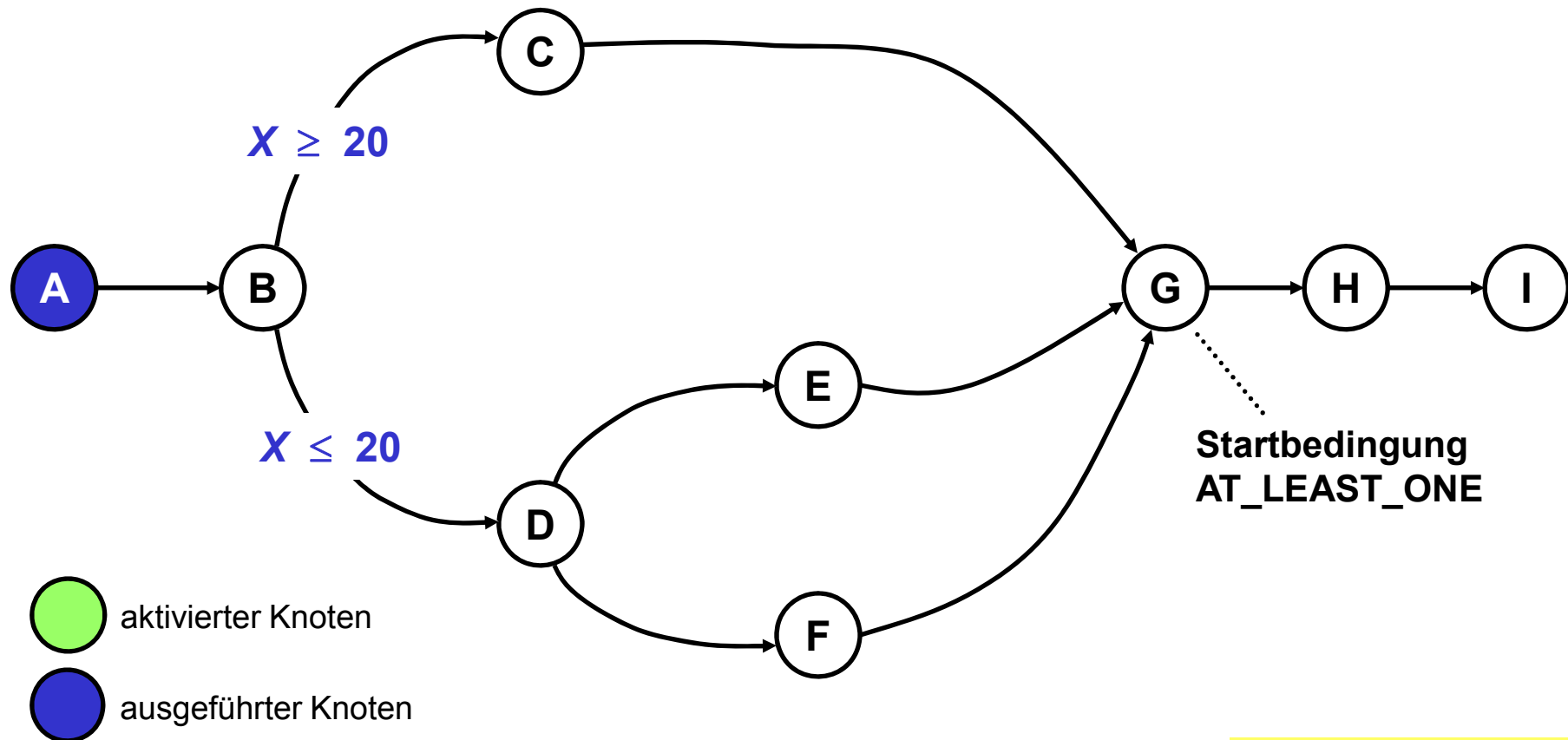


**Ausführung startet  
beim Knoten A**

## 2.5 Aktivitätensnetze

### □ Kontrollflussmodellierung (Forts.)

**Beispiel 1**

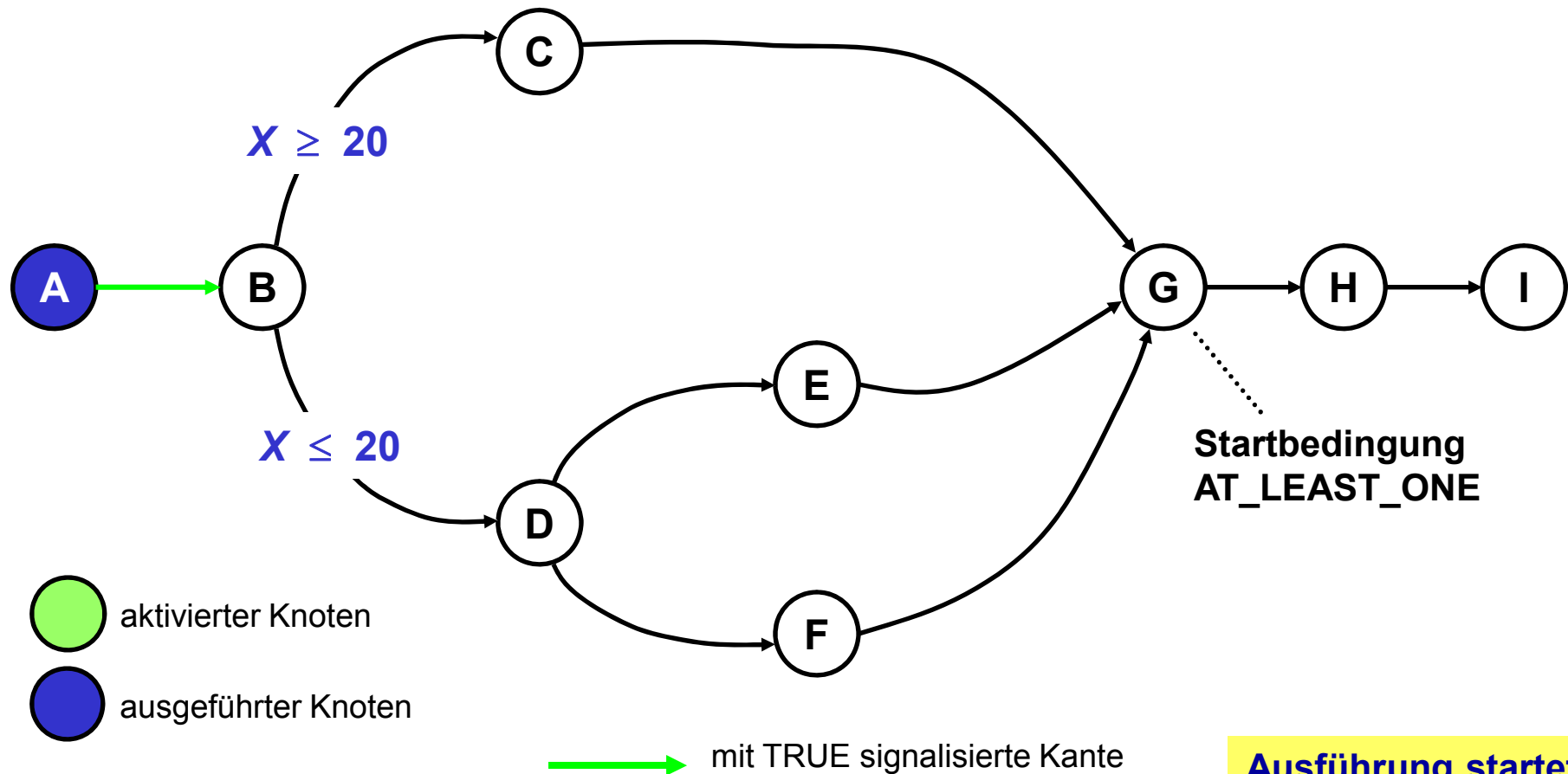


**Ausführung startet  
beim Knoten A**

## 2.5 Aktivitätensetze

### □ Kontrollflussmodellierung (Forts.)

**Beispiel 1**

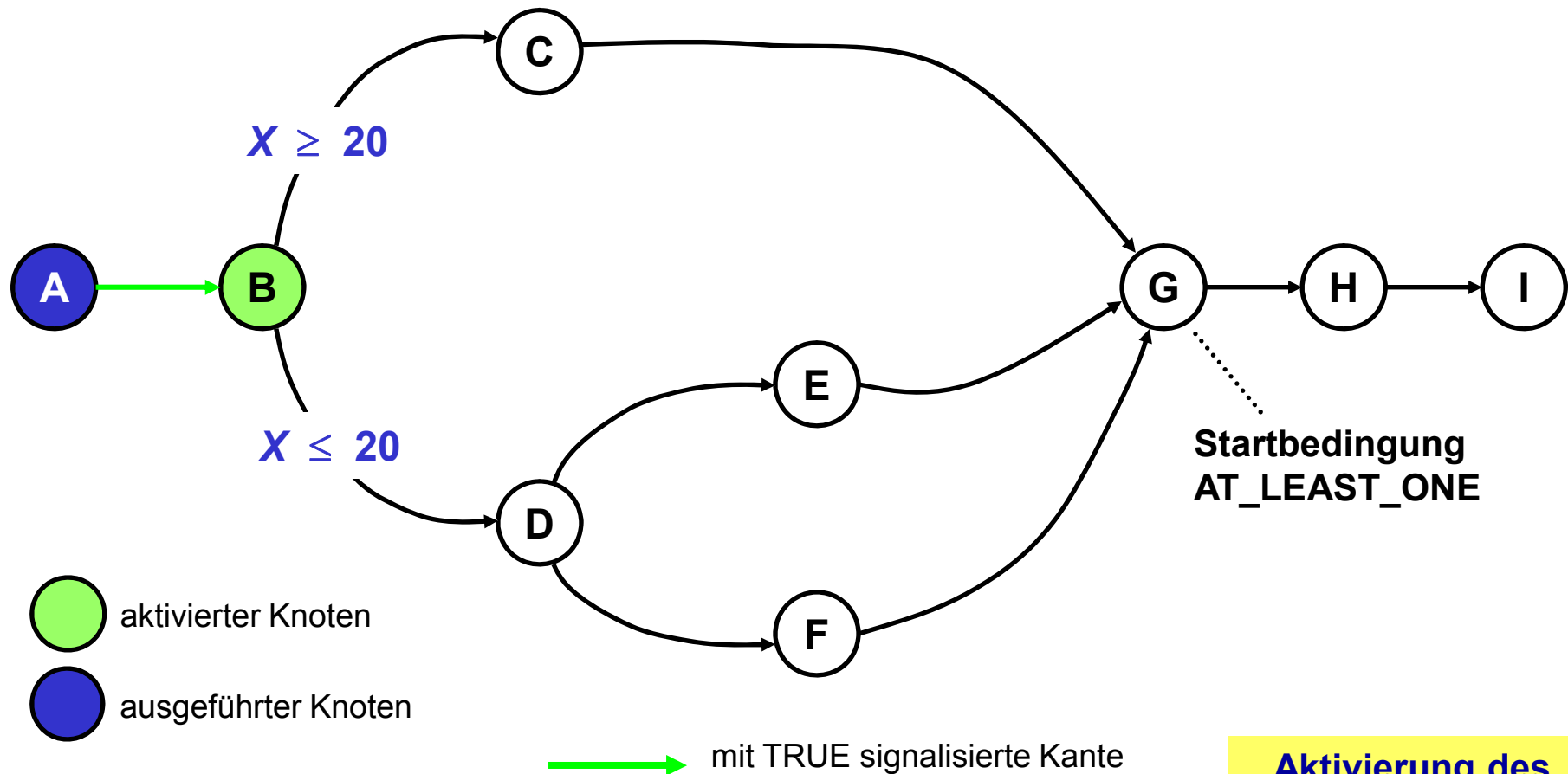


**Ausführung startet  
beim Knoten A**

## 2.5 Aktivitätensetze

### □ Kontrollflussmodellierung (Forts.)

**Beispiel 1**



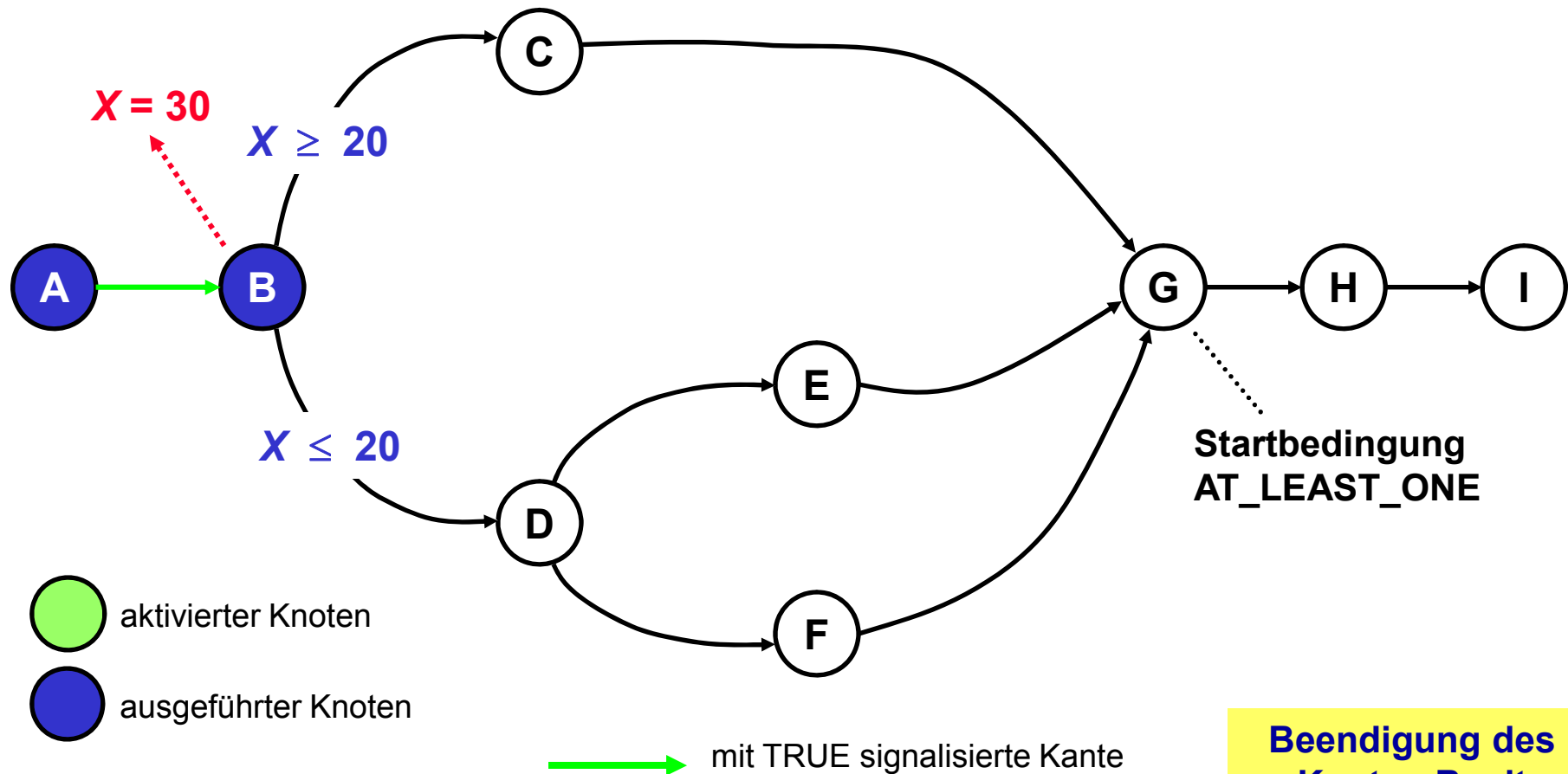
**Aktivierung des Knoten B**



## 2.5 Aktivitätensetze

### □ Kontrollflussmodellierung (Forts.)

**Beispiel 1**



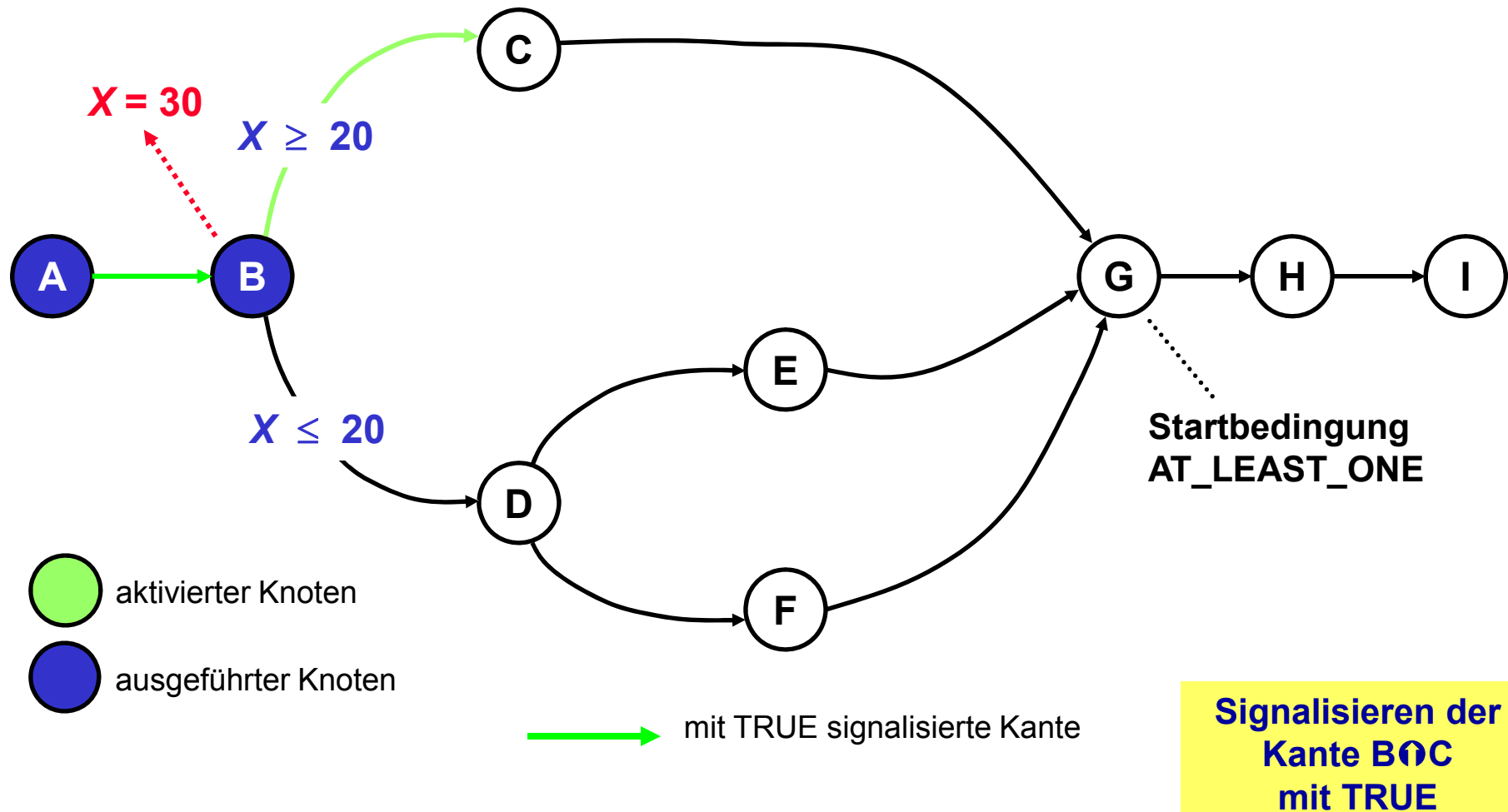
Startbedingung  
AT\_LEAST\_ONE

**Beendigung des  
Knoten B mit  
Ausgabe X = 30**

## 2.5 Aktivitätensetze

### □ Kontrollflussmodellierung (Forts.)

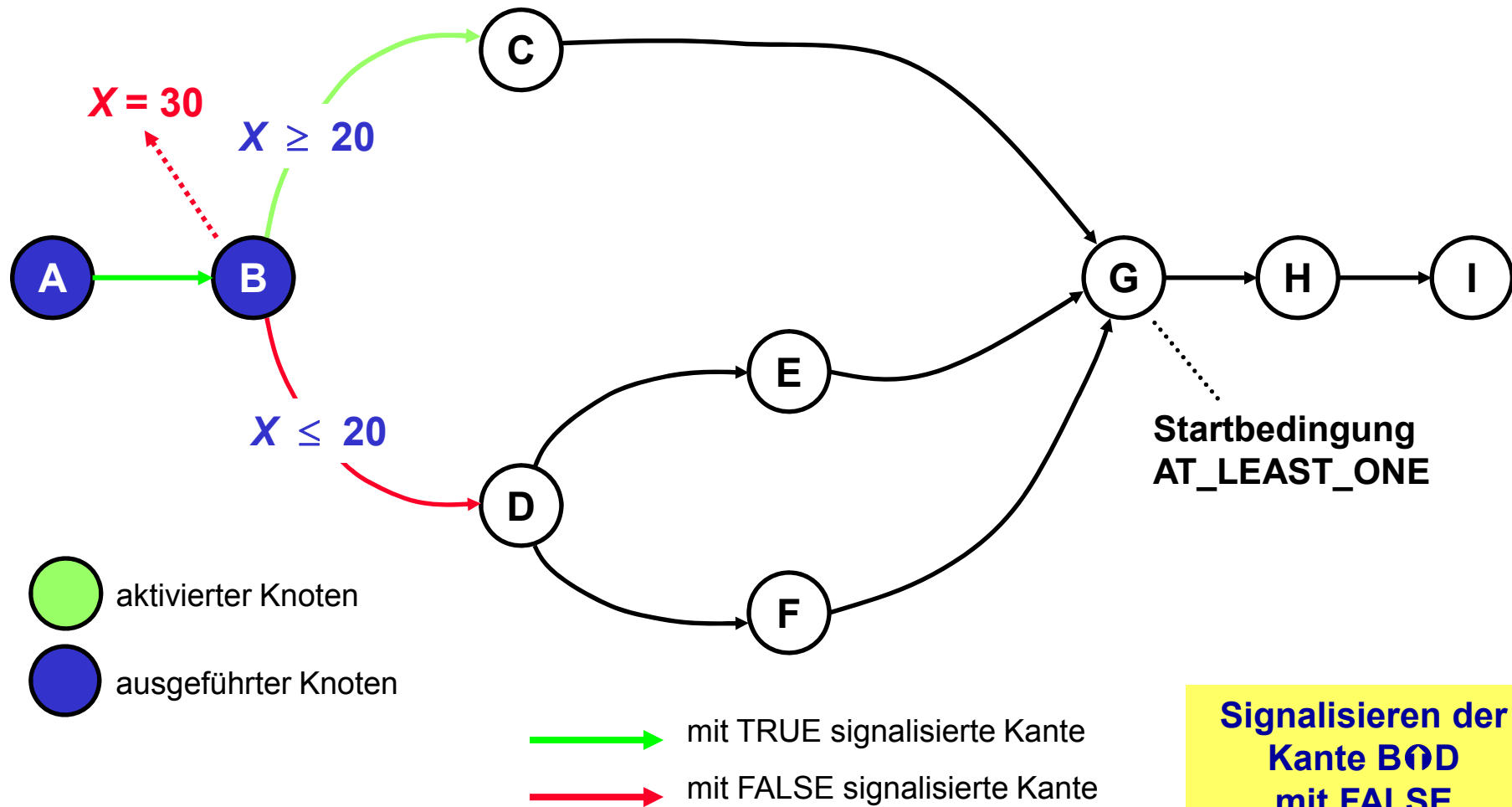
**Beispiel 1**



## 2.5 Aktivitätensetze

### □ Kontrollflussmodellierung (Forts.)

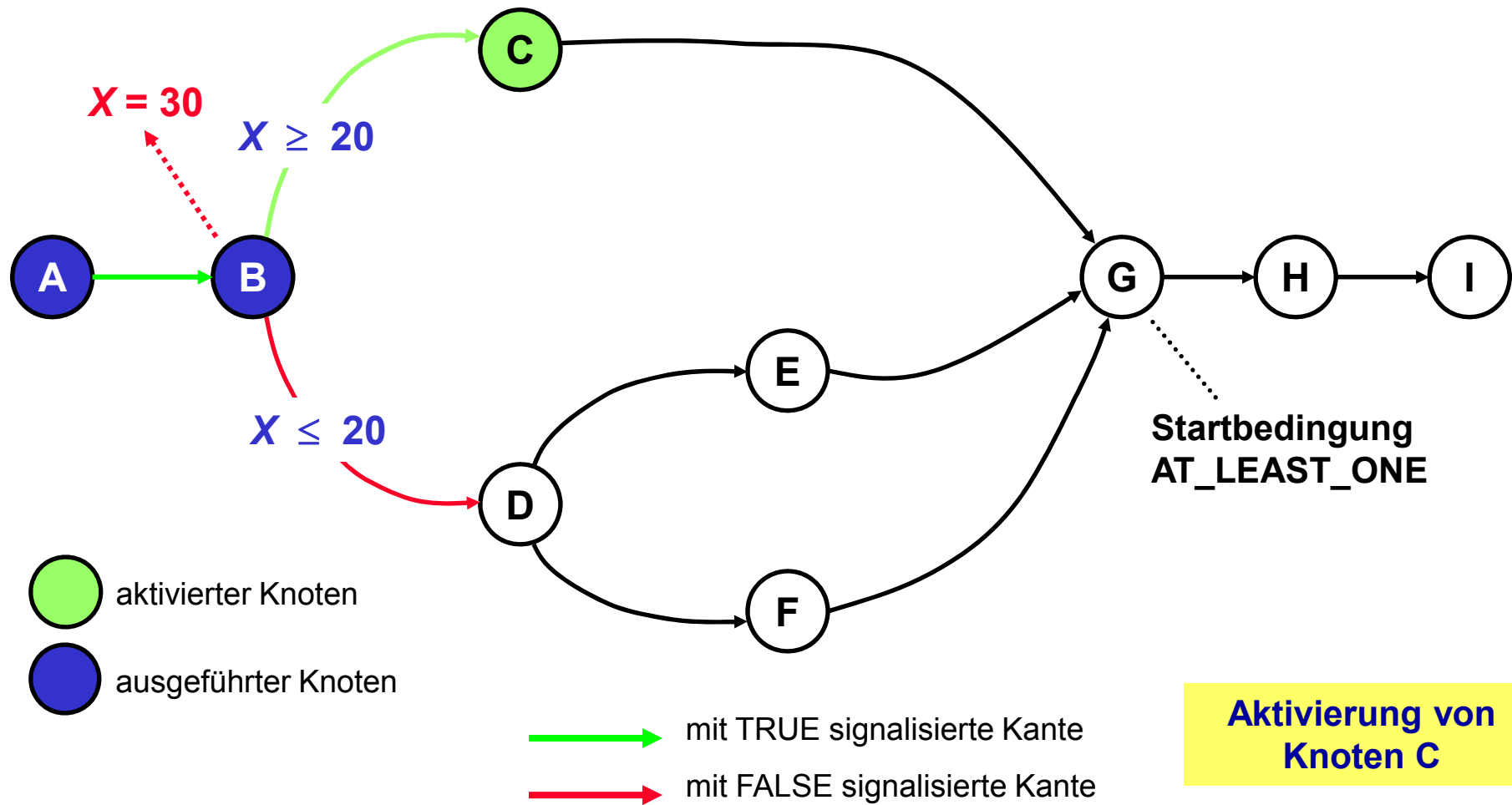
**Beispiel 1**



## 2.5 Aktivitätensetze

### □ Kontrollflussmodellierung (Forts.)

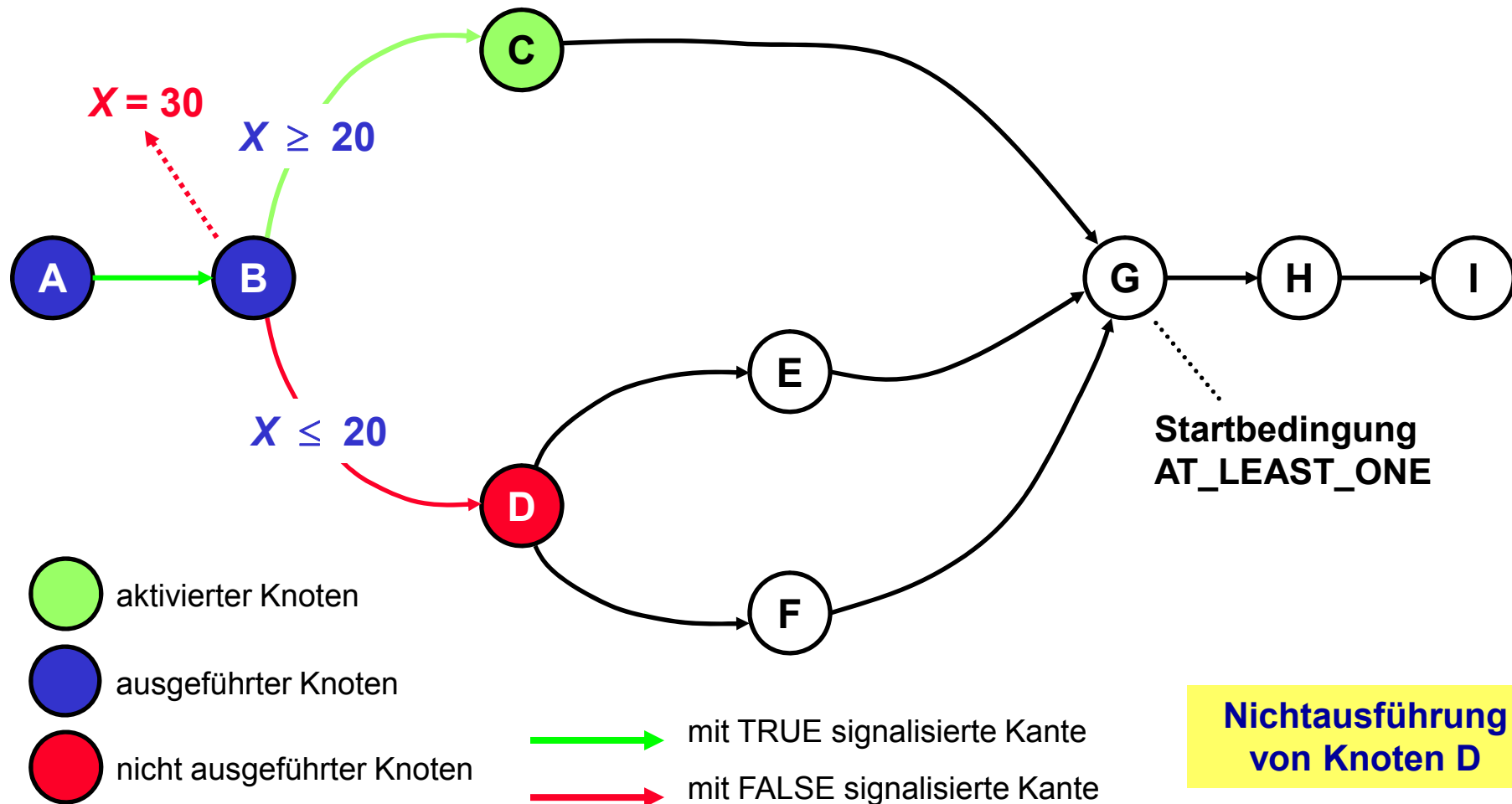
**Beispiel 1**



## 2.5 Aktivitätensetze

### □ Kontrollflussmodellierung (Forts.)

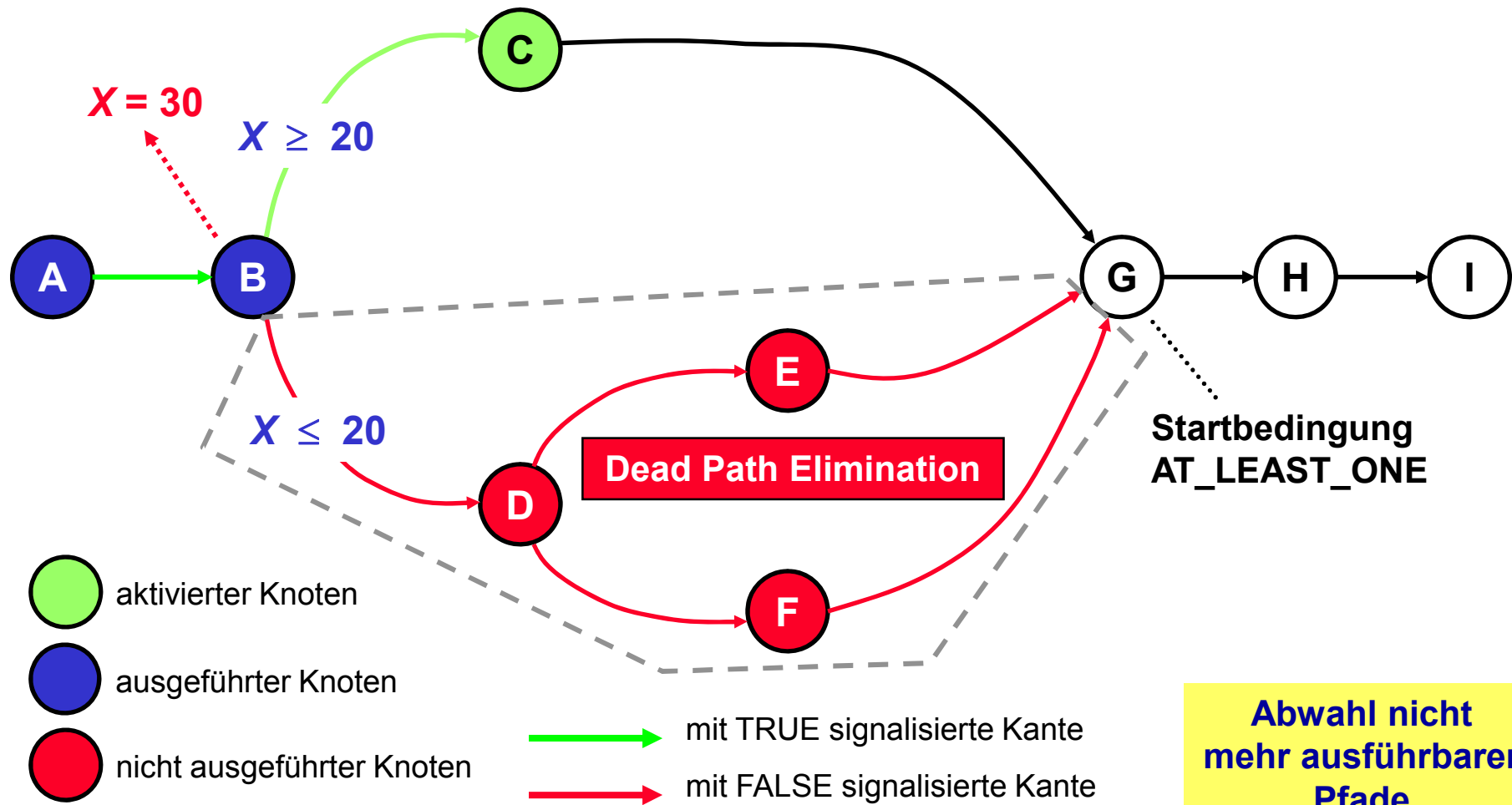
**Beispiel 1**



## 2.5 Aktivitätensetze

### □ Kontrollflussmodellierung (Forts.)

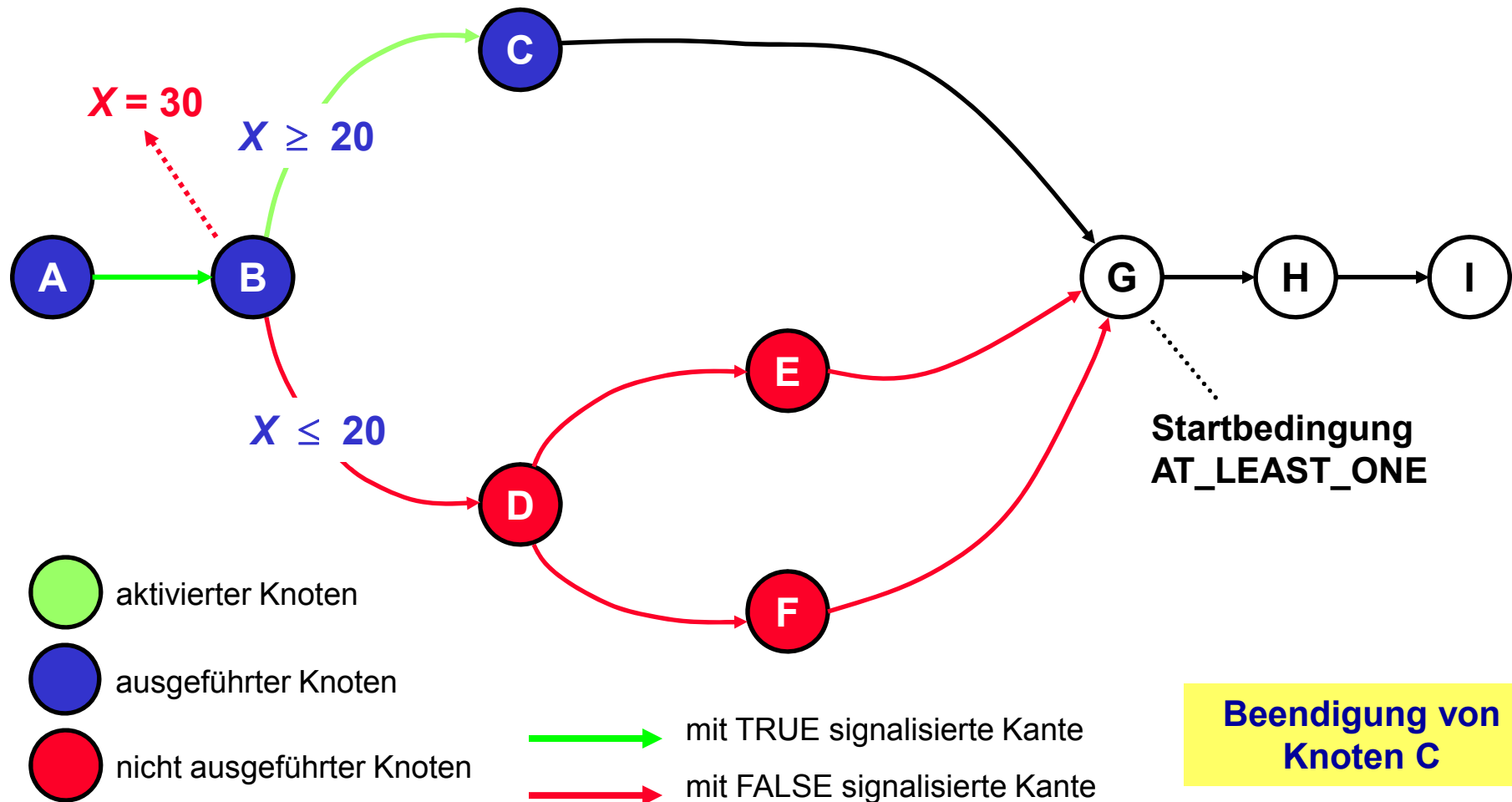
**Beispiel 1**



## 2.5 Aktivitätensetze

### □ Kontrollflussmodellierung (Forts.)

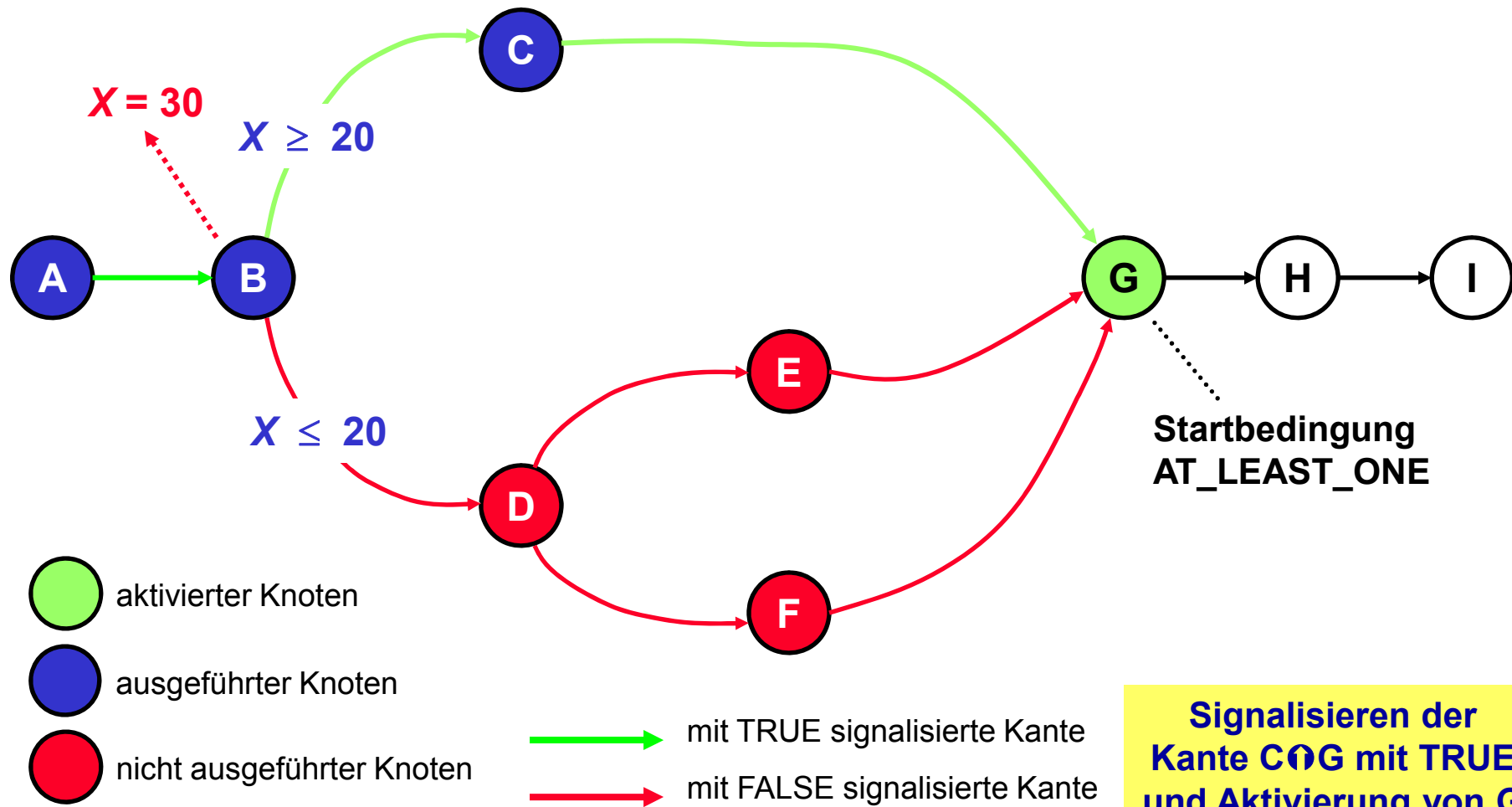
**Beispiel 1**



## 2.5 Aktivitätennetze

### □ Kontrollflussmodellierung (Forts.)

**Beispiel 1**

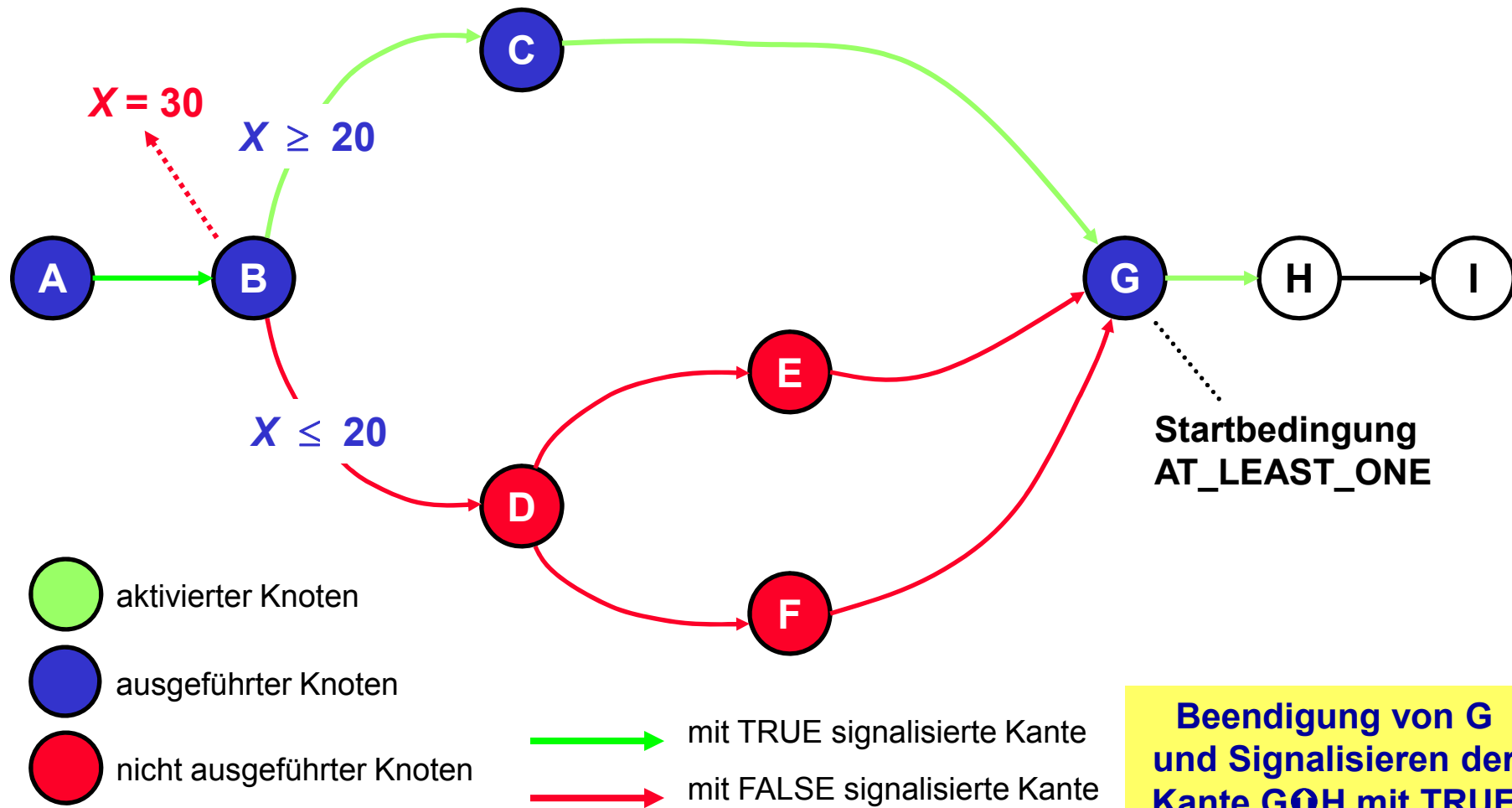




## 2.5 Aktivitätensetze

### □ Kontrollflussmodellierung (Forts.)

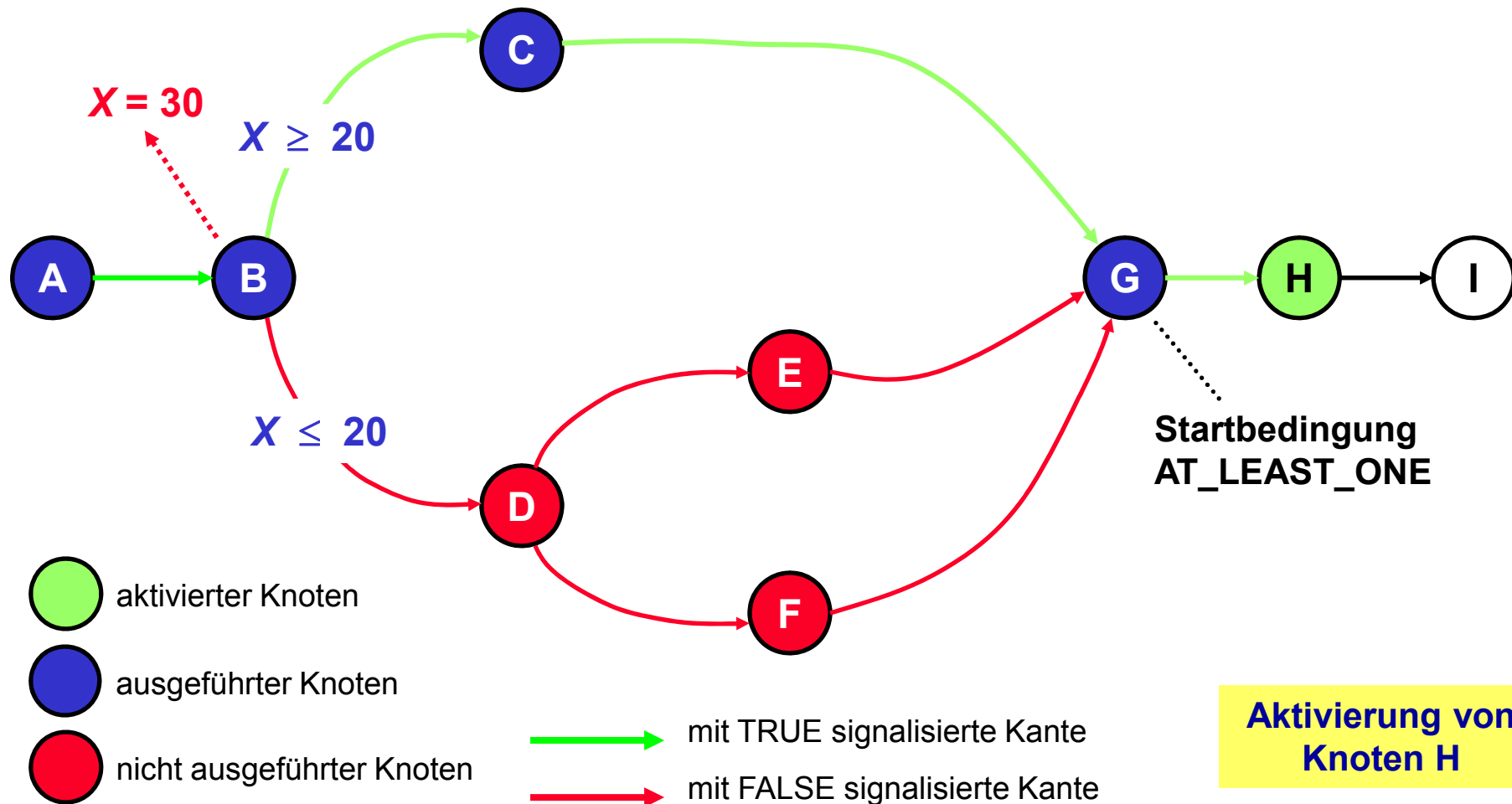
**Beispiel 1**



## 2.5 Aktivitätensnetze

### □ Kontrollflussmodellierung (Forts.)

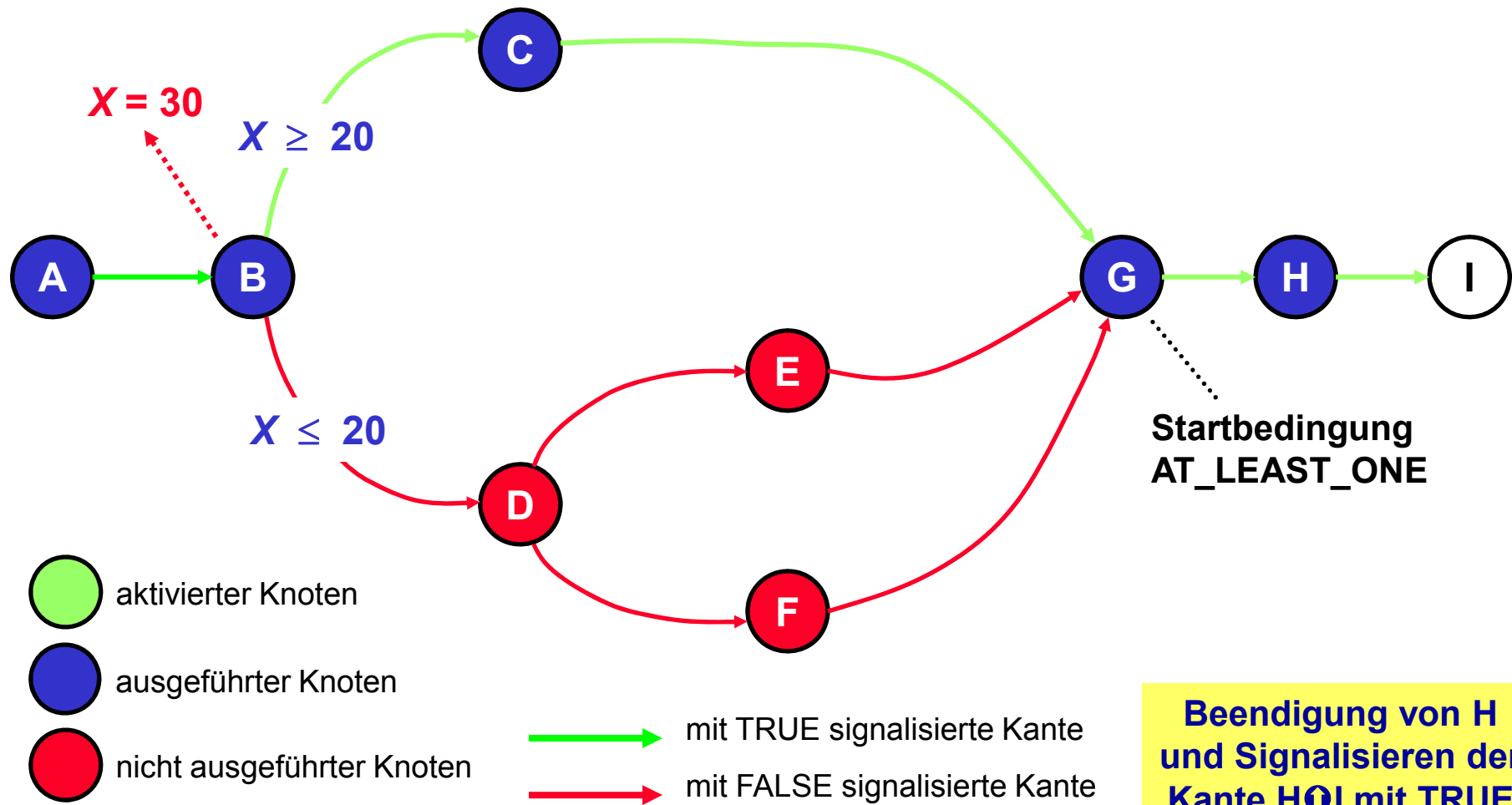
**Beispiel 1**



## 2.5 Aktivitätensetze

### □ Kontrollflussmodellierung (Forts.)

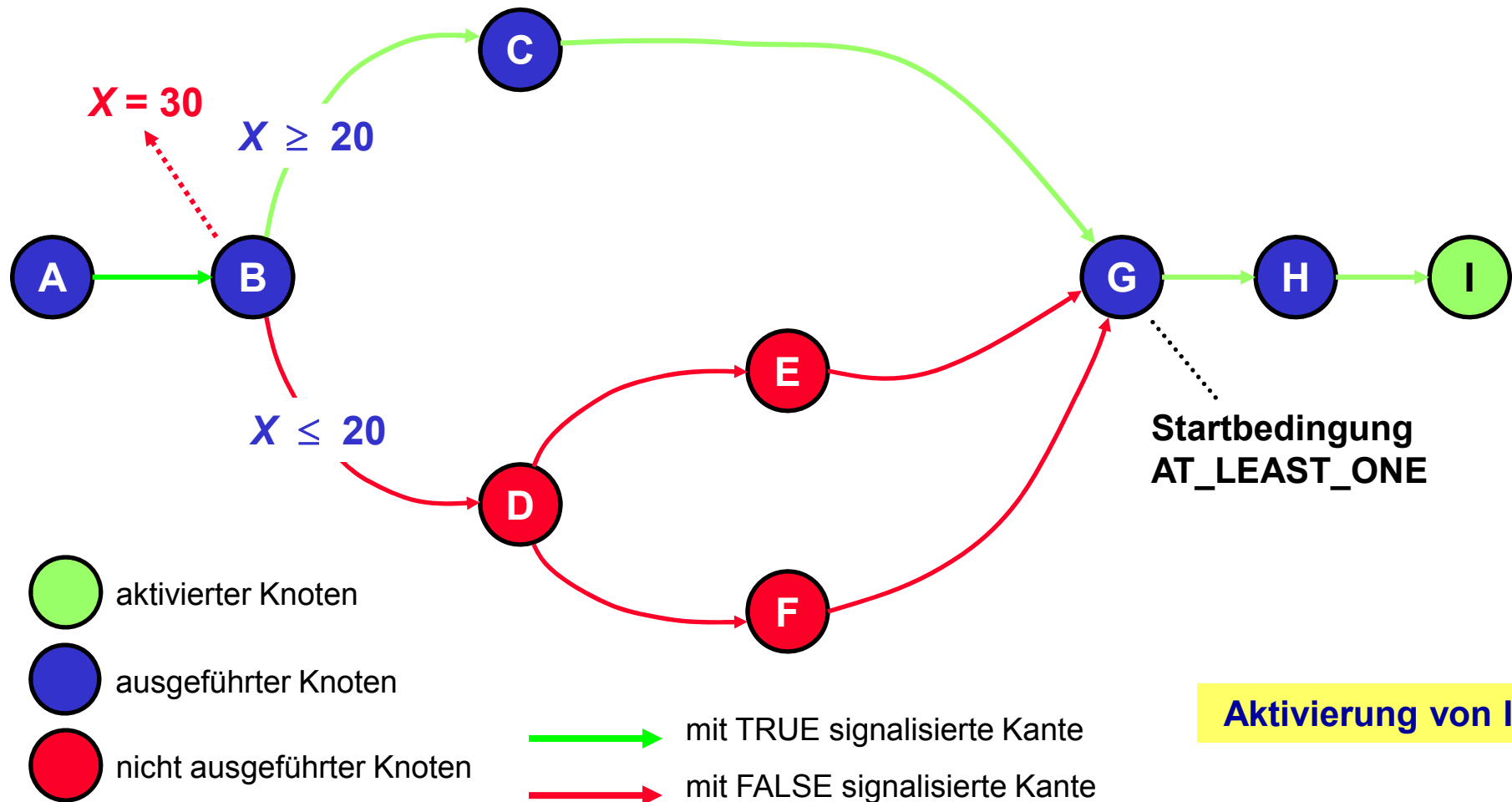
**Beispiel 1**



## 2.5 Aktivitätensetze

### □ Kontrollflussmodellierung (Forts.)

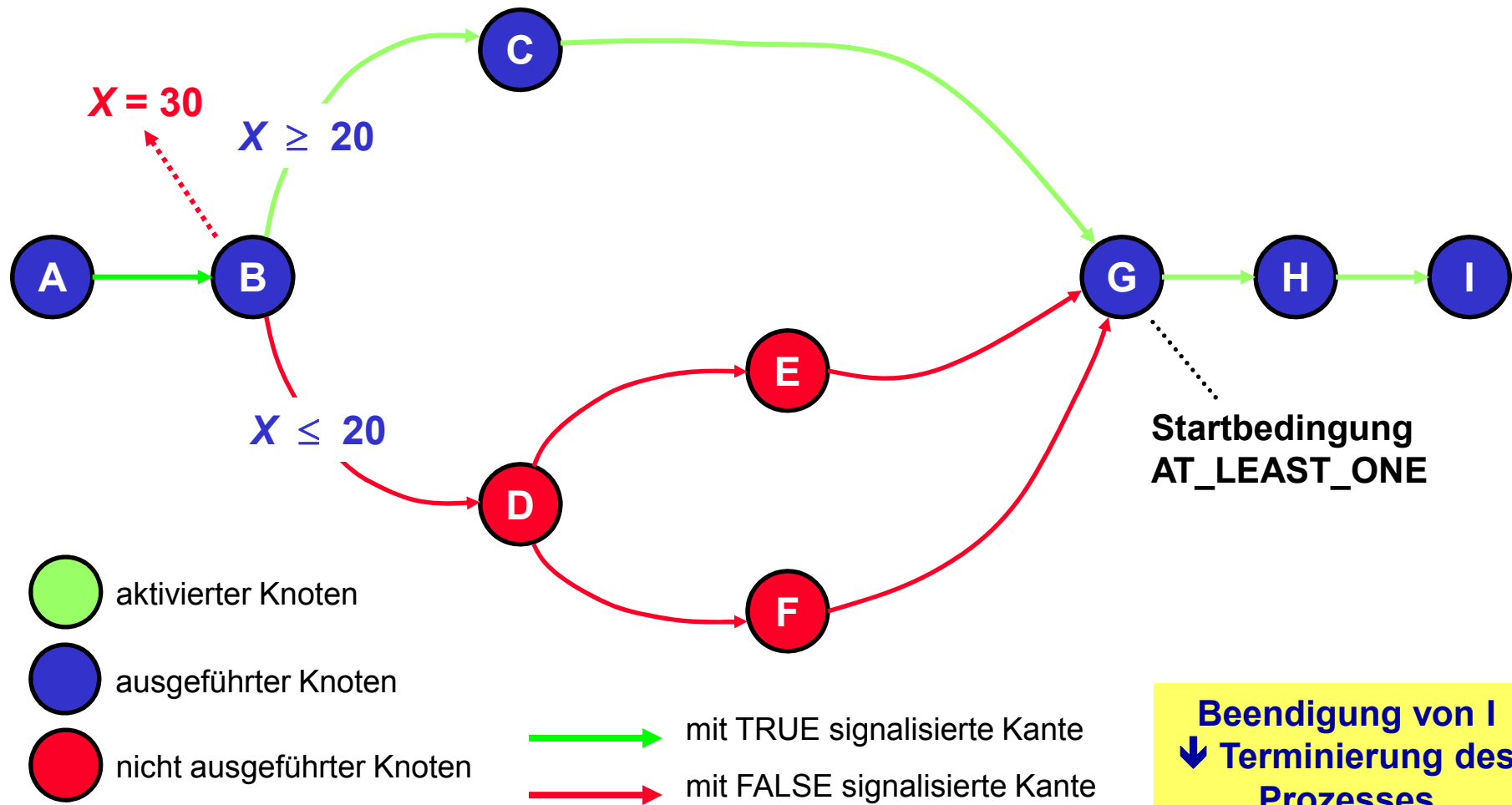
Beispiel 1



## 2.5 Aktivitätensnetze

### □ Kontrollflussmodellierung (Forts.)

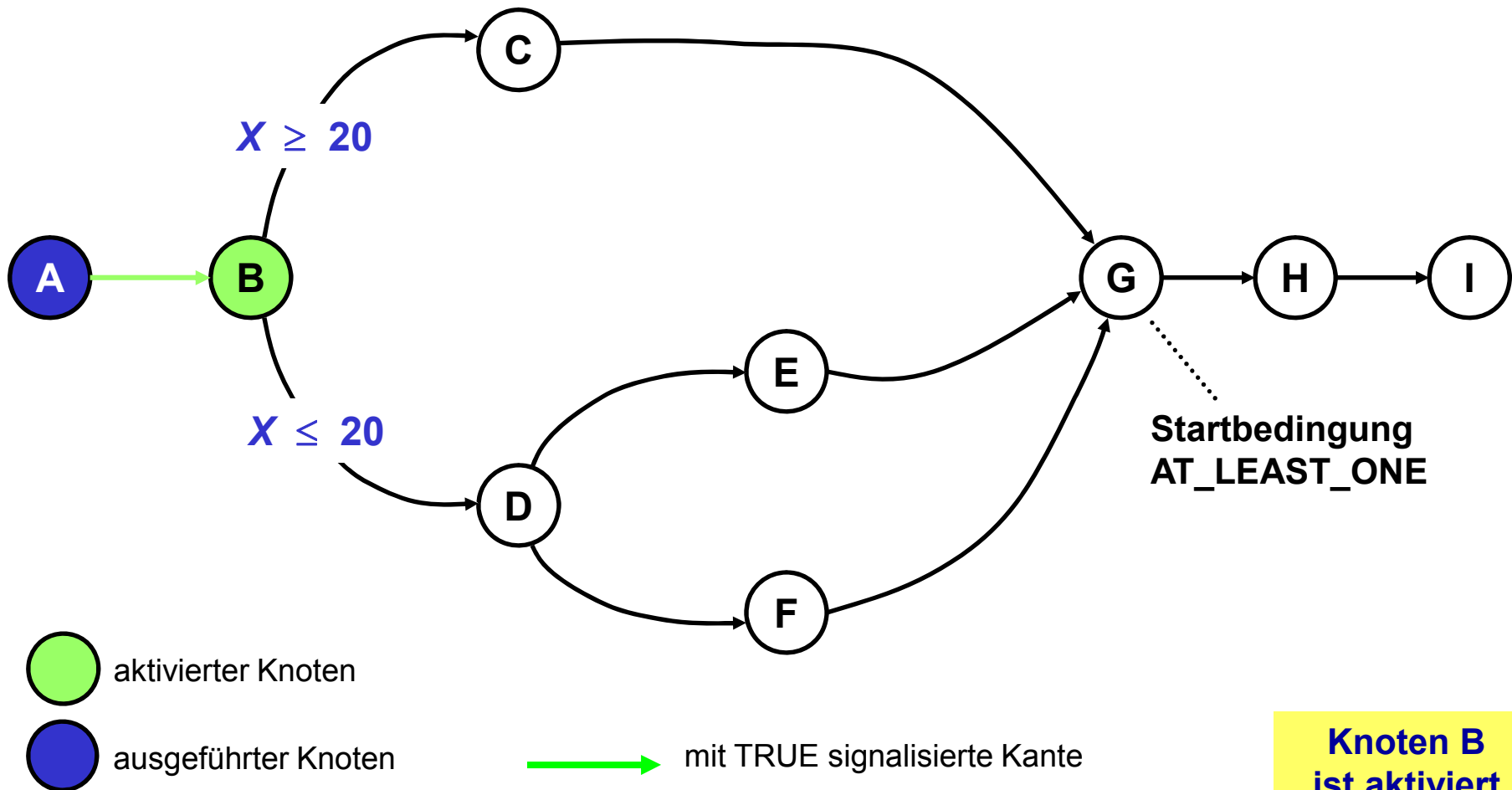
**Beispiel 1**



## 2.5 Aktivitätensetze

### □ Kontrollflussmodellierung (Forts.)

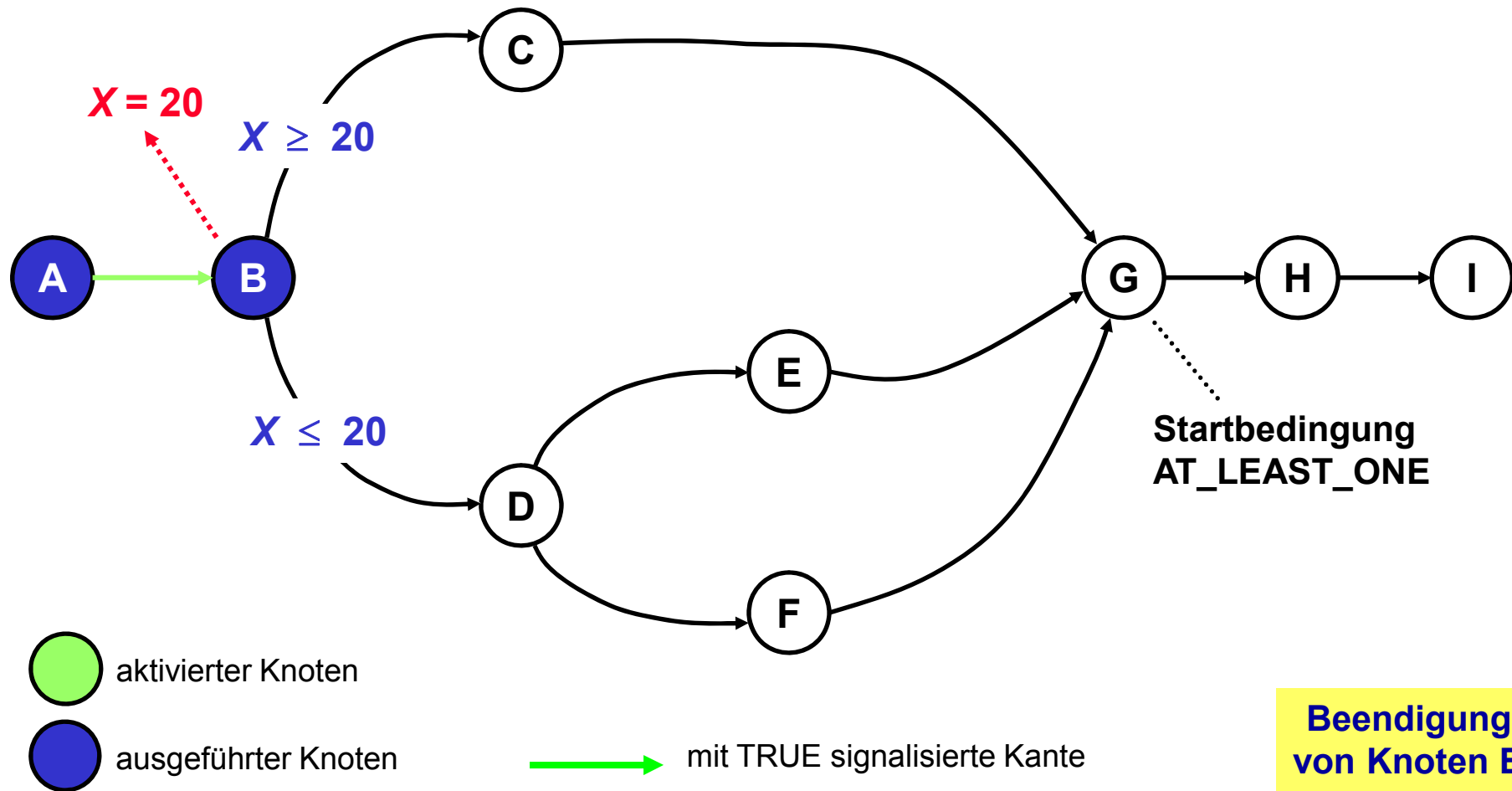
**Beispiel 2**



## 2.5 Aktivitätensetze

### □ Kontrollflussmodellierung (Forts.)

**Beispiel 2**

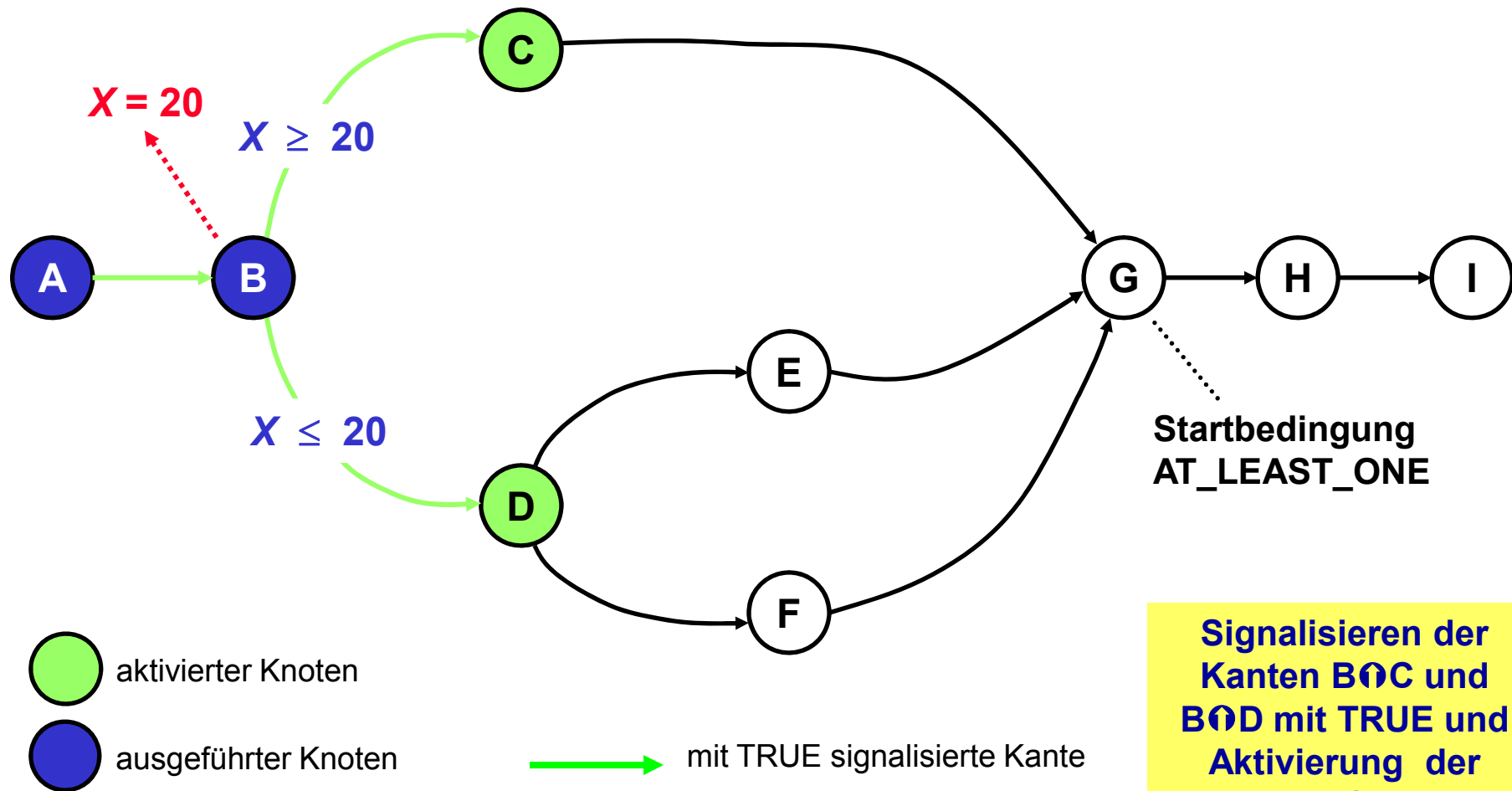


**Beendigung  
von Knoten B  
mit  $X = 20$**

## 2.5 Aktivitätensetze

### □ Kontrollflussmodellierung (Forts.)

**Beispiel 2**



Startbedingung  
AT\_LEAST\_ONE

