3.0 VU Formale Modellierung

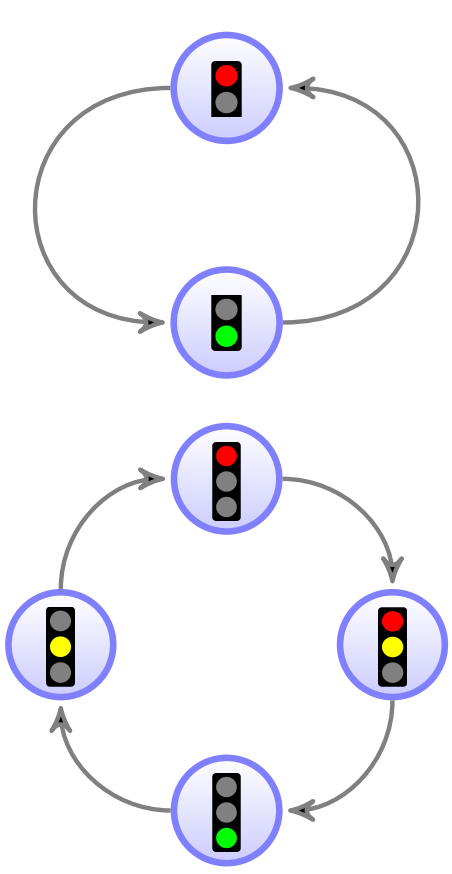
Gernot Salzer

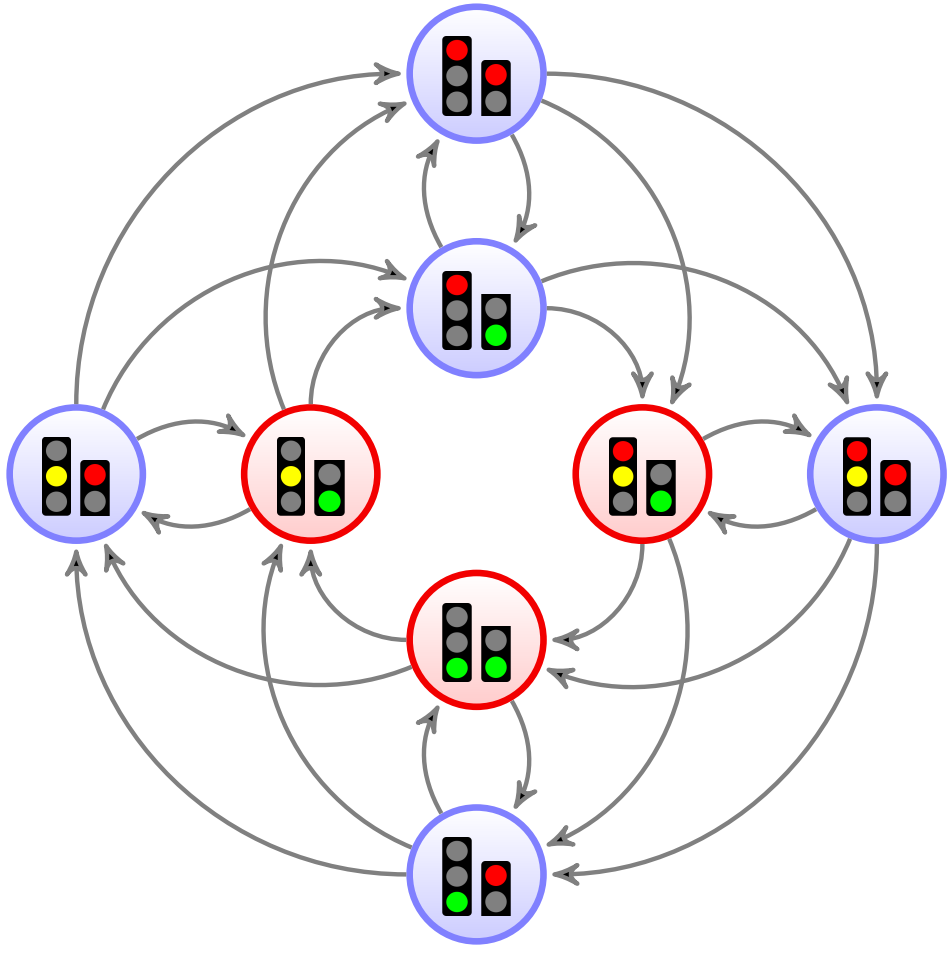
Arbeitsbereich Theoretische Informatik und Logik
Institut für Computersprachen

SS 2016

Inhalt

Fußgängerkreuzung als endlicher Automat

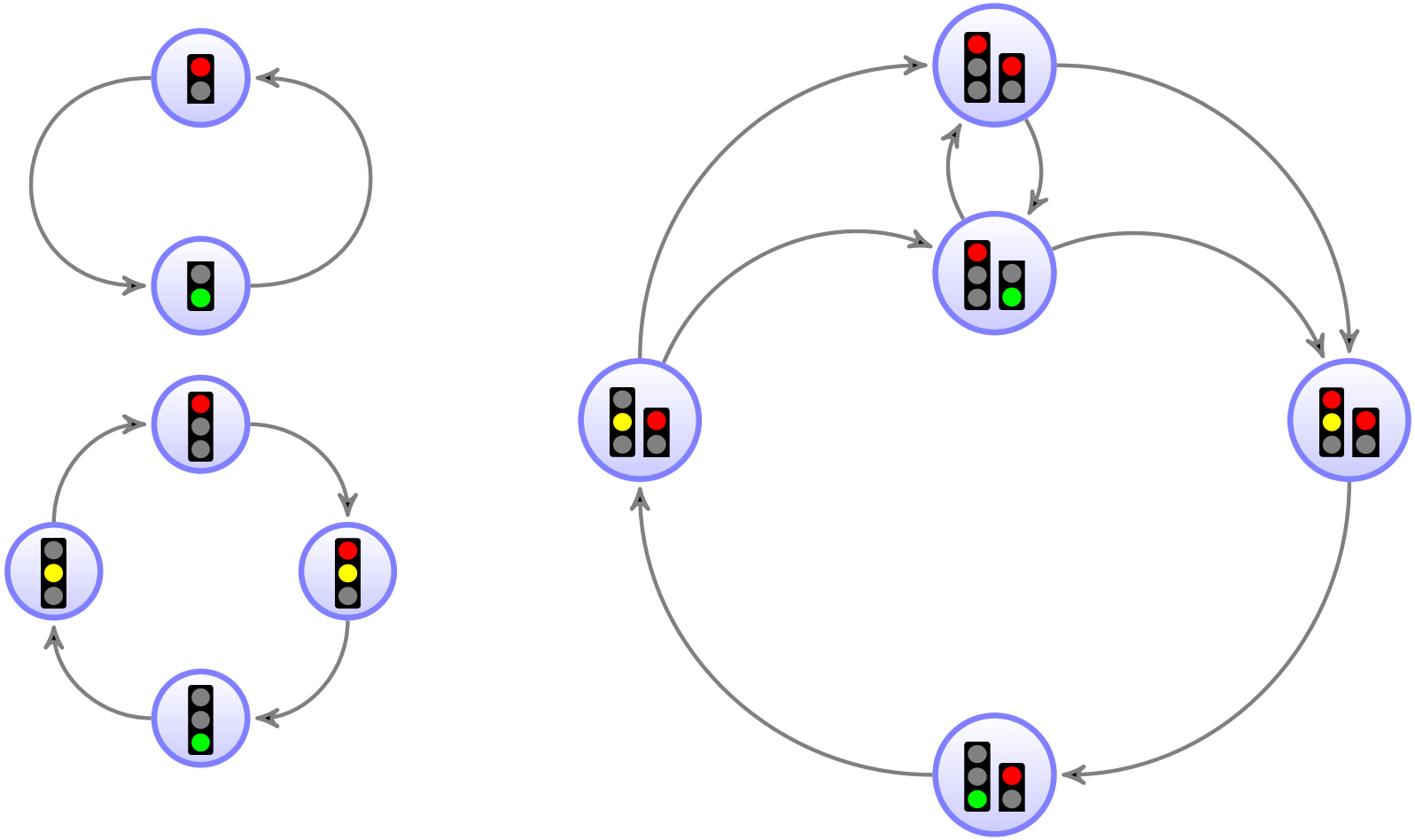




Modellierung des Gesamtsystems durch einen Produktautomaten:

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

Fußgängerkreuzung als endlicher Automat



Modellierung des Gesamtsystems durch einen Produktautomaten:

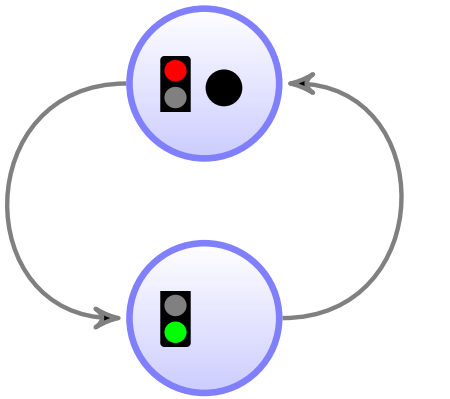
- Bilde alle Kombinationszustände.
- Übergange dort, wo die ursprünglichen Automaten welche hatten.
- Elimination unerwünschter Zustände und Übergänge.

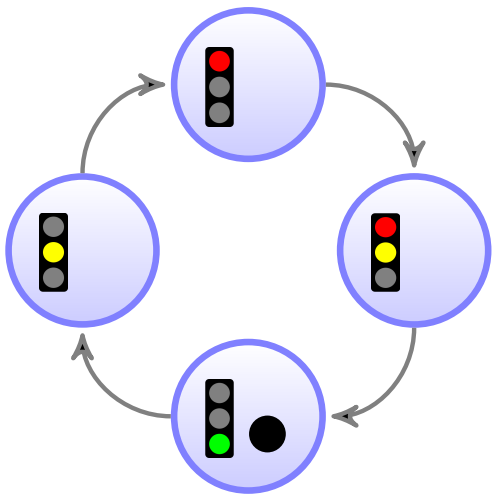
Endliche Automaten ungeeignet für Modellierung verteilter Systeme:

- 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

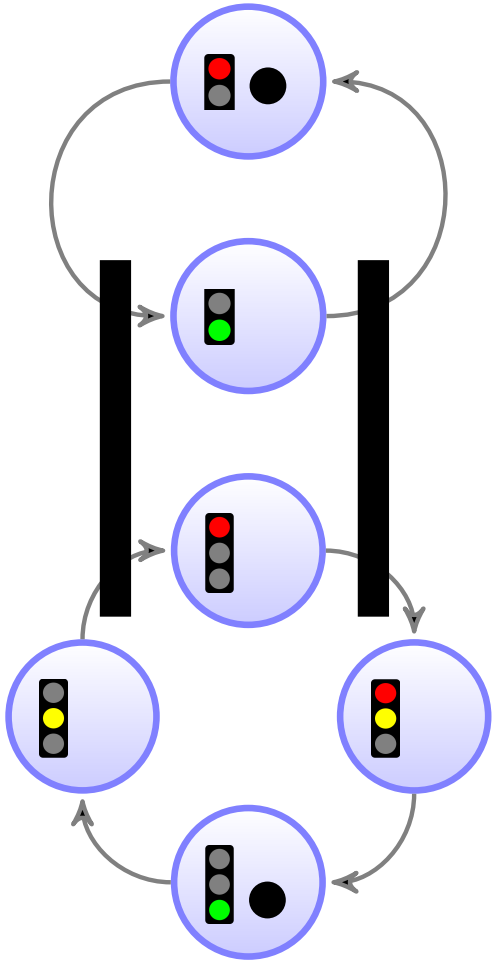
Fußgängerkreuzung als Petri-Netz





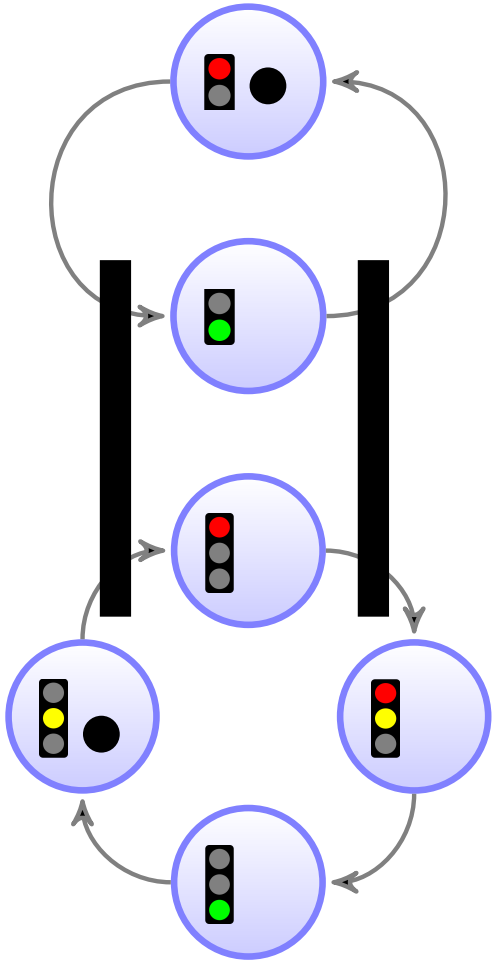
- Marken zeigen die aktiven Stellen an.

Fußgängerkreuzung als Petri-Netz



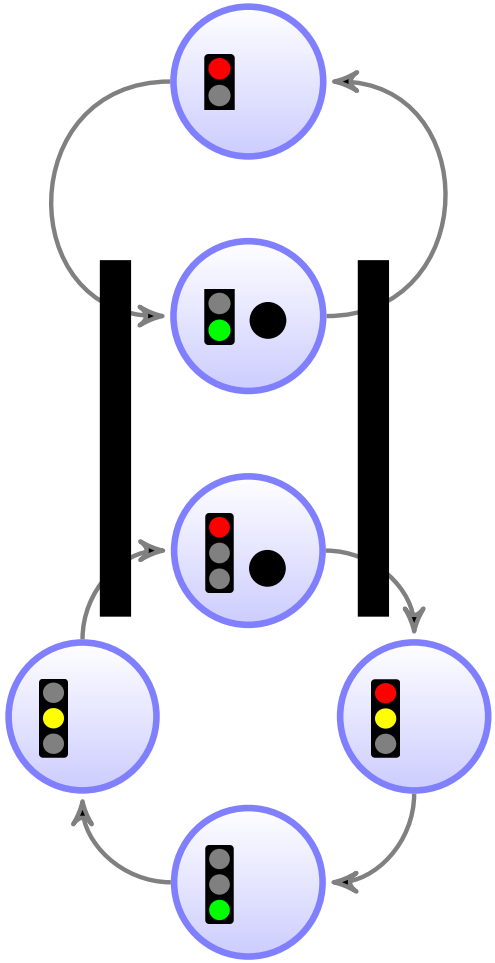
- Marken zeigen die aktiven Stellen an.
- Transitionen werden nur durchlässig („feuern“), wenn alle Eingangszustände Marken besitzen.

Fußgängerkreuzung als Petri-Netz



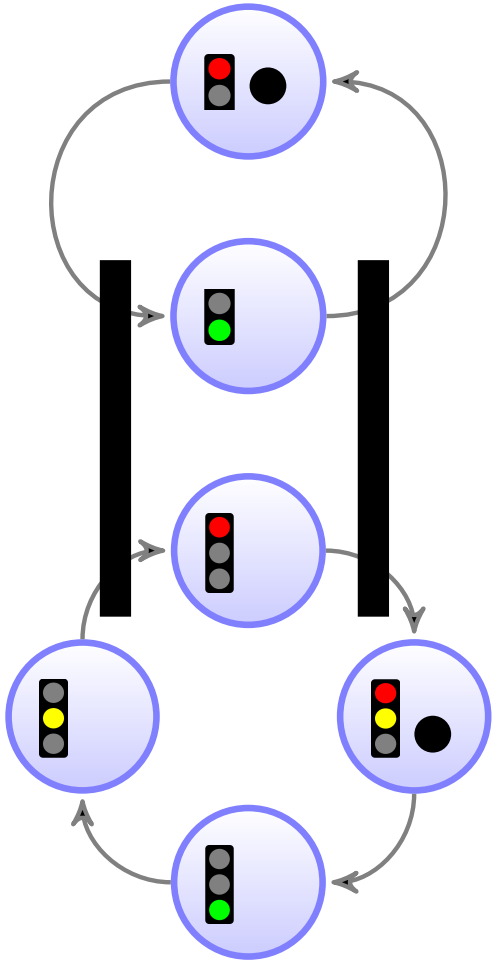
- Marken zeigen die aktiven Stellen an.
- Transitionen werden nur durchlässig („feuern“), wenn alle Eingangszustände Marken besitzen.

Fußgängerkreuzung als Petri-Netz



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

Fußgängerkreuzung als Petri-Netz



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

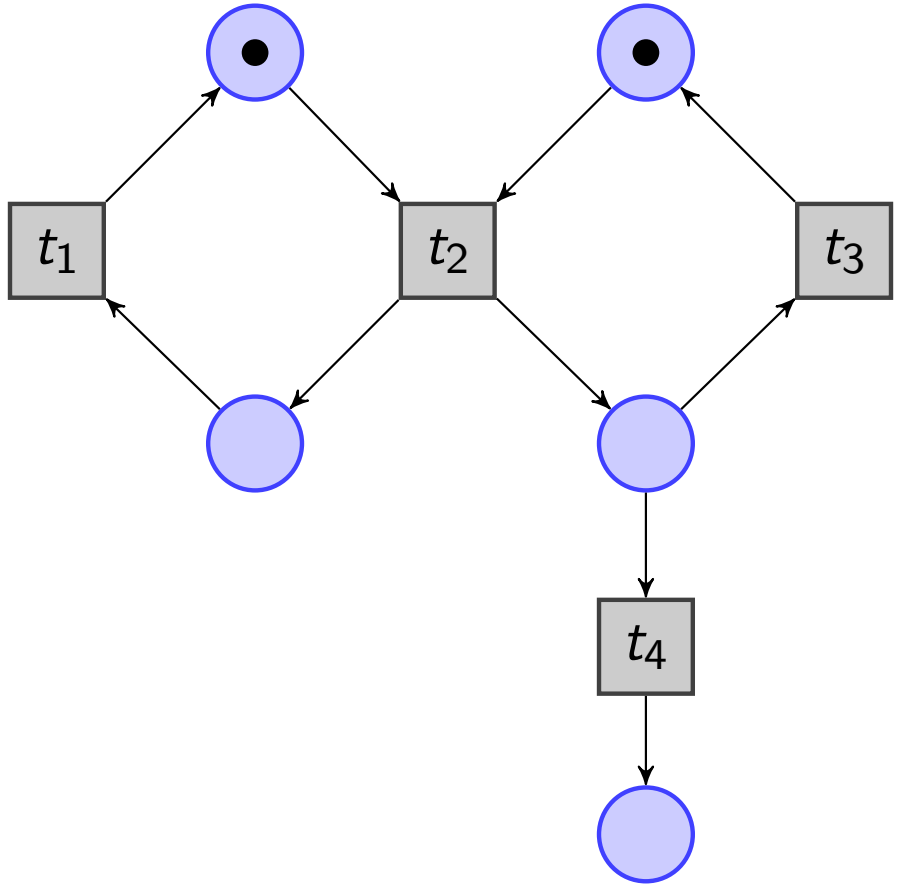
Petrinetze: Motivation

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

Petrinetze: Motivation

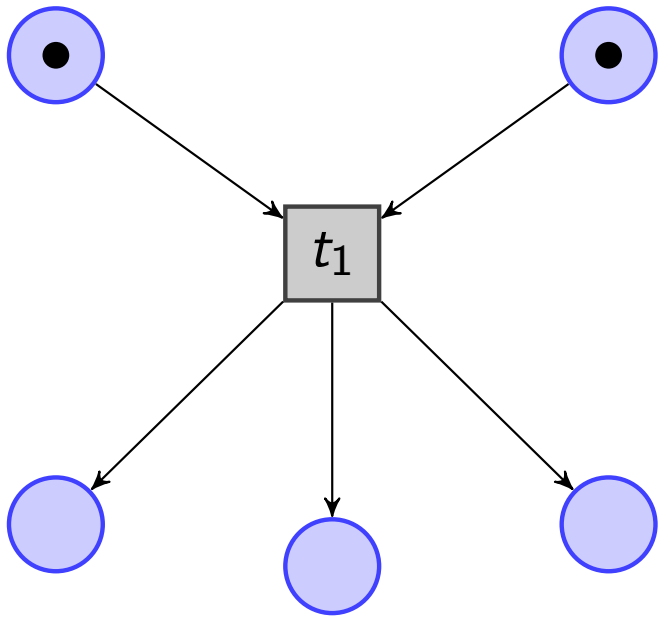


Notation:

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

Petrinetze: Motivation

Darstellung einer Transition:

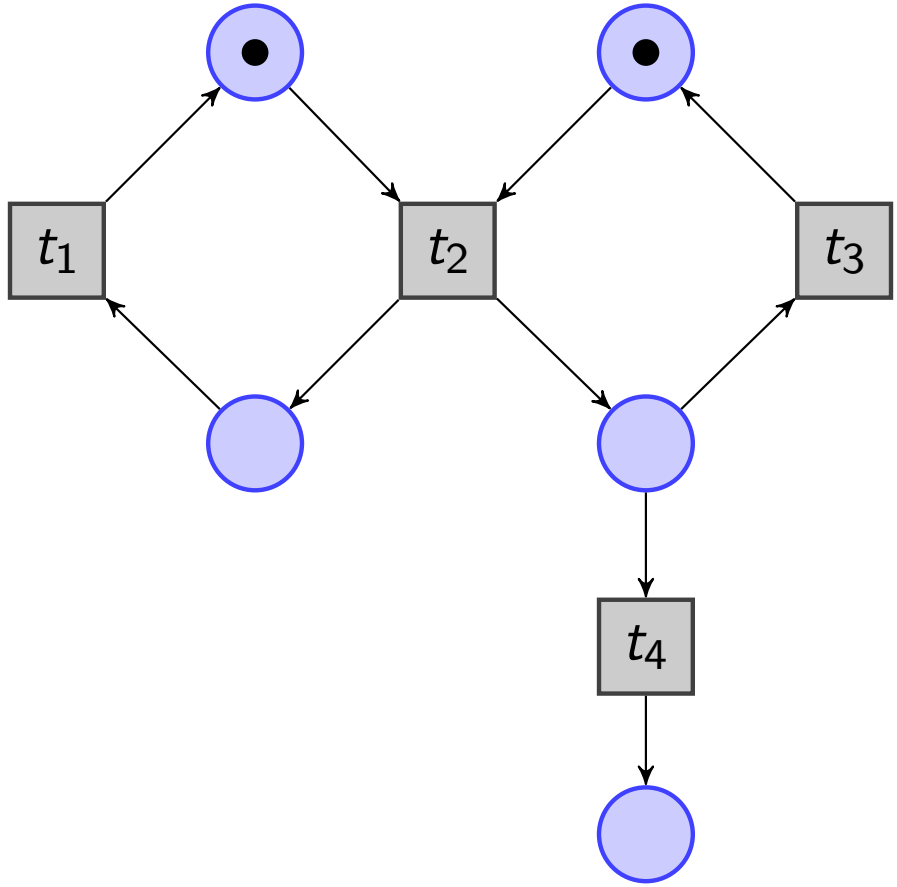


Vorbedingungen

Nachbedingungen

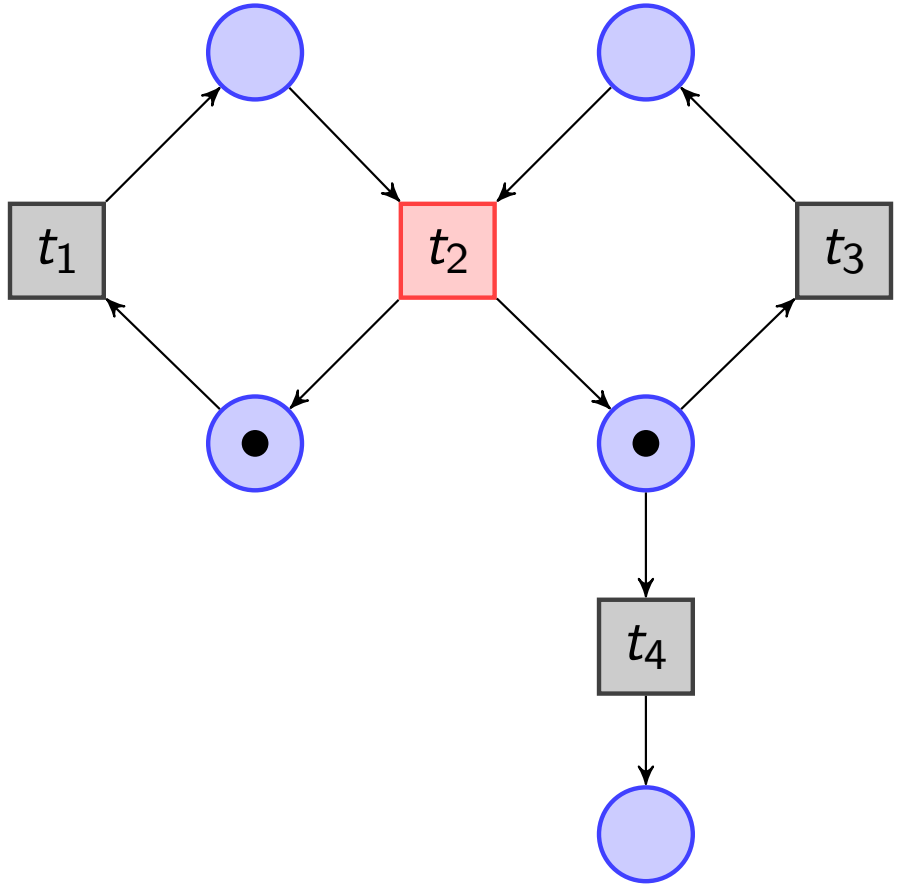
- Vorbedingungen sind die Marken, die konsumiert werden
- Nachbedingungen sind die Marken, die erzeugt werden
- Das Entfernen der Marken der Vorbedingungen und das Erzeugen der Marken der Nachbedingungen nennt man Schalten bzw. Feuern der Transition.

Petrinetze: Beispiel



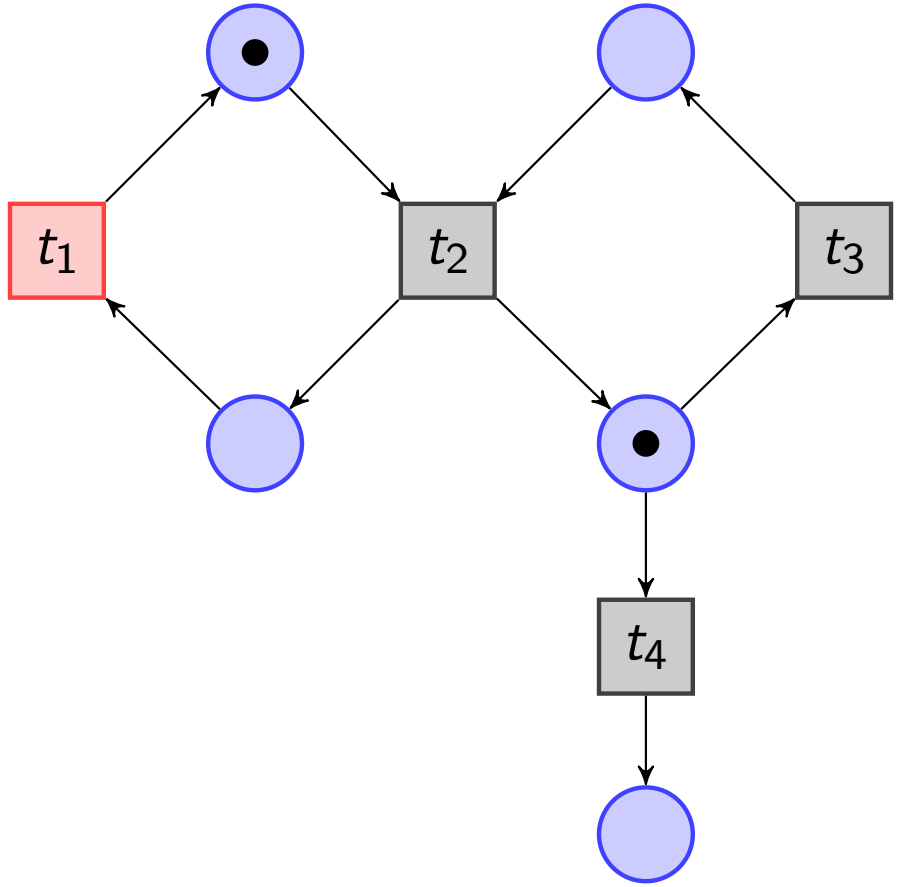
Übernommen von Barbara König, Vorlesung „Modellierung nebenläufiger Systeme“, Uni Duisburg-Essen, 2011

Petrinetze: Beispiel



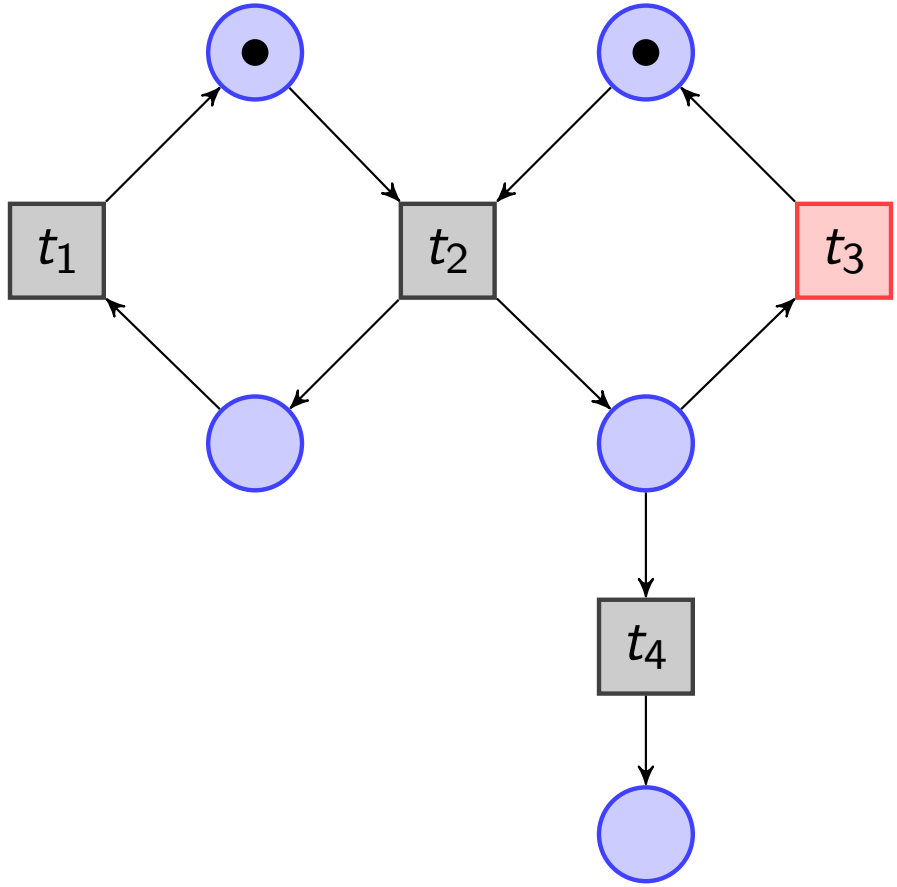
Übernommen von Barbara König, Vorlesung „Modellierung nebenläufiger Systeme“, Uni Duisburg-Essen, 2011

Petrinetze: Beispiel



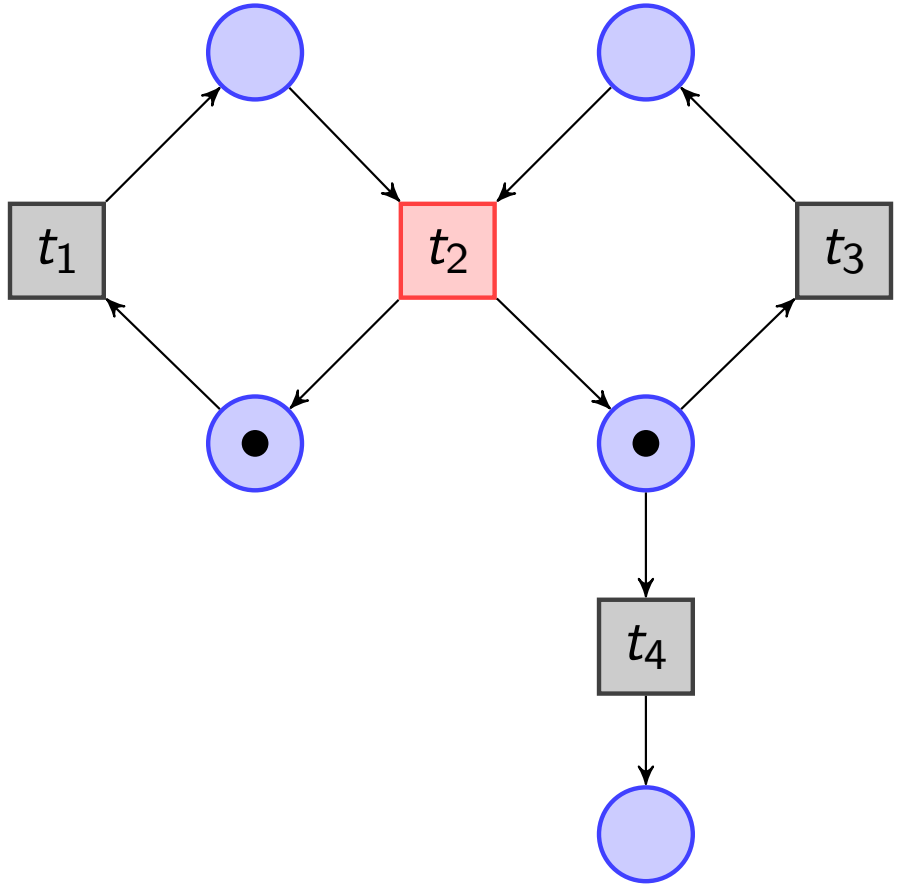
Übernommen von Barbara König, Vorlesung „Modellierung nebenläufiger Systeme“, Uni Duisburg-Essen, 2011

