Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

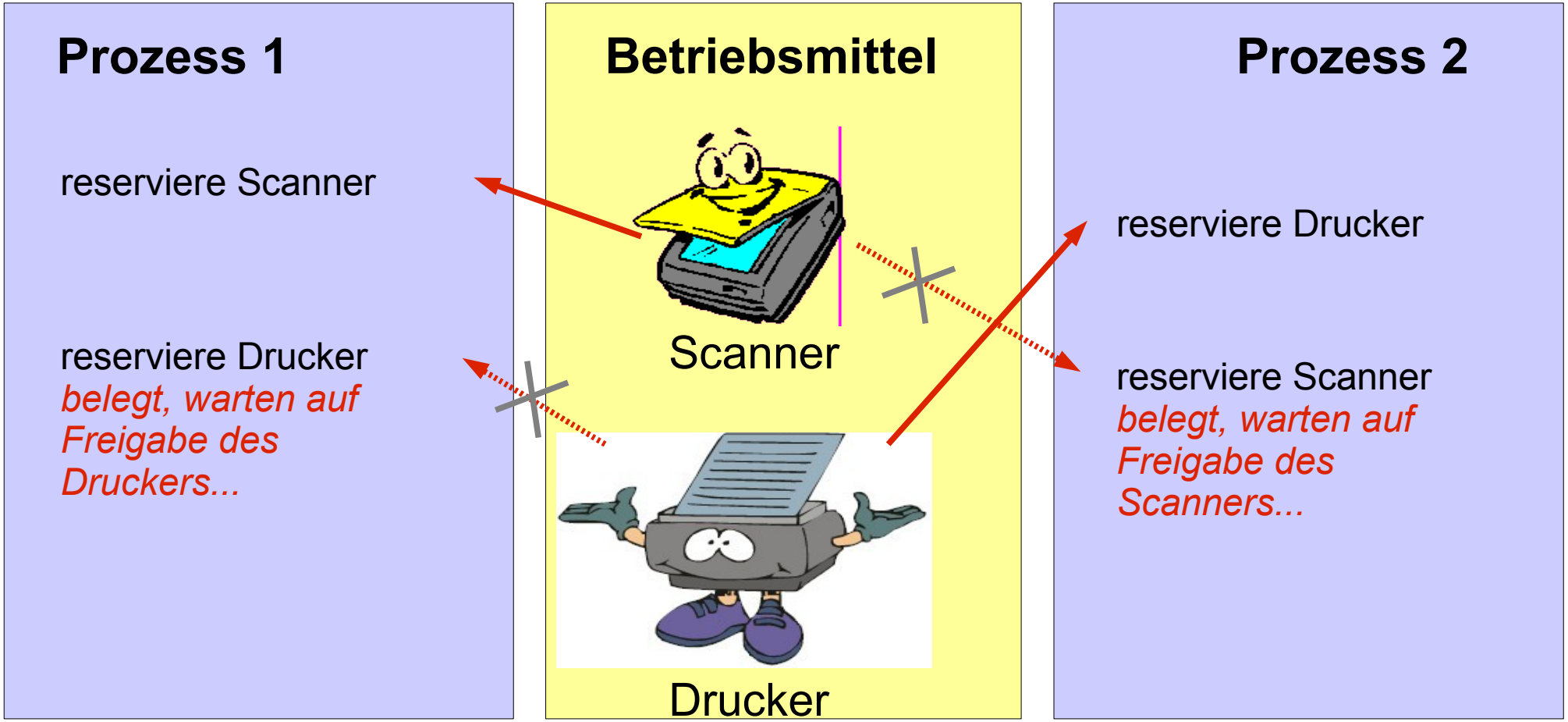
Kap. 7: Deadlocks

Motivation

- In diesem Kapitel wird ein grundlegendes Problem bei der Benutzung exklusiv benutzbarer Betriebsmittel behandelt, das sogenannte Deadlock-Problem.
- Ein Deadlock wird auch als Systemverklemmungszustand bezeichnet.
- Untersuchungen zu Deadlocks nehmen breiten Raum in der frühen Betriebssystem-Literatur ein.
- Ziel hier: Kennenlernen von Methoden zur Erkennung, Behebung, Vermeidung und Verhinderung von Deadlocks.

Hinführendes Beispiel

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim



- Beide Prozesse sind blockiert und bleiben es für immer
- Eine solche Situation heißt **Deadlock** (genauere Definition folgt)
- Deadlocks treten auch in vielen anderen Situationen auf, z.B. beim Sperren von Datensätzen in Datenbanken

Gliederung

Betriebsmittel

- Reservierbare Objekte (Objekte, auf die Zugriff erteilt werden kann) heißen **Betriebsmittel**.
- Diese können **Hard-** oder **Softwarekomponenten** sein:
  - CD-Brenner
  - Prozessor
  - ein Datensatz
  - eine Verwaltungsstruktur des Betriebssystems
  - …
- Ein Betriebsmittel(typ) kann in mehreren identischen Instanzen oder Einheiten vorliegen
- Ein Betriebsmittel ist **unterbrechbar**, wenn es einem Prozess ohne nachteilige Auswirkungen entzogen werden kann, ansonsten heißt es **ununterbrechbar**.
- Beispiele:
  - Arbeitsspeicher → unterbrechbar (Prozess aus-/einlagern)
  - Drucker, DVD-Brenner: ununterbrechbar