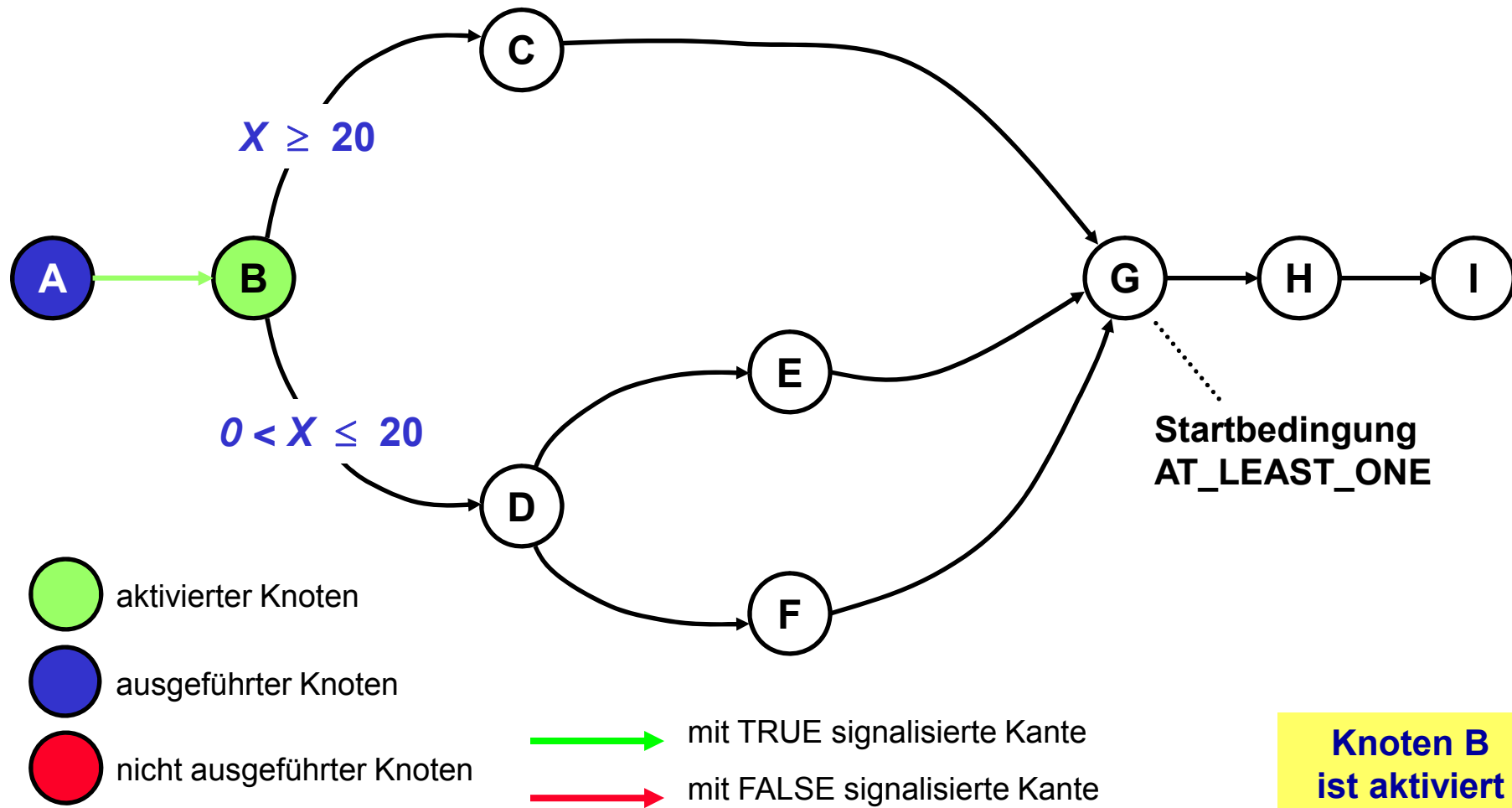
Signalisieren der  
Kanten  $B \rightarrow C$  und  
 $B \rightarrow D$  mit TRUE und  
Aktivierung der  
Knoten C und D



## 2.5 Aktivitätensetze

### □ Kontrollflussmodellierung (Forts.)

**Beispiel 3**

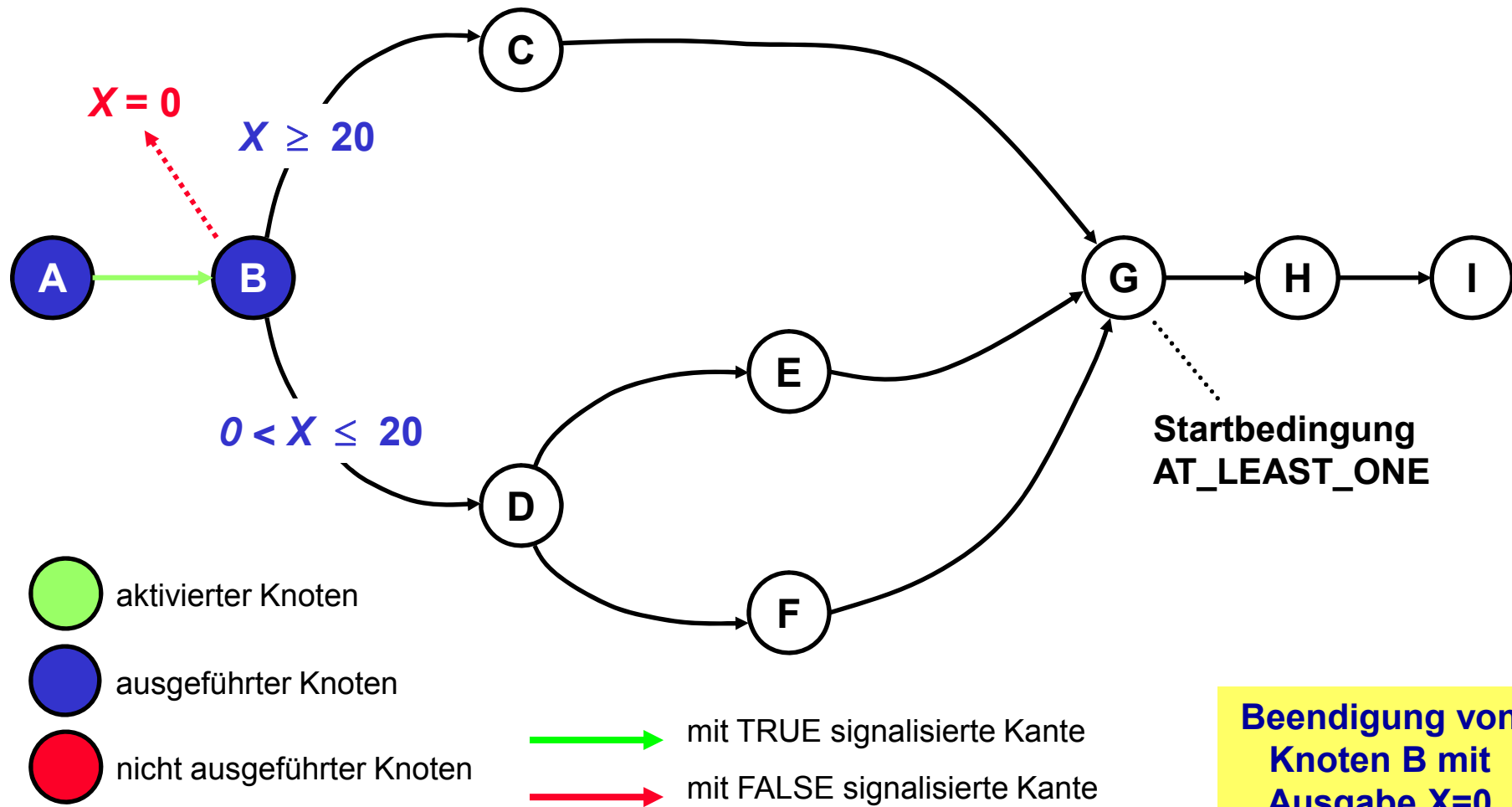


**Knoten B  
ist aktiviert**

## 2.5 Aktivitätensetze

### □ Kontrollflussmodellierung (Forts.)

**Beispiel 3**

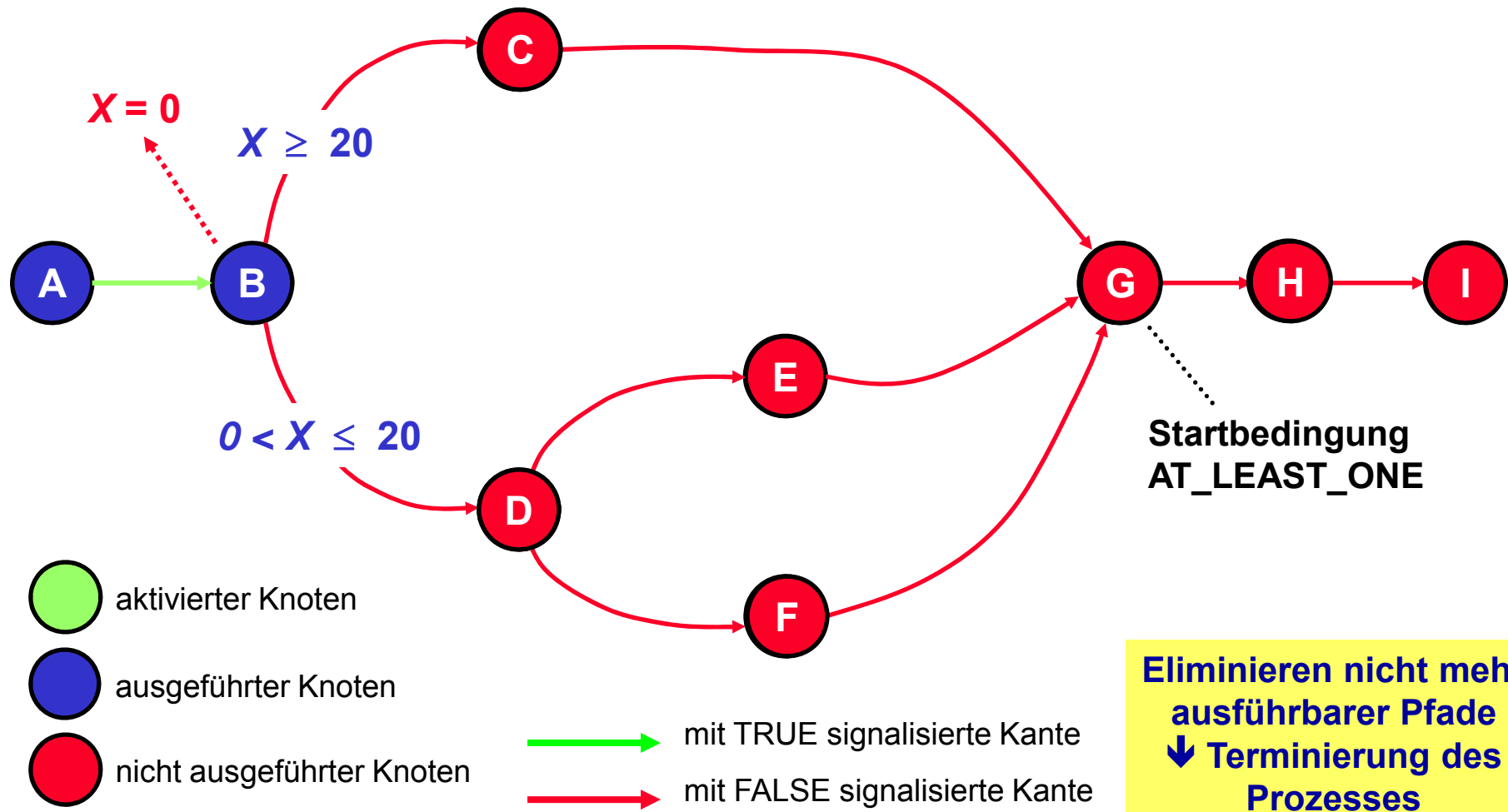


**Beendigung von  
Knoten B mit  
Ausgabe X=0**

## 2.5 Aktivitätensetze

### □ Kontrollflussmodellierung (Forts.)

Beispiel 3



## 2.5 Aktivitätennetze

---

### □ Kontrollflussmodellierung (Forts.)

#### Zusammenfassung

- Der Kontrollfluss definiert die Bearbeitungsreihenfolgen und -bedingungen für die einzelnen Prozessschritte (= Aktivitäten)
- Bearbeitungsreihenfolgen werden durch Verknüpfung von Aktivitätenknoten mittels Kontrollkonnektoren (= gerichtete Kontrollkanten) festgelegt
- Ein Kontrollkonnektor  $A \rightarrow B$  legt semantisch fest, dass – vorausgesetzt beide Aktivitäten kommen zur Ausführung – die Bearbeitung von A beendet sein muss, bevor B aktiviert werden darf
- Ein von einer Aktivität ausgehender Kontrollkonnektor kann (optional) mit einer Transitionsbedingung verknüpft werden – diese wertet Ausgabedaten der Quellaktivität aus
- Bei einer Verweigung kann ein Kontrollkonnektor ohne Transitionsbedingung als Default-Konnektor gekennzeichnet werden ( $\rightarrow \bigcirc \rightarrow$ ).  
Er wird dann gewählt, wenn keine der Transitionsbedingungen zutrifft.

## 2.5 Aktivitätennetze

---

### □ Kontrollflussmodellierung (Forts.)

#### Zusammenfassung (Forts.)

##### Bedingungen zur Aktivierung einer Aktivität X (Startbedingung)

- [B1] Notwendige Bedingung: Jeder direkte Vorgänger von X wurde beendet oder es steht fest, dass er nicht mehr zur Ausführung kommen kann.
- [B2] Hinreichende Bedingung: Bedingung [B1] ist erfüllt und
- **Join-Semantik = ALL** ↓ Die Transitionsbedingungen aller in X einmündenden Kontrollkonnektoren sind erfüllt
  - **Join-Semantik = AT-LEAST-ONE** ↓ Die Transitionsbedingung mindestens eines einmündenden Kontrollkonnektors ist erfüllt

## 2.5 Aktivitätennetze

---

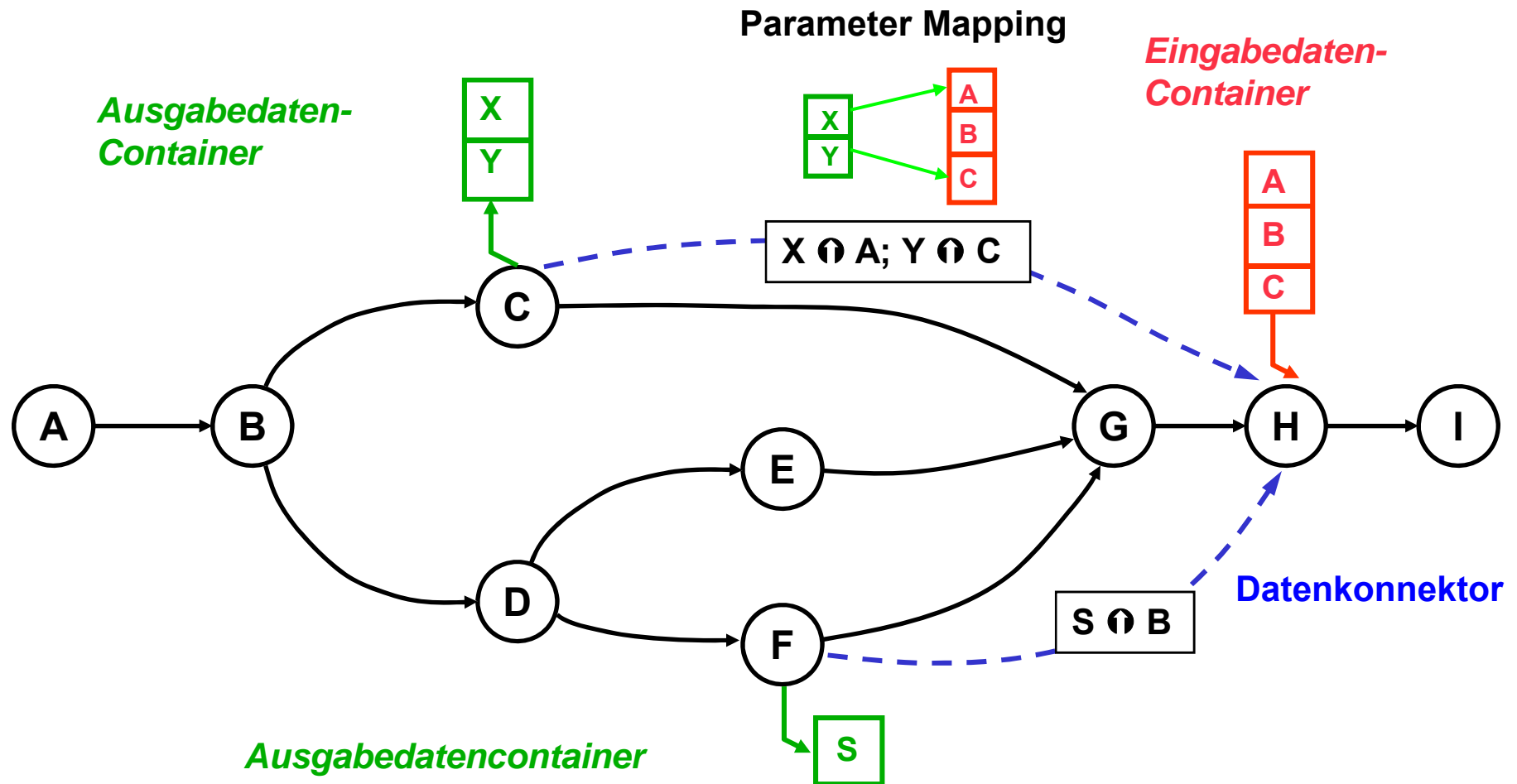
### □ Kontrollflussmodellierung (Forts.)

#### Zusammenfassung (Forts.)

- Durch Verknüpfung der Aktivitäten mittels Kontrollkonnektoren und Wahl geeigneter Transitionsbedingungen lassen sich sequenzielle, alternative und parallele Pfade modellieren
- Aktivitätennetze müssen azyklisch sein, d. h. Schleifen sind nicht (direkt) abbildbar
- Jeder Knoten eines Aktivitätennetzes muss mindestens einen assoziierten Kontrollkonnektor besitzen
- Die Ausführung eines Aktivitätennetzes beginnt mit den Knoten, in die kein Kontrollkonnektor einmündet