Petrinetze: Beispiel



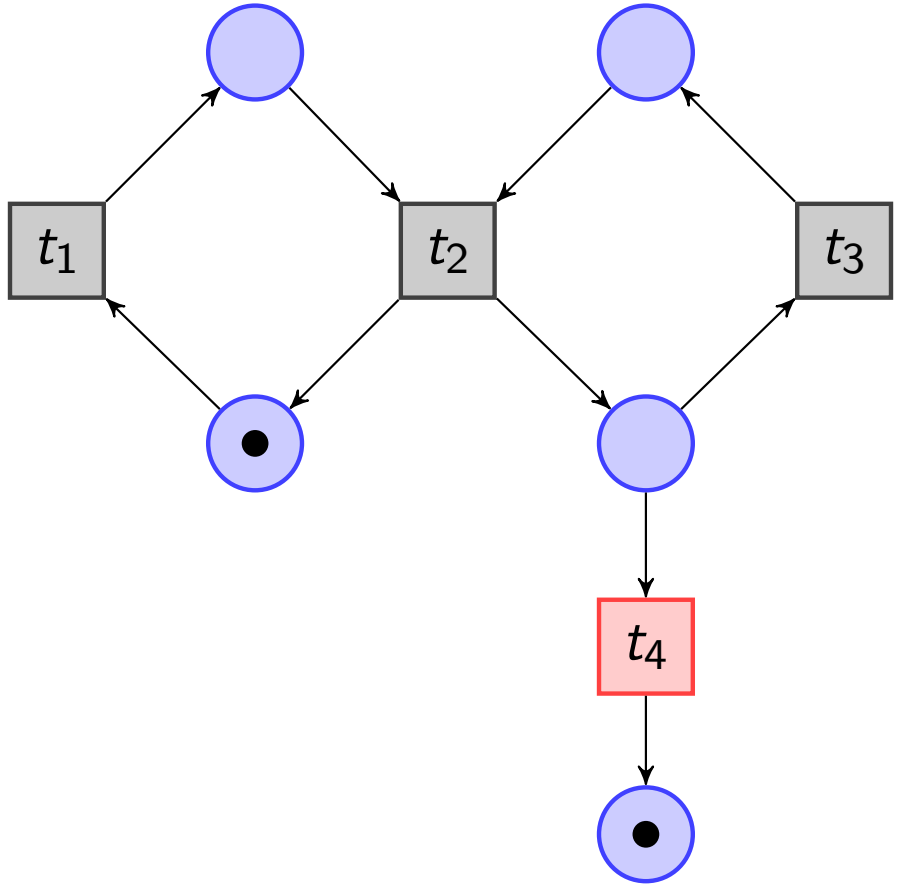
Übernommen von Barbara König, Vorlesung „Modellierung nebenläufiger Systeme“, Uni Duisburg-Essen, 2011

Petrinetze: Beispiel



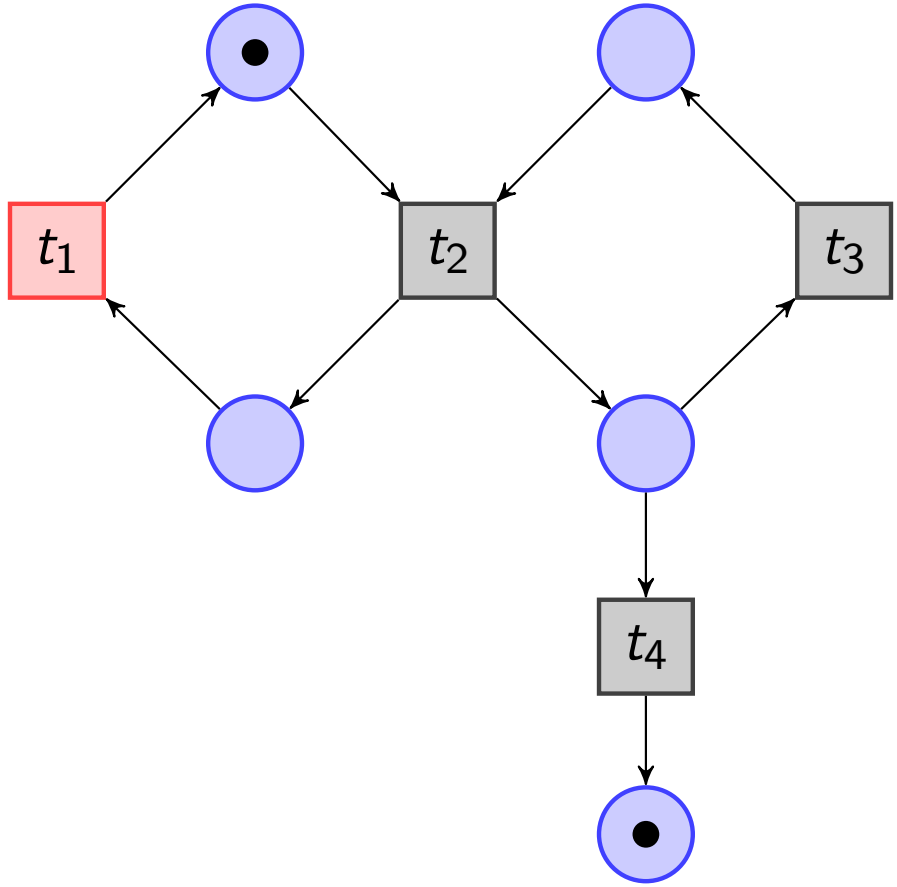
Übernommen von Barbara König, Vorlesung „Modellierung nebenläufiger Systeme“, Uni Duisburg-Essen, 2011

Petrinetze: Beispiel



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

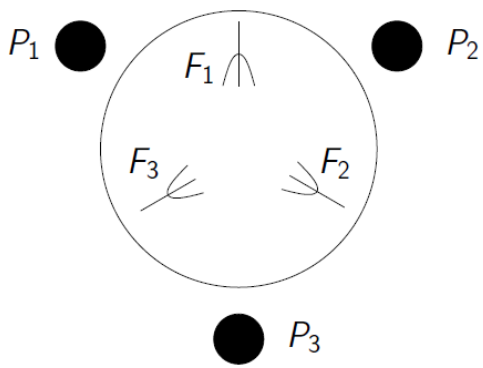
Petrinetze: Beispiel



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

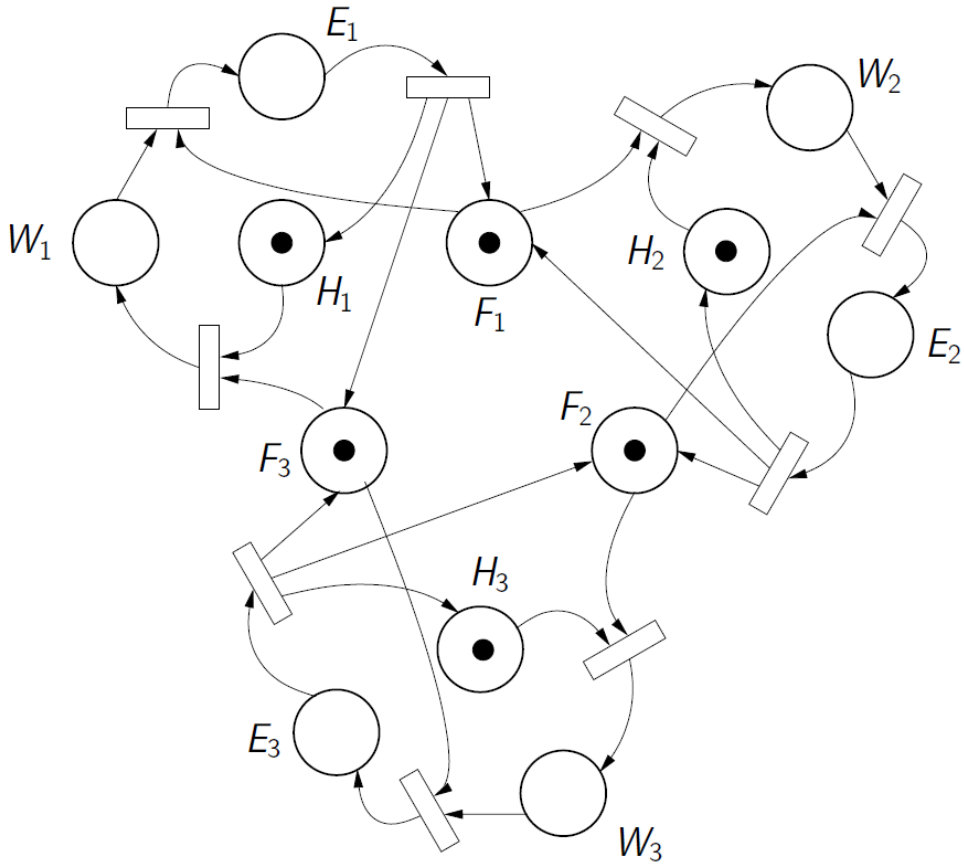
Dining Philosophers

- Drei Philosophen sitzen um einen runden Tisch, zwischen je zwei Philosophen liegt eine Gabel.
- Wird ein Philosoph hungrig, hält er sich an folgenden Ablauf:
  1. Er nimmt die Gabel zu seiner Rechten.
  2. Er nimmt die Gabel zu seiner Linken.
  3. Er isst.
  4. Er legt beide Gabeln zurück.
- Sollte eine Gabel nicht an ihrem Platz liegen, wenn der Philosoph sie benötigt, so wartet er, bis die Gabel wieder verfügbar ist.



Dining Philosophers

Modellierung als Petrinetz:



Deadlock erreichbar, d.h., eine Markierung, bei der keine Transition mehr geschaltet werden kann.