Betriebsmittel (2)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Schritte zur Benutzung eines Betriebsmittels:
  - Anfordern des BMs
  - Benutzen des BMs
  - Freigeben des BMs
- Falls BM angefordert wird aber nicht verfügbar ist, muss der anfordernde Prozess warten. Warten kann durch Blockieren oder durch Busy Waiting realisiert sein.
- Form einer BM-Anforderung ist in konkreten Systemen sehr unterschiedlich, u.U. auch BM-Typ-abhängig.
  Typisch: `open(bm)`.
- Bewilligung einer Betriebsmittelanforderung heißt auch (Betriebsmittel-)Zuteilung. Prozess wird durch Zuteilung Inhaber des Betriebsmittels.

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Def

- Definition:

Eine Menge von Prozessen befindet sich in einem Deadlock-Zustand, falls jeder Prozess der Menge auf ein Ereignis wartet, das nur ein anderer Prozess der Menge auslösen kann.

Man sagt auch:
Die Prozesse sind in einen Deadlock verstrickt.

Da alle Prozesse warten, kann keiner jemals ein Ereignis erzeugen, auf das einer der anderen wartet. Alle Prozesse warten also für immer.

Voraussetzungen für Deadlocks

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

Coffman (1971): Voraussetzungen für Deadlocks:

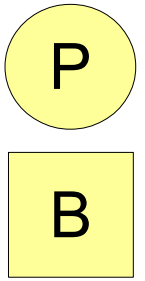
- **Wechselseitiger Ausschluß:**
  Jedes Betriebsmittel ist entweder frei
  oder genau einem Prozess zugeteilt
- **Belegungs-/Anforderungs-Bedingung (*„Hold-and-wait“*)**:
  Prozesse können zu bereits reservierten Betriebsmitteln noch weitere anfordern
- **Ununterbrechbarkeit:**
  Einmal einem Prozess zugeteilte Betriebsmittel können nicht wieder ohne dessen Zustimmung (Freigabe) entzogen werden.
- **Zyklisches Warten:**
  Es muss eine zyklische Kette von Prozessen geben, in der jeder Prozess auf ein Betriebsmittel wartet, das dem nächsten Prozess in der Kette gehört.

Alle vier Bedinungen gleichzeitig erfüllt => Ein Deadlock **ist möglich**
(falls eine nicht erfüllt ist => **kein Deadlock möglich**)

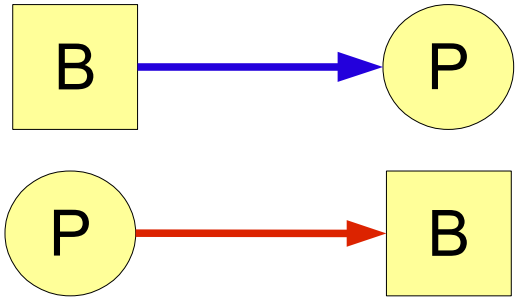
Belegungs-Anforderungs-Graphen

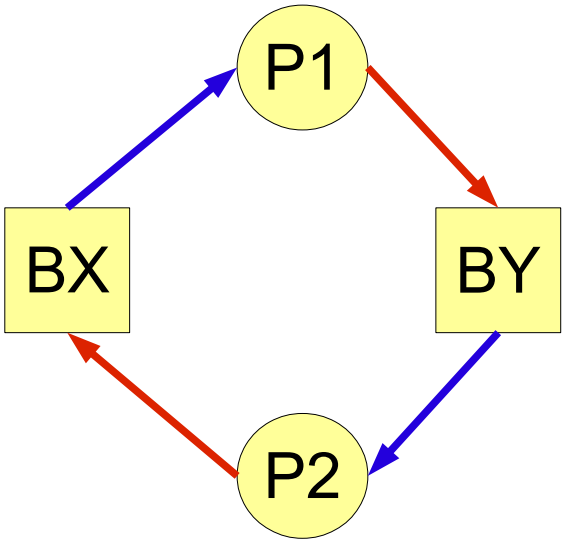
Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Graphische Darstellung der Beziehung von Prozessen zu Betriebsmitteln (Holt, 1972)
- Es gibt zwei Knotentypen:
  - Prozesse, repräsentiert durch Kreise
  - Betriebsmittel, repräsentiert durch Quadrate



- Pfeile:
  - P belegt B
  - P wartet auf B





- Zyklus im Graphen: Deadlock

Beispiel

- Gegeben:
  - drei Prozesse A, B, C und
  - drei Betriebsmittel R, S, T

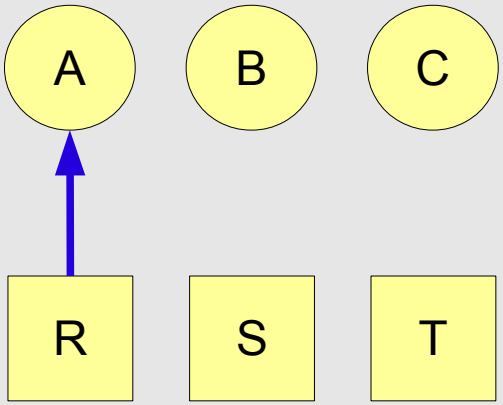
Prozess A	**Prozess B**	**Prozess C**
- Anforderung R	- Anforderung S	- Anforderung T
- Anforderung S	- Anforderung T	- Anforderung R
- Freigabe R	- Freigabe S	- Freigabe T
- Freigabe S	- Freigabe T	- Freigabe R

- Das Betriebssystem kann jeden (nicht blockierten) Prozess **jederzeit** ausführen
- Sequentielle Ausführung von A, B, C wäre unproblematisch (dann aber auch keine Nebenläufigkeit)
- Wie sieht es bei nebenläufiger Ausführung aus?

