Work in progress

# Petri-Netze

Ute Bormann, TI2

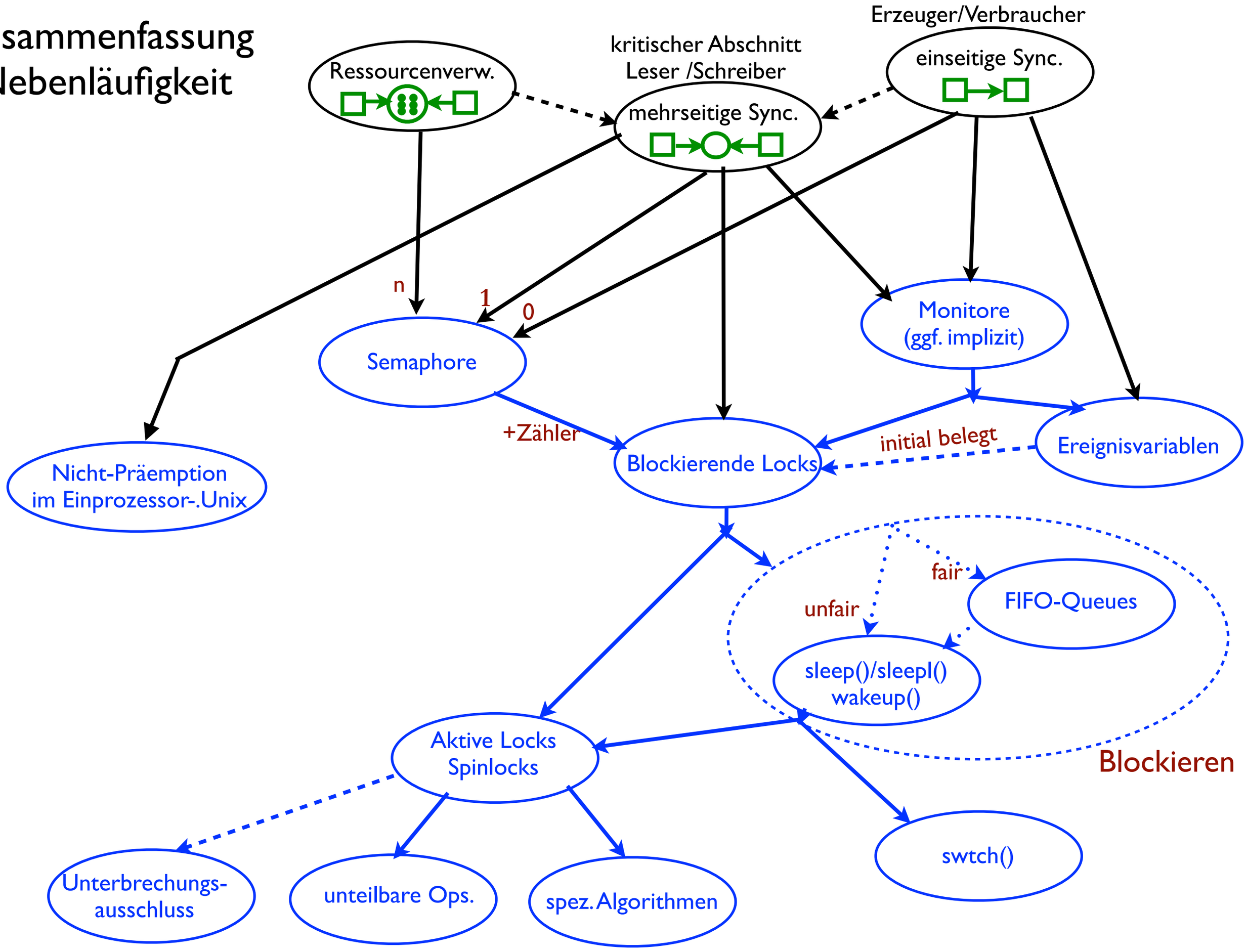
2023-10-13

# Inhalt

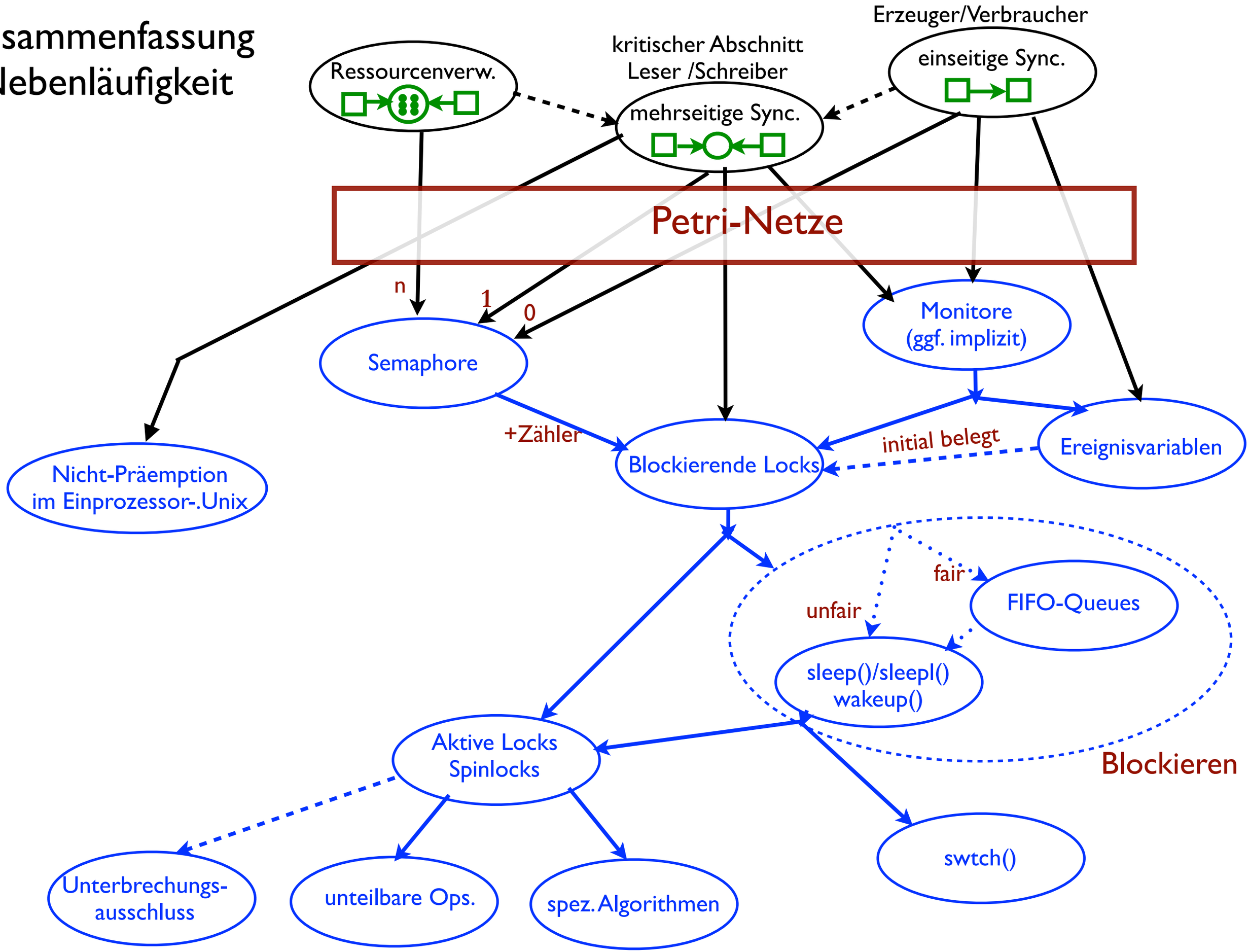
1. Aufbau von Petri-Netzen
2. Synchronisationaussagen in Petri-Netzen

# Teil 1:
# Aufbau von Petri-Netzen

# Zusammenfassung Nebenläufigkeit



# Zusammenfassung Nebenläufigkeit



# Petri-Netze (Petri, ab 1962, viele Varianten)

- Grafische Veranschaulichung von Synchronisationszusammenhängen
- Im folgenden nur „pragmatischer Überblick“, keine Theorie

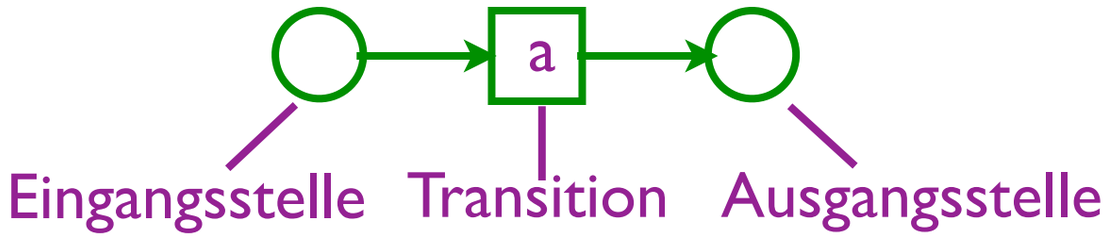
  ⇒ LV Petri-Netze

# Petri-Netze (Petri, ab 1962, viele Varianten)

- Grafische Veranschaulichung von Synchronisationszusammenhängen
- Im folgenden nur „pragmatischer Überblick“, keine Theorie

  ⇒ LV Petri-Netze

- Grundprinzip: Beschreibung eines Systems als Netz von Zuständen (Stellen) und Zustandsübergängen (Transitionen)

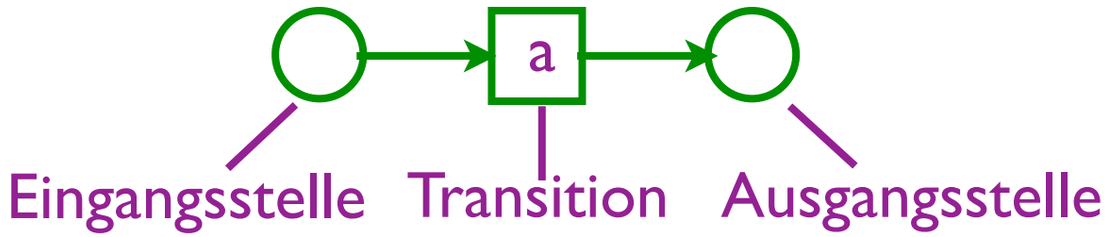



- Beispiel: Ausführung Funktion a() (+ Zustand vorher/nachher)

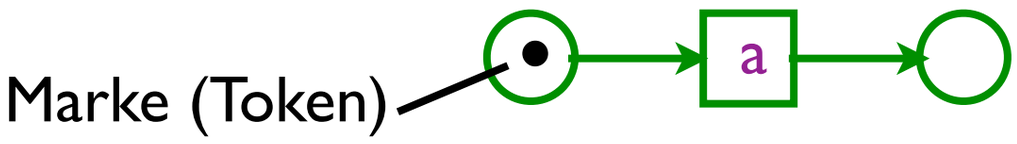
# Petri-Netze (Petri, ab 1962, viele Varianten)

- Grafische Veranschaulichung von Synchronisationszusammenhängen
- Im folgenden nur „pragmatischer Überblick“, keine Theorie

  ⇒ LV Petri-Netze
- Grundprinzip: Beschreibung eines Systems als Netz von Zuständen (Stellen) und Zustandsübergängen (Transitionen)



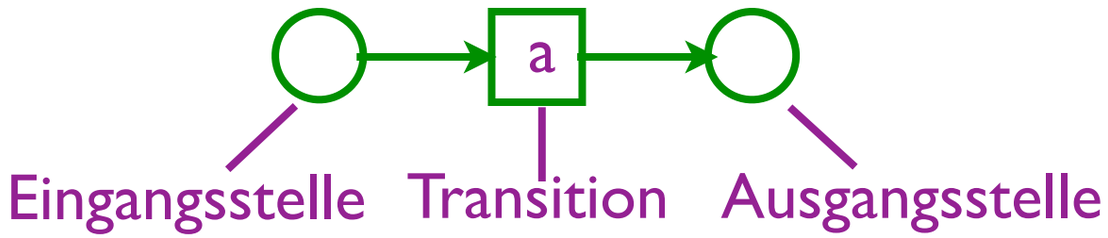

- Beispiel: Ausführung Funktion a() (+ Zustand vorher/nachher)
- Aktueller Ausführungszustand durch Markierung beschreibbar



# Petri-Netze (Petri, ab 1962, viele Varianten)

- Grafische Veranschaulichung von Synchronisationszusammenhängen
- Im folgenden nur „pragmatischer Überblick“, keine Theorie

  ⇒ LV Petri-Netze
- Grundprinzip: Beschreibung eines Systems als Netz von Zuständen (Stellen) und Zustandsübergängen (Transitionen)

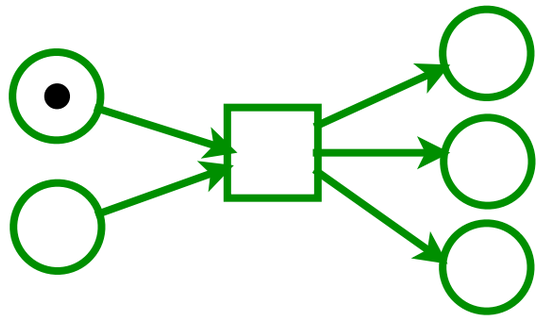



- Beispiel: Ausführung Funktion a() (+ Zustand vorher/nachher)
- Aktueller Ausführungszustand durch Markierung beschreibbar

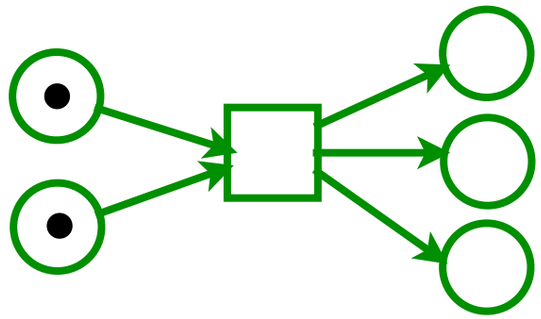

nachher

- Zustandswechsel: „Die Transition schaltet/feuert.“ (i.d.R. in „Nullzeit“)

- Transitionen können mehrere Eingangs-/Ausgangsbedingungen haben

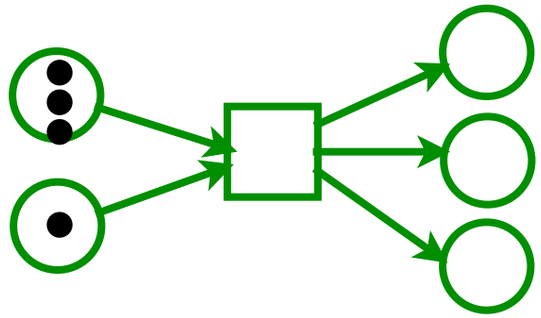


- Transitionen können mehrere Eingangs-/Ausgangsbedingungen haben



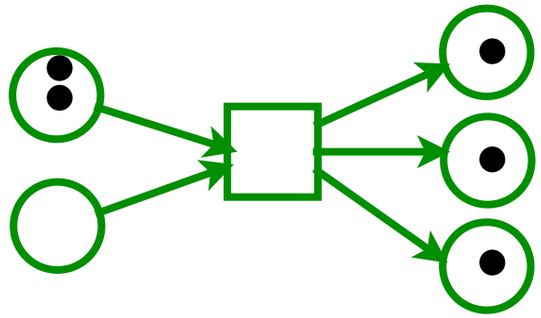
- Zum Schalten müssen alle Eingangsbedingungen eingetreten sein (Marken in allen Eingangsstellen)

- Transitionen können mehrere Eingangs-/Ausgangsbedingungen haben



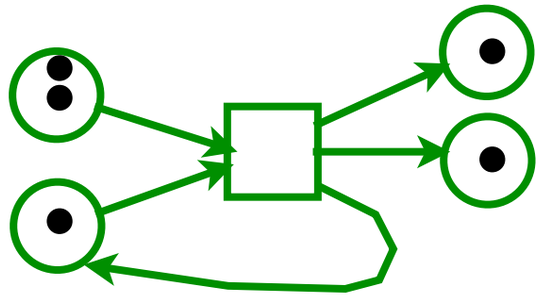
- Zum Schalten müssen alle Eingangsbedingungen eingetreten sein (Marken in allen Eingangsstellen)
- Stellen können mehr als eine Marke besitzen (z.B. mehrere Prozesse in diesem Ausführungszustand oder mehrere Ressourcen vorhanden)

- Transitionen können mehrere Eingangs-/Ausgangsbedingungen haben



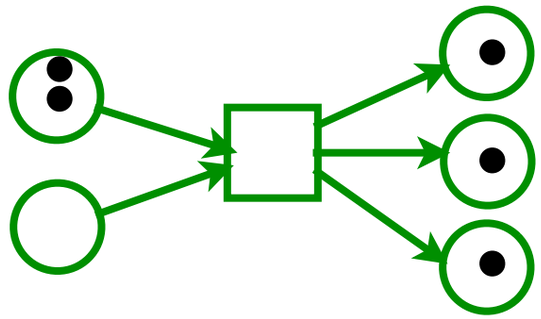
- Zum Schalten müssen alle Eingangsbedingungen eingetreten sein (Marken in allen Eingangsstellen)
- Stellen können mehr als eine Marke besitzen (z.B. mehrere Prozesse in diesem Ausführungszustand oder mehrere Ressourcen vorhanden)
- Beim Schalten:
  - Je eine Marke von jeder Eingangsstelle abziehen
  - Je eine Marke zu jeder Ausgangsstelle hinzufügen

- Transitionen können mehrere Eingangs-/Ausgangsbedingungen haben

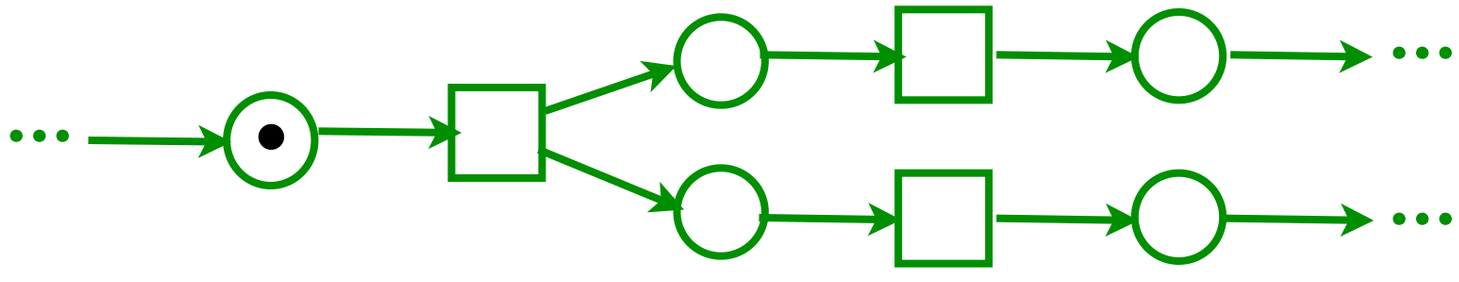


- Zum Schalten müssen alle Eingangsbedingungen eingetreten sein (Marken in allen Eingangsstellen)
- Stellen können mehr als eine Marke besitzen (z.B. mehrere Prozesse in diesem Ausführungszustand oder mehrere Ressourcen vorhanden)
- Beim Schalten:
  - Je eine Marke von jeder Eingangsstelle abziehen
  - Je eine Marke zu jeder Ausgangsstelle hinzufügen

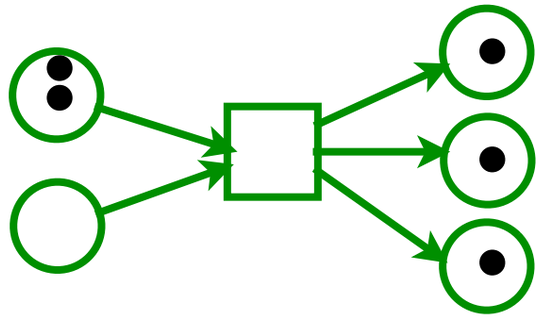
- Transitionen können mehrere Eingangs-/Ausgangsbedingungen haben



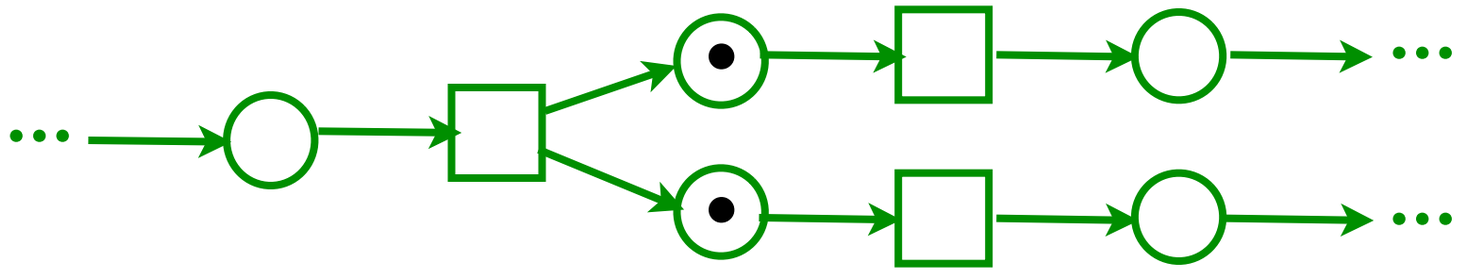
- Zum Schalten müssen alle Eingangsbedingungen eingetreten sein (Marken in allen Eingangsstellen)
- Stellen können mehr als eine Marke besitzen (z.B. mehrere Prozesse in diesem Ausführungszustand oder mehrere Ressourcen vorhanden)
- Beim Schalten:
  - Je eine Marke von jeder Eingangsstelle abziehen
  - Je eine Marke zu jeder Ausgangsstelle hinzufügen
- Ausgangsstellen können Eingangsstellen für weitere Transitionen sein



