

Betriebssysteme 1

SS 2020

Prof. Dr.-Ing. Hans-Georg Eßer
Fachhochschule Südwestfalen

Foliensatz D:

- Deadlocks

V2.0, 2020/05/22

Deadlocks – Gliederung

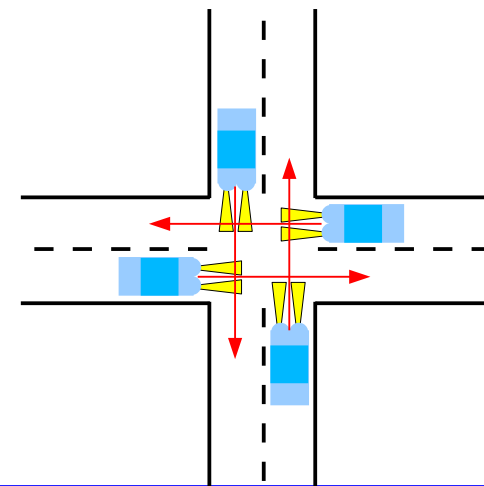
- Einführung
- Ressourcen-Typen
- Hinreichende und notwendige Deadlock-Bedingungen
- Deadlock-Erkennung und -Behebung
- Deadlock-Vermeidung (avoidance): Banker-Algorithmus
- Deadlock-Verhinderung (prevention)

Was ist ein Deadlock?

- Eine Menge von Threads (oder Prozessen) ist in einer **Deadlock-Situation**, wenn:
 - jeder Thread auf eine Ressource wartet, die von einem anderen Thread blockiert wird
 - keine der Ressourcen freigegeben werden kann, weil der haltende Thread (indem er selbst wartet) blockiert ist
- In einer Deadlock-Situation werden also die Threads dauerhaft verharren
- Deadlocks sind unbedingt zu vermeiden

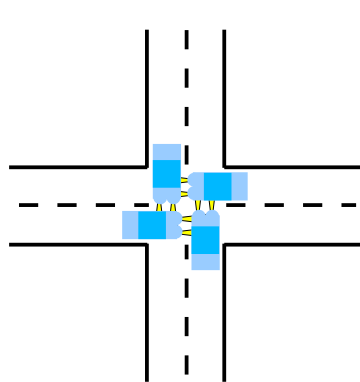
Deadlock: Rechts vor Links (1)

- Der Klassiker: Rechts-vor-Links-Kreuzung

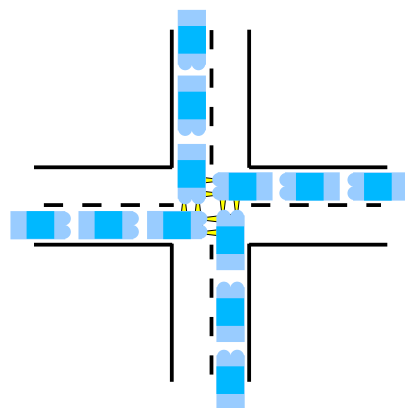


Wer darf fahren?
Potenzieller Deadlock

Deadlock: Rechts vor Links (2)

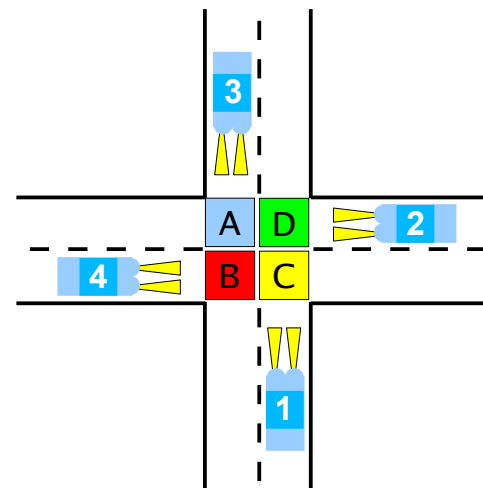


Deadlock, aber behebbar:
eines oder mehrere Autos
können zurücksetzen



Deadlock, nicht behebbar:
beteiligte Autos können nicht
zurücksetzen

Deadlock: Rechts vor Links (3)