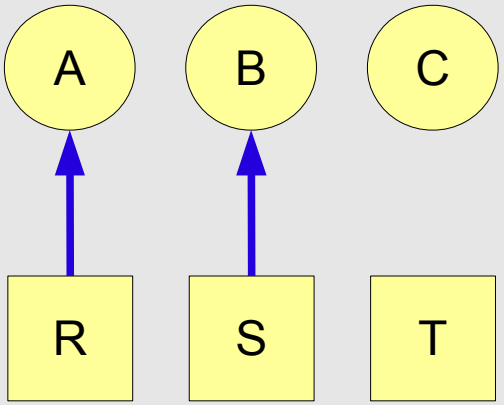
Ausführung I

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

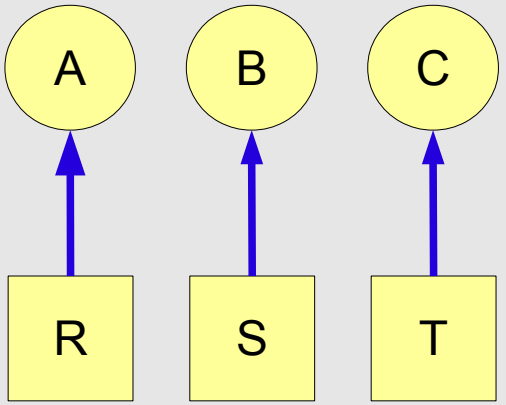
1. A fordert R an



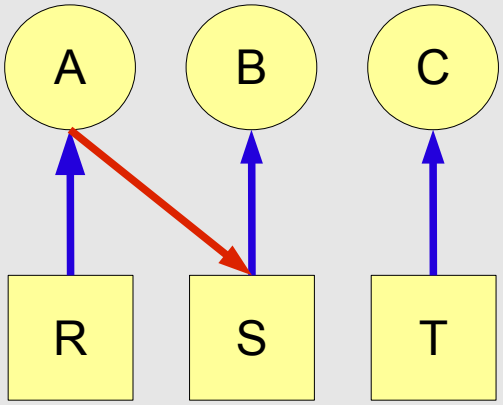
2. B fordert S an



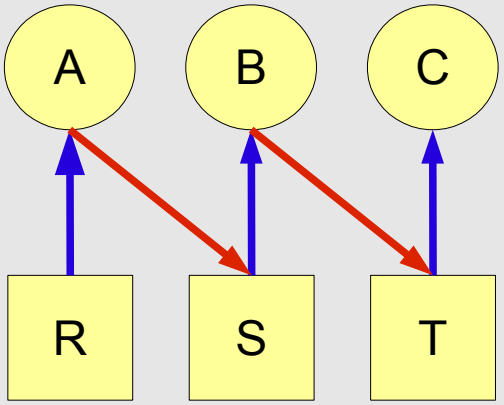
3. C fordert T an



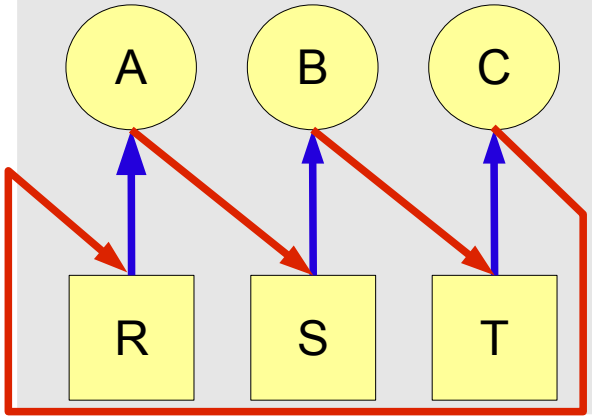
4. A fordert S an



5. B fordert T an



6. C fordert R an
Deadlock

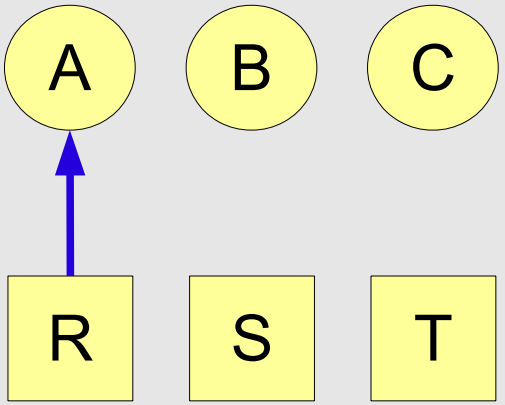


Ausführung II

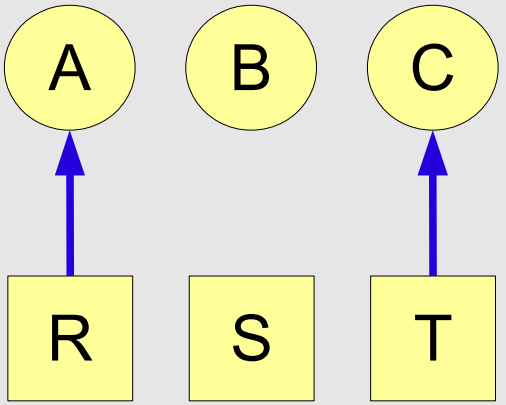
Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