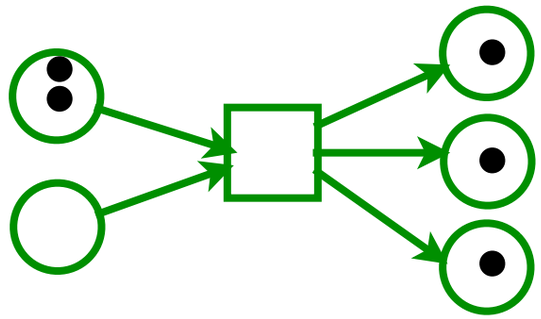
- Transitionen können mehrere Eingangs-/Ausgangsbedingungen haben



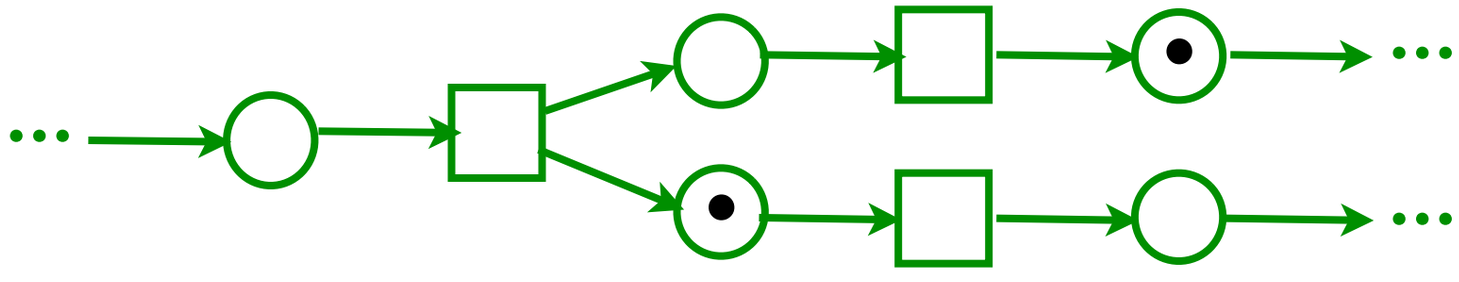
- Zum Schalten müssen alle Eingangsbedingungen eingetreten sein (Marken in allen Eingangsstellen)
- Stellen können mehr als eine Marke besitzen (z.B. mehrere Prozesse in diesem Ausführungszustand oder mehrere Ressourcen vorhanden)
- Beim Schalten:
  - Je eine Marke von jeder Eingangsstelle abziehen
  - Je eine Marke zu jeder Ausgangsstelle hinzufügen
- Ausgangsstellen können Eingangsstellen für weitere Transitionen sein



- Transitionen können mehrere Eingangs-/Ausgangsbedingungen haben

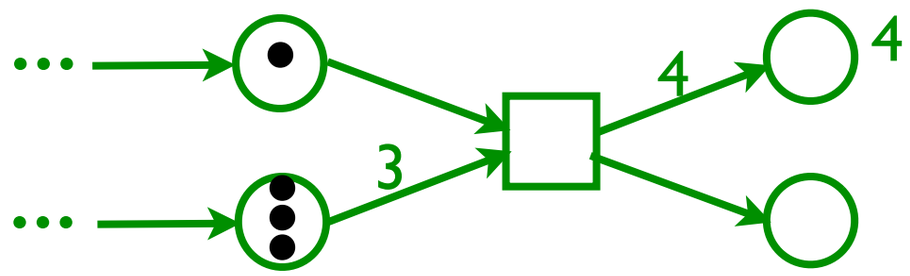


- Zum Schalten müssen alle Eingangsbedingungen eingetreten sein (Marken in allen Eingangsstellen)
- Stellen können mehr als eine Marke besitzen (z.B. mehrere Prozesse in diesem Ausführungszustand oder mehrere Ressourcen vorhanden)
- Beim Schalten:
  - Je eine Marke von jeder Eingangsstelle abziehen
  - Je eine Marke zu jeder Ausgangsstelle hinzufügen
- Ausgangsstellen können Eingangsstellen für weitere Transitionen sein



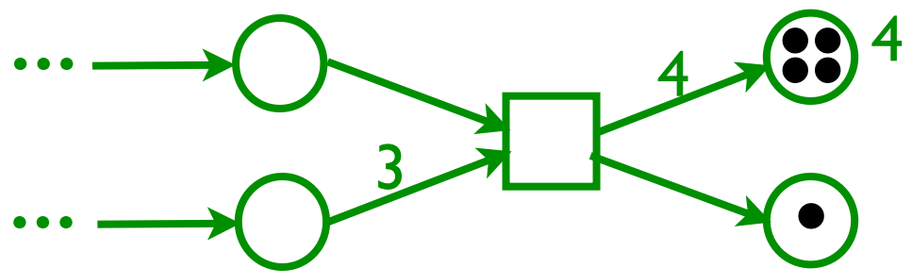
- Stellen können eine Maximalkapazität haben
  ⇒ beschränkte Anzahl von Marken

- Kanten können eine Gewichtung haben
  ⇒ ziehen mehrere Marken ab bzw. fügen mehrere Marken hinzu



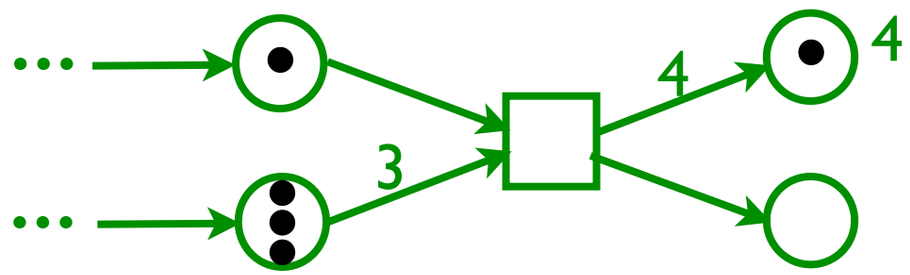
- Stellen können eine Maximalkapazität haben
  ⇒ beschränkte Anzahl von Marken

- Kanten können eine Gewichtung haben
  ⇒ ziehen mehrere Marken ab bzw. fügen mehrere Marken hinzu



- Stellen können eine Maximalkapazität haben
  ⇒ beschränkte Anzahl von Marken

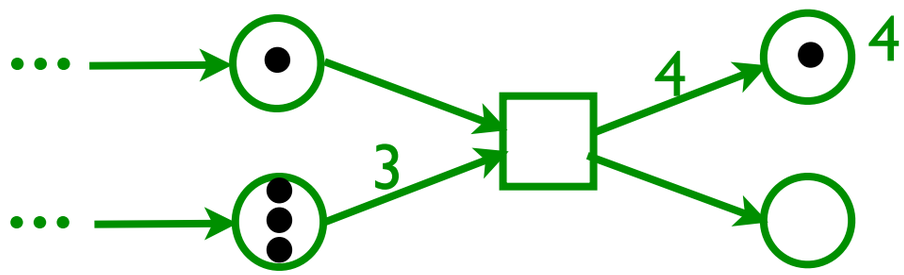
- Kanten können eine Gewichtung haben
  ⇒ ziehen mehrere Marken ab bzw. fügen mehrere Marken hinzu



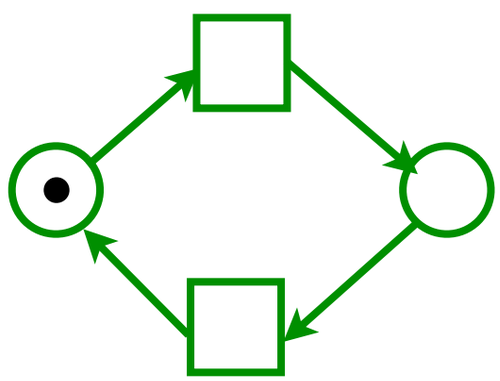

implizite zusätzliche Eingangsbedingung
⇒ hier kein Schalten möglich

- Stellen können eine Maximalkapazität haben
  ⇒ beschränkte Anzahl von Marken

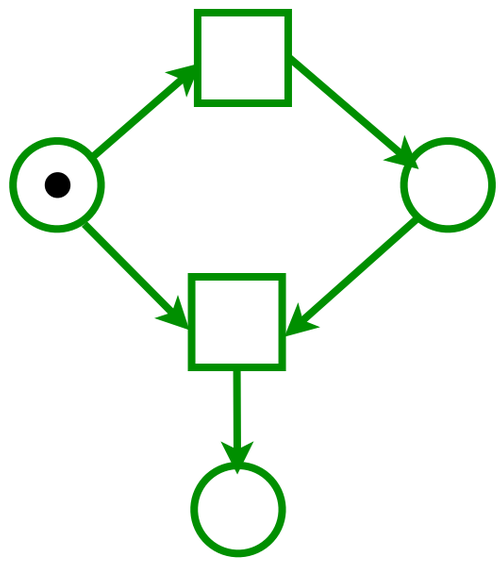
- Kanten können eine Gewichtung haben
  ⇒ ziehen mehrere Marken ab bzw. fügen mehrere Marken hinzu




- Petri-Netze können lebendig oder todesgefährdet sein



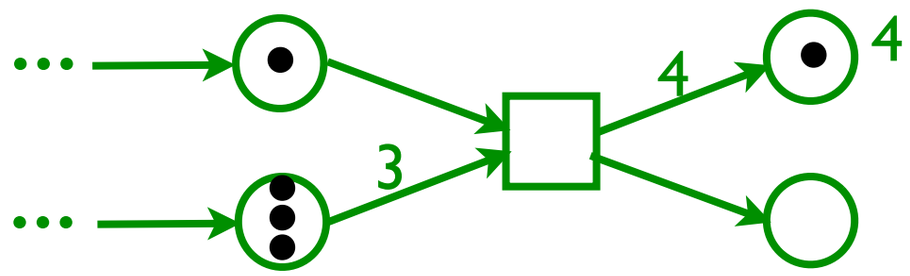
Lebendig
(es gibt immer eine Transition,
die schalten kann)



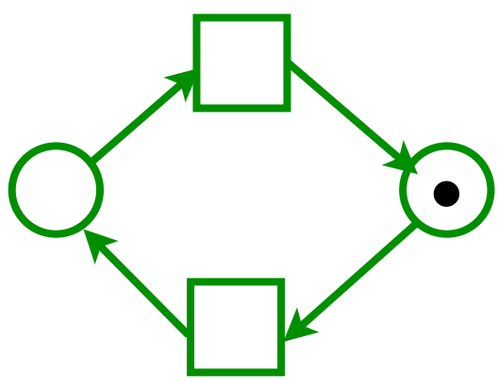
Todesgefährdet

- Stellen können eine Maximalkapazität haben
  ⇒ beschränkte Anzahl von Marken

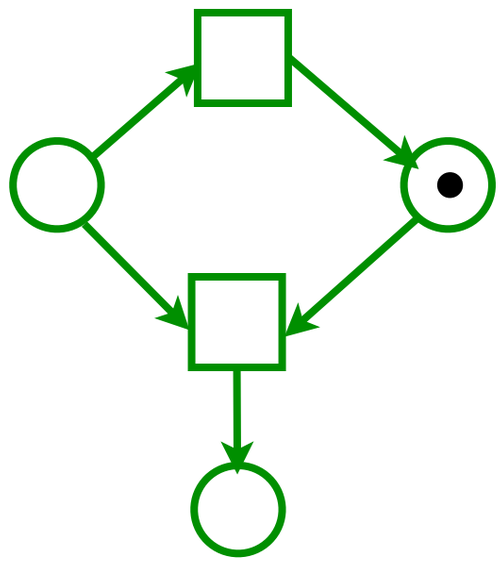
- Kanten können eine Gewichtung haben
  ⇒ ziehen mehrere Marken ab bzw. fügen mehrere Marken hinzu




- Petri-Netze können lebendig oder todesgefährdet sein



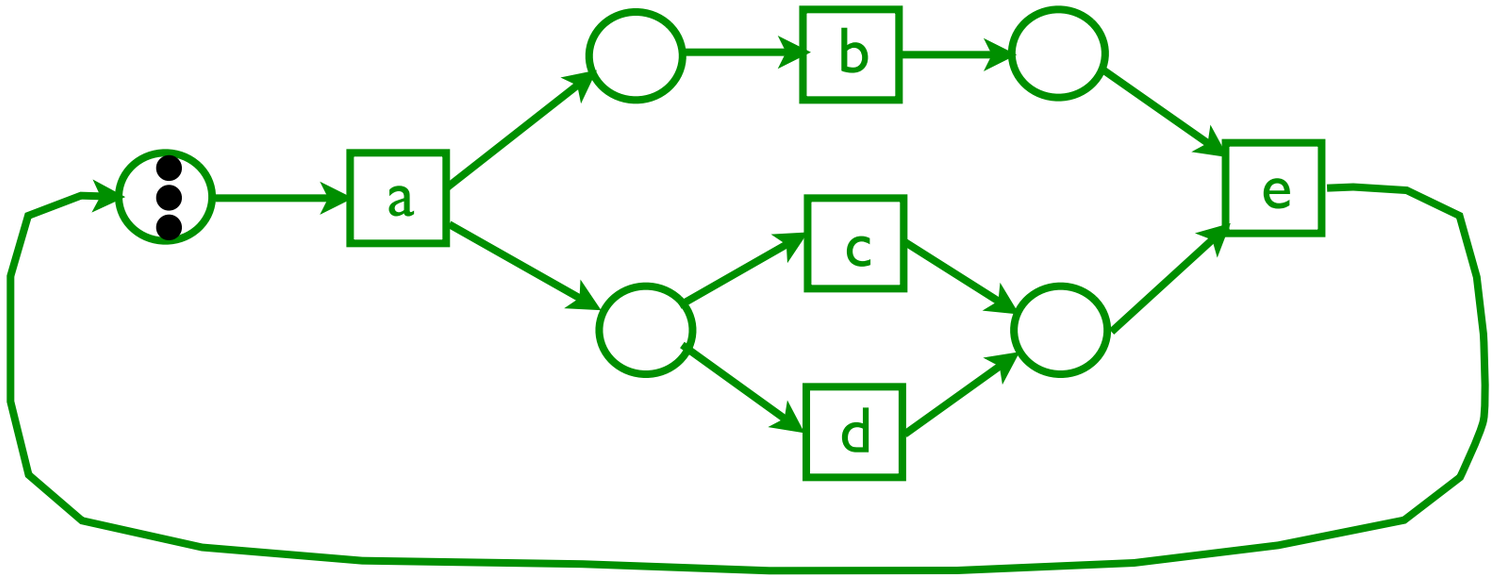
Lebendig
(es gibt immer eine Transition, die schalten kann)



Keine Transition kann noch schalten

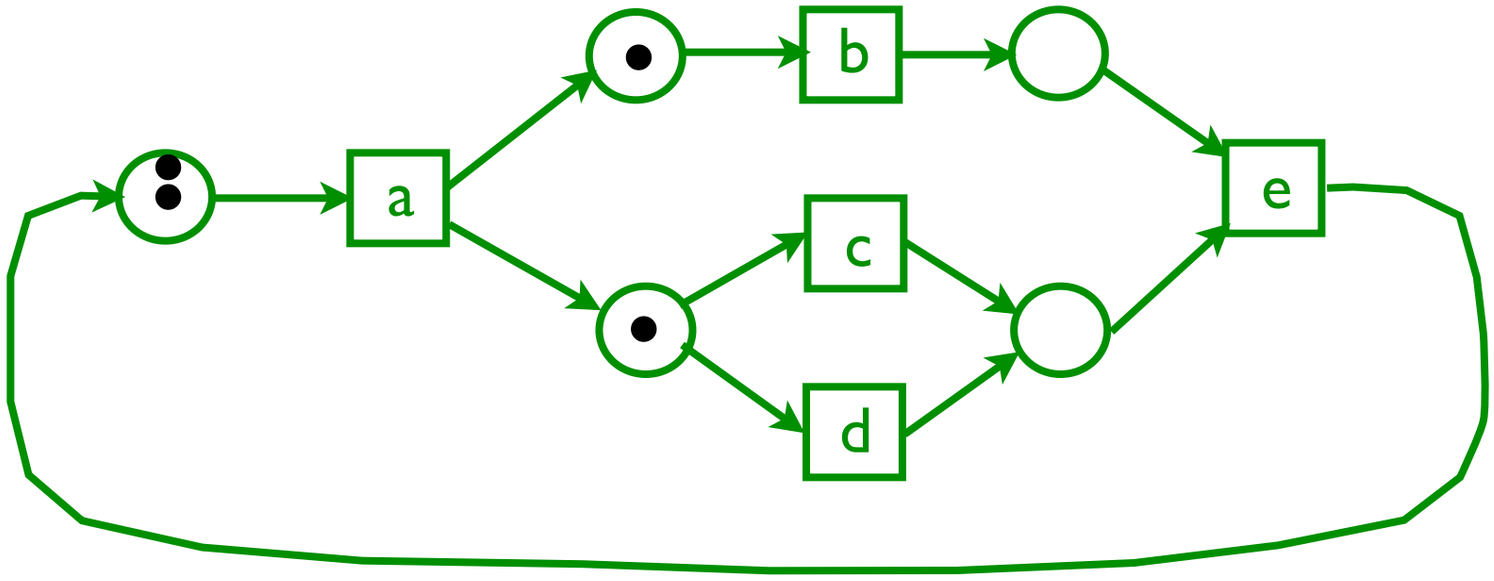
- Daraus komplexe Netze zusammenbaubar

  ⇒ Aussagen über Synchronisationseigenschaften



- Daraus komplexe Netze zusammenbaubar

  ⇒ Aussagen über Synchronisationseigenschaften

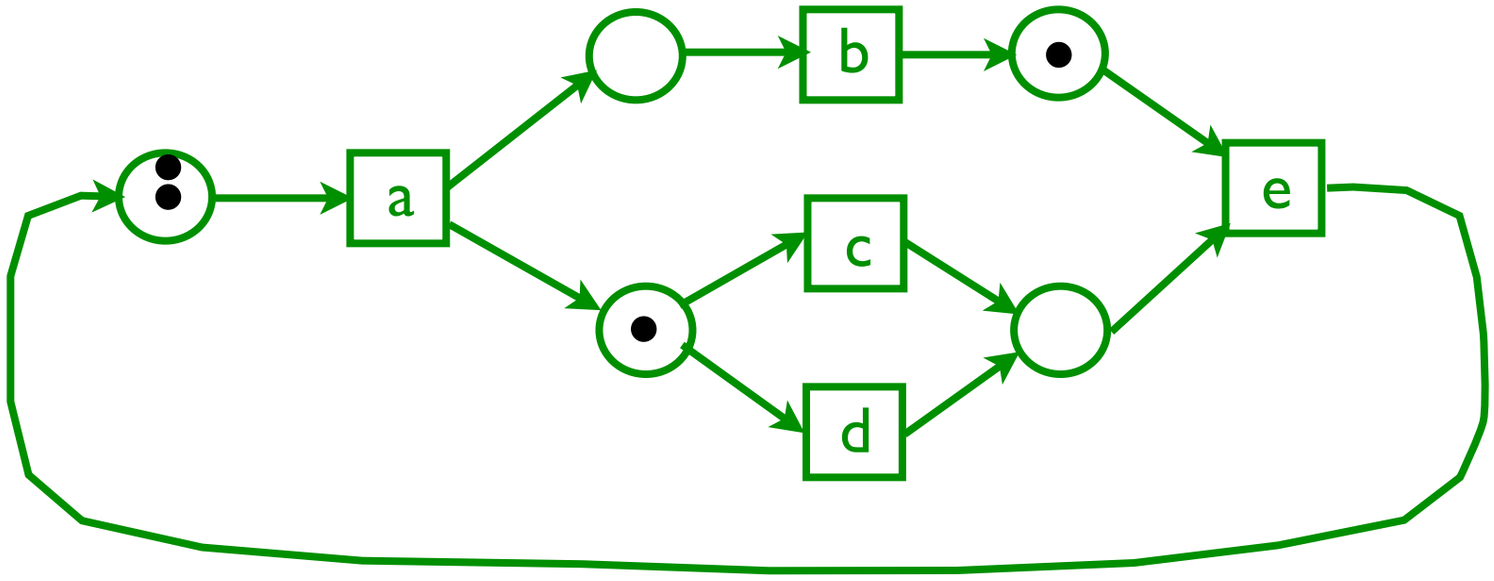



- Einige mögliche Abfolgen (falls sequentiell):

  a

- Daraus komplexe Netze zusammenbaubar

  ⇒ Aussagen über Synchronisationseigenschaften

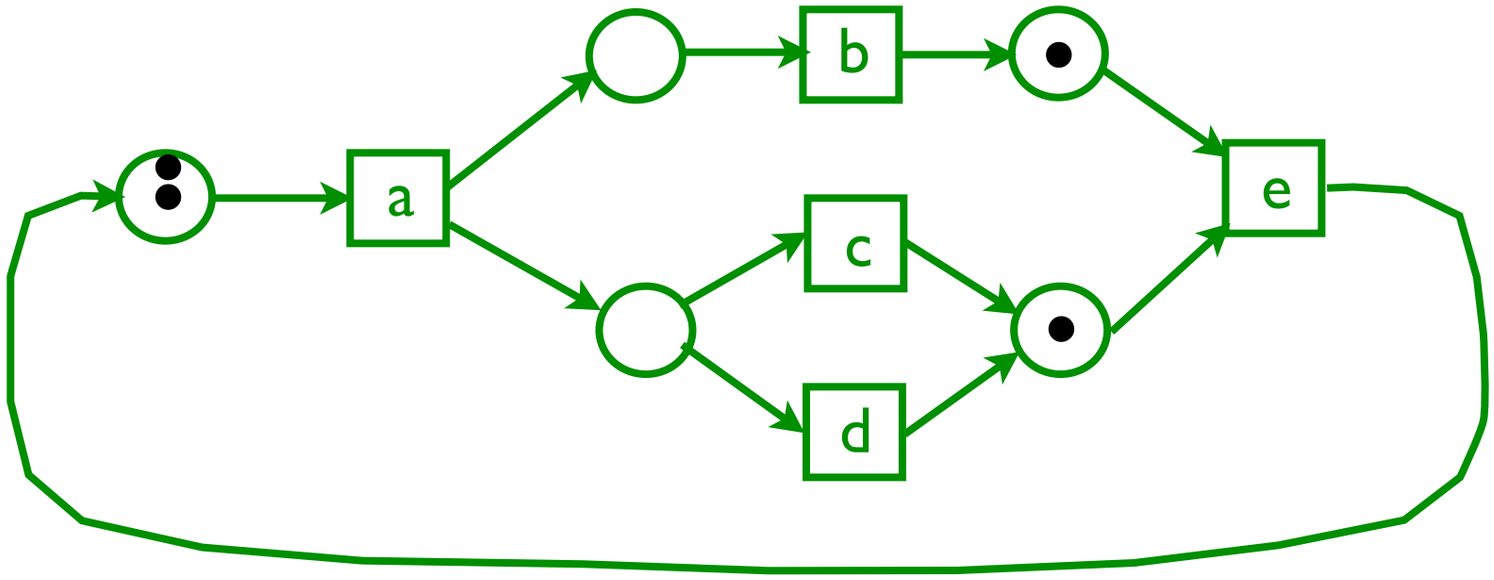



- Einige mögliche Abfolgen (falls sequentiell):

