LEHRSTUHL 14
SOFTWARE ENGINEERING

Willkommen zur Vorlesung

Methodische Grundlagen des Software-Engineering

im Sommersemester 2011

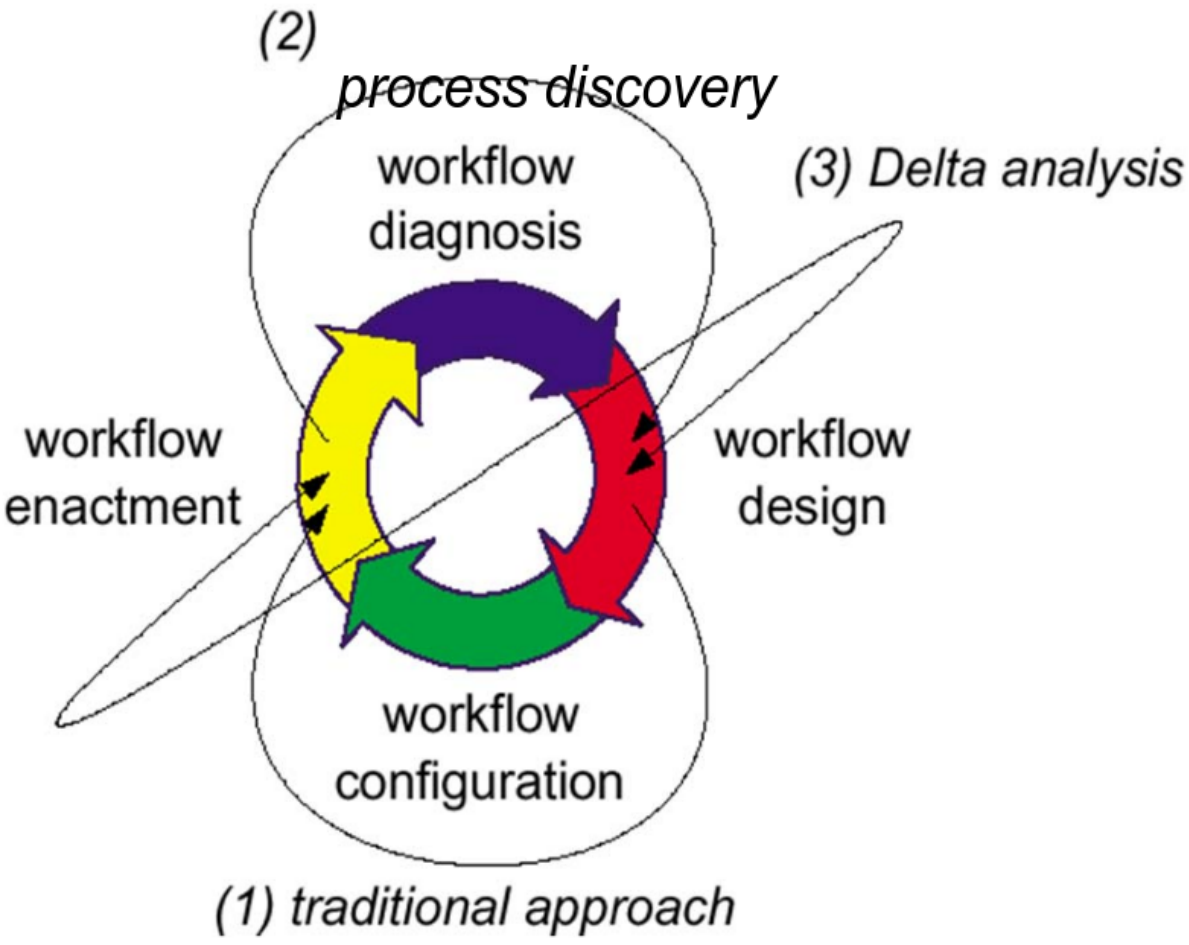
Prof. Dr. Jan Jürjens

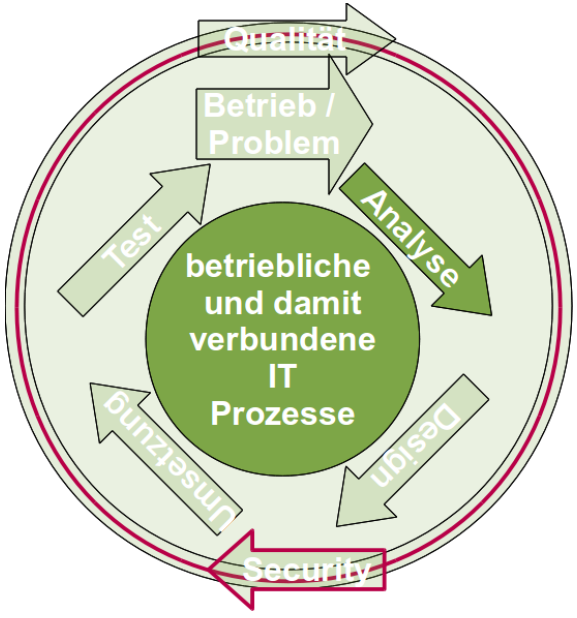
TU Dortmund, Fakultät Informatik, Lehrstuhl XIV

07. Business-Process-Mining

[inkl. Beiträge von Jutta Mülle und Dr. Silvia von Stackelberg, LS Prof. Böhm, Karlsruher Institut für Technologie]

Einordnung
Business Process Mining





Einordnung
Business Process Mining

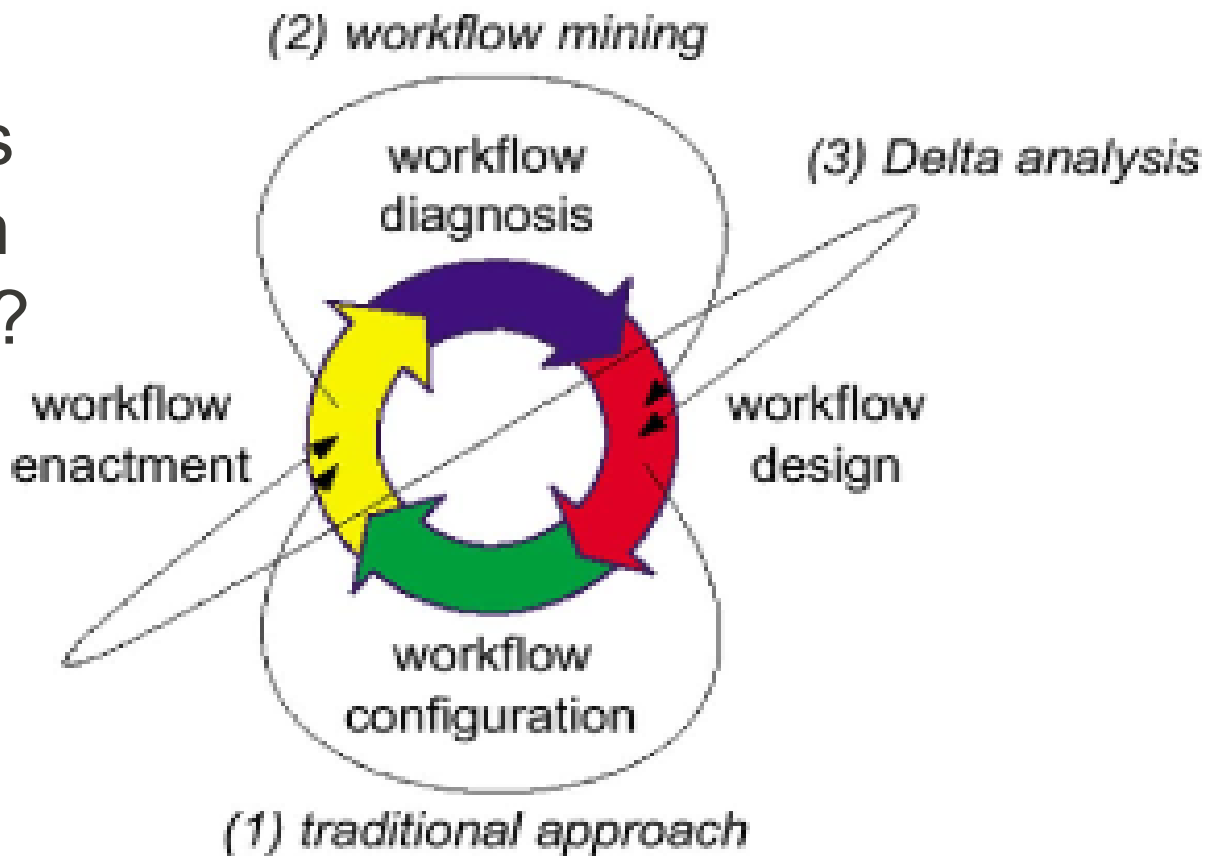
- Business Prozesse
  - Anwendungsbeispiel Finanz- und Versicherungsdomäne
  - Grundlagen Geschäfts-Prozesse
  - Elektronische Prozessketten
  - Einführung in die BPMN
  - Business-Process-Mining
  - Workflow-Management-Systeme
- Qualitätsmanagement
- Testen
- Sicherheit
- Sicheres Software Design

Business-Process-Mining

- Analyse-Techniken für bestehende Systeme.
- Ziel: Verbesserungs-Potentiale aufdecken.
- Kann als Input für BPR (Business Process Reengineering) und CPI (Continuous Process Improvement) genutzt werden.
- Synonym: Workflow-Mining.

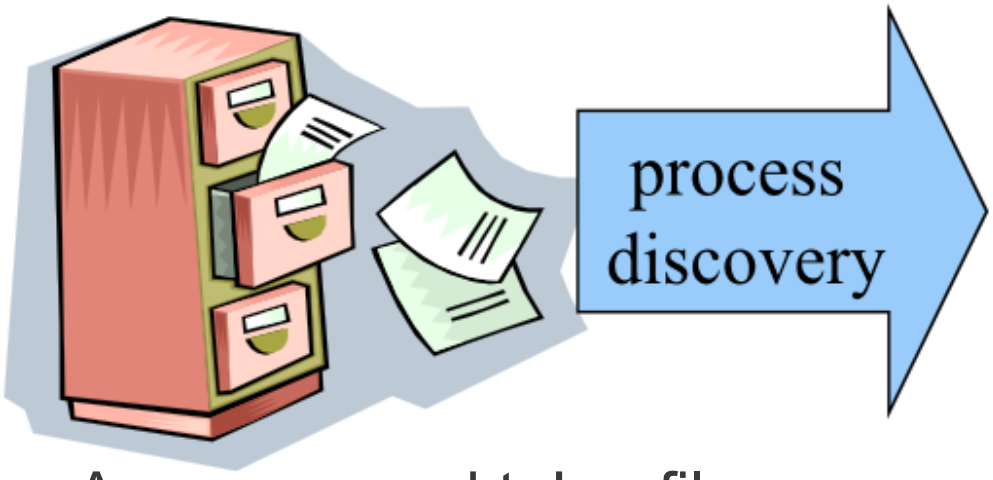
Workflow-Mining

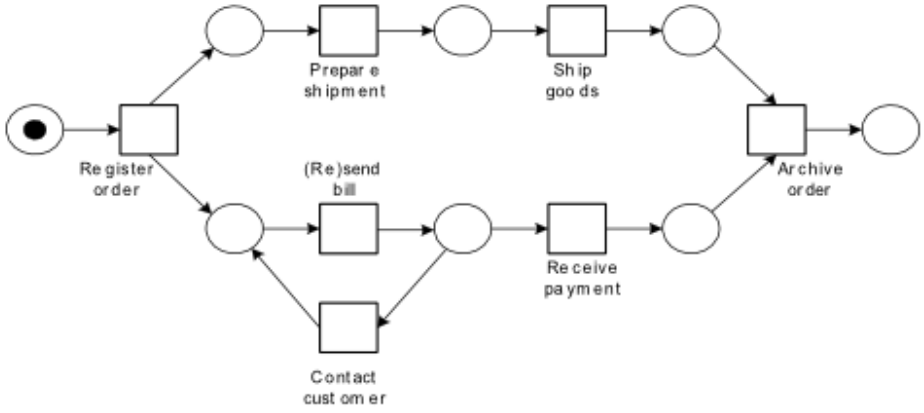
- Verschiedene Aspekte:
  - Process Discovery (2) – Wie sieht der Prozess (wirklich) aus?
  - Delta Analysis (3) – Passiert das, was spezifiziert wurde?
  - Performance Analysis – Wie können wir den Workflow verbessern?



Process-Discovery

- Idee: Umdrehen des traditionellen Ansatzes





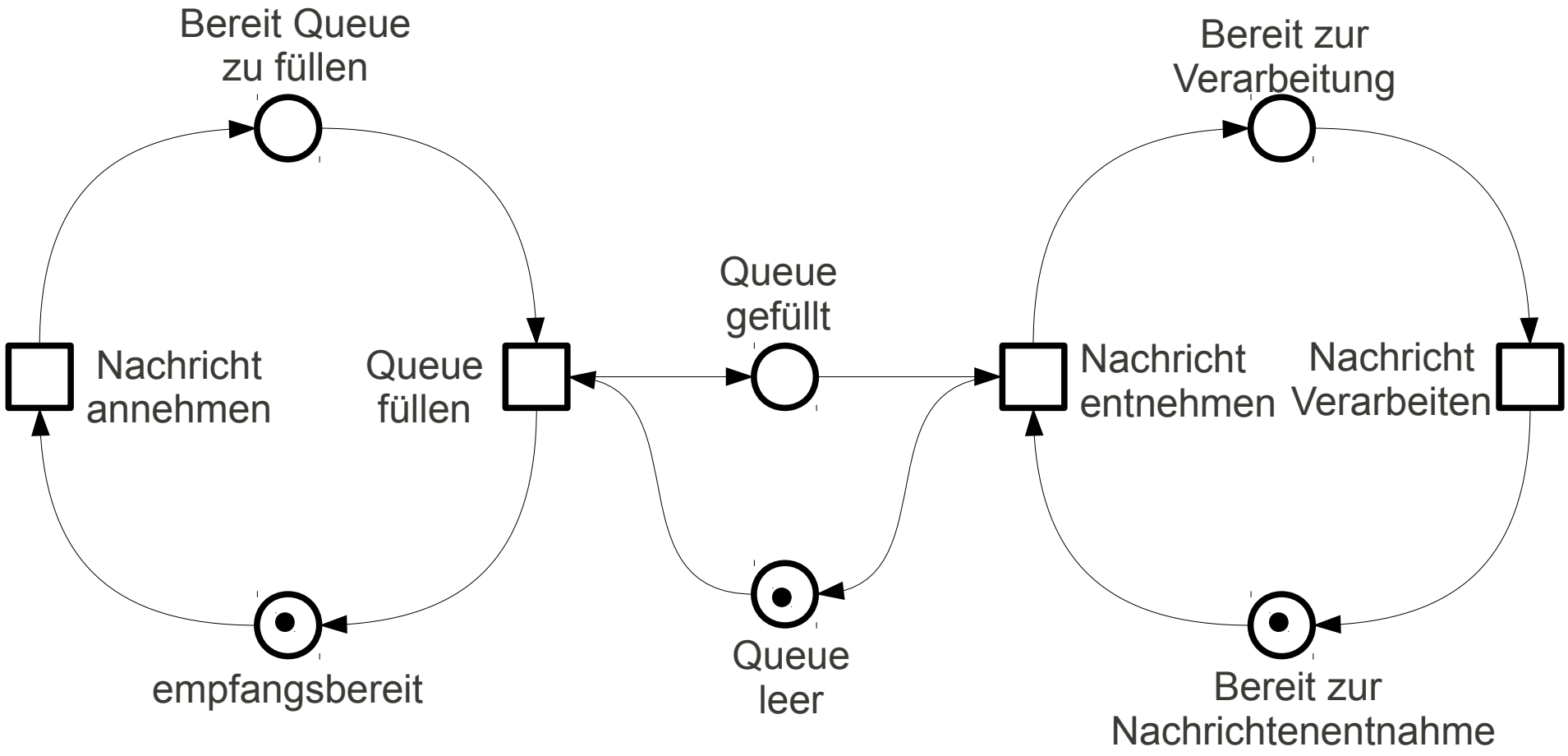
- Ausgangspunkt: Logfile
  - Minimal: Fall-Nummer, Task, zeitliche Ordnung
  - Optional: Genaue Zeit, User, assoziierte Daten etc.
  - „Rauschfrei“
  - Vollständig“
- Ergebnis: Workflow-Schema als Petri-Netz

Exkurs Petri-Netze

- Ursprung: Dissertationsschrift
  - "Kommunikation mit Automaten" von Carl Adam Petri (1962)
- Seither: mehr als 10.000 Arbeiten auf dem Gebiet.
- Bis 1985: hauptsächlich von Theoretikern benutzt.
- Seit Mitte der 80er Jahre: vermehrter Einsatz in praktischen Anwendungen.