(B wird zunächst suspendiert)

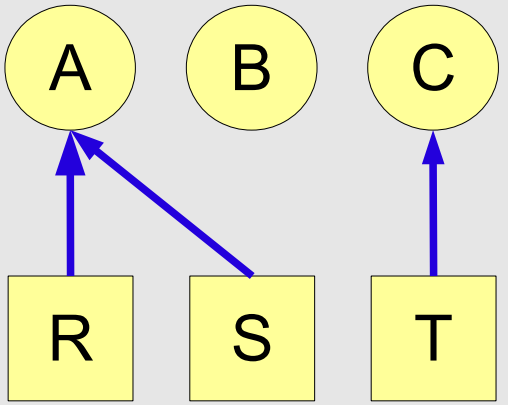
1. A fordert R an



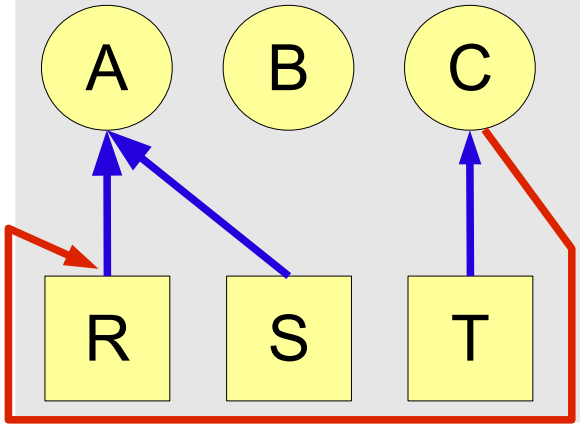
2. C fordert T an



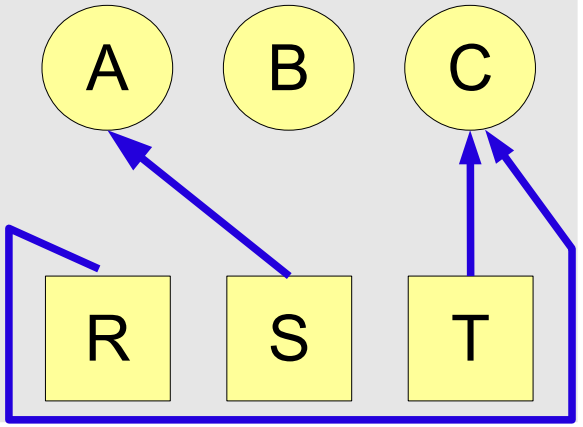
3. A fordert S an



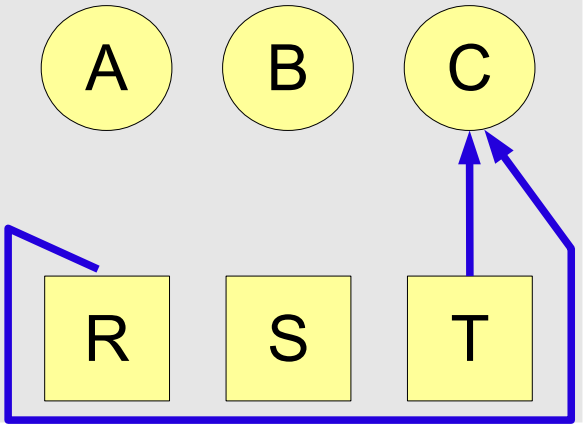
4. C fordert R an



5. A gibt R frei



6. A gibt S frei
kein Deadlock



Verfahren zur DL-Behandlung

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Mit Betriebsmittelzuteilungsgraphen („*Belegungs/Anforderungs-Graphen*“) lassen sich Deadlocks erkennen (→Zyklus im Graph)
- Wie weiter verfahren?
  - **Ignorieren** („Vogel-Strauß-Verfahren“)
  - Deadlocks **erkennen** und **beheben**
  - **Verhinderung** durch Planung der Betriebsmittelzuordnung (*Deadlock Avoidance*)
  - **Vermeidung** durch Nichterfüllung (mindestens) einer der vier Voraussetzungen für Deadlocks (*Deadlock Prevention*)
- Diese Strategien werden im folgenden untersucht

