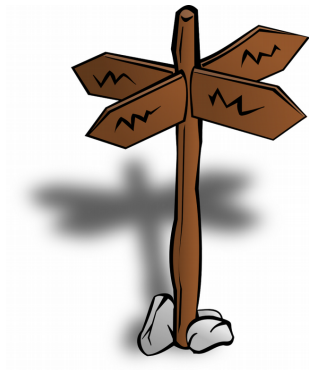


8. Petri-Netze

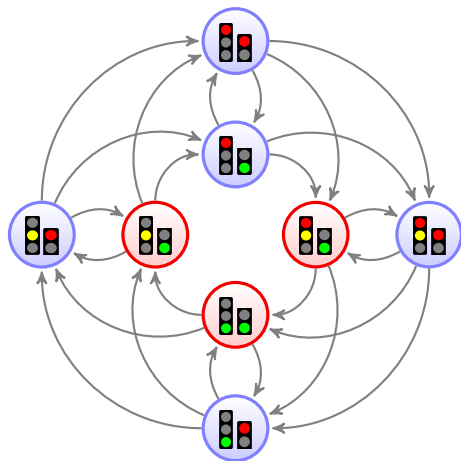
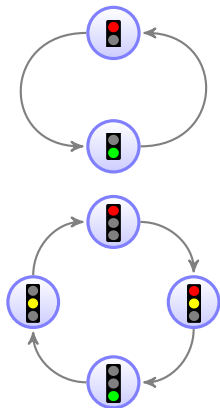
Grundzüge digitaler Systeme

Vortrag von: Gernot Salzer

- Petri-Netze
 - Motivation
 - Definitionen
 - Modellierung



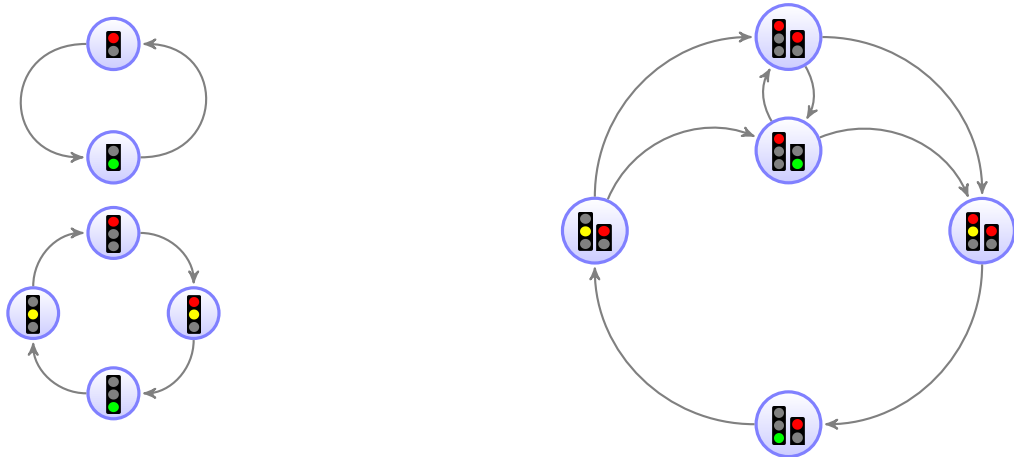
Fußgängerkreuzung als endlicher Automat



Modellierung des Gesamtsystems durch einen Produktautomaten:

- Bilde alle Kombinationszustände.
- Übergänge dort, wo die ursprünglichen Automaten welche hatten.

Fußgängerkreuzung als endlicher Automat



Modellierung des Gesamtsystems durch einen Produktautomaten:

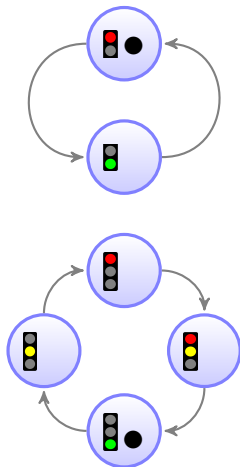
- Bilde alle Kombinationszustände.
- Übergänge dort, wo die ursprünglichen Automaten welche hatten.
- Elimination unerwünschter Zustände und Übergänge.

Endliche Automaten ungeeignet für Modellierung verteilter Systeme:

- Anzahl der Kombinationszustände steigt exponentiell mit der Zahl der Komponenten.
- Anzahl der Übergänge steigt exponentiell.
- Es ist schwierig, Änderungen einer Komponente in den Gesamtautomaten zu übertragen.
- Endliche Automaten mit *einem* aktiven Zustand entsprechen nicht der Idee verteilter Systeme mit *vielen* unabhängigen Abläufen nebeneinander, die nur bei Bedarf synchronisiert werden.

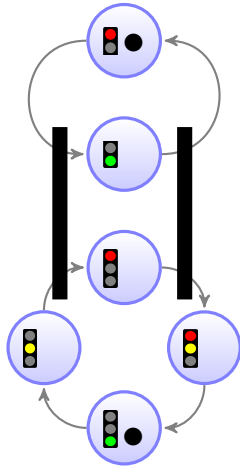
Petri-Netz = Automat mit mehreren aktiven Stellen + Synchronisation

Fußgängerkreuzung als Petri-Netz



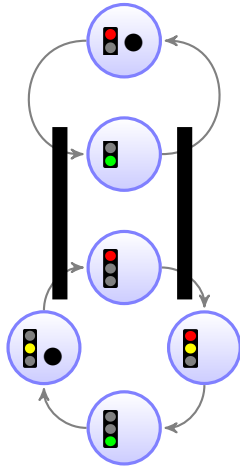
- *Marken* zeigen die aktiven Stellen an.