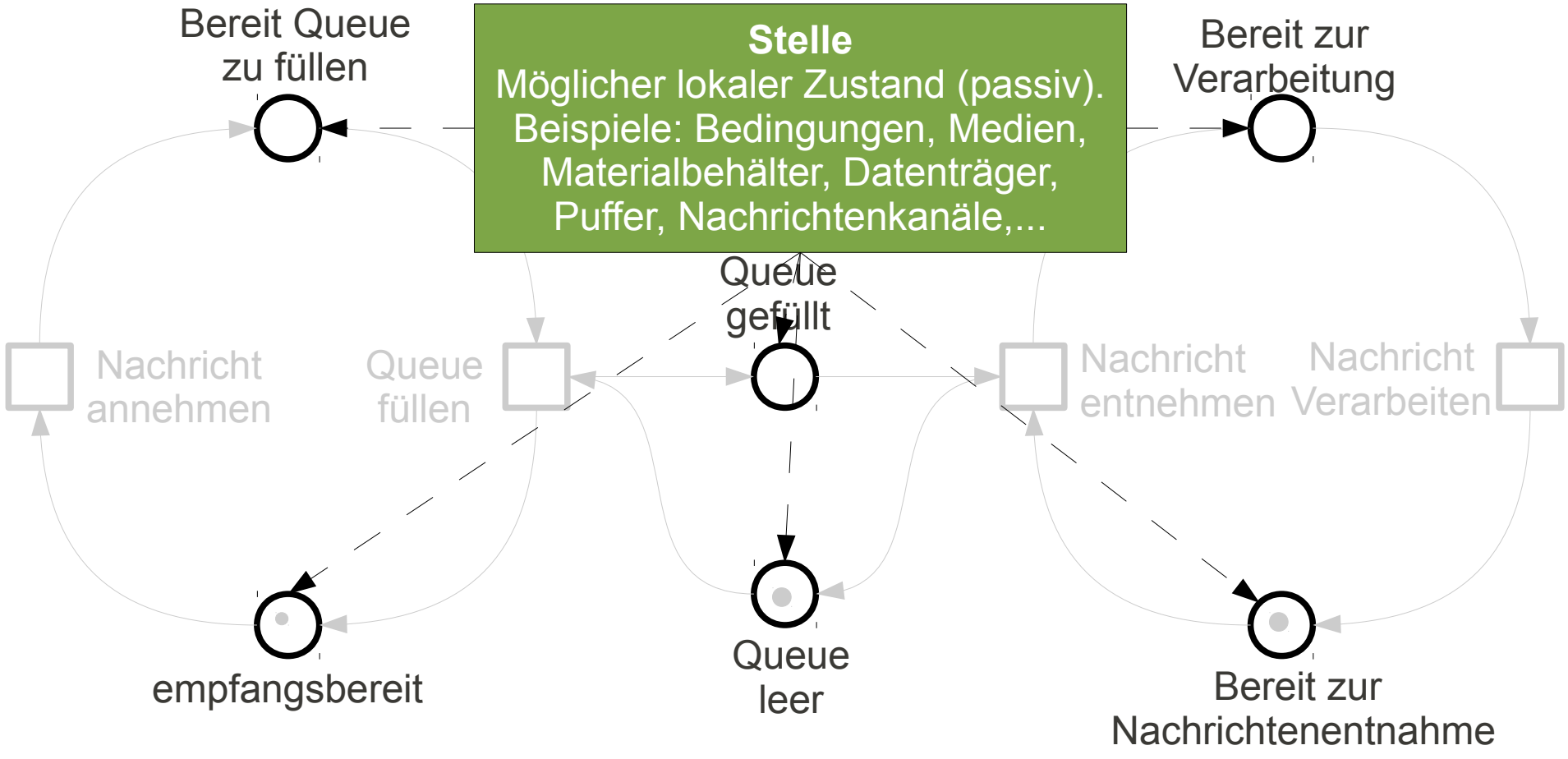
Petri-Netze: Beispiel

- Beispiel Nachrichten-Queue



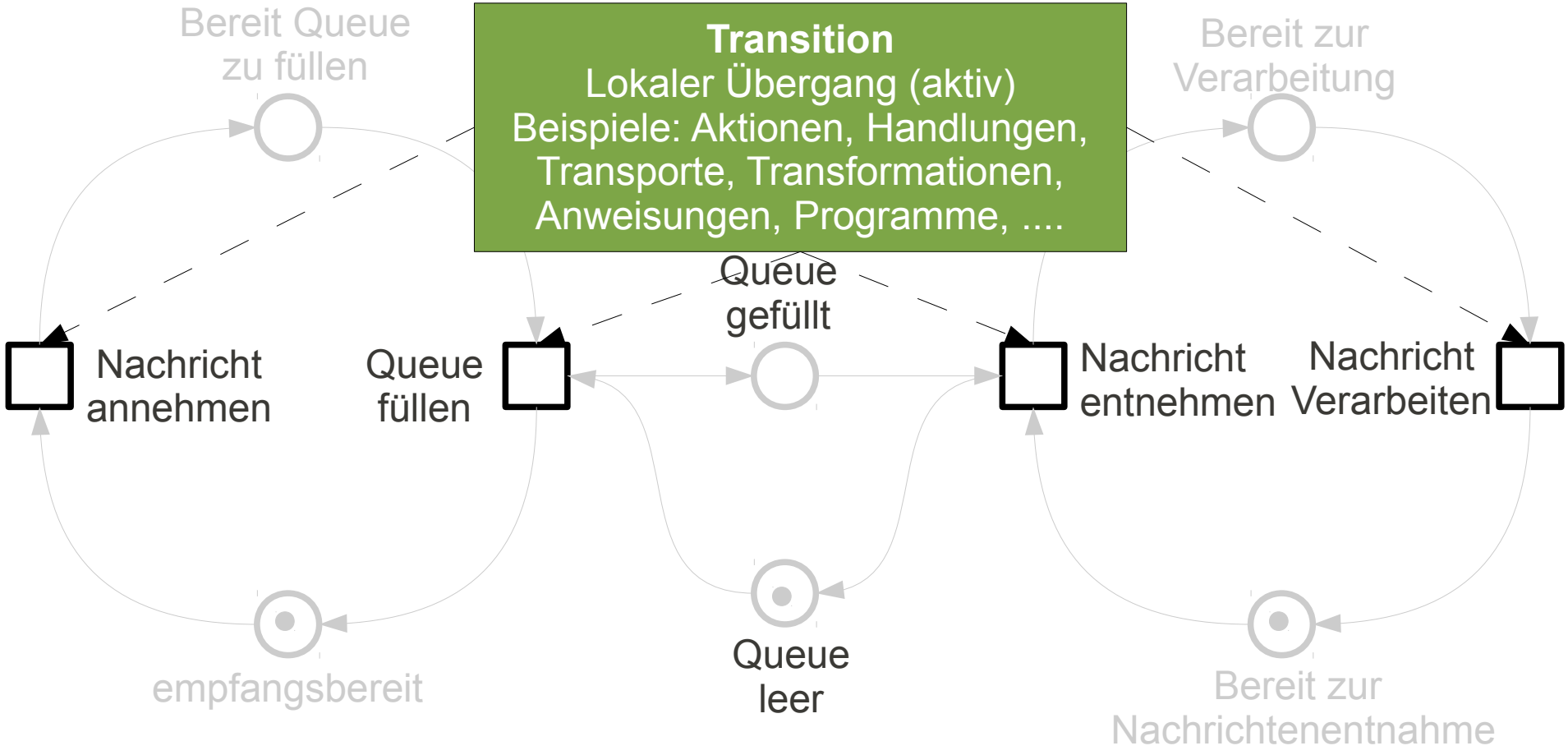
Petri-Netze: Stelle

- Beispiel Nachrichten-Queue



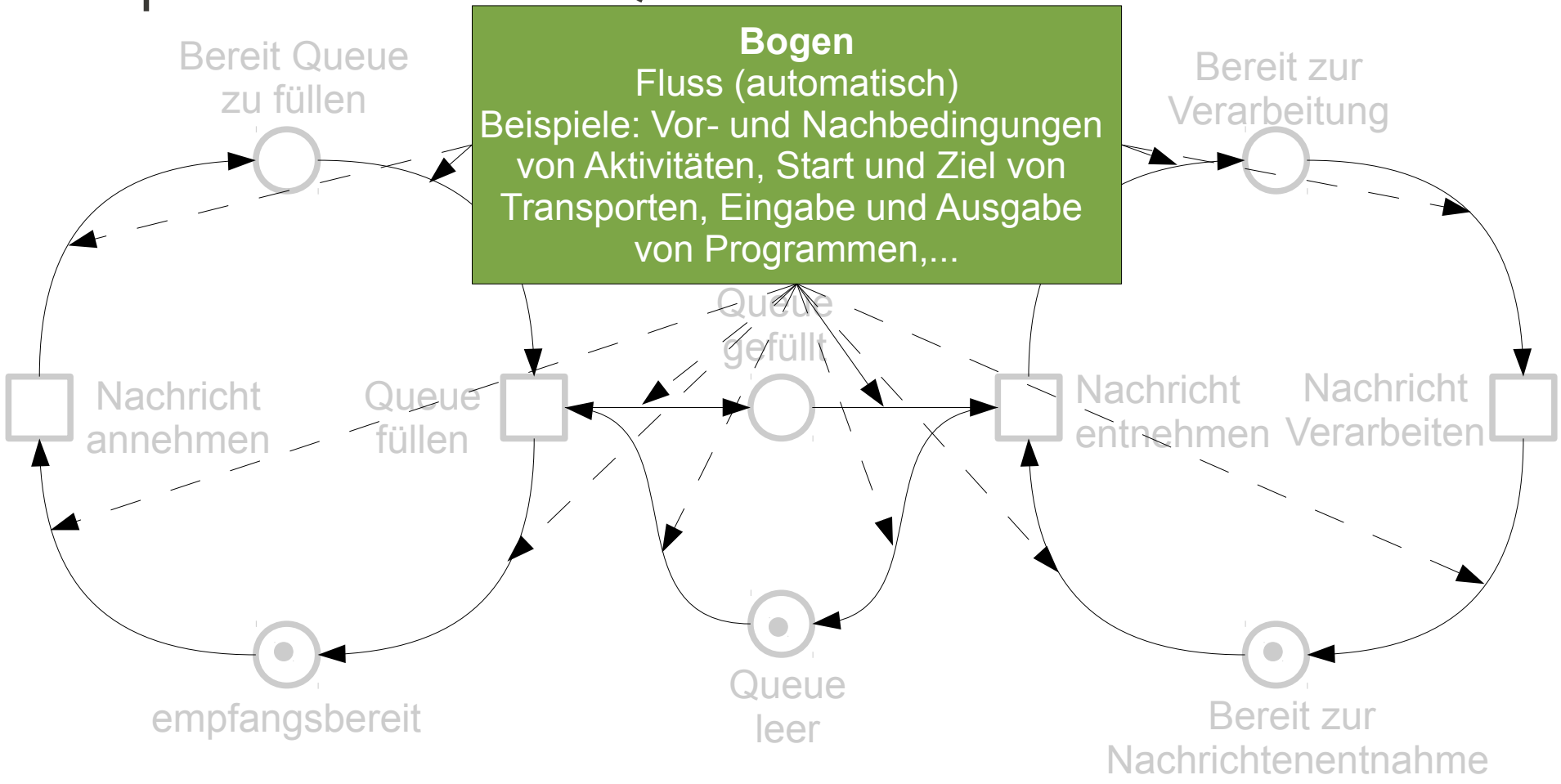
Petri-Netze: Transition

- Beispiel Nachrichten-Queue als Netz



Petri-Netze: Bogen

- Beispiel Nachrichten-Queue



Definition Netz

Definition Netz:

- Gegeben:
  - S - endliche Menge von Stellen
  - T - endliche Menge von Transitionen
  - F - Menge von Bögen
    - $F \subseteq (S \times T) \cup (T \times S)$ binäre Relation
- wenn: $S \cap T = \emptyset \; und \; S \cup T \neq \emptyset$
- dann: (S,T,F) ein Netz

Definition Netz: Beispiel

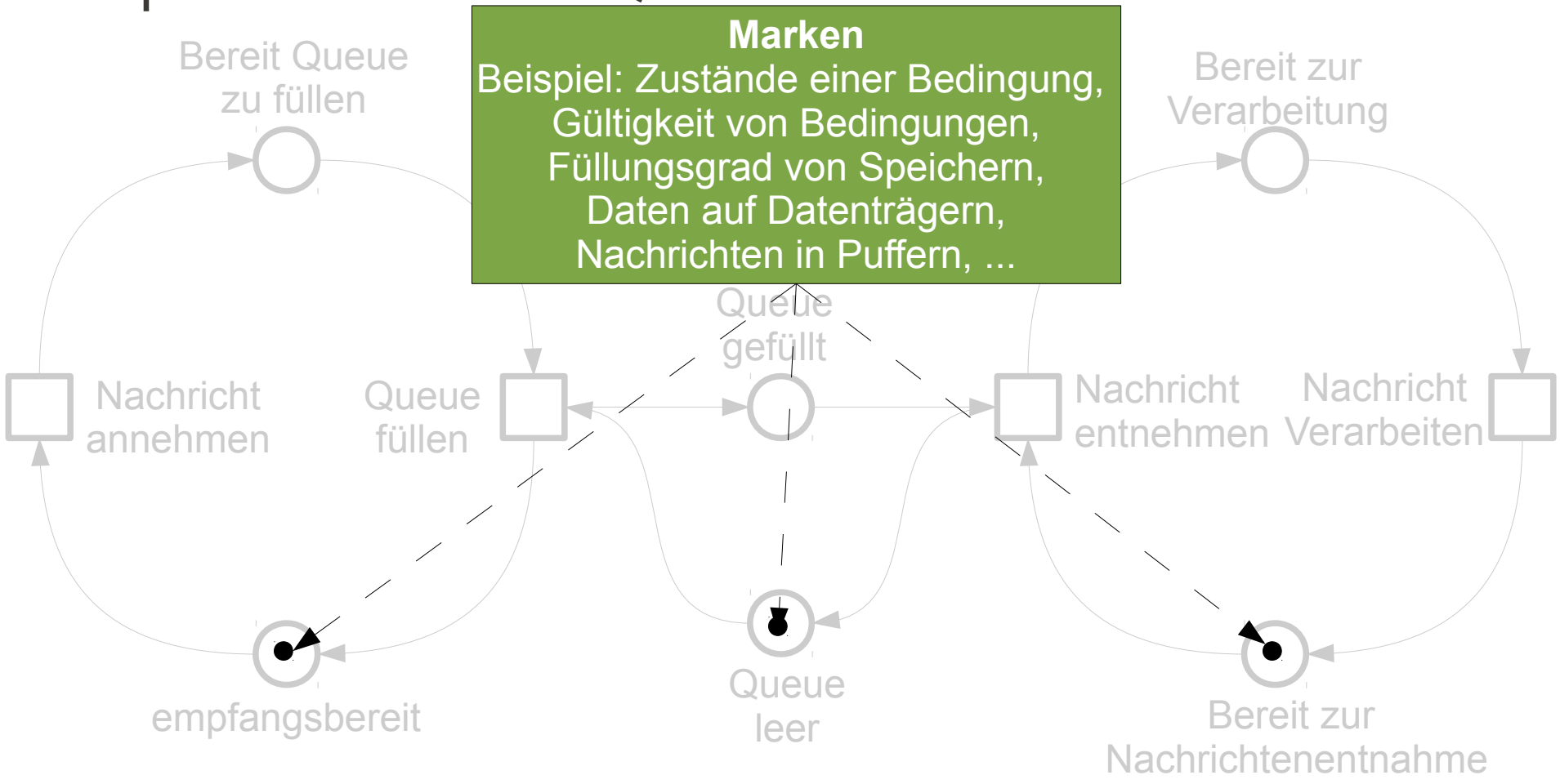
- Nachrichten-Queue: Netz (S,T,F) mit
  - S = {empfangsbereit, Bereit Queue zu füllen, Queue gefüllt, Queue leer, Bereit zur Verarbeitung, Bereit zur Nachrichtenentnahme}
  - T = {Nachricht annehmen, Queue füllen, Nachricht entnehmen, Nachricht verarbeiten}
  - F = {(empfangsbereit, Nachricht annehmen), (Nachricht annehmen, Bereit Queue zu füllen), (Bereit Queue zu füllen, Queue füllen), (Queue füllen, empfangsbereit), (Queue füllen, Queue gefüllt), (Queue gefüllt, Nachricht entnehmen), (Nachricht entnehmen, Queue leer), (Queue leer, Queue füllen), (Bereit zur Nachrichtenentnahme, Nachricht entnehmen), (Nachricht entnehmen, Bereit zur Verarbeitung), (Bereit zur Verarbeitung, Nachricht verarbeiten), (Nachricht verarbeiten, Bereit zur Nachrichtenentnahme)}

Frage

- Welche grundlegenden Eigenschaften fehlen dieser Definition von Netz?

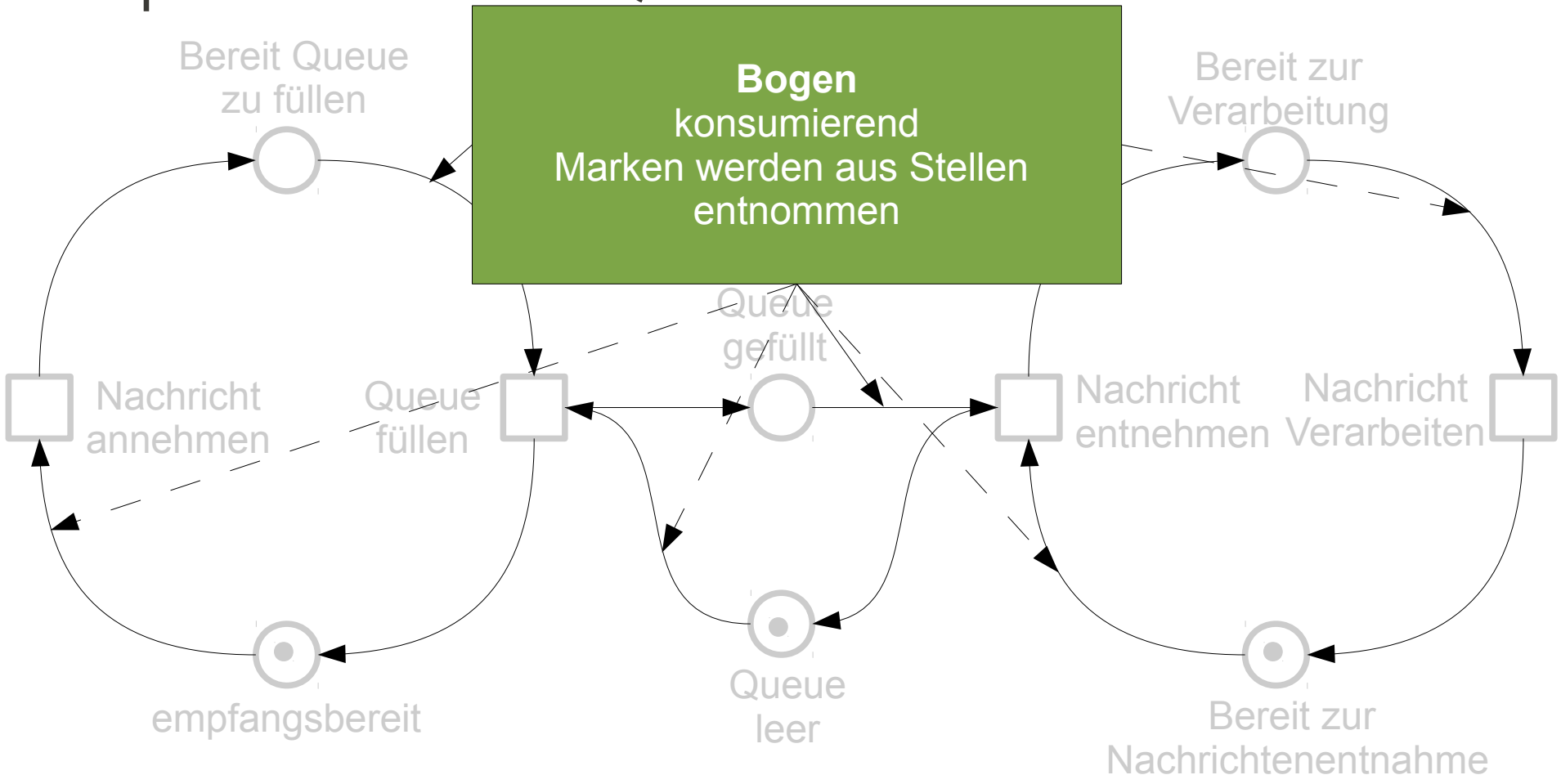
Petri-Netz: Marken

- Beispiel Nachrichten-Queue



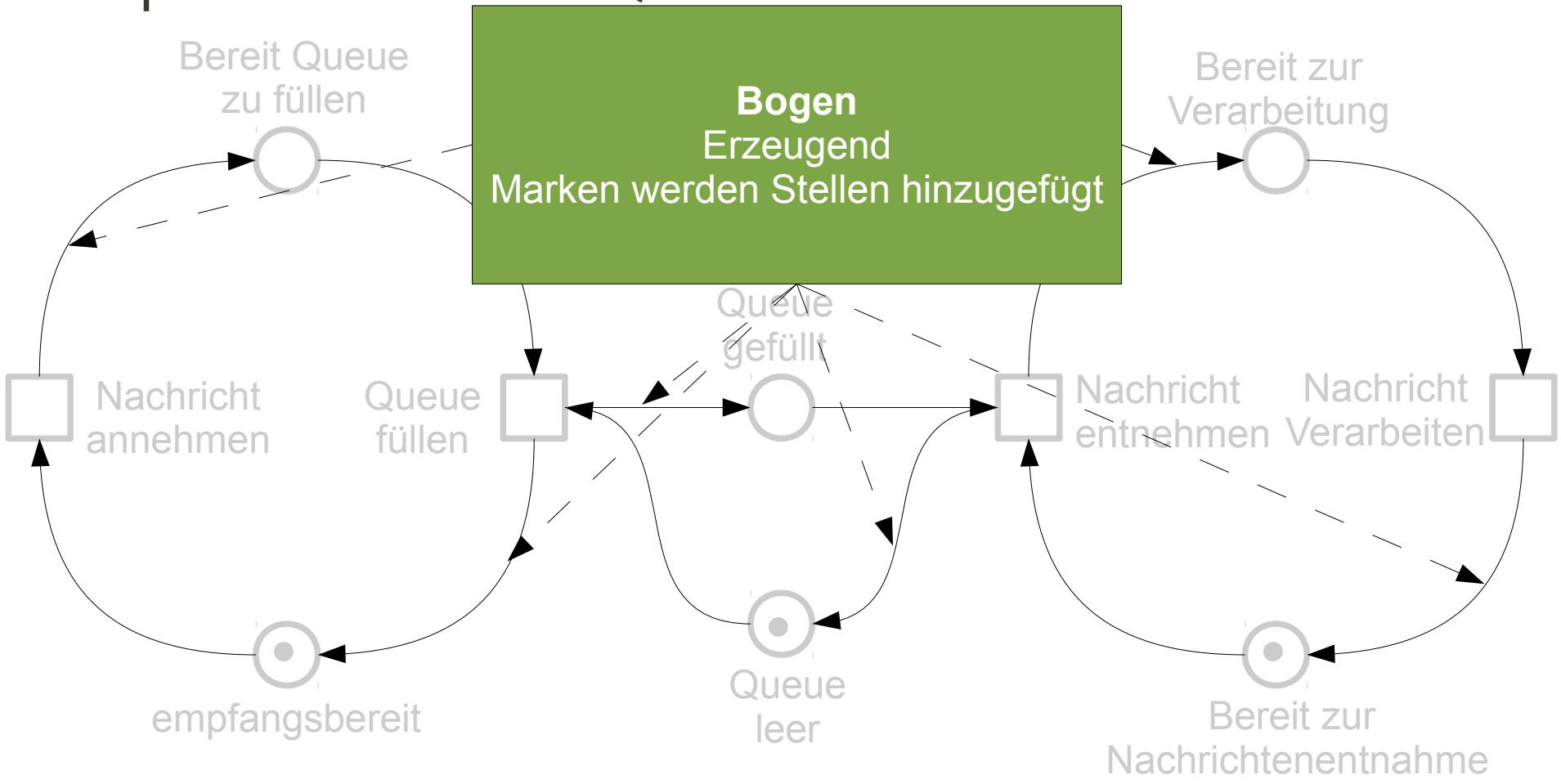
Bogen (konsumierend)

