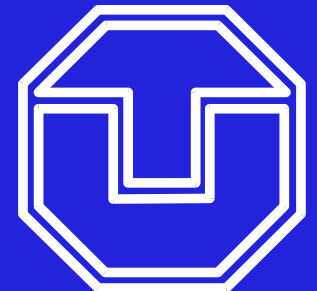


Verklemmungen

Betriebssysteme

Hermann Härtig
TU Dresden



Wegweiser

Begriff, Beispiele und Modellierung

Maßnahmen

Vermeiden per Konstruktion
Entdecken und Beseitigen

Beispiel Verklemmungen

Banküberweisung

```
void transfer(Account from, Account to, unsigned amount) {  
  
    from.lock();  
    to.lock();  
  
    from.balance -= amount;  
    to.balance   += amount;  
  
    to.unlock();  
    from.unlock();  
}
```

```
transfer(A, B, 100) || transfer(B, A, 50)
```

Definition Verklemmung

Verklemmung / Deadlock

Eine Menge von Threads heißt verklemmt (die Threads befinden sich in einem Deadlock), wenn jeder Thread dieser Menge auf ein *Ereignis* wartet, das nur von einem anderen Thread dieser Menge ausgelöst werden kann.

Ereignis:

- Botschaften, Semaphor wird frei
- Ursache in der Regel: Freigeben von Betriebsmitteln

Betriebsmittel

- CPU, Speicher, Geräte, ...
- Datenbank-Einträge, Datenstrukturen im Speicher, ...
- 1 Benutzer pro BM zu einem Zeitpunkt
- verdrängbar (preemptible) vs. nicht verdrängbar
 - CPU, Hauptspeicher können entzogen werden
 - Freigabe für Datenbank-Eintrag/Datenstruktur kann erst nach Ende des Auftrages/Transaktion entzogen werden

Sequenz

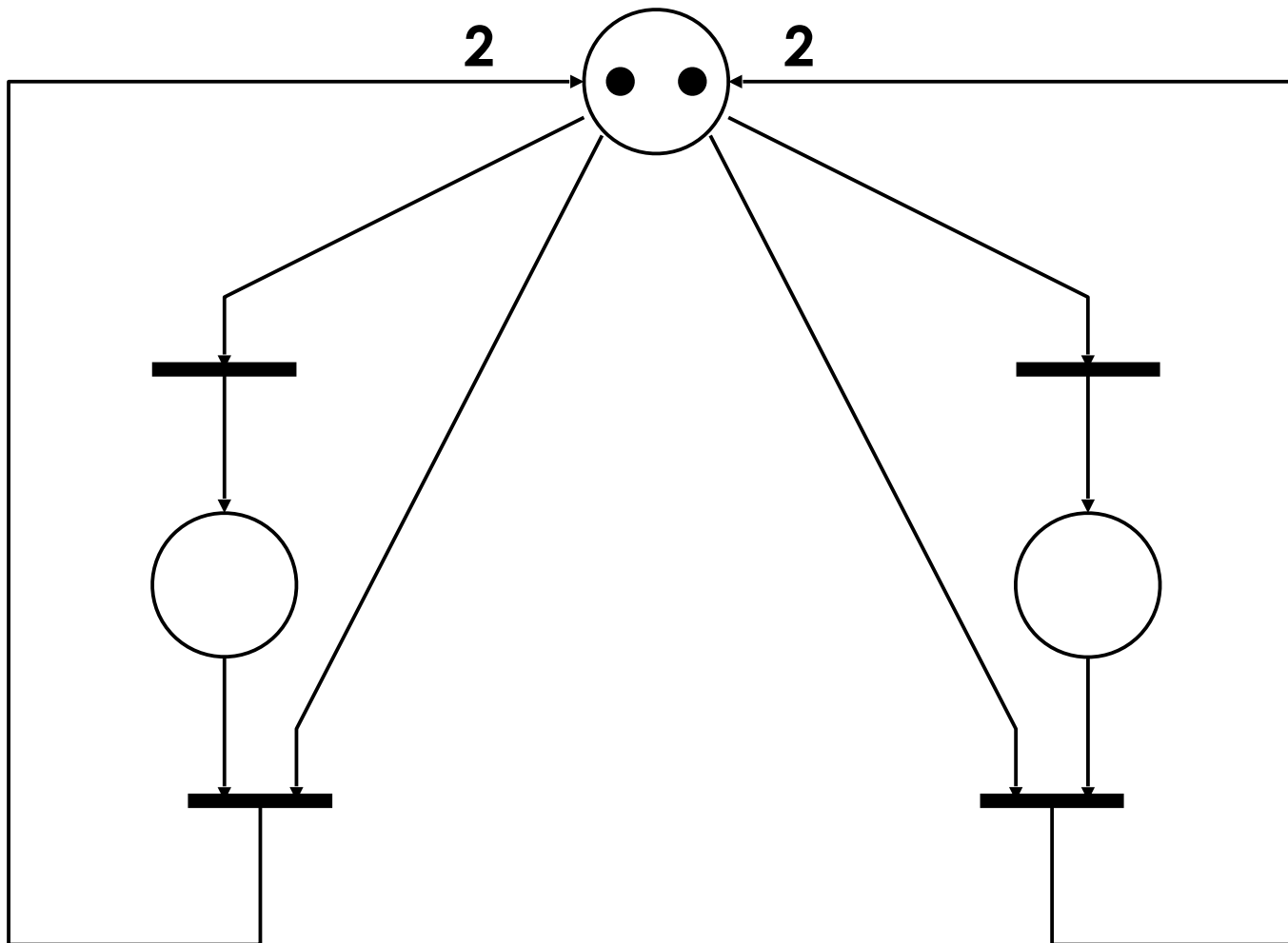
```
request ...  
use ...  
release ...
```

Zweck von Modellierung

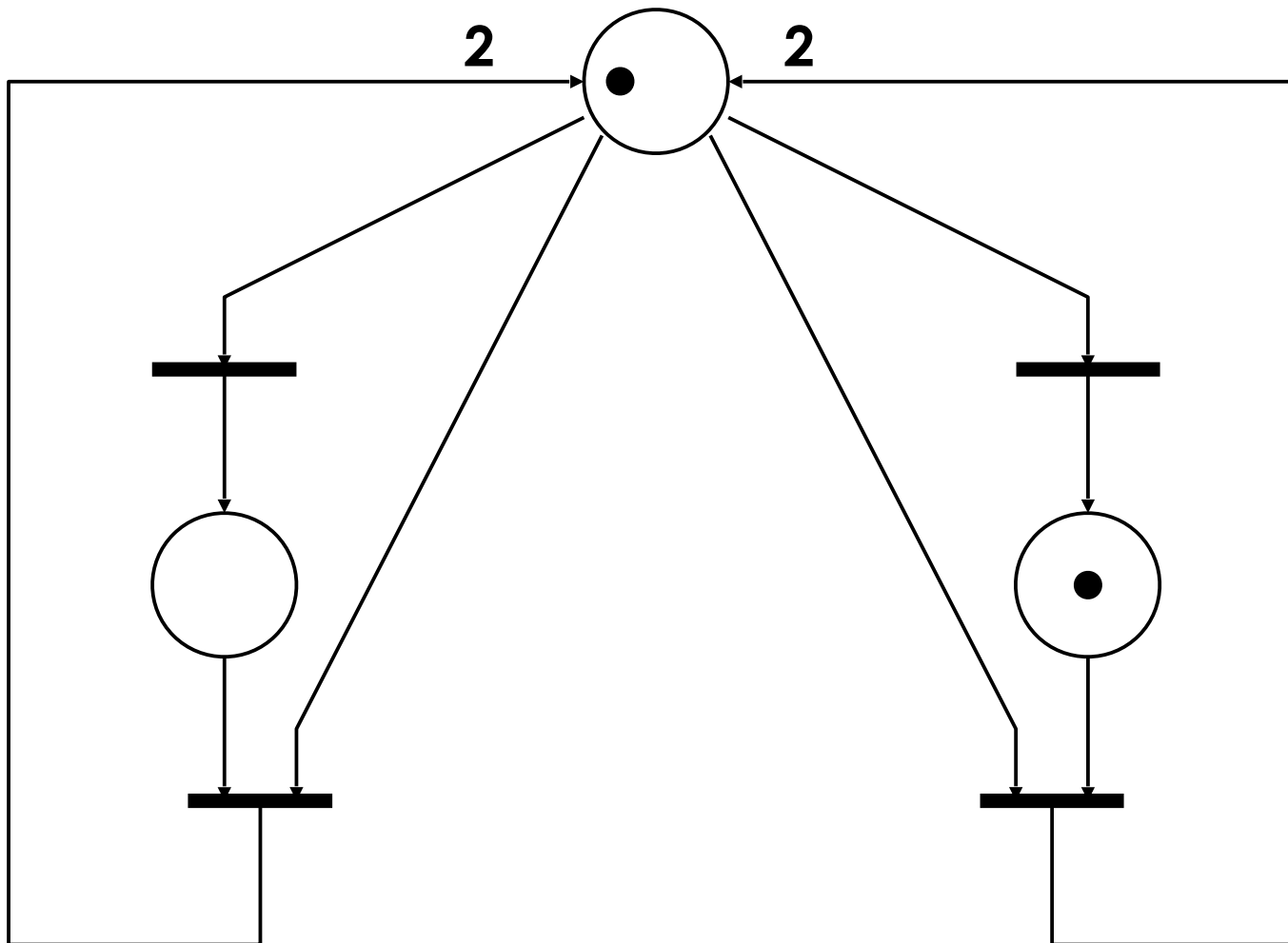
Abstrakteres Verstehen des Sachverhalts

- Feststellen, ob es sich bei einer “anomalen” Situation um eine Verklemmung handelt
- Analyse, ob für ein gegebenes Verfahren oder eine gegebene Situation eine Anforderungsfolge zur Verklemmung führen kann