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

Inhalt

Definitionen

Petrinetz

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

$m_0 \in M$ 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

$m, m' \in M$ ... 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.
- Sei t eine Transition, die für die Markierung m aktiviert ist. Dann kann t schalten (oder feuern), was zu der Nachfolgemarkierung $m' = m \ominus {}^\bullet t \oplus t^\bullet$ führt. Symbolisch $m[t\rangle m'$.
- Eine Markierung m_n heißt erreichbar in einem Netz, falls es eine Folge von Transitionen $t_1, \ldots, t_n$ gibt mit $m_0[t_1\rangle m_1 \ldots m_{n-1}[t_n\rangle m_n$, wobei m_0 die Anfangsmarkierung ist.

Graphische Notation

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.

Inhalt

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 Petrinetz mit $n = 3$ und $m = 1$. Es können maximal 2 Leserprozesse gleichzeitig die Datei lesen.

Beispiel: Leser-Schreiber-Problem

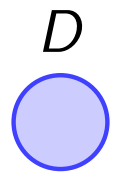
Leser-Schreiber-Problem

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 Petrinetz mit $n = 3$ und $m = 1$. Es können maximal 2 Leserprozesse gleichzeitig die Datei lesen.

Beispiel: Leser-Schreiber-Problem



Beispiel: Leser-Schreiber-Problem

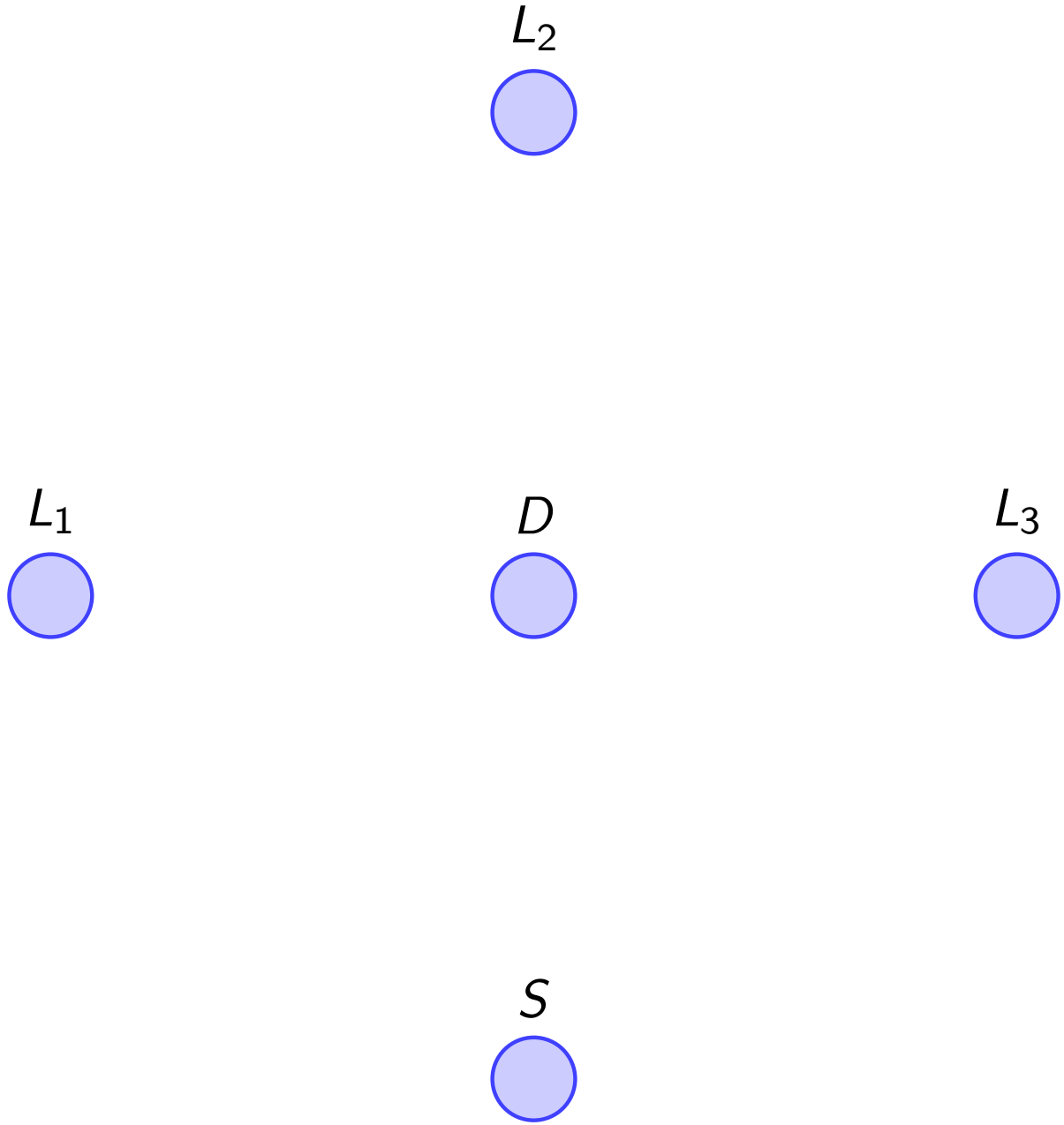
Leser-Schreiber-Problem

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:

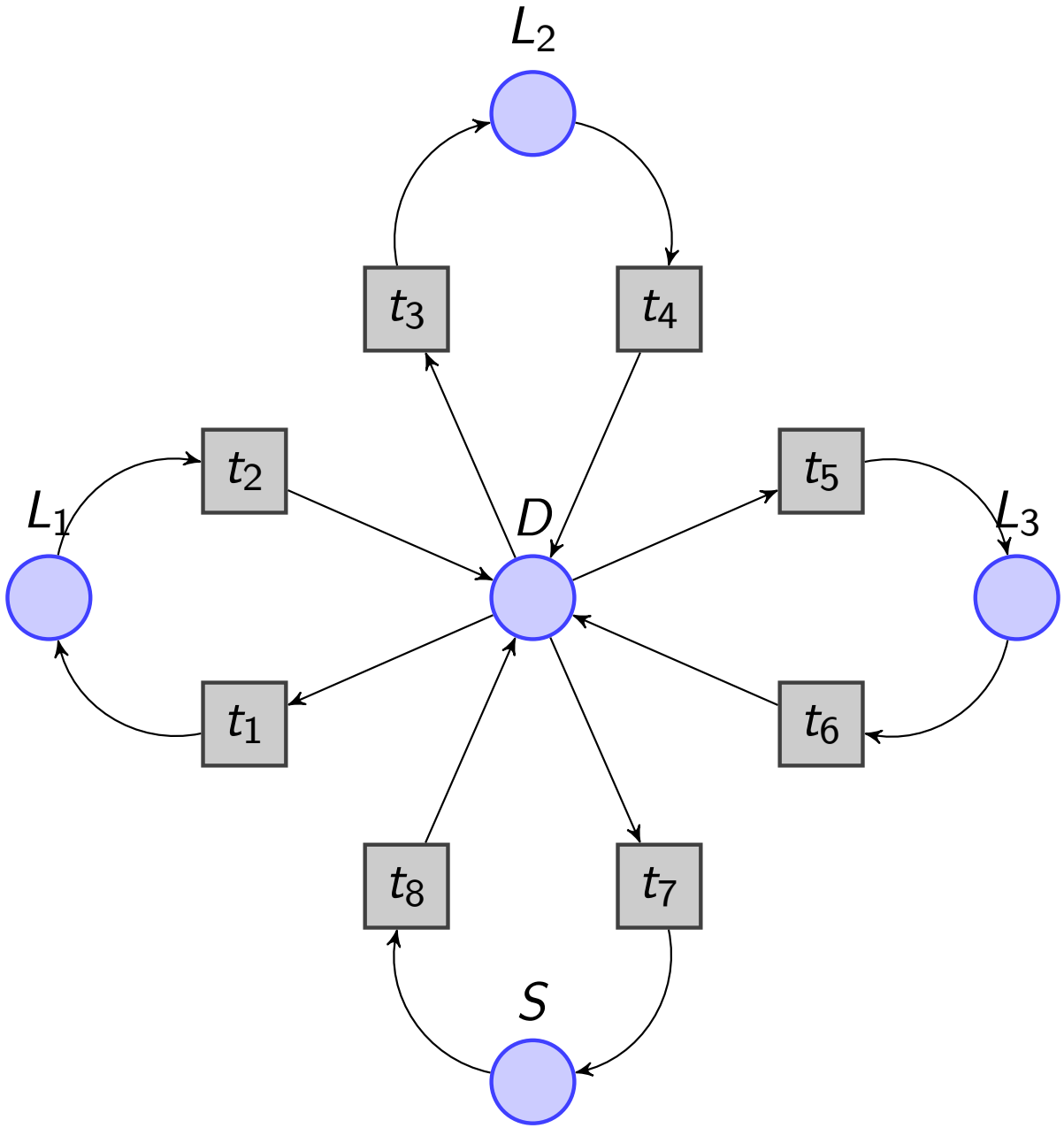
- Es können zur gleichen Zeit mehrere Leserprozesse auf die Datei zugreifen.
- Ein Schreiberprozess darf nur dann auf die Datei zugreifen, wenn gerade kein anderer Prozess (lesend oder schreibend) auf die Datei zugreift.

Modellieren Sie das Leser-Schreiber-Problem als Petrinetz 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