- Die Ausführung eines Aktivitätennetzes ist beendet, wenn kein Knoten mehr aktiviert werden kann

## 2.5 Aktivitätennetze

### □ Datenflussmodellierung



## 2.5 Aktivitätennetze

---

### □ Datenflussmodellierung (Forts.)

#### Zusammenfassung

- Jeder Aktivität ist ein Eingabedatencontainer und ein Ausgabedatencontainer zugeordnet, der ihre Eingabedaten bzw. Ausgabedaten zusammenfasst
- Ein Datencontainer kann mehrere Datenfelder umfassen
- Ein Datenfeld kann elementaren Datentyp (Integer, Char, ...) oder komplexe Datenstruktur (Nested Records, Felder fester Länge, etc.) umfassen
- Zur Modellierung von Datenflüssen zwischen zwei Aktivitäten, müssen die Aktivitätenknoten durch einen Datenkonnektor verknüpft werden
- Ein Datenkonnektor von A nach B ist nur zulässig, wenn B ein direkter oder indirekter Nachfolger von A im Kontrollfluss ist.
- Führt von der Aktivität A ein Datenkonnektor zur Aktivität B, so können Felder des Ausgabecontainers von A auf Felder des Eingabecontainers von B abgebildet werden
- Ein Datenaustausch zwischen Super- und Sub-Workflow ist ebenfalls möglich (via Eingabe- und Ausgabedatencontainer des Sub-Workflows)



## 2.5 Aktivitätennetze

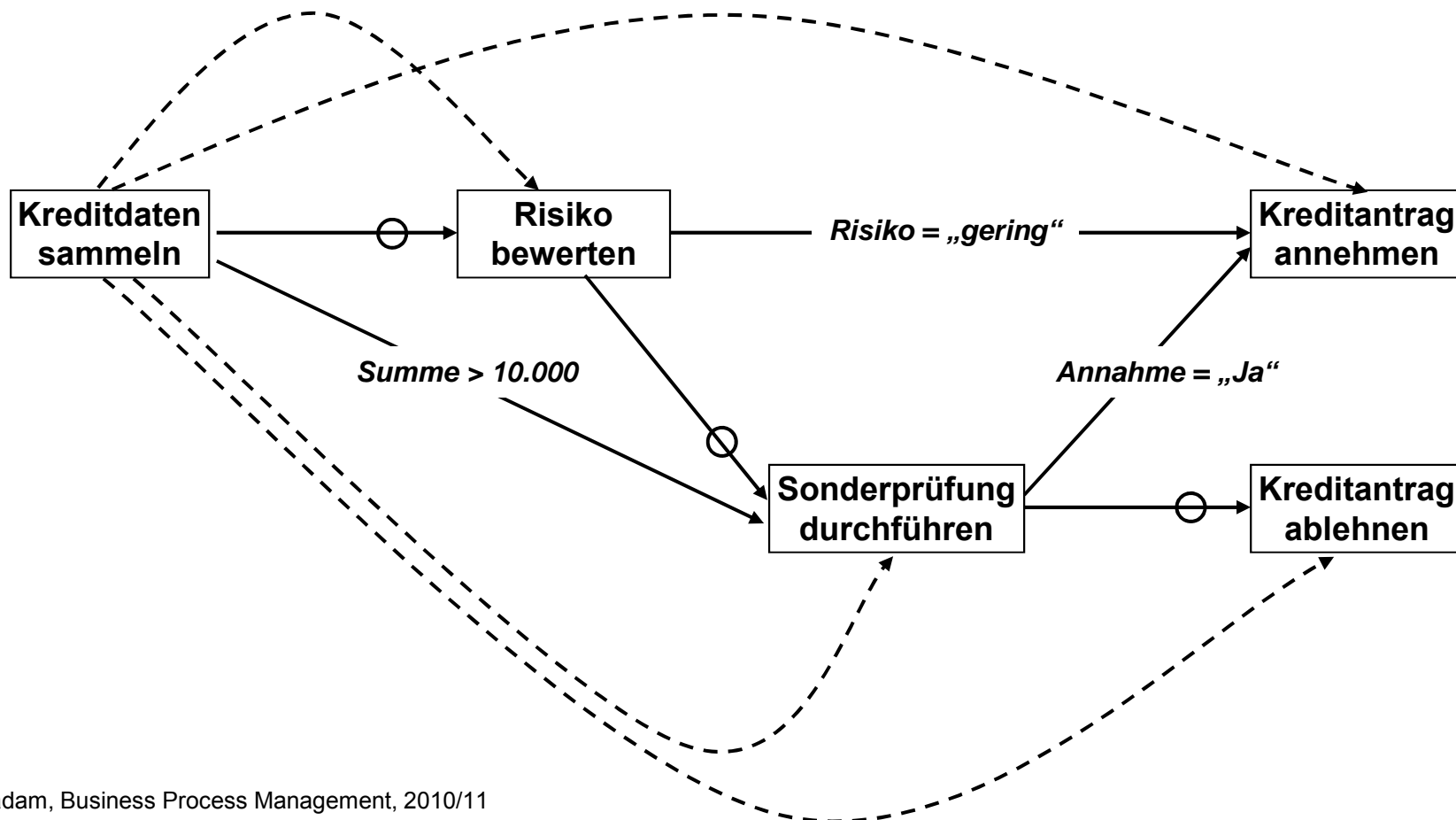
---

### □ Beispiel: Kreditantragstellung

- Zunächst füllt ein Schalterangestellter einer Bankfiliale X – zusammen mit dem Kunden – den Kreditantrag aus (Aktivität „Kreditantrag ausfüllen“). Darin werden der Name und die Adresse des Kunden sowie der Zweck des Kredites und die Kreditsumme festgelegt.
- Anschließend erfolgt, abhängig von der beantragten Kreditsumme, die Überprüfung des Kredits:
  1. Beträgt die **Kreditsumme mehr als 10.000 €**, muss eine Sonderprüfung durch einen Sachbearbeiter der Kreditabteilung erfolgen (Aktivität „Sonderprüfung durchführen“). Dabei wird entschieden, ob der Kredit angenommen oder abgelehnt wird (Annahme = „Ja“/„Nein“). Davon abhängig führt dann der Schalterangestellte, der den Kreditantrag ausgefüllt hat, entweder die Tätigkeit „Kreditantrag annehmen“ oder „Kreditantrag ablehnen“ durch.
  2. Beträgt die **Kreditsumme** dagegen **maximal 10.000 €**, kann auf die Sonderprüfung ggf. verzichtet werden. Dazu bewertet der Schalterangestellte zunächst das Risiko des Kredites (Aktivität „Risiko bewerten“). Stuft er dieses als gering ein, kann er den Kreditantrag sofort, d.h. ohne weitere Überprüfungen, annehmen (Aktivität „Kreditantrag annehmen“). Andernfalls wird wie in 1. verfahren, d.h. es wird eine Sonderprüfung durch die Kreditabteilung durchgeführt und – davon abhängig – vom Schalterangestellten der Kreditantrag entweder angenommen oder abgelehnt (Durchführung von „Kreditantrag annehmen“ oder „Kreditantrag ablehnen“).