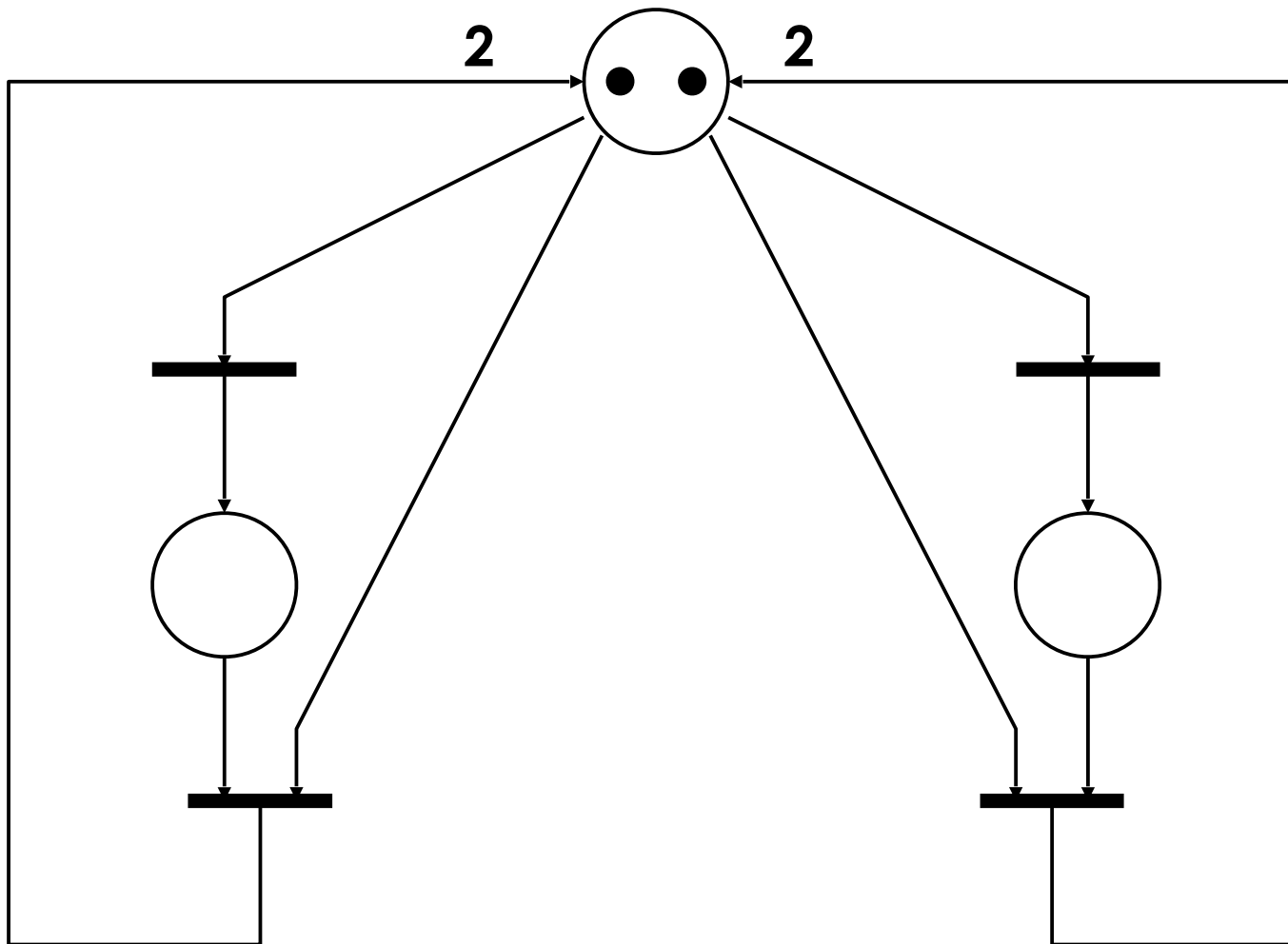
Modellierung mittels PETRI-Netzen



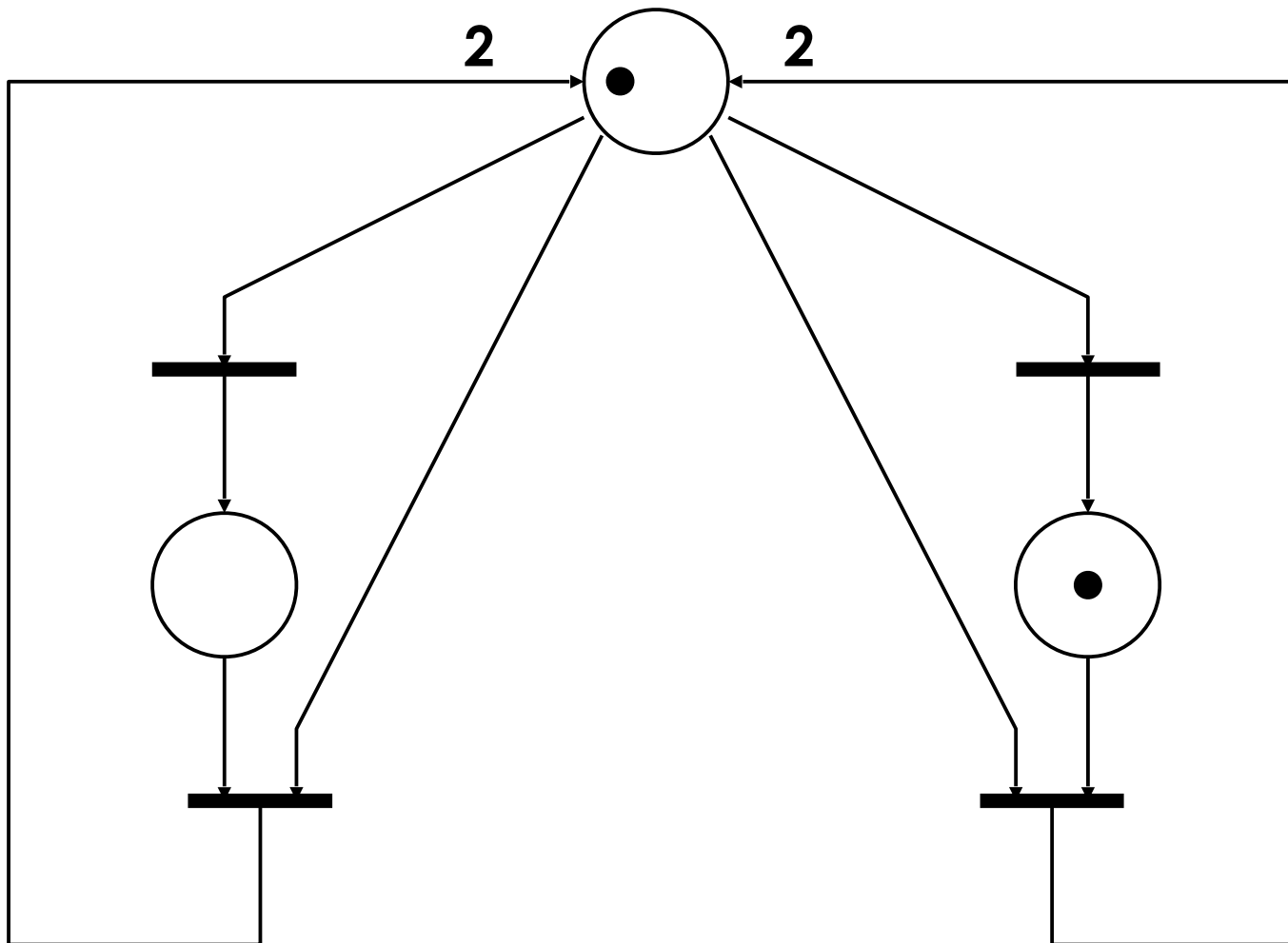
Modellierung mittels PETRI-Netzen



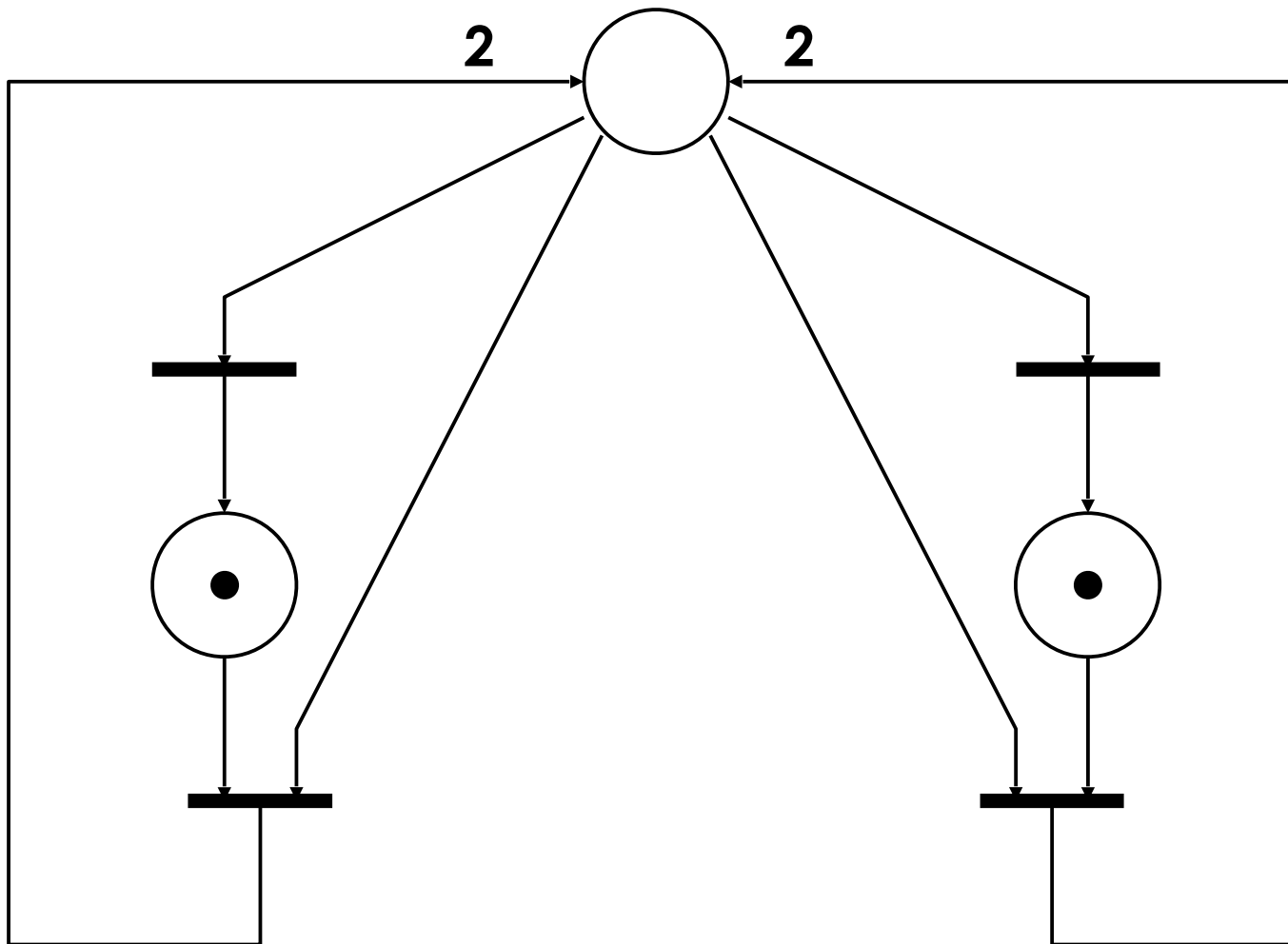
Modellierung mittels PETRI-Netzen



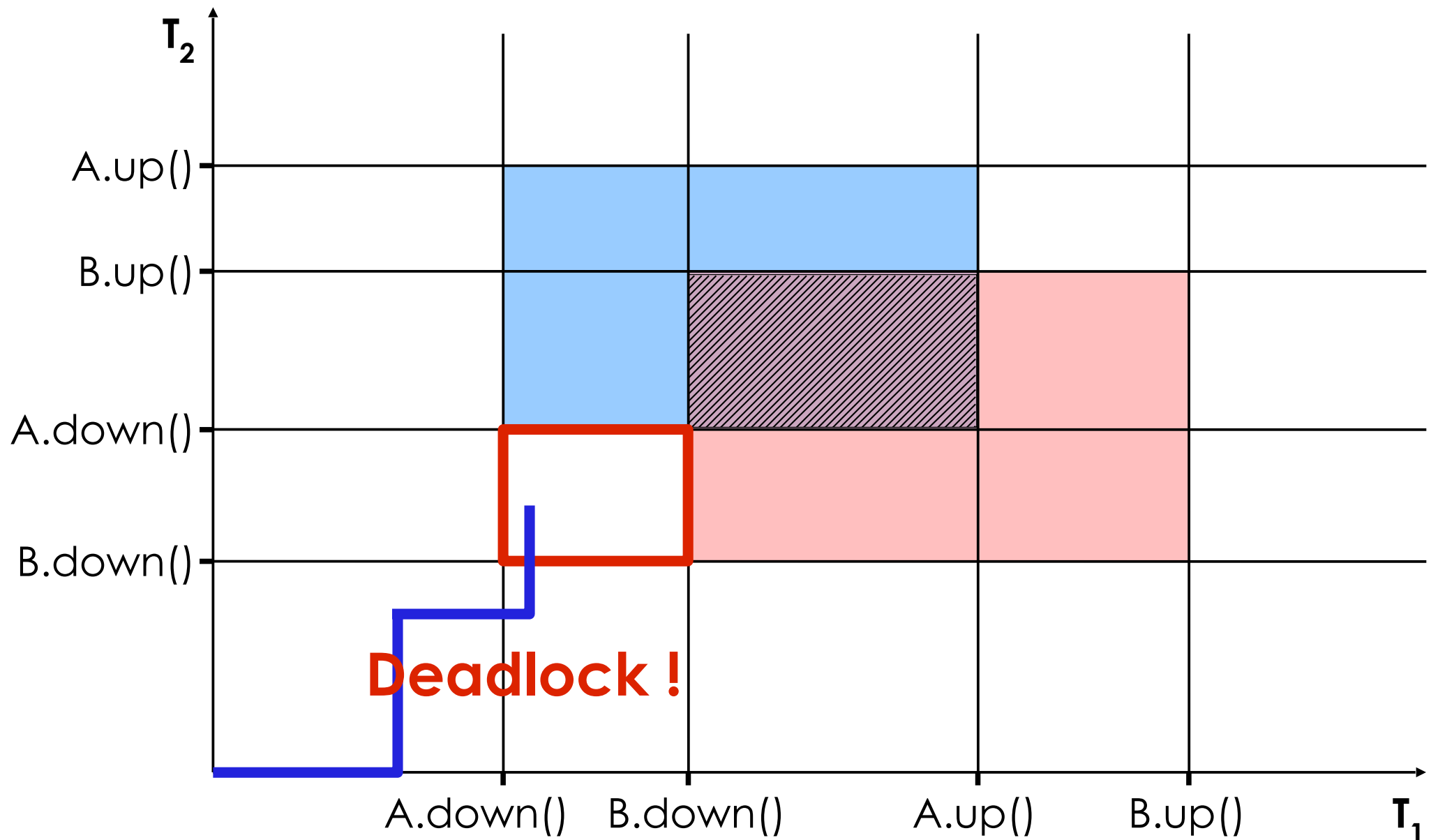
Modellierung mittels PETRI-Netzen



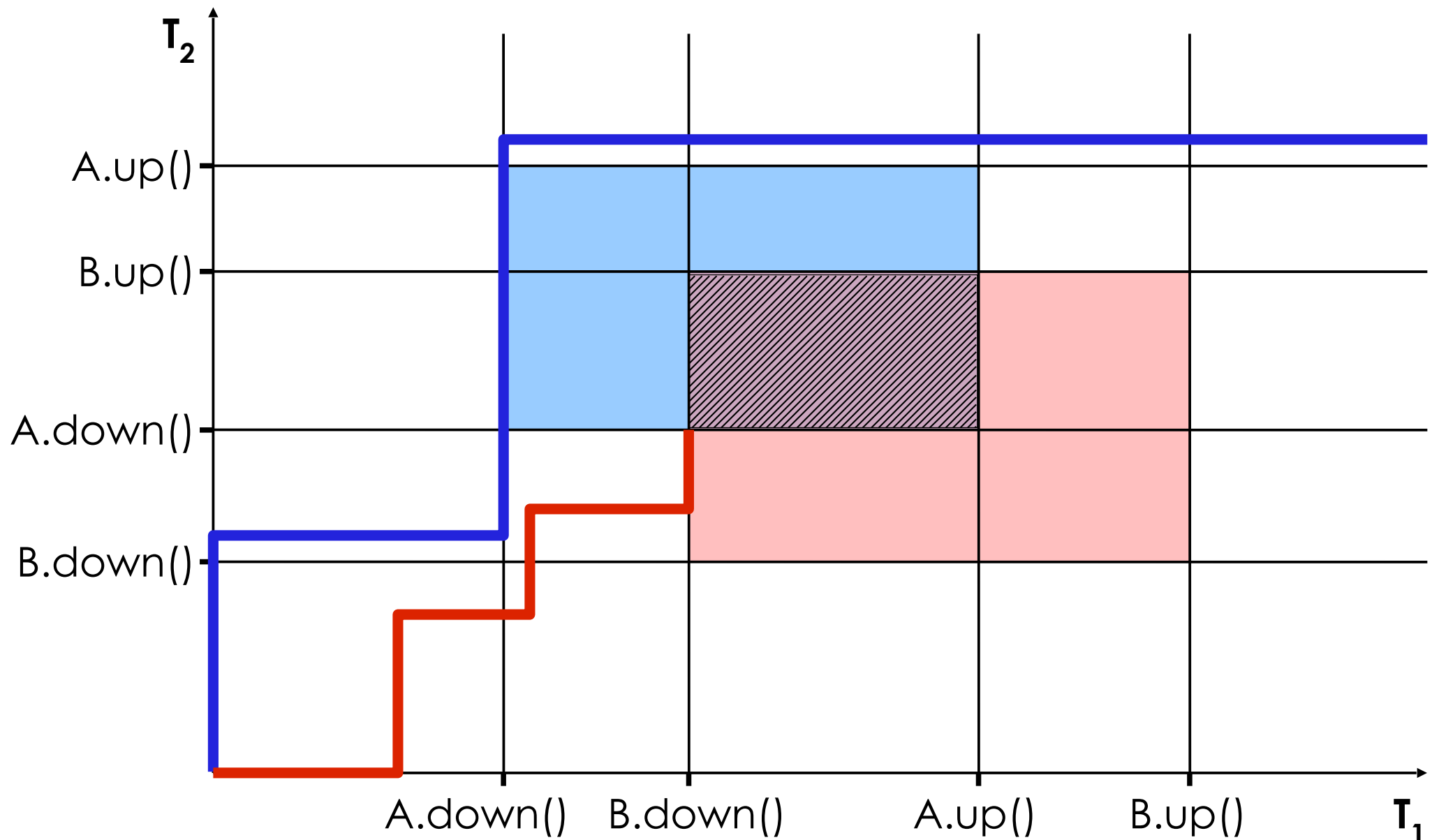
Modellierung mittels PETRI-Netzen



Modellierung mittels Prozessfortschritts-Diagrammen



Modellierung mittels Prozessfortschritts-Diagrammen



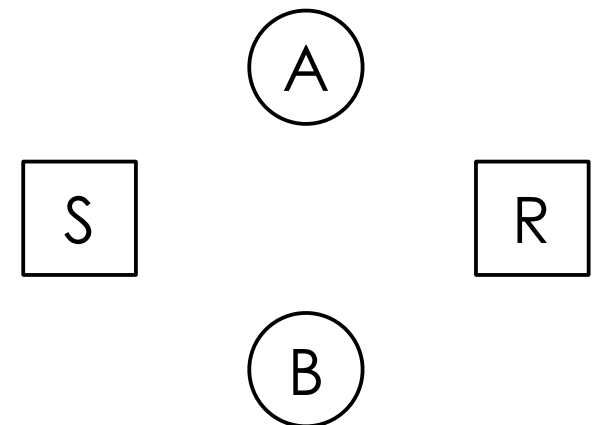
Modellierung mittels BM-Zuteilungsgraph

Betriebsmittel-Zuteilungsgraph (HOLT et al., 1972)

Zyklus im Graphen → zyklische Wartesituation

Beispiel

- ➡ • Thread A fordert S an
- Thread B fordert R an
- Thread A fordert R an
- Thread B fordert S an



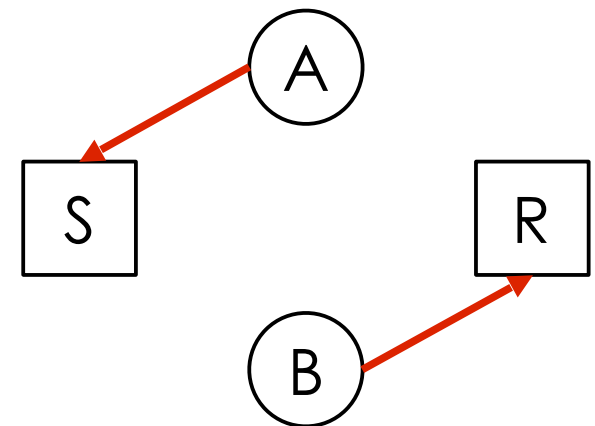
Modellierung mittels BM-Zuteilungsgraph