- Beispiel Nachrichten-Queue



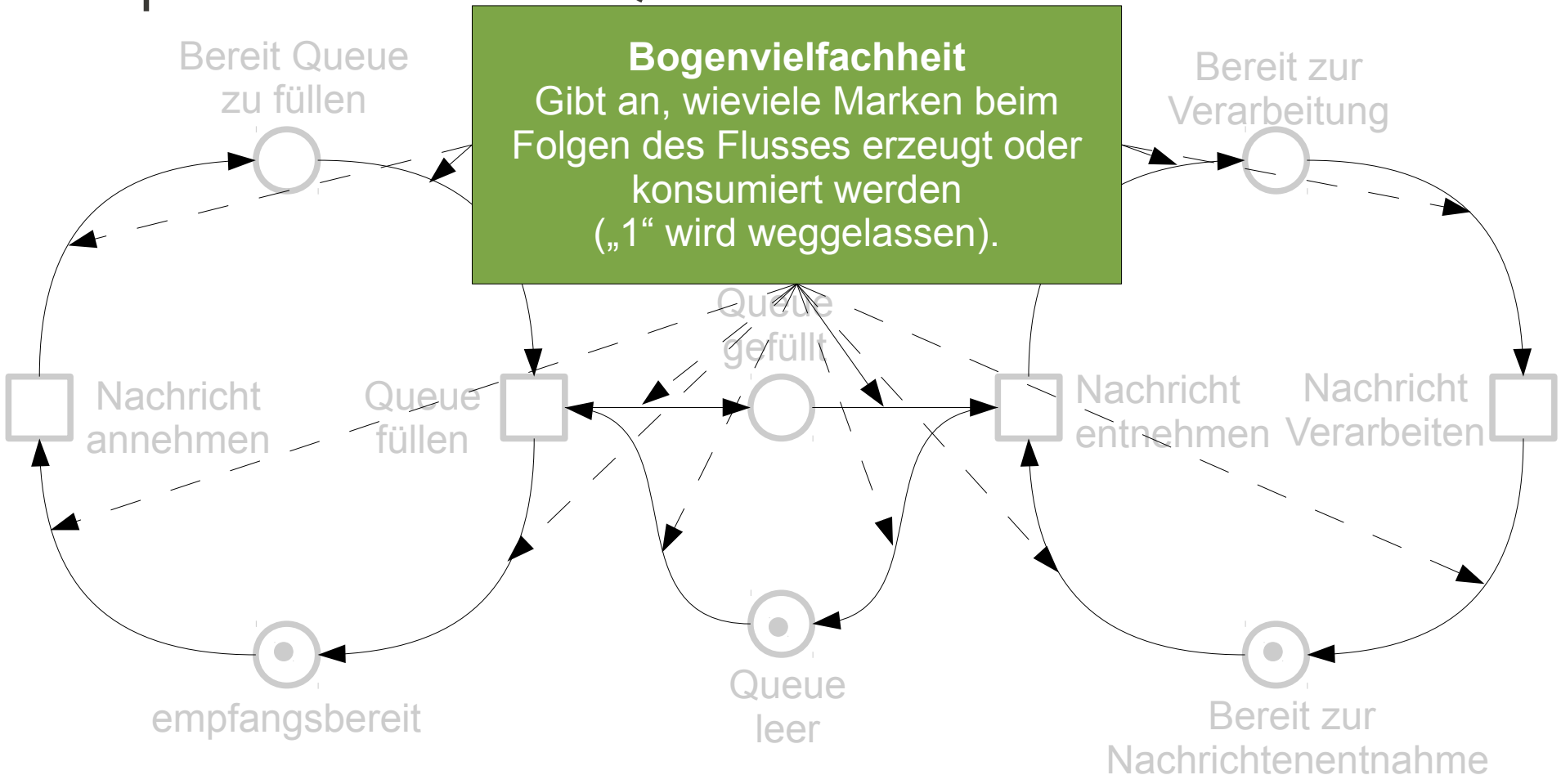
Bogen (erzeugend)

- Beispiel Nachrichten-Queue



Bogenvielfachheit

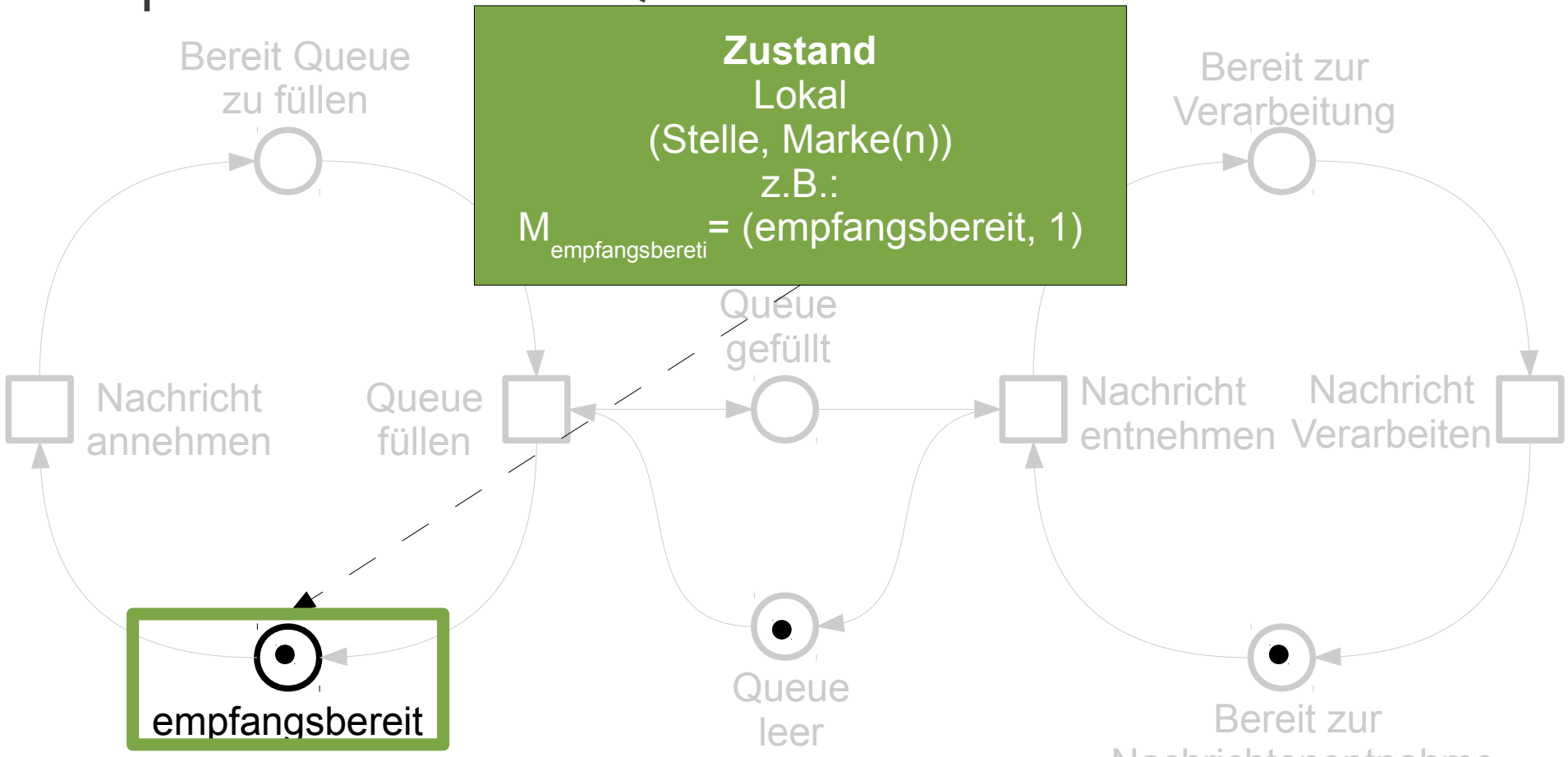
- Beispiel Nachrichten-Queue



Bogenvielfachheit
Gibt an, wieviele Marken beim Folgen des Flusses erzeugt oder konsumiert werden („1“ wird weggelassen).

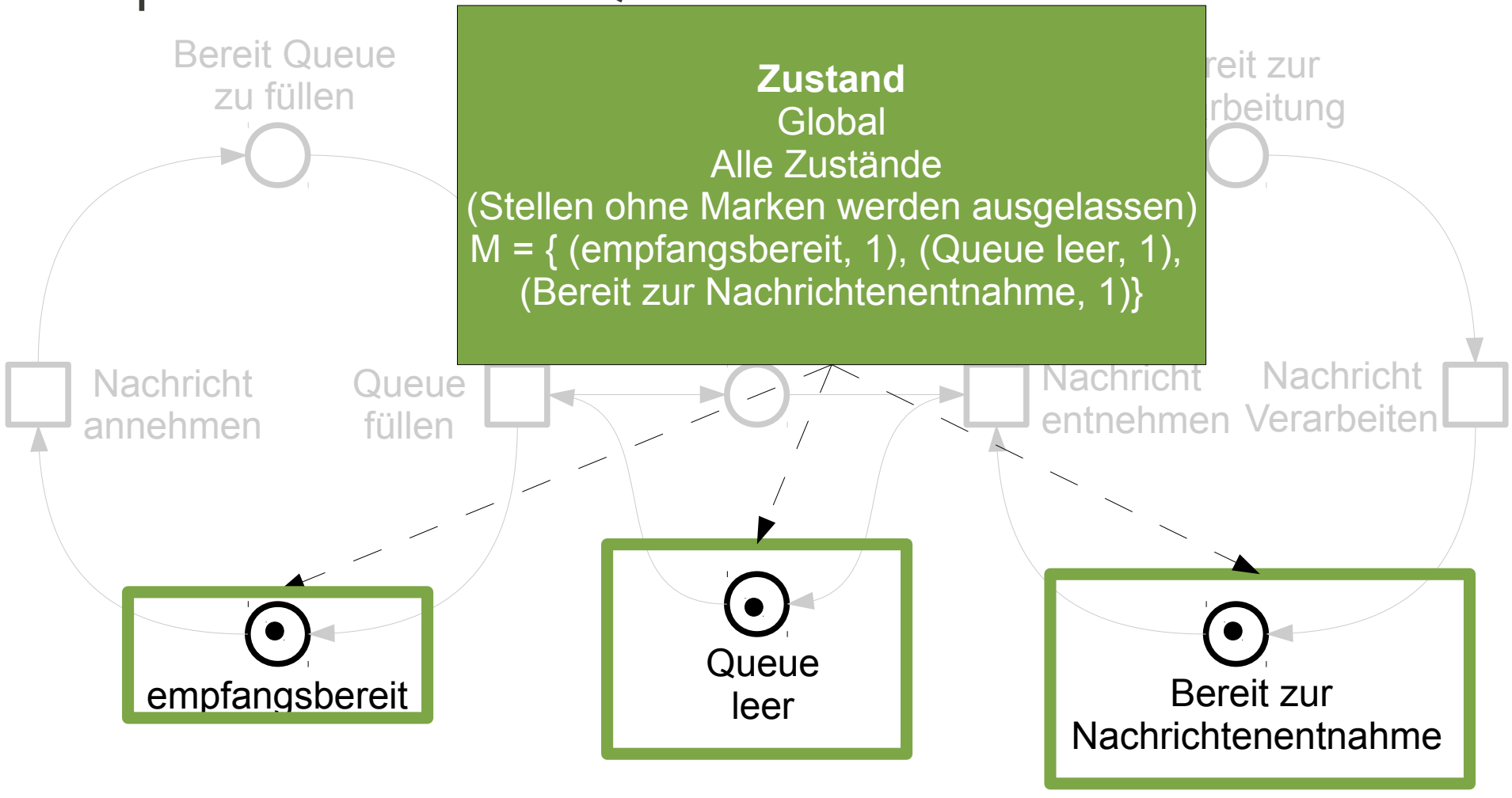
Zustand (lokal)

- Beispiel Nachrichten-Queue



Zustand (global)

- Beispiel Nachrichten-Queue



Zustand
Global
Alle Zustände
(Stellen ohne Marken werden ausgelassen)
M = { (empfangsbereit, 1), (Queue leer, 1),
(Bereit zur Nachrichtenentnahme, 1)}

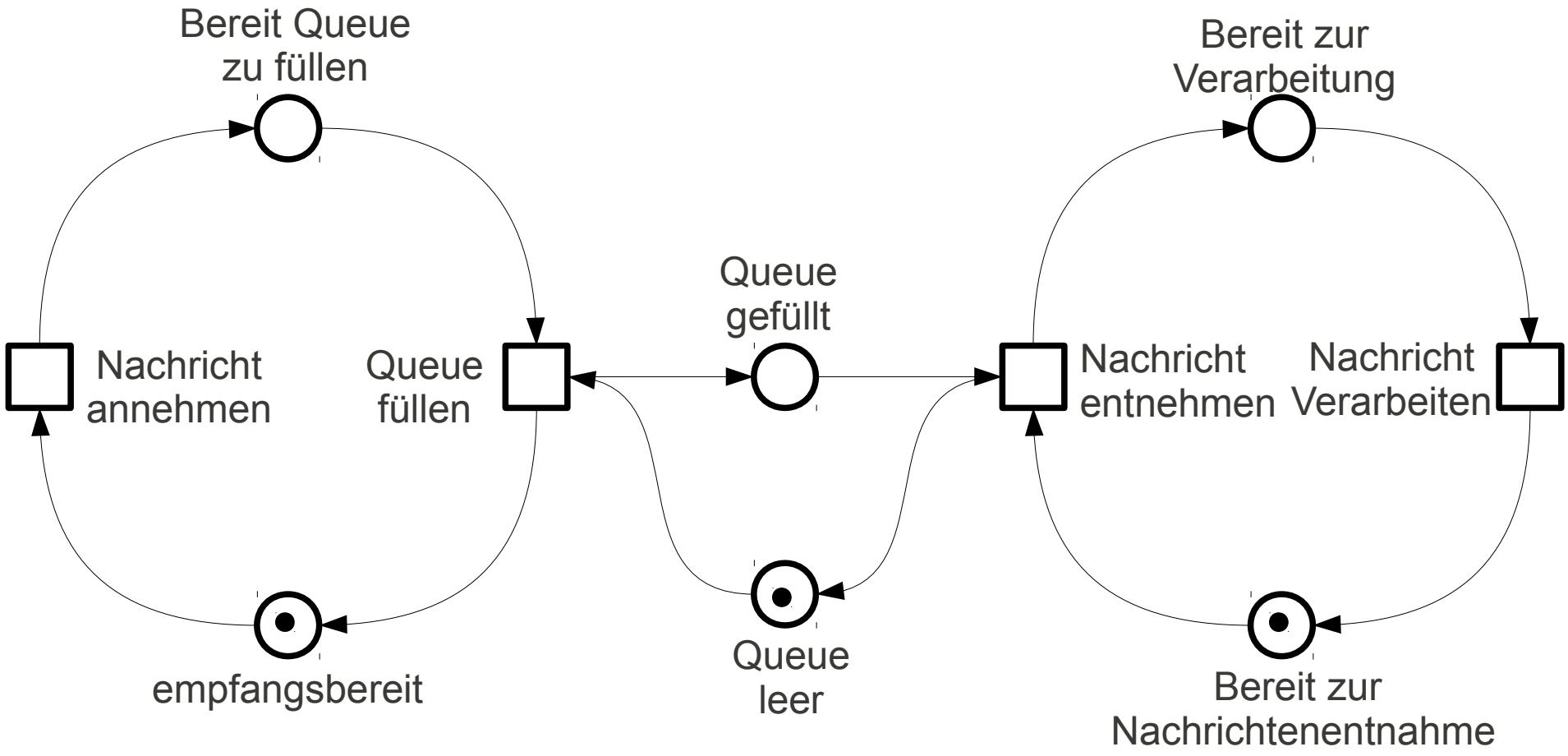
Definition Petri-Netz

Definition Petri-Netz:

- Gegeben:
  - Netz (S, T, F)
  - W – Bogenvielfachheit
    - $W : F \rightarrow \mathbb{N} \setminus 0$
  - M_0 – Globaler Startzustand
    - $M_0 : P \rightarrow \mathbb{N}$
- dann: (S,T,F,W,M_0) ein Petri Netz

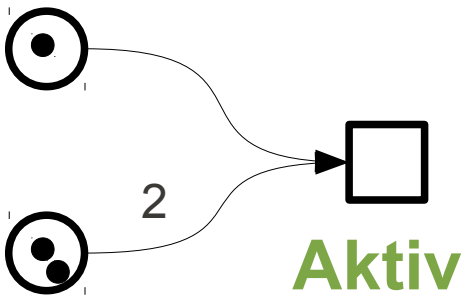
Petri-Netz Ablauf: Beispiel

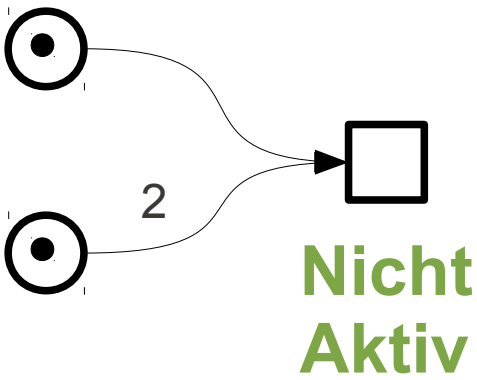
- Beispiel Nachrichten-Queue



Ablauf: Aktivierung von Transition

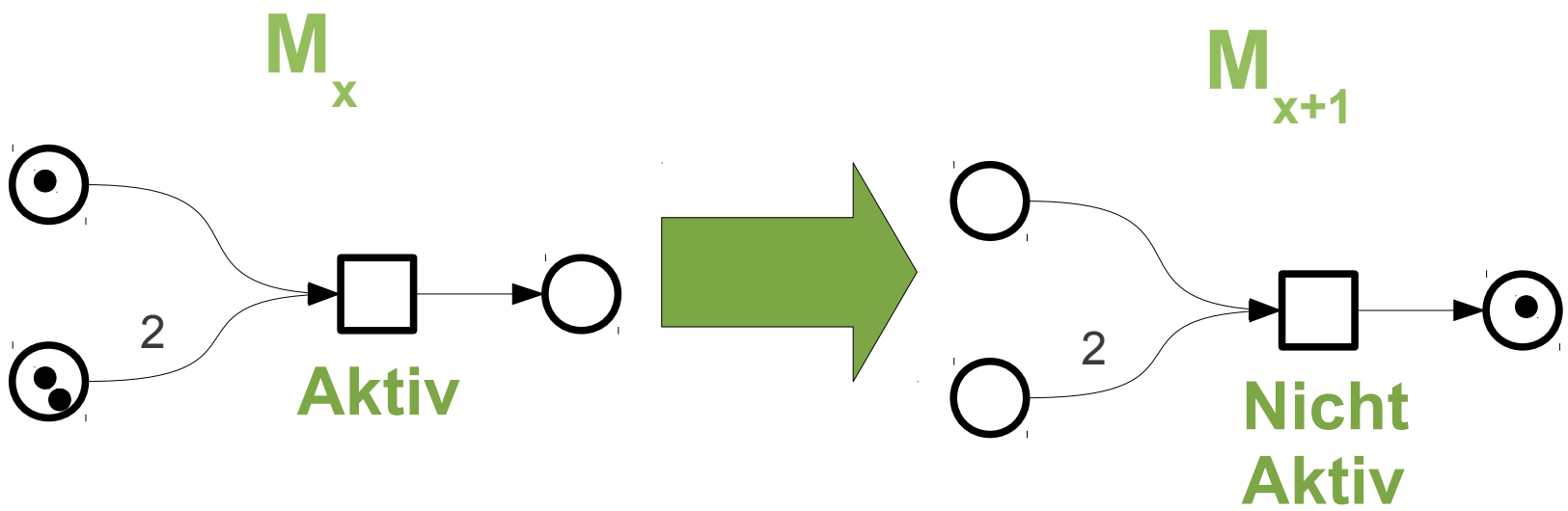
- Ablauf
  - Transition t ist aktiviert, wenn:
    - $\forall\ p \in Vorgänger\ von\ t : W((p,t)) \leqslant m_p$