7.3 Ignorieren des Problems

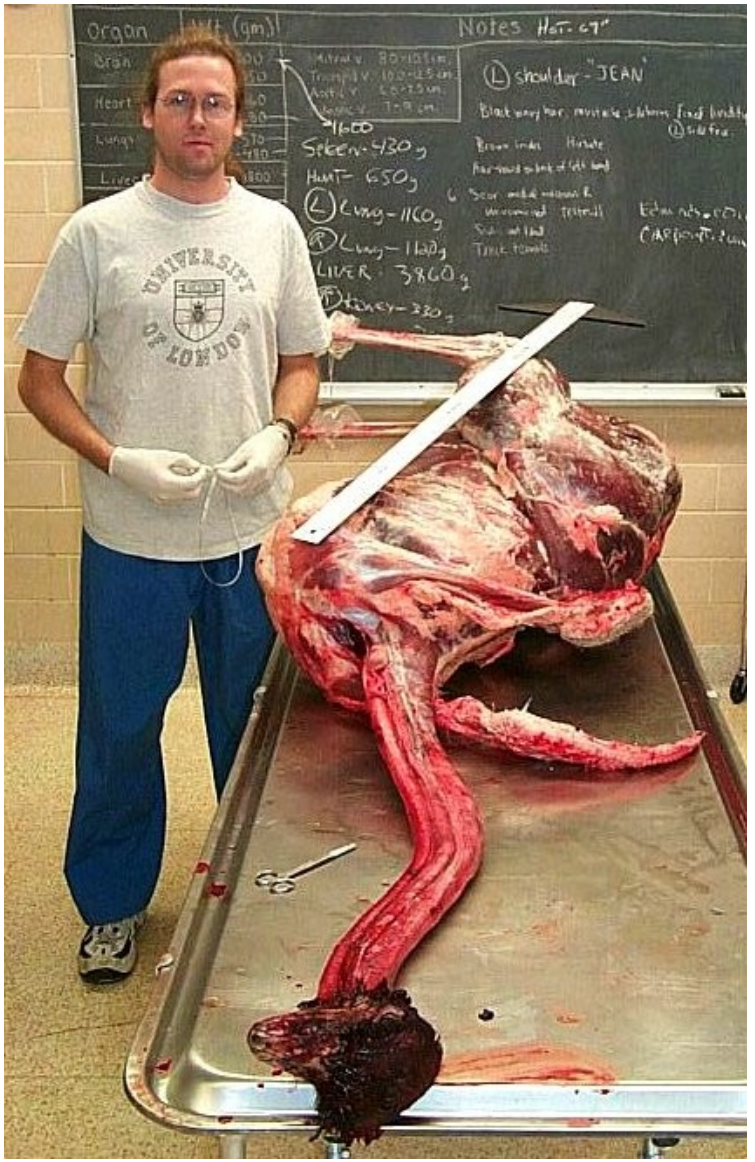
Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- „Vogel-Strauß-Algorithmus“
- Ausdruck optimistischer Lebenshaltung:

  „Deadlocks kommen in der Praxis sowieso nie vor“

- ...warum also dann Aufwand in ihre Vermeidung stecken?
- **Beispiel:**
  - UNIX-System mit z.B. 100 Einträge großer Prozesstabelle
  - 10 Programme versuchen gleichzeitig, je 12 Kindprozesse zu erzeugen
  - Deadlock nach 90 erfolgreichen `fork()`-Aufrufen (wenn keiner der Prozesse aufgibt)
- Ähnliche Beispiele sind mit anderen begrenzt großen Systemtabellen möglich (z.B. inode-Tabelle)

...manchmal nicht so gut



7.4 Deadlock-Erkennung und Behebung

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Engl.: Deadlock Detection and Resolution (Recovery)
- Vorgehensweise: Das Auftreten von Deadlocks wird vom Betriebssystem nicht verhindert. Es wird versucht, Deadlocks zu erkennen und anschließend zu beheben.
- Betrachtet werden im folgenden:
  1. Deadlock-Erkennung mit einem Betriebsmittel je Klasse (Einfacher Fall)
  2. Deadlock-Erkennung mit mehreren Betriebsmitteln je Klasse (Allgemeiner Fall)
  3. Verfahren zur Deadlock-Behebung

7.4.1 Deadlocks erkennen (Einfacher Fall)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Vereinfachende Annahme: **Ein Betriebsmittel** je Betriebsmitteltyp
- **Vorgehen**:
  - erzeuge Belegungs-/Anforderungs-Graph
  - suche nach Zyklen
  - falls ein Zyklus gefunden wurde: Deadlock beheben (s.u.)
- **Wann** wird die Untersuchung durchgeführt?
  - bei **jeder** Betriebsmittelanforderung?
  - in **regelmäßigen** Zeitabständen?
  - wenn „**Verdacht**“ auf Deadlock besteht
    (z.B. Abfall der CPU-Auslastung unter eine Grenze)

Beispiele: Sicher?

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

4 Prozesse, ein Betriebsmitteltyp (10 Stück vorhanden)

verfügbar: 10

Proz.	hat	max.
A	0	6
B	0	5
C	0	4
D	0	7

sicher!

z.B. sequentielle Ausführung von A, B, C, D in beliebiger Reihenfolge ist möglich

verfügbar: 2

Proz.	hat	max.
A	1	6
B	1	5
C	2	4
D	4	7

sicher!