  a → b

- Daraus komplexe Netze zusammenbaubar

  ⇒ Aussagen über Synchronisationseigenschaften

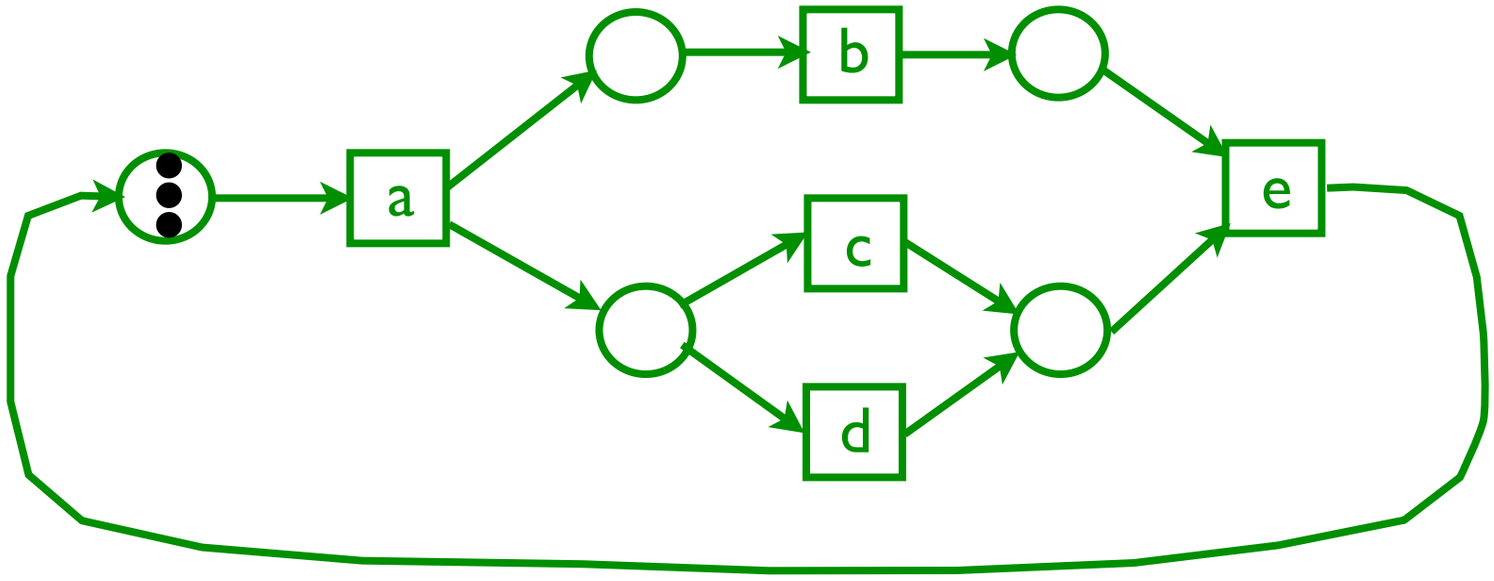



- Einige mögliche Abfolgen (falls sequentiell):

  a → b → c

- Daraus komplexe Netze zusammenbaubar

  ⇒ Aussagen über Synchronisationseigenschaften

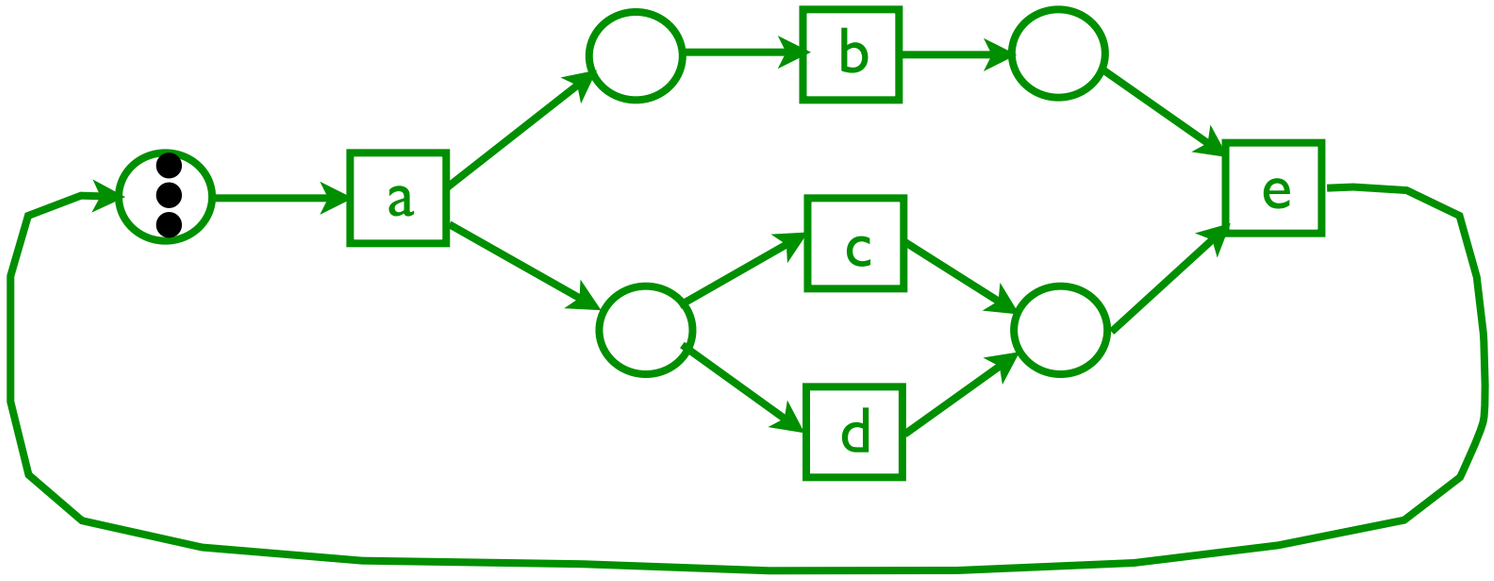



- Einige mögliche Abfolgen (falls sequentiell):

  a → b → c → e ...

- Daraus komplexe Netze zusammenbaubar

  ⇒ Aussagen über Synchronisationseigenschaften

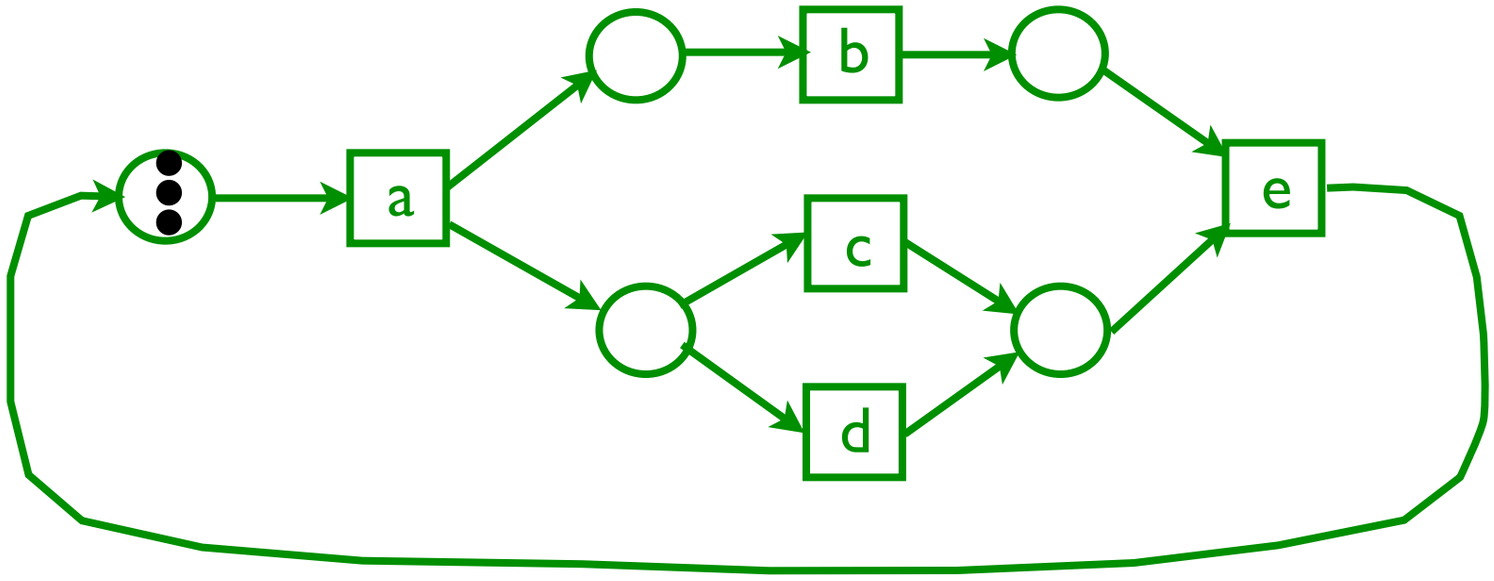



- Einige mögliche Abfolgen (falls sequentiell):

  a → b → c → e ...
  a → d → b → e ...
  a → a → b → a → d → d → e → b → e → c → b → e ...

- Daraus komplexe Netze zusammenbaubar

  ⇒ Aussagen über Synchronisationseigenschaften




- Einige mögliche Abfolgen (falls sequentiell):

  a → b → c → e ...
  a → d → b → e ...
  a → a → b → a → d → d → e → b → e → c → b → e ...

- Aber nicht:

  b → c → a → …
  a → c → d → e …
  a → b → e ...
  a → a → a → a → ...

# Fragen – Teil 1

- Aus welchen Komponenten besteht ein Petri-Netz (mit Marken)? Was kann man damit beschreiben?

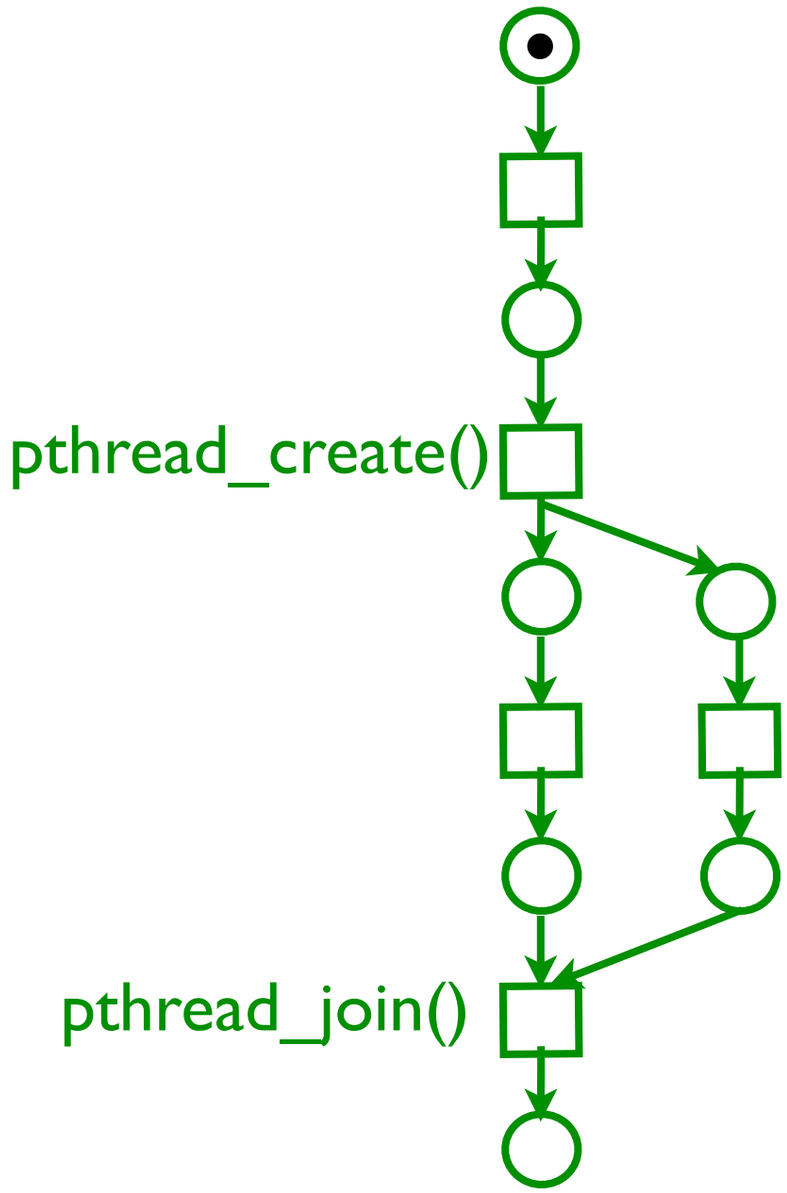
- Was kennzeichnet lebendige bzw. todesgefährdete Petri-Netze?

# Teil 2: Synchronisationsaussagen in Petri-Netzen

- Ablauf eines einzelnen Prozesses

sequentielles Programm

- Ablauf eines einzelnen Prozesses

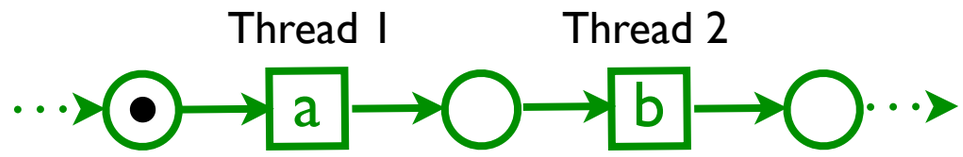



Multithreading

- Petri-Netze allerdings besonders interessant für Synchronisationsbedingungen zwischen Prozessen/Threads (im folgenden darauf fokussiert)

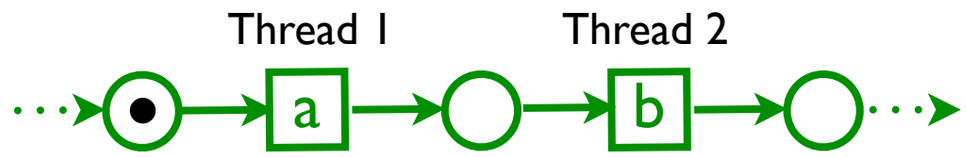
- Mit Marken-behafteten Petri-Netzen kann man Synchronisationseigenschaften ausdrücken: ⇒ „Bausteine“ für Synchronisationbedingungen
  - Erst a, dann b

    ⇒ Sequenz (einseitige Synchronisation)



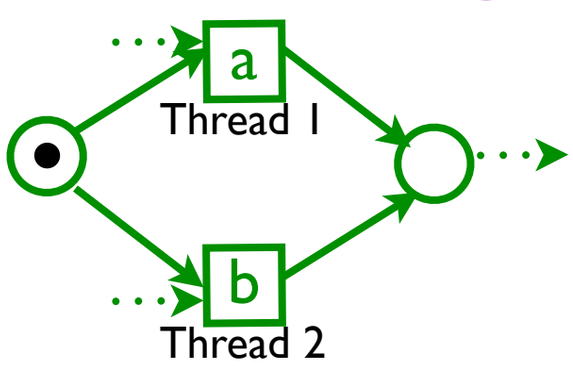
- Mit Marken-behafteten Petri-Netzen kann man Synchronisationseigenschaften ausdrücken: ⇒ „Bausteine“ für Synchronisationbedingungen
  - Erst a, dann b

    ⇒ Sequenz (einseitige Synchronisation)



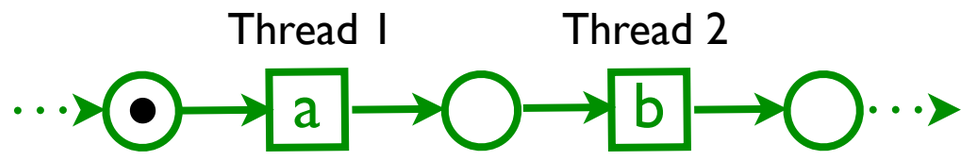

  - (Pro Marke:) Entweder a oder b

    ⇒ Auswahl (bei 1 Marke: mehrseitige Synchronisation)



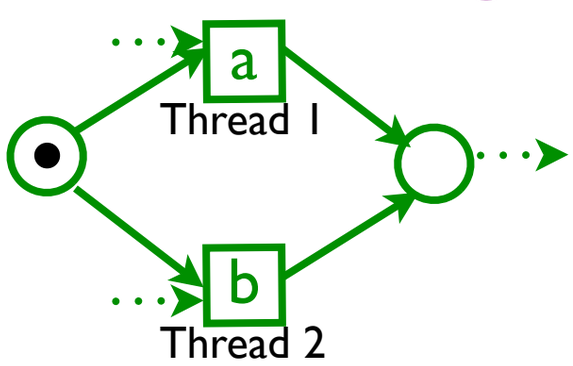
- Mit Marken-behafteten Petri-Netzen kann man Synchronisationseigenschaften ausdrücken: ⇒ „Bausteine“ für Synchronisationbedingungen
  - Erst a, dann b

    ⇒ Sequenz (einseitige Synchronisation)



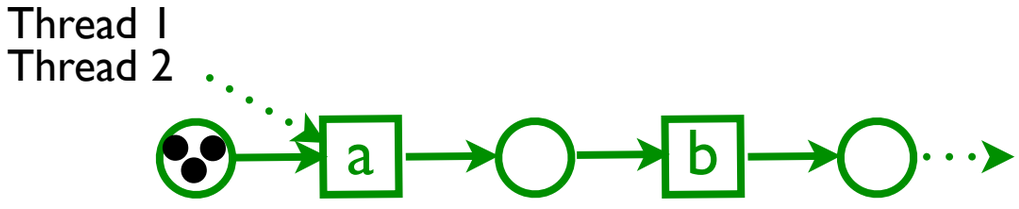

  - (Pro Marke:) Entweder a oder b

    ⇒ Auswahl (bei 1 Marke: mehrseitige Synchronisation)



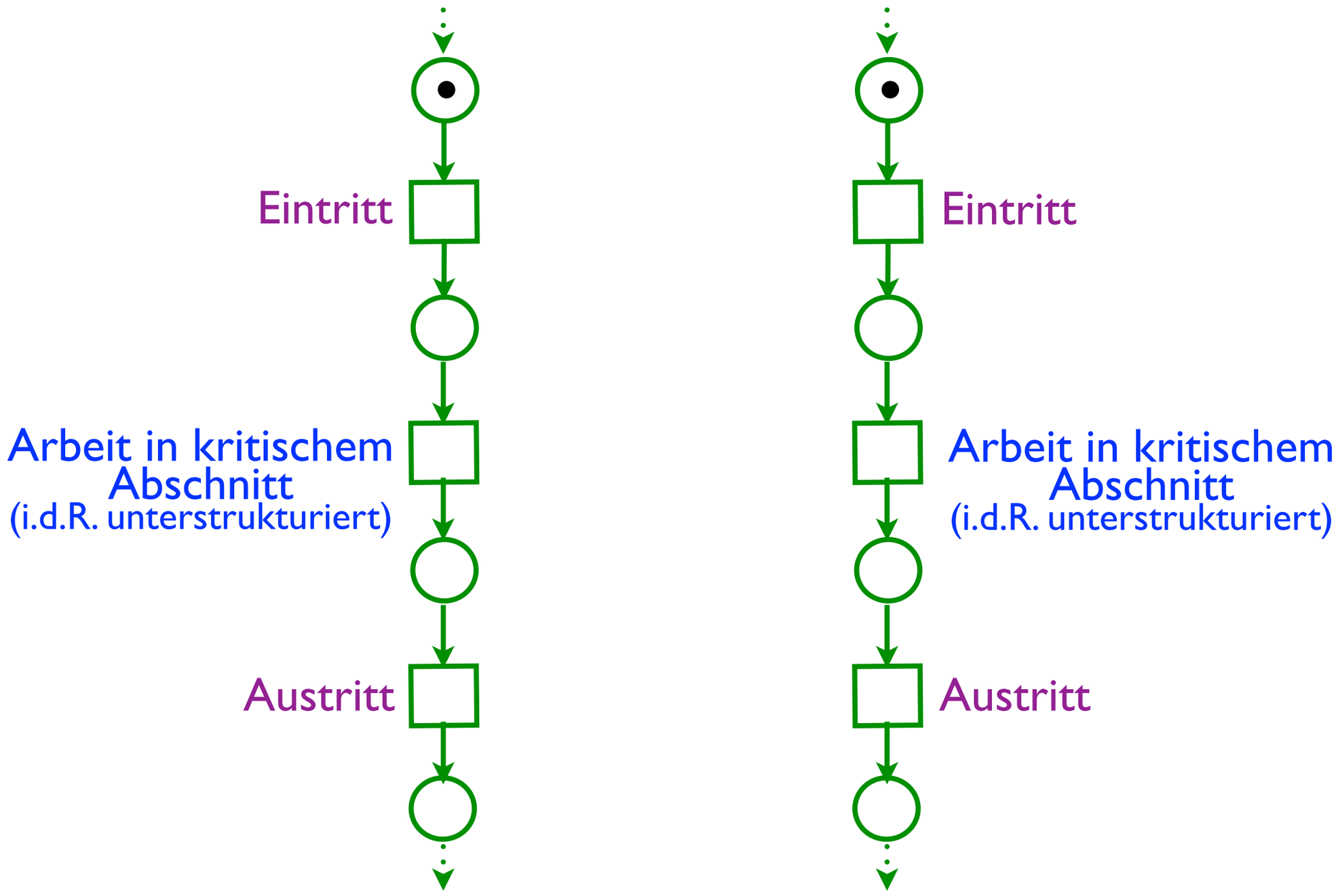

  - Beschränkte Nebenläufigkeit (z.B. 3 Ausführungen nebenläufig)