Fußgängerkreuzung als Petri-Netz



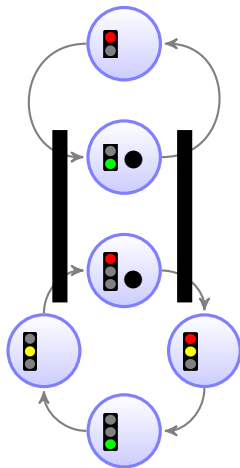
- *Marken* zeigen die aktiven Stellen an.
- *Transitionen* werden nur durchlässig („feuern“), wenn alle Eingangszustände Marken besitzen.
Marken dürfen nur gleichzeitig weiterwandern.

Fußgängerkreuzung als Petri-Netz



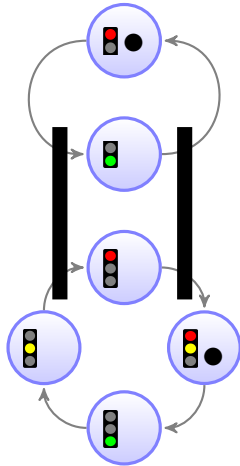
- *Marken* zeigen die aktiven Stellen an.
- *Transitionen* werden nur durchlässig („feuern“), wenn alle Eingangszustände Marken besitzen.
Marken dürfen nur gleichzeitig weiterwandern.

Fußgängerkreuzung als Petri-Netz



- *Marken* zeigen die aktiven Stellen an.
- *Transitionen* werden nur durchlässig („feuern“), wenn alle Eingangszustände Marken besitzen.
Marken dürfen nur gleichzeitig weiterwandern.

Fußgängerkreuzung als Petri-Netz



- *Marken* zeigen die aktiven Stellen an.
- *Transitionen* werden nur durchlässig („feuern“), wenn alle Eingangszustände Marken besitzen.
Marken dürfen nur gleichzeitig weiterwandern.

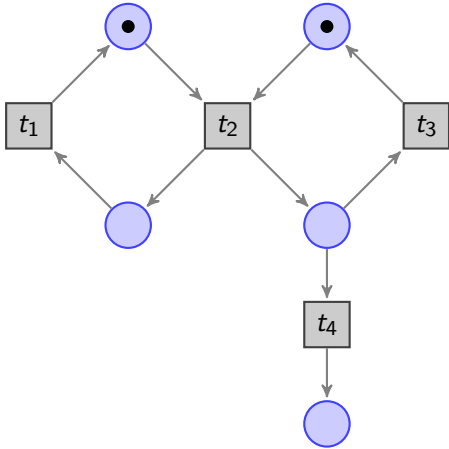
Petri-Netze: Motivation

Formalismus zur Modellierung von nebenläufigen Systemen (concurrent/parallel systems).

- Bei Systemübergängen können Ressourcen konsumiert und neu erzeugt werden.
- Natürliche Modellierung der räumlichen Verteilung von Ressourcen, Nebenläufigkeit und (Zugriffs-)Konflikten.
- Intuitive graphische Darstellung.
- Weit verbreitet, z.B. Aktivitätsdiagramme (activity diagrams) in UML (Unified Modeling Language).

Übernommen von Barbara König, Vorlesung "Modellierung nebenläufiger Systeme", Uni Duisburg-Essen, 2011

Petri-Netze: Motivation

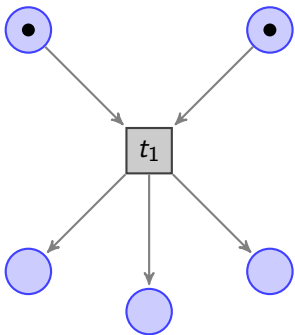


Notation:

- Stellen (dargestellt als Kreise): mögliche Plätze für Ressourcen
- Marken (dargestellt als kleine gefüllte Kreise): Ressourcen
- Transitionen (dargestellt als Rechtecke oder Balken): Systemübergänge

Petri-Netze: Motivation

Darstellung einer *Transition*:

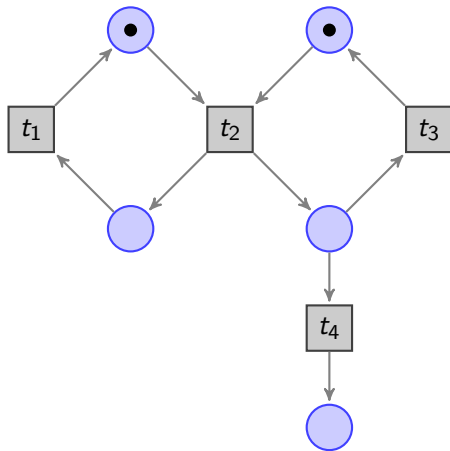


Vorbedingungen sind die Marken, die konsumiert werden

Nachbedingungen sind die Marken, die erzeugt werden

Das Entfernen der Marken der Vorbedingungen und das Erzeugen der Marken der Nachbedingungen nennt man *Schalten* bzw. *Feuern* der Transition.

Petri-Netze: Beispiel



Übernommen von Barbara König, Vorlesung "Modellierung nebenläufiger Systeme", Uni Duisburg-Essen, 2011

- Petri-Netze
 - Motivation
 - Definitionen
 - Modellierung



Definitionen

M ... Menge der Markierungen, d.h., aller Abbildungen $S \mapsto \mathbb{N}$

Petri-Netz

... wird beschrieben durch ein 5-Tupel $N = \langle S, T, \bullet(), ()^\bullet, m_0 \rangle$, wobei

- S ... endliche Menge von Stellen
- T ... endliche Menge von Transitionen
- $\bullet(): T \mapsto M$... Vorbedingungen
- $()^\bullet: T \mapsto M$... Nachbedingungen
- $m_0 \in M$... Anfangsmarkierung

$m_0 \in M$ legt fest, wieviele Marken zu Beginn in jeder Stelle liegen.