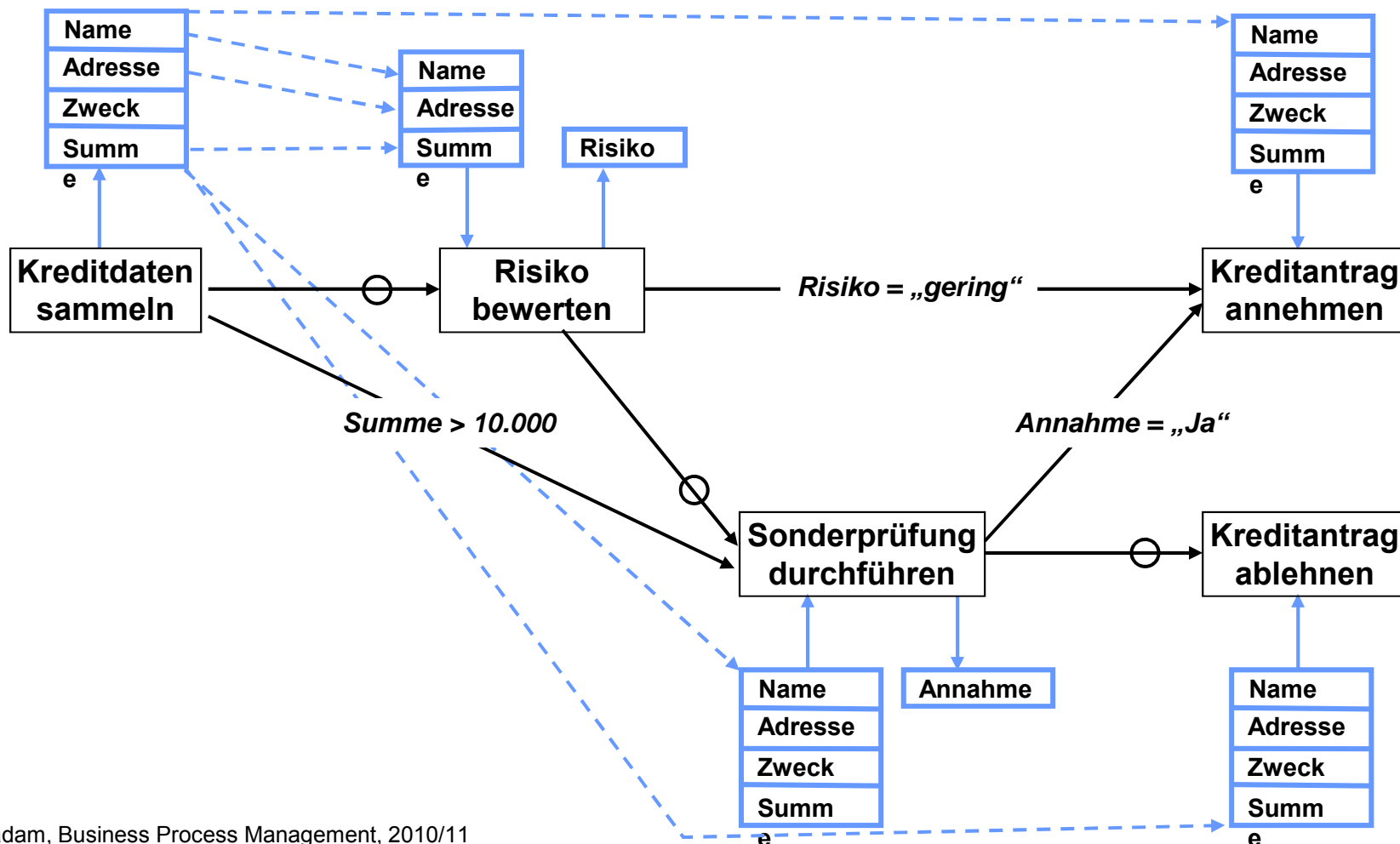
## 2.5 Aktivitätennetze

### □ Beispiel: Kreditantragstellung – eine mögliche Lösung



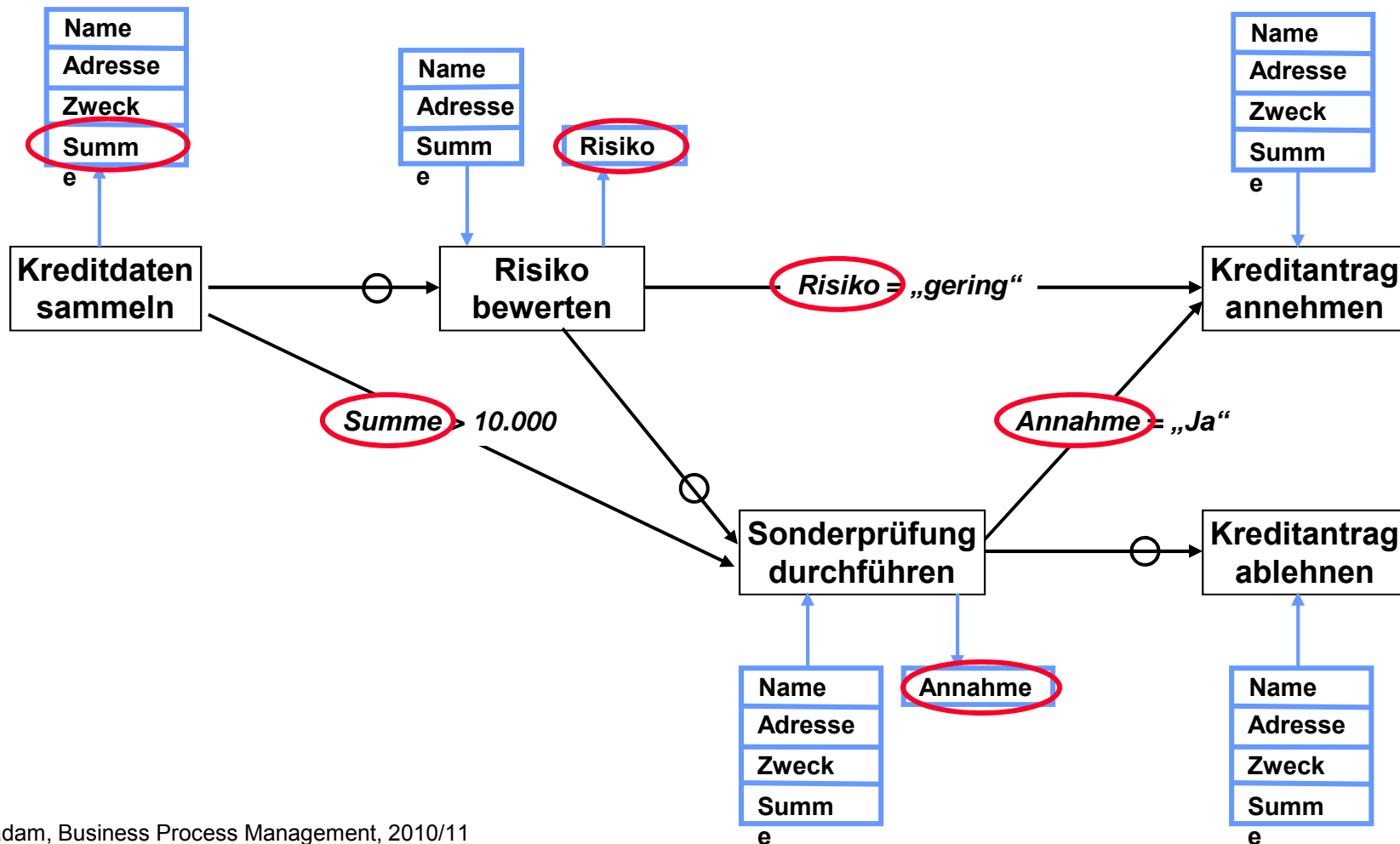
## 2.5 Aktivitätennetze

### □ Beispiel: Kreditantragstellung – eine mögliche Lösung



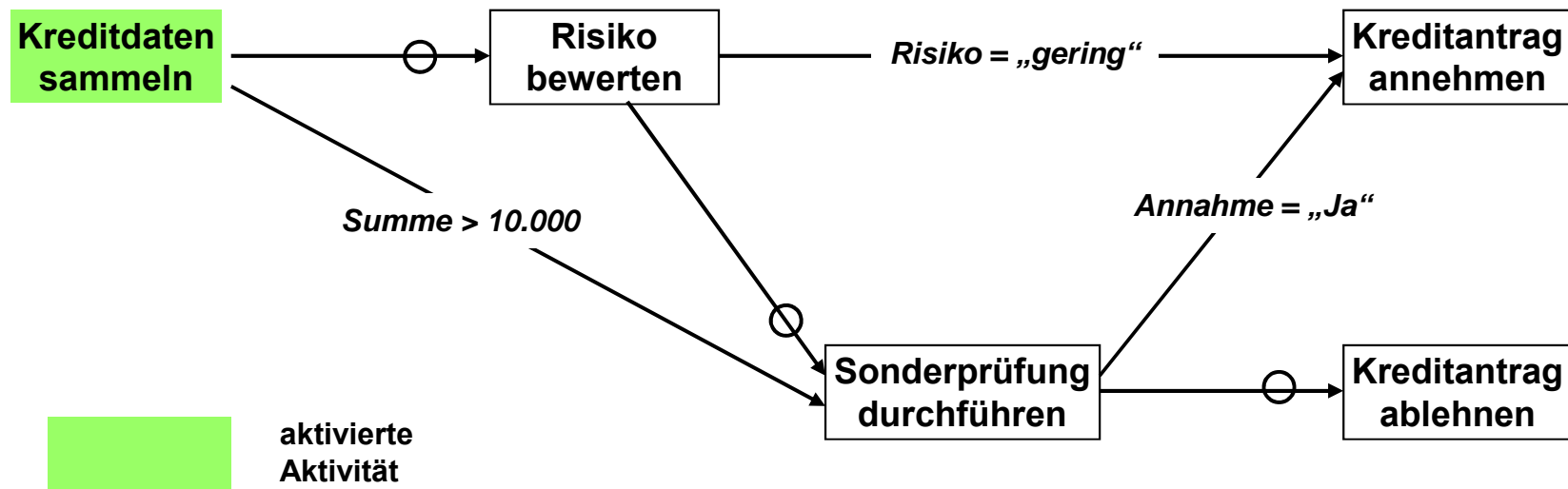
## 2.5 Aktivitätennetze

### □ Beispiel: Kreditantragstellung – eine mögliche Lösung

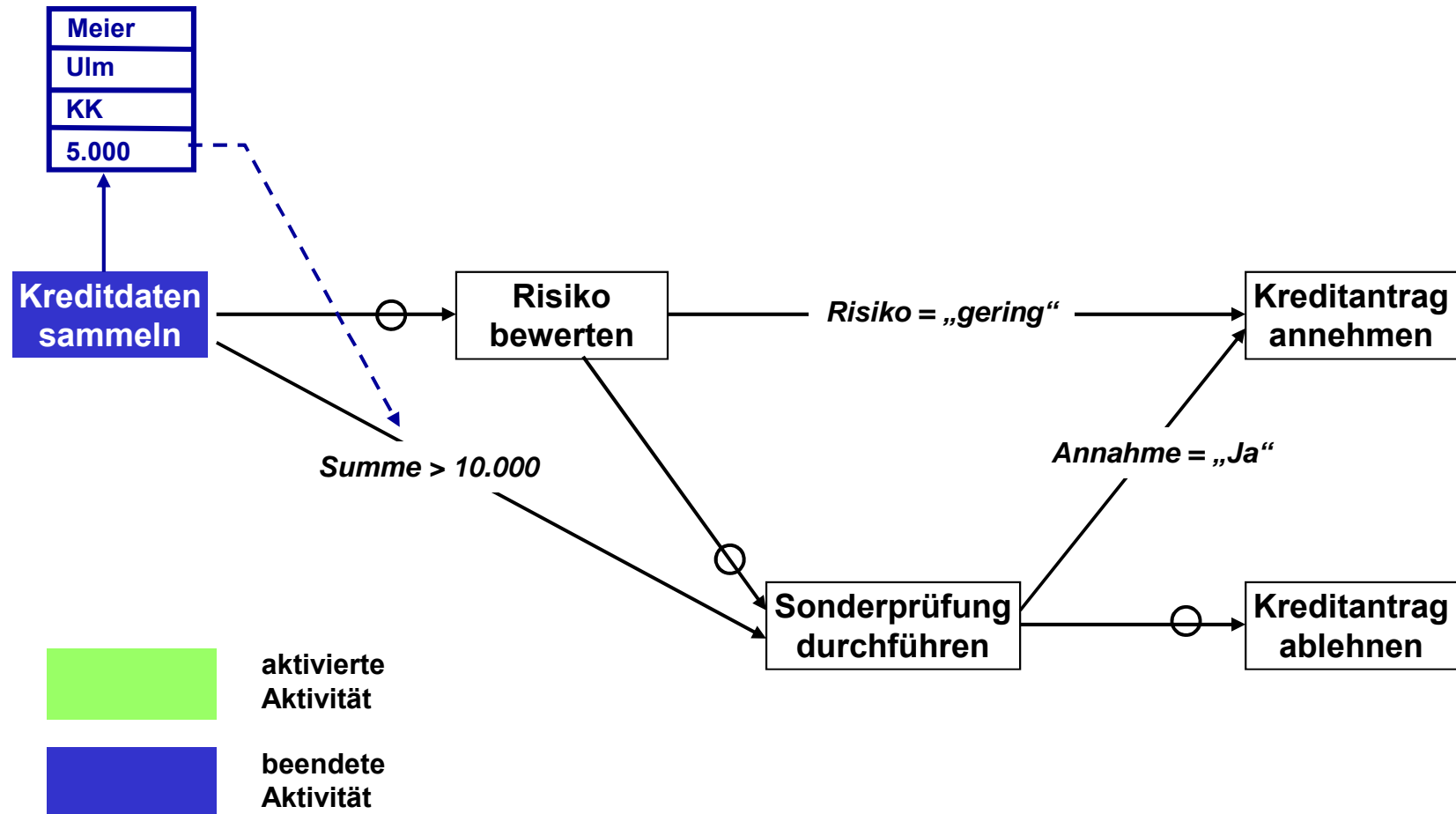


## 2.5 Aktivitätensetze

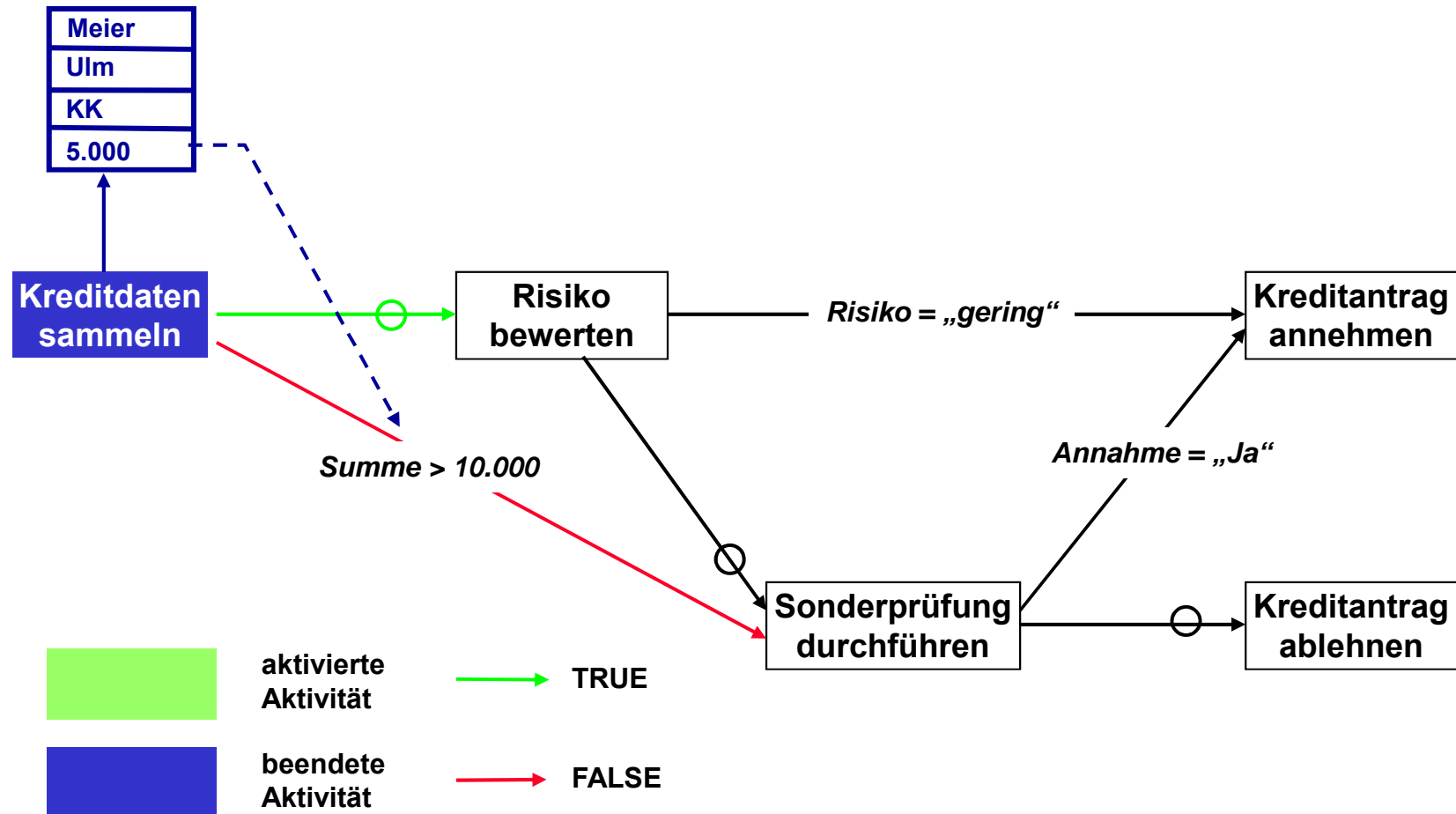
- Beispiel: Kreditantragstellung – eine mögliche Lösung – **Anwendungsbeispiel**