    ⇒ Ressourcenverwaltung



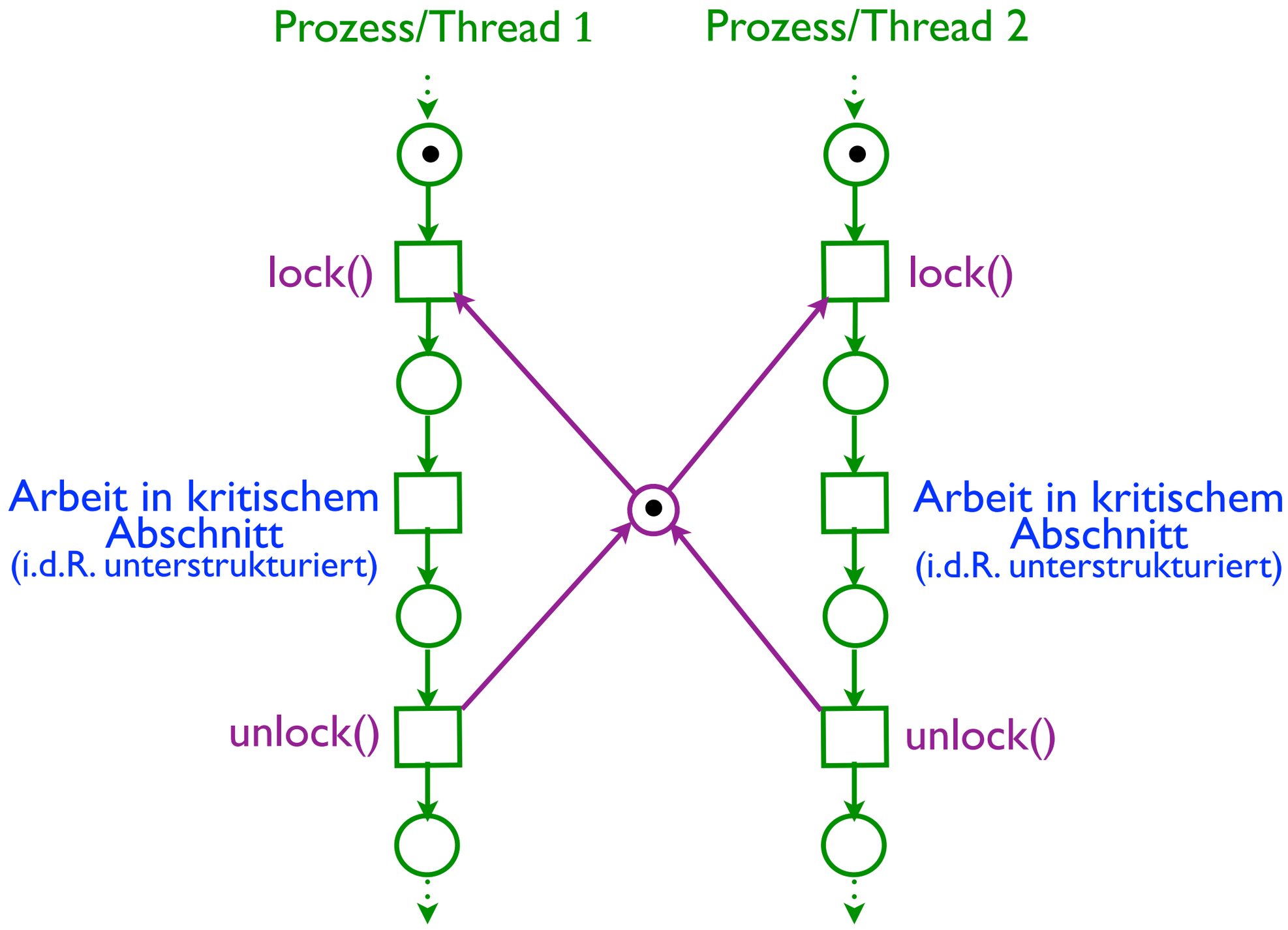
# Beispiel: Kritischer Abschnitt

noch ungeschützt



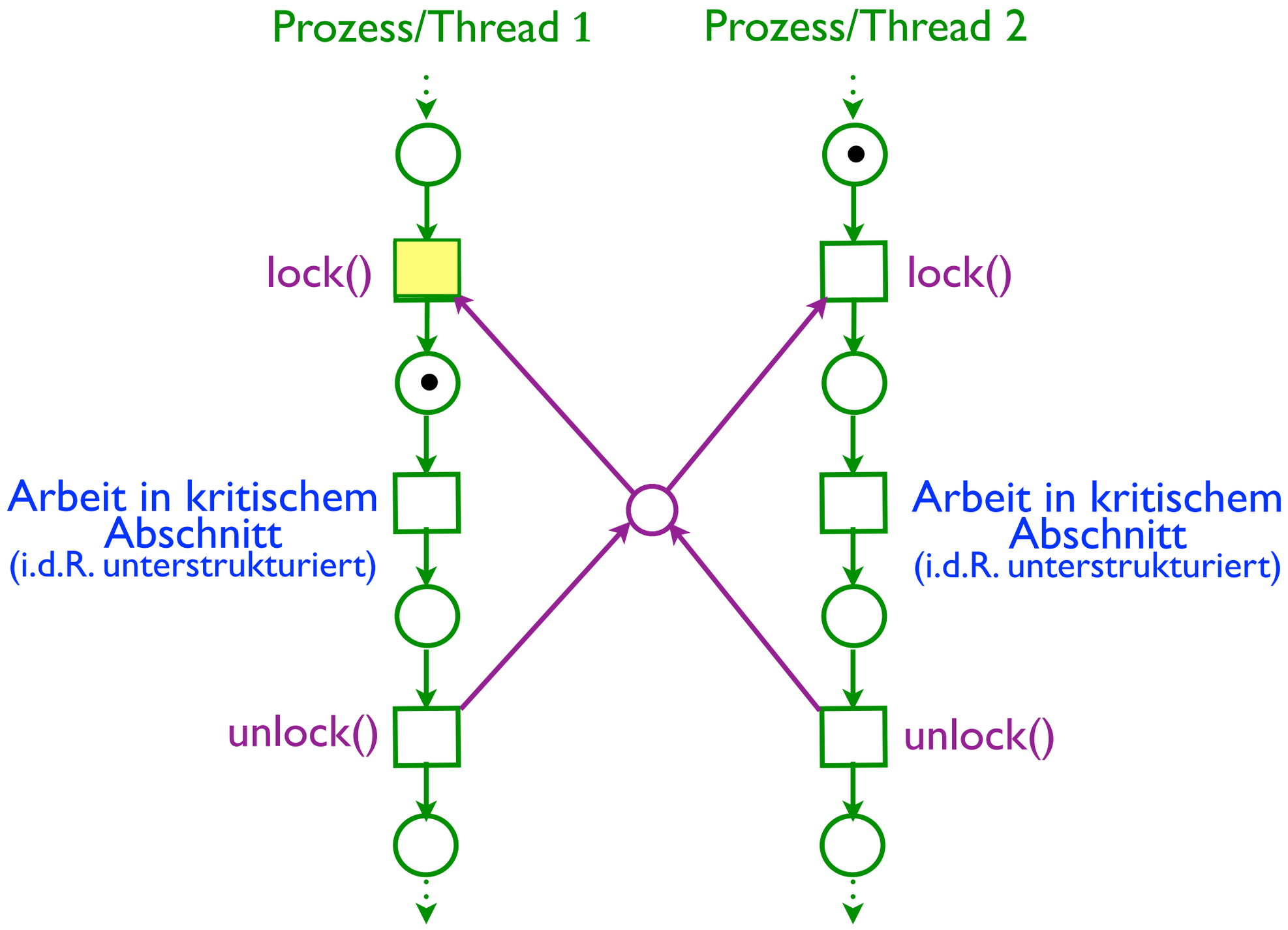

- Kritischer Abschnitt selbst noch beliebig verfeinerbar

# Beispiel: Kritischer Abschnitt



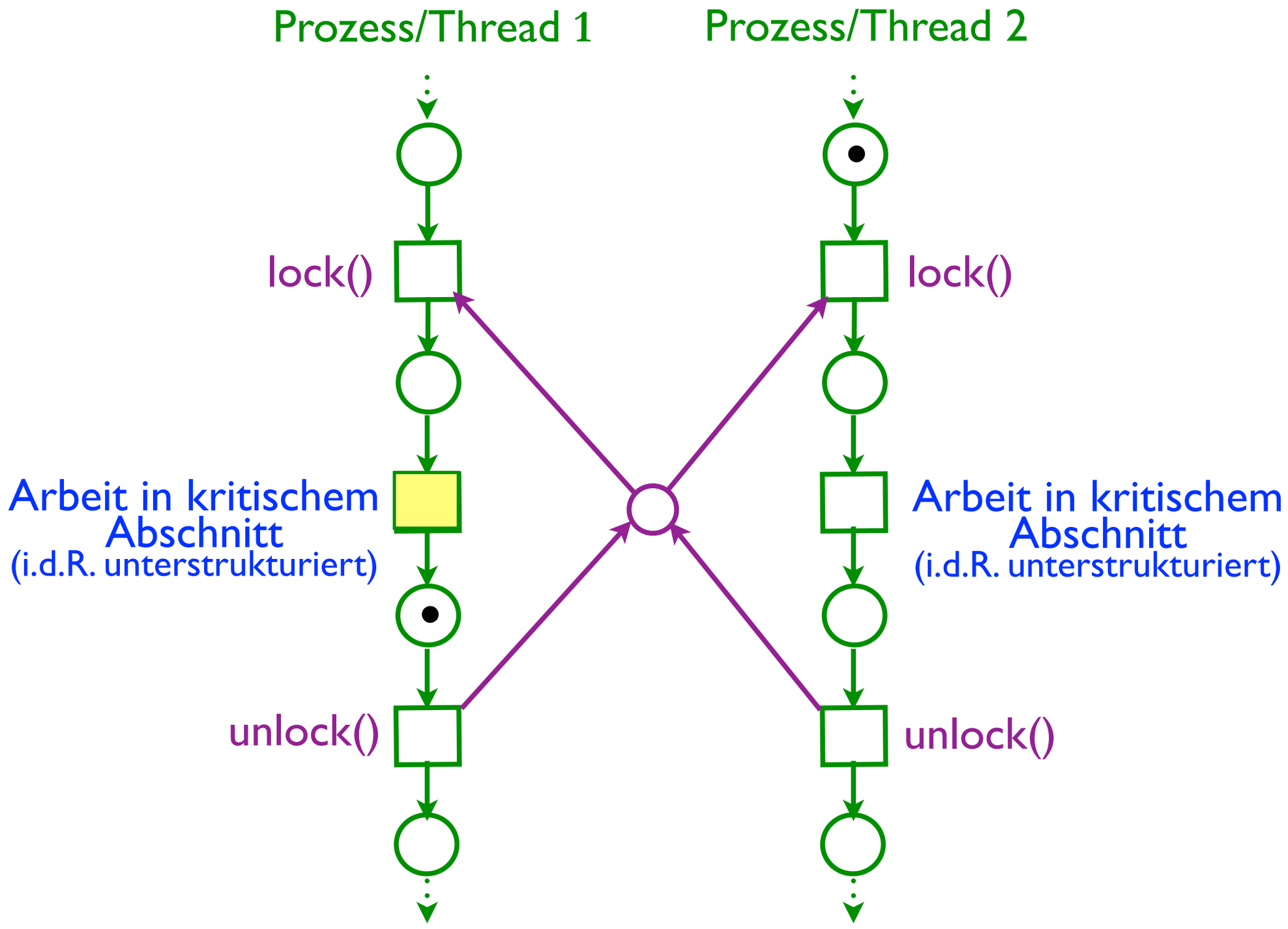

- Kritischer Abschnitt selbst noch beliebig verfeinerbar

# Beispiel: Kritischer Abschnitt



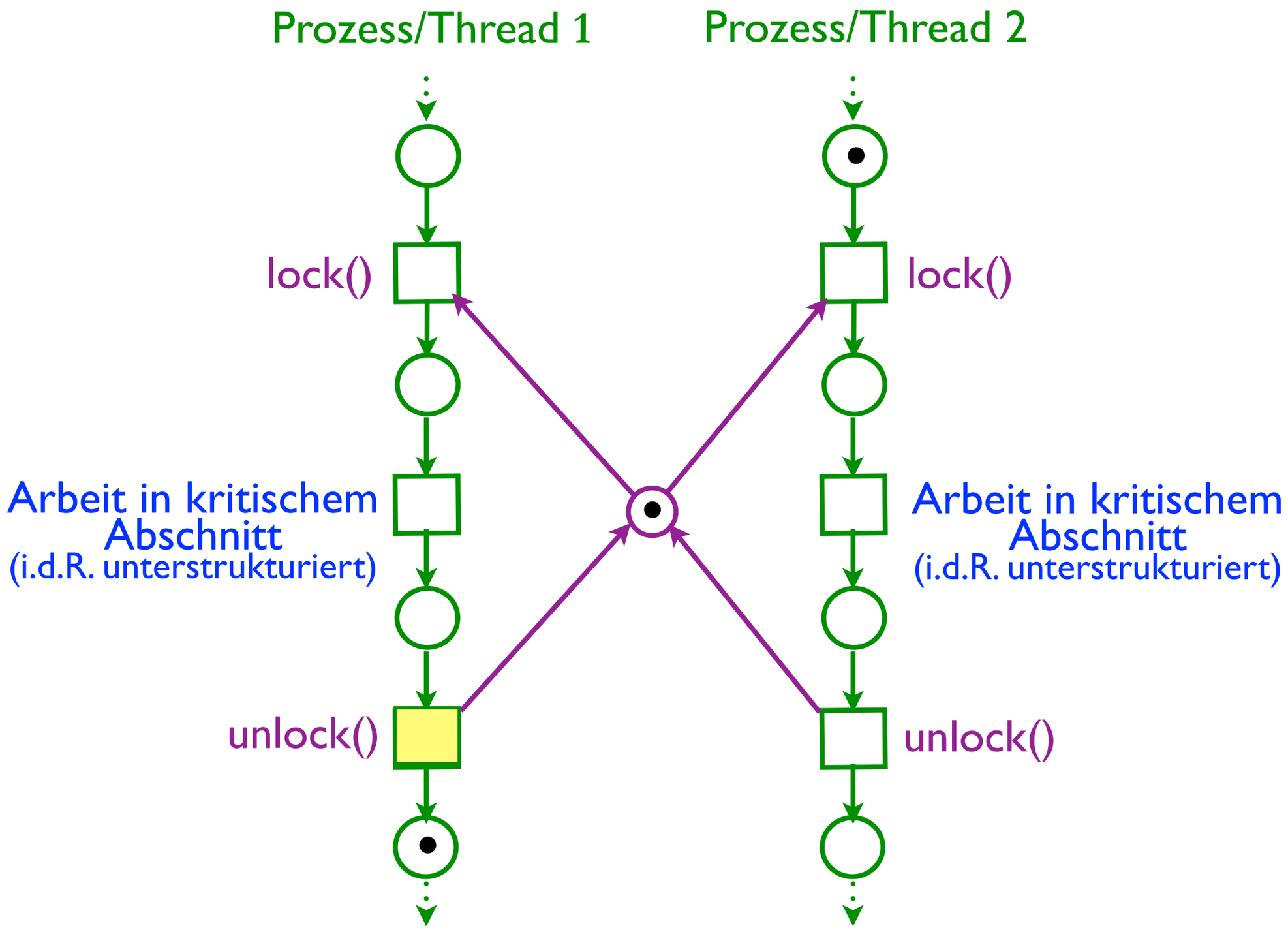

- Kritischer Abschnitt selbst noch beliebig verfeinerbar

# Beispiel: Kritischer Abschnitt



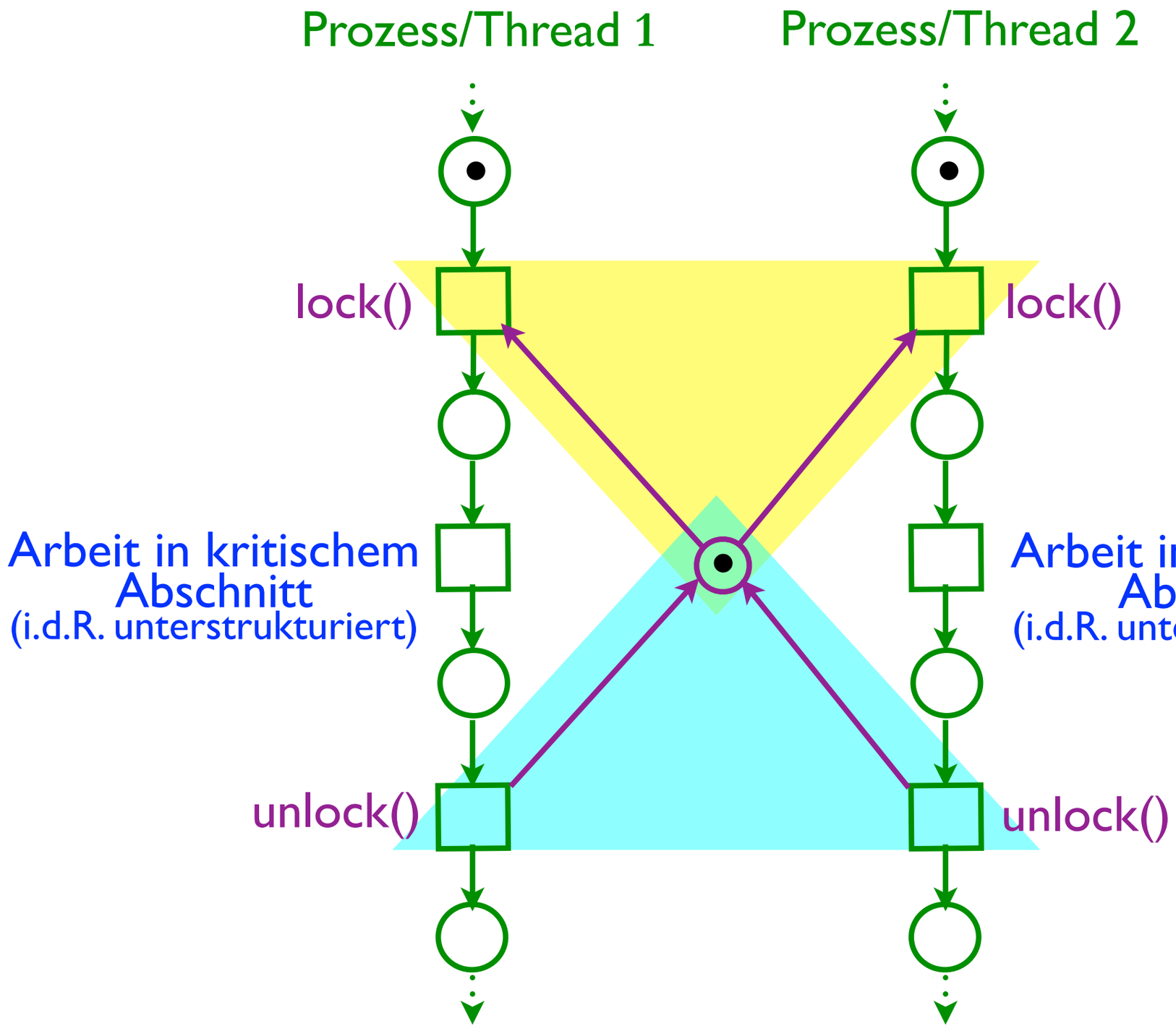

- Kritischer Abschnitt selbst noch beliebig verfeinerbar

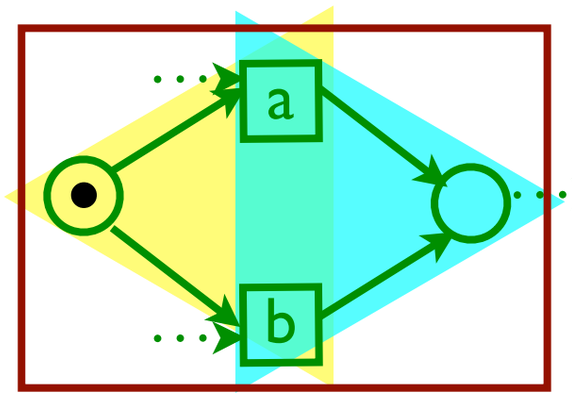
# Beispiel: Kritischer Abschnitt




- Kritischer Abschnitt selbst noch beliebig verfeinerbar

# Beispiel: Kritischer Abschnitt

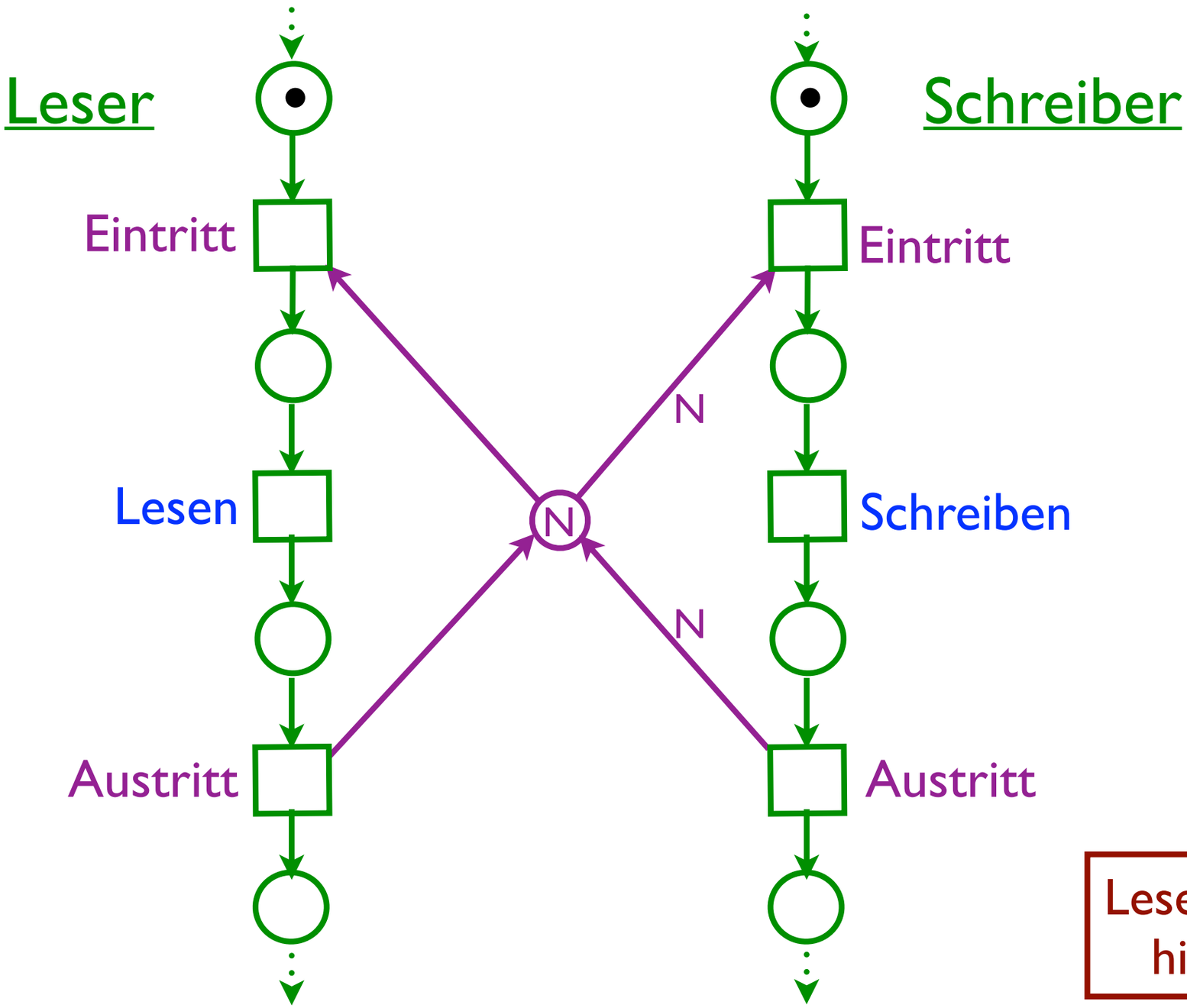