Betriebsmittel-Zuteilungsgraph (HOLT et al., 1972)

Zyklus im Graphen → zyklische Wartesituation

Beispiel

- Thread A besitzt S
- Thread B besitzt R
- ➡ • Thread A fordert R an
- Thread B fordert S an



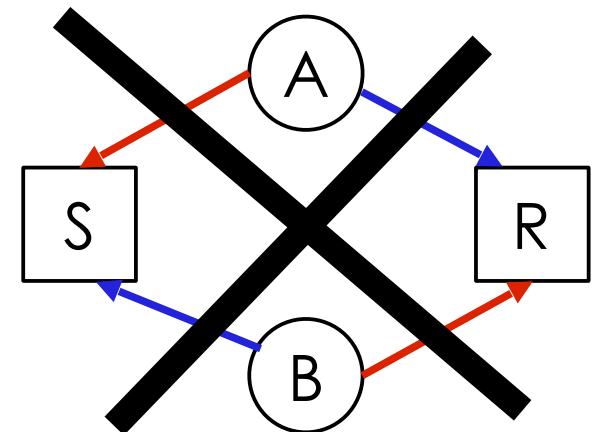
Modellierung mittels BM-Zuteilungsgraph

Betriebsmittel-Zuteilungsgraph (HOLT et al., 1972)

Zyklus im Graphen → zyklische Wartesituation

Beispiel

- Thread A besitzt S
- Thread B besitzt R
- Thread A fordert R an
- Thread B fordert S an



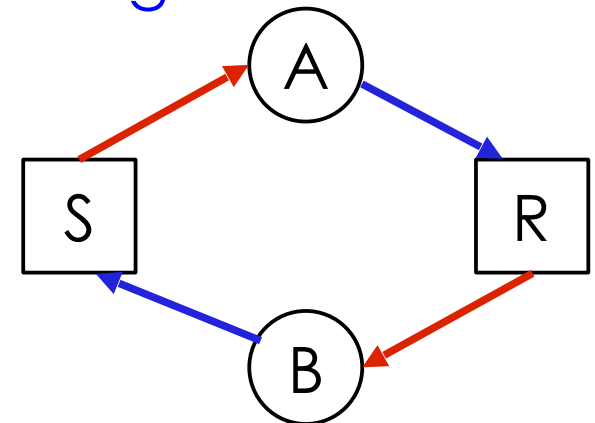
Modellierung mittels BM-Zuteilungsgraph

Betriebsmittel-Zuteilungsgraph (HOLT et al., 1972)

Zyklus im Graphen → zyklische Wartesituation

Beispiel

- Thread A besitzt S → S wartet für Freigabe auf A
- Thread B besitzt R → R wartet für Freigabe auf B
- Thread A fordert R an → A wartet auf Freigabe von R
- Thread B fordert S an → B wartet auf Freigabe von S