$\bullet t \in M$ legt fest, wieviele Marken die Transition t aus jeder Stelle entfernt.

$t^\bullet \in M$ legt fest, wieviele Marken die Transition t zu jeder Stelle hinzufügt.

Schalten und Erreichbarkeit

$m, m' \in M \dots$ Markierungen

Ordnung: $m \leq m'$ falls $m(s) \leq m'(s)$ für alle $s \in S$

Addition: $m \oplus m' = m''$ falls $m''(s) = m(s) + m'(s)$ für alle $s \in S$.

Subtraktion: $m \ominus m' = m''$ falls $m''(s) = m(s) - m'(s)$ für alle $s \in S$.

- Eine Transition t ist für eine Markierung m *aktiviert*, wenn $\bullet t \leq m$ gilt, d.h., wenn genug Marken vorhanden sind, um die Transition zu schalten.
- Ist die Transition t für die Markierung m aktiviert, kann t *schalten* (*feuern*). Das führt zur neuen Markierung $m' = m \ominus \bullet t \oplus t^\bullet$, symbolisch $m[t \rangle m'$.
- Eine Markierung m_n heißt *erreichbar* in einem Netz, falls es eine Folge von Transitionen t_1, \dots, t_n gibt mit $m_0[t_1 \rangle m_1 \dots m_{n-1}[t_n \rangle m_n$, wobei m_0 die Anfangsmarkierung ist.

In der graphischen Notation werden $\bullet t$ und t^\bullet folgendermaßen dargestellt:

- Kein Pfeil zwischen s und t , falls $\bullet t(s) = 0$ (bzw. $t^\bullet(s) = 0$).
- Ein Pfeil zwischen s und t , falls $\bullet t(s) = 1$ (bzw. $t^\bullet(s) = 1$).
- Ein Pfeil mit Beschriftung n zwischen s und t , falls $\bullet t(s) = n > 1$ (bzw. $t^\bullet(s) = n > 1$).

Der Wert $\bullet t(s)$ bzw. $t^\bullet(s)$ wird auch als *Gewicht* bezeichnet.

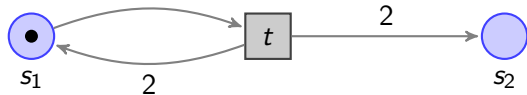
Beispiel

Petri-Netz $\langle \{s_1, s_2\}, \{t\}, \bullet(), ()^\bullet, m_0 \rangle$ mit

$$\bullet t = \{s_1 \mapsto 1, s_2 \mapsto 0\}$$

$$t^\bullet = \{s_1 \mapsto 2, s_2 \mapsto 2\}$$

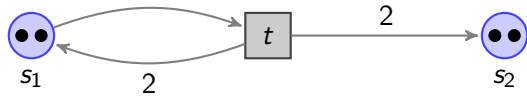
$$m_0 = \{s_1 \mapsto 1, s_2 \mapsto 0\}$$



Feuern von t : $m_1 = m_0 \ominus \bullet t \oplus t^\bullet$

$$\begin{aligned} m_1(s_1) &= m_0(s_1) - \bullet t(s_1) + t^\bullet(s_1) \\ &= 1 - 1 + 2 = 2 \end{aligned}$$

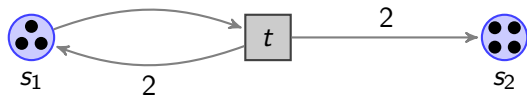
$$\begin{aligned} m_1(s_2) &= m_0(s_2) - \bullet t(s_2) + t^\bullet(s_2) \\ &= 0 - 0 + 2 = 2 \end{aligned}$$



Feuern von t : $m_2 = m_1 \ominus \bullet t \oplus t^\bullet$

$$\begin{aligned} m_2(s_1) &= m_1(s_1) - \bullet t(s_1) + t^\bullet(s_1) \\ &= 2 - 1 + 2 = 3 \end{aligned}$$

$$\begin{aligned} m_2(s_2) &= m_1(s_2) - \bullet t(s_2) + t^\bullet(s_2) \\ &= 2 - 0 + 2 = 4 \end{aligned}$$



- Petri-Netze
 - Motivation
 - Definitionen
 - Modellierung

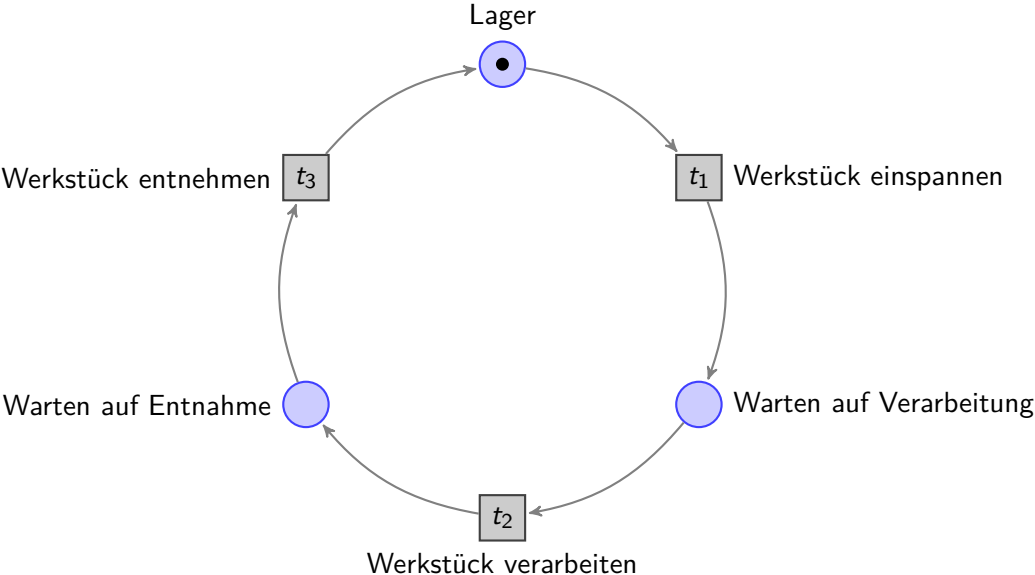


Modellierungsmuster: Sequentieller Ablauf

Entspricht den Zustandsübergängen bei endlichen Automaten.