- Kritischer Abschnitt selbst noch beliebig verfeinerbar

# Beispiel: Leser/Schreiber-Problem

- Beliebig viele Leser oder ein Schreiber
  ⇒ durch feste Anzahl von Marken nicht beschreibbar
- Im folgenden „N“ Marken vorsehen
  (max. N Prozesse erzeugbar ➔ „Systemparameter“)



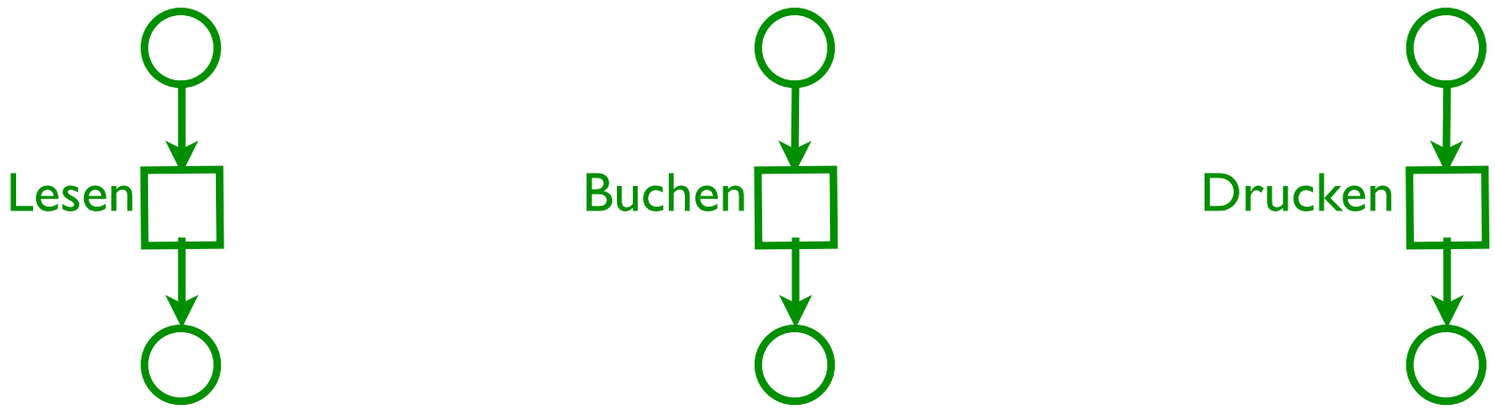

Leser werden auch hier bevorzugt

# Beispiel: Erzeuger-/Verbraucher-Problem

- Einfaches Buchungssystem

noch unsynchronisiert

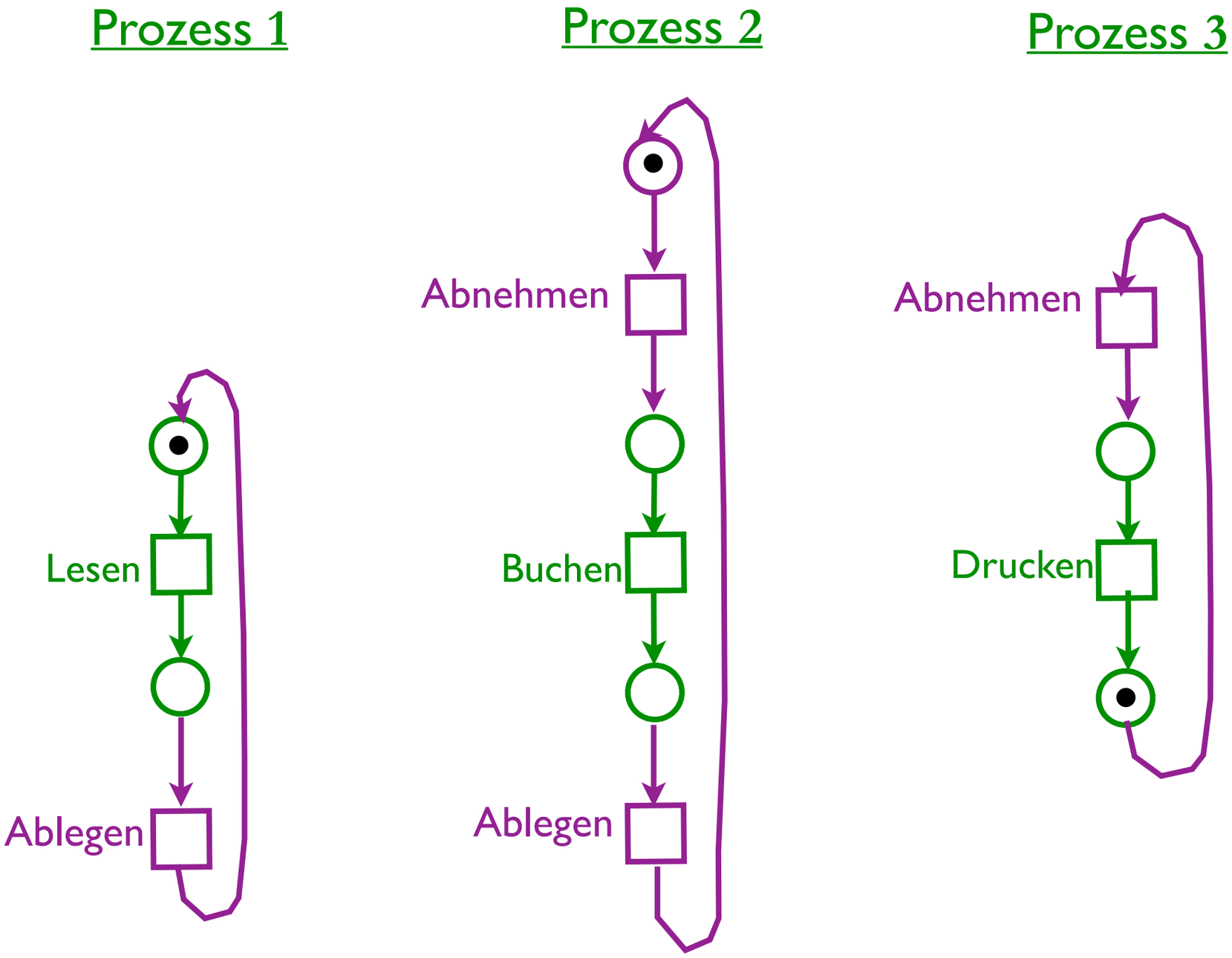
Prozess 1 Prozess 2 Prozess 3



# Beispiel: Erzeuger-/Verbraucher-Problem

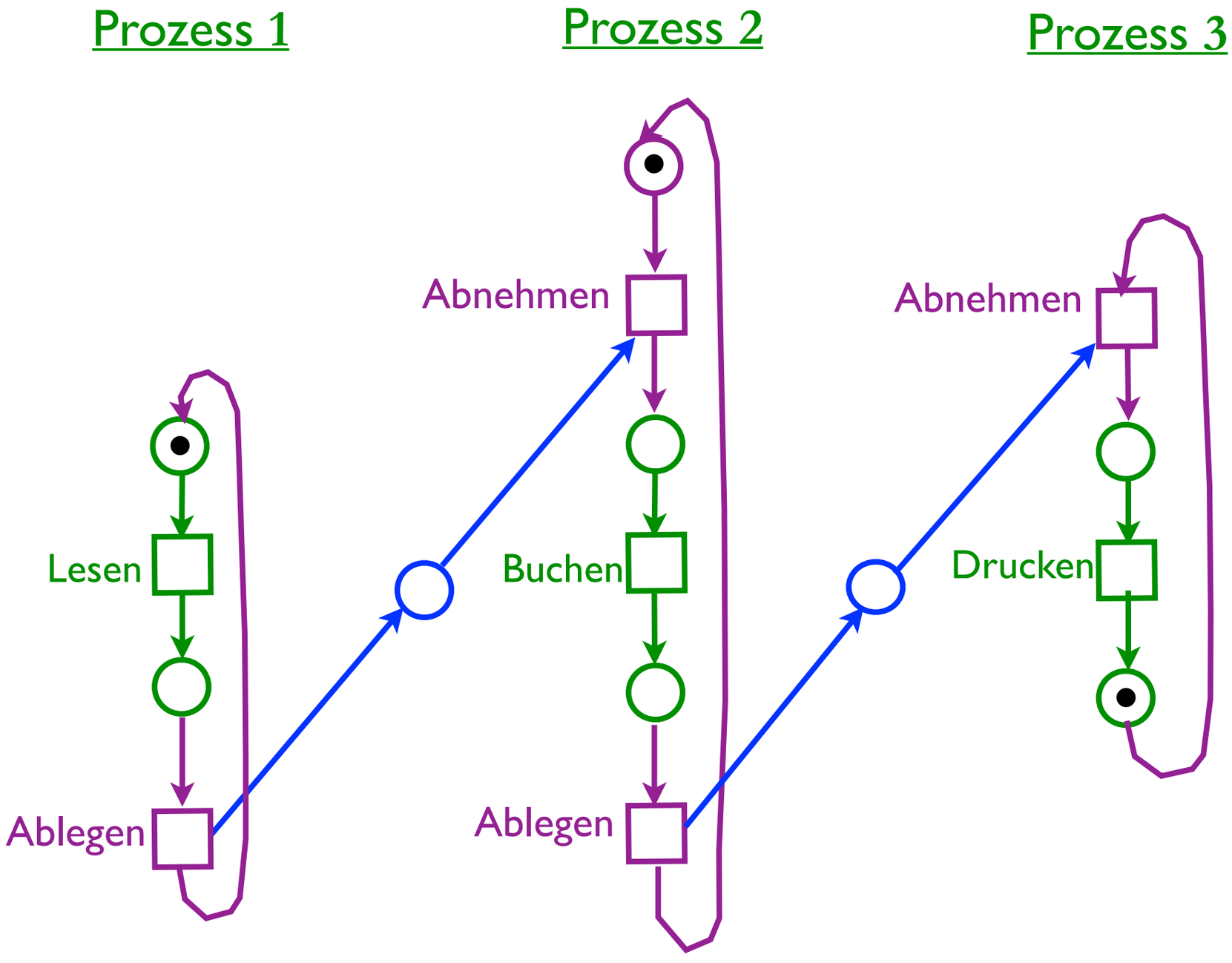
- Einfaches Buchungssystem

noch unsynchronisiert



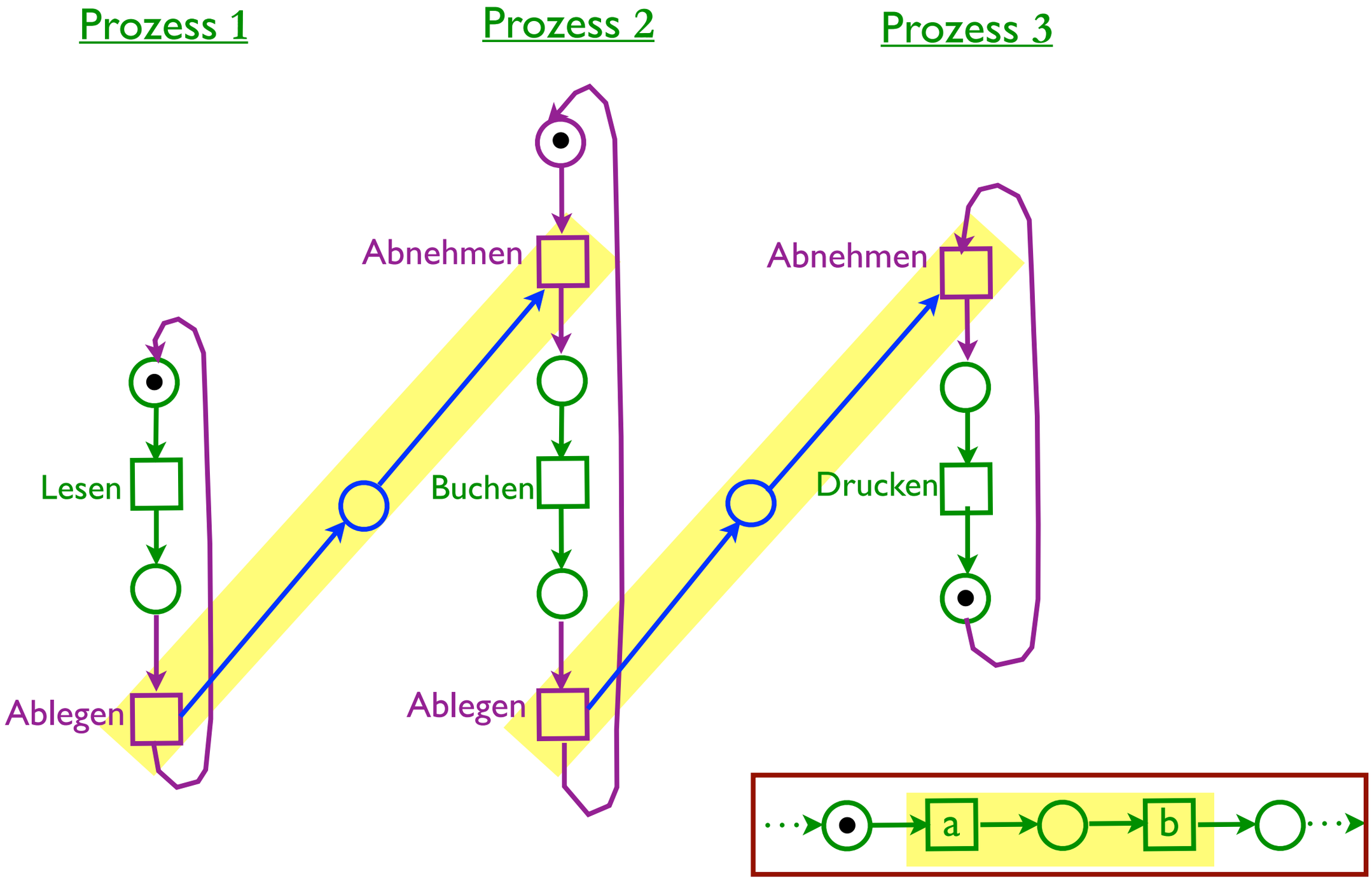
# Beispiel: Erzeuger-/Verbraucher-Problem

- Einfaches Buchungssystem



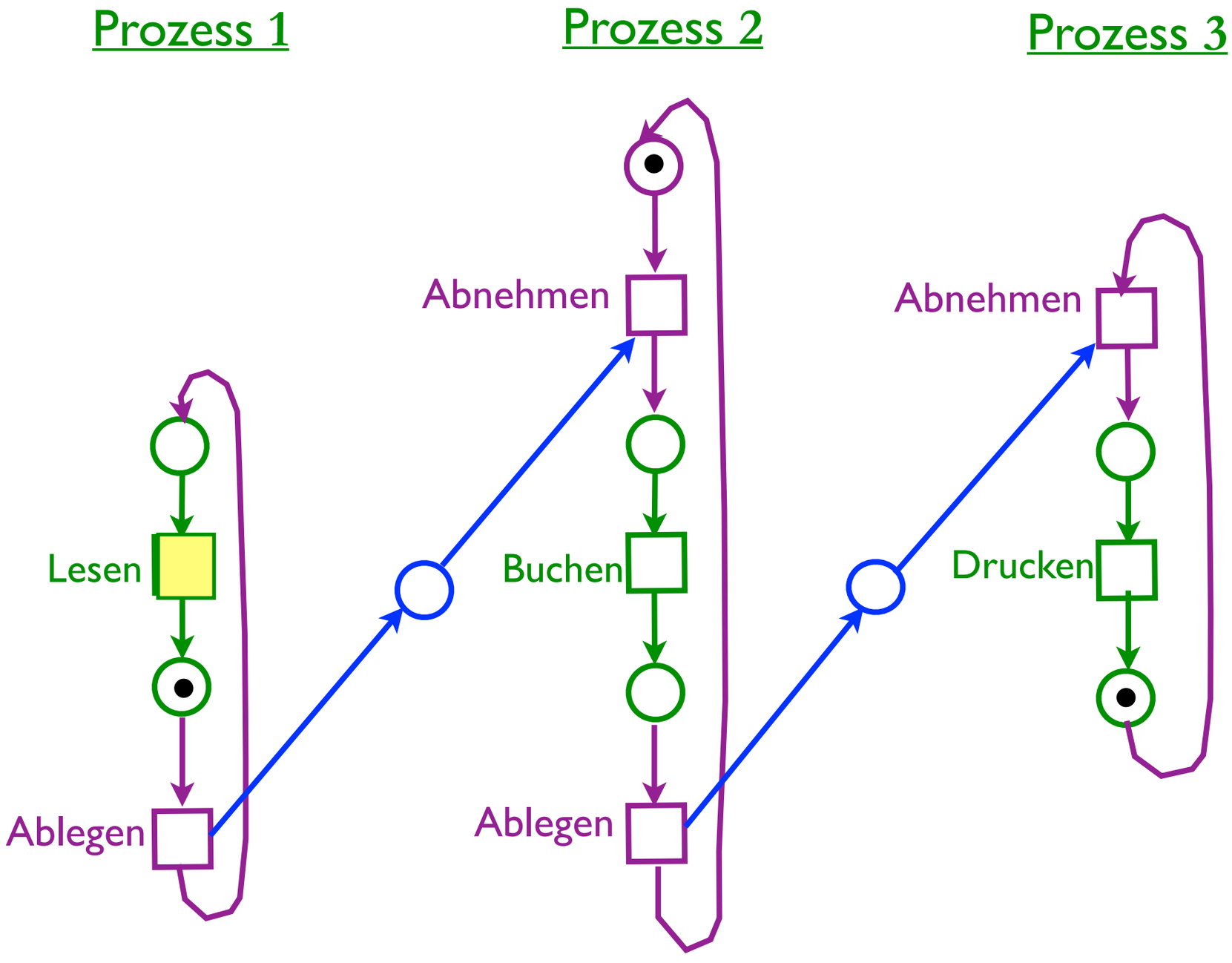
# Beispiel: Erzeuger-/Verbraucher-Problem