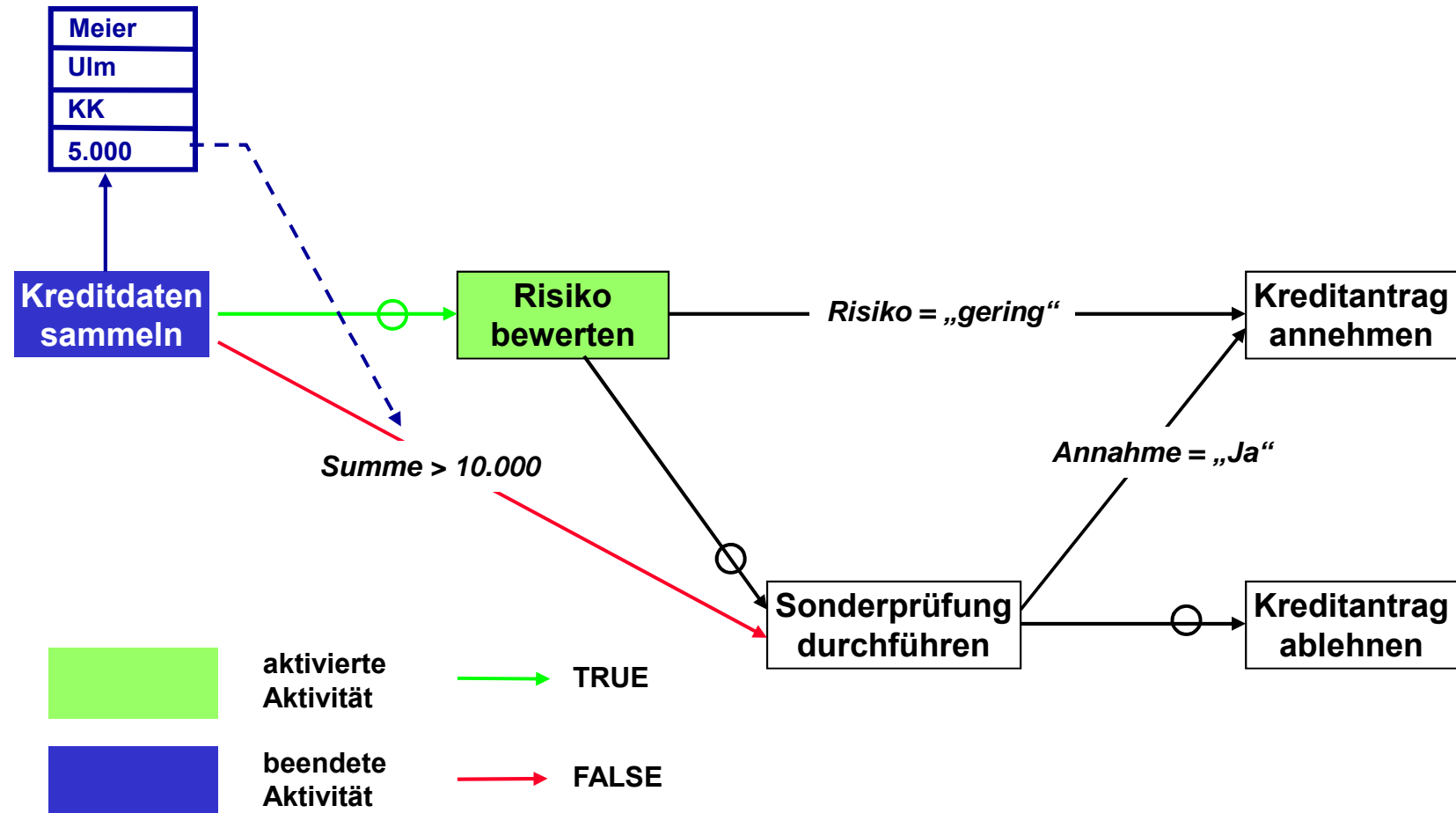
## 2.5 Aktivitätennetze



## 2.5 Aktivitätennetze

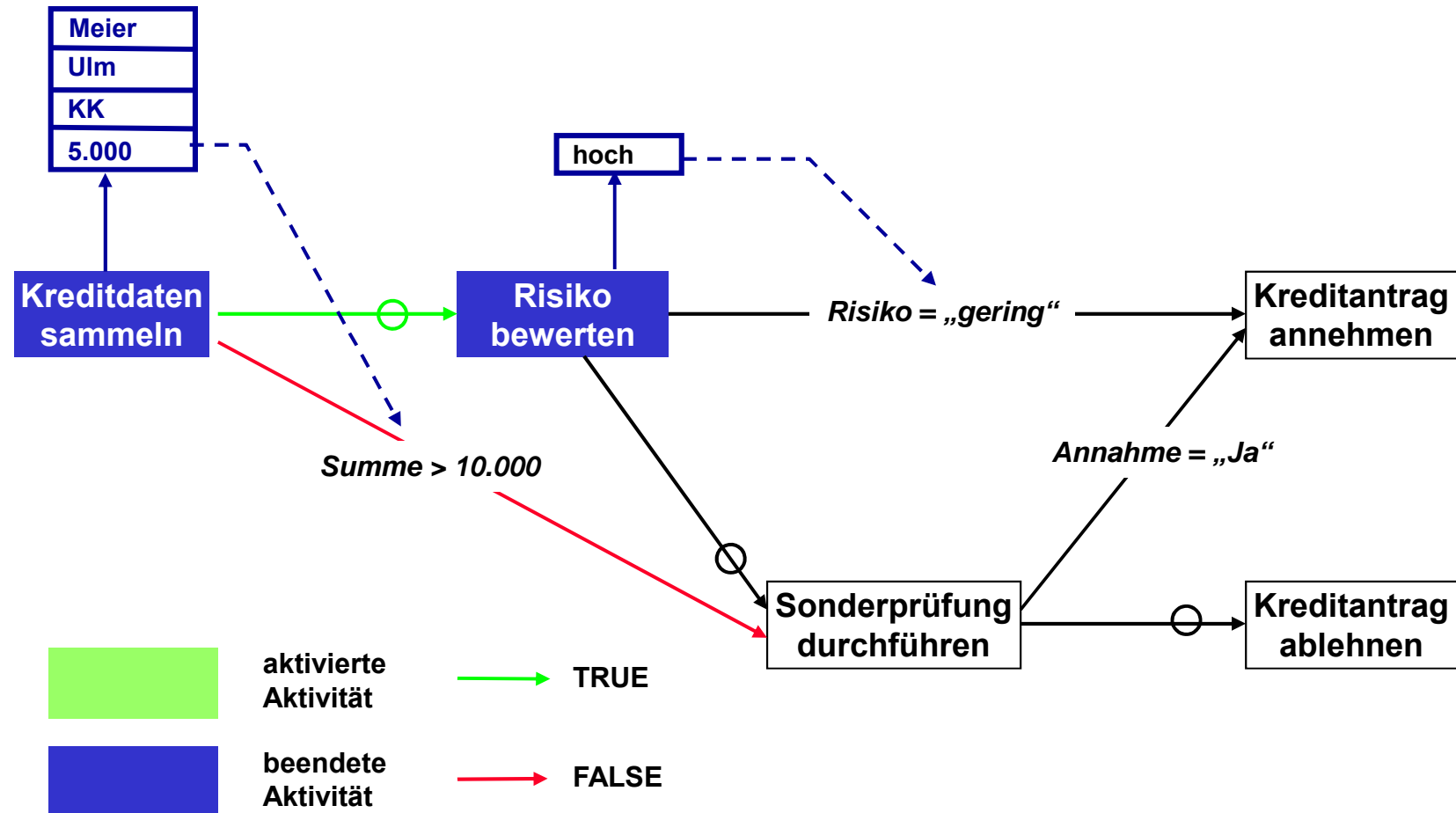


## 2.5 Aktivitätennetze

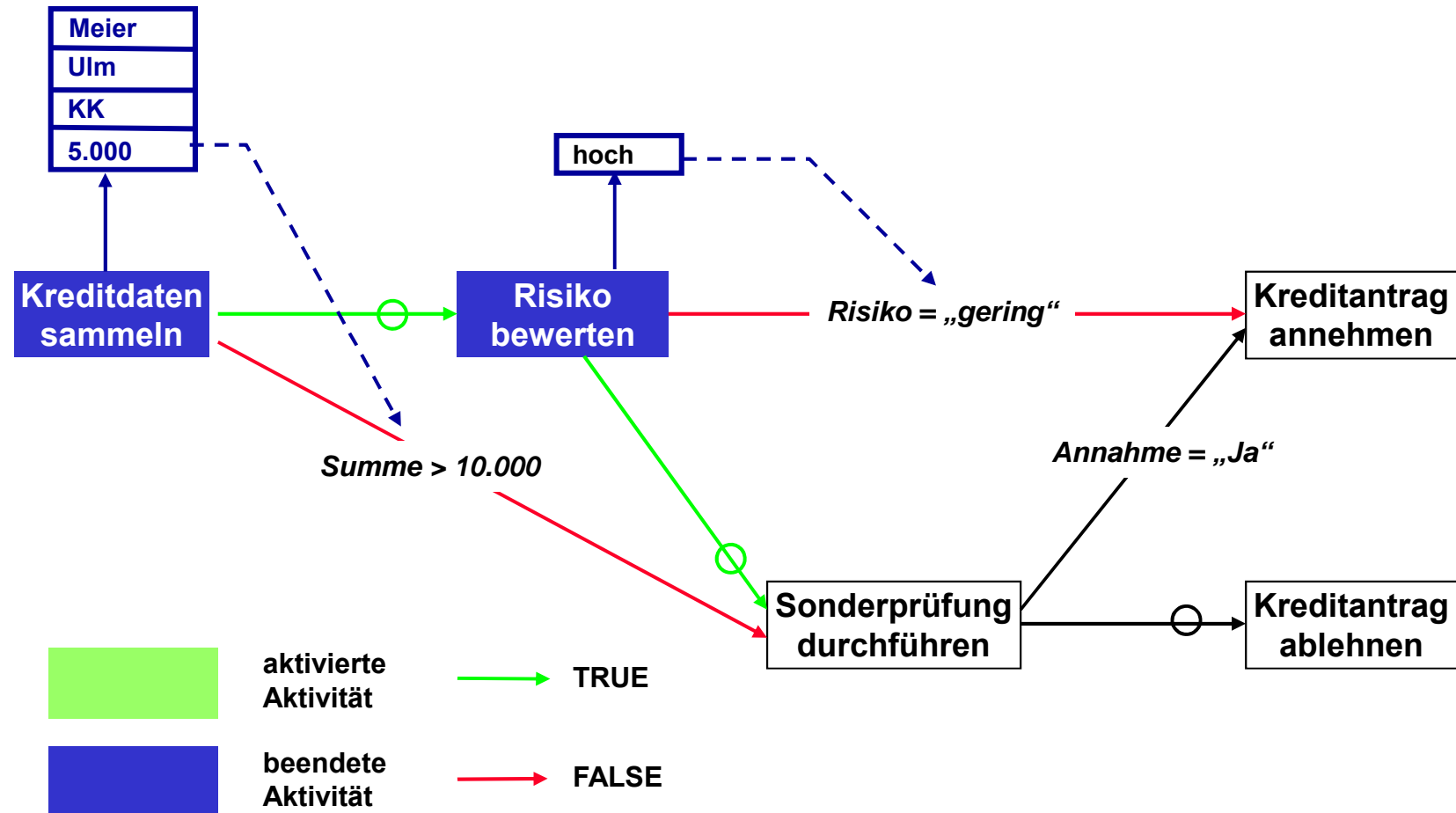




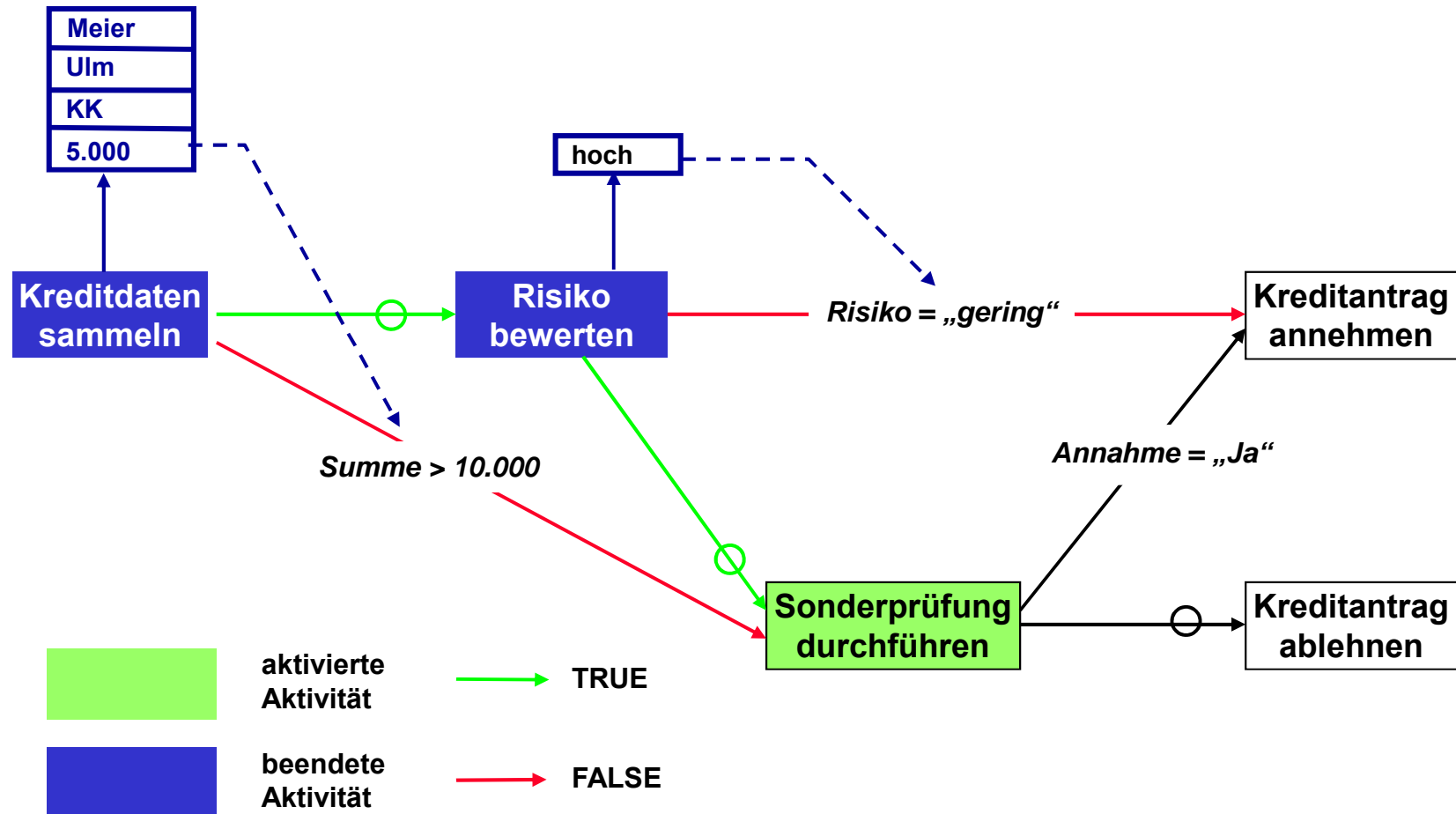
## 2.5 Aktivitätennetze



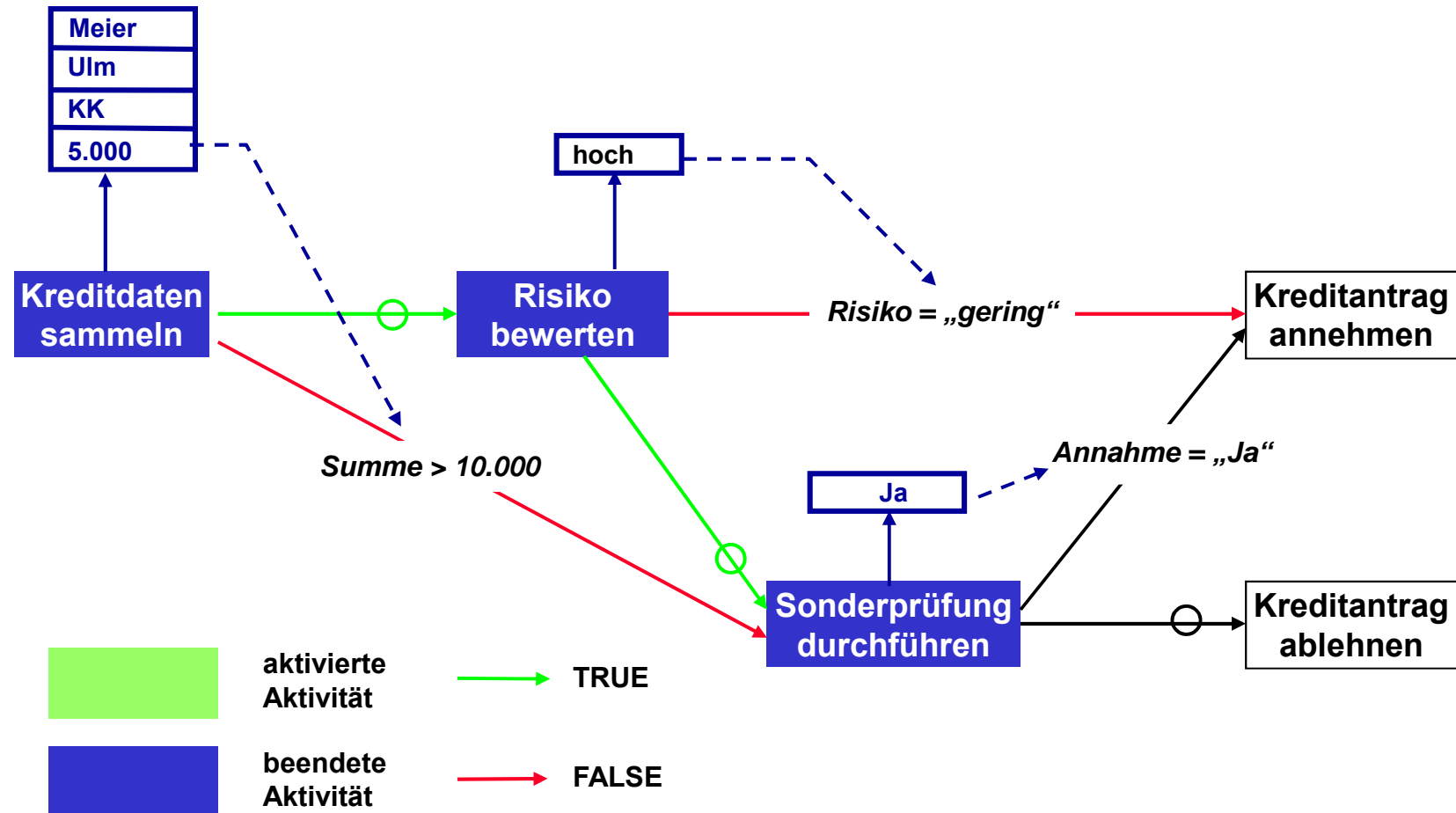
## 2.5 Aktivitätennetze



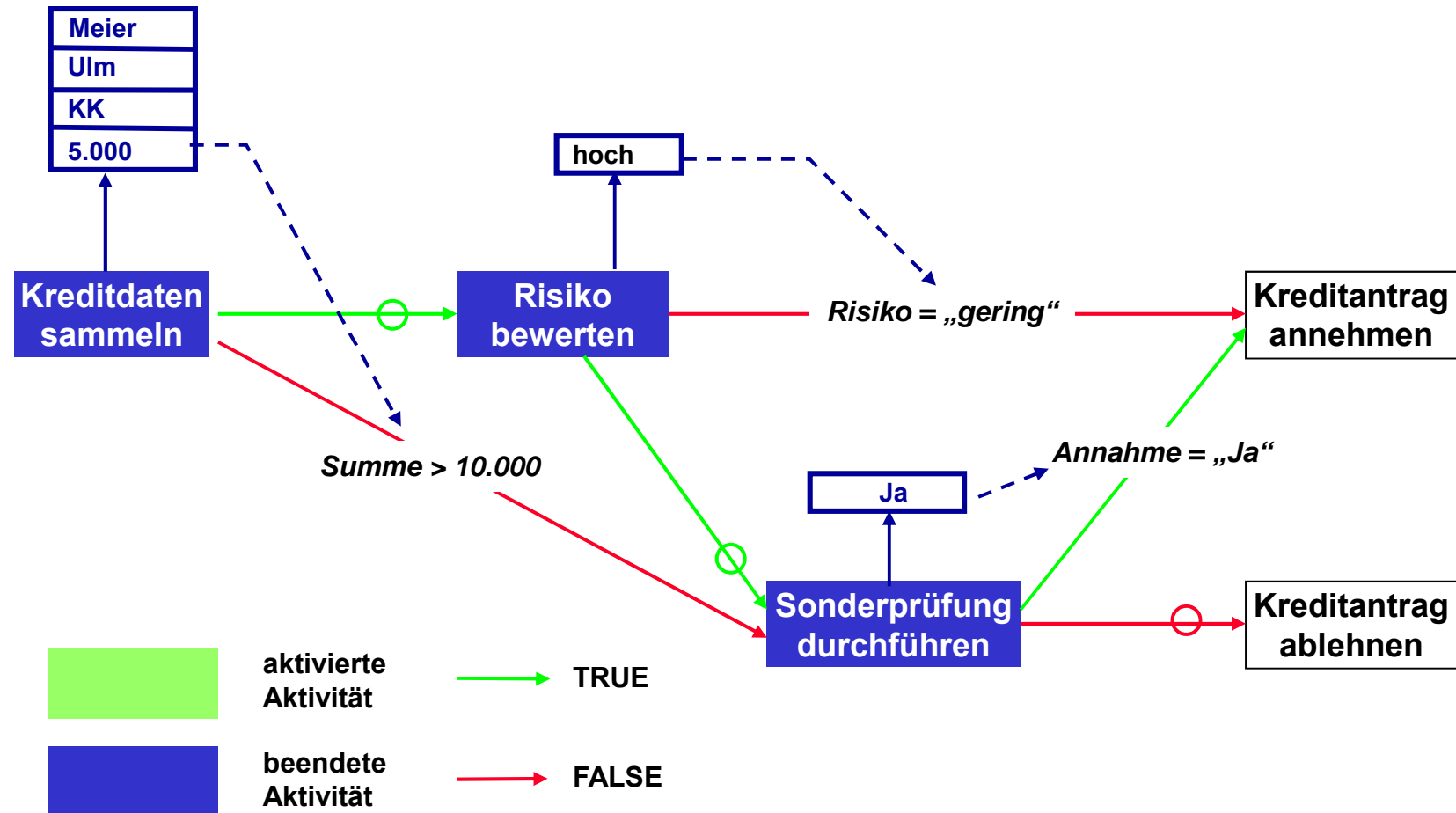
## 2.5 Aktivitätennetze



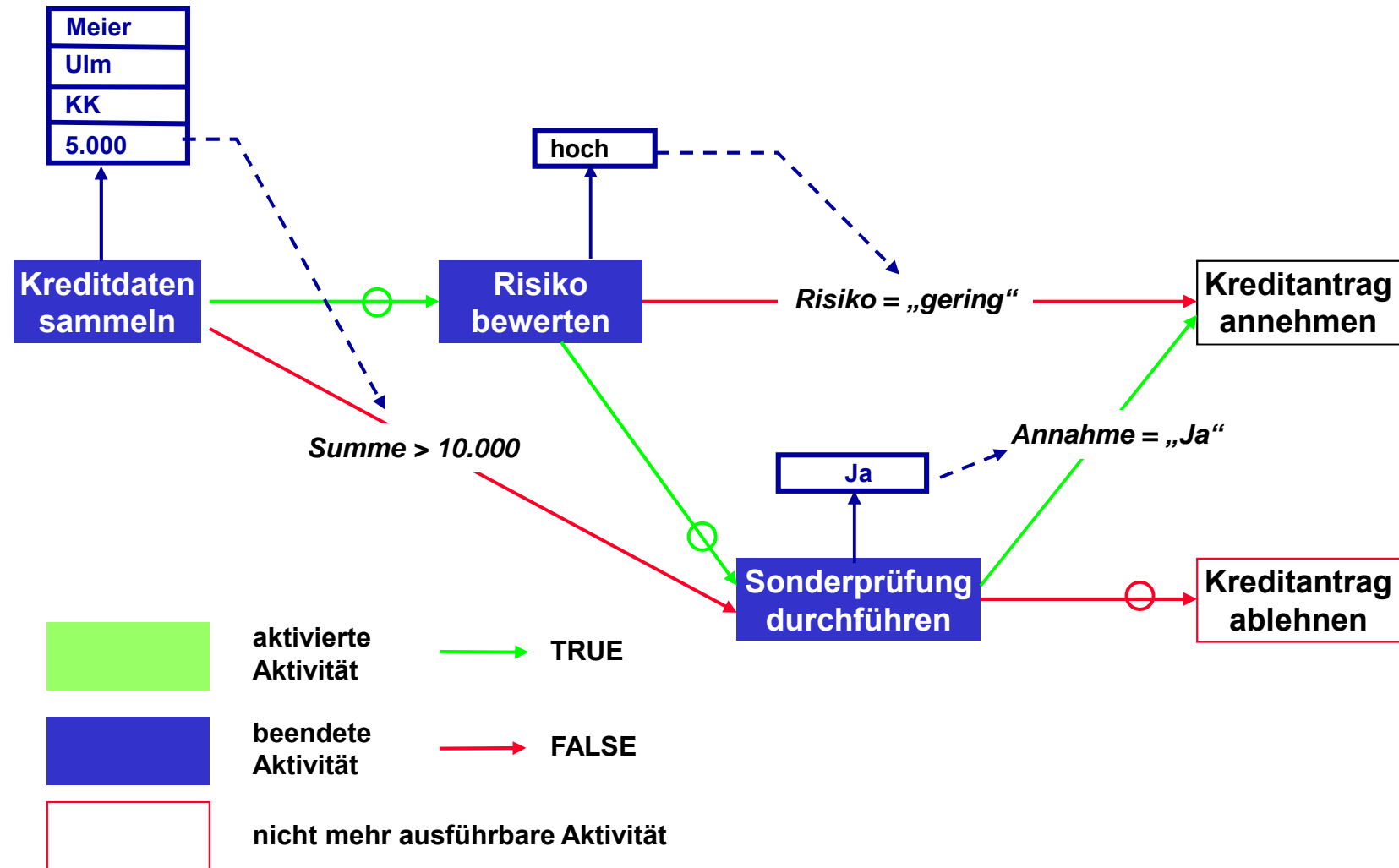
## 2.5 Aktivitätennetze



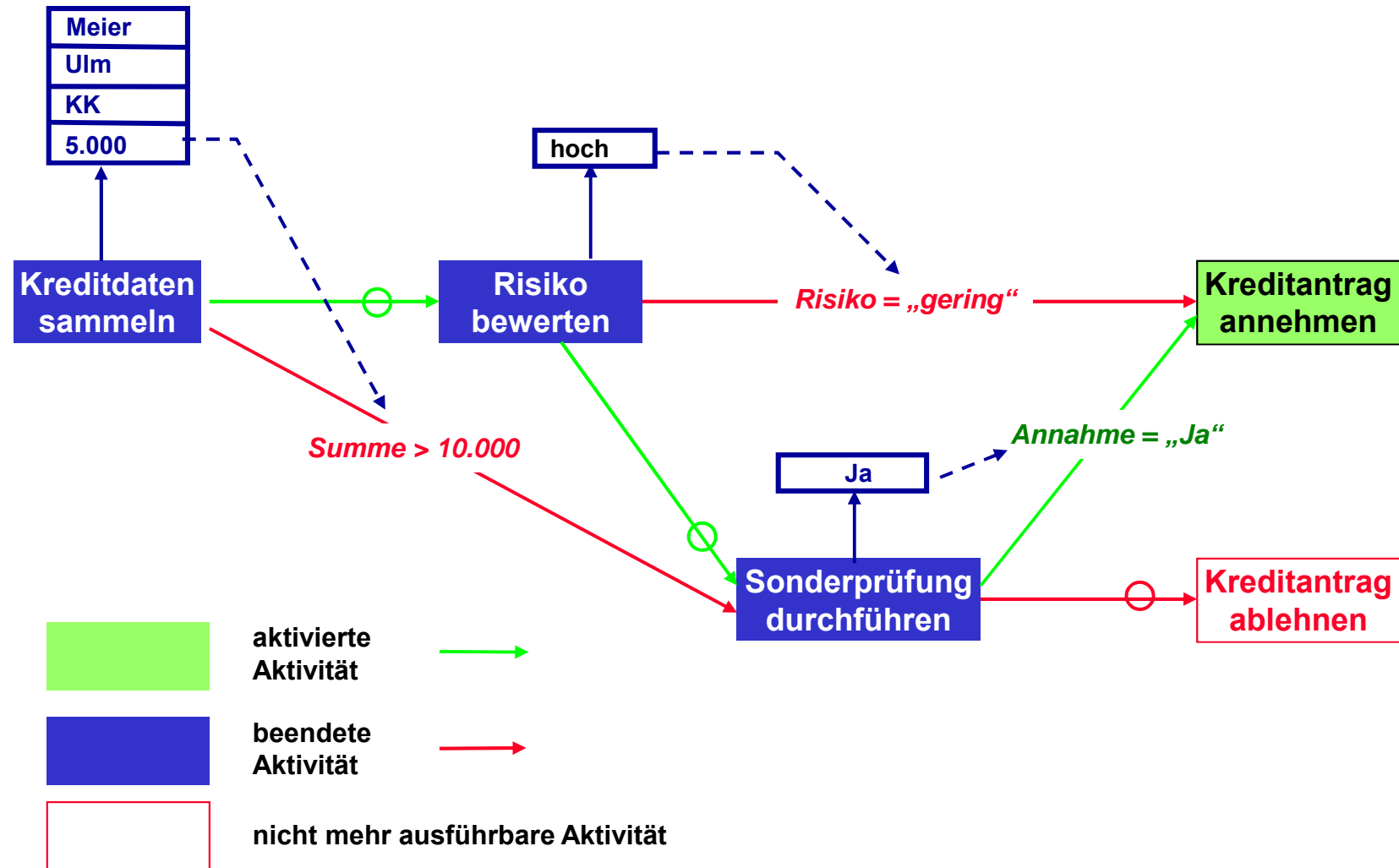
## 2.5 Aktivitätennetze



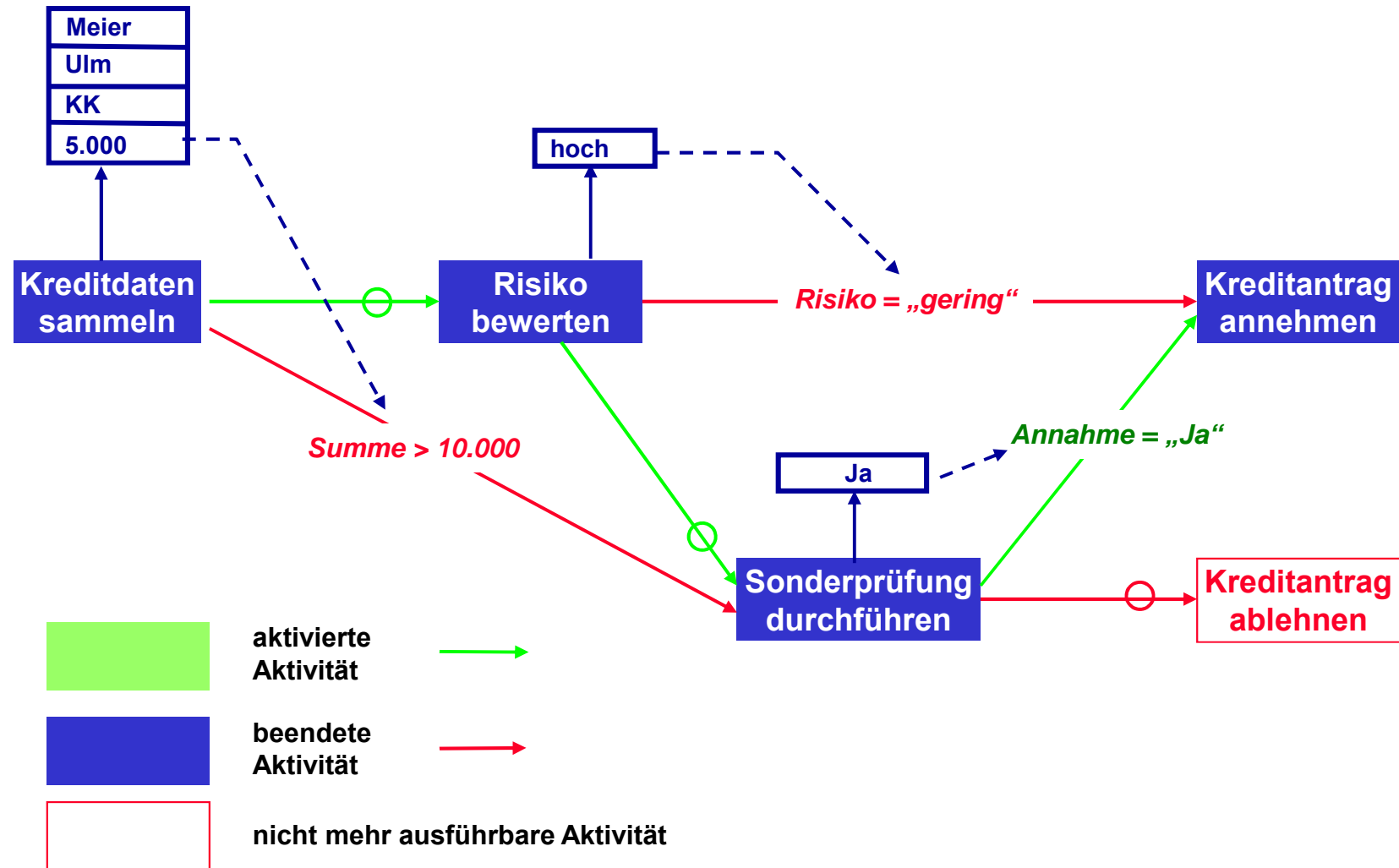
## 2.5 Aktivitätennetze



## 2.5 Aktivitätennetze



## 2.5 Aktivitätennetze





## 2.5 Aktivitätennetze

---

### □ Bewertung

#### ○ Vorbemerkungen

- Aktivitätennetze wurden für die Modellierung ausführbarer Prozesse entwickelt und weisen deshalb noch weitere Merkmale, wie z.B. Mitarbeiterzuordnung; wir haben uns hier (zunächst einmal) nur die Kontroll- und Datenflussmodellierung beschränkt

#### ○ Positiv

- Explizite (und bei Bedarf) detaillierte Modellierung des Datenflusses möglich
- Kompaktere Darstellung des Prozessmodells im Vergleich zu Petrinetzen

#### ○ Negativ

- Keine Schleifen (Loops) im Modell darstellbar
- Im Allg. nicht statisch entscheidbar, wie sich eine Verzweigung mit Transitionsbedingungen zur Laufzeit verhält (XOR, parallel, gemischt)
- Damit
  - ◆ hohes Potenzial für Modellierungsfehler
  - ◆ nur stark eingeschränkte Möglichkeiten, die Korrektheit von Datenflüssen zu prüfen

# Inhalt

---

## **2.0 Vorbemerkungen**

## **2.1 Korrektheitsaspekte von Prozessmodellen – eine Einführung**

## **2.2 Klassische Petri-Netze**

## **2.3 Höhere Petri-Netze**

## **2.4 Workflow-Netze**

## **2.5 Aktivitätennetze**

## **2.6 AristaFlow-Prozessmodell**

## **2.7 Andere Ansätze**

## **2.8 Ausdrucksmächtigkeit von Prozessbeschreibungssprachen**

## **2.9 Abschließende Bemerkungen**

## **2.10 Weiterführende Literatur**

## 2.6 AristaFlow-Prozessmodell

---

### □ Hintergrund

- Das AristaFlow-Prozessmodell basiert auf dem **ADEPT-Prozessmodell**, das im Kontext des ADEPT-Forschungsprojektes in den Jahren 1995 bis 1998 vom Institut DBIS entwickelt wurde

### ○ Entwicklungsziele

- Klares, einfach zu verstehendes und trotzdem mächtiges Prozessmodell
- **Universell einsetzbar** für möglichst alle Arten von Anwendungen, insbesondere für die Integration von Anwendungsfunktionen verschiedenster Art
- **Umfangreiche Korrektheitsprüfungen** zur Modellierungszeit in Bezug auf Kontroll- und Datenflüsse
  - ◆ zur Beschleunigung der Implementierung
  - ◆ sowie zur Vermeidung von „bad surprises at run-time“
- **Unterstützung von Ad-hoc-Abweichungen** (auf Prozessinstanzebene) zur Laufzeit
  - ◆ mit „semantisch hohen“ Änderungsoperationen
  - ◆ mit Korrektheitsprüfungen; möglichst im gleichen Umfang wie zur Modellierungszeit
  - ◆ mit rascher Entscheidung, ob gewünschte Abweichung gewährt werden kann

mehr hierzu  
später

## 2.6 AristaFlow-Prozessmodell

---

### □ Hintergrund (Forts.)

#### ○ Resultat (ADEPT-Prozessmodell)

- Ein relativ puristisches Prozessmodell mit wenigen Symbolen und Konstrukten (wäre aber durch „syntactic sugaring“ einfach erweiterbar)
- Explizite und feingranulare Modellierung von Datenflüssen
- Saubere, formale Basis mit strikten Korrektheitseigenschaften (damit kann man z.B. Modell-Editoren bauen, die ein „Correctness by Construction“-Prinzip anwenden; mehr dazu später)
- Umsetzung in diversen experimentellen ADEPT-Prototypen (1998 ff.)
- Heute:
  - ◆ Basis-Prozessmodell des AristaFlow<sup>®</sup> Prozess-Management-Systems
  - ◆ AristaFlow-Prozessmodell teils eingeschränkter als ADEPT-Prozessmodell
  - ◆ enthält aber auch diverse Erweiterungen, wie z.B. variable Parallelität

- AristaFlow derzeit in Bezug auf **Korrektheitsprüfungen + Ad-hoc-Abweichungen** mit Abstand das (modernste und) mächtigste lauffähige PMS

## 2.6 AristaFlow-Prozessmodell

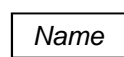
---

### □ Modell-Konstrukte für den Kontrollfluss

Start-/Endknoten



Aktivität



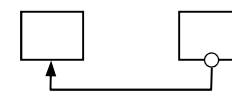
XOR-Split/-Join ♣



AND-Split/-Join



LOOP-Start/-Ende



Datenelement



Kontrollkonnektor



LeseKante



SchreibKante



Lese-Schreib-Kante



Sync-Kante



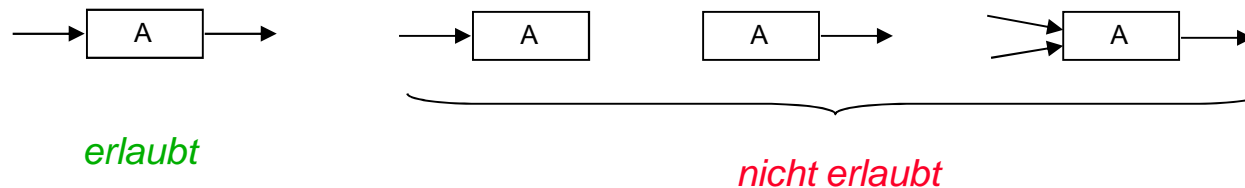
- 
- ♣ Das AristaFlow-Prozessmodell geht davon aus, dass dem **XOR-Split**-Knoten eine Entscheidungsfunktion zugeordnet ist, die aufgrund der Eingabeparameter entscheidet, welche Alternative gewählt werden soll. Die vom XOR-Split-Knoten ausgehenden Kanten können zur besseren Lesbarkeit entsprechend beschriftet werden.

## 2.6 AristaFlow-Prozessmodell

---

### □ Strukturregeln für den Kontrollfluss (Auswahl) ♣

- Jedes AristaFlow-Prozessmodell hat genau je einen START- und END-Knoten; der START-Knoten hat keinen einmündenden und ein END-Knoten keinen ausgehenden Kontrollkonnektor.
- Alle gewöhnlichen Aktivitäten-Knoten haben genau einen eingehenden und genau einen ausgehenden Kontrollkonnektor.



- Für die Modellierung von XOR-, AND- und LOOP-Strukturen, werden spezielle (Struktur-)Aktivitätenknoten verwendet.

♣ Eine vollständige Beschreibung der Struktur- und Ausführungsregeln des ADEPT-Prozessmodells findet sich in: Reichert, M.: Dynamische Ablaufänderungen in Workflow-Management-Systemen. Dissertation, Universität Ulm, Fakultät für Informatik, Mai 2000

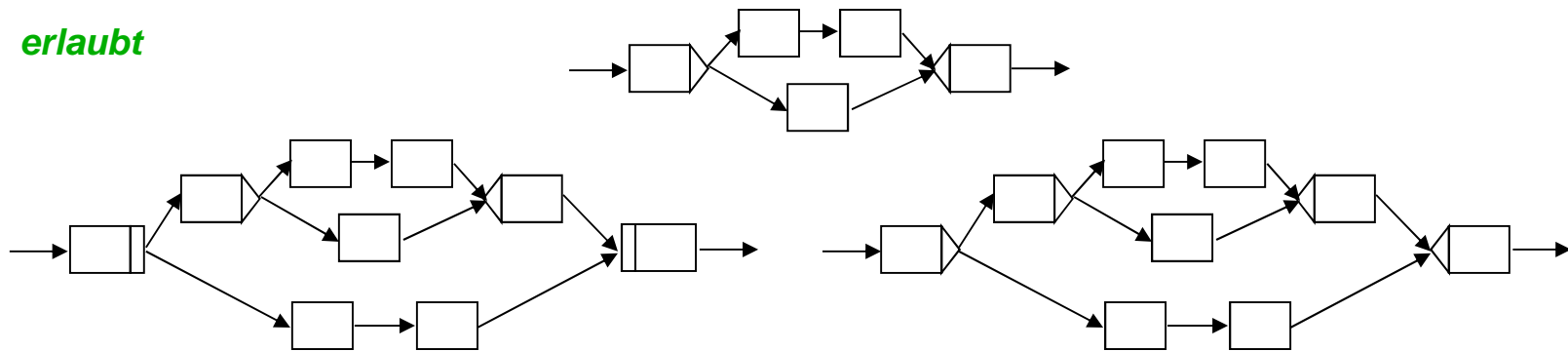
Eine etwas kompaktere Beschreibung findet sich in: Reichert, Manfred and Dadam, Peter: ADEPT<sub>flex</sub>-Supporting Dynamic Changes of Workflows Without Losing Control. Kluwer, Journal of Intelligent Information Systems, Special Issue on Workflow Management Systems, Vol. 10, No. 2, 1998, pp. 93-129 (Download via DBIS-Webseite)

## 2.6 AristaFlow-Prozessmodell

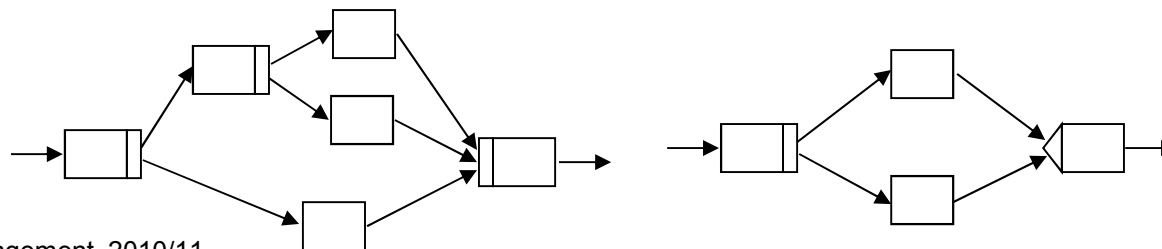
---

- Jedem **XOR-Split**- ist eineindeutig ein **XOR-Join**-Knoten zugeordnet.
- Jedem **AND-Split**- ist eineindeutig ein **AND-Join**-Knoten zugeordnet.
- Jedem **LOOP-Start**- ist eineindeutig ein **LOOP-End**-Knoten zugeordnet.
- Treten XOR-, AND- oder LOOP-Konstrukte verschachtelt auf, dann müssen diese sauber in einander verschachtelt sein.

*erlaubt*



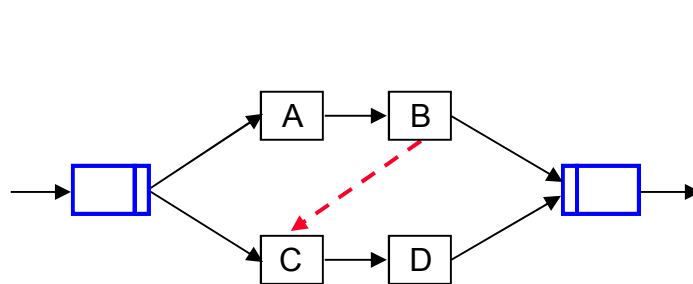
*nicht erlaubt*



## 2.6 AristaFlow-Prozessmodell

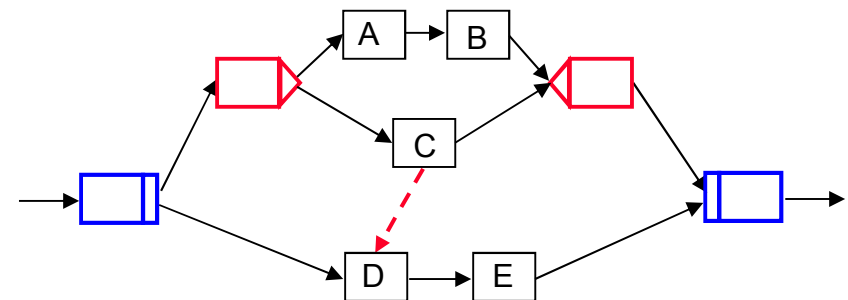
- Alle Aktivitäten- und Strukturknoten liegen auf einem durch die Kontrollkonnektoren beschriebenen Pfad, der beim Startknoten beginnt und beim Endknoten endet.  
↑ es gibt keine isolierten Knoten und keine „Sackgassen“
- Zyklen sind nicht erlaubt, ausgenommen in Form des Loop-Konstrukts.
- **(Soft-)Sync-Kanten** sind spezielle Kontrollkonnektoren. Sie sind nur zwischen Aktivitäten erlaubt, die in verschiedenen AND-Zweigen liegen.

Wir erklären die Wirkungsweise der Sync-Kanten an Beispielen:



a)

Die Sync-Kante wirkt hier wie ein normaler Kontrollkonnektor. D.h. Aktivität C muss (auch) auf den Abschluss von B warten (**strikte Synchronisation**).

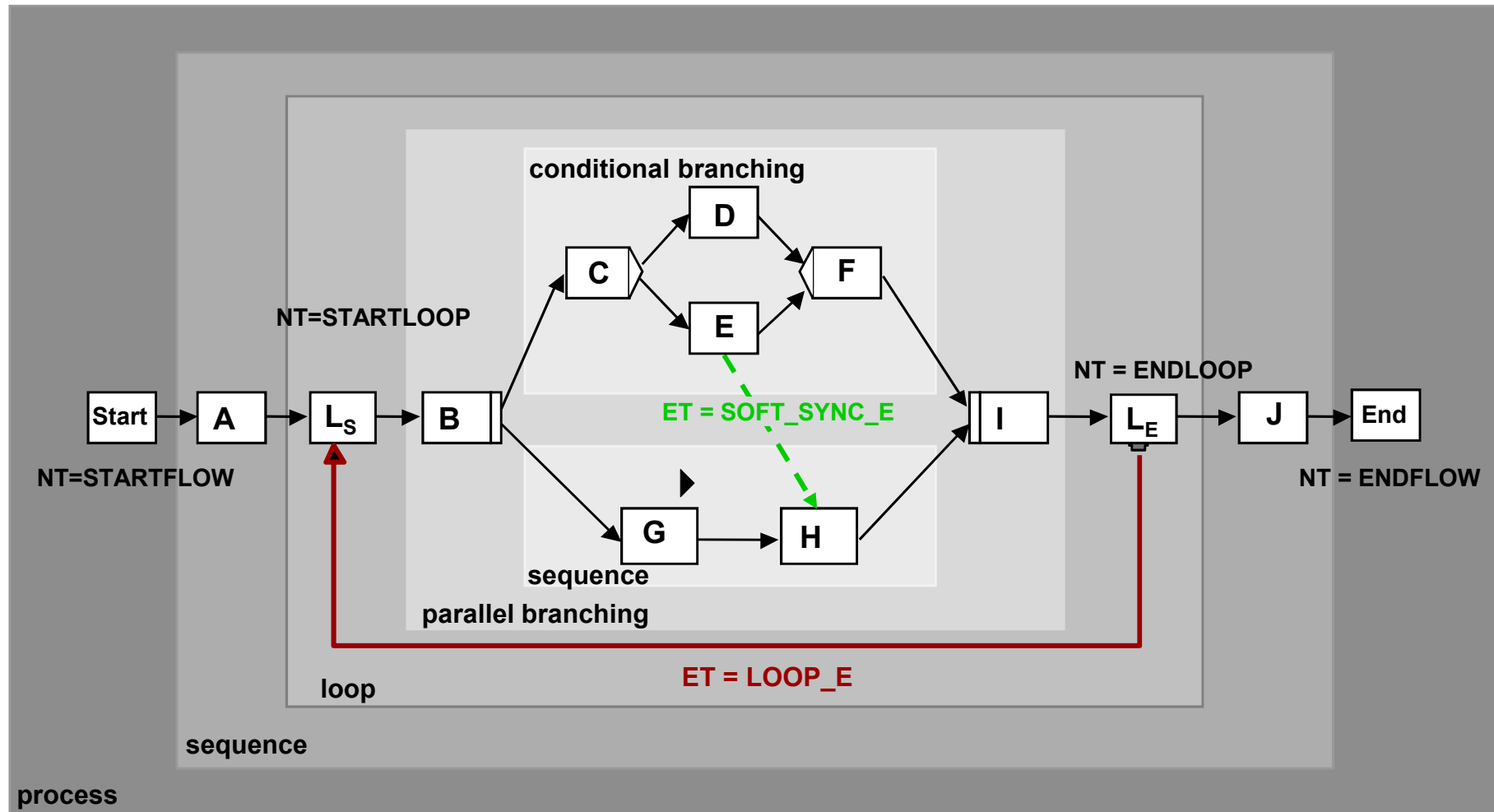


b)

**Bedingte Synchronisation.** Wird der Zweig, der Aktivität C enthält, „abgewählt“, wird die Sync-Kante wirkungslos; ansonsten muss D (auch) auf den Abschluss von C warten.  
(Ein Beispiel hierzu folgt etwas später.)



## 2.6 AristaFlow-Prozessmodell



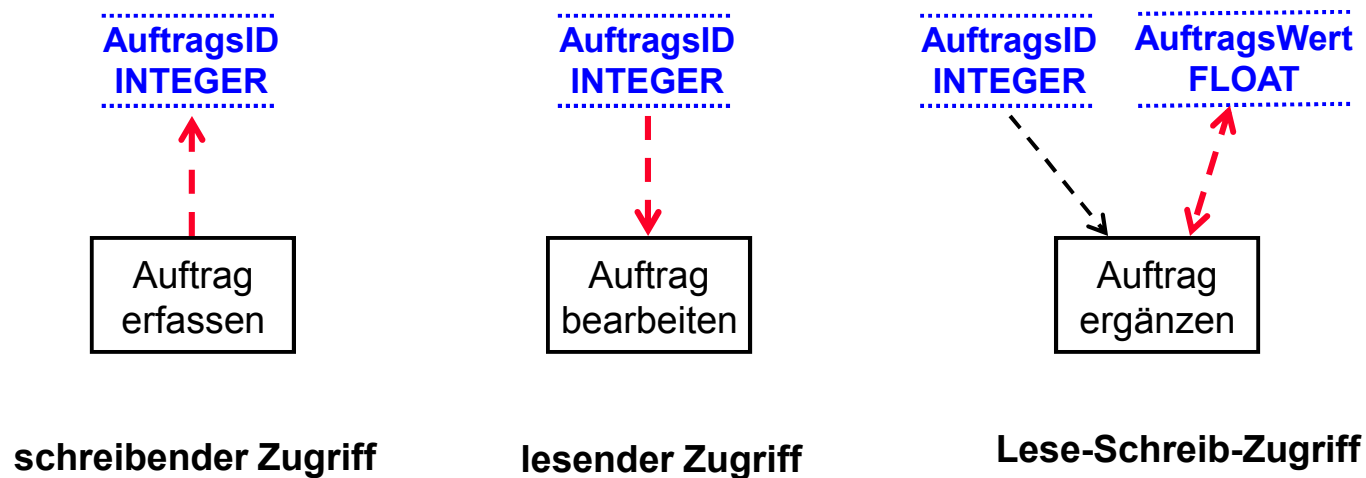
Strukturelemente des AristaFlow-Prozessmodells

## 2.6 AristaFlow-Prozessmodell

---

### □ Modellierung des Datenflusses

- Aktivitäten kommunizieren untereinander über (typisierte) **Datenelemente**, d.h. Prozessvariablen, mittels Schreiben und Lesen von Datenelementen
- **Schreibender Zugriff** : **Schreibkante** von der Aktivität zum Datenelement
- **Lesender Zugriff** : **LeseKante** vom Datenelement zur Aktivität
- **Lese-Schreib-Zugriff** : **Lese-Schreib-Kante** zwischen Aktivität und Datenelement



## 2.6 AristaFlow-Prozessmodell

---

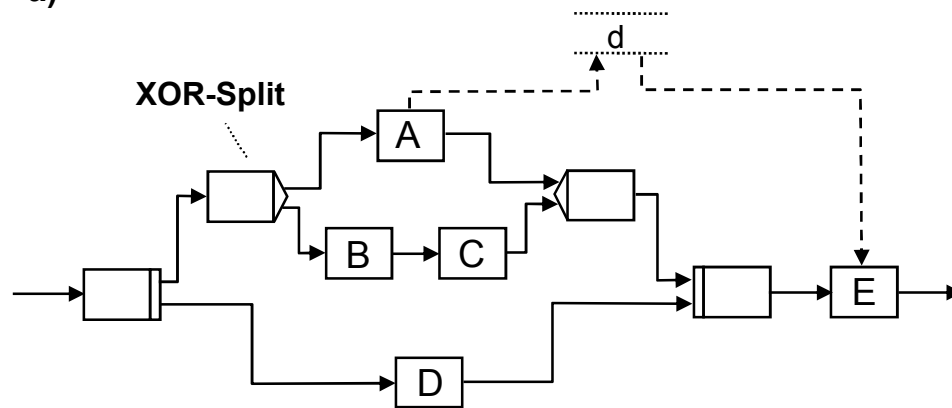
### □ Modellierung des Datenflusses (Forts.)