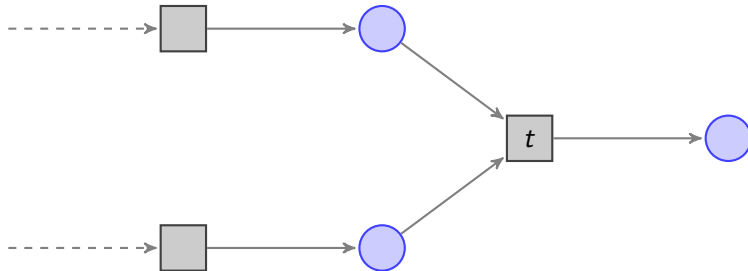


Modellierungsmuster: Zyklen



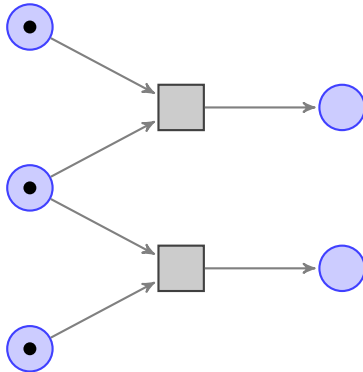
Modellierungsmuster: Abhängigkeiten, Synchronisation

Transition t kann erst feuern, wenn aus beiden Zweigen Marken vorliegen.



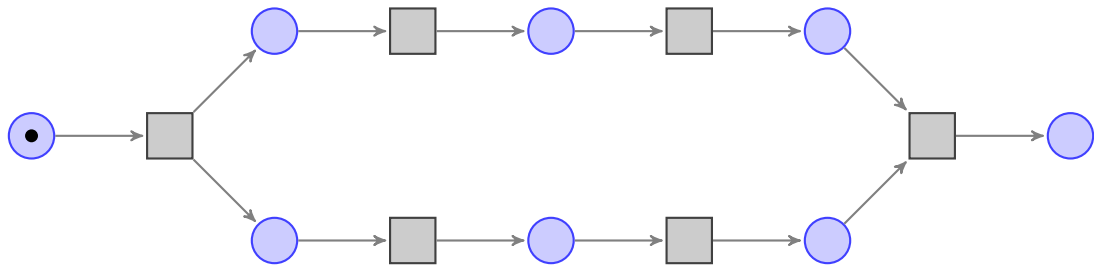
Modellierungsmuster: Entweder – oder

Nur eine der beiden Transitionen kann feuern.



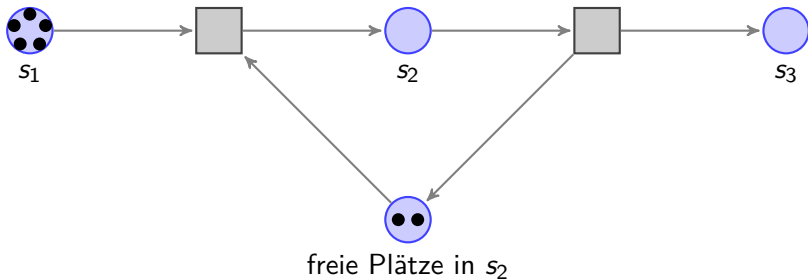
Modellierungsmuster: Parallelität, Nebenläufigkeit

Fork-Join: Zwei sequentielle Abläufe, die gleichzeitig beginnen, dann unabhängig voneinander ablaufen und zuletzt aufeinander warten.



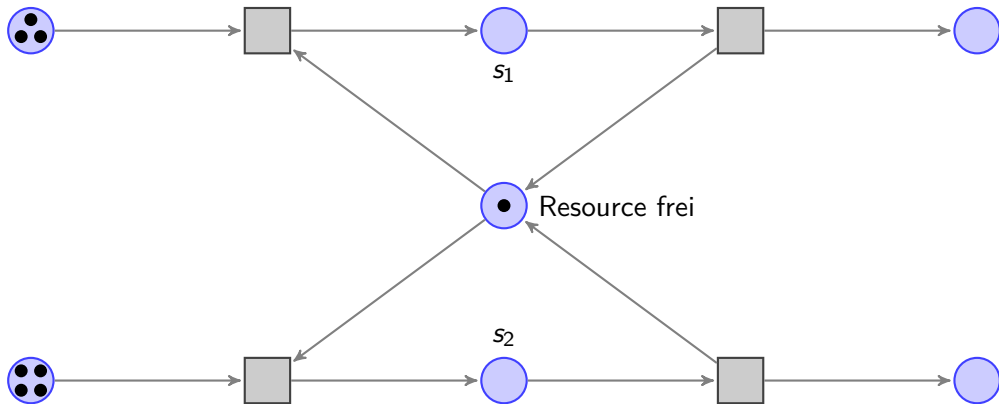
Modellierungsmuster: Markenzahl beschränken

Nicht mehr als zwei Marken in s_2 gleichzeitig möglich.



Modellierungsmuster: geteilte Ressourcen

In den Stellen s_1 und s_2 kann nicht gleichzeitig eine Marke liegen.