- Einfaches Buchungssystem



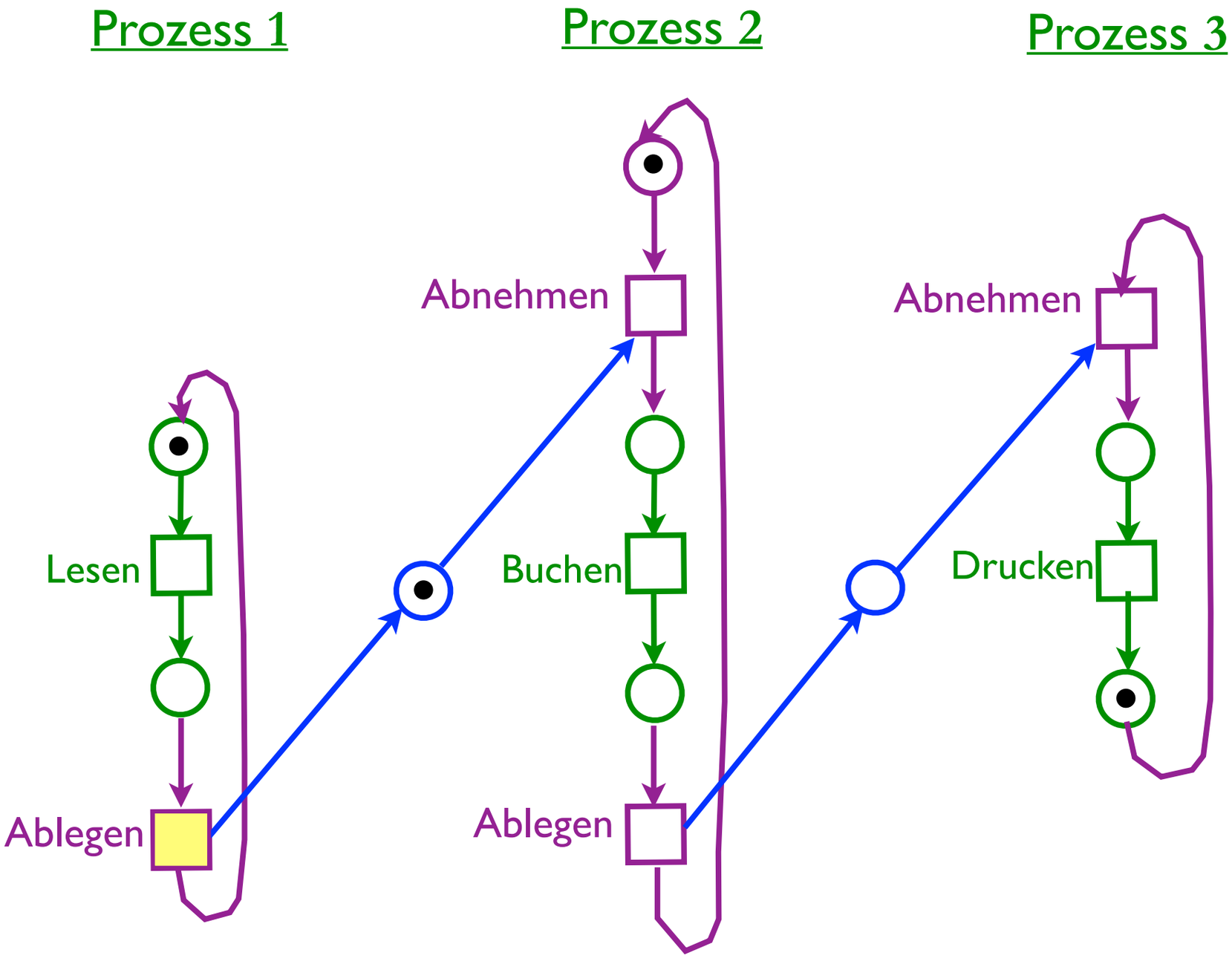
# Beispiel: Erzeuger-/Verbraucher-Problem

- Einfaches Buchungssystem



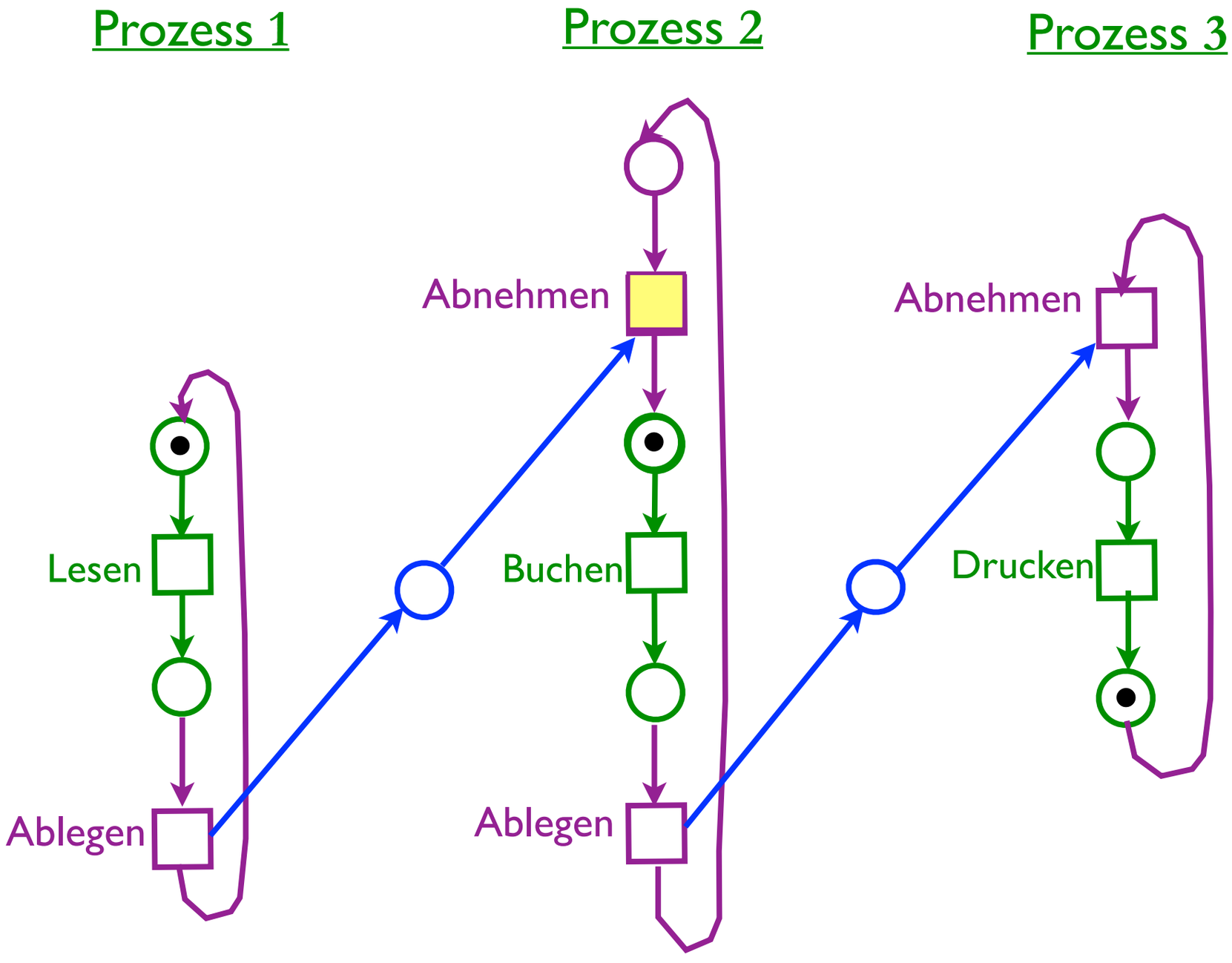
# Beispiel: Erzeuger-/Verbraucher-Problem

- Einfaches Buchungssystem



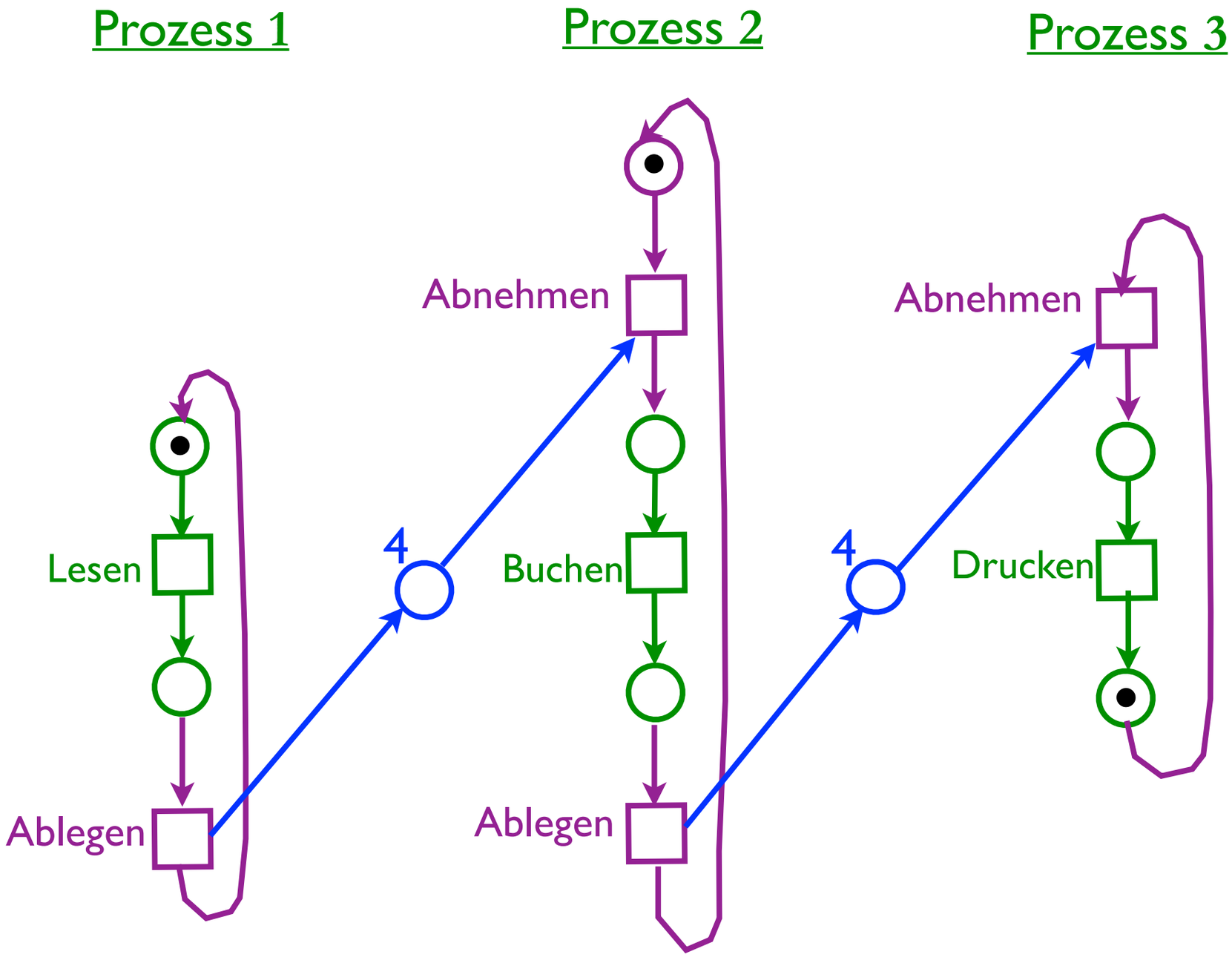
# Beispiel: Erzeuger-/Verbraucher-Problem

- Einfaches Buchungssystem



# Beispiel: Erzeuger-/Verbraucher-Problem

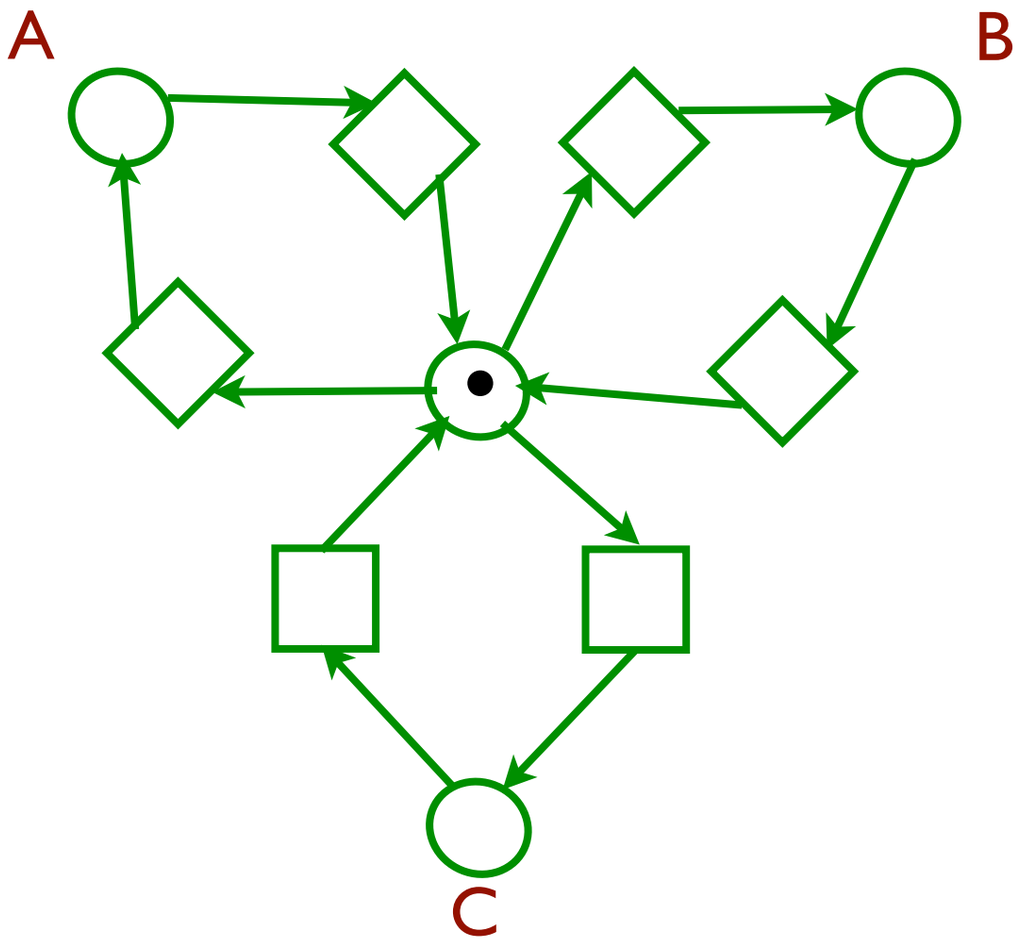
- Einfaches Buchungssystem




- Modellierung eines beschränkten Puffers, z.B. 4 Plätze

# Kleine Aufgabe

Was bewirkt das folgende Petri-Netz?



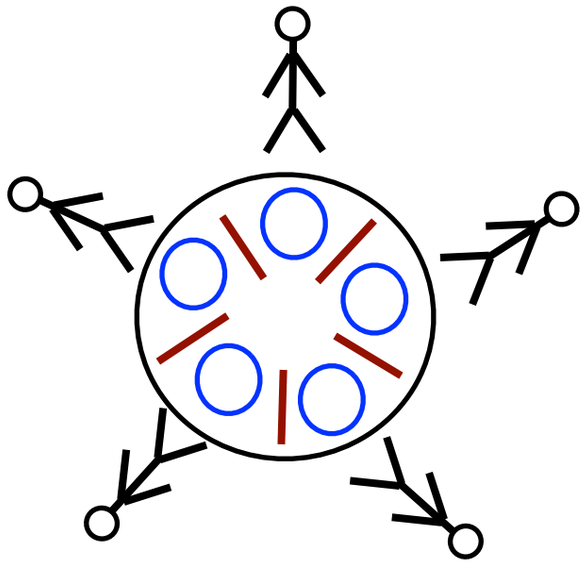
# Beispiel: Speisende Philosophen

Zur Erinnerung:  Mögliche Semaphor-Lösung

(

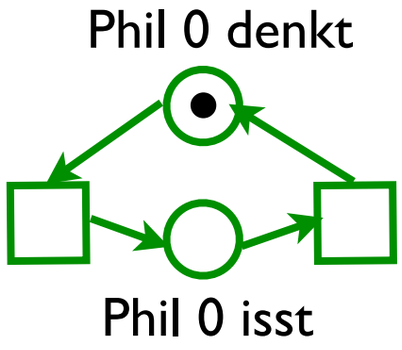
```
struct Sema {
        ...
        Sema (int count=1)
           //Default-Initialisierung
     }
```

```
Sema stick[5];   //Array von 5 Semaphoren, mit 1 initialisiert
Sema chair(4);   //mit 4 initialisiertes Semaphor

philosoph(int i) {
  while (1) {
     ... denken ...
     chair.P();
     stick[i].P();
     stick[(i+1)%5].P();
     ... speisen ...
     stick[i].V();
     stick[(i+1)%5].V();
     chair.V();
  };
}
```

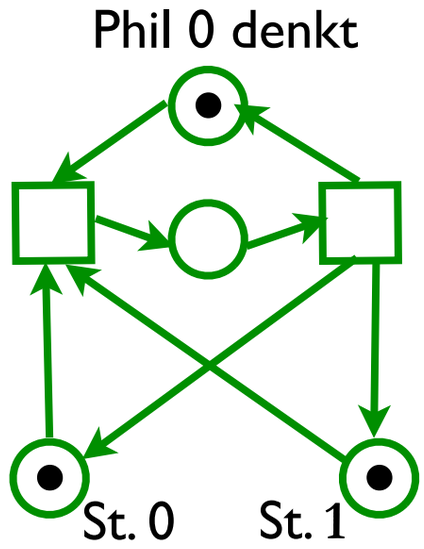


Lösung erinnert an Bankiers-Algorithmus zur Vermeidung von Deadlocks

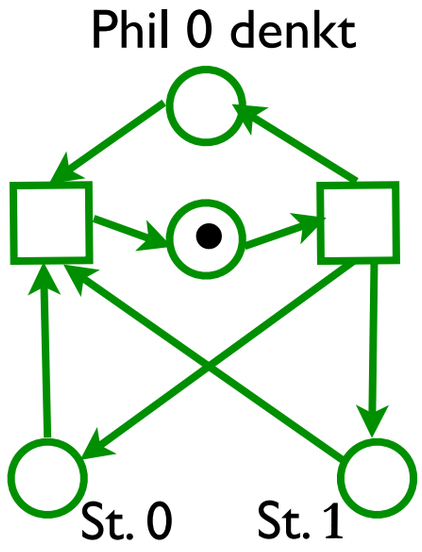
# Beispiel: Speisende Philosophen (Lösung 1)



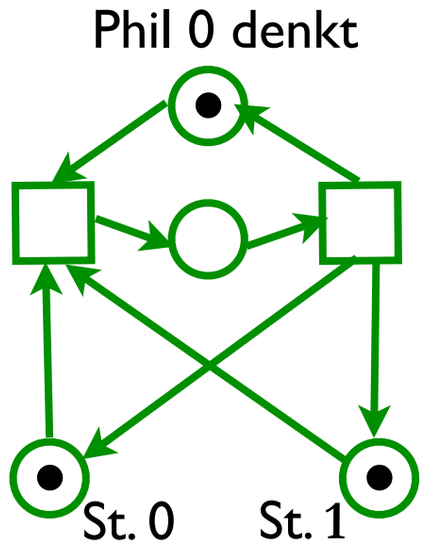
# Beispiel: Speisende Philosophen (Lösung 1)



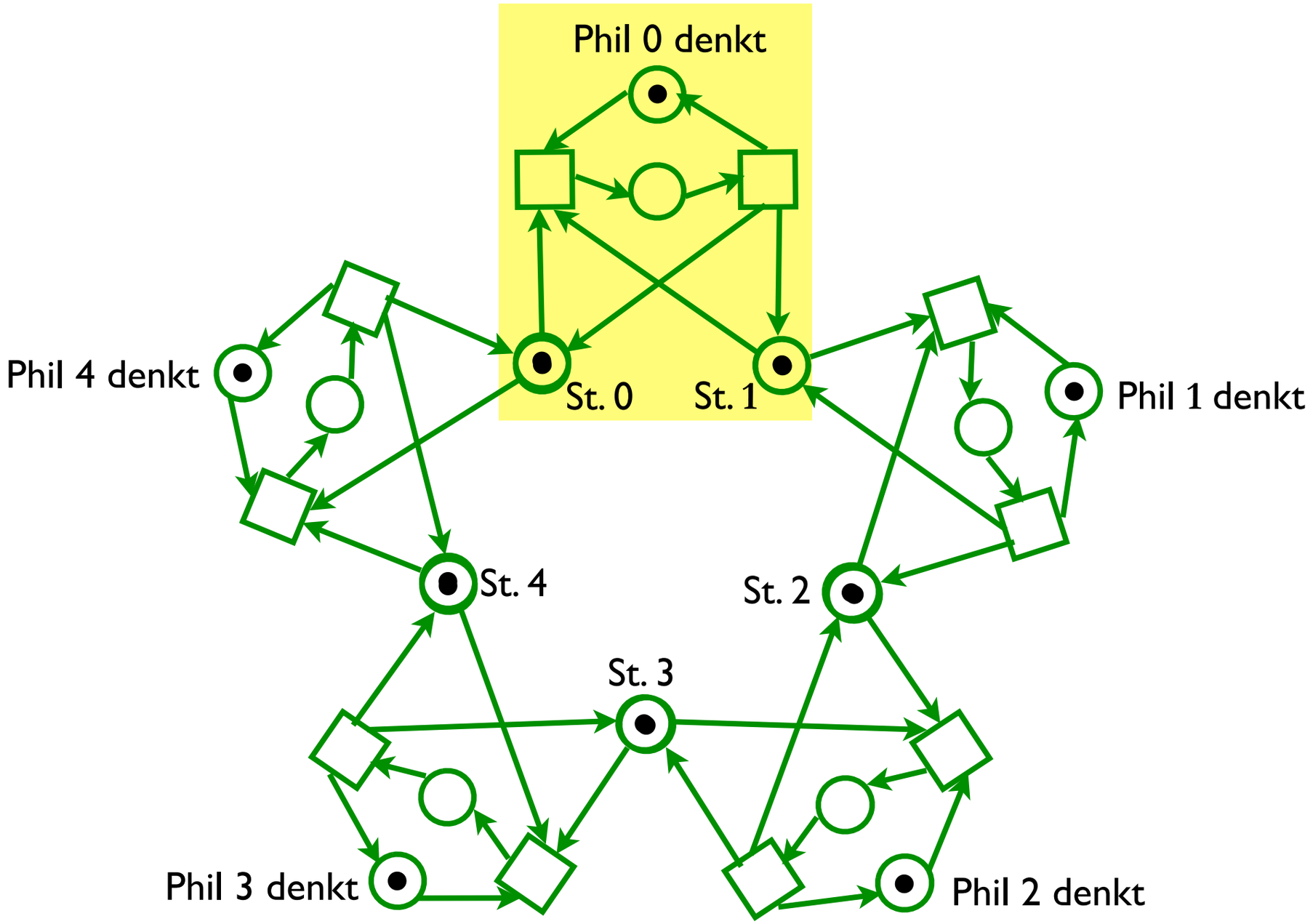
# Beispiel: Speisende Philosophen (Lösung 1)