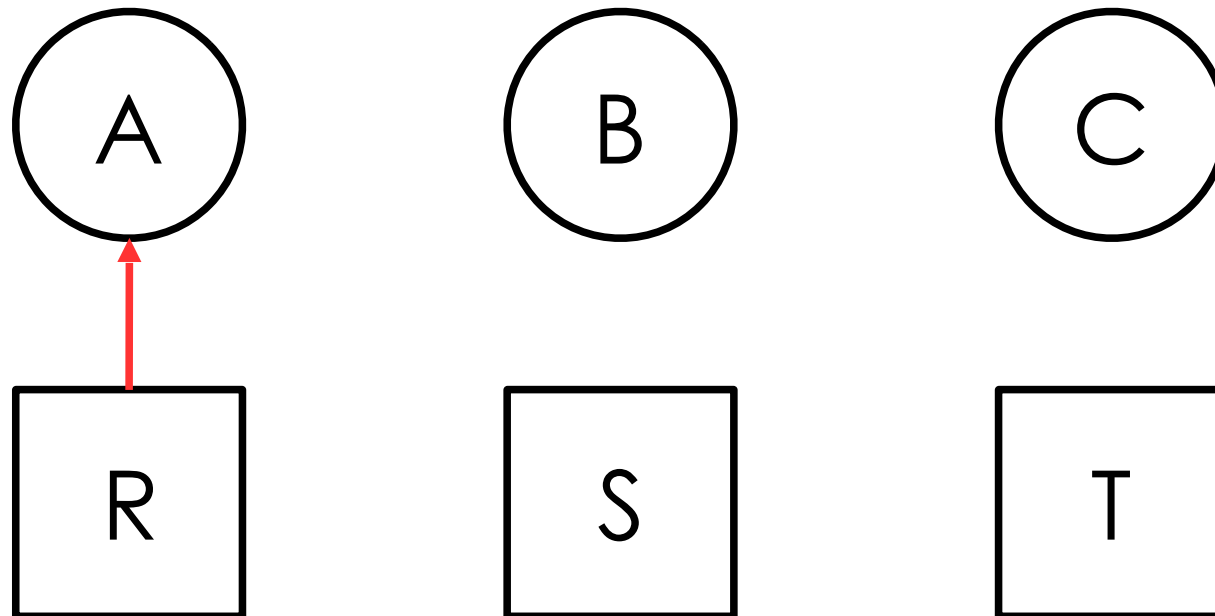
Beispiel (1) (aus Tanenbaum)

A :
→ request (R) ;
request (S) ;
release (R) ;
release (S) ;

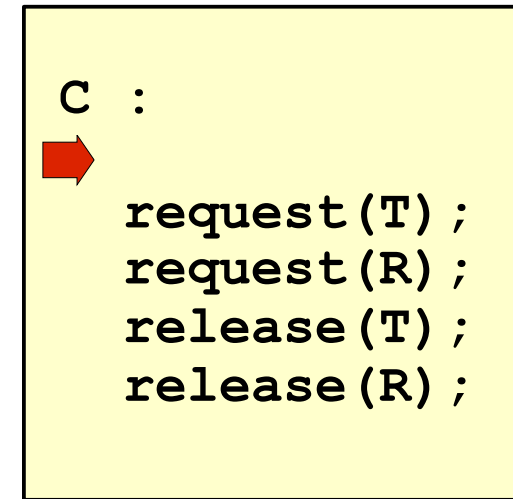
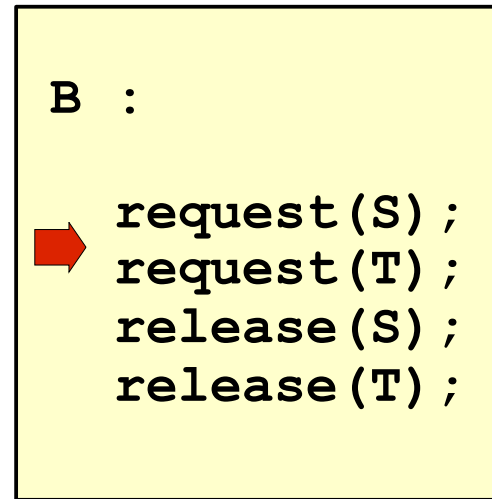
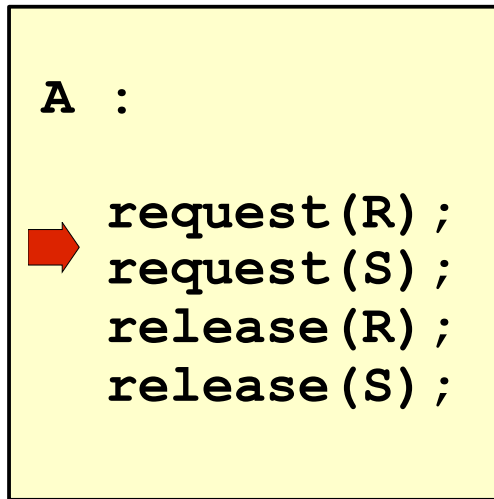
B :
→ request (S) ;
request (T) ;
release (S) ;
release (T) ;

C :
→ request (T) ;
request (R) ;
release (T) ;
release (R) ;

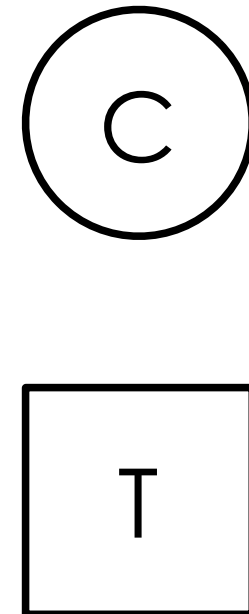
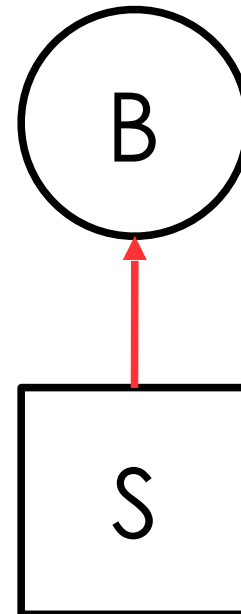
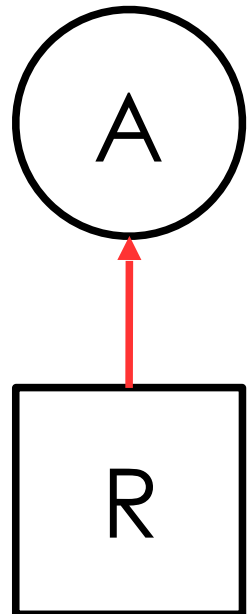
1. A fordert R



Beispiel (1)



1. A fordert R
2. B fordert S



Beispiel (1)

A :

➔ request (R) ;
request (S) ;
release (R) ;
release (S) ;

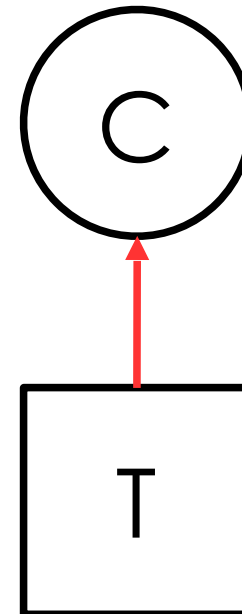
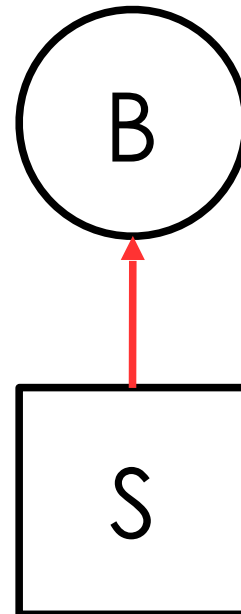
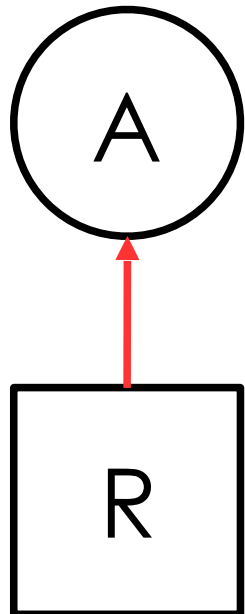
B :

➔ request (S) ;
request (T) ;
release (S) ;
release (T) ;

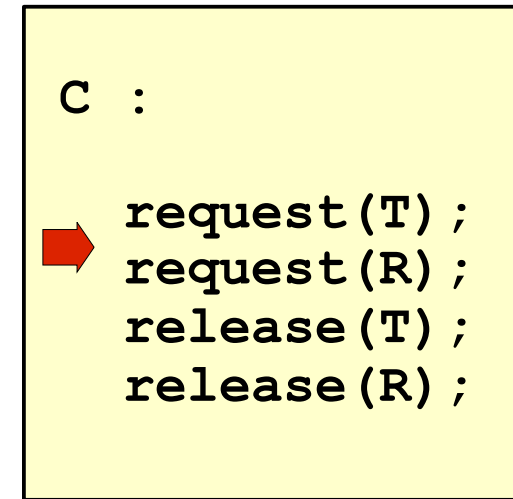
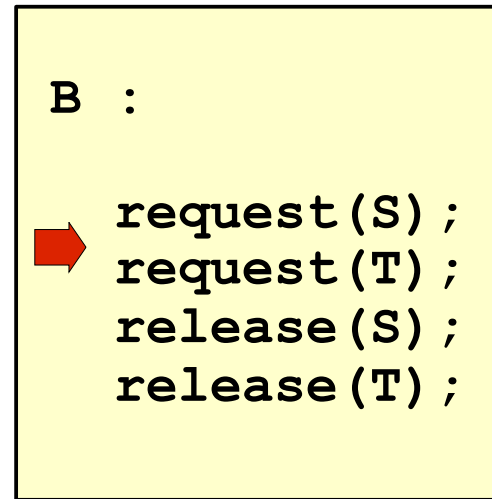
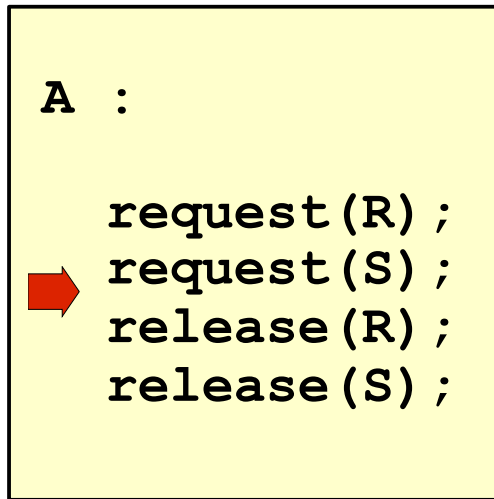
C :

➔ request (T) ;
request (R) ;
release (T) ;
release (R) ;

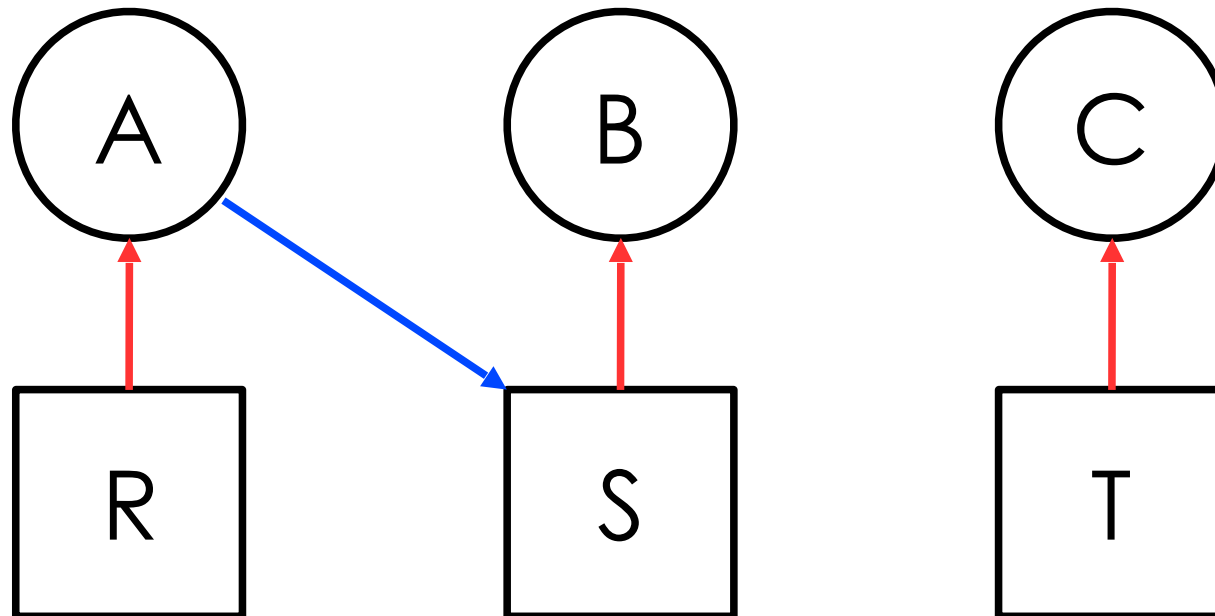
1. A fordert R
2. B fordert S
3. C fordert T