Beispiel: Leser-Schreiber-Problem

Leser-Schreiber-Problem

Beim Leser-Schreiber-Problem operieren n Leserprozesse und m Schreiberprozesse auf ein und derselben Datei. Damit die Dateiinhalte nicht inkonsistent werden, müssen die folgenden Bedingungen beachtet werden:

- Es können zur gleichen Zeit mehrere Leserprozesse auf die Datei zugreifen.
- Ein Schreiberprozess darf nur dann auf die Datei zugreifen, wenn gerade kein anderer Prozess (lesend oder schreibend) auf die Datei zugreift.

Modellieren Sie das Leser-Schreiber-Problem als Petri-Netz mit $n = 3$ und $m = 1$. Es können maximal 2 Leserprozesse gleichzeitig die Datei lesen.

Beispiel: Leser-Schreiber-Problem

Leser-Schreiber-Problem

Beim Leser-Schreiber-Problem operieren n Leserprozesse und m Schreiberprozesse **auf ein und derselben Datei**. Damit die Dateiinhalte nicht inkonsistent werden, müssen die folgenden Bedingungen beachtet werden:

- Es können zur gleichen Zeit mehrere Leserprozesse auf die Datei zugreifen.
- Ein Schreiberprozess darf nur dann auf die Datei zugreifen, wenn gerade kein anderer Prozess (lesend oder schreibend) auf die Datei zugreift.

Modellieren Sie das Leser-Schreiber-Problem als Petri-Netz mit $n = 3$ und $m = 1$. Es können maximal 2 Leserprozesse gleichzeitig die Datei lesen.

Beispiel: Leser-Schreiber-Problem



Beispiel: Leser-Schreiber-Problem

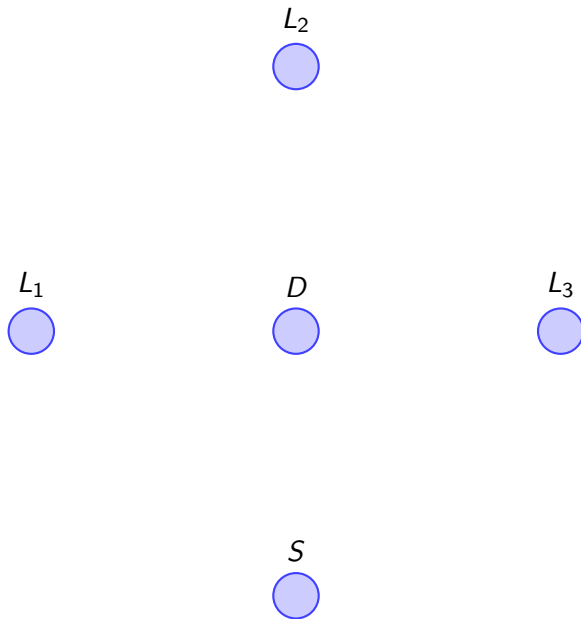
Leser-Schreiber-Problem

Beim Leser-Schreiber-Problem operieren n **Leserprozesse** und m **Schreiberprozesse** auf ein und derselben Datei. Damit die Dateiinhalte nicht inkonsistent werden, müssen die folgenden Bedingungen beachtet werden:

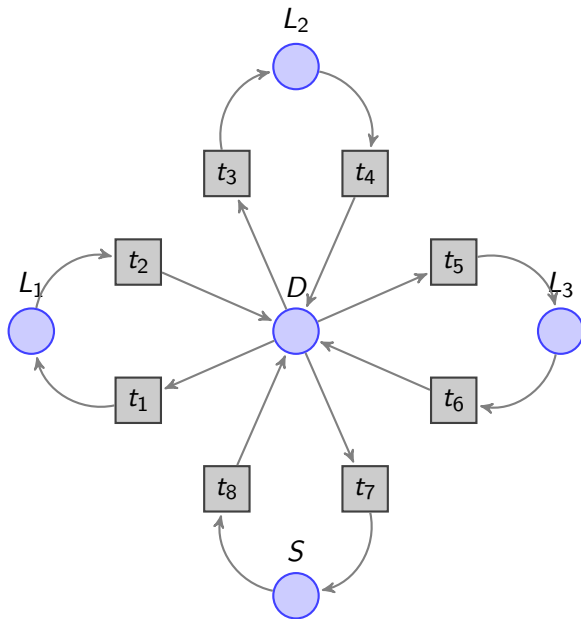
- Es können zur gleichen Zeit mehrere Leserprozesse auf die Datei zugreifen.
- Ein Schreiberprozess darf nur dann auf die Datei zugreifen, wenn gerade kein anderer Prozess (lesend oder schreibend) auf die Datei zugreift.

Modellieren Sie das Leser-Schreiber-Problem als Petri-Netz mit $n = 3$ und $m = 1$. Es können maximal 2 Leserprozesse gleichzeitig die Datei lesen.