# Beispiel: Speisende Philosophen (Lösung 1)

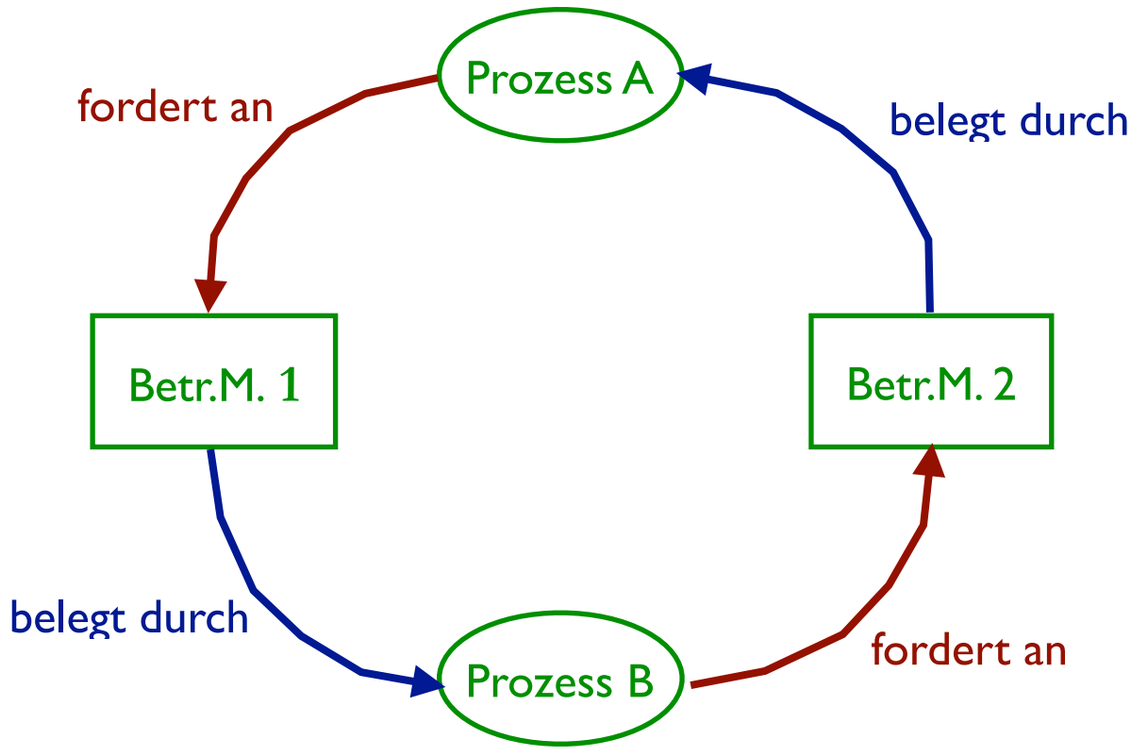


# Beispiel: Speisende Philosophen (Lösung 1)



# Behandlung von Verklemmungen

- Situation: Zwei (oder mehr) Prozesse warten auf „Betriebsmittel“, die nur der/die andere(n) Wartende(n) freigeben kann/können

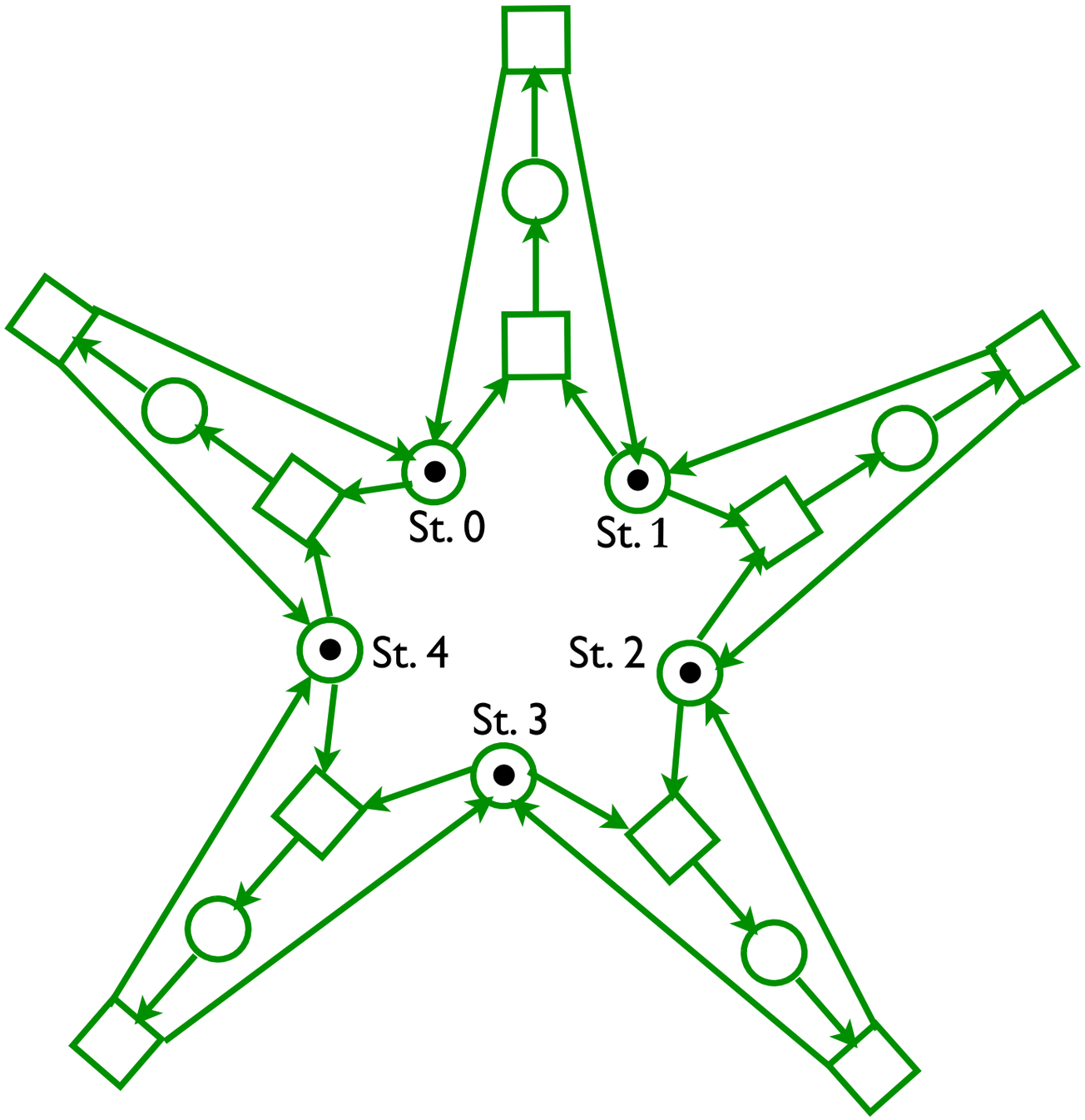



## c) Verhindern

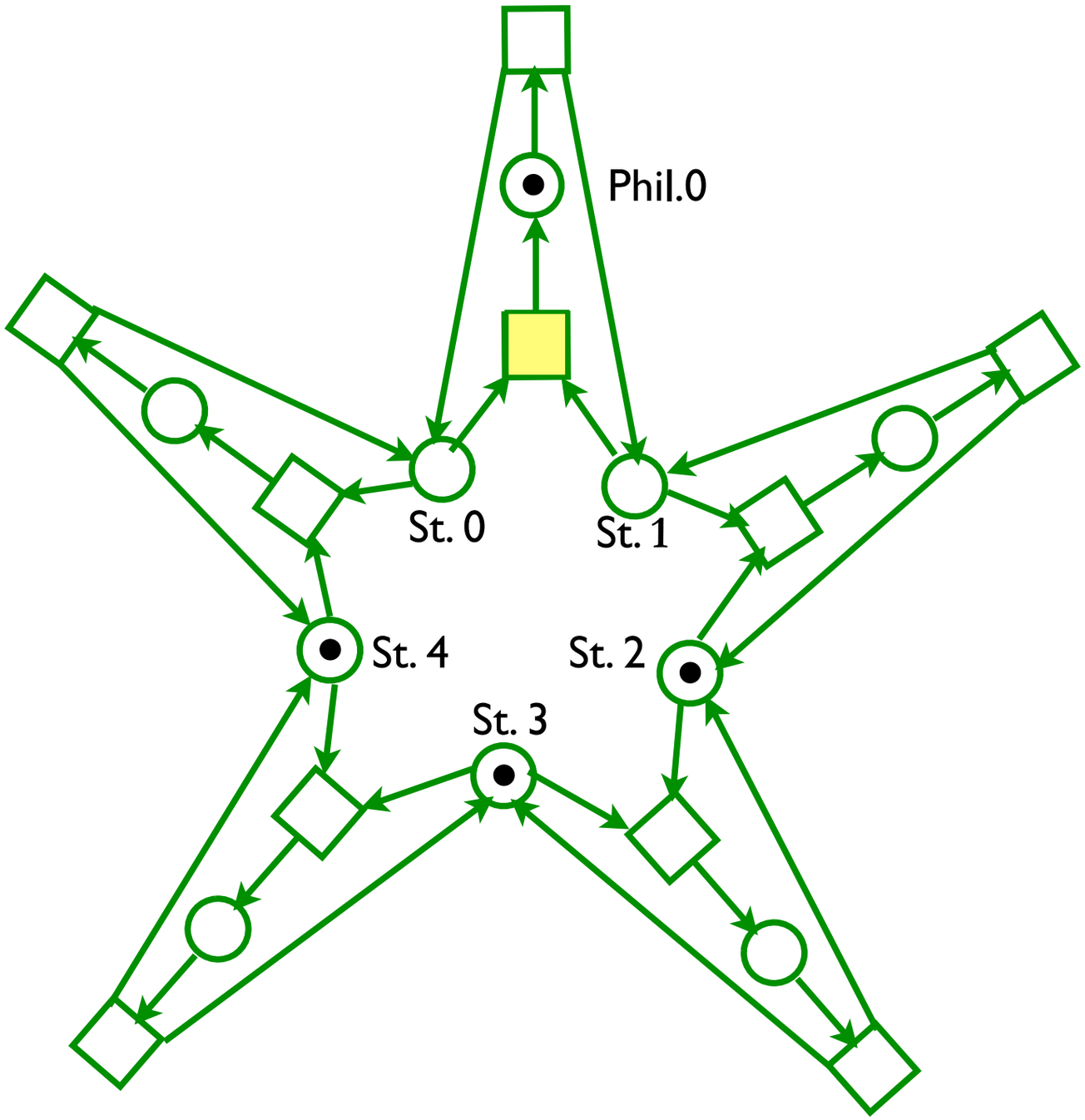
- Eine der Randbedingungen der Entstehung verhindern. Welche?

1. Exklusiven Zugriff mehrerer Prozesse vermeiden? ⇒ Spezialfall Spooling
2. Alles auf einmal anfordern? ⇒ Unteilbarkeit modellieren, oft Verschwendung
3. Zwangsentzug? ⇒ oft Chaos
4. Zyklus verhindern? (z.B. Betriebsmittel nach aufsteigender „Nummer“ anfordern)
   ⇒ oft wenig praktikabel bzw. ziemlich einschränkend

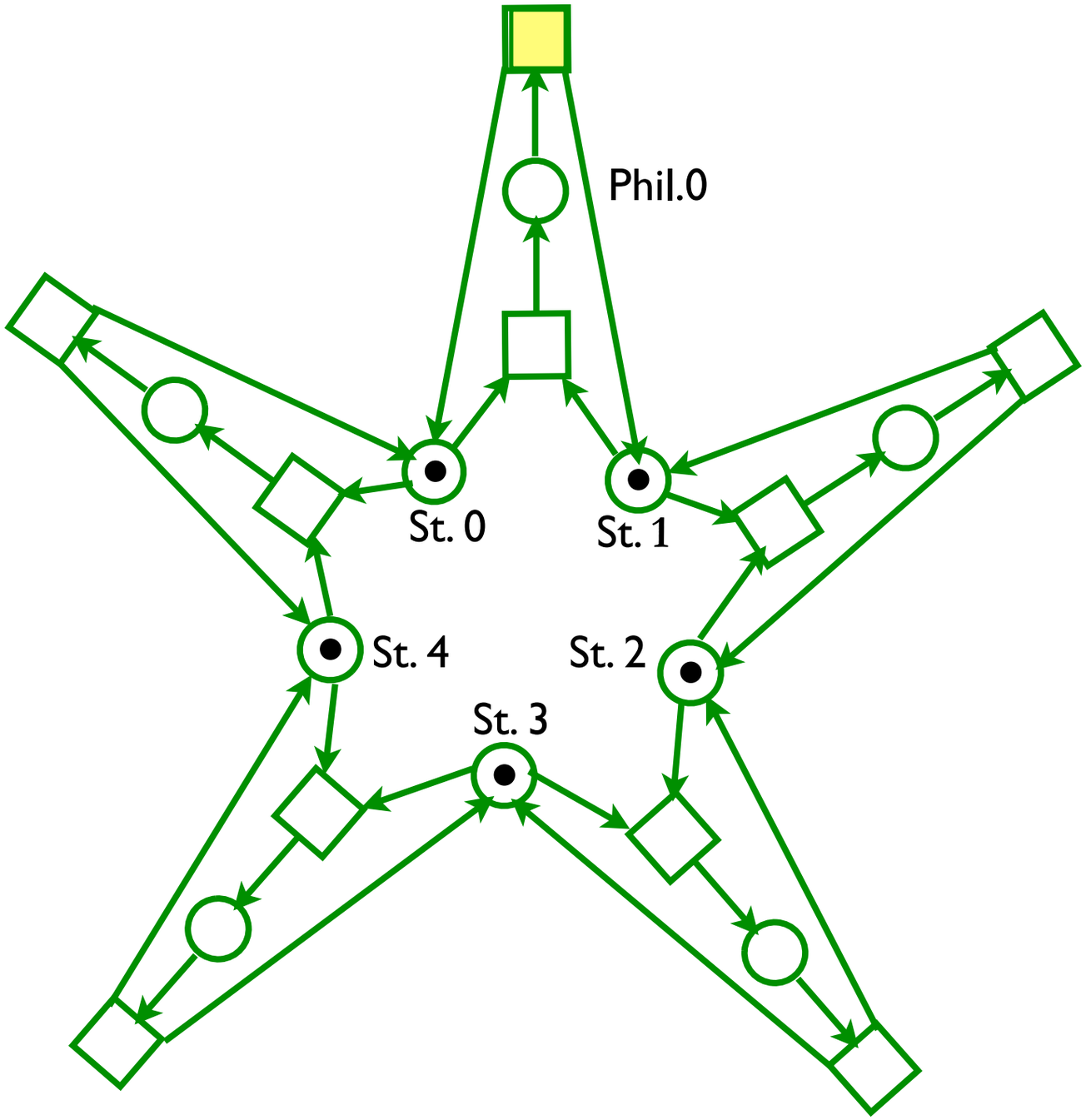
# Beispiel: Speisende Philosophen (Lösung 2)



# Beispiel: Speisende Philosophen (Lösung 2)



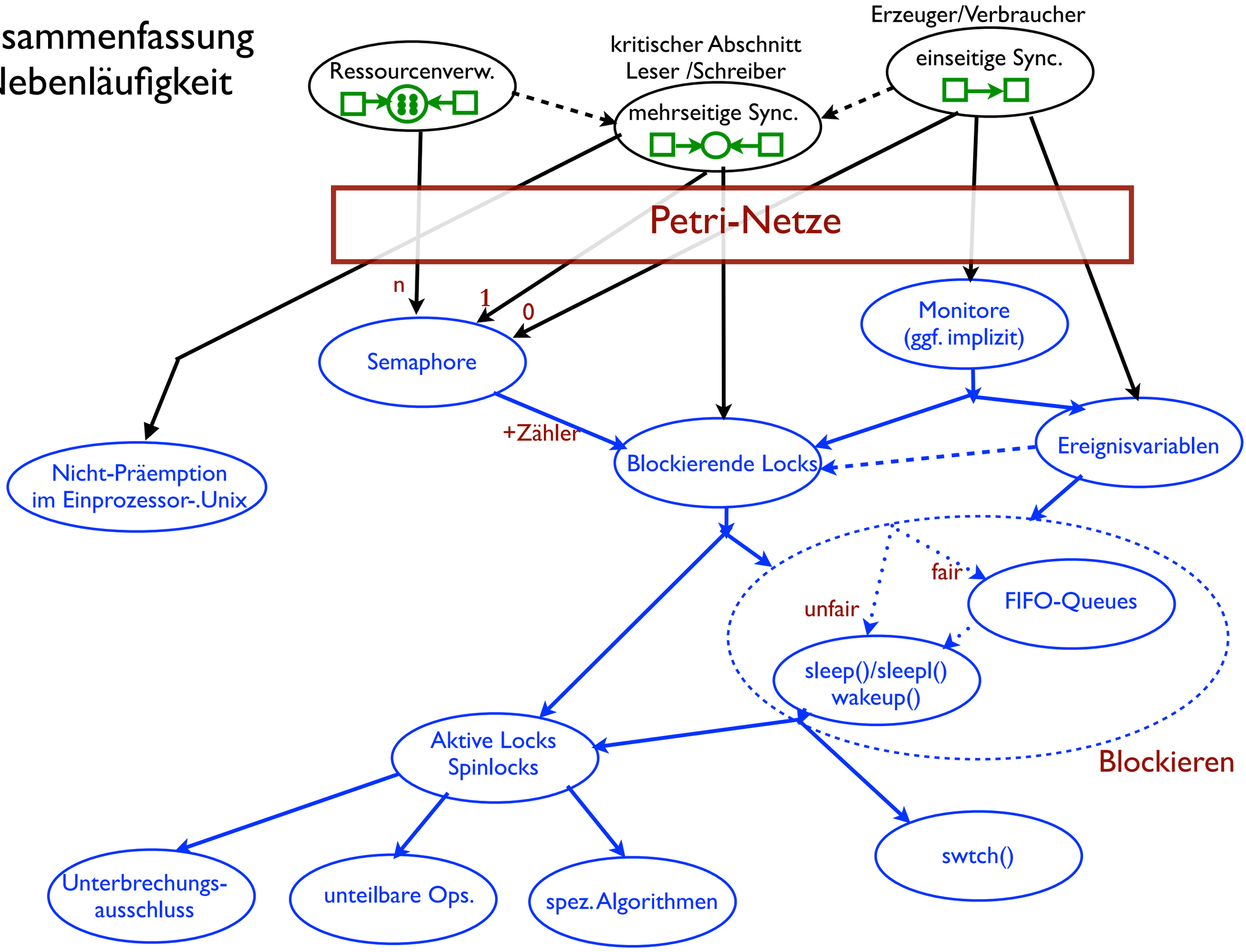
# Beispiel: Speisende Philosophen (Lösung 2)




- Auch bei dieser Lösung: Gleichzeitiges Hochnehmen der Stäbchen modelliert, dadurch Verklemmung verhindert

  ⇒ Modellierung des Wunschverhaltens

# Zusammenfassung Nebenläufigkeit



# Umsetzen der Synchronisationsaussagen in Petri-Netzen

- Durch Semaphore
- Durch Locks/Ereignisvariablen + ggf. Zähler

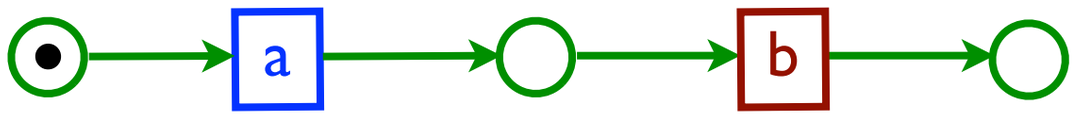
  ⇒ entsprechende Eintrittsprotokolle (Prologe) bzw. Austrittsprotokolle (Epiloge) der verwendeten Operationen vorsehen

# Umsetzen der Synchronisationsaussagen in Petri-Netzen

- Durch Semaphore
- Durch Locks/Ereignisvariablen + ggf. Zähler

  ⇒ entsprechende Eintrittsprotokolle (Prologe) bzw. Austrittsprotokolle (Epiloge) der verwendeten Operationen vorsehen

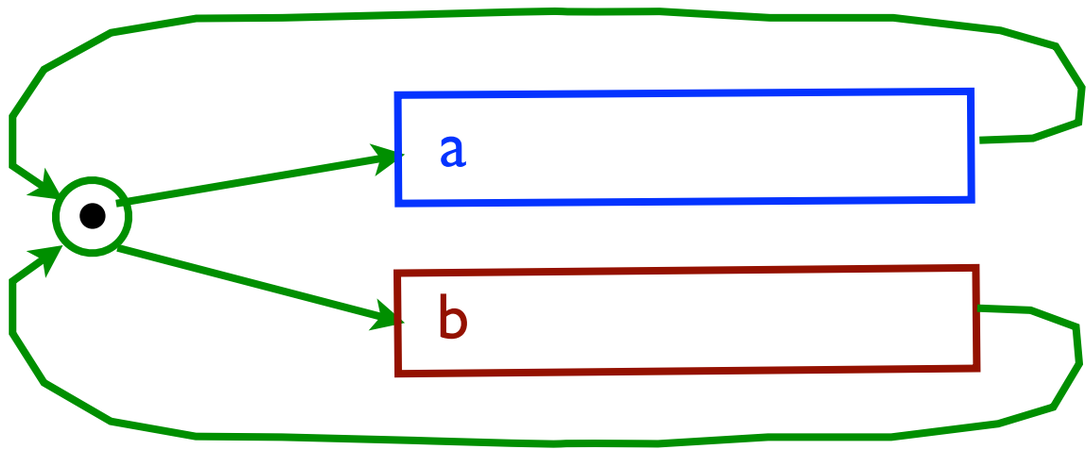
## a) Einseitige Synchronisation




```
Sema s(0);

Thread1                 Thread2
                          s.P();
  a();                    b();
  s.V();
```