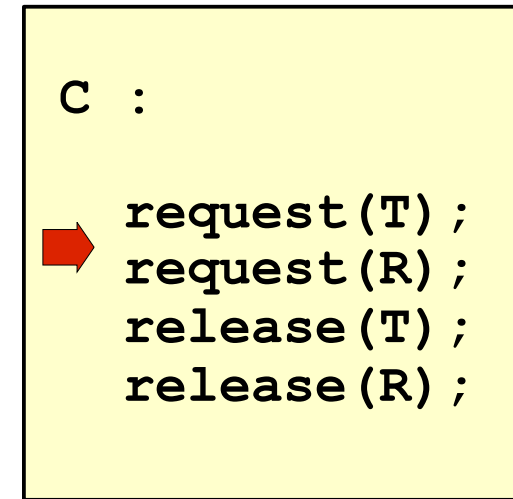
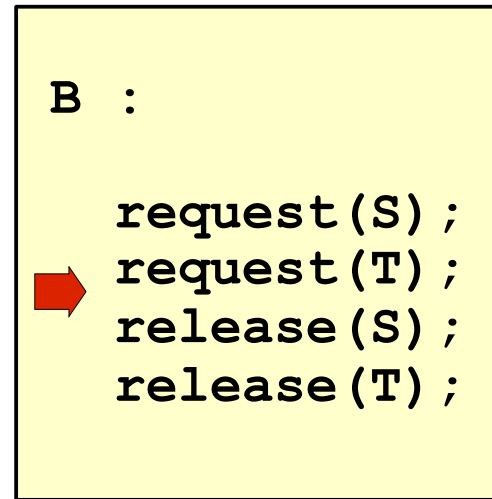
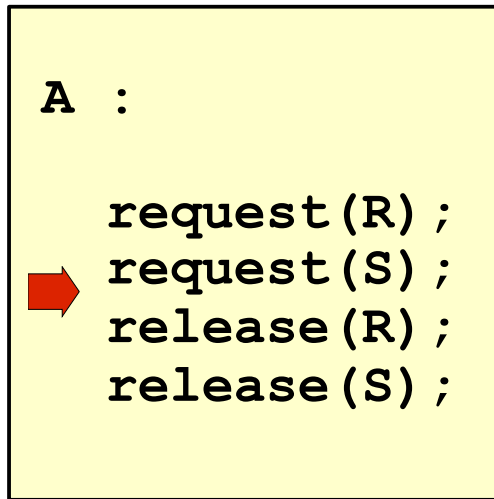
Beispiel (1)



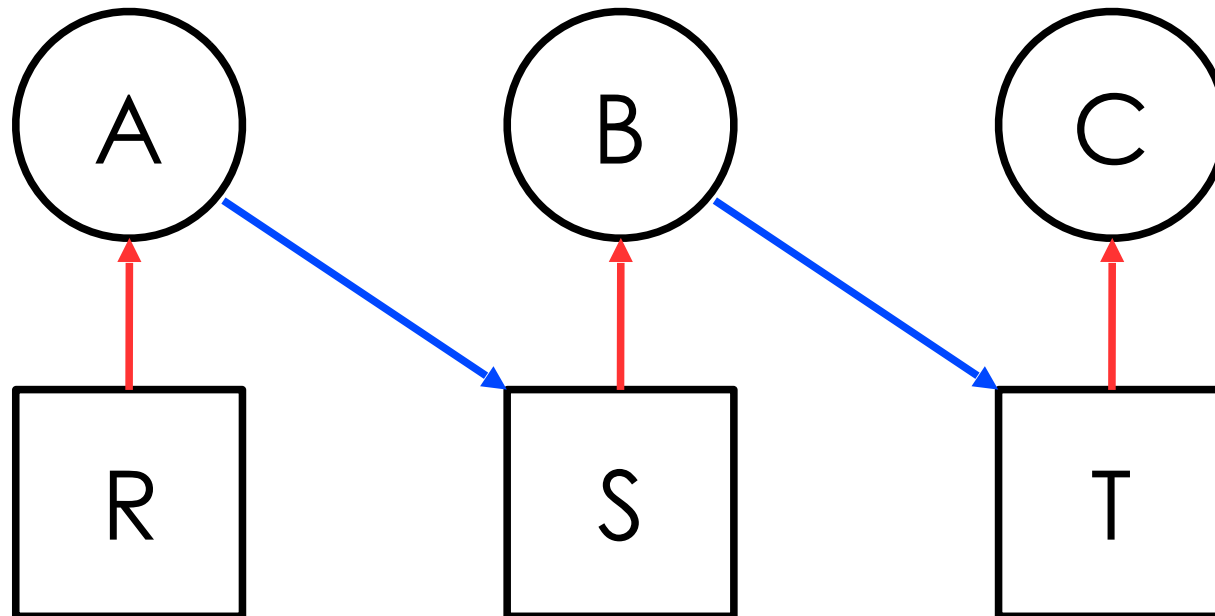
1. A fordert R
2. B fordert S
3. C fordert T
4. A fordert S



Beispiel (1)



1. A fordert R
2. B fordert S
3. C fordert T
4. A fordert S
5. B fordert T



Beispiel (1)

A :

request (R) ;
→ request (S) ;
release (R) ;
release (S) ;

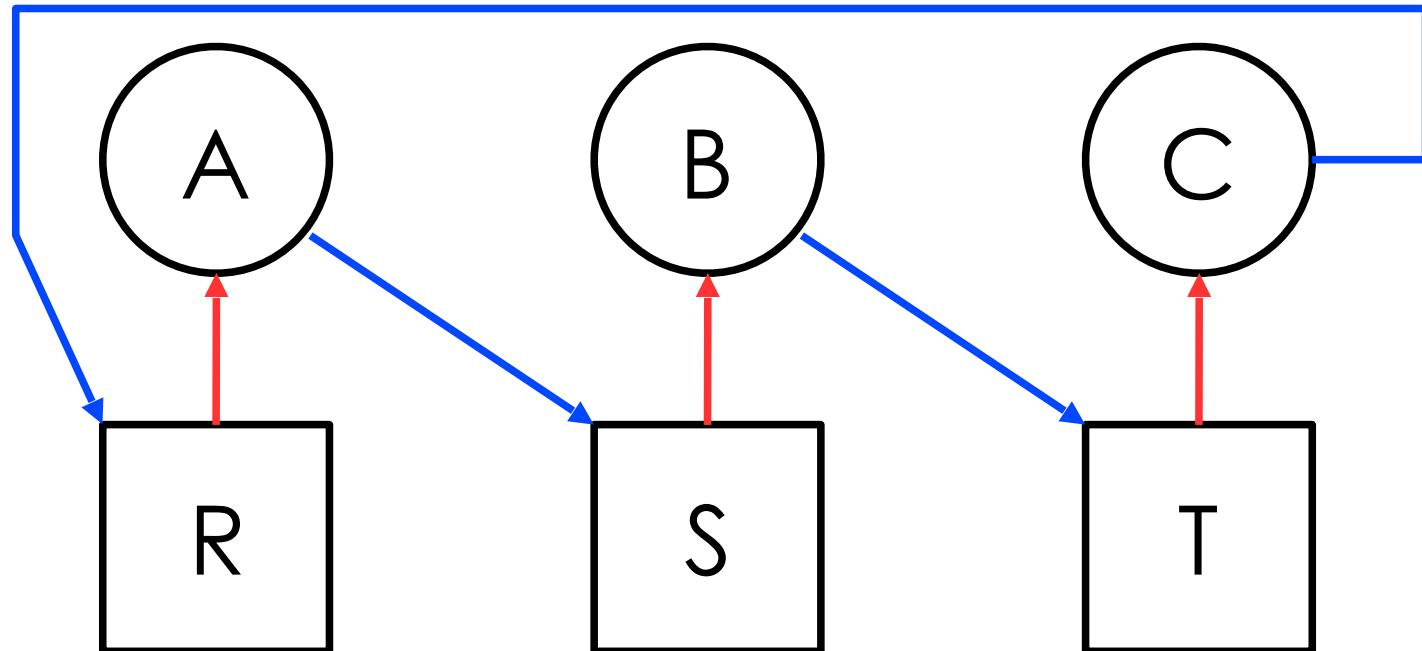
B :

request (S) ;
→ request (T) ;
release (S) ;
release (T) ;

C :

request (T) ;
→ request (R) ;
release (T) ;
release (R) ;

1. A fordert R
2. B fordert S
3. C fordert T
4. A fordert S
5. B fordert T
6. C fordert R



Beispiel (1)

A :

```
request (R) ;  
→ request (S) ;  
release (R) ;  
release (S) ;
```

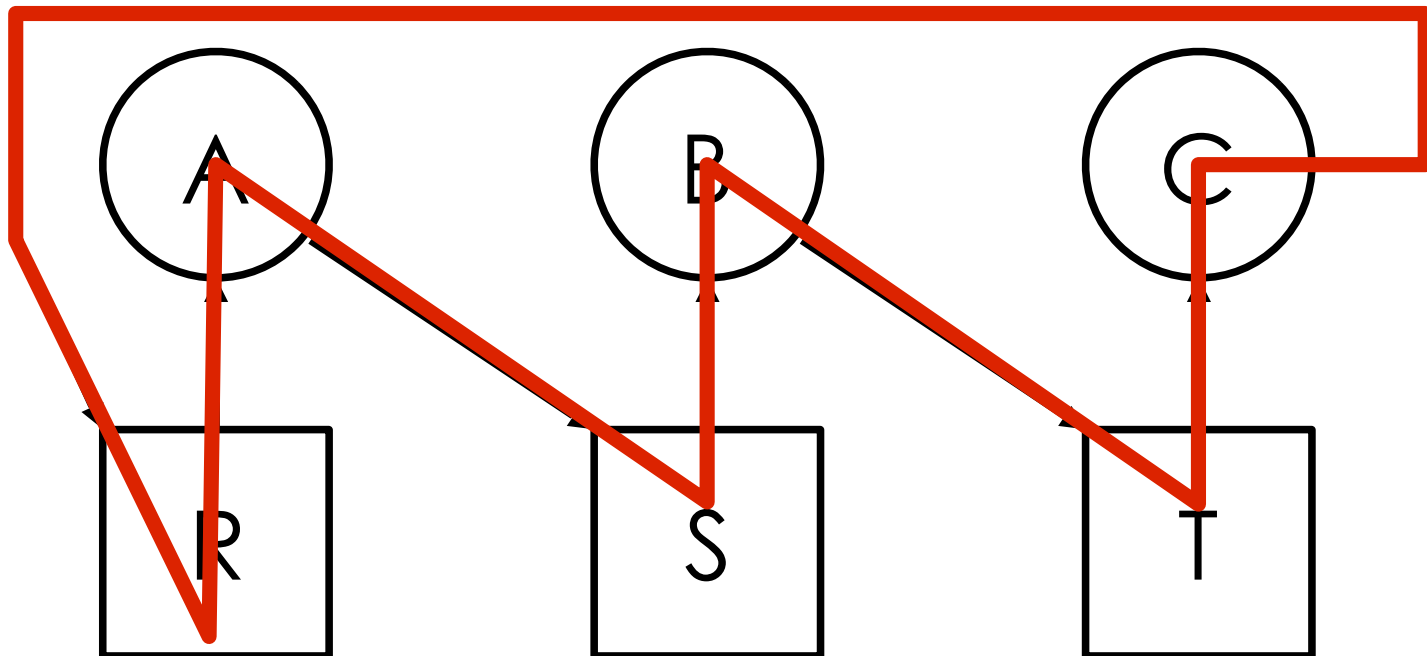
B :

```
request (S) ;  
→ request (T) ;  
release (S) ;  
release (T) ;
```