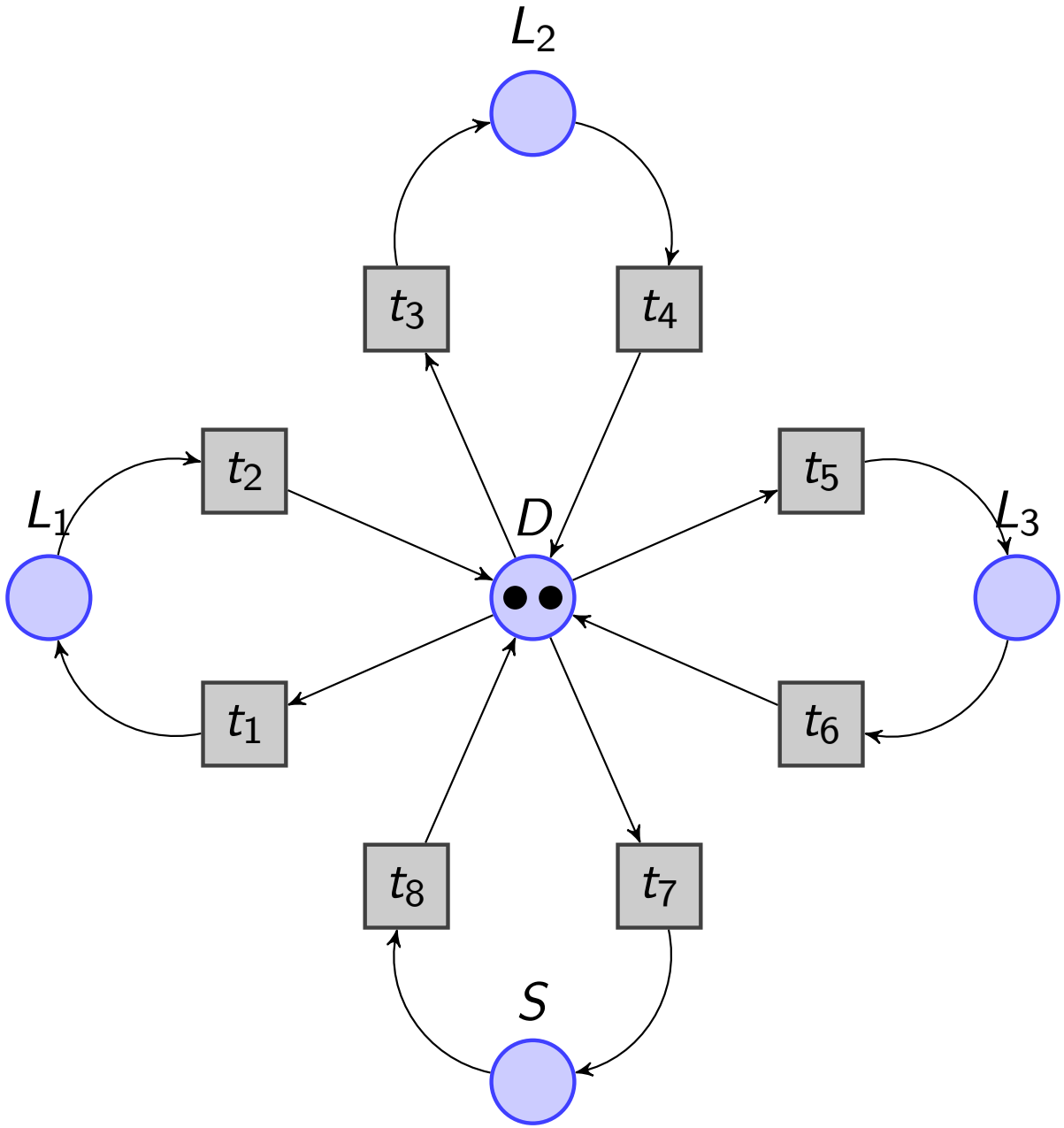
Leser-Schreiber-Problem

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 Petrinetz mit $n = 3$ und $m = 1$. Es können maximal 2 Leserprozesse gleichzeitig die Datei lesen.

Beispiel: Leser-Schreiber-Problem



Beispiel: Leser-Schreiber-Problem

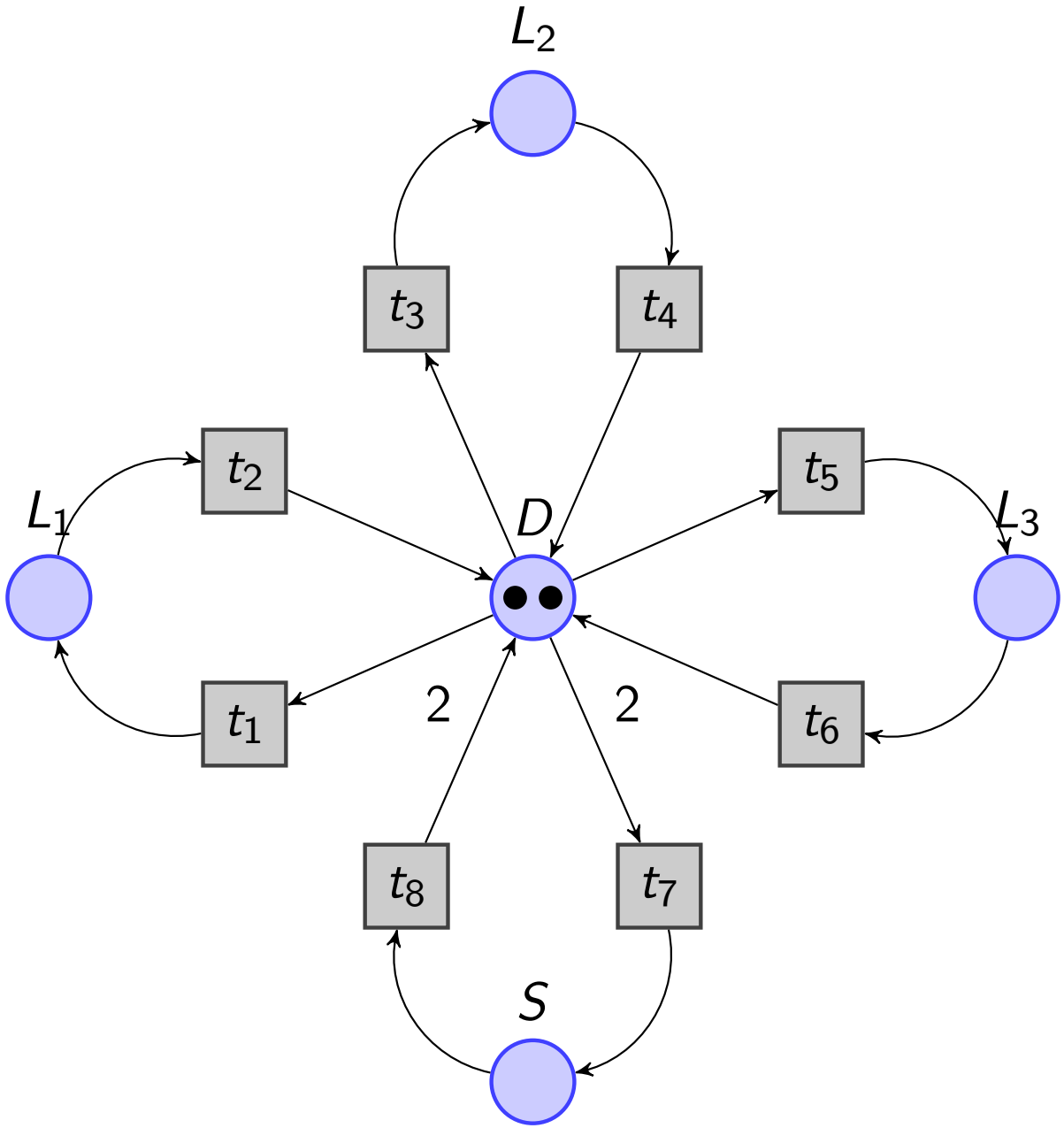
Leser-Schreiber-Problem

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:

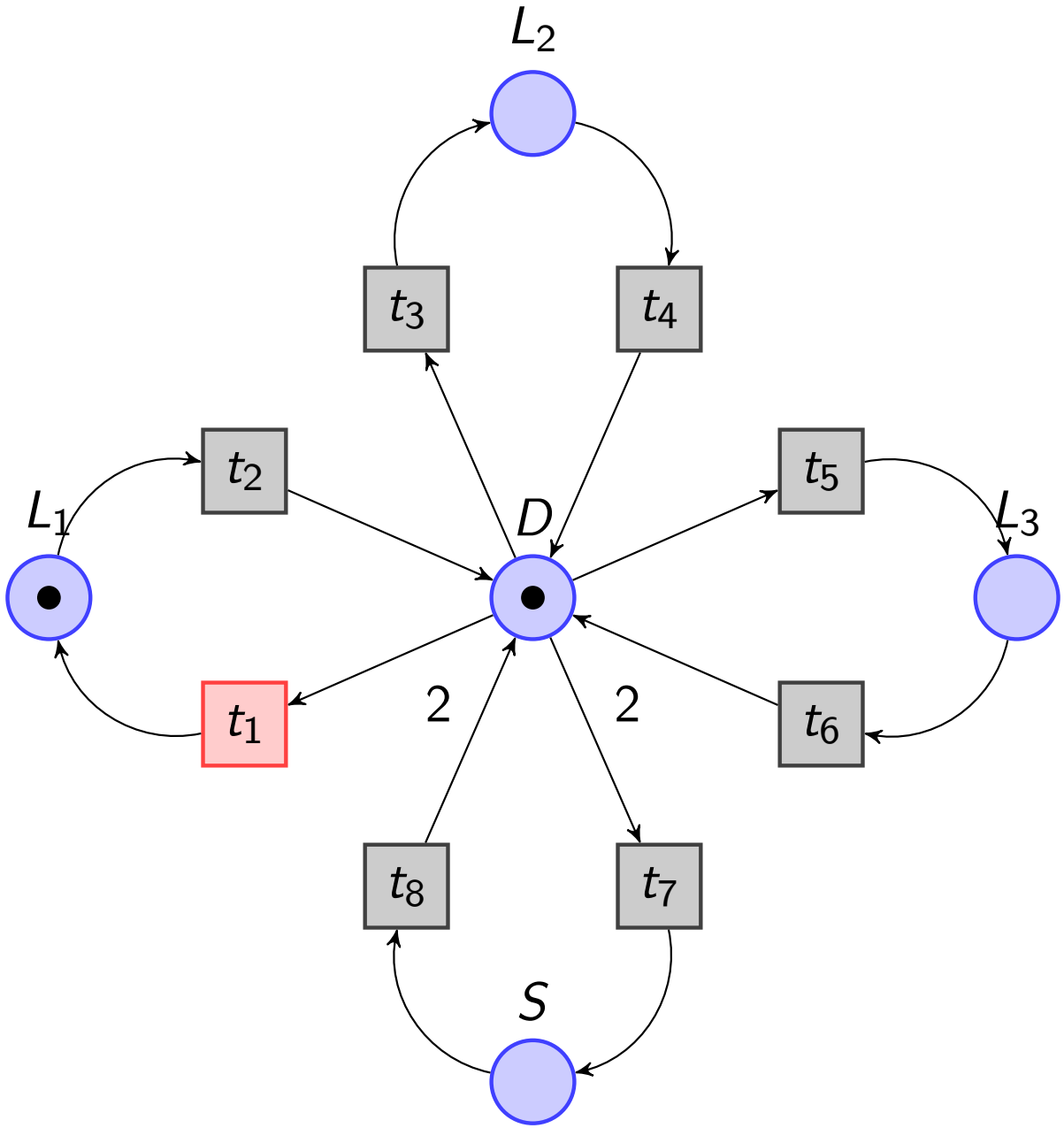
- Es können zur gleichen Zeit mehrere Leserprozesse auf die Datei zugreifen.
- Ein Schreiberprozess darf nur dann auf die Datei zugreifen, wenn gerade kein anderer Prozess (lesend oder schreibend) auf die Datei zugreift.