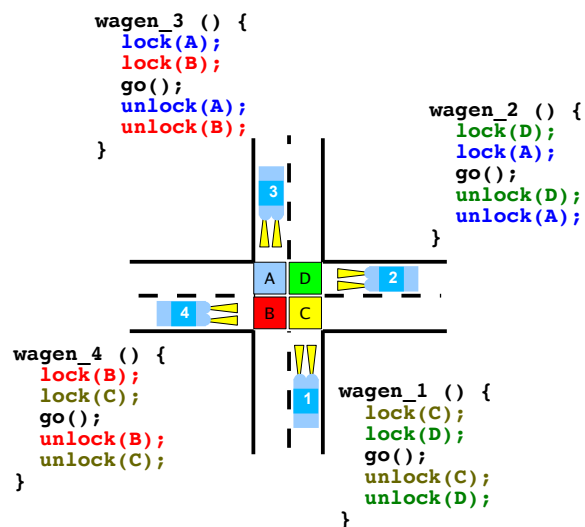


Analyse:

Kreuzungsbereich besteht
aus vier Quadranten
A, B, C, D

Wagen 1 benötigt C, D
Wagen 2 benötigt D, A
Wagen 3 benötigt A, B
Wagen 4 benötigt B, C

Deadlock: Rechts vor Links (4)



Problematische Reihenfolge:

w1: lock(C)
w2: lock(D)
w3: lock(A)
w4: lock(B)
w1: lock(D) ← blockiert
w2: lock(A) ← blockiert
w3: lock(B) ← blockiert
w4: lock(C) ← blockiert

Deadlock: kleinstes Beispiel (1)

- Zwei Locks A und B
 - z. B. A = Scanner, B = Drucker,
Prozesse P, Q wollen beide eine Kopie erstellen
- Locking in verschiedenen Reihenfolgen

Prozess P

```
lock (A);
lock (B);

/* krit. Bereich */

unlock (A);
unlock (B);
```

Prozess Q

```
lock (B);
lock (A);

/* krit. Bereich */

unlock (B);
unlock (A);
```

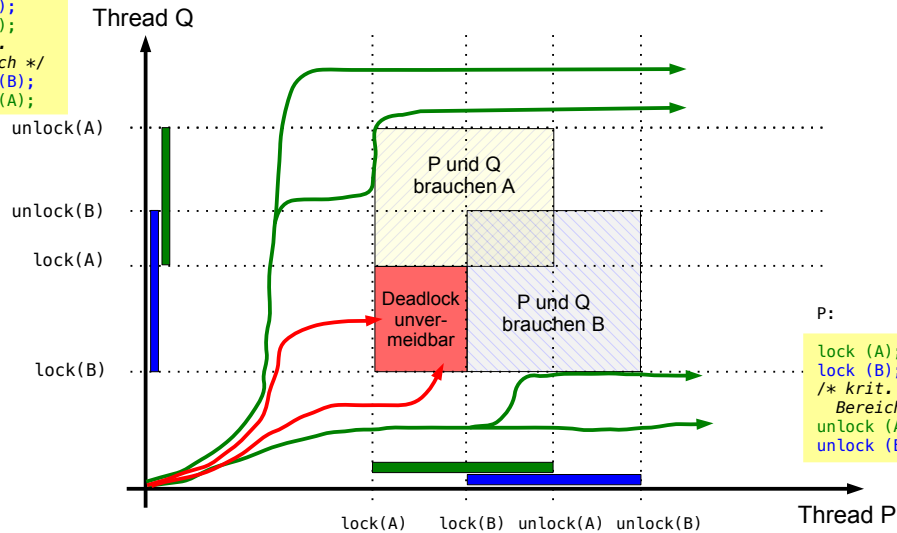
Problematische Reihenfolge:

```
P: lock(A)
Q: lock(B)
P: lock(B) ← blockiert
Q: lock(A) ← blockiert
```

Deadlock: kleinstes Beispiel (2)

Q:

```
lock (B);
lock (A);
/* krit.
Bereich */
unlock (B);
unlock (A);
```



Deadlock: kleinstes Beispiel (3)

- Problem beheben:
P benötigt die Locks nicht gleichzeitig

Prozess P

```
lock (A);
/* krit. Bereich */
unlock (A);
```

Prozess Q

```
lock (B);
lock (A);
/* krit. Bereich */
unlock (B);
unlock (A);
```