C :

```
request (T) ;  
→ request (R) ;  
release (T) ;  
release (R) ;
```

1. A fordert R
2. B fordert S
3. C fordert T
4. A fordert S
5. B fordert T
6. C fordert R



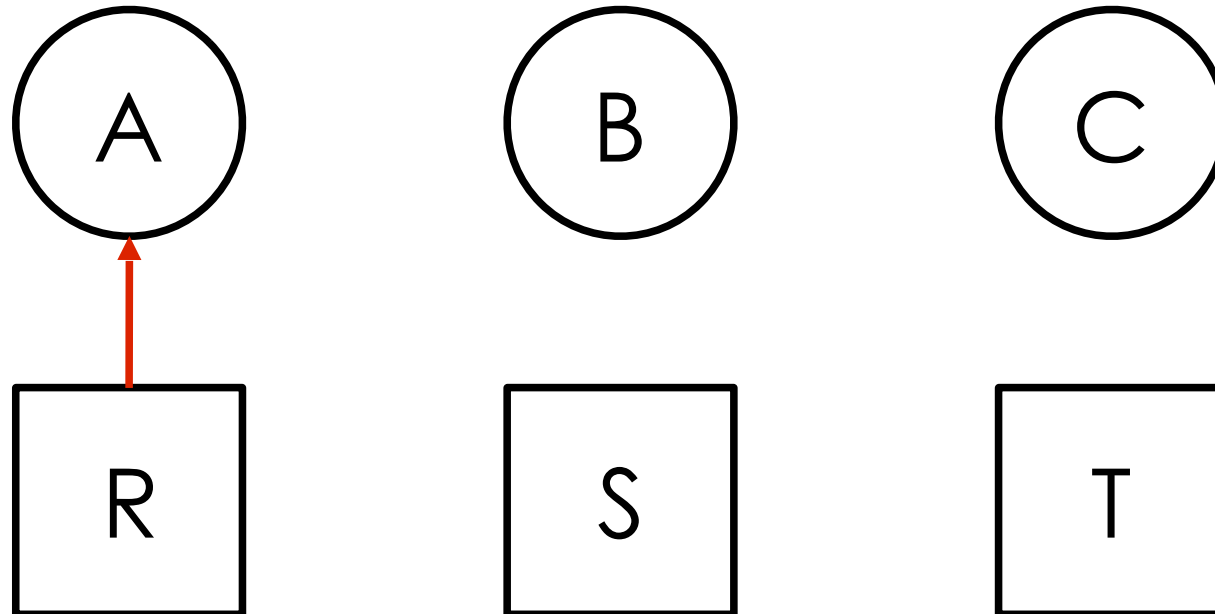
Beispiel (2)

A :
→ request (R) ;
request (S) ;
release (R) ;
release (S) ;

B :
→ request (S) ;
request (T) ;
release (S) ;
release (T) ;

C :
→ request (T) ;
request (R) ;
release (T) ;
release (R) ;

1. A fordert R



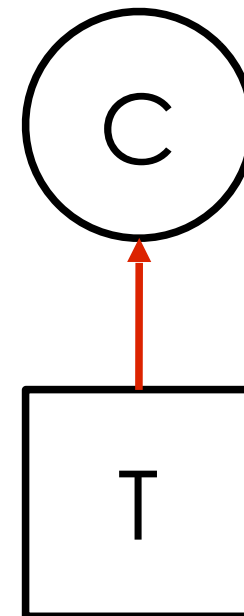
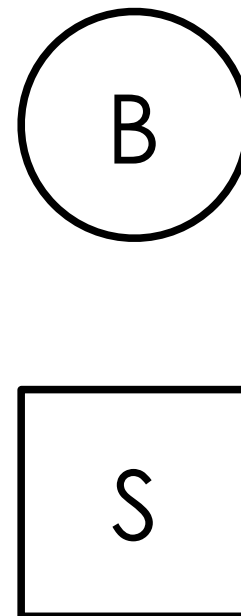
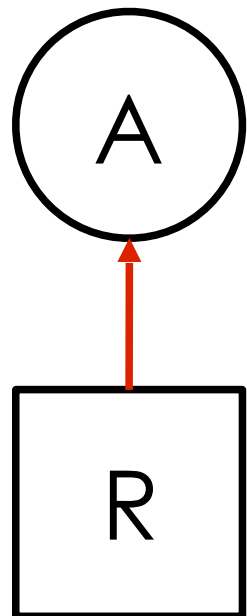
Beispiel (2)

A :
→ request (R) ;
request (S) ;
release (R) ;
release (S) ;

B :
→ request (S) ;
request (T) ;
release (S) ;
release (T) ;

C :
→ request (T) ;
request (R) ;
release (T) ;
release (R) ;

1. A fordert R
2. C fordert T



Beispiel (2)

A :

```
request (R) ;  
request (S) ;  
→ release (R) ;  
release (S) ;
```

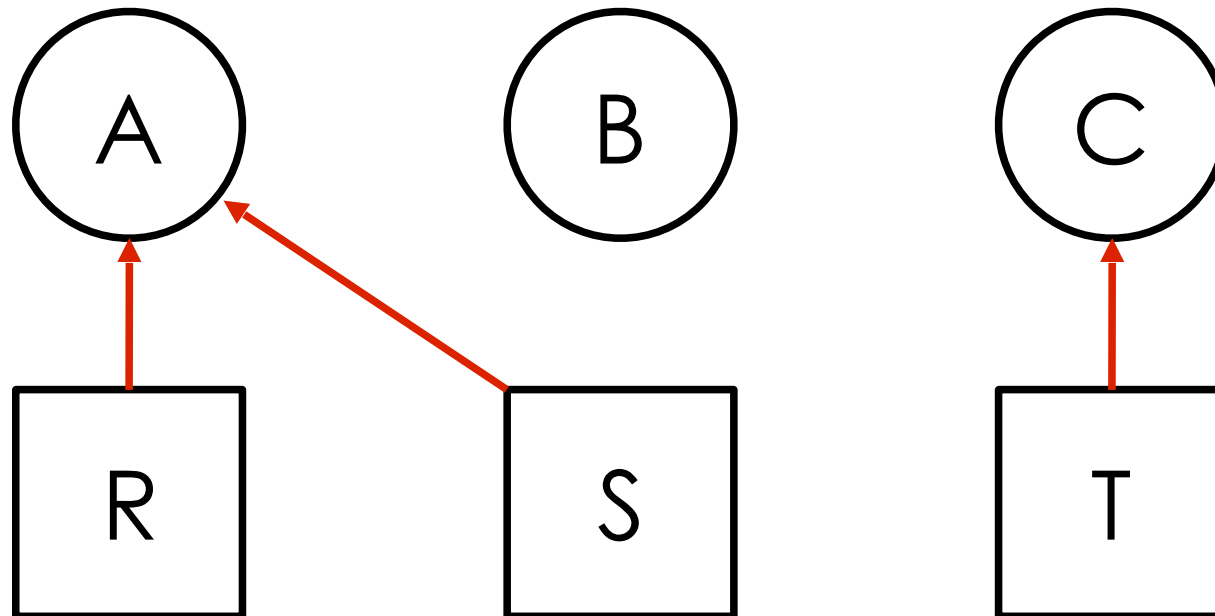
B :

```
→ request (S) ;  
request (T) ;  
release (S) ;  
release (T) ;
```

C :

```
→ request (T) ;  
request (R) ;  
release (T) ;  
release (R) ;
```

1. A fordert R
2. C fordert T
3. A fordert S



Beispiel (2)

A :

```
request (R) ;  
request (S) ;  
→ release (R) ;  
release (S) ;
```

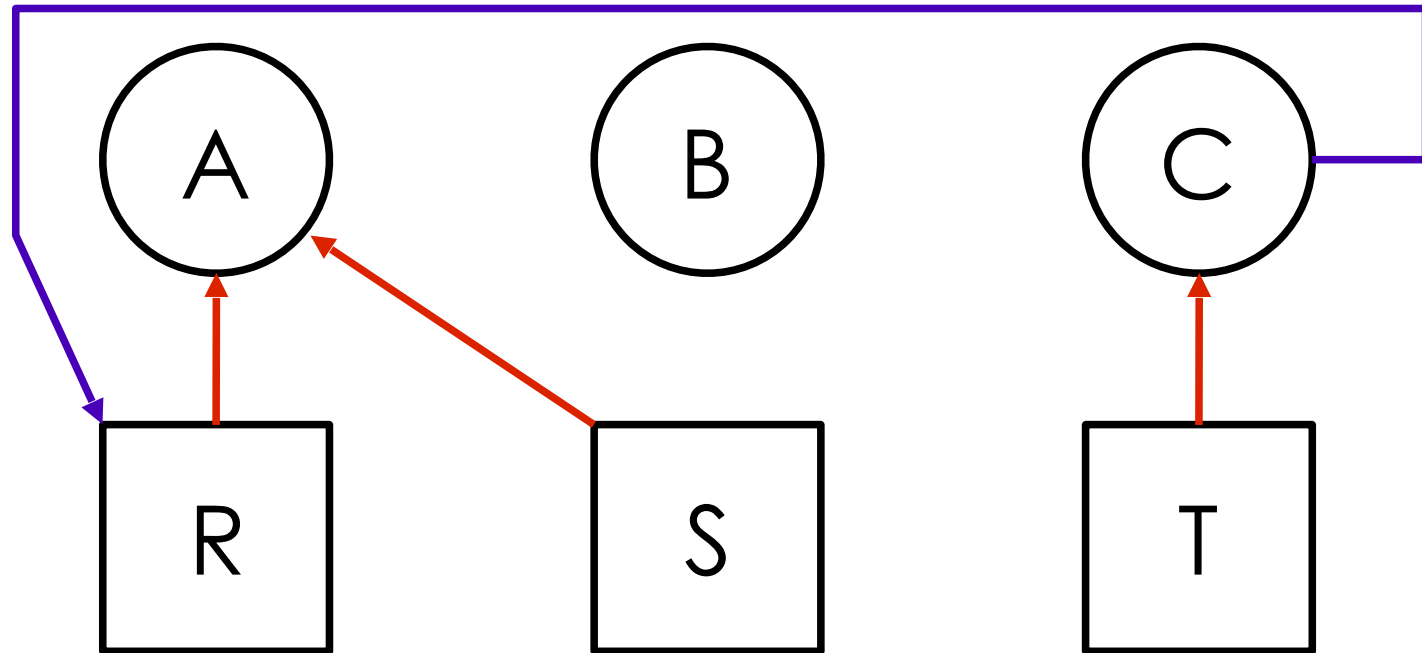
B :

```
→ request (S) ;  
request (T) ;  
release (S) ;  
release (T) ;
```

C :

```
request (T) ;  
request (R) ;  
→ release (T) ;  
release (R) ;
```

1. A fordert R
2. C fordert T
3. A fordert S
4. C fordert R



Beispiel (2)