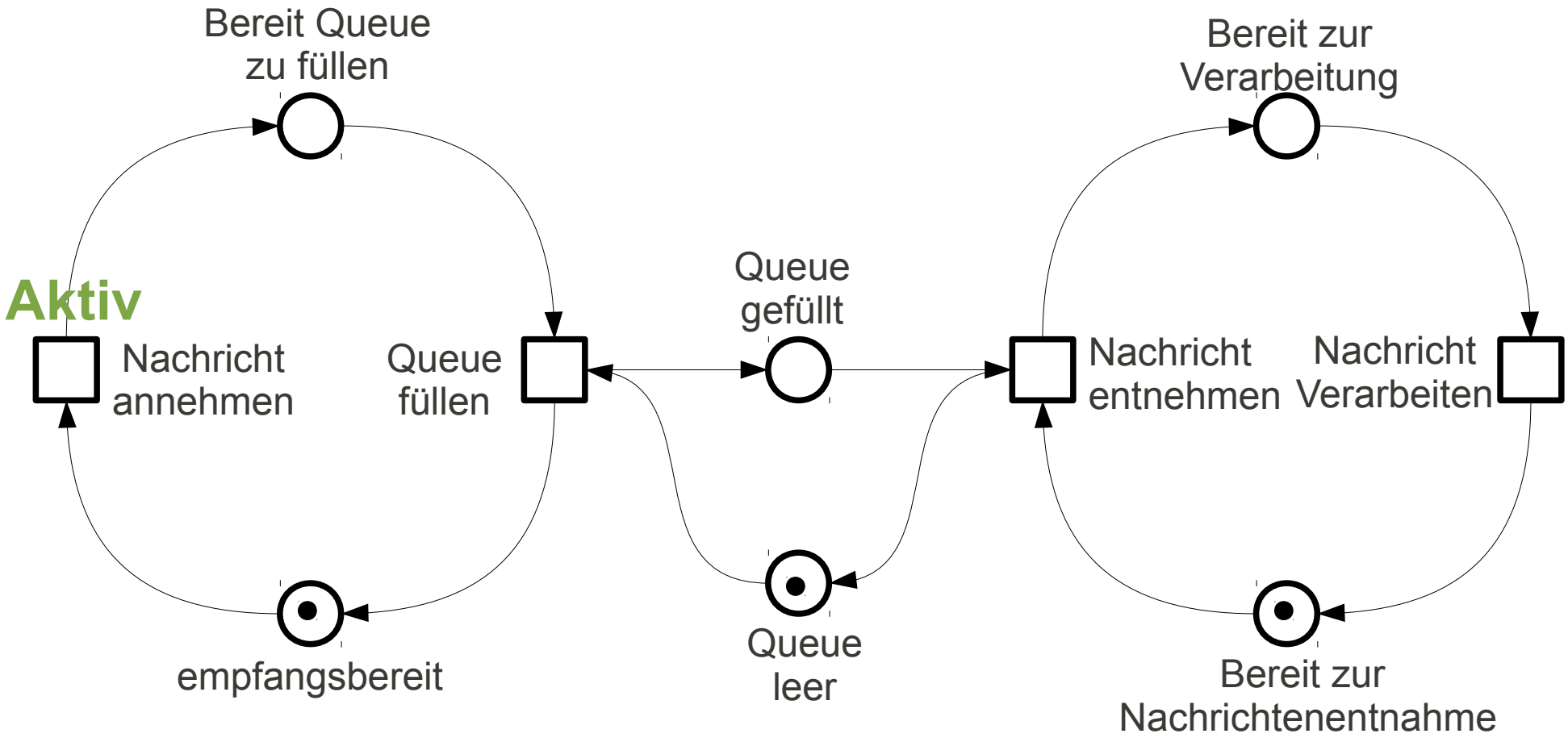
Ablauf: Schalten von Transition

- Ablauf
  - Transition t ist aktiviert, wenn:
    - $\forall\ p \in Vorgänger\ von\ t : W((p,t)) \leqslant m_p$
  - Alle aktivierten t werden beim Übergang M_x nach M_{x+1} geschaltet.



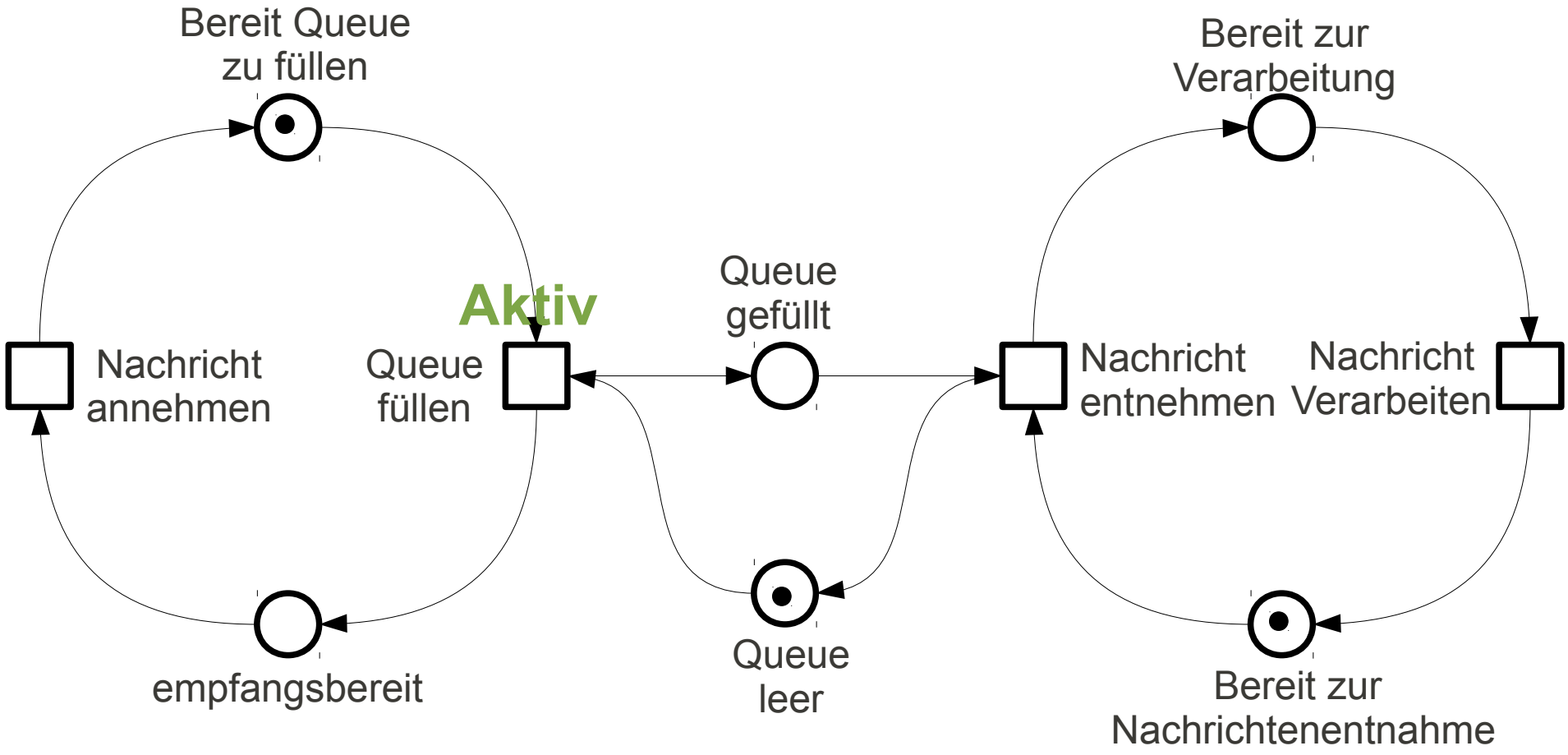
Ablauf: Beispiel (M_0)

- Beispiel Nachrichten-Queue: M_0



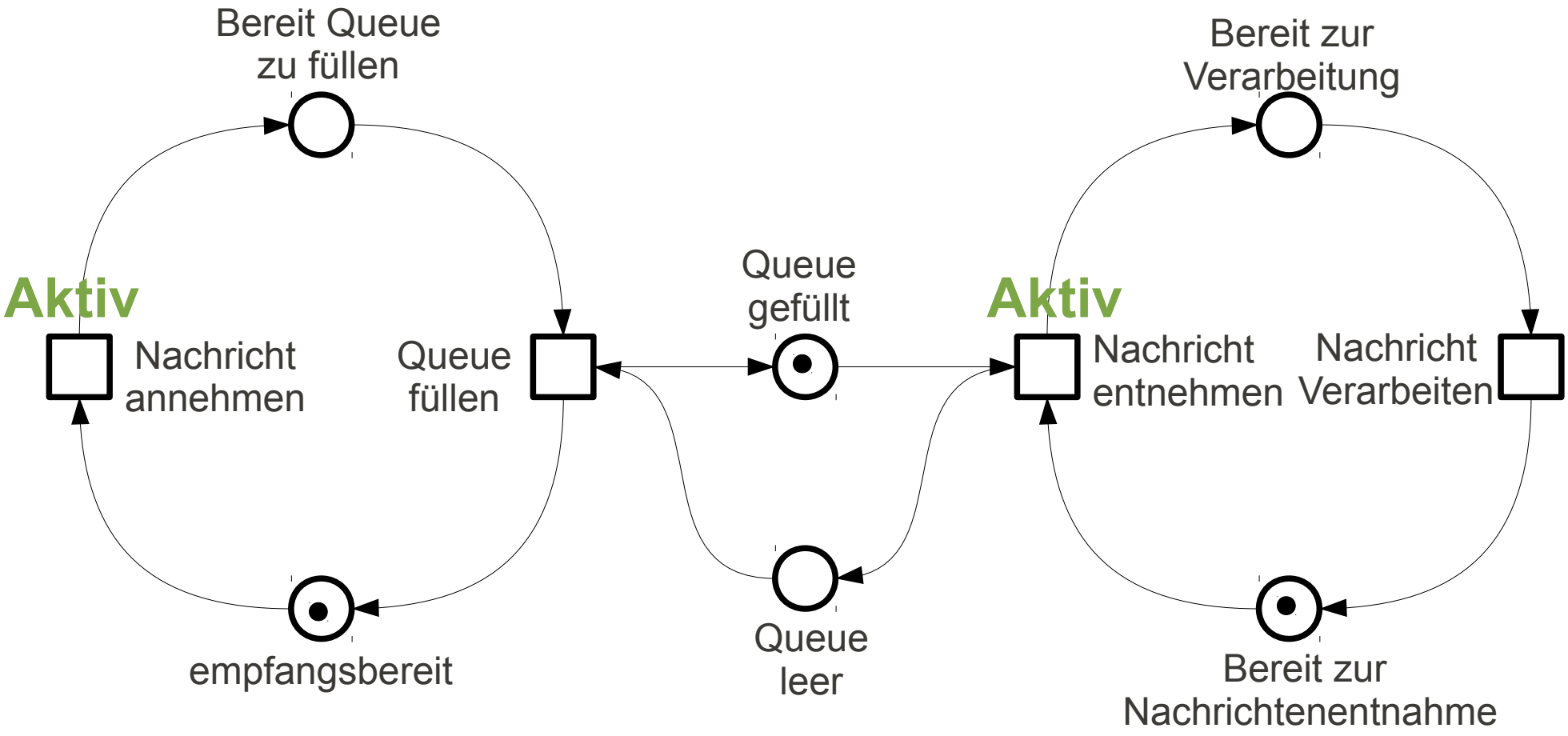
Ablauf: Beispiel (M_1)

- Beispiel Nachrichten-Queue: M_1



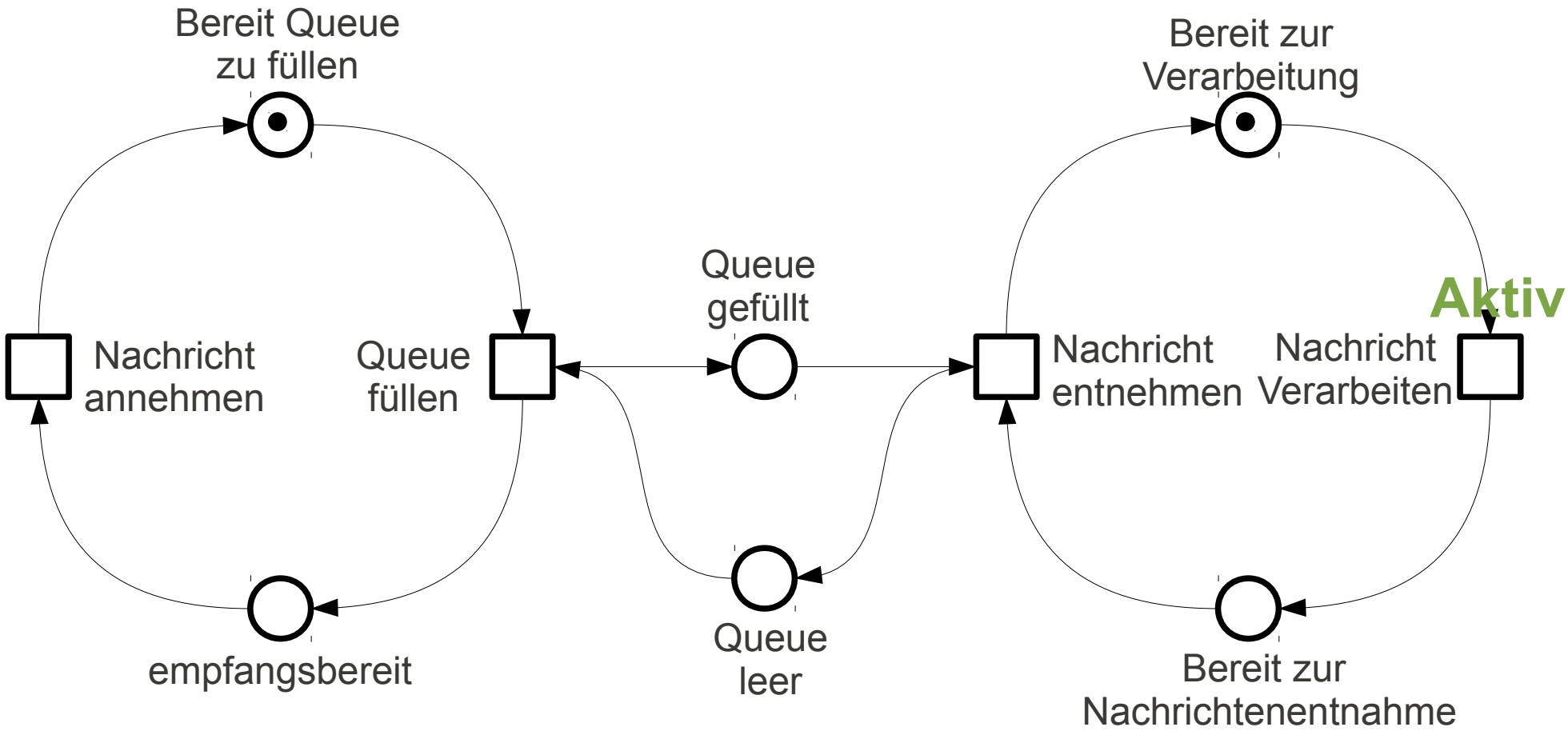
Ablauf: Beispiel (M_2)

- Beispiel Nachrichten-Queue: M_2



Ablauf: Beispiel (M_3)

- Beispiel Nachrichten-Queue: M_3



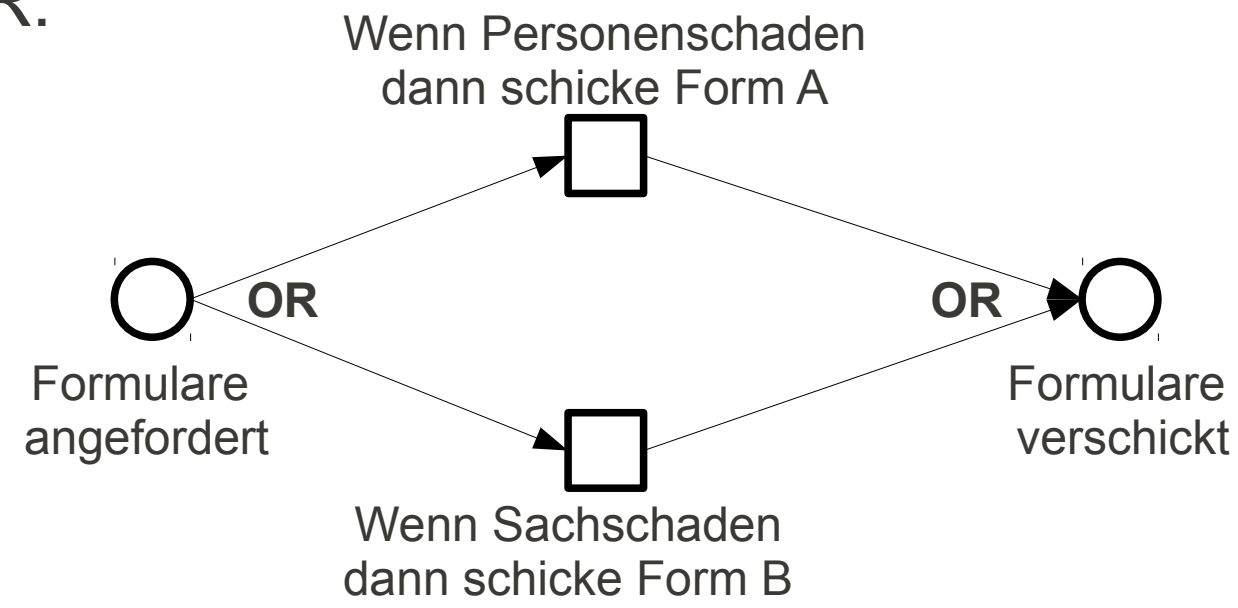
Frage

- Was ist untypisch an der Nachrichten-Queue, wie sie bisher modelliert wurde?

Implizite Verzweigungen

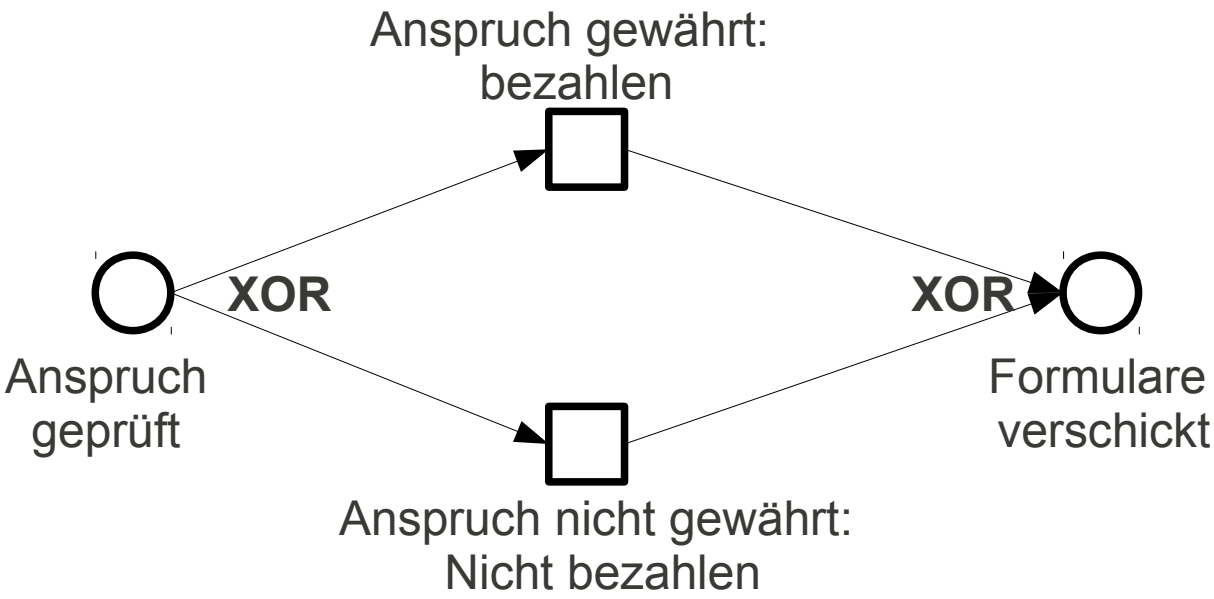
- Gewählter Bogen hängt von Zieltransitionen ab
- Annotation mit AND/XOR/OR kann weggelassen werden, da Folgetransitionen den Fluss vorgeben und interpretiert werden müssen.