Beispiel: Leser-Schreiber-Problem



Beispiel: Leser-Schreiber-Problem



Beispiel: Leser-Schreiber-Problem

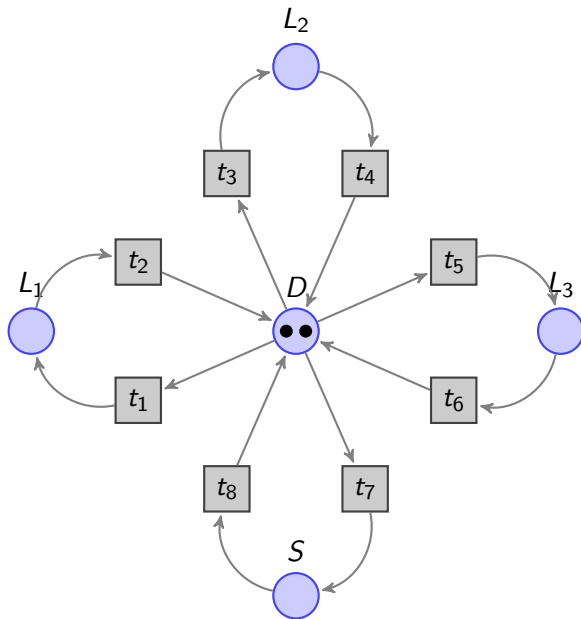
Leser-Schreiber-Problem

Beim Leser-Schreiber-Problem operieren n Leserprozesse und m Schreiberprozesse auf ein und derselben Datei. Damit die Dateiinhalte nicht inkonsistent werden, müssen die folgenden Bedingungen beachtet werden:

- Es können zur gleichen Zeit **mehrere Leserprozesse** auf die Datei zugreifen.
- Ein Schreiberprozess darf nur dann auf die Datei zugreifen, wenn gerade kein anderer Prozess (lesend oder schreibend) auf die Datei zugreift.

Modellieren Sie das Leser-Schreiber-Problem als Petri-Netz mit $n = 3$ und $m = 1$. Es können **maximal 2 Leserprozesse** gleichzeitig die Datei lesen.

Beispiel: Leser-Schreiber-Problem



Beispiel: Leser-Schreiber-Problem

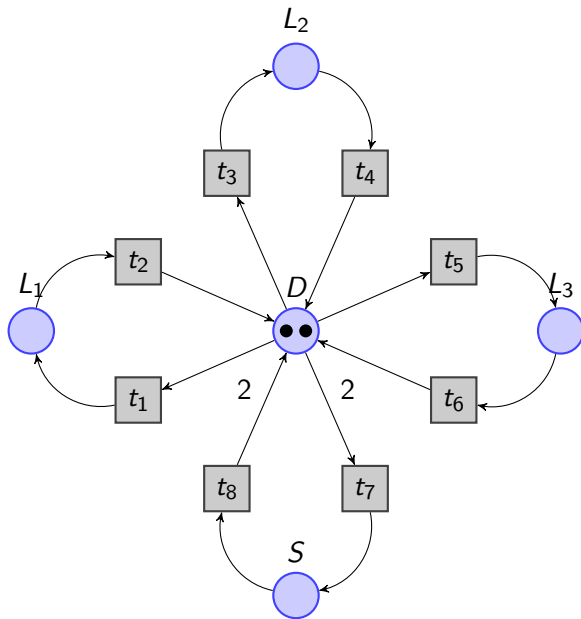
Leser-Schreiber-Problem

Beim Leser-Schreiber-Problem operieren n Leserprozesse und m Schreiberprozesse auf ein und derselben Datei. Damit die Dateiinhalte nicht inkonsistent werden, müssen die folgenden Bedingungen beachtet werden:

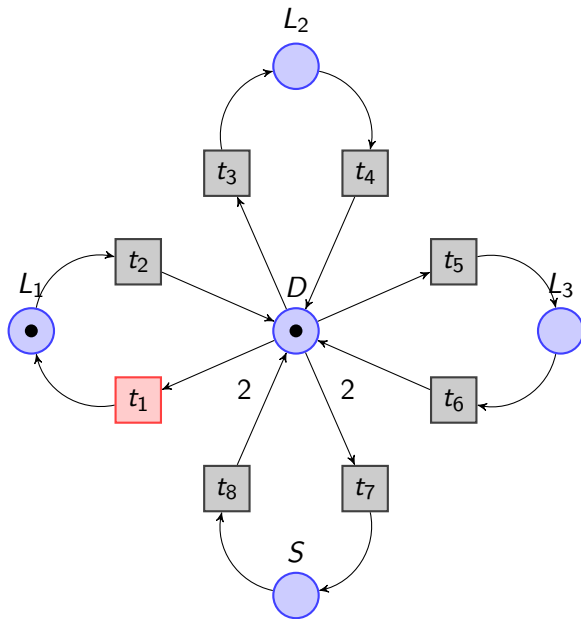
- Es können zur gleichen Zeit mehrere Leserprozesse auf die Datei zugreifen.
- Ein **Schreiberprozess** darf nur dann auf die Datei zugreifen, wenn gerade **kein anderer Prozess (lesend oder schreibend) auf die Datei zugreift**.

Modellieren Sie das Leser-Schreiber-Problem als Petri-Netz mit $n = 3$ und $m = 1$. Es können maximal 2 Leserprozesse gleichzeitig die Datei lesen.