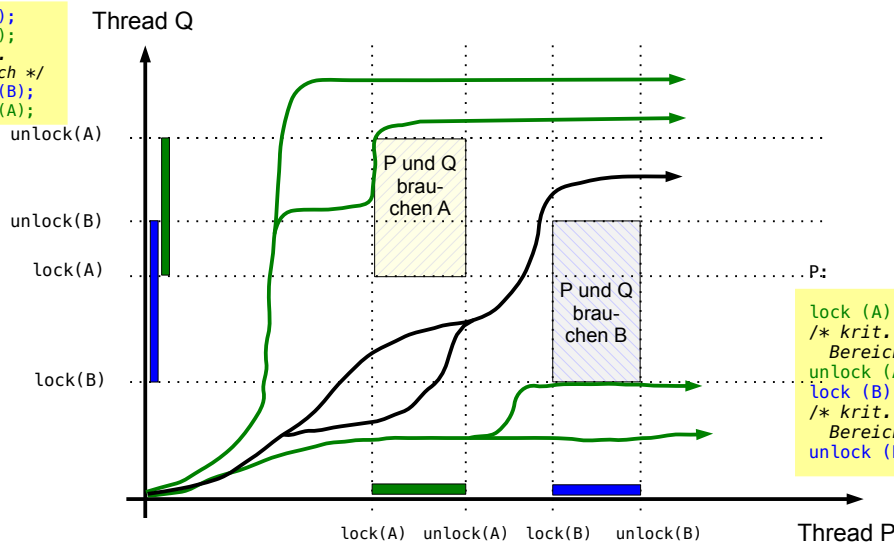
Jetzt kann kein Deadlock mehr auftreten

- Andere Lösung: P und Q fordern A, B in gleicher Reihenfolge an

Deadlock: kleinstes Beispiel (4)

Q:

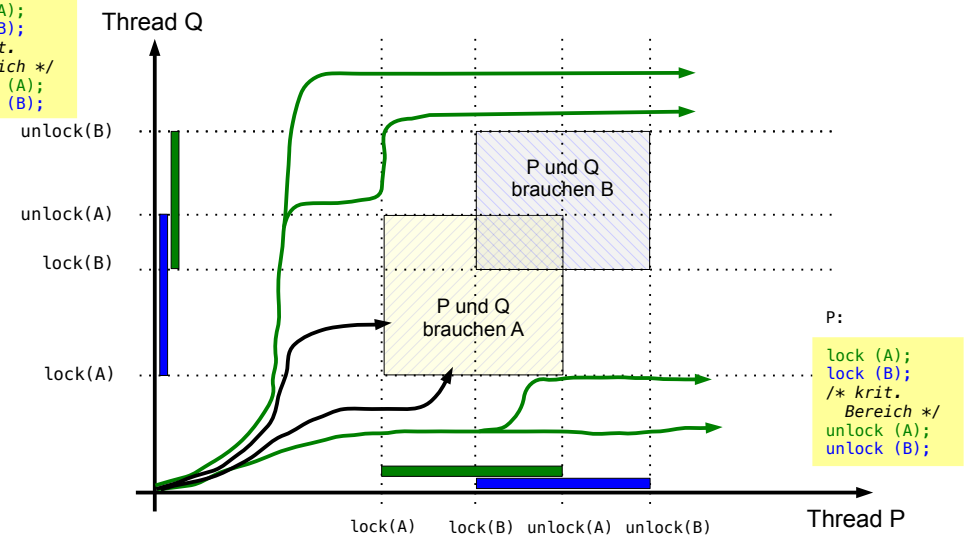
```
lock (B);
lock (A);
/* krit.
Bereich */
unlock (B);
unlock (A);
```



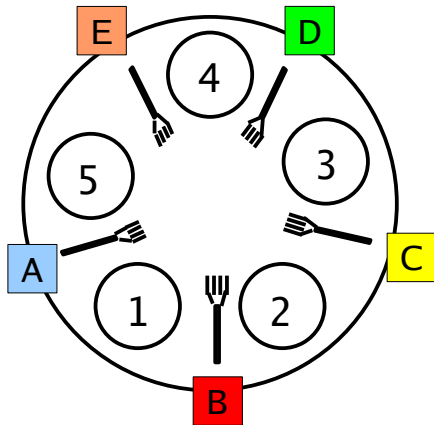
Deadlock: kleinstes Beispiel (5)

Q:

```
lock (A);
lock (B);
/* krit.
Bereich */
unlock (A);
unlock (B);
```



Fünf-Philosophen-Problem



Philosoph 1 braucht Gabeln A, B
Philosoph 2 braucht Gabeln B, C
Philosoph 3 braucht Gabeln C, D
Philosoph 4 braucht Gabeln D, E
Philosoph 5 braucht Gabeln E, A

Problematische Reihenfolge:

p1: lock (B)
p2: lock (C)
p3: lock (D)
p4: lock (E)
p5: lock (A)
p1: lock (A) ← blockiert
p2: lock (B) ← blockiert
p3: lock (C) ← blockiert
p4: lock (D) ← blockiert
p5: lock (E) ← blockiert

Deadlock-Bedingungen (1)

1. Gegenseitiger Ausschluss (mutual exclusion)

Ressource ist exklusiv: Es kann stets nur ein Prozess darauf zugreifen

2. Hold and Wait (besitzen und warten)

Ein Prozess ist bereits im Besitz einer oder mehrerer Ressourcen, und er kann noch weitere anfordern