A :

```
request (R) ;  
request (S) ;  
→ release (R) ;  
release (S) ;
```

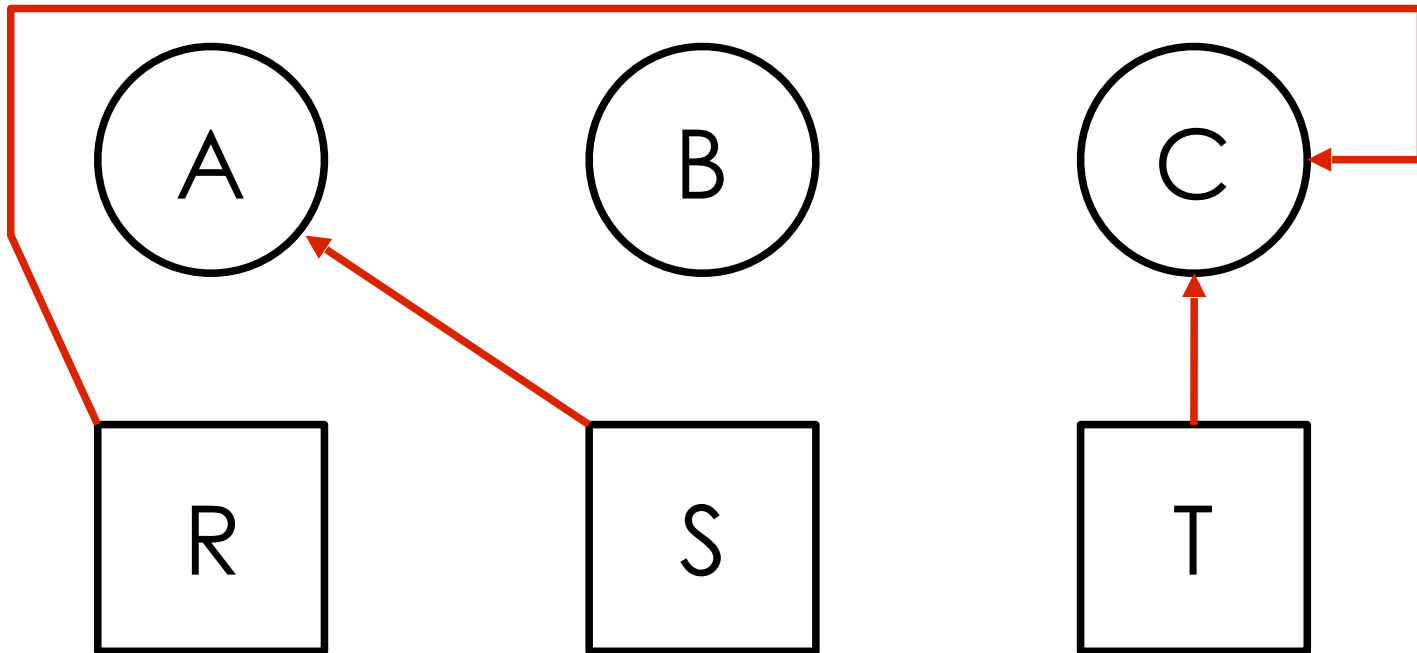
B :

```
→ request (S) ;  
request (T) ;  
release (S) ;  
release (T) ;
```

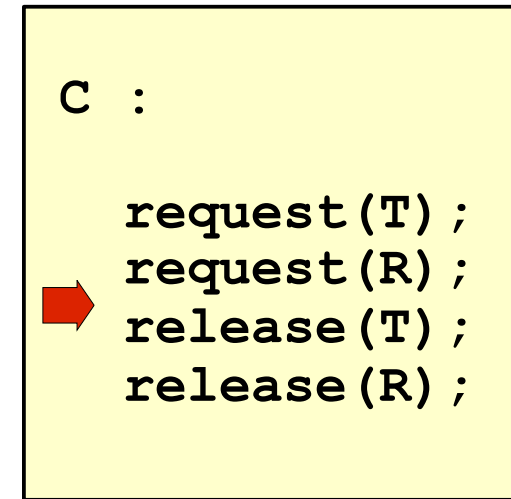
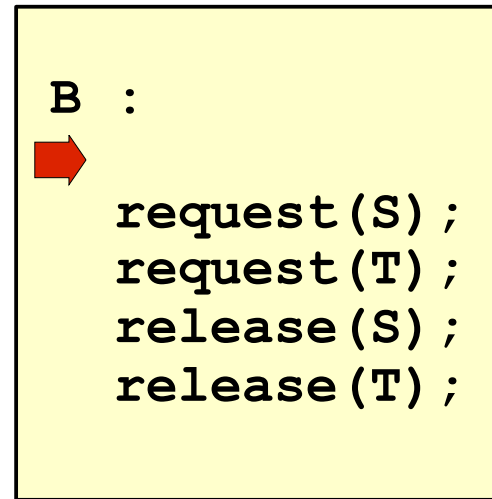
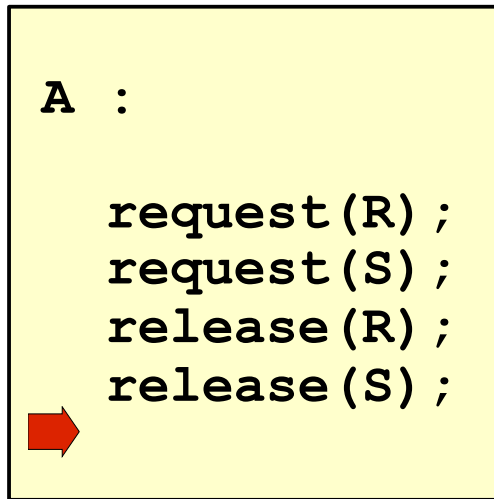
C :

```
request (T) ;  
request (R) ;  
→ release (T) ;  
release (R) ;
```

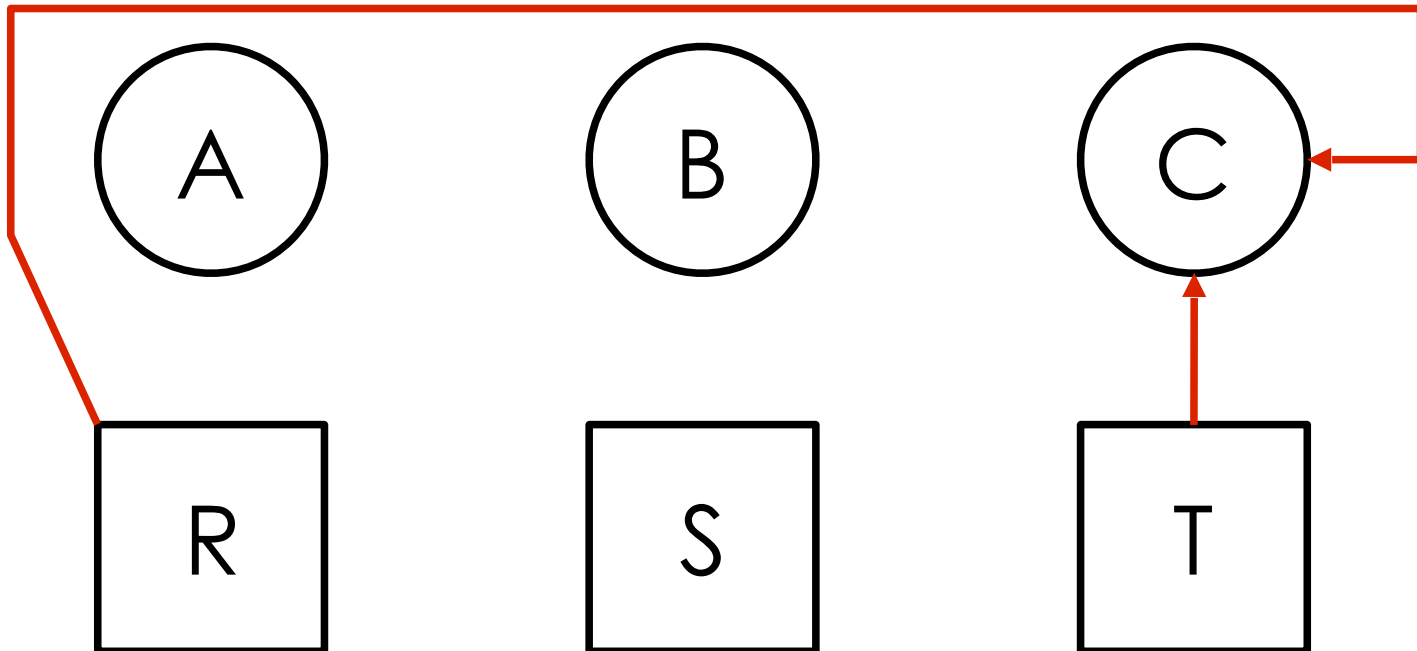
1. A fordert R
2. C fordert T
3. A fordert S
4. C fordert R
5. A gibt frei R



Beispiel (2)



1. A fordert R
2. C fordert T
3. A fordert S
4. C fordert R
5. A gibt frei R
6. A gibt frei S



Charakterisierende Bedingungen

Notwendige und hinreichende Bedingungen für das Auftreten von Verklemmungen bei Ein-Exemplar-BM
COFFMAN (1971)

Notwendig ist jede einzelne der Bedingungen:

1. gegenseitiger Ausschluss bei Benutzung von BM
2. Nachfordern von BM
3. keine Verdrängung (Entzug der BM) möglich
4. zyklische Wartesituation

Hinreichend ist:

Die Konjunktion der obigen vier Bedingungen.

Wegweiser

Begriff, Beispiele und Modellierung

Maßnahmen

Vermeiden per Konstruktion
Entdecken und Beseitigen

Ausschließen der notwendigen Bedingungen

1. Gegenseitiger Ausschluss (Mutual Exclusion)
z. B. durch asynchrone Bearbeitung von Aufträgen
2. Nachfordern
alle BM müssen bei Start angefordert werden,
Freigabe aller BM bei Neuforderung
Probleme: Dynamik, BM-Auslastung
3. Nicht-Entziehbarkeit
bei logischen Betriebsmitteln (Transaktionen)
4. Zyklische Wartebedingung
sorgfältige Betriebsmittelzuteilung (nächste Folien)

Sorgfältige Zuteilung: globale Reihenfolge

- Anforderung der BM in festgelegter Reihenfolge
- damit ist ein Zyklus ausgeschlossen
- zahlreiche Einzelsituationen bei Konstruktion komplexer paralleler Systeme
- Implementierungen in Sprachumgebungen (C++ **std::lock**)
- aber: nicht verträglich mit Funktionsaufrufen

Sorgfältige Zuteilung: Banker's Algorithm

Bank-Algorithmus für 1 Betriebsmittel-Typ (HABERMANN 1969)

Gegeben:

- Thread-System T
- maximale Anzahl G von BM-Exemplaren
- maximale Anzahl KP von BM-Exemplaren, die Thread $T_i \in T$ im Laufe seiner Existenz benötigt mit $KP \leq G$

Zustand:

- Vektor der momentan den Threads zugeteilten BM-Exemplare

Sicherer Zustand

Ein Zustand heißt **sicher**, wenn

- keine Verklemmung vorliegt
- es eine Reihenfolge der Erfüllung aller Anforderungen gibt, ohne daß eine Verklemmung auftritt.