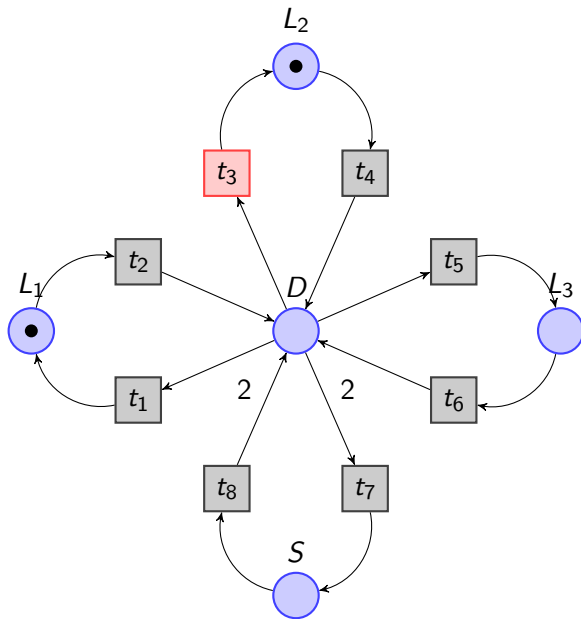
Beispiel: Leser-Schreiber-Problem



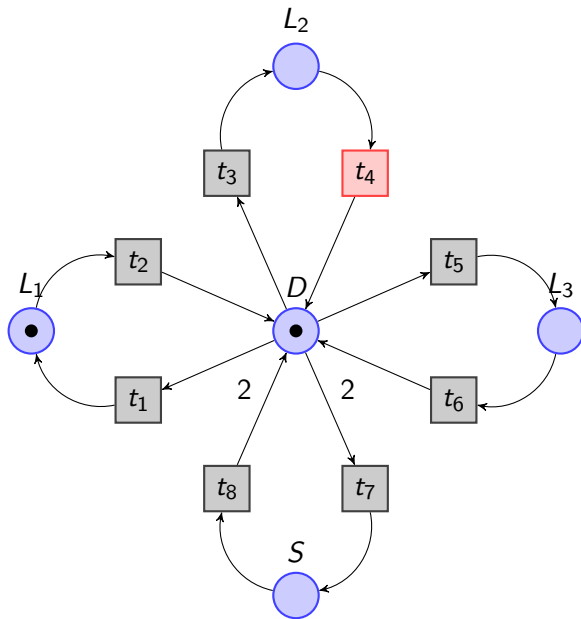
Beispiel: Leser-Schreiber-Problem



Beispiel: Leser-Schreiber-Problem

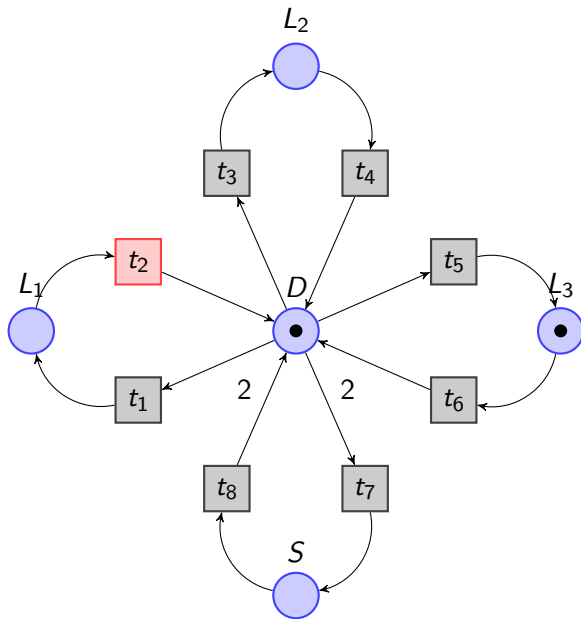


Beispiel: Leser-Schreiber-Problem

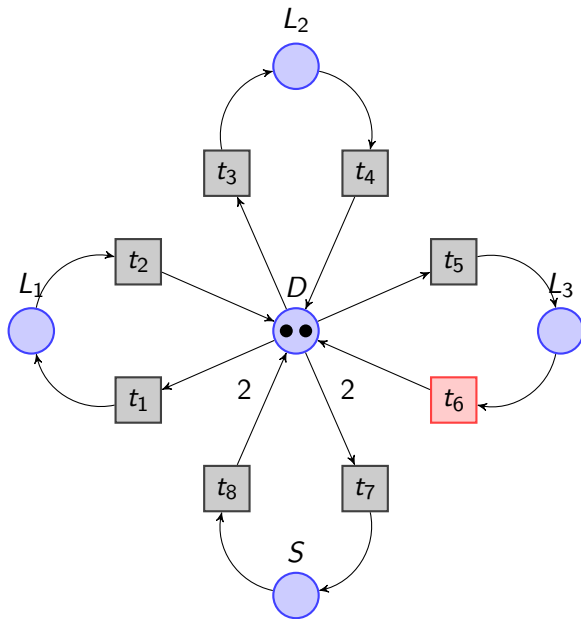




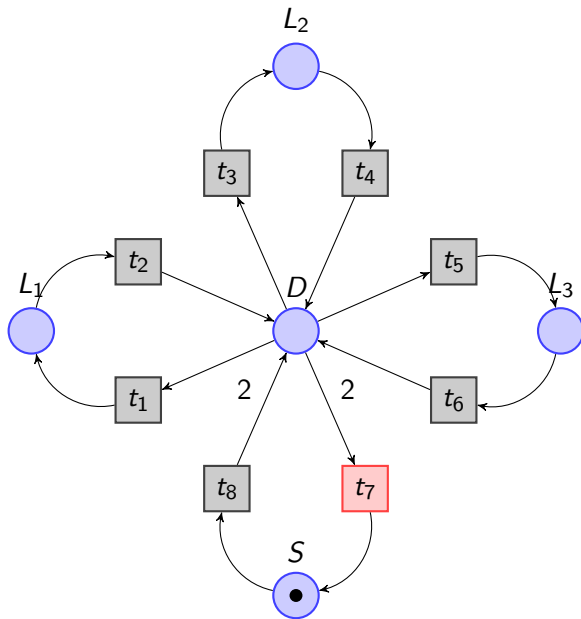
Beispiel: Leser-Schreiber-Problem



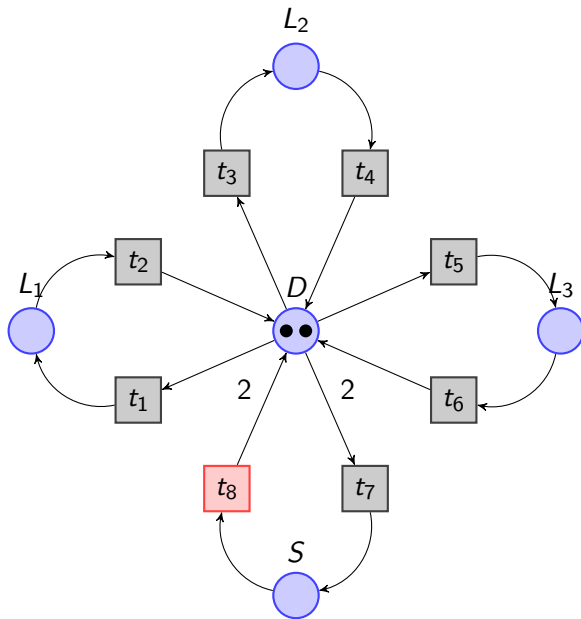
Beispiel: Leser-Schreiber-Problem



Beispiel: Leser-Schreiber-Problem



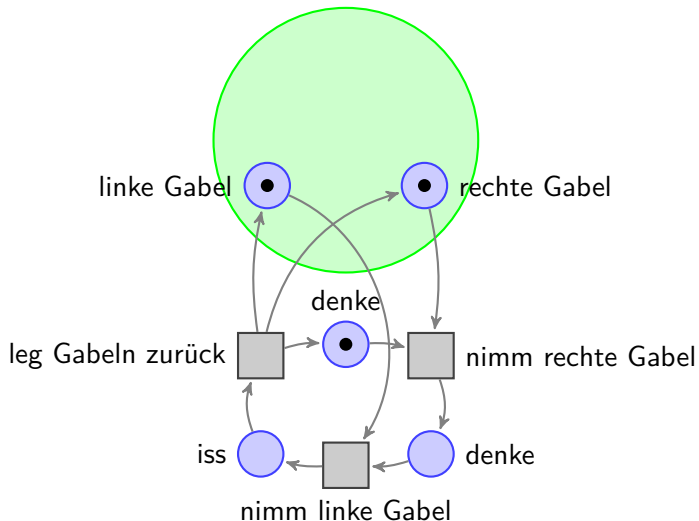
Beispiel: Leser-Schreiber-Problem



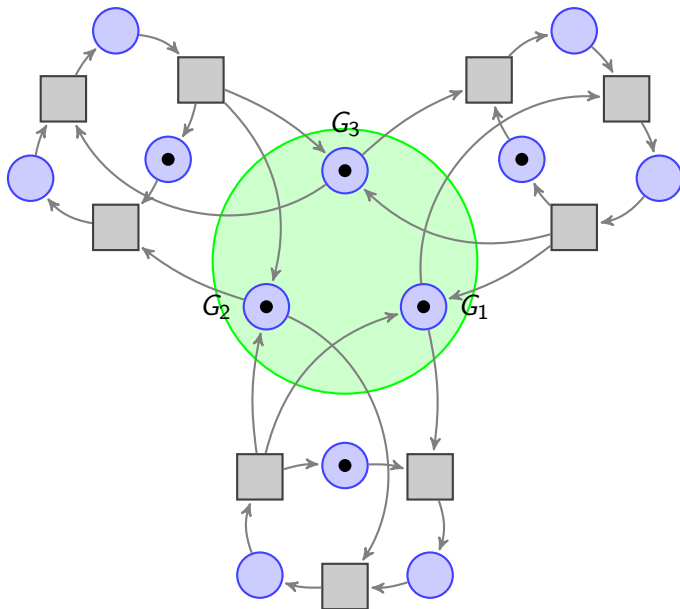