C ist ausführbar, (→dann 4 verfügbar) dann D, B, A möglich.

verfügbar: 1

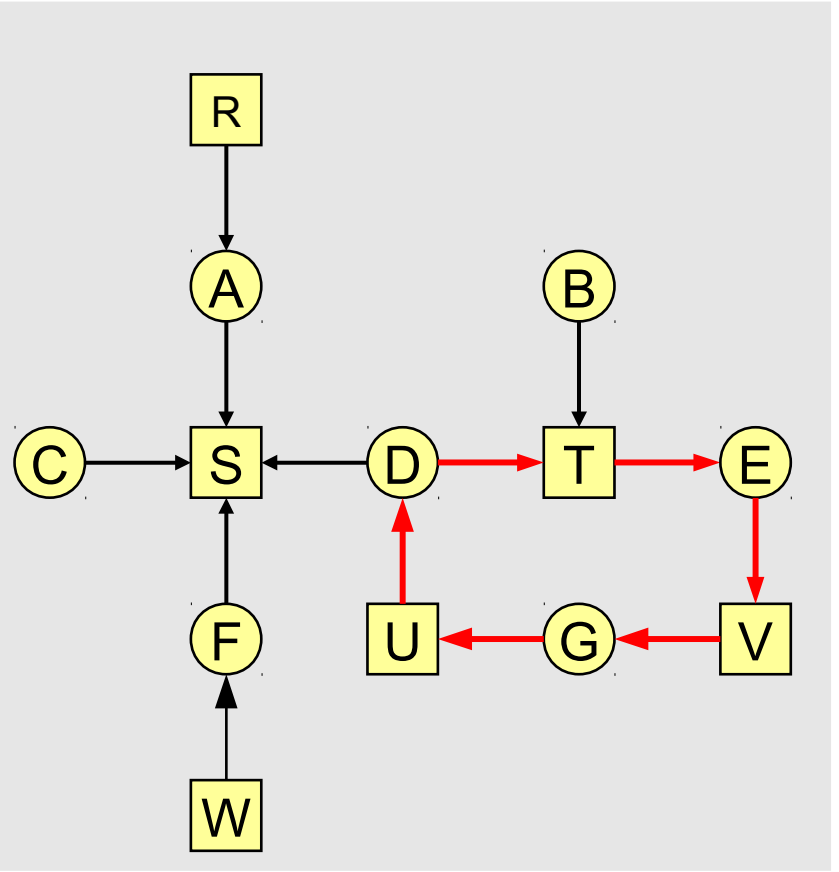
Proz.	hat	max.
A	1	6
B	2	5
C	2	4
D	4	7

unsicher!

Differenz *max* - *hat* immer > *verfügbar*. Deadlock, sobald irgendein Prozess auf sein Maximum zugeht

Beispiel

1. A belegt R und fordert S an.
2. B fordert T an.
3. C fordert S an.
4. D belegt U und fordert S und T an.
5. E belegt T und fordert V an.
6. F belegt W und fordert S an.
7. G belegt V und fordert U an.



Deadlocks erkennen

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Erweiterung: **Mehrere** (E_i-viele) Betriebsmittel je Betriebsmitteltyp i (z.B. mehrere Drucker)
- Prozesse $P_1,...,P_n$

Betriebsmittelvektor $E=(E_1, E_2, ..., E_m)$ - Gesamtzahl der BM je Typ i

Verfügbarkeitsvektor $A=(A_1, A_2, ..., A_m)$ - noch verfügbare BM je Typ i

Belegungsmatrix C: Zeile j gibt BM-Belegung durch Prozess j an ("Prozess j belegt C_{jk} Einheiten von BM k")

$$\begin{pmatrix} C_{11} & C_{12} & \dots & C_{1m} \\ C_{21} & C_{22} & \dots & C_{2m} \\ \dots & \dots & \dots & \dots \\ C_{n1} & C_{n2} & \dots & C_{nm} \end{pmatrix}$$

Anforderungsmatrix R: Zeile j gibt BM-Anforderungen durch Prozess j an ("Prozess j fordert R_{jk} Einheiten von BM k")

$$\begin{pmatrix} R_{11} & R_{12} & \dots & R_{1m} \\ R_{21} & R_{22} & \dots & R_{2m} \\ \dots & \dots & \dots & \dots \\ R_{n1} & R_{n2} & \dots & R_{nm} \end{pmatrix}$$