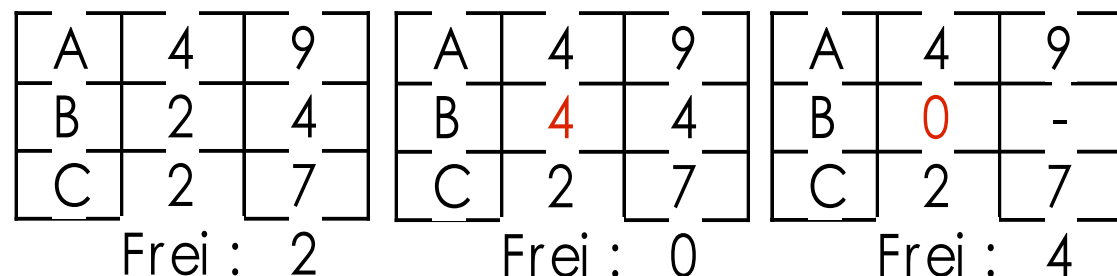
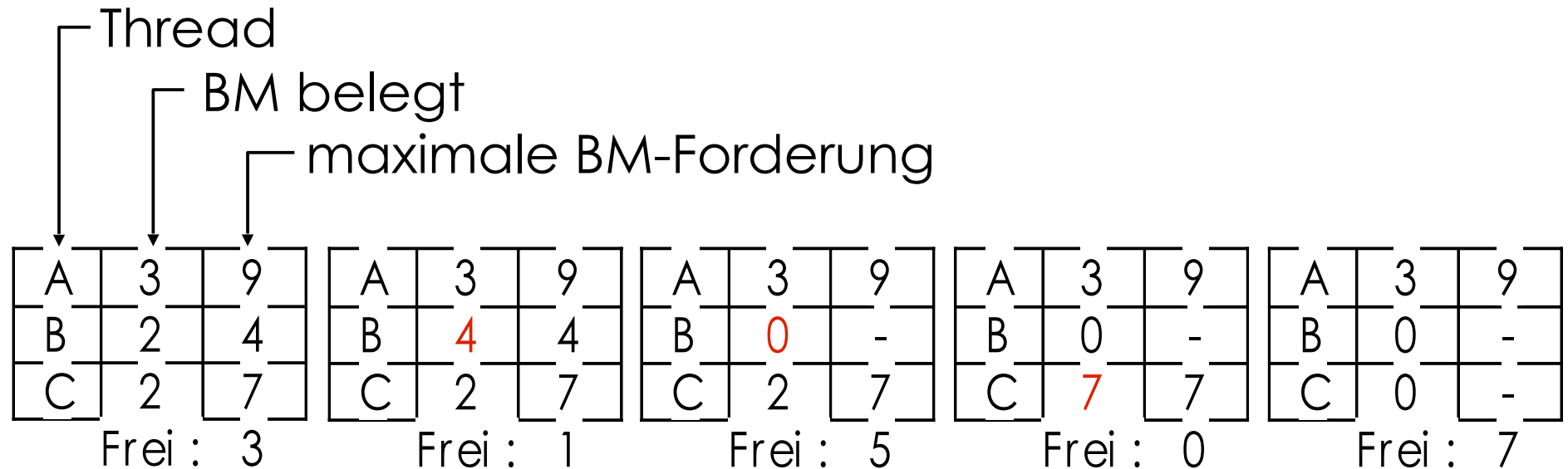
Andernfalls heißt der Zustand **unsicher**.

Idee des Bank-Algorithmus:

- Zuteilung nur vornehmen, wenn Nachfolgezustand immer noch sicher ist, andernfalls muss Thread warten.

Beispiel (nach Tanenbaum)

→ 10 Betriebsmittel-Exemplare eines Typs verfügbar



Bank-Algorithmus – Probleme

- Komplexität: $O(n^2)$, n : Anzahl der Threads
- Test bei jeder Anforderung nötig
- Mehrere BM-Typen: mehrere „Währungen“
- Basis: maximale Anzahl genutzter BM-Exemplare
 - Kenntnis
 - Eintreten dieser Situation („pessimistischer Algorithmus“)
- Blockierdauer nicht kalkulierbar

Entdecken (und Beseitigen)

Symptom: Operation dauert wesentlich länger als erwartet

- z.B. timeout als zusätzlicher Parameter blockierender Operationen

Diagnose: Feststellen ob zyklische Wartebedingung vorliegt

- `int sem_wait(sem_t *s)`
- **EDEADLK** als Rückgabewert bei Gefahr eines Deadlocks

Therapie: Verdrängung von Betriebsmitteln

- je nach Betriebsmittel sehr aufwendig

Entdecken (und Beseitigen): Diagnose

Thread

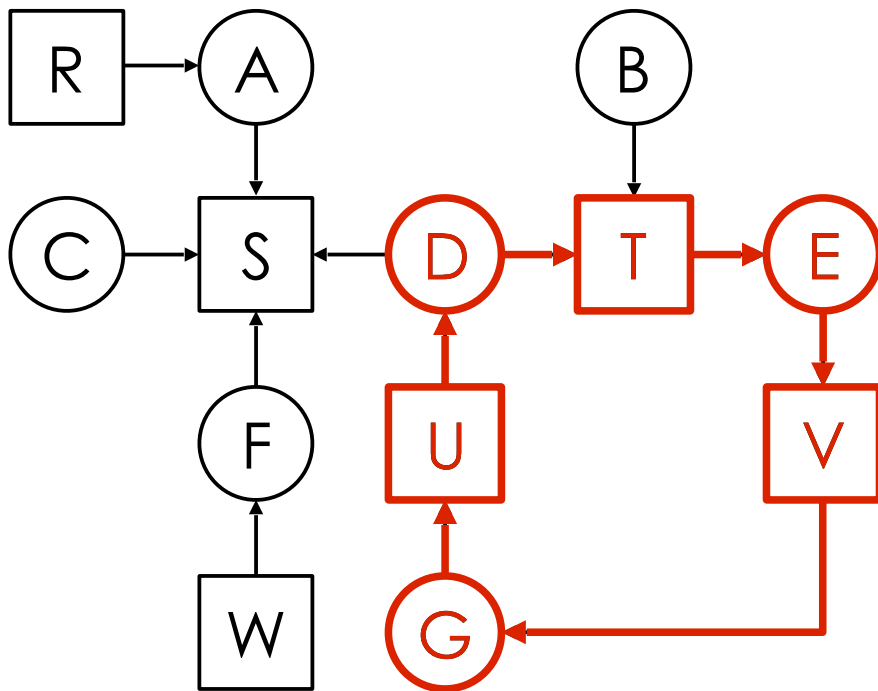
A
B
C
D
E
F
G

belegt

R
nichts
nichts
U
T
W
V

fordert

S
T
S
S und T
V
S
U



(Entdecken und) Beseitigen: Therapie

Verdrängen

- „von Hand“
- Entfernen von Prozessen

Zurücksetzen

- regelmäßiges Erstellen von Rücksetzpunkten
- Zurücksetzen vor die verursachende BM-Anforderung
- Außenwelt-Problem
(Terminal, Dateisystem, Netzwerk, ...)
- Transaktionen

Transaktionen

Zwei-Phasen-Technik (two-phase locking)

- in Datenbanken, Transaktionssysteme
- Aufsammeln der Locks, BM nutzen und freigeben am Ende der Transaktion
- falls ein Lock nicht erworben werden kann, alle erworbenen BM freigeben und auf alten Zustand zurücksetzen

Verwandte Begriffe

Verhungern / Starvation

BM werden nach Gesichtspunkten vergeben, die im Normalfall gut funktionieren, aber im Einzelfall zum „Verhungern“ eines Threads führen können

Beispiel Scheduling: shortest job next

Livelock

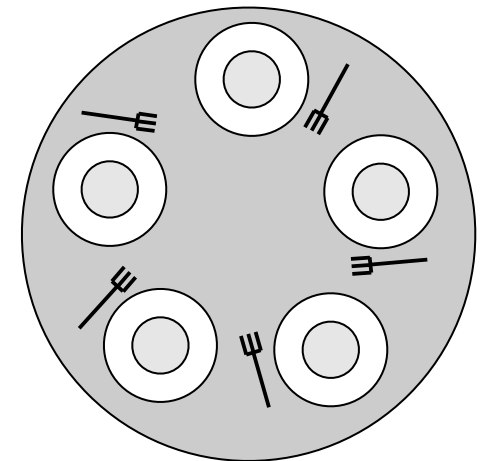
Threads sind aus Betriebssystemersicht nicht blockiert, machen kollektiv aber dennoch keinen Fortschritt.

Distributed Deadlock

Verklemmung über mehrere Rechner in einem verteilten System.

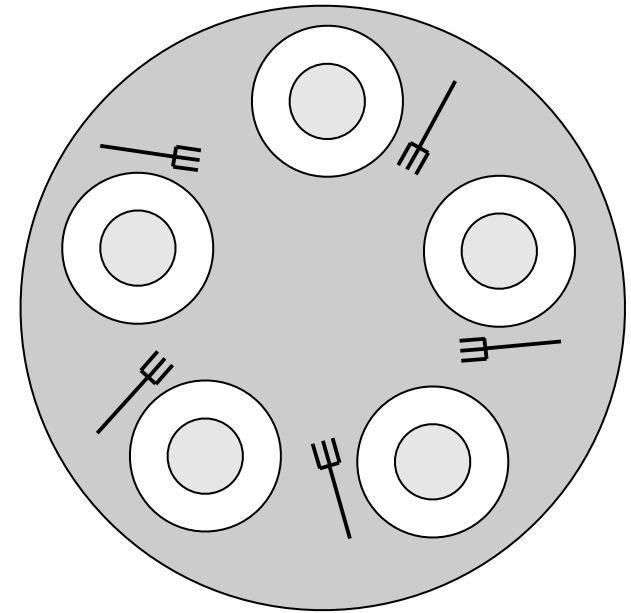
Dining Philosophers

- 5 Philosophen sitzen um einen Tisch, denken und essen Spaghetti
- zum Essen braucht jeder zwei Gabeln, es gibt aber insgesamt nur 5
- Problem: kein Philosoph soll verhungern



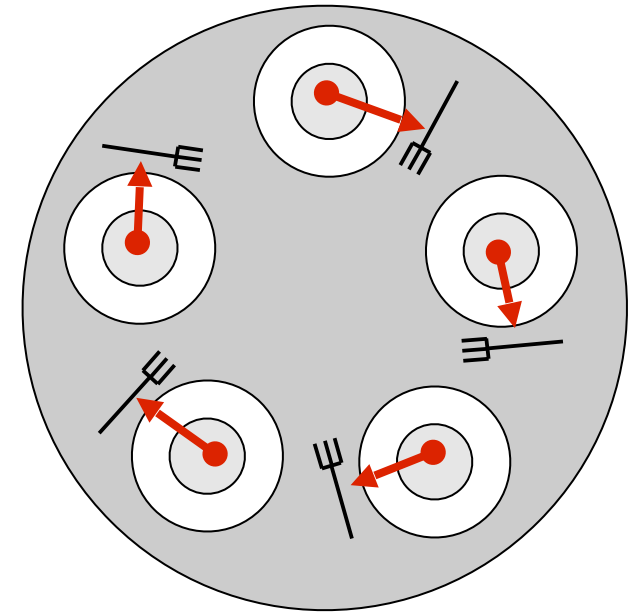
Die offensichtliche Lösung

```
void Philosoph(int i) {  
    Denken();  
    NimmGabel(i);  
    NimmGabel((i+1)%5);  
    Essen();  
    LegeZurückGabel(i);  
    LegeZurückGabel((i+1)%5);  
}
```



Die offensichtliche (aber falsche) Lösung

```
void Philosoph(int i) {  
    Denken();  
    NimmGabel(i);  
    NimmGabel((i+1)%5);  
    Essen();  
    LegeZurückGabel(i);  
    LegeZurückGabel((i+1)%5);  
}
```



- kann zu Verklemmungen/Deadlocks führen (alle Philosophen nehmen gleichzeitig die linken Gabeln)
- zwei verbündete Phil. können einen dritten aushungern

Zusammenfassung Verklemmungen

- Modellierung mit
 - PETRI-Netzen
 - Prozessfortschritts-Diagrammen
 - Betriebsmittel-Zuteilungsgraph
- 4 notwendige Bedingungen, Konjunktion ist hinreichend
- Vermeiden per Konstruktion: Reihenfolge für Locks
- Wenn das zu aufwendig ist: Erkennen und Beseitigen