Bei den Philosophen

Tagesablauf:

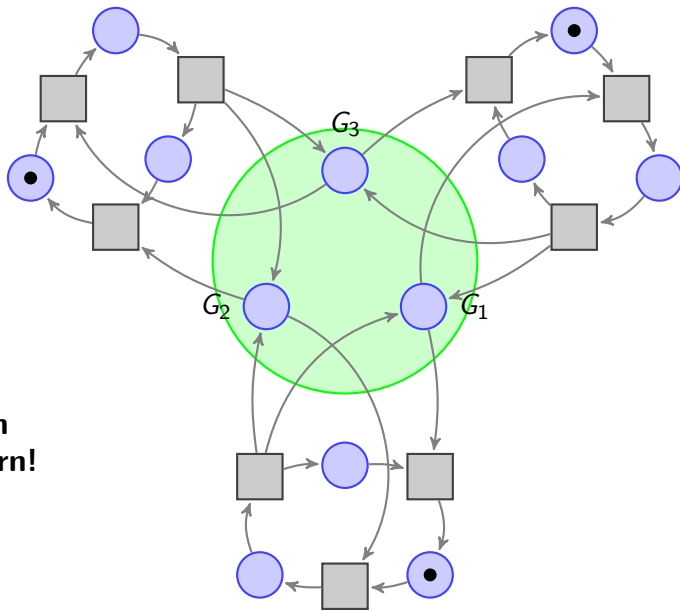
- 1 Denke.
- 2 Nimm die rechte Gabel.
- 3 Denke.
- 4 Nimm die linke Gabel.
- 5 Iss.
- 6 Leg die Gabeln zurück.
- 7 Beginne von vorne.



Dining Philosophers (Dijkstra, Hoare 1965)



Dining Philosophers (Dijkstra, Hoare 1965)



Deadlock:
Keine Transition
kann mehr feuern!

- Petri-Netze eignen sich gut zur Modellierung verteilter Abläufe
- Standardmuster (geteilte Ressourcen, Parallelverarbeitung, Synchronisation) sind leicht darstellbar
- ausdrucksstärker als reguläre Automaten
- automatisierte Deadlock- und Erreichbarkeitstests