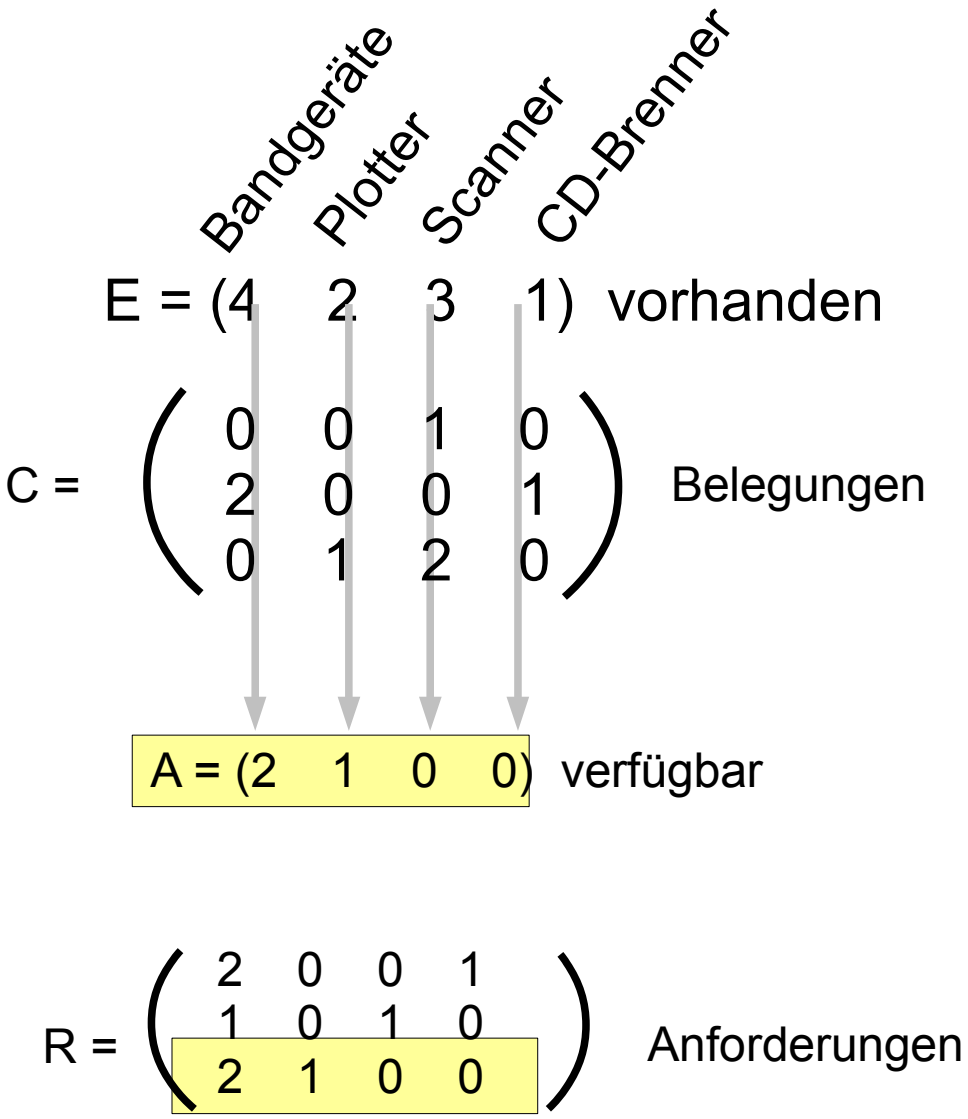
Erkennungsalgorithmus

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Zu Beginn sind alle Prozesse aus P unmarkiert (Markierung heißt, dass der Prozess in keinem DL steckt)
- Suche einen Prozess, der ungehindert durchlaufen kann, also einen unmarkierten Prozess P_i, dessen Zeile in der Anforderungsmatrix-Zeile R_i (komponentenweise) kleiner als oder gleich dem Verfügbarkeitsvektor A ist
- Kein passendes P_i gefunden? Dann **Ende**
- Gefunden? Dann kann P_i durchlaufen, gibt danach seine belegten Betriebsmittel zurück: $A = A + C_i$, wird markiert und es geht beim nächsten unmarkierten Prozess weiter
- Beim Ende des Verfahrens sind alle unmarkierten Prozesse an einem Deadlock beteiligt.

Beispiel

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim



Ausführbar ist zunächst nur **P3**

Freigabe von C_3 = (0 1 2 0)
=> A = (2 1 0 0) + (0 1 2 0)
=> A = (2 2 2 0)

Nun ausführbar: **P2**
(benötigt R_2 = (1 0 1 0))

Freigabe von C_2 = (2 0 0 1)
Danach: A = (4 2 2 1)

Schließlich auch **P1** ausführbar
A = (4 2 3 1)

Alle Prozesse markiert,
kein Deadlock aufgetreten.

7.4.3 Beheben von Deadlocks

Wie kann man auf erkannte Deadlocks reagieren?

- **Prozessunterbrechung**
  - Betriebsmittel zeitweise entziehen, anderem Prozess bereitstellen und dann zurückgeben
  - Kann je nach Betriebsmittel schwer oder nicht möglich sein
- **Teilweise Wiederholung (Rollback)**
  - System sichert regelmäßig Prozesszustände (Checkpoints)
  - Dadurch ist Abbruch und späteres Wiederaufsetzen möglich
  - Arbeit seit letztem Checkpoint geht beim Rücksetzen verloren und wird beim Neuaufsetzen wiederholt (ungünstig z.B. bei seit Checkpoint ausgedruckten Seiten)
  - Beispiel: Transaction Abort bei Datenbanken
- **Prozessabbruch**
  - Härteste, aber auch einfachste Maßnahme
  - Nach Möglichkeit Prozesse auswählen, die relativ problemlos neu gestartet werden können (z.B. Compilierung)

7.5 Verhindern von Deadlocks

- Bisher: Erkennung von Deadlocks, gegebenenfalls „drastische“ Maßnahmen zur Auflösung
- Annahme bisher: Prozesse fordern alle Betriebsmittel „auf ein Mal“ an (vgl. 7.4.2).
- In den meisten praktischen Fällen werden BM jedoch nacheinander angefordert
- Betriebssystem muss dann dynamisch über Zuteilung entscheiden