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

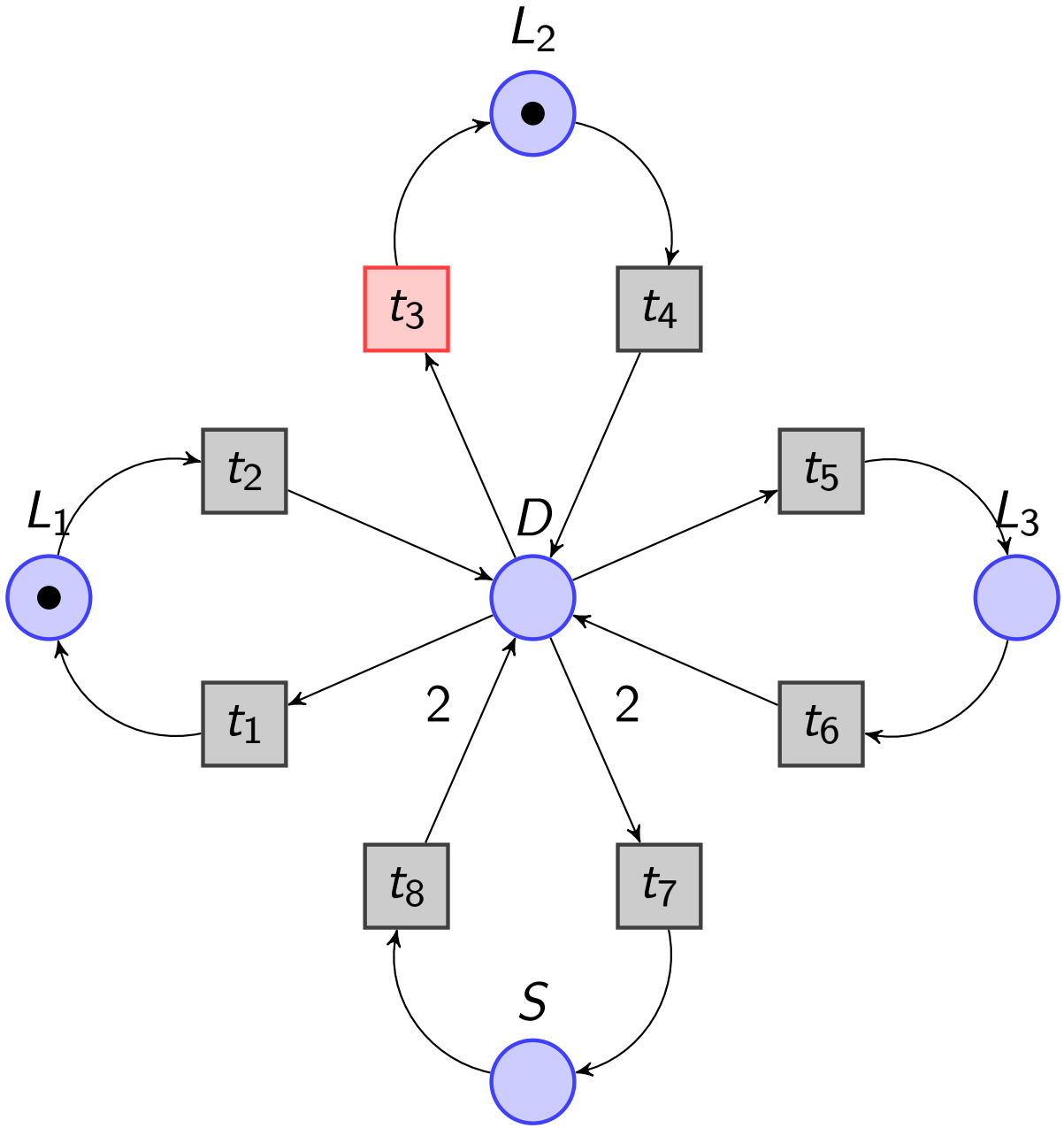
Beispiel: Leser-Schreiber-Problem



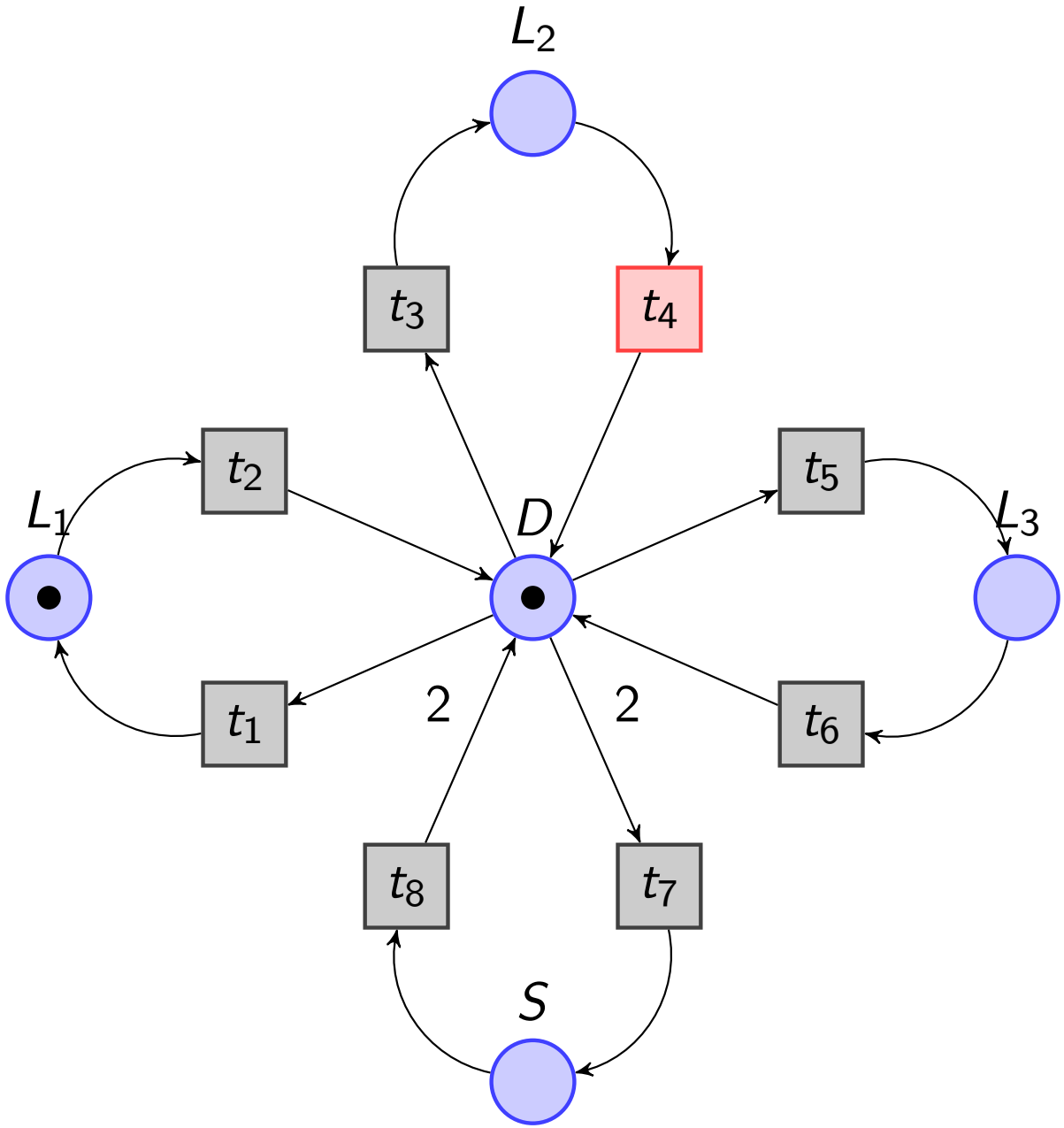
Beispiel: Leser-Schreiber-Problem



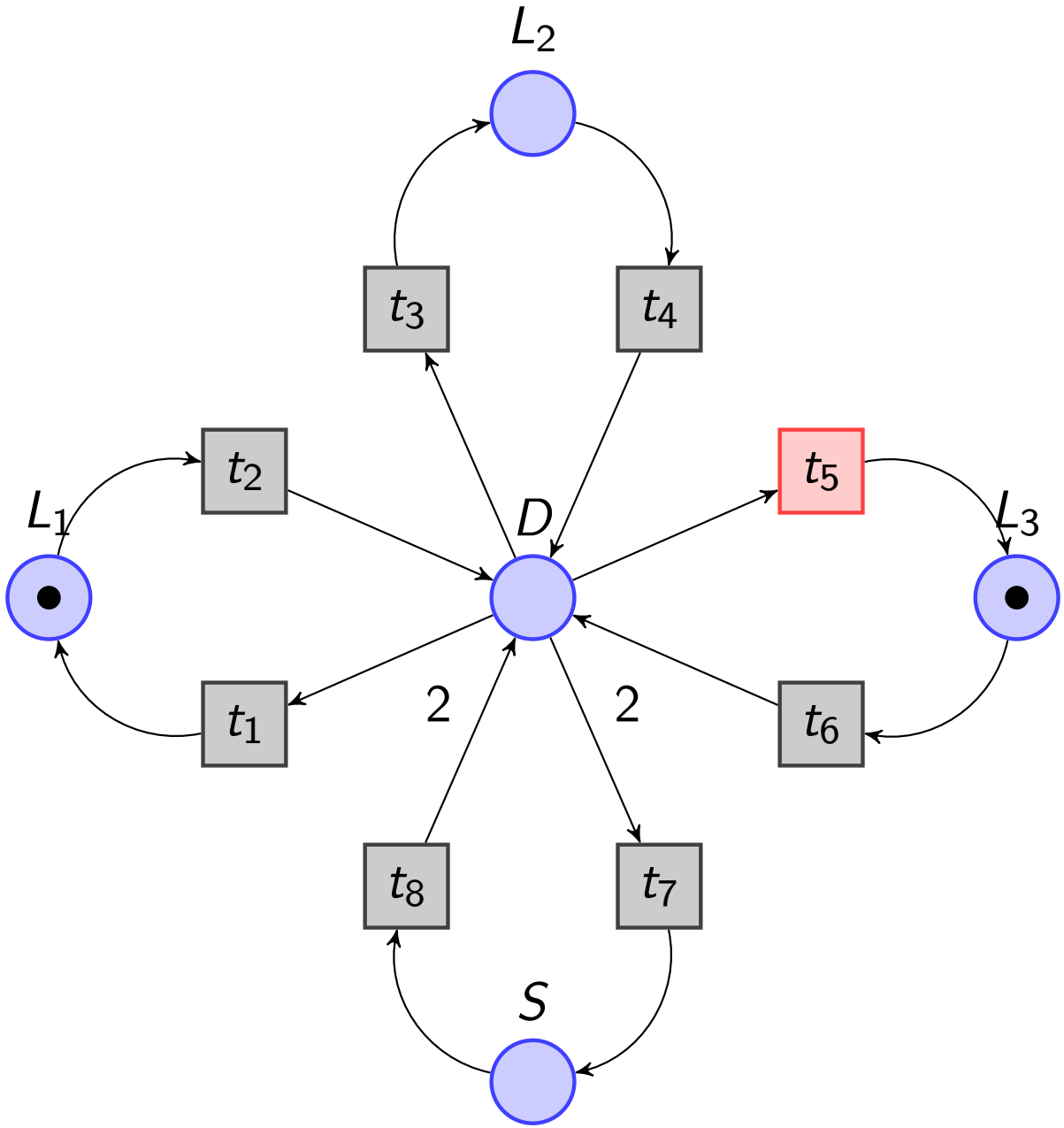
Beispiel: Leser-Schreiber-Problem



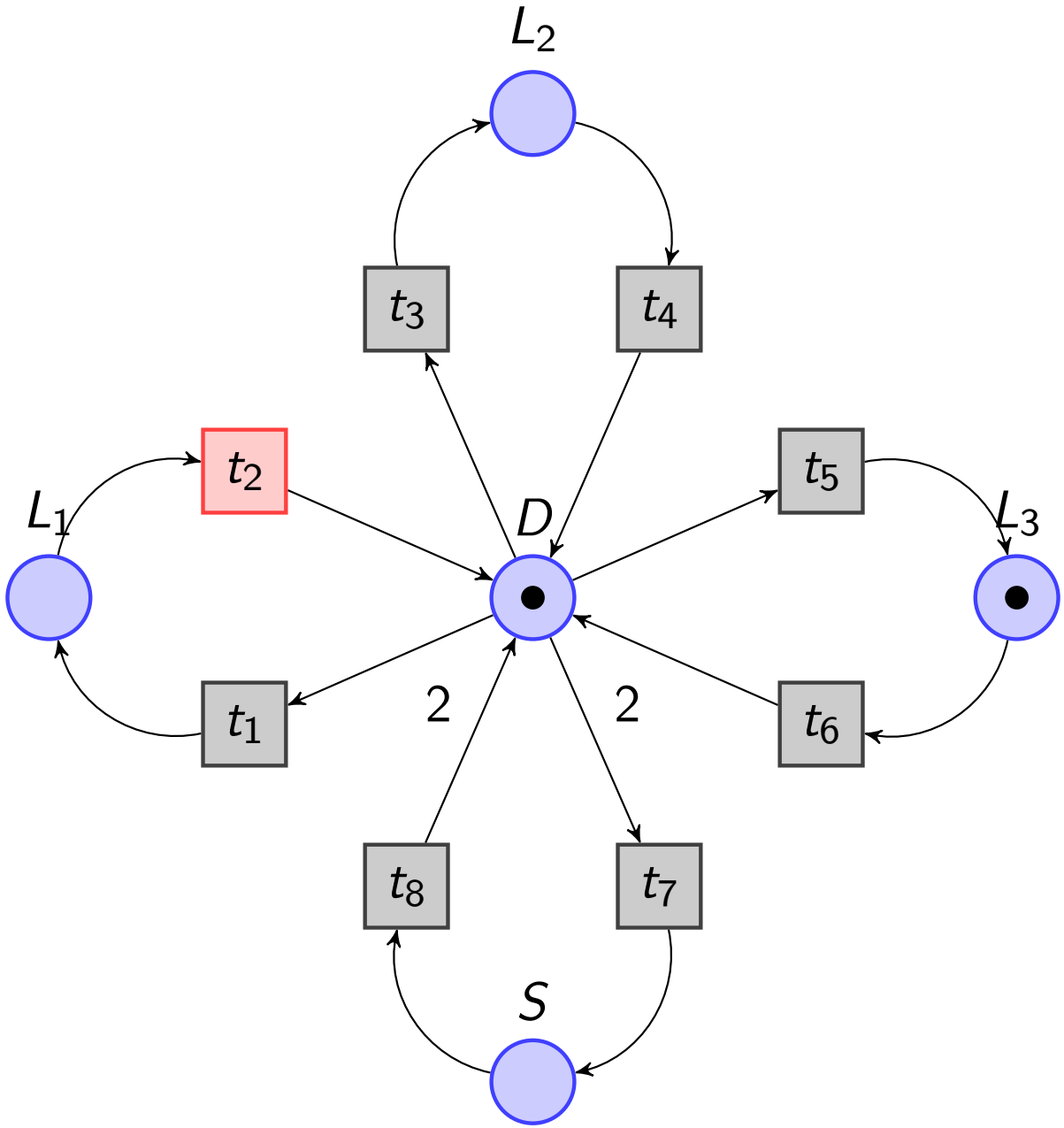