3. Ununterbrechbarkeit der Ressourcen

Die Ressource kann nicht durch das Betriebssystem entzogen werden

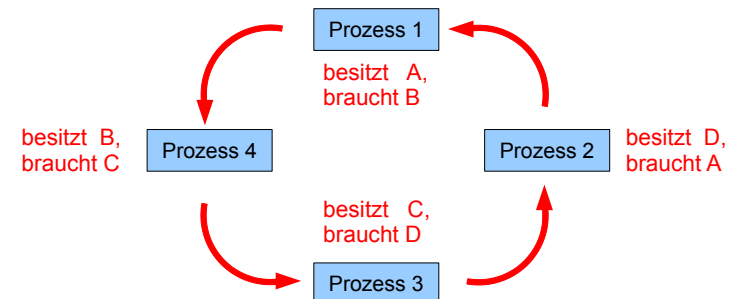
Deadlock-Bedingungen (2)

- (1) bis (3) sind **notwendige** Bedingungen für einen Deadlock
- (1) bis (3) sind aber auch „wünschenswerte“ Eigenschaften eines Betriebssystems, denn:
 - gegenseitiger Ausschluss ist nötig für korrekte Synchronisation
 - Hold & Wait ist nötig, wenn Prozesse exklusiven Zugriff auf mehrere Ressourcen benötigen
 - Bei manchen Betriebsmitteln ist eine Präemption prinzipiell nicht sinnvoll (z. B. DVD-Brenner, Streamer)

Deadlock-Bedingungen (3)

4. Zyklisches Warten

Man kann die Prozesse in einem Kreis anordnen, in dem jeder Prozess eine Ressource benötigt, die der folgende Prozess im Kreis belegt hat



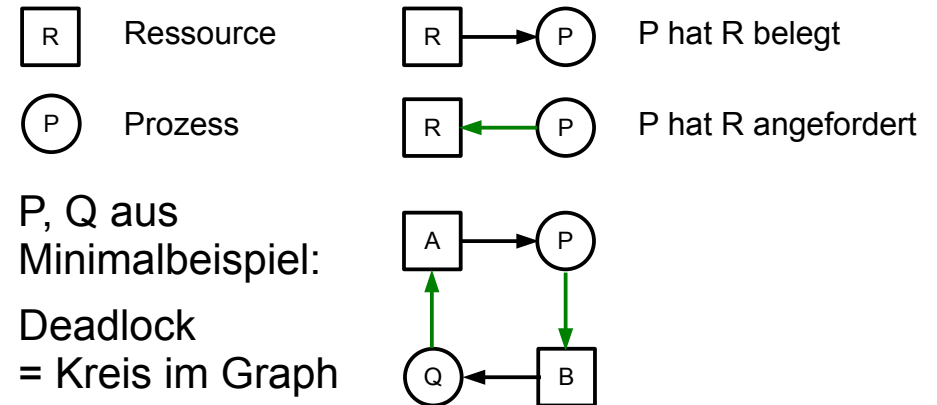
Deadlock-Bedingungen (4)

1. Gegenseitiger Ausschluss
2. Hold and Wait
3. Ununterbrechbarkeit der Ressourcen
4. Zyklisches Warten

- (1) bis (4) sind **notwendige und hinreichende** Bedingungen für einen Deadlock
- Das zyklische Warten (4) (und dessen Unauflösbarkeit) sind Konsequenzen aus (1) bis (3)
- (4) ist der erfolgversprechendste Ansatzpunkt, um Deadlocks aus dem Weg zu gehen

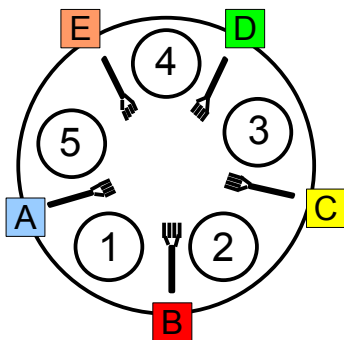
Ressourcen-Zuordnungs-Graph (1)

- Belegung und (noch unerfüllte) Anforderung grafisch darstellen:

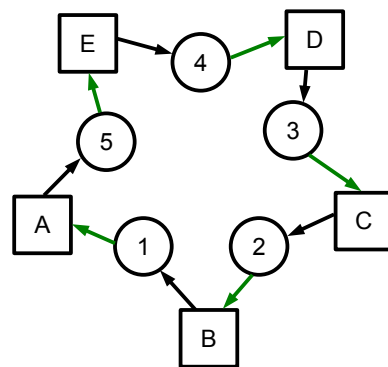


Ressourcen-Zuordnungs-Graph (2)