Beispiel OR:



Implizite Verzweigungen

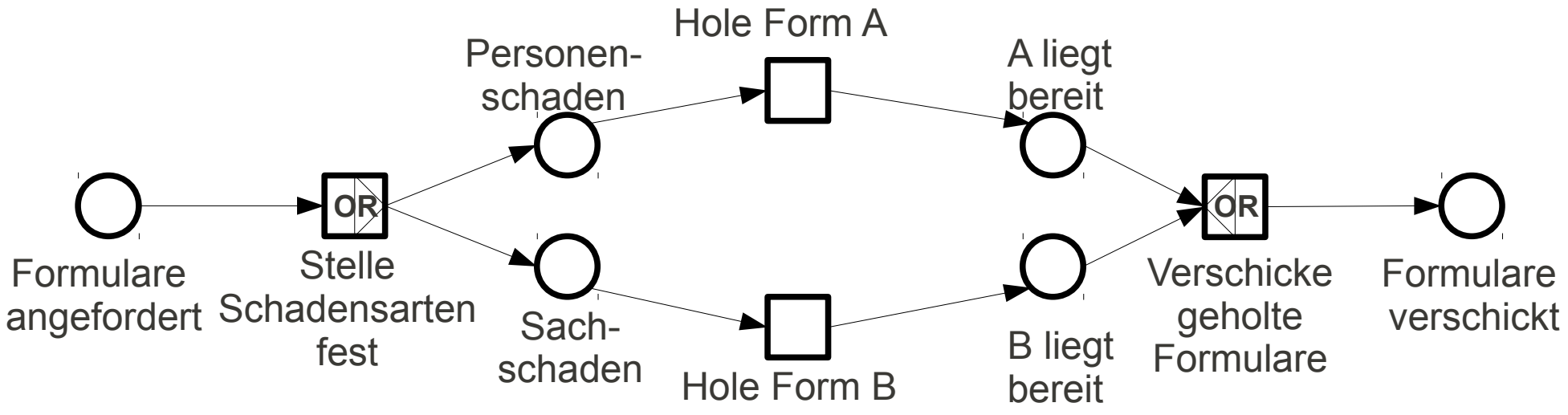
Beispiel XOR:



Explizite Verzweigung vs. Entscheidungstransition: Beispiel 1

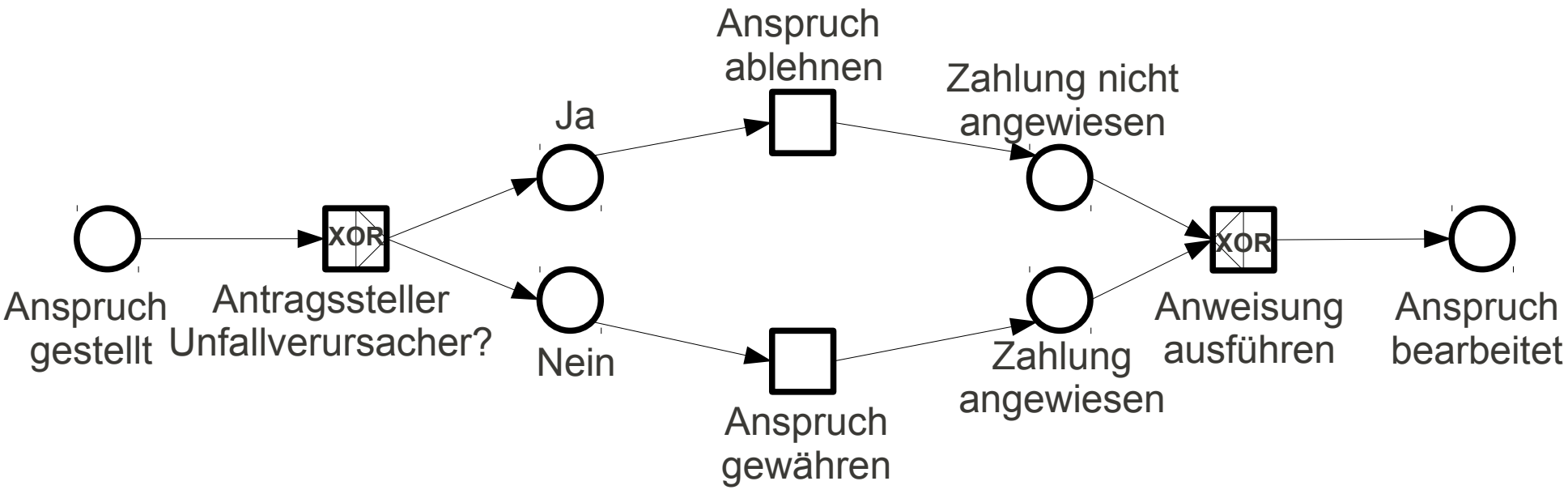
- Gewählter Bogen hängt von Entscheidungstransition ab
- Transition, in der die Entscheidung fällt, die Art der Entscheidung (AND/OR/XOR), sowie alle möglichen Entscheidungen als Nachfolgestellen müssen modelliert werden.

Beispiel OR:



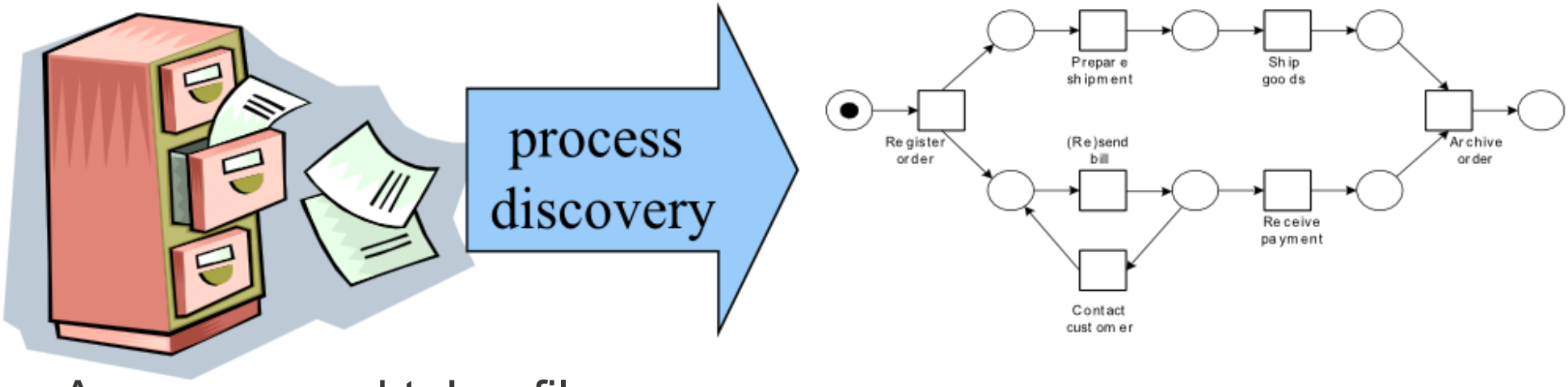
Explizite Verzweigung vs. Entscheidungstransition: Beispiel 2

Beispiel XOR:



Process-Discovery
Zur Erinnerung

- Idee: Umdrehen des traditionellen Ansatzes

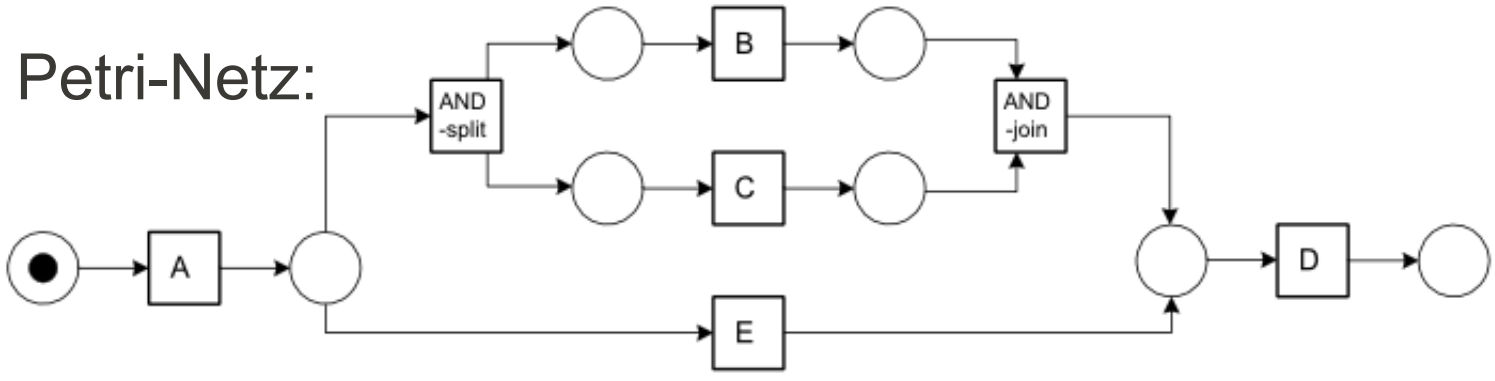


- Ausgangspunkt: Logfile
  - Minimal: Fall-Nummer, Task, zeitliche Ordnung
  - Optional: Genaue Zeit, User, assoziierte Daten etc.
  - „Rauschfrei“
  - Vollständig“
- Ergebnis: Workflow-Schema als Petri-Netz

Beispiel-Logfile

- Das Beispiel enthält die folgenden Sequenzen:
  - ABCD (case 1, case 3) [cases=Prozessinstanzen]
  - ACBD (case 2, case 4)
  - AED (case 5)
- Vereinfachte Darstellung:
  - {ABCD, ACBD, AED}

Petri-Netz:



Log-Datei:

```
case 1, task A
case 2, task A
case 3, task A
case 3, task B
case 1, task B
case 1, task C
case 2, task C
case 4, task A
case 2, task B
case 2, task D
case 5, task A
case 4, task C
case 1, task D
case 3, task C
case 3, task D
case 4, task B
case 5, task E
case 5, task D
case 4, task D
```

Beispiel-Logfile

- Das Beispiel enthält die folgenden Sequenzen:
  - ABCD (case 1, case 3)
  - ACBD (case 2, case 4)
  - AED (case 5)
- Vereinfachte Darstellung:
  - {ABCD, ACBD, AED}

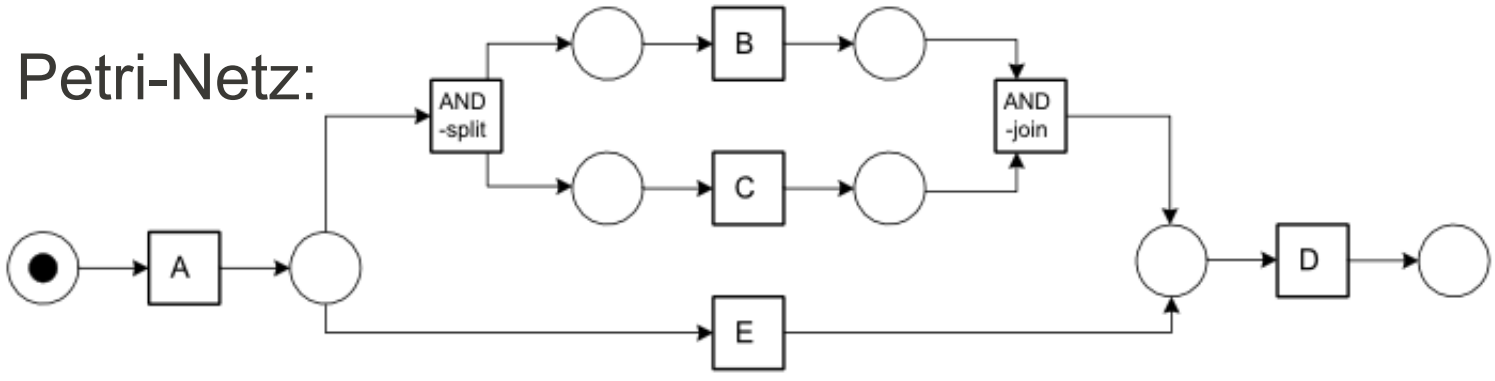
[cases=Prozessinstanzen]

Die generierten Logeinträge müssen immer eindeutig einer Prozessinstanz zugeordnet werden können, um die Sequenzen ableiten zu können.

Log-Datei:

```
case 1, task A
case 2, task A
case 3, task A
case 3, task B
case 1, task B
case 1, task C
case 2, task C
case 4, task A
case 2, task B
case 2, task D
case 5, task A
case 4, task C
case 1, task D
case 5, task C
case 3, task D
case 4, task B
case 5, task E
case 5, task D
case 4, task D
```

Petri-Netz:



Anmerkungen

Anmerkung: Für theoretische Ergebnisse ist Zugehörigkeit zu Case nicht relevant, daher wird im Folgenden davon abstrahiert; für weitergehende Analysen ist sie durchaus wichtig.

Annahmen:

- Bezeichnungen der Transitionen unterschiedlich (immer in Petri-Nets).
- Keine Bogen-Vielfachheiten, Iterationen, Bedingungen an Verzweigungen.
- Verzweigungen nicht beobachtbar.

Ordnungs-Relationen

Für Menge W von Sequenzen σ gilt:

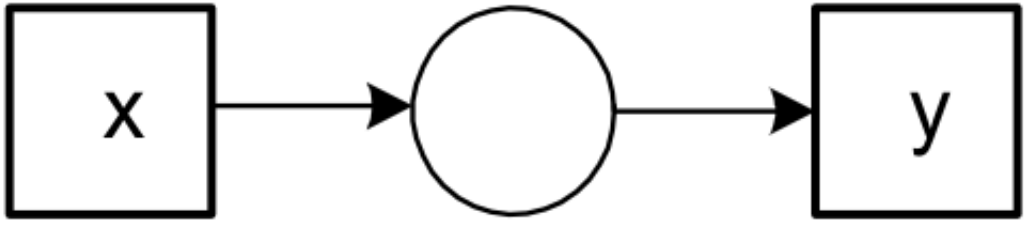
- Direkte Nachfolge: **x>y**
  $\exists\ \sigma \in W$ worin y folgt direkt auf x
- Kausalität: **x→y**
  Gdw. x>y und nicht y>x
- Parallelität: **x||y**
  Gdw. x>y und y>x
- Unabhängigkeit: **x#y**
  Gdw. nicht x>y und nicht y>x
  und x≠y

[Jeweils relativ zu W; Index hier weggelassen.]

- Im Beispiel
  {ABCD, ACBD, AED}:
- A>B, B>C, C>D, A>C, C>B, B>D, A>E, E>D
- A→B, A→C, A→E, B→D, C→D, E→D
- B||C (und symm.)
- A#D, B#E, C#E (und symm.)

Idee: Erzeugung eines Petri-Netzes (1)

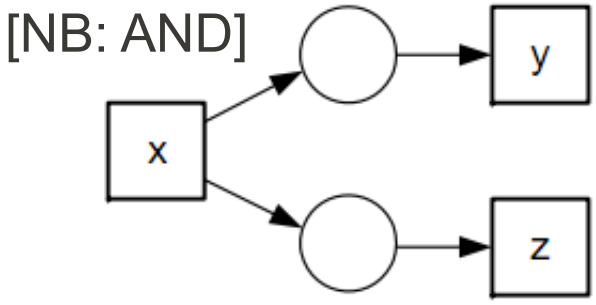
- Wie kann aus den Relationen ein Workflow-Schema abgeleitet werden?
  - Einfachster Fall: Kausalität



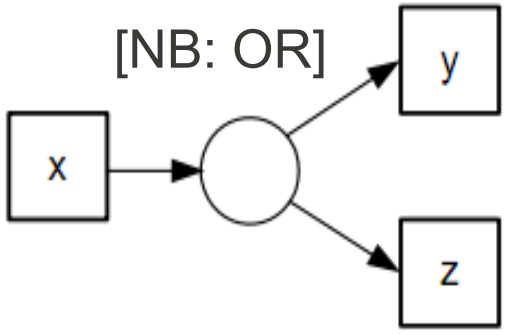
$x \rightarrow y$

Frage: Korrekt ?

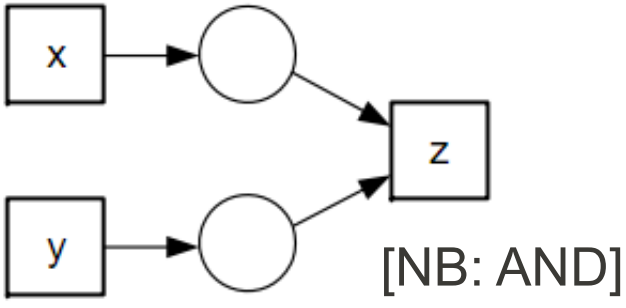
Idee: Erzeugung eines Petri-Netzes (2)



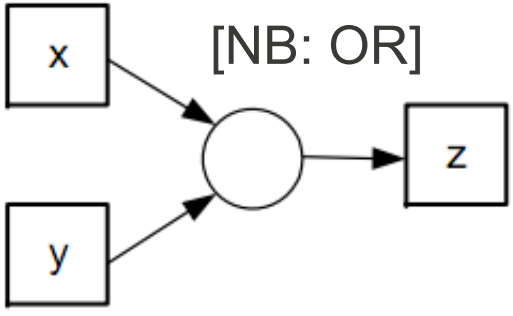
$x \rightarrow y,\ \ x \rightarrow z,\ y || z$



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



$x \rightarrow z,\ \ y \rightarrow z,\ x || y$



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

Frage: Korrekt ?

Formalisierung: Der α-Algorithmus

Geg.: Workflow-Log W = Menge von workflow traces σ (z.B. ABCD)

Gesucht: Workflow-Schema als Petri-Netz

$$\alpha(W) = (\text{Stellen } P_W, \text{ Transitionen } T_W, \text{ Verbindungen } F_W)$$

1. $T_W = \{t \in T \mid \exists_{\sigma \in W}\, t \in \sigma\}$, Transitionen (= Vereinigung der Transitionsmengen der workflow traces)
2. $T_I = \{t \in T \mid \exists_{\sigma \in W}\, t = first(\sigma)\}$, Start-Transitionen
3. $T_O = \{t \in T \mid \exists_{\sigma \in W}\, t = last(\sigma)\}$, End-Transitionen

Formalisierung: Der α-Algorithmus

4. $X_W = \{(A,B) \mid A \subseteq T_W \wedge B \subseteq T_W \wedge \forall_{a \in A} \forall_{b \in B} a \rightarrow_W b \wedge \forall_{a_1, a_2 \in A}^{\neq} a_1 \#_W a_2 \wedge \forall_{b_1, b_2 \in B}^{\neq} b_1 \#_W b_2\}$,

„verallgemeinerte Kausalitätsrelationen“
[NB: ({a},{b}) $\in X_W$ für alle a→b]
Kandidaten für Stellen, um Paare von kausal aufeinanderfolgenden Transitionen zu verbinden (geht gleichzeitig wegen Unabhängigkeit).