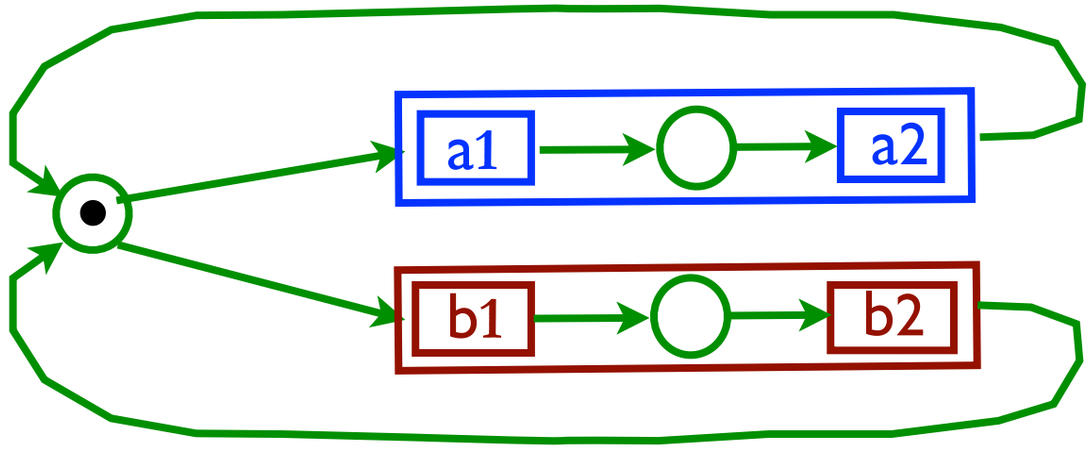
# b) Mehrseitige Synchronisation

```
Sema m(1);

Thread1                 Thread2
 m.P();                  m.P();
 a();                    b();
 m.V();                  m.V();
```



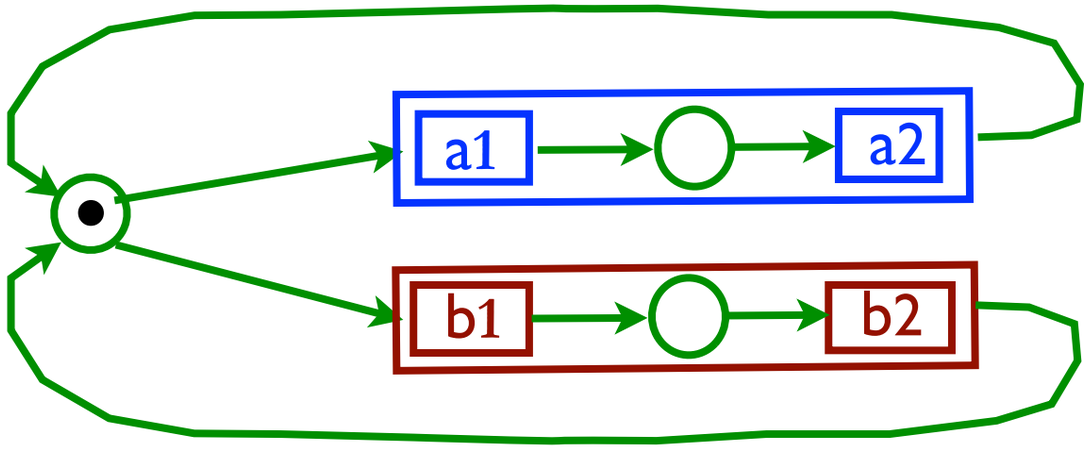
# b) Mehrseitige Synchronisation

```
Sema m(1);

Thread1                 Thread2
 m.P();                  m.P();
 a1();                   b1();
 a2();                   b2();
 m.V();                  m.V();
```



## b) Mehrseitige Synchronisation

```
Sema m(1);

Thread1          Thread2
 m.P();           m.P();
 a1();            b1();
 a2();            b2();
 m.V();           m.V();
```



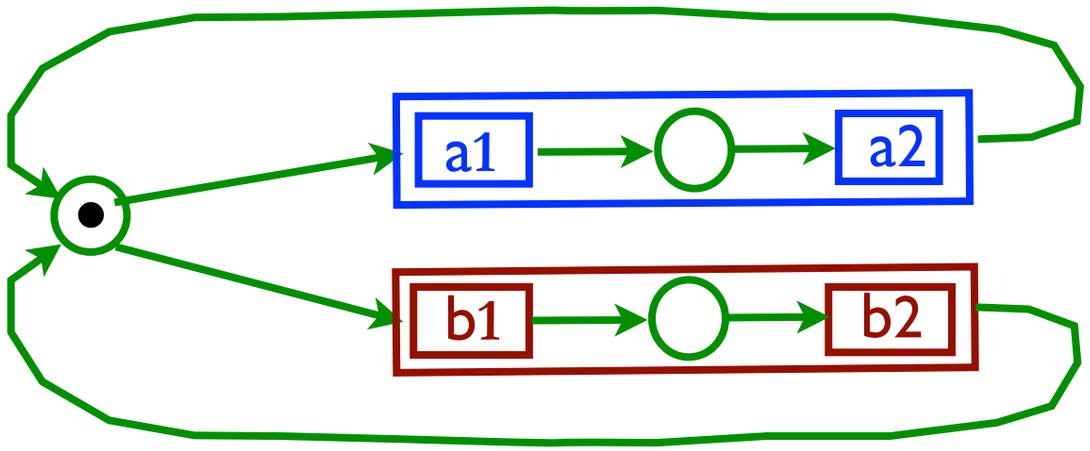

## c) Ressourcenverwaltung

Beispiel: > 3 Threads wollen das entsprechende Programmstück durchlaufen,
aber nur 3 können es nebenläufig tun,…

```
Sema n(3);

n.P();
a();
n.V();
```

## b) Mehrseitige Synchronisation

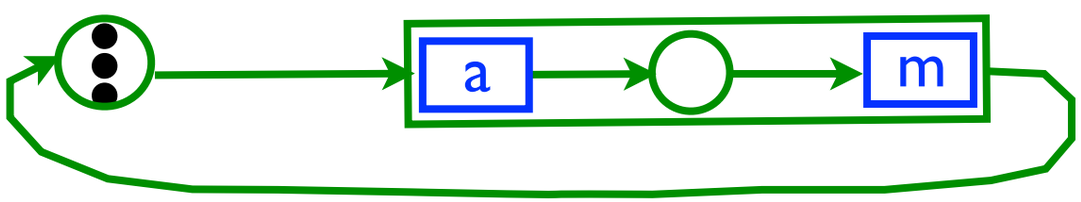
```
Sema m(1);

Thread1                 Thread2
 m.P();                  m.P();
 a1();                   b1();
 a2();                   b2();
 m.V();                  m.V();
```




## c) Ressourcenverwaltung

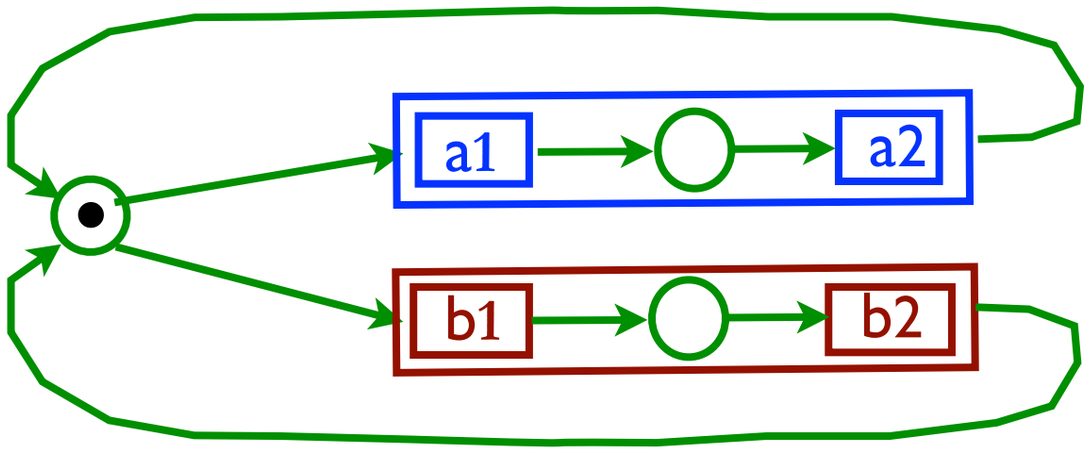
Beispiel: > 3 Threads wollen das entsprechende Programmstück durchlaufen,
aber nur 3 können es nebenläufig tun,…

```
Sema n(3);

n.P();
a();
      m();
      n.V();
```



## b) Mehrseitige Synchronisation

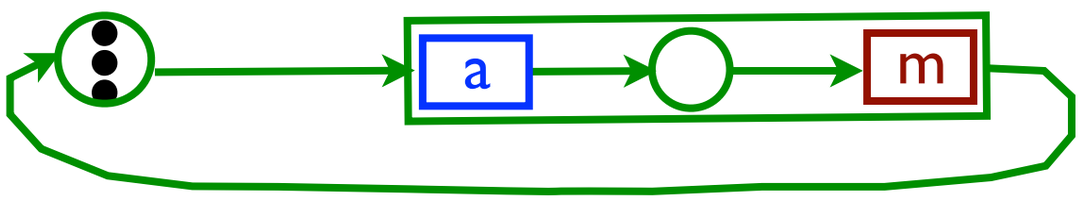
```
Sema m(1);

Thread1          Thread2
 m.P();           m.P();
 a1();            b1();
 a2();            b2();
 m.V();           m.V();
```




## c) Ressourcenverwaltung

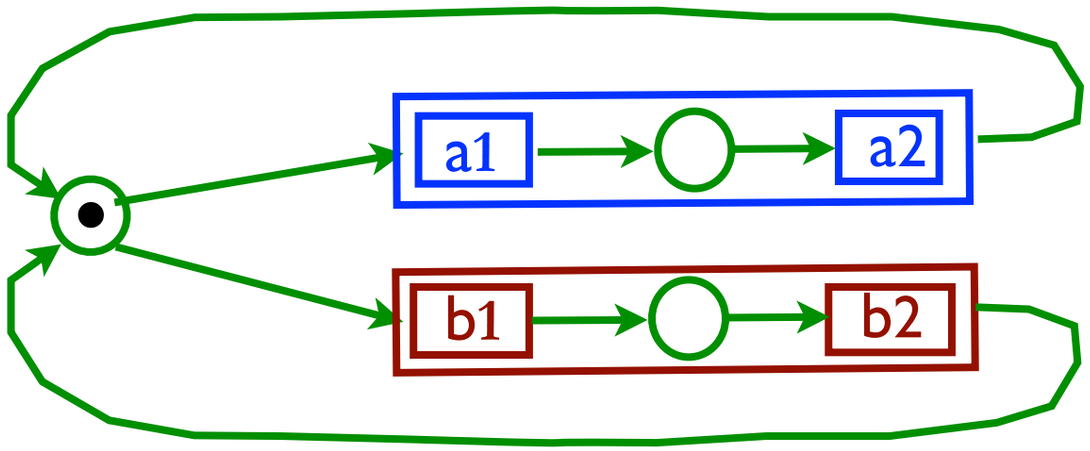
Beispiel: > 3 Threads wollen das entsprechende Programmstück durchlaufen,
aber nur 3 können es nebenläufig tun,…

```
Sema n(3);


n.P();
a();              m();
                  n.V();
```



## b) Mehrseitige Synchronisation

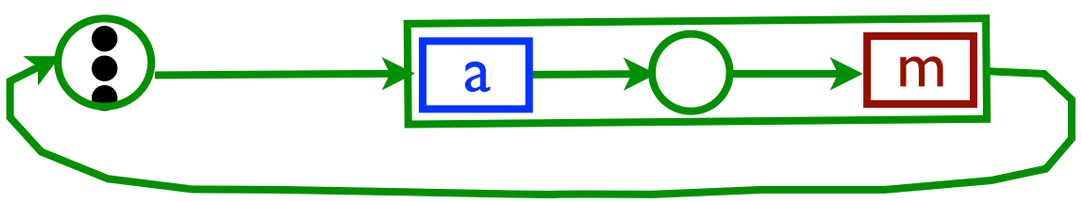
```
Sema m(1);

Thread1          Thread2
 m.P();           m.P();
 a1();            b1();
 a2();            b2();
 m.V();           m.V();
```




## c) Ressourcenverwaltung

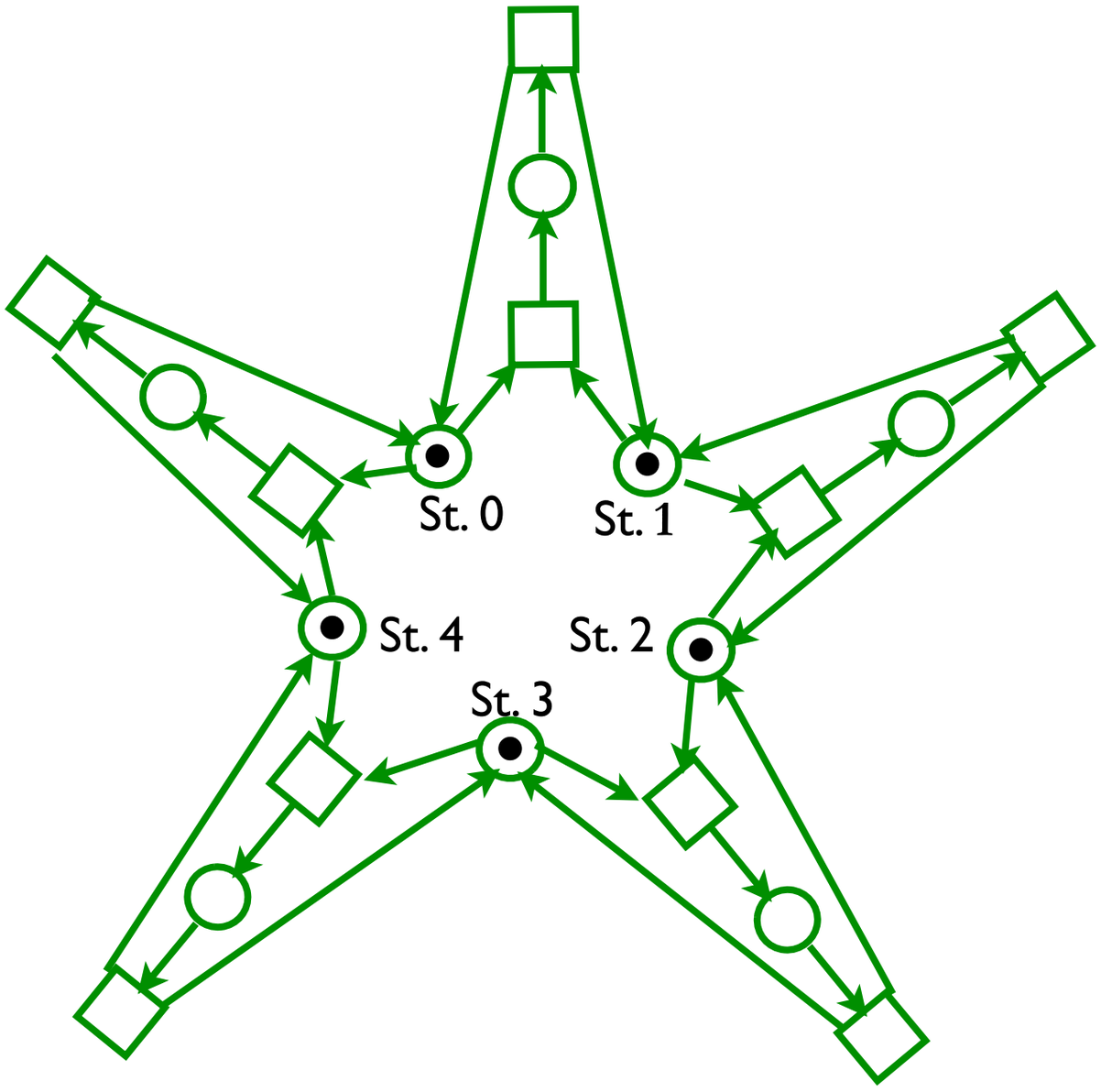
Beispiel: > 3 Threads wollen das entsprechende Programmstück durchlaufen,
aber nur 3 können es nebenläufig tun,…

```
Sema n(3);
Sema s(0);

n.P();           s.P();
a();             m();
s.V();           n.V();
```




Ggf. Kombination dieser Programmfragmente bei der Umsetzung komplexerer Petri-Netze

# Beispiel: Speisende Philosophen (Lösung 2)




Mit obigen Regeln nicht umsetzbar

- Auch bei dieser Lösung: Gleichzeitiges Hochnehmen der Stäbchen modelliert, dadurch Verklemmung verhindert

  ⇒ Modellierung des Wunschverhaltens

  ⇒ keine direkte Semaphor-Umsetzung...

# Grenzen von „klassischen“ Petri-Netzen:

- nicht unbedingt direkt in Code umsetzbar

Außerdem:

- werden leicht zu groß ⇒ unübersichtlich
- fehlende Prädikate/Zeitaussagen

  ⇒ viele Varianten entwickelt

# Exkurs: Pfadausdrücke (historisch)

Notation für explizite Synchronisationsvorschriften zwischen Prozessen/Threads

⇒ enthält Aussagen über Einschränkungen der möglichen Nebenläufigkeit der Operationen (+ automatische Umsetzung in Semaphoren)

# Exkurs: Pfadausdrücke (historisch)

Notation für explizite Synchronisationsvorschriften zwischen Prozessen/Threads

⇒ enthält Aussagen über Einschränkungen der möglichen Nebenläufigkeit der Operationen (+ automatische Umsetzung in Semaphoren)

## a) Erzeuger/Verbraucher

`erzeugen; verbrauchen` Erst erzeugen, dann verbrauchen

# Exkurs: Pfadausdrücke (historisch)

Notation für explizite Synchronisationsvorschriften zwischen Prozessen/Threads

⇒ enthält Aussagen über Einschränkungen der möglichen Nebenläufigkeit der Operationen (+ automatische Umsetzung in Semaphoren)

## a) Erzeuger/Verbraucher

`erzeugen; verbrauchen` Erst erzeugen, dann verbrauchen

`5:(erzeugen; verbrauchen)` Erst erzeugen, dann verbrauchen, 5 Pufferpl.

# Exkurs: Pfadausdrücke (historisch)

Notation für explizite Synchronisationsvorschriften zwischen Prozessen/Threads

⇒ enthält Aussagen über Einschränkungen der möglichen Nebenläufigkeit der Operationen (+ automatische Umsetzung in Semaphoren)

## a) Erzeuger/Verbraucher

| | |
|---|---|
| `erzeugen; verbrauchen` | Erst erzeugen, dann verbrauchen |
| `5:(erzeugen; verbrauchen)` | Erst erzeugen, dann verbrauchen, 5 Pufferpl. |

## b) Kritischer Abschnitt

| | |
|---|---|
| `(1:a) + (1:b)` | Ein a alternativ zu einem b |
| `1:(a+b)` | Eine Ausführung, entweder a oder b |

# Exkurs: Pfadausdrücke (historisch)

Notation für explizite Synchronisationsvorschriften zwischen Prozessen/Threads

⇒ enthält Aussagen über Einschränkungen der möglichen Nebenläufigkeit der Operationen (+ automatische Umsetzung in Semaphoren)

## a) Erzeuger/Verbraucher

| | |
|---|---|
| `erzeugen; verbrauchen` | Erst erzeugen, dann verbrauchen |
| `5:(erzeugen; verbrauchen)` | Erst erzeugen, dann verbrauchen, 5 Pufferpl. |

## b) Kritischer Abschnitt

| | |
|---|---|
| `(1:a) + (1:b)` | Ein a alternativ zu einem b |
| `1:(a+b)` | Eine Ausführung, entweder a oder b |

## c) Leser/Schreiber

| | |
|---|---|
| `lesen + (1:schreiben)` | Beliebig viele Leser oder ein Schreiber |

# Fragen – Teil 2

- Wie kann man durch ein Petri-Netz typische Synchronisationsvorschriften ausdrücken?
  a) Sequenz
  b) Beschränkte Nebenläufigkeit
  c) Alternative
  d) Unabhängigkeit
- Wie kann man den Schutz eines *kritischen Abschnitts* bzw. ein *Erzeuger-/Verbraucher*-Szenario mit Hilfe eines Petri-Netzes modellieren?

# Zusammenfassung

- Petri-Netze
  - Stellen, Transitionen, Markierungen
  - Visualisierung von Synchronisationsbedingungen
  - Umsetzung mit Semaphoren
- Kleiner Exkurs:  Pfadausdrücke

# Petri-Netze – Fragen

1. Aus welchen Komponenten besteht ein Petri-Netz (mit Marken)? Was kann man damit beschreiben?

2. Was kennzeichnet lebendige bzw. todesgefährdete Petri-Netze?

3. Wie kann man durch ein Petri-Netz typische Synchronisationsvorschriften ausdrücken?
   a) Sequenz
   b) Beschränkte Nebenläufigkeit
   c) Alternative
   d) Unabhängigkeit

4. Wie kann man den Schutz eines *kritischen Abschnitts* bzw. ein *Erzeuger-/Verbraucher*-Szenario mit Hilfe eines Petri-Netzes modellieren?