- Datenelemente können mit Aktivitätenknoten sowie mit XOR-Split und LOOP-End-Knoten verknüpft werden.
  - Lese-, Schreib- und Lese-Schreibkanten können als **nicht-optional** (der Default) oder **optional** deklariert werden.  
Sie repräsentieren nicht-optimale bzw. optionale Input- oder Output-Parameter der zugeordneten oder zuzuordnenden Anwendungsfunktionen (siehe später)
  - **Lesekanten** repräsentieren (optionale und nicht-optimale) **Eingabeparameter**, **Schreibkanten** (optionale und nicht-optimale) **Ausgabeparameter** des Prozessschrittes
- Loop-Start-, XOR-Join- sowie AND-Join-Knoten können nicht mit Datenelementen verknüpft werden.
- **Wichtige Strukturregel des AristaFlow-Prozessmodells:**  
**Nicht-optimale Eingabeparameter** von Prozessschritten müssen in allen möglichen Ausführungsreihenfolgen, die das Prozessmodell zulässt, stets vor der Aktivierung des jeweiligen Prozessschrittes mittels **nicht-optimalem Schreibzugriff** auf das zugehörige Datenelement „versorgt“ werden.

## 2.6 AristaFlow-Prozessmodell

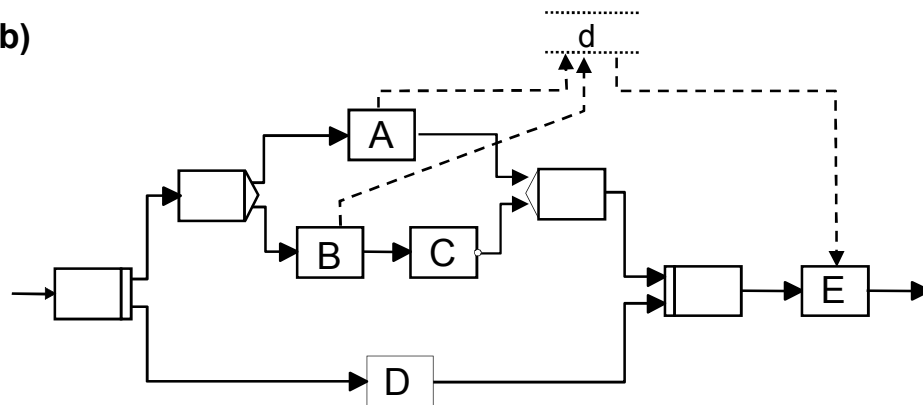
### Beispiel: Korrektheit von Datenflüssen

a)



Korrekt?

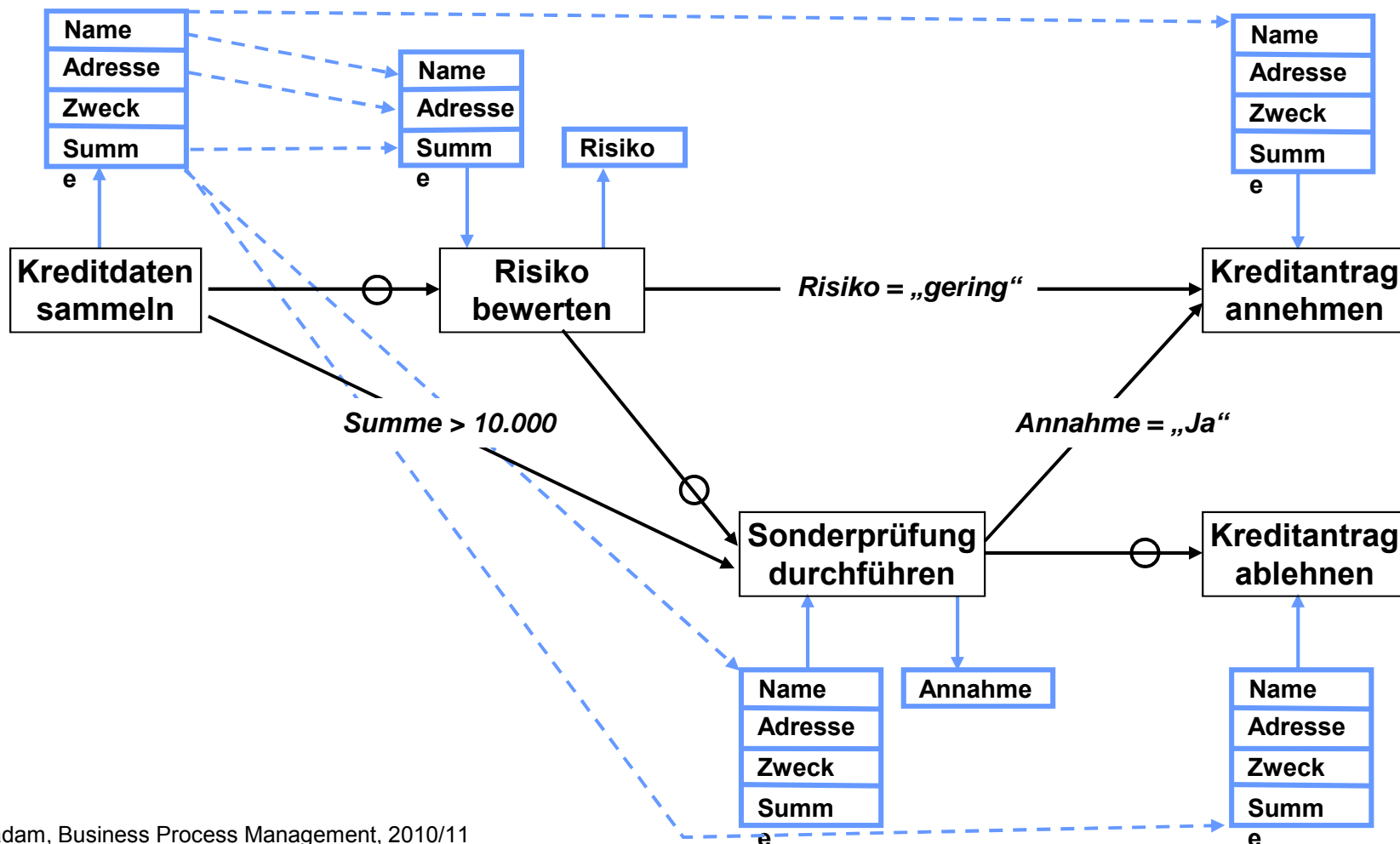
b)



Korrekt?

## 2.6 AristaFlow-Prozessmodell

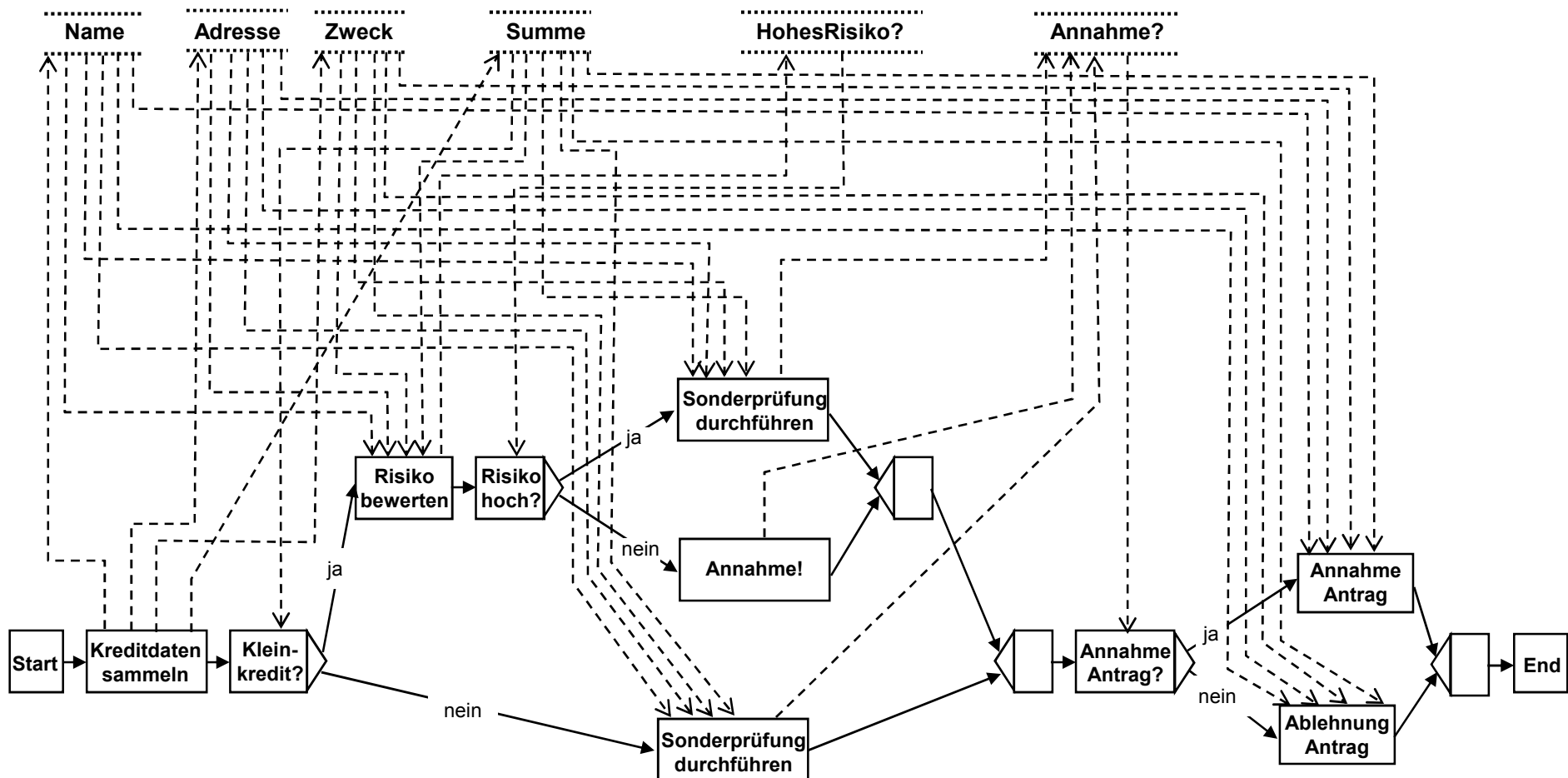
### □ Beispiel: Kreditantragstellung – Lösung mit Aktivitätennetz-Modell



## 2.6 AristaFlow-Prozessmodell

Wenn alle Datenkanten dargestellt sind, werden die Modelle rasch unübersichtlich. Der AristaFlow Process Template Editor unterstützt daher verschiedene „selektive“ Ansichten

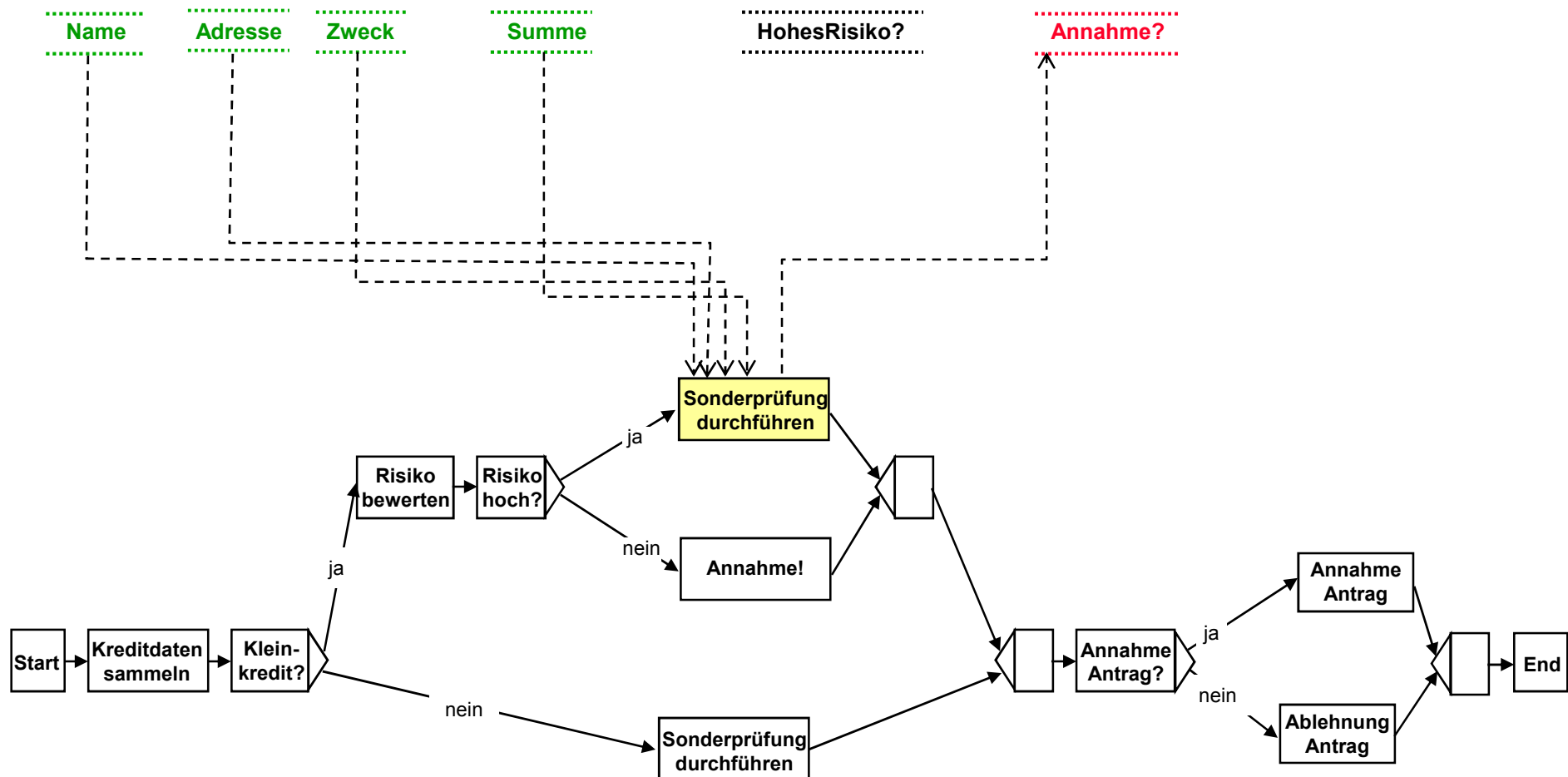
### □ Beispiel: Kreditantragstellung – Lösung mit AristaFlow-Prozessmodell



## 2.6 AristaFlow-Prozessmodell

Wenn alle Datenkanten dargestellt sind, werden die Modelle rasch unübersichtlich. Der AristaFlow Process Template Editor unterstützt daher verschiedene „selektive“ Ansichten

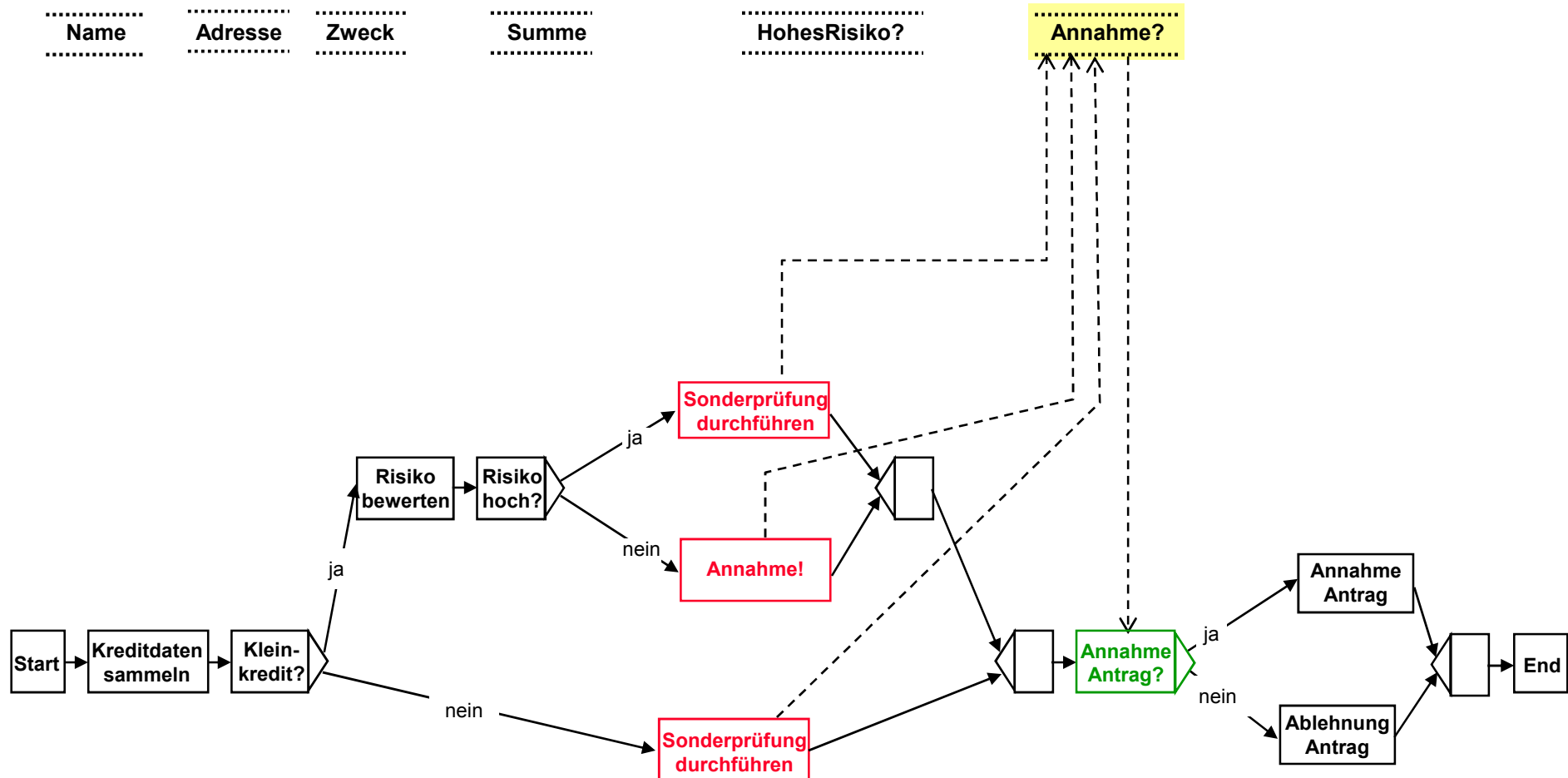
... wie etwa die aktivitätszentrierte Auswahl



## 2.6 AristaFlow-Prozessmodell

Wenn alle Datenkanten dargestellt sind, werden die Modelle rasch unübersichtlich. Der AristaFlow-Modelleditor unterstützt daher verschiedene „selektive“ Ansichten

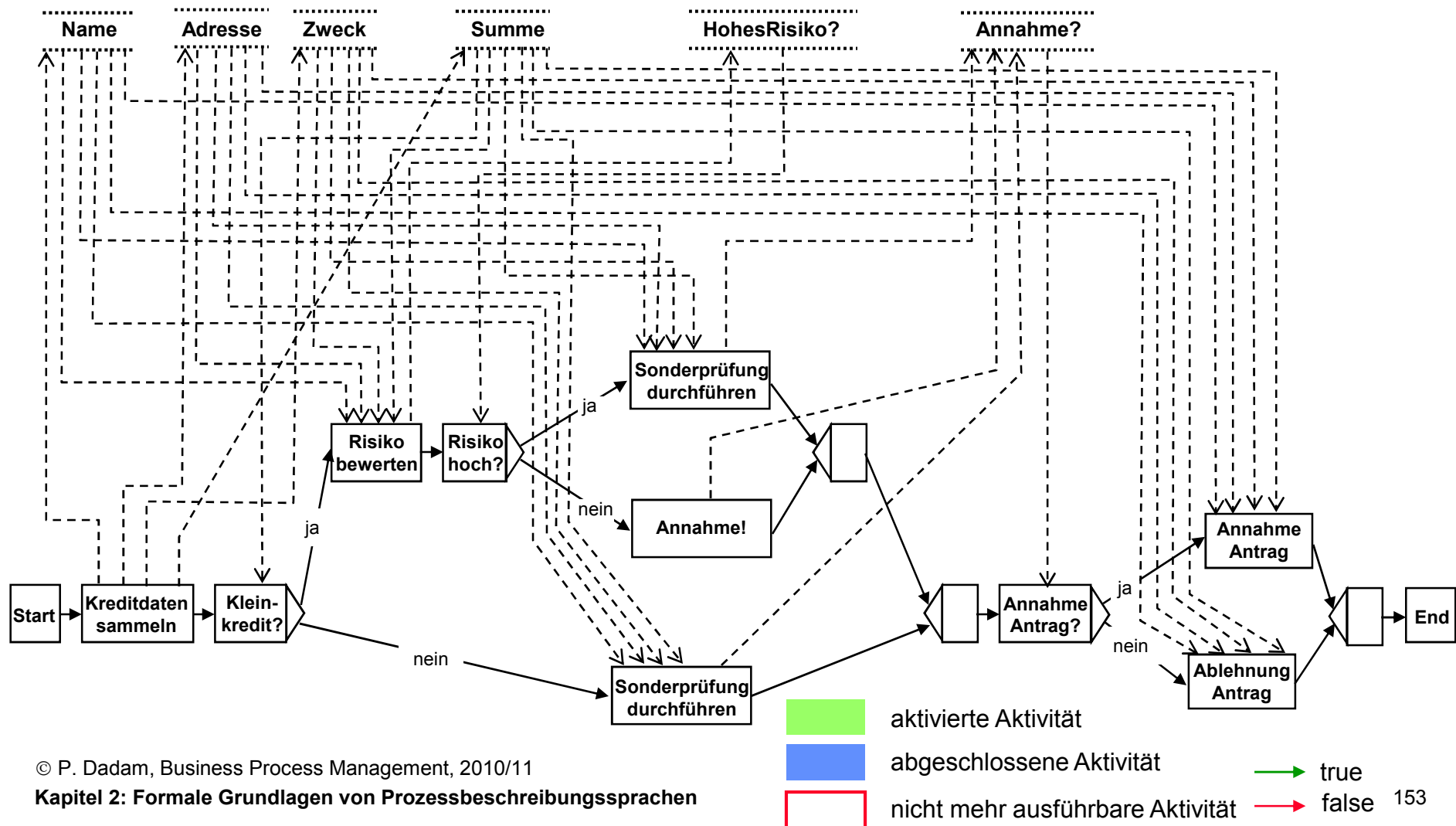
... und die datenelement-zentrierte Auswahl



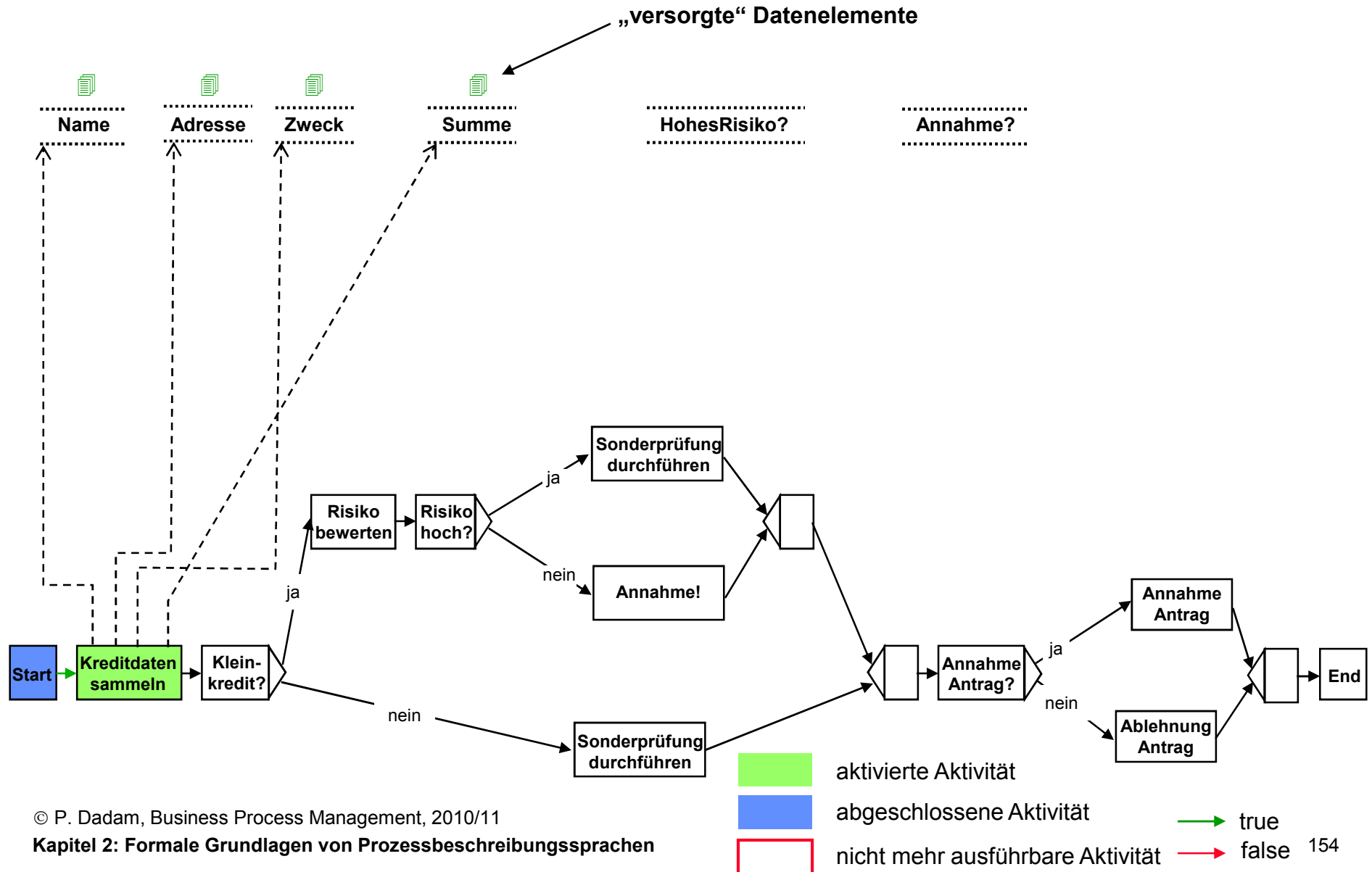


## 2.6 AristaFlow-Prozessmodell

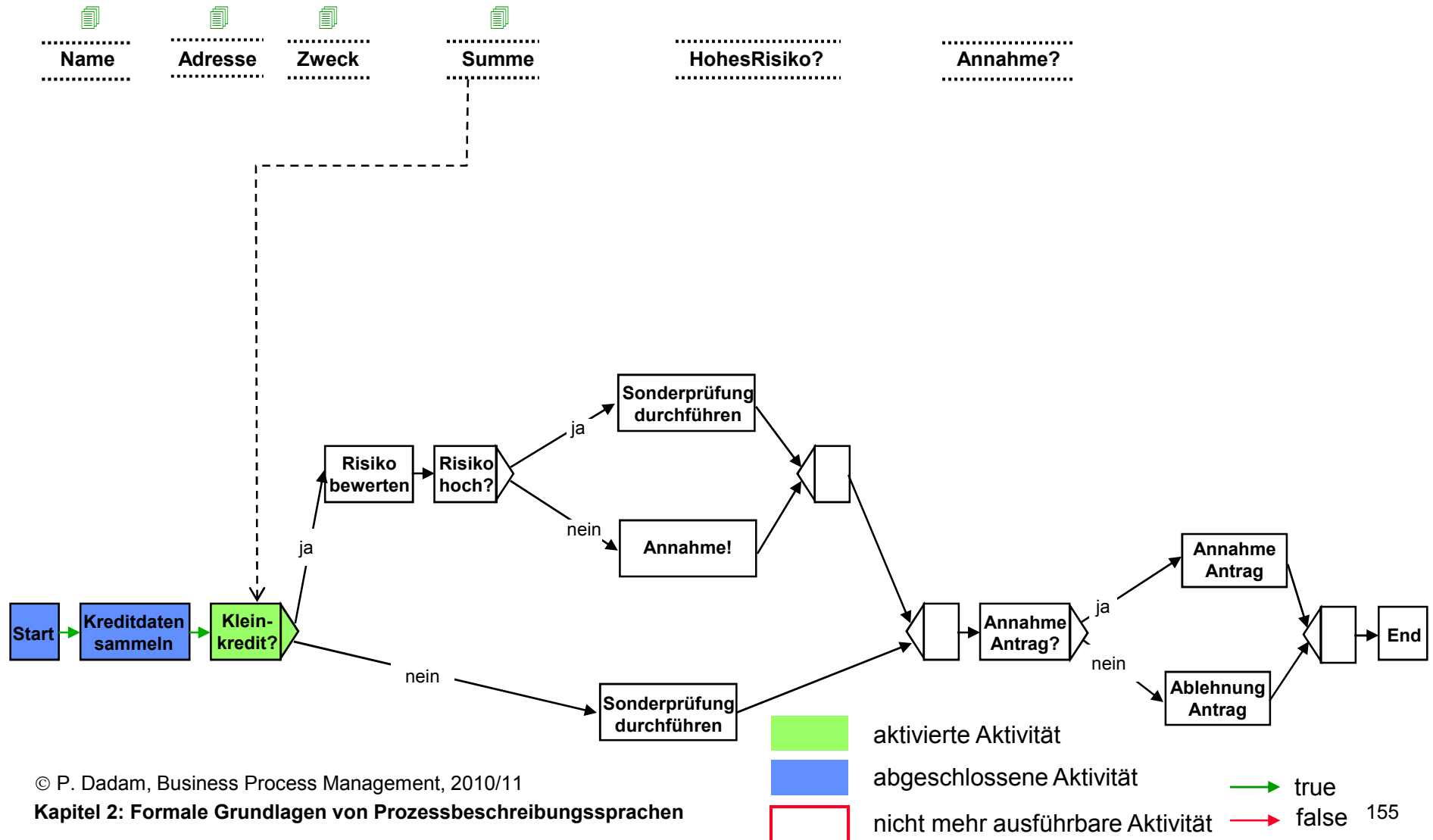
### □ Beispiel: Kreditantragstellung – **Ausführungsverhalten**