5. $Y_W = \{(A,B) \in X_W \mid \forall_{(A',B') \in X_W} A \subseteq A' \wedge B \subseteq B' \Rightarrow (A,B) = (A',B')\}$,

„Maximale Kausalitätsrelationen“. Bringen Minimalität des Petri-Netzes: Minimale Anzahl von Stellen, die notwendig sind, um Paare von kausal aufeinanderfolgenden Transitionen zu verbinden.

6. $P_W = \{(p_{(A,B)}) \mid (A,B) \in Y_W\} \cup \{i_W, o_W\}$,

Stellen: Jede maximale Kausalitätsrelation definiert genau eine Stelle. Zusätzlich: i_W, o_W: eine Start- und eine Endstelle

7. $F_W = \{(a, p_{(A,B)}) \mid (A,B) \in Y_W \wedge a \in A\} \cup \{p_{(A,B)}, b \mid (A,B) \in Y_W \wedge b \in B\} \cup \{(i_W, t) \mid t \in T_I\} \cup \{(t, o_W) \mid t \in T_O\}$, and

Verbindungen: Die durch die Kausalitätsrelationen definierten Verbindungen. Zusätzlich Verbindungen zwischen Start- bzw. Endstellen und Start- bzw. Endtransitionen.

8. $\alpha(W) = (P_W, T_W, F_W)$.

Petri-Netz

Frage: Korrektheit ?

α-Algorithmus – Beispiel

1. T_W = {A,B,C,D,E}
2. T_I = {A}
3. T_O = {D}
4. X_W = {({A}, {B}), ({B}, {D}), ({C},{D}), ({E}, {D}), ({A}, {E {B, E}, {D}), ({C, E}, {D}) }
5. Y_W = {({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}), ({C, E},{D})}

Ursprungs-Log:
{ABCD, ACBD, AED}

$$T_W = \{t \in T \mid \exists_{\sigma \in W}\, t \in \sigma\}$$

Extrahiere alle Transitionen aus dem Log

α-Algorithmus – Beispiel

1. T_W = {A,B,C,D,E}

2. T_I = {A}

3. T_O = {D}

4. X_W = {({A}, {B}, ({B}, {D}), ({C},{D}), ({E}, {D}), ({A}, {B, E}, {D}), ({C, E}, {D}) }

5. Y_W = {({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}), ({C, E},{D})}

Ursprungs-Log:
{ABCD, ACBD, AED}

$$T_W = \{t \in T \mid \exists_{\sigma \in W}\, t \in \sigma\}$$

Extrahiere alle Transitionen aus dem Log

α-Algorithmus – Beispiel

1. T_W = {A,B,C,D,E}

2. T_I = {A}

3. T_O = {D}

4. X_W = {({A}, {B}), ({B}, {D}), ({C},{D}), ({E}, {D}), ({A}, {E {B, E}, {D}), ({C, E}, {D}) }

5. Y_W = {({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}), ({C, E},{D})}

Ursprungs-Log:
{ABCD, ACBD, AED}

$$T_W = \{t \in T \mid \exists_{\sigma \in W}\, t \in \sigma\}$$

Extrahiere alle Transitionen aus dem Log

α-Algorithmus – Beispiel

Ursprungs-Log:
{ABCD, ACBD, AED}

1. T_W = {A,B,C,D,E}
2. T_I = {A}
3. T_O = {D}
4. X_W = {({A}, {B}), ({B}, {D}), ({C},{D}), ({E}, {D}), ({A}, {E {B, E}, {D}), ({C, E}, {D}) }
5. Y_W = {({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}), ({C, E},{D})}

$$T_I = \{ t \in T \mid \exists_{\sigma \in W}\, t = first(\sigma) \}$$

Sammle Starttransitionen aus den einzelnen Logs

α-Algorithmus – Beispiel

Ursprungs-Log:
{ABCD, ACBD, AED}

1. T_W = {A,B,C,D,E}
2. T_I = {A}
3. T_O = {D}
4. X_W = {({A}, {B}), ({B}, {D}), ({C},{D}), ({E}, {D}), ({A}, {E {B, E}, {D}), ({C, E}, {D}) }
5. Y_W = {({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}), ({C, E},{D})}

$$T_I = \{ t \in T \mid \exists_{\sigma \in W}\, t = first(\sigma) \}$$

Sammle Starttransitionen aus den einzelnen Logs

α-Algorithmus – Beispiel

Ursprungs-Log:
{ABCD, ACBD, AED}

1. T_W = {A,B,C,D,E}
2. T_I = {A}
3. T_O = {D},

$$T_O = \{t \in T \mid \exists_{\sigma \in W}\, t = last(\sigma)\}$$

Sammle Endtransitionen aus den einzelnen Logs

4. X_W = {({A}, {B}), ({B}, {D}), ({C},{D}), ({E}, {D}), ({A}, {E {B, E}, {D}), ({C, E}, {D}) }
5. Y_W = {({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}), ({C, E},{D})}

α-Algorithmus – Beispiel

Ursprungs-Log:
{ABCD, ACBD, AED}

1. T_W = {A,B,C,D,E}
2. T_I = {A}
3. T_O = {D}

$$T_O = \{t \in T \mid \exists_{\sigma \in W}\, t = last(\sigma)\}$$

Sammle Endtransitionen aus den einzelnen Logs

4. X_W = {({A}, {B}), ({B}, {D}), ({C},{D}), ({E}, {D}), ({A}, {E {B, E}, {D}), ({C, E}, {D}) }
5. Y_W = {({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}), ({C, E},{D})}

α-Algorithmus – Beispiel

1. $T_W = \{A,B$

2. $T_I = \{A\}$

3. $T_O = \{D\}$

Bilde Tuple möglicher Flüsse.

Ursprungs-Log:
{ABCD, ACBD, AED}

4. $X_W = \{(\{A\}, \{B\}), (\{A\}, \{C\}), (\{A\}, \{E\}), (\{B\}, \{D\}), (\{C\},\{D\}), (\{E\}, \{D\}), (\{A\}, \{B, E\}), (\{A\}, \{C, E\}), (\{B, E\}, \{D\}), (\{C, E\}, \{D\})\}$

5. $Y_W = \{(\{A\}, \{B, E\}), (\{A\}, \{C, E\}), ($

$(\{C, E\},\{D\})\}$

$$X_W = \{(A,B) \mid A \subseteq T_W \wedge B \subseteq T_W \wedge \forall_{a \in A} \forall_{b \in B} a \rightarrow_W b \wedge \forall_{a_1, a_2 \in A} a_1 \#_W a_2 \wedge \forall_{b_1, b_2 \in B} b_1 \#_W b_2\}$$

α-Algorithmus – Beispiel

Bilde Tuple möglicher Flüsse:
Bilde Submengen mit unabhängigen Elemente
als Kandidaten für Mengen *A* und *B*
(in Def. rechts unten auf dieser Folie)

Ursprungs-Log:
{ABCD, ACBD, AED}

3. T_O = {D}

4. X_W = {({A}, {B}), ({A}, {C}), ({A}, {E}), ({B}, {D}), ({C},{D}), ({E}, {D}), ({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}), ({C, E}, {D}) }

5. Y_W = {({A}, {B, E}), ({A}, {C, E}), ({C, E},{D})}

$$X_W = \{(A,B) \mid A \subseteq T_W \wedge B \subseteq T_W \wedge \forall_{a \in A} \forall_{b \in B} a \rightarrow_W b \wedge \forall_{a_1, a_2 \in A} a_1 \#_W a_2 \wedge \forall_{b_1, b_2 \in B} b_1 \#_W b_2\}$$

α-Algorithmus – Beispiel

1. T_W = {A,B,C,D,E}

2.

3. T_O = {D}

Bilde Tupel möglicher Flüsse:
Bilde Submengen mit unabhängigen Elemente als Kandidaten für Mengen *A* und *B*
{A},{A,D}

Ursprungs-Log:
{ABCD, ACBD, AED}

Alle andere Transitionen Abhängig bis auf D

4. X_W = {({A}, {B}), ({A}, {C}), ({A}, {E}), ({B}, {D}), ({C},{D}), ({E}, {D}), ({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}), ({C, E}, {D}) }

5. Y_W = {({A}, {B, E}), ({A}, {C, E}), (({C, E},{D})}

$$X_W = \{(A,B) \mid A \subseteq T_W \wedge B \subseteq T_W \wedge \forall_{a \in A} \forall_{b \in B}\, a \rightarrow_W b \wedge \boxed{\forall_{a_1, a_2 \in A}\, a_1 \#_W a_2 \wedge \forall_{b_1, b_2 \in B}\, b_1 \#_W b_2}\}$$

α-Algorithmus – Beispiel

Bilde Tupel möglicher Flüsse:
Bilde Submengen mit unabhängigen Elemente
als Kandidaten für Mengen *A* und *B*
{A},{A,D},{B},{B, E}

Ursprungs-Log:
{ABCD, ACBD, AED}

A,C,D abhängig
E unabhängig

3. T_O = {D}

4. X_W = {({A}, {B}), ({A}, {C}), ({A}, {E}), ({B}, {D}), ({C},{D}), ({E}, {D}), ({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}), ({C, E}, {D}) }

5. Y_W = {({A}, {B, E}), ({A}, {C, E}), ({A}, ({C, E},{D}))}

$$X_W = \{(A,B) \mid A \subseteq T_W \wedge B \subseteq T_W \wedge \forall_{a \in A} \forall_{b \in B} a \rightarrow_W b \wedge \forall_{a_1, a_2 \in A} a_1 \#_W a_2 \wedge \forall_{b_1, b_2 \in B} b_1 \#_W b_2\}$$

α-Algorithmus – Beispiel

Bilde Tupel möglicher Flüsse:
Bilde Submengen mit unabhängigen Elemente
als Kandidaten für Mengen *A* und *B*
{A},{A,D},{B},{B, E},{C},{C,E},{D},{E}

Ursprungs-Log:
{ABCD, ACBD, AED}

{A,D} nicht relevant
obwohl unabhängig,
weil kein gemeinsamer
Vorgänger / Nachfolger

3. T_O = {B}

4. X_W = {({A}, {B}), ({A}, {C}), ({A}, {E}), ({B}, {D}), ({C},{D}), ({E}, {D}), ({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}), ({C, E}, {D}) }

5. Y_W = {({A}, {B, E}), ({A}, {C, E}), (({C, E},{D})}

$$X_W = \{(A,B) \mid A \subseteq T_W \wedge B \subseteq T_W \wedge \forall_{a \in A} \forall_{b \in B}\, a \rightarrow_W b \wedge \boxed{\forall_{a_1, a_2 \in A}\, a_1 \#_W a_2 \wedge \forall_{b_1, b_2 \in B}\, b_1 \#_W b_2}\}$$

α-Algorithmus – Beispiel

Bilde Tupel möglicher Flüsse:
Bilde Tupel der Submengen, sodass Menge *A* nur kausale Vorgänger der Elemente in Menge *B* enthält
{A},{A,D},{B},{B, E},{C},{C,E},{D},{E}

Ursprungs-Log:
{ABCD, ACBD, AED}

3. T_O = {D}

4. X_W = {({A}, {B}), ({A}, {C}), ({A}, {E}), ({B}, {D}), ({C},{D}), ({E}, {D}), ({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}), ({C, E}, {D}) }

5. Y_W = {({A}, {B, E}), ({A}, {C, E}), ({C, E},{D})}

$$X_W = \{(A,B) \mid A \subseteq T_W \wedge B \subseteq T_W \wedge \forall_{a \in A} \forall_{b \in B} a \rightarrow_W b \wedge \forall_{a_1, a_2 \in A} a_1 \#_W a_2 \wedge \forall_{b_1, b_2 \in B} b_1 \#_W b_2\}$$

α-Algorithmus – Beispiel

Bilde Tupel möglicher Flüsse:
Bilde Tupel der Submengen, sodass Menge *A* nur kausaler Vorgänger der Elemente in Menge *B* enthält
{A},{A,D},{B},{B, E},{C},{C, E},{D}

Ursprungs-Log:
{ABCD, ACBD, AED}

{C} nicht wegen BC und CB: Abfolge nicht eindeutig

3. T_O = {D}

4. X_W = {({A}, {B}), ({A}, {C}), ({A}, {E}), ({B}, {D}), ({C},{D}), ({E}, {D}), ({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}), ({C, E}, {D}) }

5. Y_W = {({A}, {B, E}), ({A}, {C, E}), ({C, E},{D})}

$$X_W = \{(A,B) \mid A \subseteq T_W \wedge B \subseteq T_W \wedge \forall_{a \in A} \forall_{b \in B} a \rightarrow_W b \wedge \forall_{a_1, a_2 \in A} a_1 \#_W a_2 \wedge \forall_{b_1, b_2 \in B} b_1 \#_W b_2\}$$

α-Algorithmus – Beispiel

Bilde Tupel möglicher Flüsse

Ursprungs-Log:
{ABCD, ACBD, AED}

1

2

3. T_O = {D}

4. X_W = {({A}, {B}), ({A}, {C}), ({A}, {E}), ({B}, {D}), ({C},{D}), ({E}, {D}), ({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}), ({C, E}, {D}) }

5. Y_W = {({A}, {B, E}), ({A}, {C, E}), ({C, E},{D})}

$$X_W = \{(A,B) \mid A \subseteq T_W \wedge B \subseteq T_W \wedge \forall_{a \in A} \forall_{b \in B} a \rightarrow_W b \wedge \forall_{a_1, a_2 \in A} a_1 \#_W a_2 \wedge \forall_{b_1, b_2 \in B} b_1 \#_W b_2\}$$

α-Algorithmus – Beispiel

Ursprungs-Log:
{ABCD, ACBD, AED}

1. $T_W = \{A,B,C,D,E\}$
2. $T_I = \{A\}$
3. $T_O = \{D\}$
4. $X_W = \{(\{A\}, \{B\}), (\{A\}, \{C\}), (\{A\}, \{E\}), (\{B\}, \{D\}), (\{C\},\{D\}), (\{E\}, \{D\}), (\{A\}, \{B, E\}), (\{A\}, \{C, E\}), (\{B, E\}, \{D\}), (\{C, E\}, \{D\}) \}$
5. $Y_W = \{(\{A\}, \{B, E\}), (\{A\}, \{C, E\}), (\{B, E\}, \{D\}), (\{C, E\},\{D\})\}$

$$Y_W = \{(A,B) \in X_W \mid \forall_{(A',B') \in X_W} A \subseteq A' \wedge B \subseteq B' \Rightarrow (A,B) = (A',B')\}$$

Entferne alle Tupel aus X_W, die Teilmenge eines anderen Tupels sind.

α-Algorithmus – Beispiel

Ursprungs-Log:
{ABCD, ACBD, AED}

1. T_W = {A,B,C,D,E}
2. T_I = {A}
3. T_O = {D}
4. X_W = {~~({A}, {B}),~~ ({A}, {C}), ~~({A}, {E}),~~ ({B}, {D}), ({C},{D}), ({E}, {D}), ({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}), ({C, E}, {D}) }
5. Y_W = {({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}), ({C, E},{D})}

$$Y_W = \{(A,B) \in X_W \mid \forall_{(A',B') \in X_W} A \subseteq A' \wedge B \subseteq B' \Rightarrow (A,B) = (A',B')\}$$

Entferne alle Tupel aus X_W, die Teilmenge eines anderen Tupels sind.

α-Algorithmus – Beispiel

Ursprungs-Log:
{ABCD, ACBD, AED}

1. T_W = {A,B,C,D,E}
2. T_I = {A}
3. T_O = {D}
4. X_W = {~~({A}, {B}), ({A}, {C}), ({A}, {E}), ({B}, {D}), ({C},{D}), ({E}, {D})~~, ({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}), ({C, E}, {D}) }
5. Y_W = {({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}), ({C, E},{D})}

$$Y_W = \{(A,B) \in X_W \mid \forall_{(A',B') \in X_W} A \subseteq A' \wedge B \subseteq B' \Rightarrow (A,B) = (A',B')\}$$

Entferne alle Tupel aus X_W, die Teilmenge eines anderen Tupels sind.

α-Algorithmus – Beispiel

Ursprungs-Log:
{ABCD, ACBD, AED}

1. $T_W = \{A,B,C,D,E\}$
2. $T_I = \{A\}$
3. $T_O = \{D\}$
4. $X_W = \{(\{A\}, \{B\}), (\{A\}, \{C\}), (\{A\}, \{E\}), (\{B\}, \{D\}), (\{C\},\{D\}), (\{E\}, \{D\}), (\{A\}, \{B, E\}), (\{A\}, \{C, E\}), (\{B, E\}, \{D\}), (\{C, E\}, \{D\}) \}$
5. $Y_W = \{(\{A\}, \{B, E\}), (\{A\}, \{C, E\}), (\{B, E\}, \{D\}), (\{C, E\},\{D\})\}$

α-Algorithmus – Beispiel

5. Y_W = {({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}), ({C, E},{D})}

6. P_W = {i_W, o_W, p({A},{B,E}), p{A},{C,E}), p({B,E},{D}), p({C,E},{D})},

7. F_W = {(i_W, A), (A, p({A},{B,E})), (p({A},{B,E}), B), …, (D, o_W)}

8. a(W) = (P_W, T_W, F_W)

$$P_W = \{(p_{(A,B)}) \mid (A,B) \in Y_W\} \cup \{i_W, o_W\}$$

Füge für jedes Tupel eine Stelle, sowie eine Start- und Endstelle ein.

α-Algorithmus – Beispiel

5. Y_W = {({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}), ({C, E},{D})}

6. P_W = {i_W, o_W, p({A},{B,E}), p{A},{C,E}), p({B,E},{D}), p({C,E},{D})},

7. F_W = {(i_W, A), (A, p({A},{B,E})), (p({A},{B,E}), B), …, (D, o_W)}

8. a(W) = (P_W, T_W, F_W)

$$P_W = \{(p_{(A,B)}) \mid (A,B) \in Y_W\} \cup \{i_W, o_W\}$$

Füge für jedes Tupel eine Stelle, sowie eine Start- und Endstelle ein.

α-Algorithmus – Beispiel

5. Y_W = {({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}), ({C, E},{D})}

6. P_W = {i_W, o_W, p({A},{B,E}), p{A},{C,E}), p({B,E},{D}), p({C,E},{D})},

7. F_W = {(i_W, A), (A, p({A},{B,E})), (p({A},{B,E}), B), …, (D, o_W)}

8. a(W) = (P_W, T_W, F_W)

$$P_W = \{(p_{(A,B)}) \boxed{(A,B) \in Y_W} \cup \{i_W, o_W\}$$

Füge für jedes Tupel eine Stelle, sowie eine Start- und Endstelle ein.

α-Algorithmus – Beispiel

5. Y_W = {({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}), ({C, E},{D})}

6. P_W = {i_W, o_W, p({A},{B,E}), p{A},{C,E}), p({B,E},{D}), p({C,E},{D})},

7. F_W = {(i_W, A), (A, p({A},{B,E})), (p({A},{B,E}), B), …, (D, o_W)}

8. a(W) = (P_W, T_W, F_W)

$$P_W = \{(p_{(A,B)}) \mid (A,B) \in Y_W\} \cup \{i_W, o_W\}$$

Füge für jedes Tupel eine Stelle, sowie eine Start- und Endstelle ein.

α-Algorithmus – Beispiel

$$F_W = \{(a, p_{(A,B)}) | (A,B) \in Y_W \wedge a \in A\} \cup \{p_{(A,B)}, b | (A,B) \in Y_W \wedge b \in B\} \cup \{(i_W, t) | t \in T_I\} \cup \{(t, o_W) | t \in T_O\}$$

1. T_W = {A,B,C,D,E}

2. T_I = {A} 3. T_O = {D}

5. Y_W = {({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}), ({C, E},{D})}

6. P_W = {i_W, o_W, p({A},{B,E}), p{A},{C,E}), p({B,E},{D}), p({C,E},{D})},

7. F_W = {(i_W, A), (A, p({A},{B,E})), (p({A},{B,E}), B), …, (D, o_W)}

8. a(W) = (P_W, T_W, F_W)

Füge die zugehörigen Bögen ein.