Beispiel: Leser-Schreiber-Problem



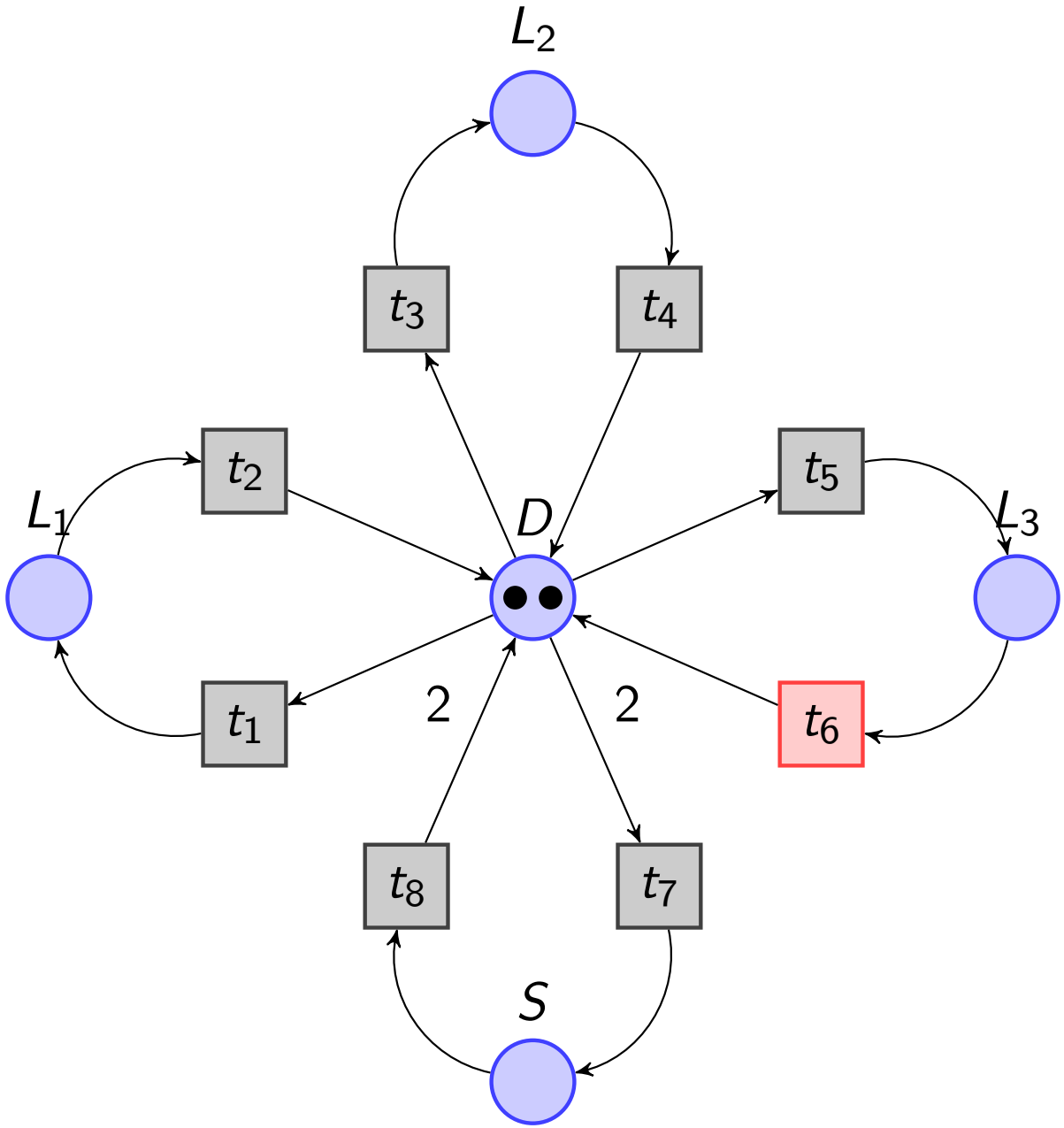
Beispiel: Leser-Schreiber-Problem



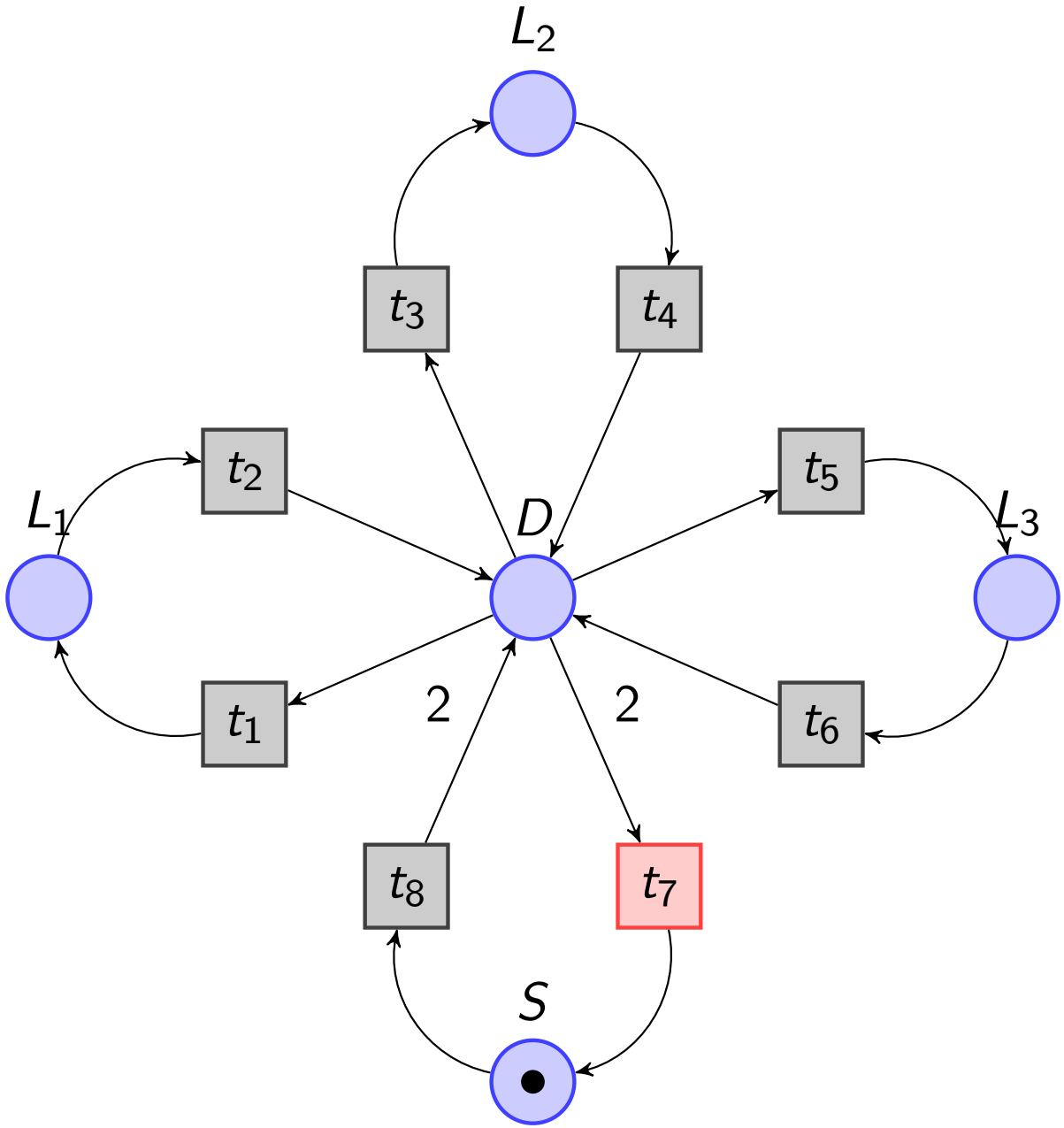
Beispiel: Leser-Schreiber-Problem



Beispiel: Leser-Schreiber-Problem



Beispiel: Leser-Schreiber-Problem



Beispiel: Leser-Schreiber-Problem