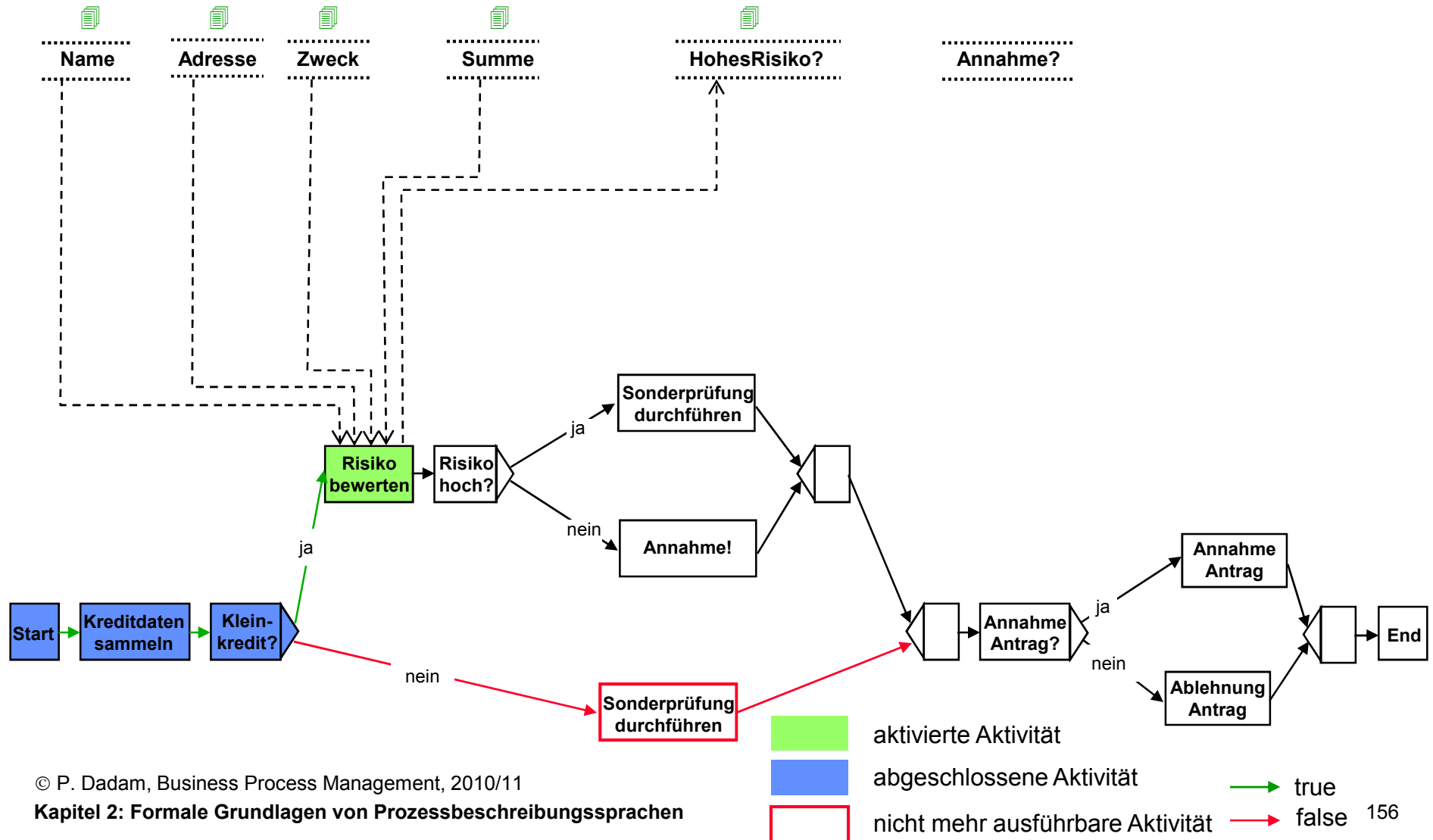
## 2.6 AristaFlow-Prozessmodell



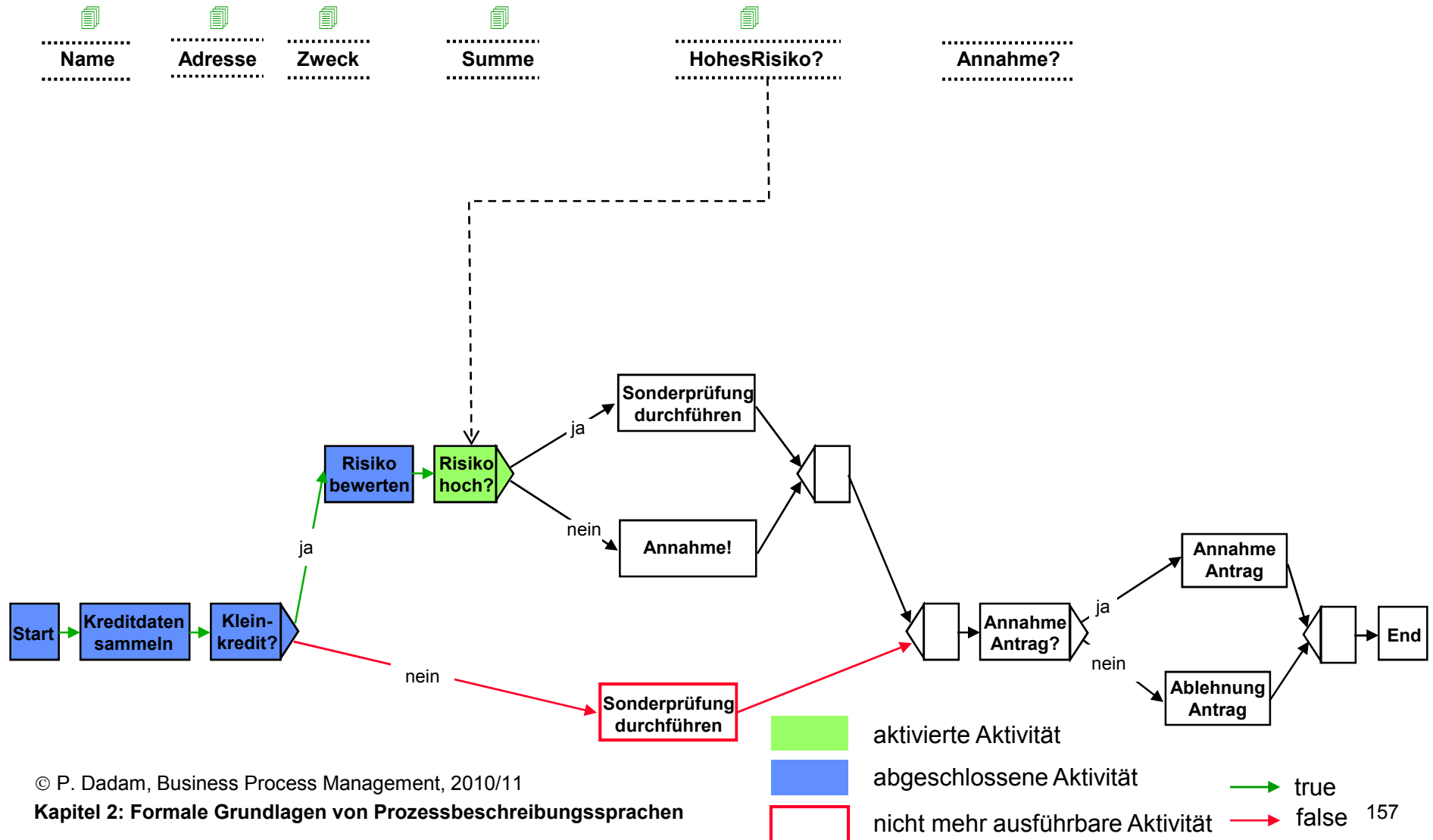
## 2.6 AristaFlow-Prozessmodell



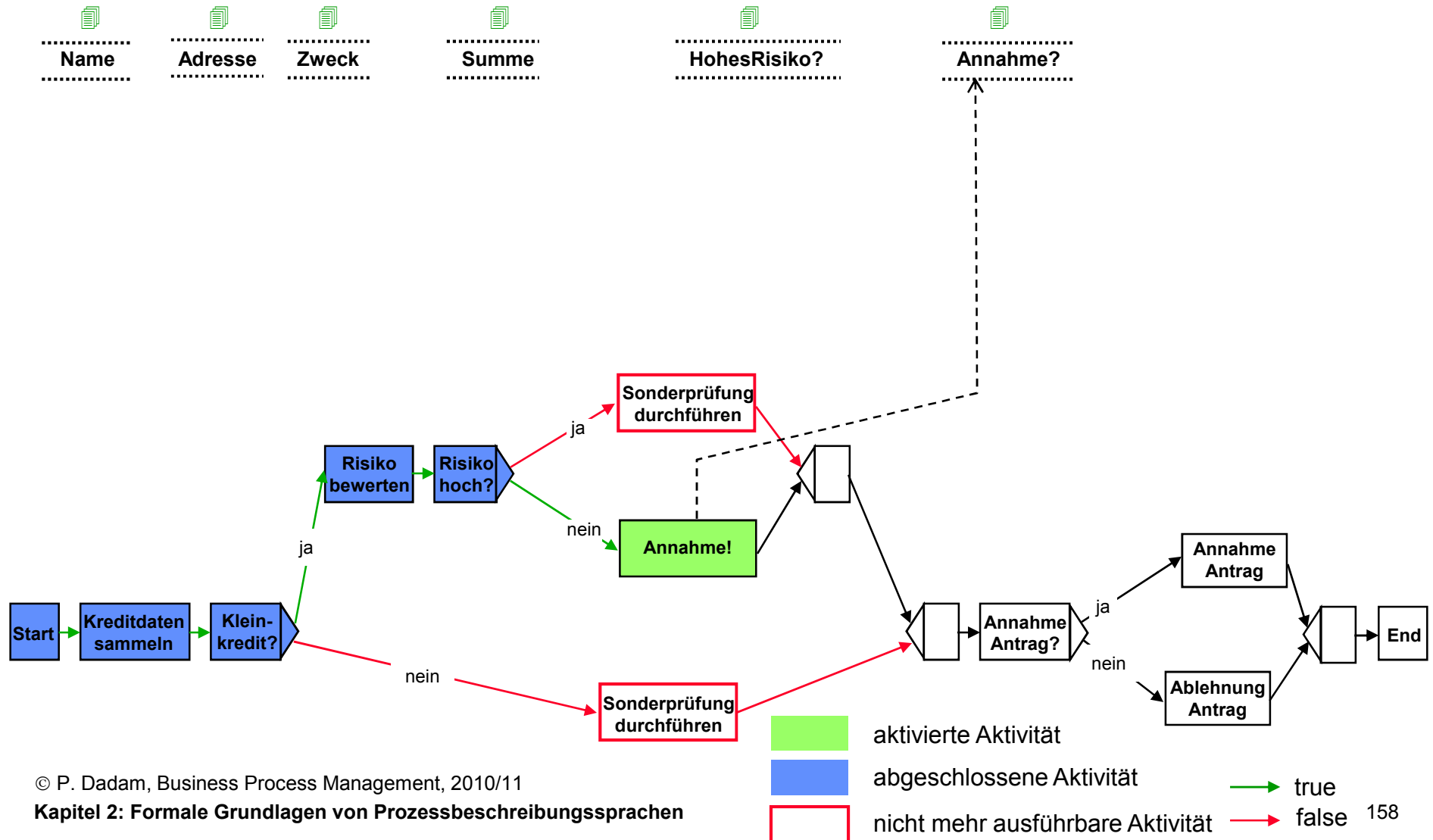
## 2.6 AristaFlow-Prozessmodell



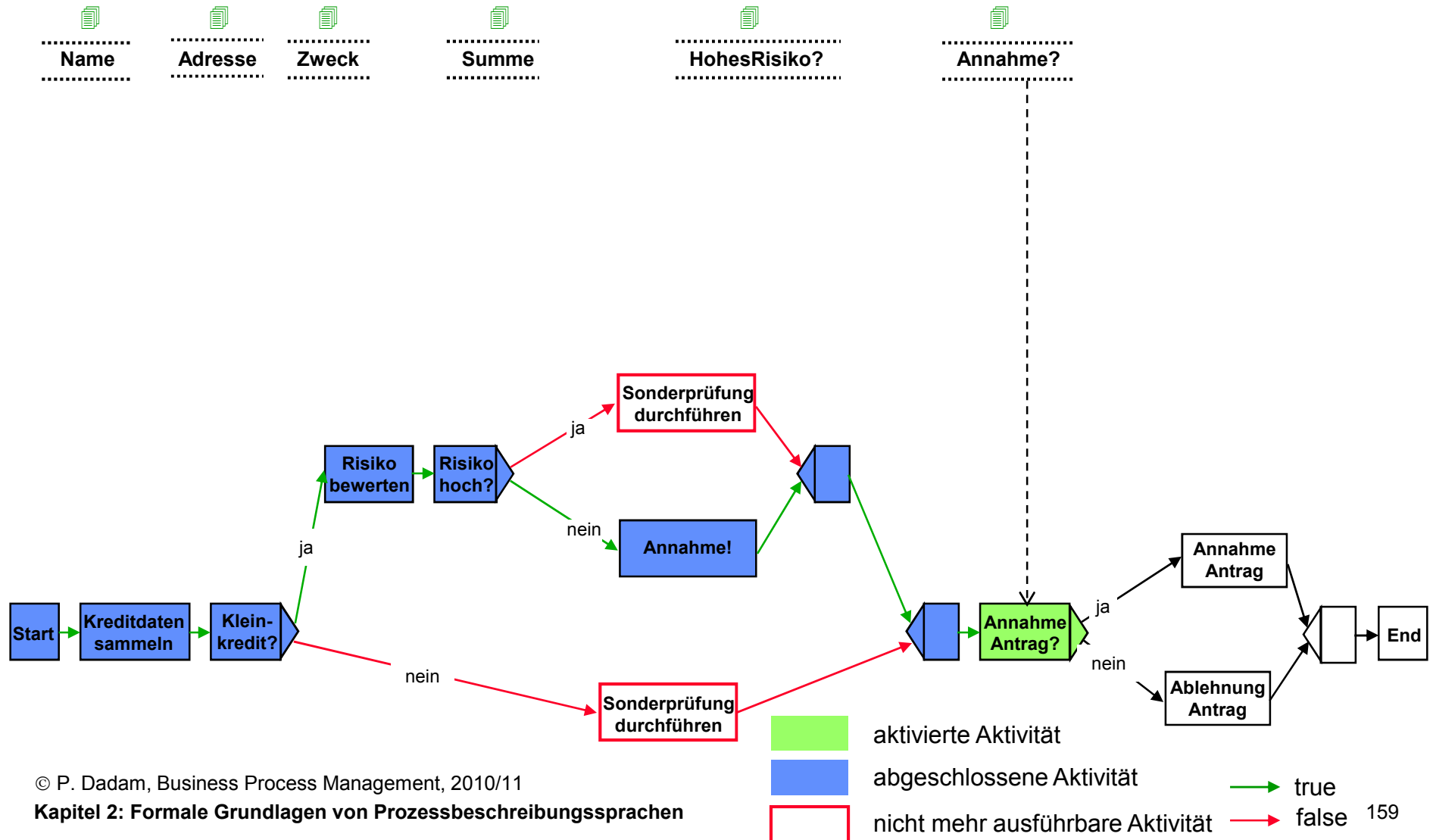
## 2.6 AristaFlow-Prozessmodell



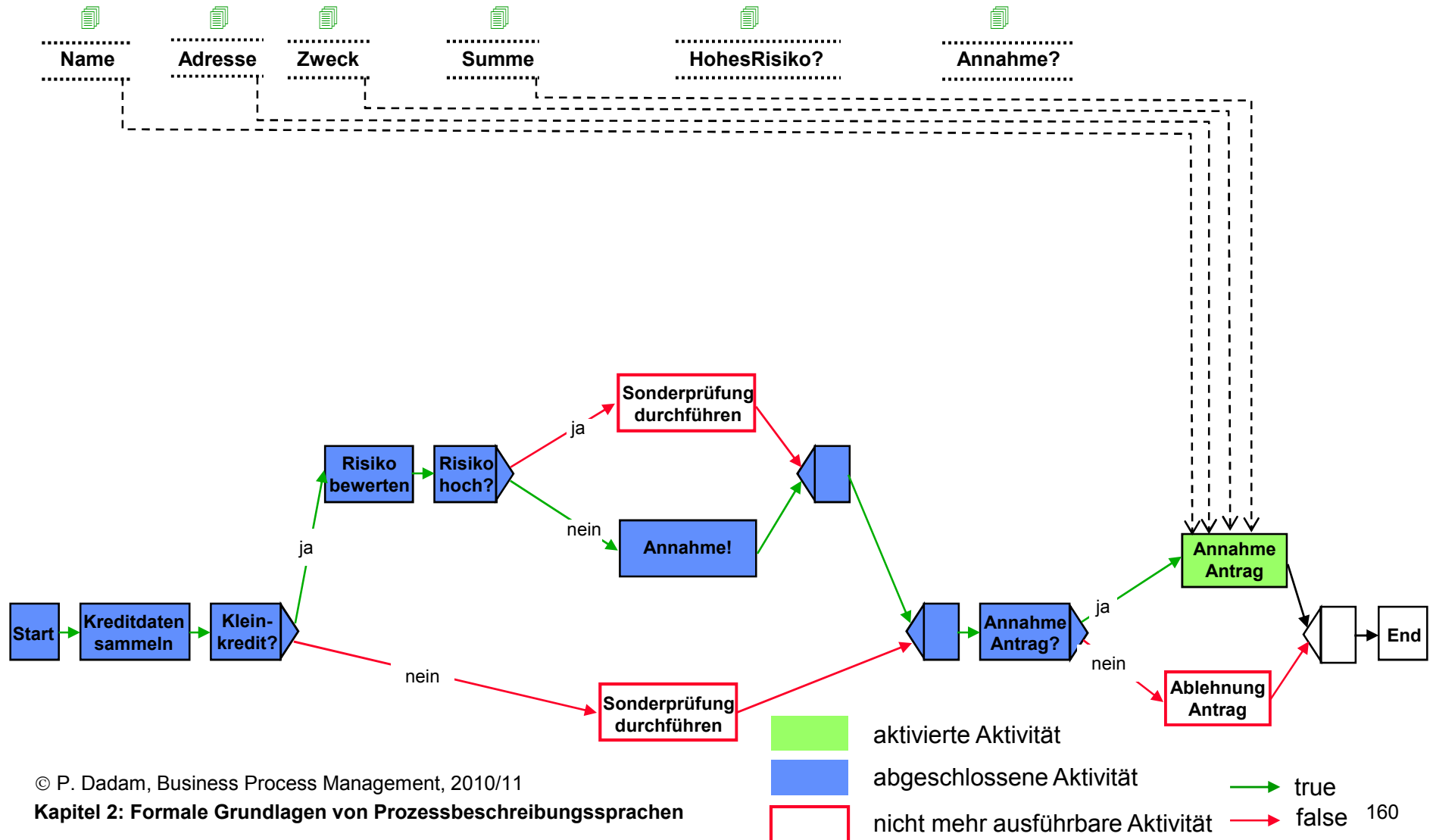
## 2.6 AristaFlow-Prozessmodell



## 2.6 AristaFlow-Prozessmodell



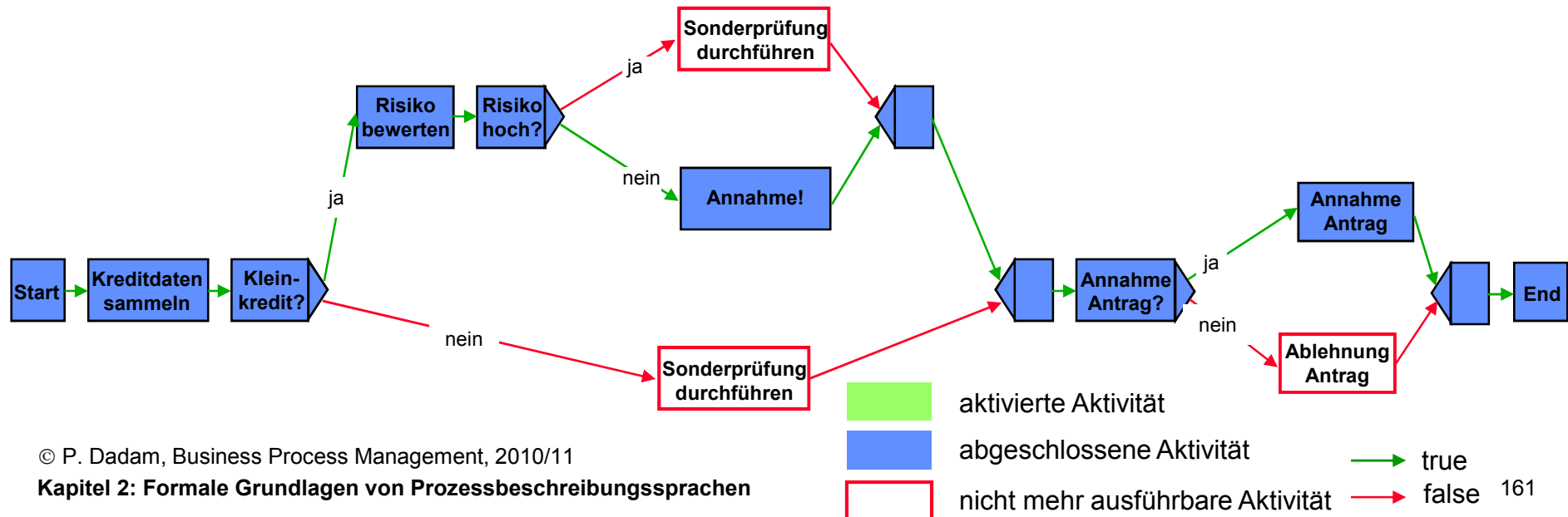
## 2.6 AristaFlow-Prozessmodell





## 2.6 AristaFlow-Prozessmodell

					
Name	Adresse	Zweck	Summe	HohesRisiko?	Annahme?



## 2.6 AristaFlow-Prozessmodell

---

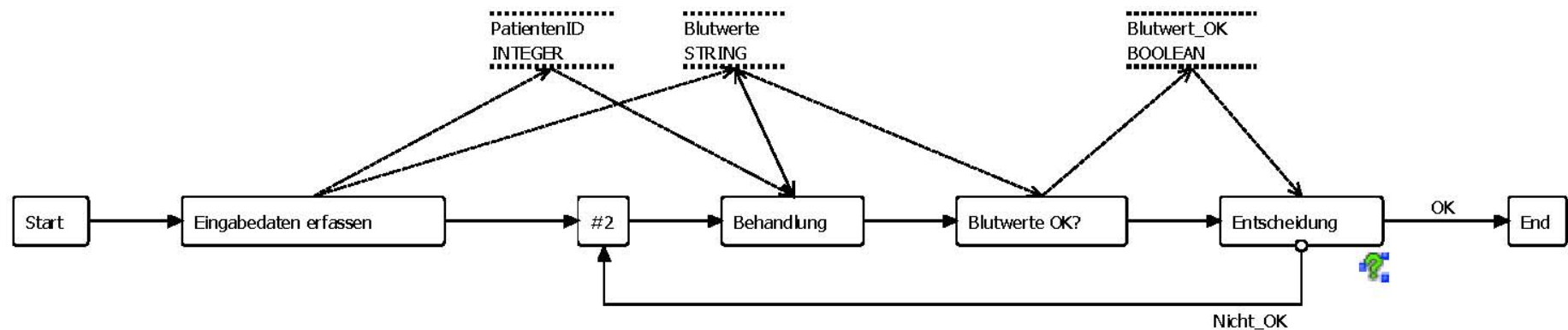
### □ Hierarchisierung von Prozessmodellen

- Im AristaFlow-Prozessmodell kann jede Aktivität eine elementare oder eine komplexe Aktivität (= Sub-Prozess) sein
- Sub-Prozesse sind vollständig gekapselt, d.h. es findet kein direkter Zugriff auf Datenelemente des Vater-Prozesses statt
- Aus Sicht der Prozessmodellierung verhalten sich Subprozesse im Wesentlichen wie gewöhnliche Aktivitäten mit Aufruf- und Rückgabe-Parametern
- Parameter-Übergabe
  - Die **Aufruf-Parameter** des Subprozesses werden vom **Startknoten** des Subprozess mittels Schreiboperationen auf Datenelemente in den lokalen Kontext eingebracht.
  - Die Datenelemente des Subprozesses, welche die Werte für die **Rückgabe-Parameter** enthalten, werden vom **Endknoten** des Subprozesses gelesen

## 2.6 AristaFlow-Prozessmodell

### ○ Beispiel

Gegeben sei der folgende Behandlungsprozess

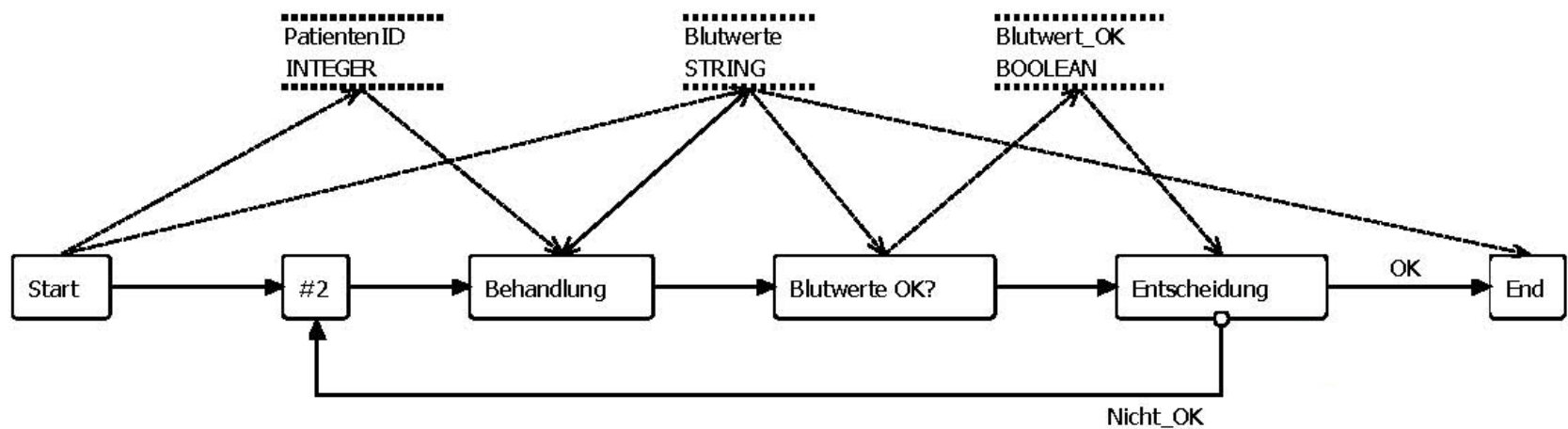


- Dieser Prozess soll als **Subprozess** realisiert werden, damit er in anderen Prozessen als (komplexe) Aktivität verwendet werden kann
- Der Prozessschritt „*Eingabedaten erfassen*“ soll im Subprozess entfallen
- stattdessen sollen dem Subprozess *PatientenID* und *Blutwerte* als Aufrufparameter übergeben werden
- Die (neuen) *Blutwerte* sollen als Rückgabeparameter an den Aufrufkontext zurückgegeben werden

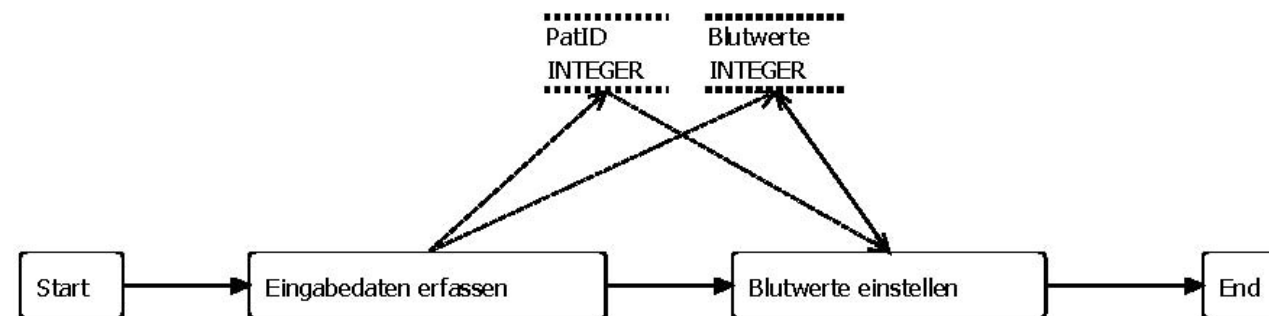
## 2.6 AristaFlow-Prozessmodell

### ○ Beispiel (Forts.)

#### Resultierender Subprozess:



#### und der Vaterprozess dazu:



# Inhalt

---

## **2.0 Vorbemerkungen**

## **2.1 Korrektheitsaspekte von Prozessmodellen – eine Einführung**

## **2.2 Klassische Petri-Netze**

## **2.3 Höhere Petri-Netze**

## **2.4 Workflow-Netze**

## **2.5 Aktivitätennetze**

## **2.6 AristaFlow-Prozessmodell**

## **2.7 Andere Ansätze**

## **2.8 Ausdrucksmächtigkeit von Prozessbeschreibungssprachen**

## **2.9 Abschließende Bemerkungen**

## **2.10 Weiterführende Literatur**

## 2.7 Andere Ansätze

---

### Inhalt

- „Imperative“ regelbasierte Ansätze
- Constraint-basierte Ansätze

## 2.7 Andere Ansätze

---

### □ „Imperative“ regelbasierte Ansätze

#### ○ Ansatz:

Darstellung von Kontrollflussbedingungen als (evtl. erweiterte) ECA-Regeln  
(**E**CA = **E**vent **C**ondition **A**ction)

#### ○ Prinzip

##### ■ Einfache ECA-Regel:

<b>ON</b>	<i>Event</i>
<b>IF</b>	<i>Condition</i>
<b>DO</b>	<i>Action</i>

##### ■ Erweiterte ECA-Regel:

<b>ON</b>	<i>Event</i>
<b>IF</b>	<i>Condition</i>
<b>Then DO</b>	<i>Action</i>
<b>Else DO</b>	<i>Alternative Action</i>

## 2.7 Andere Ansätze

---

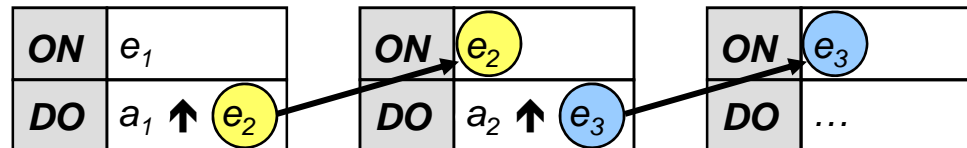
### ○ Sequenz

**ON**  $e_1$  **DO**  $a_1$  **SIGNAL**  $e_2$ ;

**ON**  $e_2$  **DO**  $a_2$  **SIGNAL**  $e_3$ ;

**ON**  $e_3$  **DO** ...

Effekt:



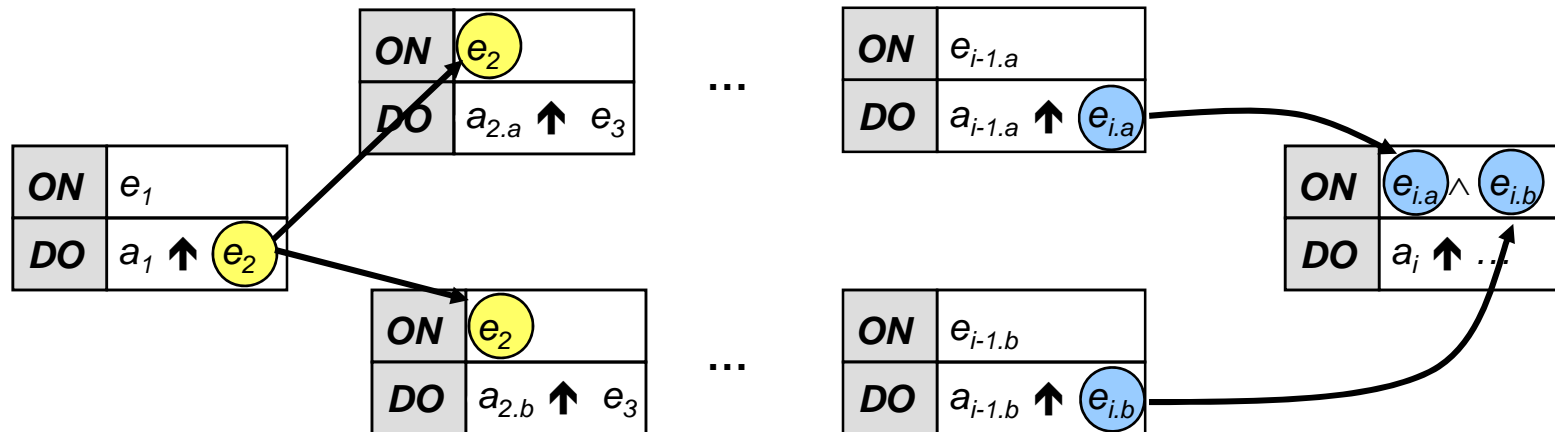


## 2.7 Andere Ansätze

### ○ Parallel (AND-Split/-Join)

ON  $e_1$  DO  $a_1$  SIGNAL  $e_2$ ;  
 ON  $e_2$  DO  $a_{2.a}$  SIGNAL  $e_{3.a}$ ;  
 ON  $e_2$  DO  $a_{2.b}$  SIGNAL  $e_{3.b}$ ;  
 ON  $e_{i-1.a}$  DO  $a_{i-1.a}$  SIGNAL  $e_{i.a}$ ;  
 ON  $e_{i-1.b}$  DO  $a_{i-1.b}$  SIGNAL  $e_{i.b}$ ;  
 ...  
 ON  $e_{i-1.a}$  **AND**  $e_{i-1.b}$  DO  $a_i$  SIGNAL ...;

**Effekt:**

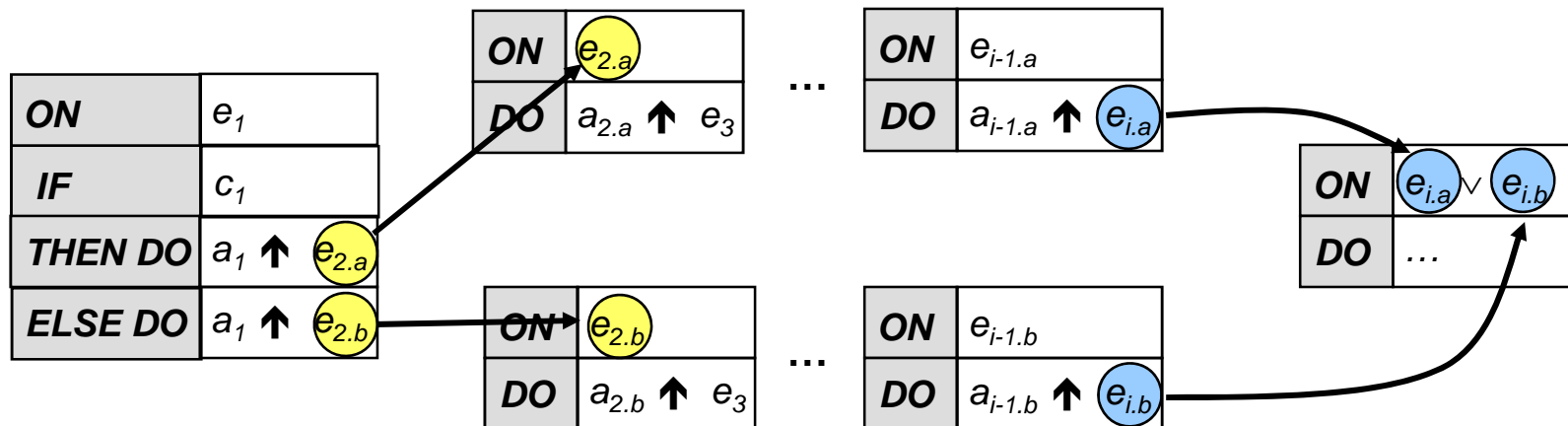


## 2.7 Andere Ansätze

### ○ Alternativ (XOR-Split/-Join)

ON  $e_1$  **IF**  $c_1$  **THEN DO**  $a_{1a}$  **SIGNAL**  $e_{2a}$  **ELSE DO**  $a_{1b}$  **SIGNAL**  $e_{2b}$ ;  
 ON  $e_{2.a}$  **DO**  $a_{2.a}$  **SIGNAL**  $e_{3.a}$ ;  
 ON  $e_{2.b}$  **DO**  $a_{2.b}$  **SIGNAL**  $e_{3.b}$ ;  
 ON  $e_{i-1.a}$  **DO**  $a_{i-1.a}$  **SIGNAL**  $e_{i.a}$ ;  
 ON  $e_{i-1.b}$  **DO**  $a_{i-1.b}$  **SIGNAL**  $e_{i.b}$ ;  
 ...  
 ( ON  $e_{i-1.a}$  **OR**  $e_{i-1.b}$  ) **DO**  $a_i$  **SIGNAL** ...;

Effekt:



## 2.7 Andere Ansätze

---

### ○ **Bewertung „imperative“ regelbasierte Prozessmodellierung**

- + sehr ausdrucksstark (falls nicht eingeschränkt)
- + für (sehr) kleine Prozesse manchmal verständlicher und kompakter als graphbasierte Darstellung
- + Eventbezogene Modellierung manchmal natürlicher als aktivitätenorientierte Sicht
- Prozessmodelle werden rasch unübersichtlich
- keine „globale Sicht“ auf den Prozess
- Korrektheitsanalysen i.d.R. sehr aufwendig (↑ Zustandsexplosionsproblem)
- Für ausführbare Prozesse:  
Ad-hoc-Abweichungen auf Prozessinstanzebene praktisch nicht realisierbar

### ○ **Anmerkung**

- Eher eine „Bottom-up“-Sicht auf den Prozess
- Die Gesamtheit der Reaktion auf die behandelten Ereignisse „ergibt“ den Prozess