α-Algorithmus – Beispiel

$$F_W = \{(a, p_{(A,B)}) \mid (A,B) \in Y_W \wedge a \in A\} \cup \{p_{(A,B)}, b \mid (A,B) \in Y_W \wedge b \in B\} \cup \boxed{\{(i_W, t) \mid t \in T_I\} \cup \{(t, o_W) \mid t \in T_O\}}$$

1. T_W = {A,B,C,D,E}

2. T_I = {A} 3. T_O = {D}

5. Y_W = {({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}), ({C, E},{D})}

6. P_W = {i_W, o_W, p({A},{B,E}), p{A},{C,E}), p({B,E},{D}), p({C,E},{D})},

7. F_W = {(i_W, A), (A, p({A},{B,E})), (p({A},{B,E}), B), …, (D, o_W)}

8. a(W) = (P_W, T_W, F_W)

Füge die zugehörigen Bögen ein
Verbinde Startstelle mit Starttransitionen
Verbinde Endstelle mit Starttransitionen

α-Algorithmus – Beispiel

$$F_W = \{(a, p_{(A,B)}) \mid (A,B) \in Y_W \wedge a \in A\} \cup \{p_{(A,B)}, b \mid (A,B) \in Y_W \wedge b \in B\} \cup \{(i_W, t) \mid t \in T_I\} \cup \{(t, o_W) \mid t \in T_O\}$$

1. $T_W = \{A,B,C,D,E\}$

2. $T_I = \{A\}$ 3. $T_O = \{D\}$

5. $Y_W = \{(\{A\}, \{B, E\}), (\{A\}, \{C, E\}), (\{B, E\}, \{D\}), (\{C, E\},\{D\})\}$

6. $P_W = \{i_W, o_W, p(\{A\},\{B,E\}), p\{A\},\{C,E\}), p(\{B,E\},\{D\}), p(\{C,E\},\{D\})\}$,

7. $F_W = \{(i_W, A), (A, p(\{A\},\{B,E\})), (p(\{A\},\{B,E\}), B), \ldots, (D, o_W)\}$

8. $a(W) = (P_W, T_W, F_W)$

Füge die zugehörigen Bögen ein
Verbinde Elemente aus A eines Y_W – Tupels mit der zugehörigen Stelle

α-Algorithmus – Beispiel

$F_W = \{(a, p_{(A,B)}) | (A,B) \in Y_W \wedge a \in A\} \cup$
$\{p_{(A,B)}, b | (A,B) \in Y_W \wedge b \in B\} \cup$
$\{(i_W, t) | t \in T_I\} \cup$
$\{(t, o_W) | t \in T_O\}$

1. T_W = {A,B,C,D,E}

2. T_I = {A} 3. T_O = {D}

5. Y_W = {({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}), ({C, E},{D})}

6. P_W = {i_W, o_W, p({A},{B,E}), p{A},{C,E}), p({B,E},{D}), p({C,E},{D})},

7. F_W = {(i_W, A), (A, p({A},{B,E})), (p({A},{B,E}), B), …, (D, o_W)}

8. a(W) = (P_W, T_W, F_W)

Füge die zugehörigen Bögen ein
Verbinde zugehörigen Stelle eines Y_W – Tupels mit den Elementen aus B

α-Algorithmus – Beispiel

$$F_W = \{(a, p_{(A,B)}) | (A,B) \in Y_W \wedge a \in A\} \cup \{p_{(A,B)}, b | (A,B) \in Y_W \wedge b \in B\} \cup \{(i_W, t) | t \in T_I\} \cup \{(t, o_W) | t \in T_O\}$$

1. T_W = {A,B,C,D,E}

2. T_I = {A} 3. T_O = {D}

5. Y_W = {({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}), ({C, E},{D})}

6. P_W = {i_W, o_W, p({A},{B,E}), p{A},{C,E}), p({B,E},{D}), p({C,E},{D})},

7. F_W = {(i_W, A), (A, p({A},{B,E})), (p({A},{B,E}), B), …, (D, o_W)}

8. a(W) = (P_W, T_W, F_W)

Füge die zugehörigen Bögen ein

α-Algorithmus – Beispiel

5. Y_w = {({A}, {B, E}), ({A}, {C, E}), ({B, E}, {D}), ({C, E},{D})}

6. P_w = {i_w, o_w, p({A},{B,E}), p{A},{C,E}), p({B,E},{D}), p({C,E},{D})},

7. F_w = {(i_w, A), (A, p({A},{B,E})), (p({A},{B,E}), B), …, (D, o_w)}

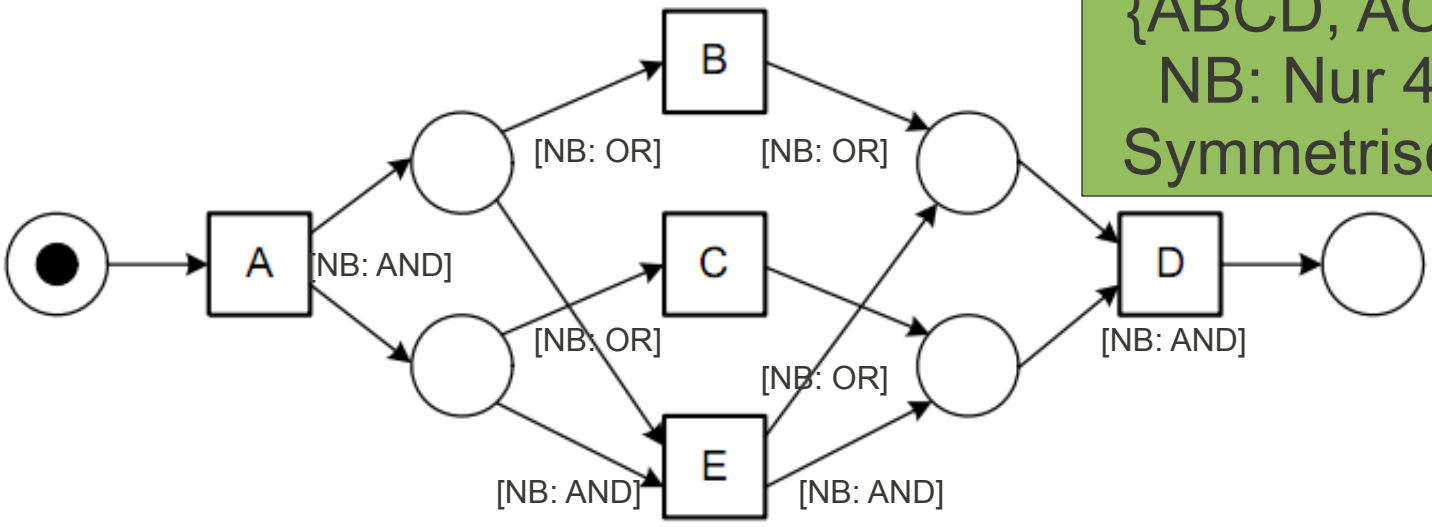
8. a(W) = (P_w, T_w, F_w)

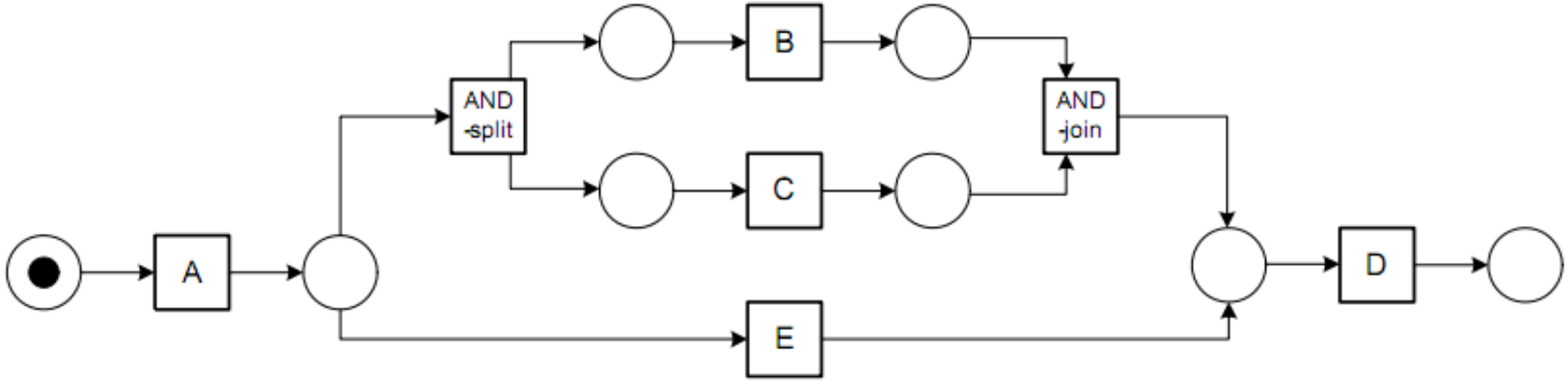
$$\alpha(W) = (P_W, T_W, F_W)$$

Das extrahierte Petri Netz ist nun definiert.

α-Algorithmus – Beispiel-Netz

Ursprungs-Log:
{ABCD, ACBD, AED}
NB: Nur 4 Stellen !
Symmetrisch in AED.



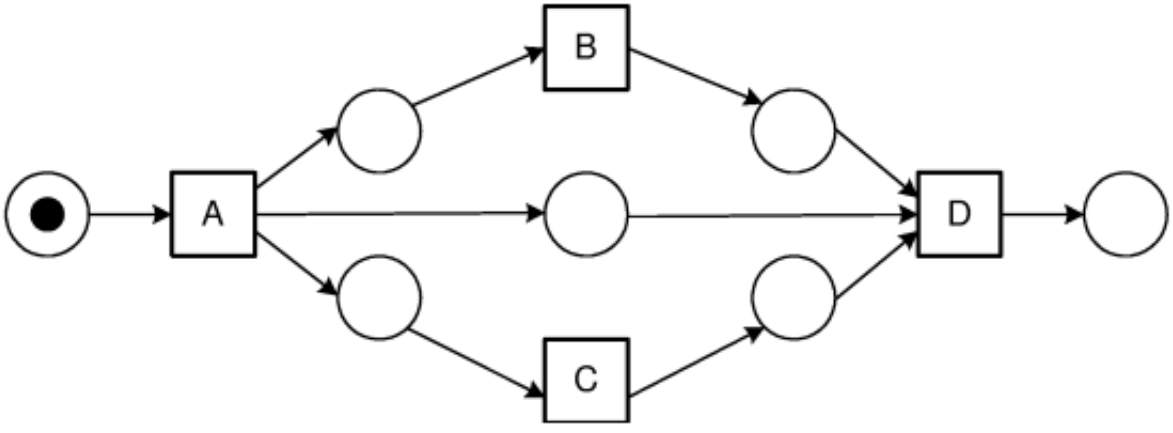


α-Algorithmus Unzulänglichkeiten (1)

- Wie auf der vorherigen Folie gesehen, können Log-Daten im Allgemeinen von mehr als einem Petri-Netz erzeugt werden. Der α-Algorithmus erzeugt das Petri-Netz mit der minimalen Anzahl Stellen. Wie wir ebenfalls gesehen haben, ist es nicht unbedingt das am leichtesten verständliche Petri-Netz.
- Es können nur „beobachtbare“ Netzwerke erzeugt werden.
  - Splits und Joins nicht immer direkt beobachtbar.
  - Beobachtetes Verhalten wird aber korrekt wiedergegeben.

α-Algorithmus Unzulänglichkeiten (1a)

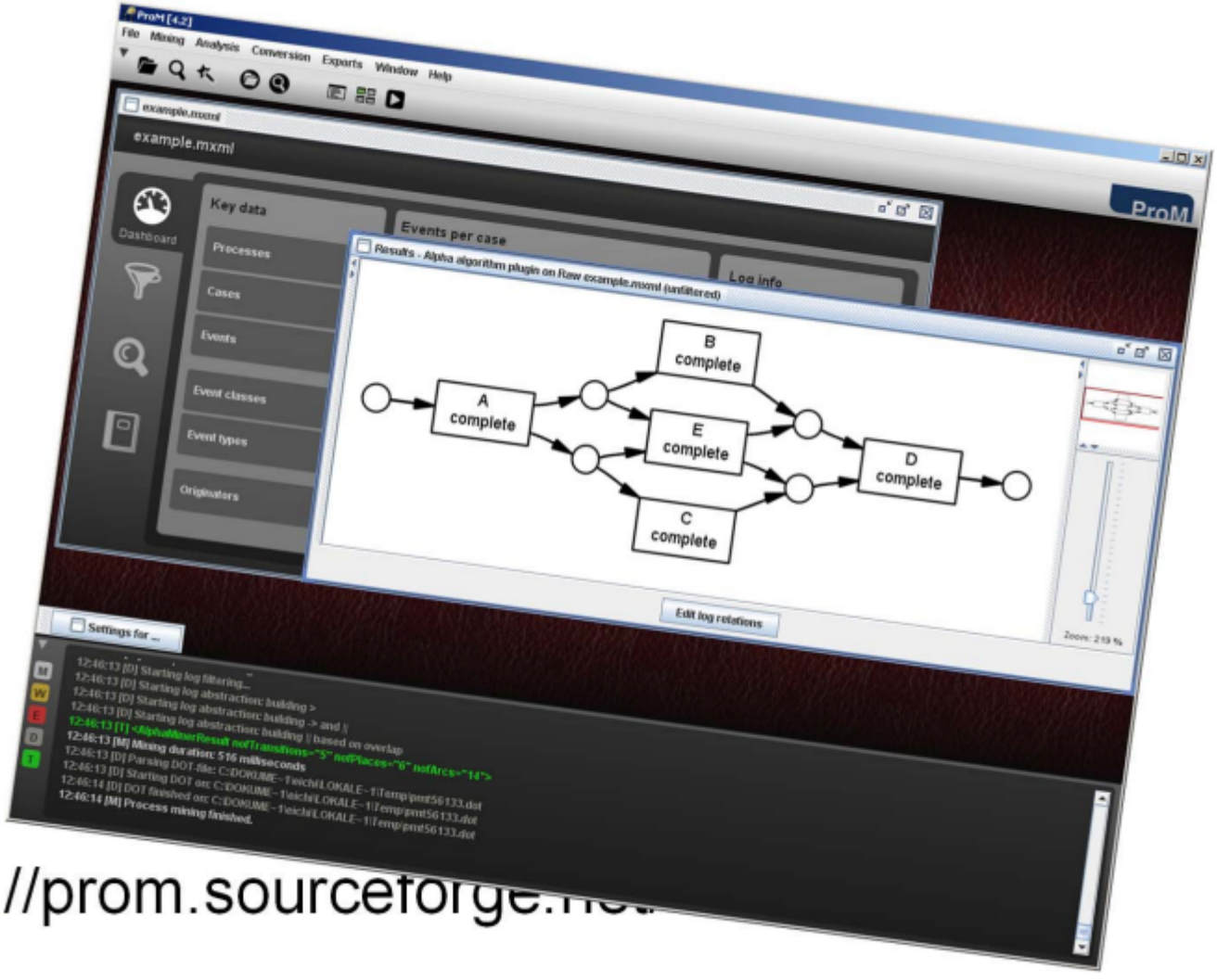
- Eindeutigkeit der Bezeichner: Bei der getroffenen Annahme, dass jeder Transitionsname nur einmal im Petri-Netz auftaucht, ist nicht jede Folge von Transitionen aus einem wohlgeformten Petri-Netz erzeugbar !
  [NB: Falls die Annahme nicht gemacht wird, gibt es immer die triviale Möglichkeit, das Petri-Netz als OR-Verknüpfung über Transitionen, die je eine Sequenz erzeugen, darzustellen.]
- Insbesondere hinsichtlich Abstraktion („unsichtbar machen“) von Transitionen: Wegabstrahieren der Transition E in den Log-Daten im obigen Beispiel für zu Log-Daten, die (unter o.g. Annahme) nicht durch ein Petri-Netz darstellbar sind:



α-Algorithmus Unzulänglichkeiten (2)

- Mehrfach auftretende Aktivitäten
  - Schwierig, der α-Algorithmus konstruiert nur eine Transition pro Aktivität.
- Iterationen
  - Führen zu mehrfach auftretenden Aktivitäten.
  - Müssen ggf. extra erkannt werden.
    - Mitunter sehr schwierig!
- Verrauschte Daten
  - Lösung z.B. durch Betrachtung der Häufigkeit.

Demo: ProM

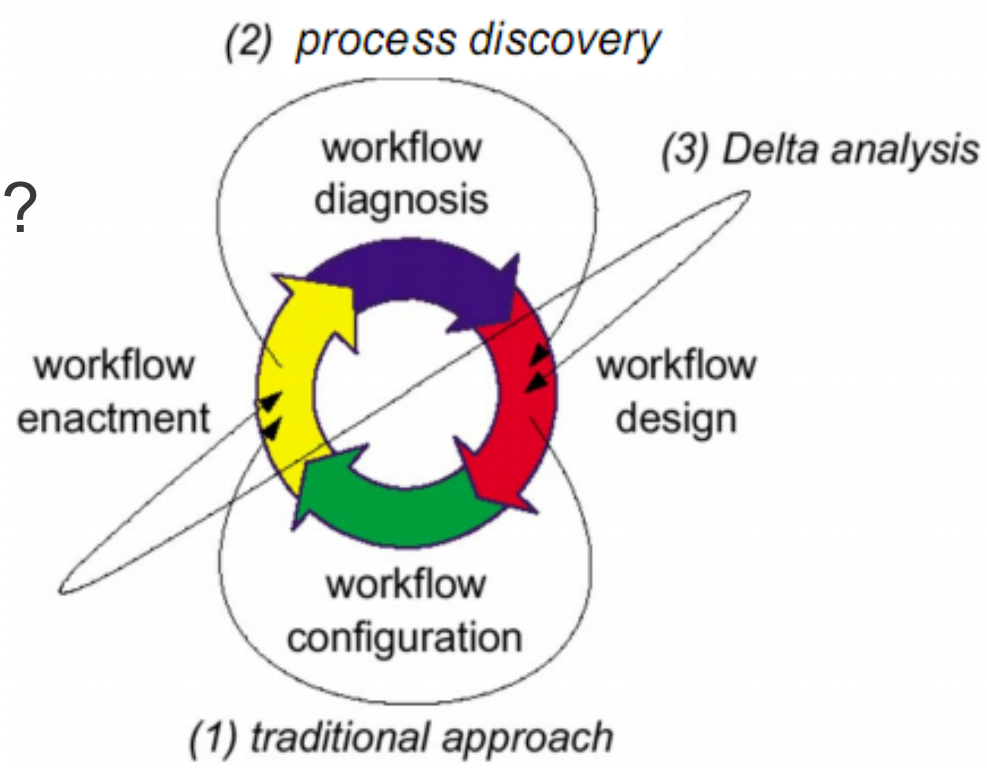


http://prom.sourceforge.net

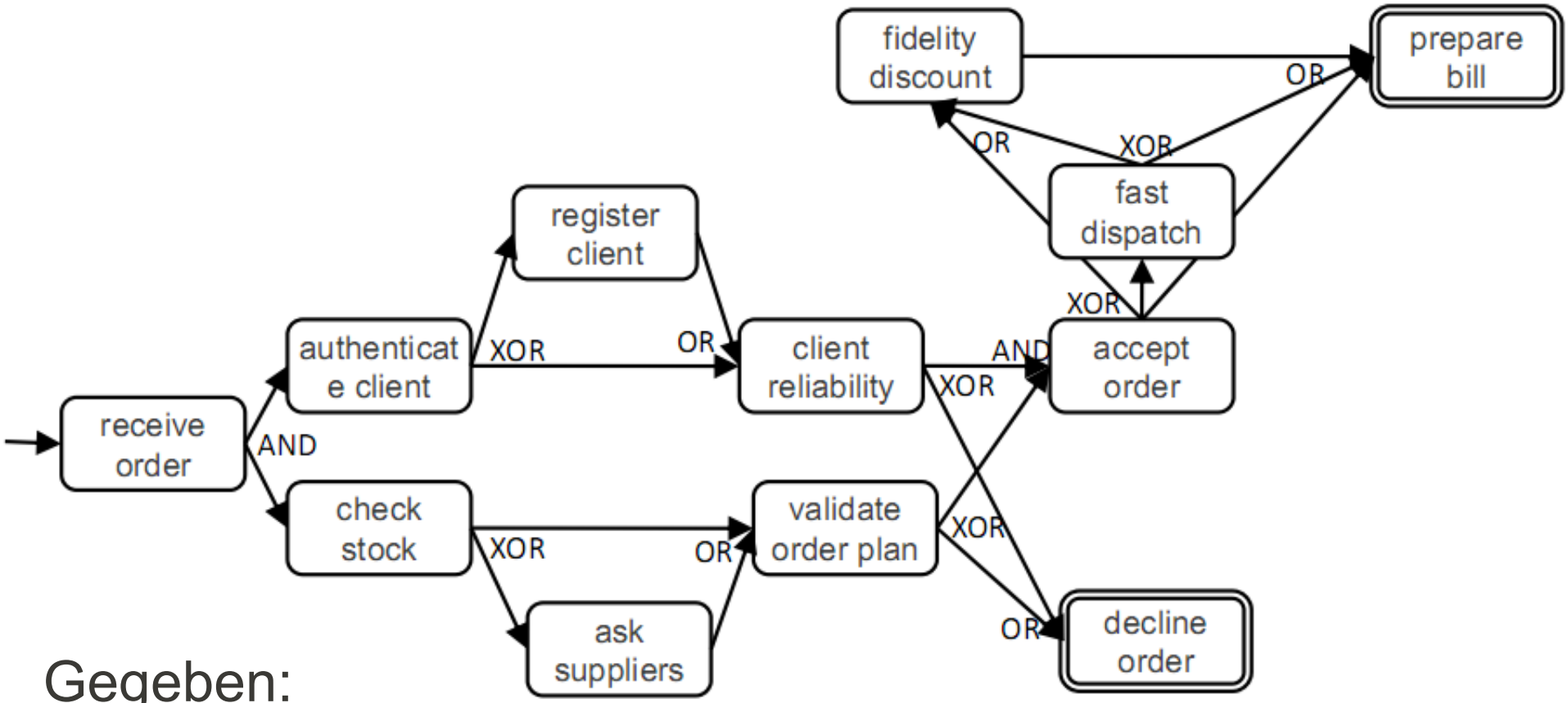
Download: Siehe Links auf Vorlesungs-Webseite.