Philosophen-Beispiel

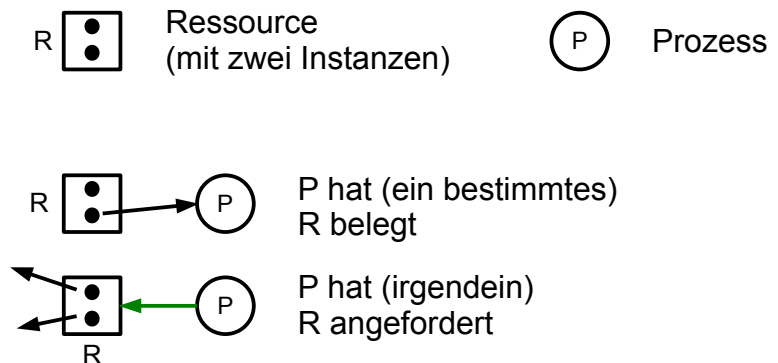


Situation, nachdem alle Philosophen ihre rechte Gabel aufgenommen haben



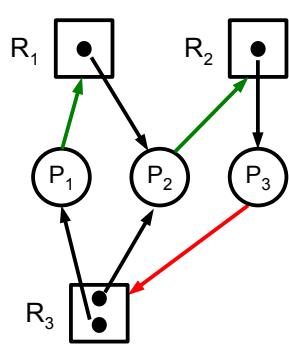
Ressourcen-Zuordnungs-Graph (3)

- Variante für Ressourcen, die mehrfach vorkommen können

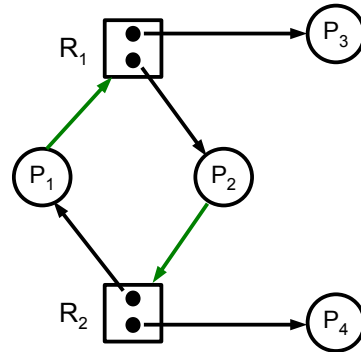


Ressourcen-Zuordnungs-Graph (4)

- Beispiele mit mehreren Instanzen



Mit roter Kante ($P_3 \rightarrow R_3$) gibt es einen Deadlock (ohne nicht)



Kreis, aber kein Deadlock – Bedingung ist nur **notwendig**, nicht hinreichend!

Mehrfach-Ressourcen: Matrixverfahren (1)

	Netzwerkkarte	Streamer	Scanner	DVD-Brenner	
$E = (1\ 3\ 2\ 4)$					Ressourcenvektor
$A = (0\ 0\ 1\ 2)$					Ressourcenrestvektor
$C = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 2 \\ 0 & 2 & 1 & 0 \end{pmatrix}$					Belegungsmatrix
$R = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 2 \end{pmatrix}$					Anforderungsmatrix
					Thread 1
					Thread 2
					Thread 3

Mehrfach-Ressourcen: Matrixverfahren (2)

1.

$$E = (1\ 3\ 2\ 4) \quad A = (0\ 0\ 1\ 2)$$

$$C = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 2 \\ 0 & 2 & 1 & 0 \end{pmatrix} \quad R = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 2 \end{pmatrix}$$

2.

$$E = (1\ 3\ 2\ 4) \quad A = (0\ 2\ 2\ 2)$$

$$C = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 2 \\ 0 & 2 & 1 & 0 \end{pmatrix} \quad R = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 2 \end{pmatrix}$$

3.

$$E = (1\ 3\ 2\ 4) \quad A = (1\ 2\ 2\ 4)$$

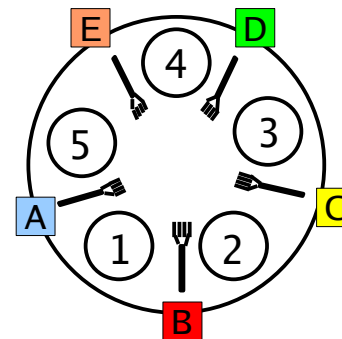
$$C = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 2 \\ 0 & 2 & 1 & 0 \end{pmatrix} \quad R = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 2 \end{pmatrix}$$

4.

$$E = (1\ 3\ 2\ 4) \quad A = (1\ 3\ 2\ 4)$$

$$C = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 2 \\ 0 & 2 & 1 & 0 \end{pmatrix} \quad R = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 2 \end{pmatrix}$$

Fünf-Philosophen-Problem



$$E = (1\ 1\ 1\ 1\ 1) \quad A = (0\ 0\ 0\ 0\ 0)$$

$$C = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad R = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$