## 2.7 Andere Ansätze

### □ Constraint-basierte Ansätze

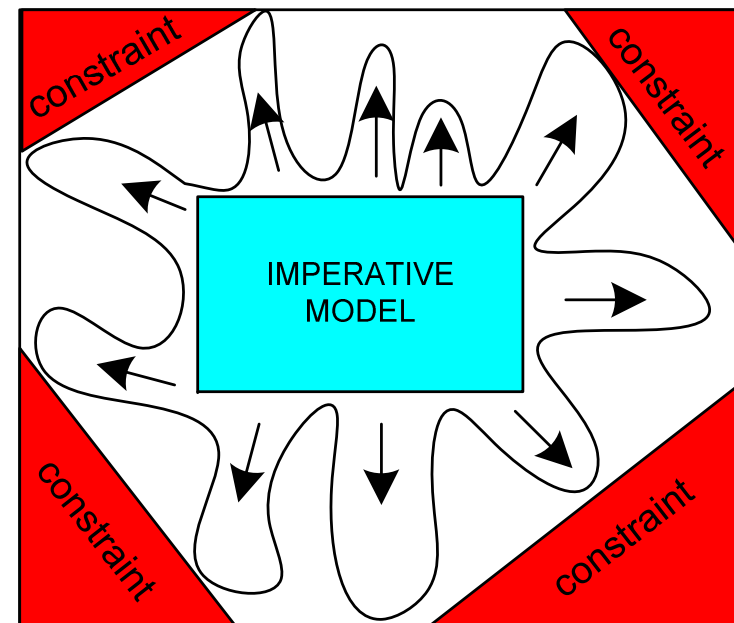
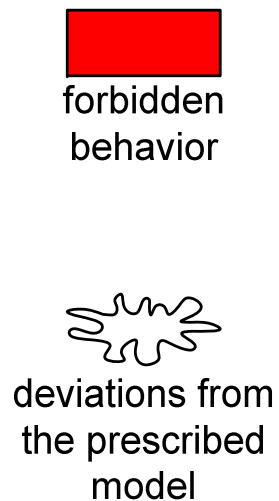
↓ Beispiel DECLARE \*

**Grundidee:**

**Alles was nicht gegen  
Vorschriften (Constraints)  
verstößt, ist erlaubt.**

**Formale Basis:**

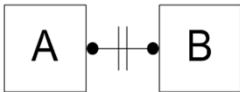
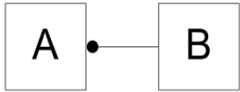
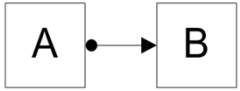
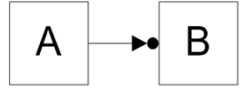
**Linear Temporal Logic (LTL)**



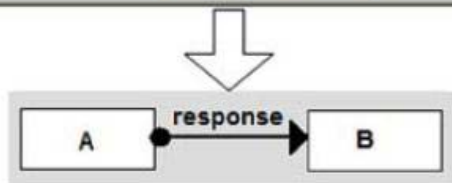
\* Abbildungen im Folgenden teilweise aus Tutorial von Prof. van der Aalst

Literatur: Pesic, M.; Schonenberg, H.; van der Aalst, W.M.P.: DECLARE: Full Support for Loosely-Structured Processes, Proc. EDOC 2007, Annapolis, Maryland, USA, Oct. 2007, pp. 287-300

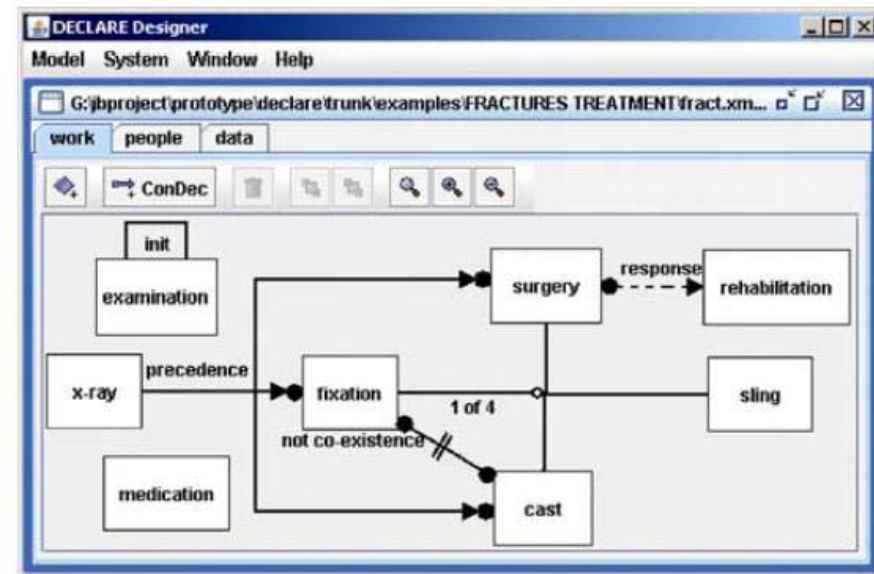
## 2.7 Andere Ansätze

Symboldarstellung	Bedeutung	Beispiele für	
		erlaubt:	nicht erlaubt:
	Wechselseitiger Ausschluss		
	Falls A, dann auch B	<p>[ ]</p> <p>[A,B,C,D,E]</p> <p>[A,A,A,C,D,E,B,B,B]</p> <p>[B,B,A,A,C,D,E]</p> <p>[B,C,D,E]</p>	<p>[A]</p> <p>[A,A,C,D,E]</p>
	Auf A muss B folgen	<p>[ ]</p> <p>[A,B,C,D,E]</p> <p>[A,A,A,B,C,D,E]</p> <p>[B,B,A,A,B,C,D,E]</p> <p>[B,C,D,E]</p>	<p>[A]</p> <p>[B,B,B,B,A,A]</p>
	Falls B, dann muss A vorausgehen	<p>[ ]</p> <p>[A,B,C,D,E]</p> <p>[A,A,A,C,D,E,B,B,B]</p> <p>[A,A,C,D,E]</p>	<p>[B]</p> <p>[B,A,C,D,E]</p>

## 2.7 Andere Ansätze



Constraint template *response*

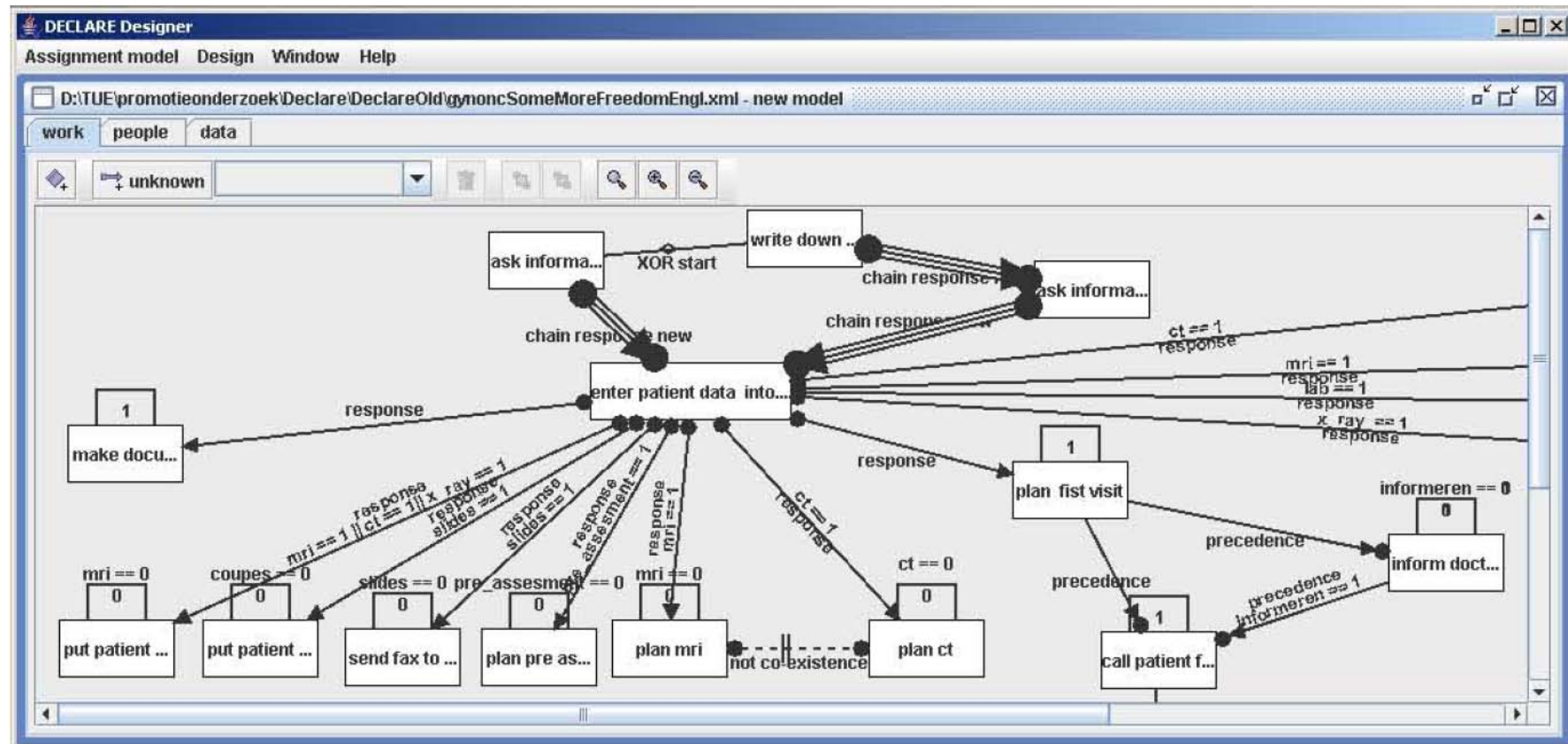


Declarative model for handling patients using *Declare*

### Screenshot (1) DECLARE Designer

Quelle: W.M.P. van der Aalst, M. Pesic, H. Schonenberg:  
Balancing Between Flexibility and Support.

## 2.7 Andere Ansätze



**Screenshot (2) DECLARE Designer**

Quelle: R.S. Mans, W.M.P. van der Aalst et al: From Requirements via Colored Workflow Nets to an Implementation in Several Workflow Systems. Proc. 8th Workshop on the Practical Use of Coloured Petri Nets and CPN Tools (CPN 2007), Aarhus, Denmark, Oct. 2007

<http://www.is.win.tue.nl/~wvdaalst/publications/publications.htm> Paper no. 398

## 2.7 Andere Ansätze

---

### ○ **Bewertung constraint-basierte Prozessmodellierung**

- + sehr ausdrucksstark (falls nicht eingeschränkt)
- + kompakte Darstellung vieler möglicher Prozessausführungen
- Prozessmodelle werden rasch (extrem) unübersichtlich
- keine „globale Sicht“ auf den Prozess
- Korrektheitsanalysen i.d.R. sehr aufwendig (↑ Zustandsexplosionsproblem)
- Performanzproblem, für große Prozessmodelle und viele Instanzen praktisch ungeeignet
- Für ausführbare Prozesse:  
Ad-hoc-Abweichungen auf Prozessinstanzebene praktisch nicht realisierbar

### ○ **Anmerkung**

Allerdings als Ergänzung zur klassischen Modellierung geeignet, z.B. zur Formulierung semantischer Constraints (siehe z.B. SeaFlows-Projekt des Instituts DBIS)



# Inhalt

---

## **2.0 Vorbemerkungen**

## **2.1 Korrektheitsaspekte von Prozessmodellen – eine Einführung**

## **2.2 Klassische Petri-Netze**

## **2.3 Höhere Petri-Netze**

## **2.4 Workflow-Netze**

## **2.5 Aktivitätennetze**

## **2.6 AristaFlow-Prozessmodell**

## **2.7 Andere Ansätze**

## **2.8 Ausdrucksmächtigkeit von Prozessbeschreibungssprachen**

## **2.9 Abschließende Bemerkungen**

## **2.10 Weiterführende Literatur**

## 2.8 Ausdrucksmächtigkeit von Prozessbeschreibungssprachen

---

### □ **Problem: Sehr viele unterschiedliche Ansätze und Formalismen**

- Welcher ist „besser“?
- Wie vergleichen sie sich in Bezug auf ihre Ausdrucksmächtigkeit?

### □ **Ein Ansatz: Vergleich anhand von „Mustern“ (engl. pattern)**

- ↓ workflow control-flow patterns
- ↓ workflow resource patterns
- ↓ workflow data patterns
- ↓ exception handling patterns

## 2.8 Ausdrucksmächtigkeit von Prozessbeschreibungssprachen

---

<http://www.workflowpatterns.com/patterns/control/index.php>

### Control-Flow Patterns

**Downloads of the original and revised control-flow patterns papers:**

N. Russell, A.H.M. ter Hofstede, W.M.P. van der Aalst, and N. Mulyar.

[Workflow Control-Flow Patterns: A Revised View](#). (PDF, 1.04Mb)

*BPM Center Report BPM-06-22*, BPMcenter.org, 2006.

W.M.P van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros.

[Workflow Patterns](#). (PDF, 718 Kb).

*Distributed and Parallel Databases*, 14(3), pages 5-51, July 2003

### Introduction

The *Workflow Patterns Initiative* was established with the aim of delineating the fundamental requirements that arise during business process modelling on a recurring basis and describe them in an imperative way. The first deliverable of this research project was a set of twenty patterns describing the control-flow perspective of workflow systems. Since their release, these patterns have been widely used by practitioners, vendors and academics alike in the selection, design and development of workflow systems [vdAtHKB03]. This body of work presents the first systematic review of the original twenty control-flow patterns and provides a formal description of each of them in the form of a Coloured Petri-Net (CPN) model. It also identifies twenty three new patterns relevant to the control-flow perspective. Detailed context conditions and evaluation criteria are presented for each pattern and their implementation is assessed in fourteen commercial offerings including workflow and case handling systems, business process modelling formalisms and business process execution languages.

## 2.8 Ausdrucksmächtigkeit von Prozessbeschreibungssprachen

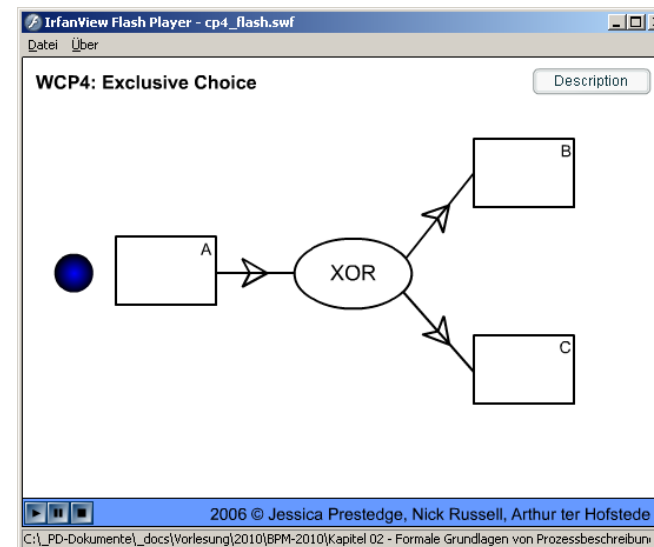
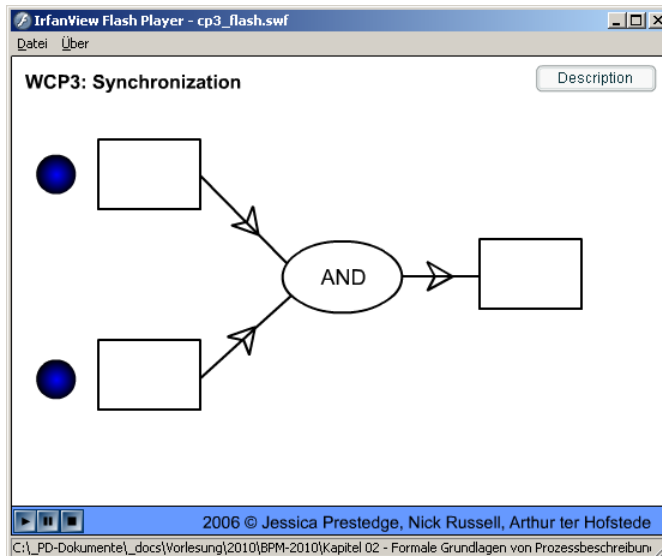
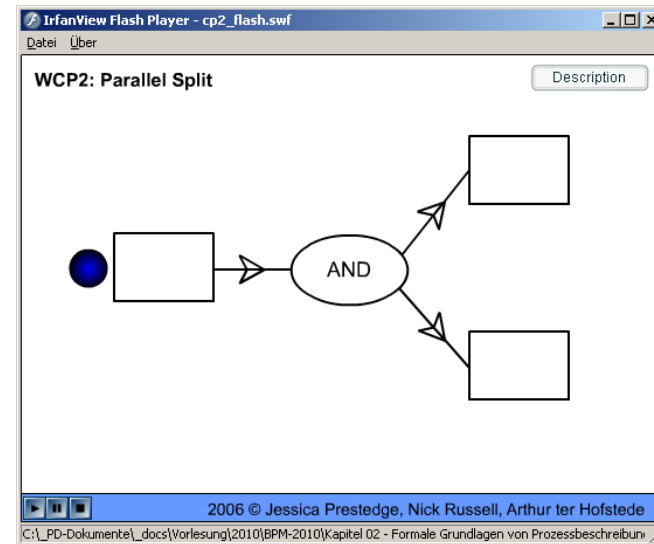
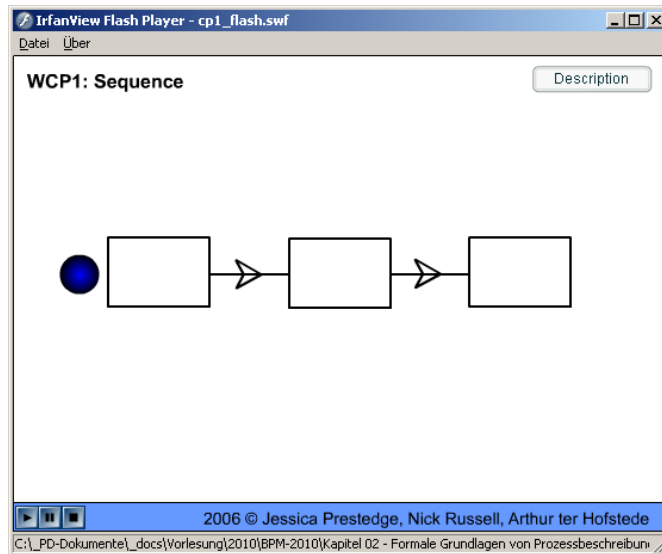
<http://www.workflowpatterns.com/evaluations/commercial/index.php>

### Control-Flow Patterns

Pattern	Product						
	<a href="#">Staffware</a>	<a href="#">WebSphere MQ Workflow</a>	<a href="#">FLOWer</a>	<a href="#">COSA</a>	<a href="#">iPlanet</a>	<a href="#">SAP Workflow</a>	<a href="#">FileNet</a>
<a href="#">Sequence</a>	+	+	+	+	+	+	+
<a href="#">Parallel Split</a>	+	+	+	+	+	+	+
<a href="#">Synchronization</a>	+	+	+	+	+	+	+
<a href="#">Exclusive Choice</a>	+	+	+	+	+	+	+
<a href="#">Simple Merge</a>	+	+	+	+	+	+	+
<a href="#">Multi-Choice</a>	-	+	+	+	+	-	+
<a href="#">Structured Synchronizing Merge</a>	-	+	+	-	-	-	+
<a href="#">Multi-Merge</a>	-	-	+/-	-	+	-	+
<a href="#">Structured Discriminator</a>	-	-	-	-	+	+/-	-
<a href="#">Arbitrary Cycles</a>	+	-	-	+	+	-	+
<a href="#">Implicit Termination</a>	+	+	+	-	-	-	+
<a href="#">Multiple Instances without Synchronization</a>	+	-	+	+	+	+/-	+
<a href="#">Multiple Instances with a Priori Design-Time Knowledge</a>	+	-	+	-	-	+	-
<a href="#">Multiple Instances with a Priori Run-Time</a>	,		,			,	

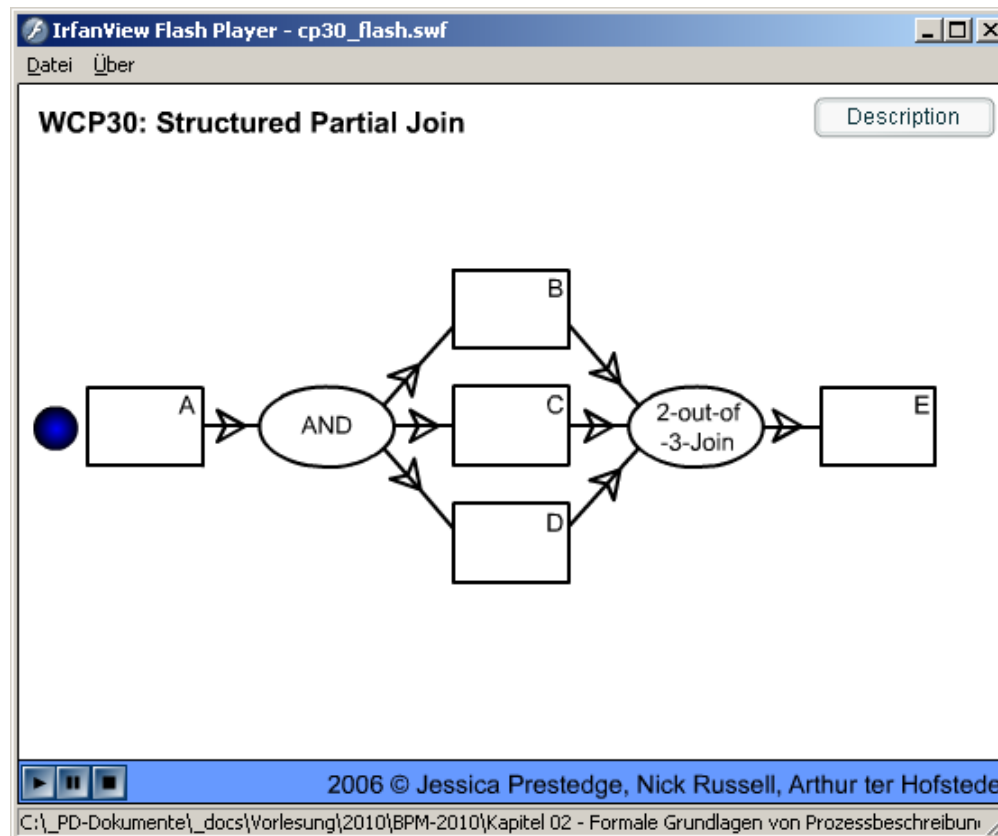
...

## 2.8 Ausdrucksmächtigkeit von Prozessbeschreibungssprachen



## 2.8 Ausdrucksmächtigkeit von Prozessbeschreibungssprachen

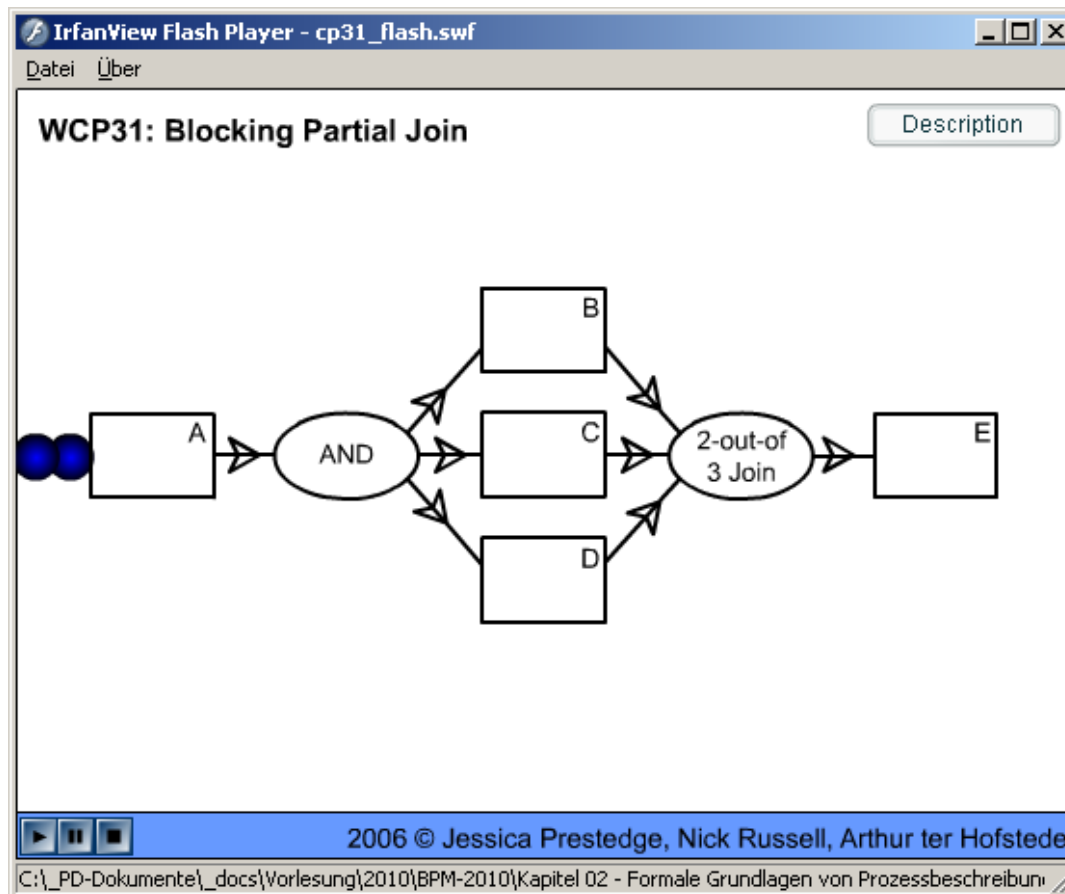
Hier noch einige spezielle Patterns:



### WCP30 Description

The convergence of M branches into a single subsequent branch following a corresponding divergence earlier in the process model. The thread of control is passed to the subsequent branch when N of the incoming branches have been enabled. Subsequent enablements of incoming branches do not result in the thread of control being passed on. The join construct resets when all active incoming branches have been enabled.

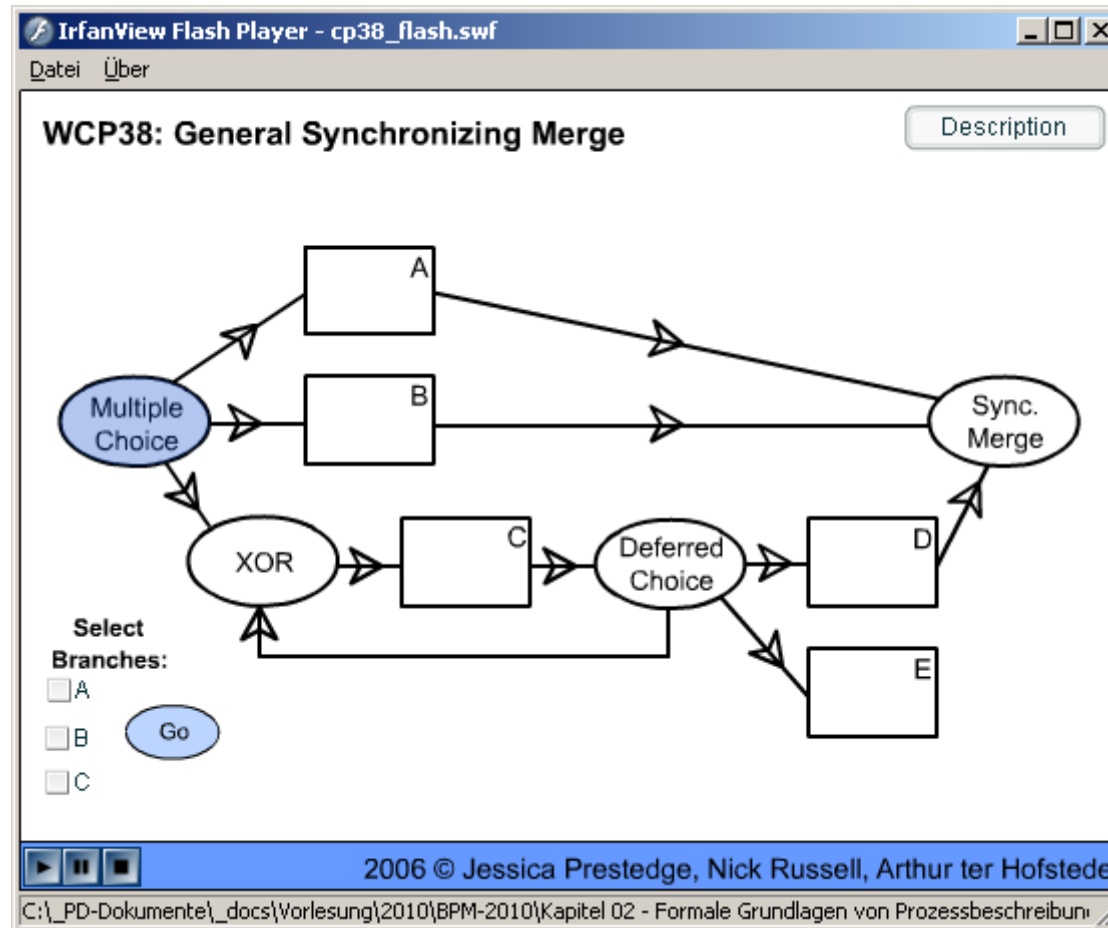
## 2.8 Ausdrucksmächtigkeit von Prozessbeschreibungssprachen



### WCP31 Description

The convergence of two or more branches into a single subsequent branch following one or more corresponding divergences earlier in the process model. The thread of control is passed to the subsequent branch when N of the incoming branches has been enabled. The join construct resets when all active incoming branches have been enabled once for the same process instance. Subsequent enablements of incoming branches are blocked until the join has reset.

## 2.8 Ausdrucksmächtigkeit von Prozessbeschreibungssprachen



### WCP38 Description

The convergence of two or more branches which diverged earlier in the process into a single subsequent branch. The thread of control is passed to the subsequent branch when each active incoming branch has been enabled or it is not possible that the branch will be enabled at any future time.



## 2.8 Ausdrucksmächtigkeit von Prozessbeschreibungssprachen

---

### □ Anmerkungen

- Der Vergleich von Prozessmodellierungssprachen hinsichtlich ihrer Mächtigkeit ist nicht einfach – Patterns können hierbei eine wertvolle Orientierungshilfe leisten
- Allerdings:
  - Viele dieser Patterns wirken sehr „konstruiert“ und dürften nur von theoretischem Interesse sein
  - Testfrage für praktische Relevanz:  
Sind die damit assoziierten Anwendungsfälle so generisch lösbar, dass man hierfür ein (statisches) Prozessmodell erstellen würde?
- Außerdem:
  - Die beschriebenen Patterns stark von PMS beeinflusst, die keine (oder nur eine stark eingeschränkte) dynamische Adaption von Prozessinstanzen erlauben
  - Bei flexiblen PMS ergeben sich vielfältige andere Lösungsmöglichkeiten

# Inhalt

---

## **2.0 Vorbemerkungen**

## **2.1 Korrektheitsaspekte von Prozessmodellen – eine Einführung**

## **2.2 Klassische Petri-Netze**

## **2.3 Höhere Petri-Netze**

## **2.4 Workflow-Netze**

## **2.5 Aktivitätennetze**

## **2.6 AristaFlow-Prozessmodell**

## **2.7 Andere Ansätze**

## **2.8 Ausdrucksmächtigkeit von Prozessbeschreibungssprachen**

## **2.9 Abschließende Bemerkungen**

## **2.10 Weiterführende Literatur**

## 2.9 Abschließende Bemerkungen

---

- ❑ Prozessbeschreibungssprachen sollten generell eine formale, wohldefinierte „Schaltsemantik“ haben, um das Ausführungsverhalten eindeutig beschreiben zu können  
... leider trifft dies nicht immer zu (siehe später)
- ❑ Für die Modellierung *ausführbarer Prozesse* ist eine solche „Schaltsemantik“ unabdingbar  
... und liegt – realisiert in den „Prozess-Engines“ von PMS – auch vor
- ❑ Neben der exakten „Schaltsemantik“ ist auch die Analysierbarkeit auf mögliche Modellierungsfehler sehr wichtig  
... hier ergeben sich je nach Formalismus z.T. erhebliche Unterschiede
- ❑ Derzeit noch große Heterogenität bei den verschiedenen Ansätzen
- ❑ Deshalb sehr wichtig, sich nicht von Oberflächlichkeiten beeindrucken zu lassen,  
... sondern auf die (für die eigenen Belange) essentiellen Dinge achten!

## **2.0 Vorbemerkungen**

## **2.1 Korrektheitsaspekte von Prozessmodellen – eine Einführung**

## **2.2 Klassische Petri-Netze**

## **2.3 Höhere Petri-Netze**

## **2.4 Workflow-Netze**

## **2.5 Aktivitätennetze**

## **2.6 AristaFlow-Prozessmodell**

## **2.7 Andere Ansätze**

## **2.8 Ausdrucksmächtigkeit von Prozessbeschreibungssprachen**

## **2.9 Abschließende Bemerkungen**

## **2.10 Weiterführende Literatur**

## 2.10 Weiterführende Literatur

---

### □ Lehrbücher

- Bernd Baumgarten: Petri-Netze - Grundlagen und Anwendungen, 2. Auflage, Spektrum-Verlag, 1996
- Mathias Weske: Business Process Management - Concepts, Languages, Architectures. Springer, 2007
- M. Dumas, W.M.P. van der Aalst, A.H.M. ter Hofstede: Process-Aware Information Systems. Wiley-Interscience, 2005

### □ Fachartikel / Dissertation

- W.M.P. van der Aalst: The Application of Petri Nets to Workflow Management. The Journal of Circuits, Systems and Computers, Vol. 8, No. 1, 1998, S. 21-66
- K. Jensen, W.M.P. van der Aalst (Eds.): Transactions on Petri Nets and Other Models of Concurrency II – Special Issue on Concurrency in Process-Aware Information Systems  
Springer, Lecture Notes in Computer Science, LNCS 5460, 2009
- W.M.P. van der Aalst, A.H.M. ter Hofstede: YAWL: Yet Another Workflow Language. Information Systems, Vol. 30, Issue 4, June 2005, pp. 245-275
- F. Leymann, W. Altenhuber: Managing Business Processes as an Information Resource. IBM Systems Journal, Vol. 33, No. 2, 1994, pp. 326-348
- M. Reichert: Dynamische Ablaufänderungen in Workflow-Management-Systemen. Dissertation, Fakultät für Informatik, Universität Ulm, 2000 \*
- M. Reichert, P. Dadam: ADEPT<sub>flex</sub> – Supporting Dynamic Changes of Workflows Without Losing Control. Kluwer, Journal of Intelligent Information Systems, Vol. 10, No. 2, pp. 93-129 \*

---

\* Download via [www.uni-ulm.de/dbis](http://www.uni-ulm.de/dbis) → Publikationen