Workflow-Diagnose

- Nächster Schritt („workflow diagnosis“):
  - Welche Pfade sind häufig?
  - Welche Instanzen werden wahrscheinlich abgebrochen?

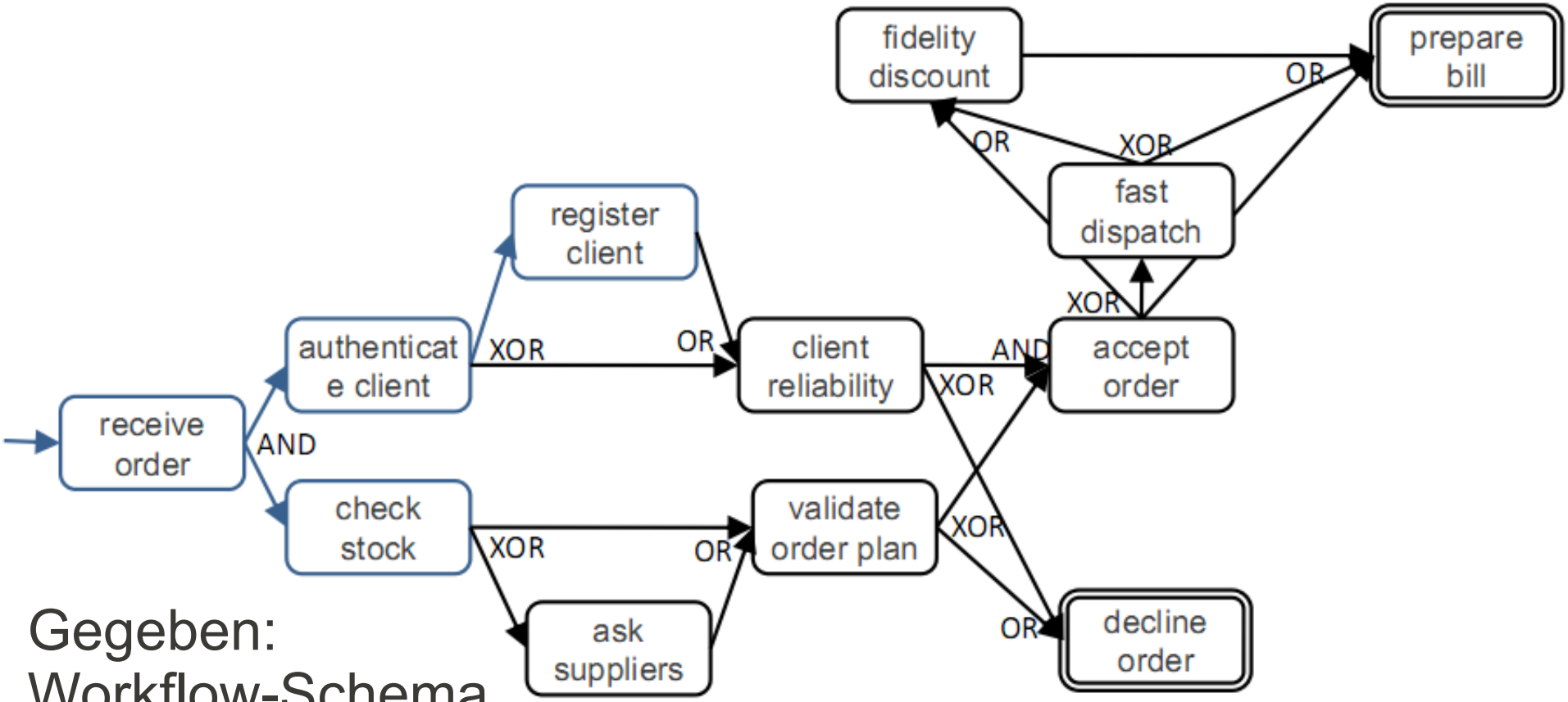


Workflow-Diagnose mit Graph Mining (1)



Gegeben:
Workflow-Schema

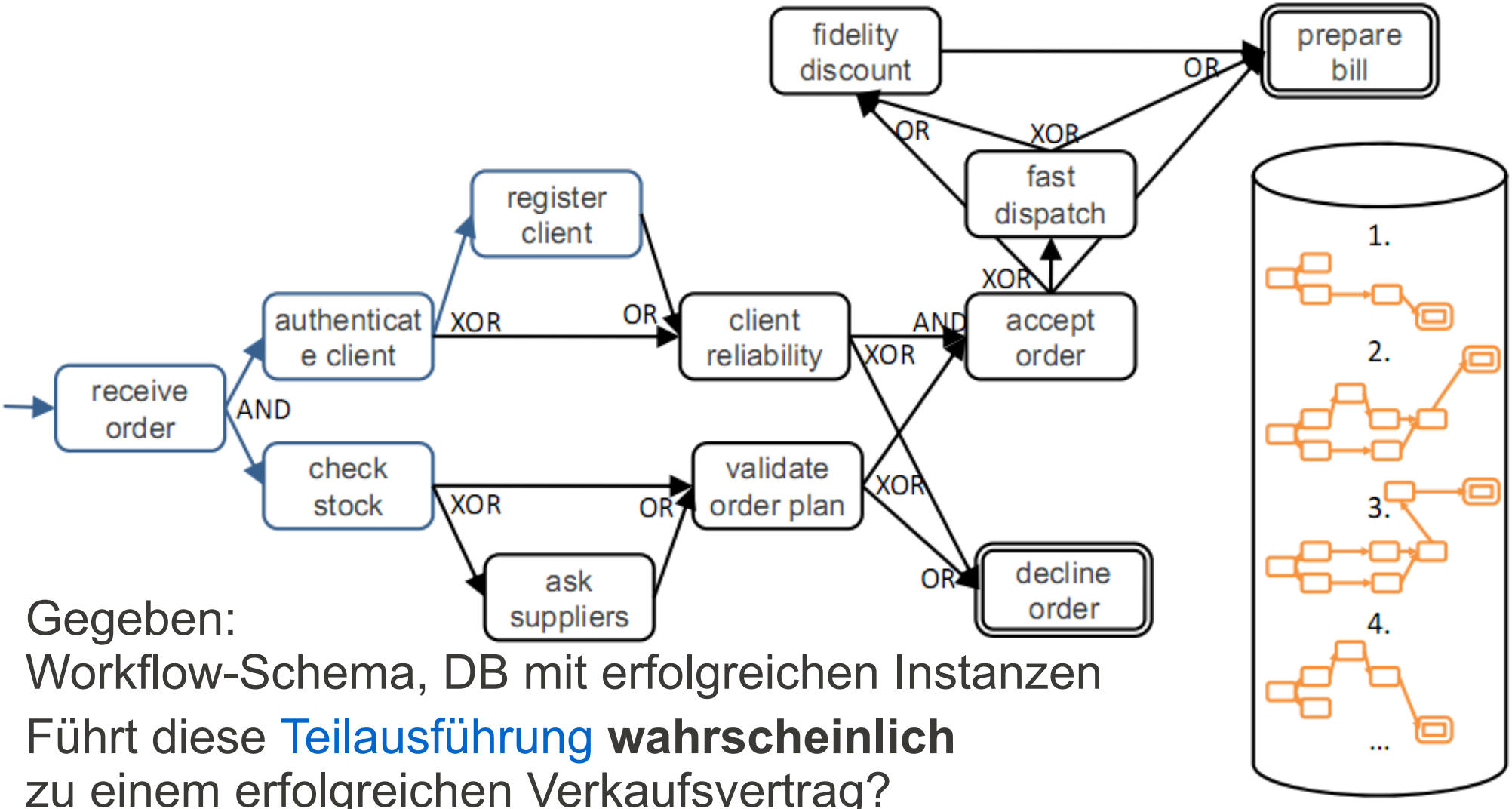
Workflow-Diagnose mit Graph Mining (2)



Gegeben:
Workflow-Schema

Führt diese Teilausführung möglicherweise zu einem erfolgreichen Verkaufsvertrag?

Workflow-Diagnose mit Graph Mining (3)



Gegeben:
Workflow-Schema, DB mit erfolgreichen Instanzen

Führt diese Teilausführung **wahrscheinlich** zu einem erfolgreichen Verkaufsvertrag?

Workflow-Diagnose mit Graph Mining (4)

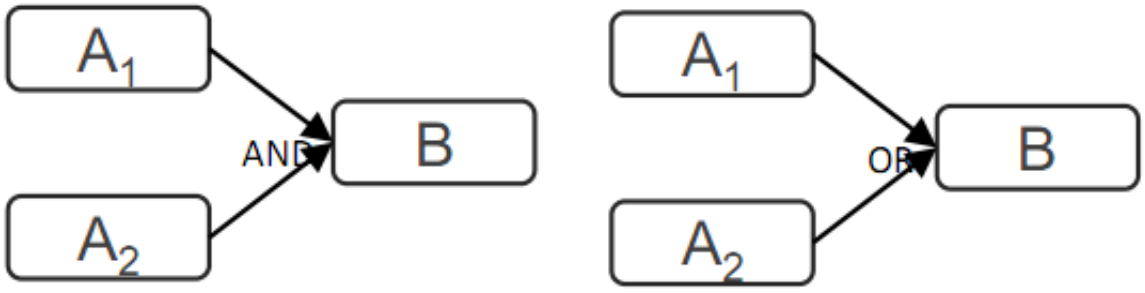
- Weitere interessante Fragestellungen
  - Fehlerfreie Workflow-Ausführung?
  - Workflow-Ausführung mit geringem oder normalem Ressourcenverbrauch?
  - Identifikation kritischer Aktivitäten (besonders hoher Ressourcenverbrauch).
  - Wie sehen häufige/typische Ausführungen aus?
- Verwertung der Information
  - Zur Laufzeit: Scheduling
  - Information für Management/Optimierung (Business Process Reengineering (BPR), Continuous Process Improvement (CPI))
- Ansatz: Ermitteln von häufigen Mustern bei gegebenem Workflow-Schema und Prozess-Logs.

Graph Mining

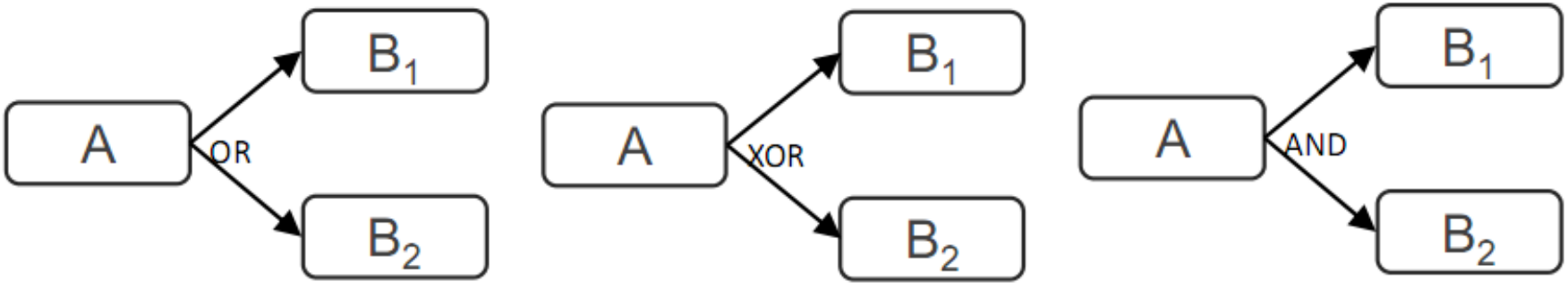
- Finden von häufigen Untergraphen in einer Menge von Graphen (Graph-Datenbank).
  - Häufig: Ein Graph G ist Untergraph von mindestens *minSup* Graphen der Graph-Datenbank.
- Typische Anwendungen
  - Chemie und Pharmaforschung (Moleküle)
  - Softwaretechnik (Call-Graphen)
  - Soziale Netzwerke etc.
- Auch Workflow-Schemata und -Instanzen sind Graphen!

Einschränkungen bei Workflow-Graphen

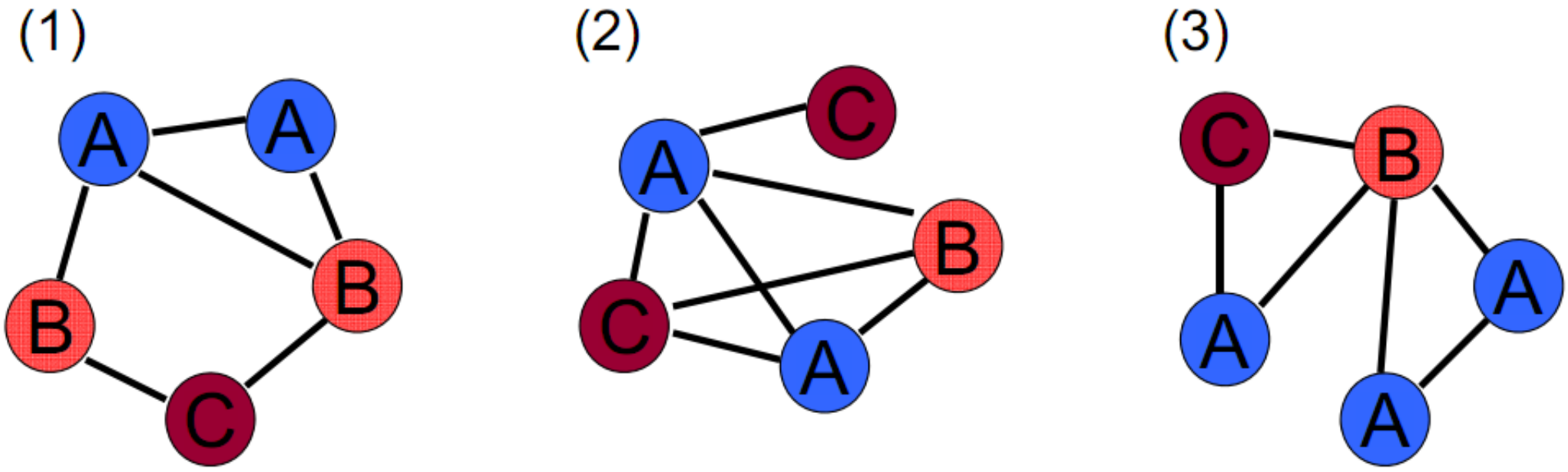
- AND-Join / OR-Join

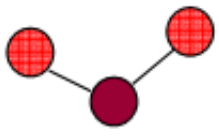
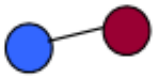
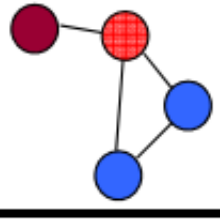


- Full Fork (OR) / Exlusive Fork (XOR) / Deterministic Fork (AND)



Graph Mining - Beispiel



Support	1	2	3
Teilgraph			

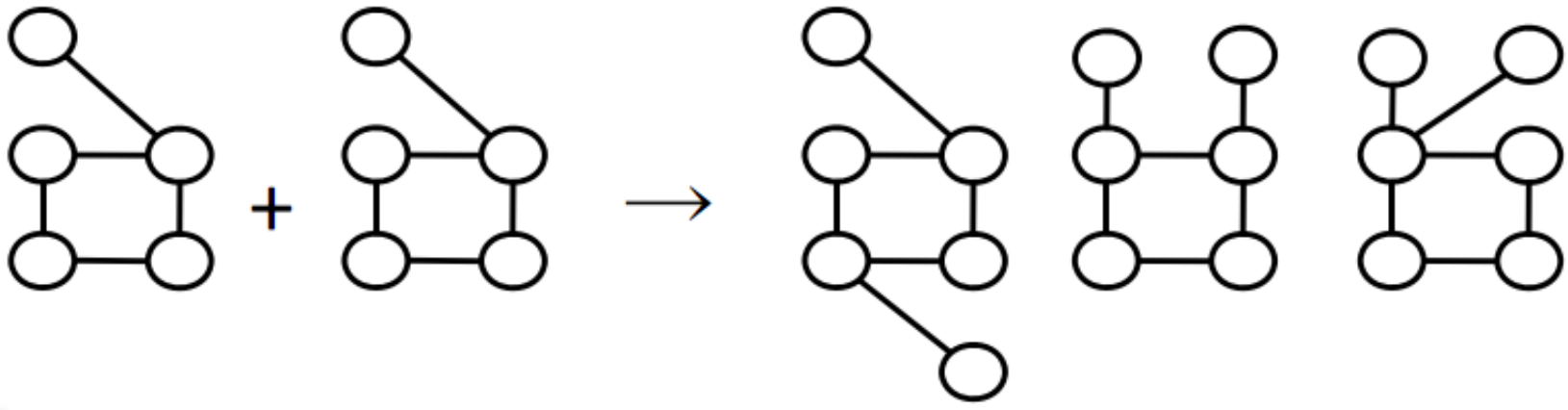
Apriori-Prinzip: G häufig → Alle Untergraphen häufig!

Apriori Graph Mining Algorithmus

- Iteratives Vorgehen (Breitensuche):
  1. Suche alle häufigen Kanten in der Graph-Datenbank (n=1)
  2. Generieren von Kandidaten der Größe n+1 aus häufigen Untergraphen der Größe n
  3. Überprüfen der Häufigkeit der Kandidaten in der Graph-Datenbank
  4. Wiederhole 2+3 solange häufige Graphen gefunden werden

Apriori Graph Mining Algorithmus

- Kandidatengenerierung durch Join von n-1–Kernen:



Nachteile von Apriori-basiertem Graph Mining

- Aufwändige Kandidatengenerierung
  - Jeder Graph der Größe n wird mit jedem verglichen.
  - Finden gemeinsamer Kerne beinhaltet Untergraphisomorphie
  - Mergen der Kerne beinhaltet Suche nach Automorphien.
- Aufwändige Kandidatenüberprüfung
  - Es entstehen im allgemeinen sehr viele Kandidaten.
  - Jede Überprüfung beinhaltet Untergraphisomorphie.
- Untergraphisomorphie ist NP-vollständig.
- **Kein Skalieren für große Workflow-Netze.**
- Es existieren andere Graph-Mining-Algorithmen, die besser skalieren (PatternGrowth-Ansatz). Nicht Gegenstand heute.

Zusammenfassung

- Diese Folien beinhalteten:
  - Grundlagen Petri Netze
  - Grundlagen Business-Process-Mining
  - α-Algorithmus im Detail
  - Grundlagen Graph-Mining