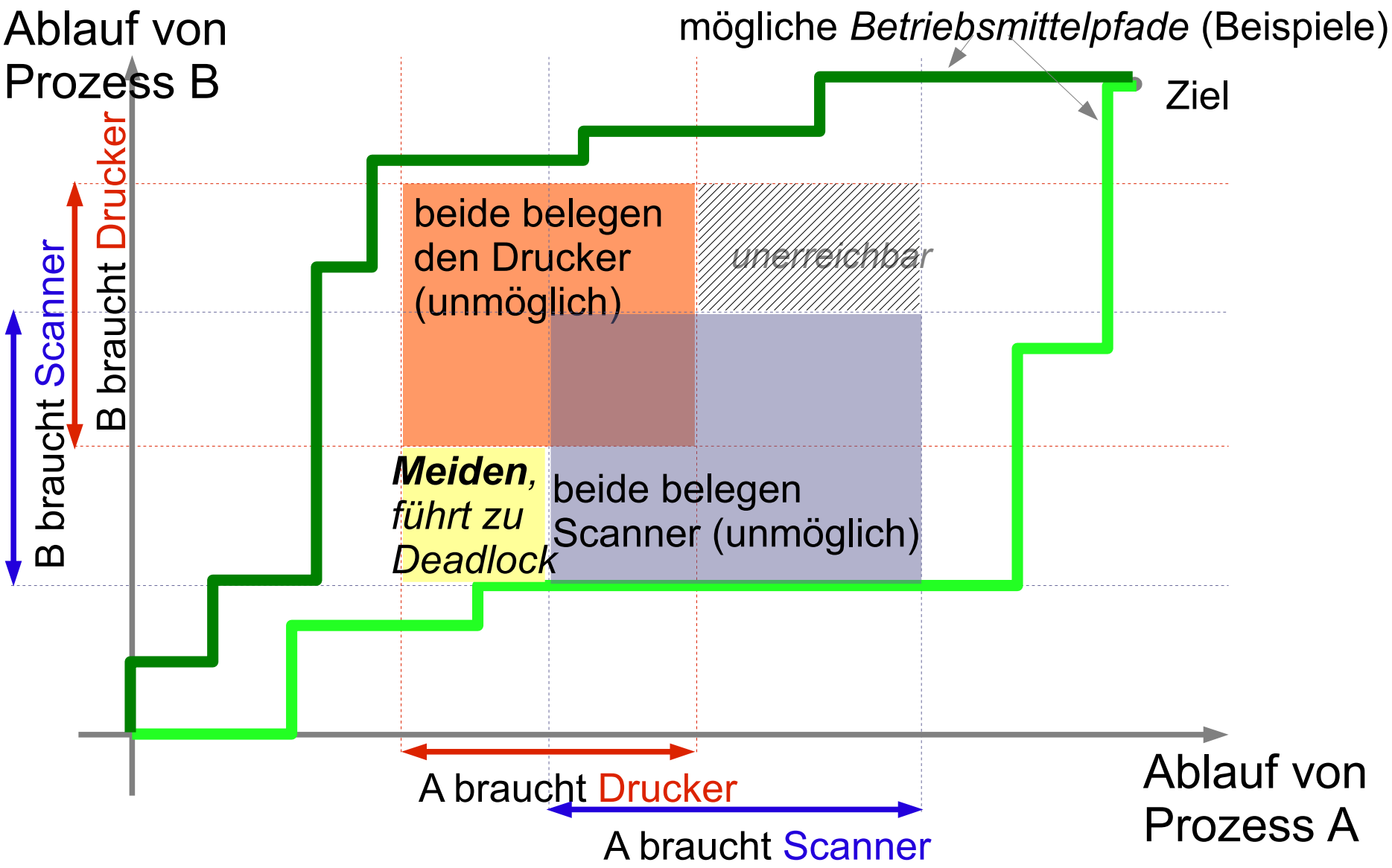
7.5 Verhindern von Deadlocks

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Kann man **Deadlocks** durch „geschicktes“ Vorgehen bei der Betriebsmittelzuteilung **von vornherein verhindern**?
- Welche Informationen müssen dazu vorab zur Verfügung stehen?
- Im folgenden betrachtet
  1. Betriebsmittelpfade (Grafische Veranschaulichung)
  2. Sichere und unsichere Zustände
  3. Der vereinfachte Bankiersalgorithmus für eine BM-Klasse
  4. Der Bankiersalgorithmus für mehrere BM-Klassen

7.5.1 Betriebsmittelpfade

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim



7.5.2 (Un-)Sichere Zustände

Def

- Ein Systemzustand ist **sicher**, wenn er
  - **keinen Deadlock** repräsentiert und
  - es eine geeignete Prozessausführungsreihenfolge gibt, bei der alle Anforderungen erfüllt werden
    (die also *auch dann* nicht in einen Deadlock führt, wenn alle Prozesse gleich ihre max. Ressourcenanzahl anfordern)
- Sonst heißt der Zustand **unsicher**.
- Im folgenden: Datenstrukturen aus 7.4.2:
  - E Betriebsmittelvektor
  - A Verfügbarkeitsvektor
  - C Belegungsmatrix
  - R Anforderungsmatrix

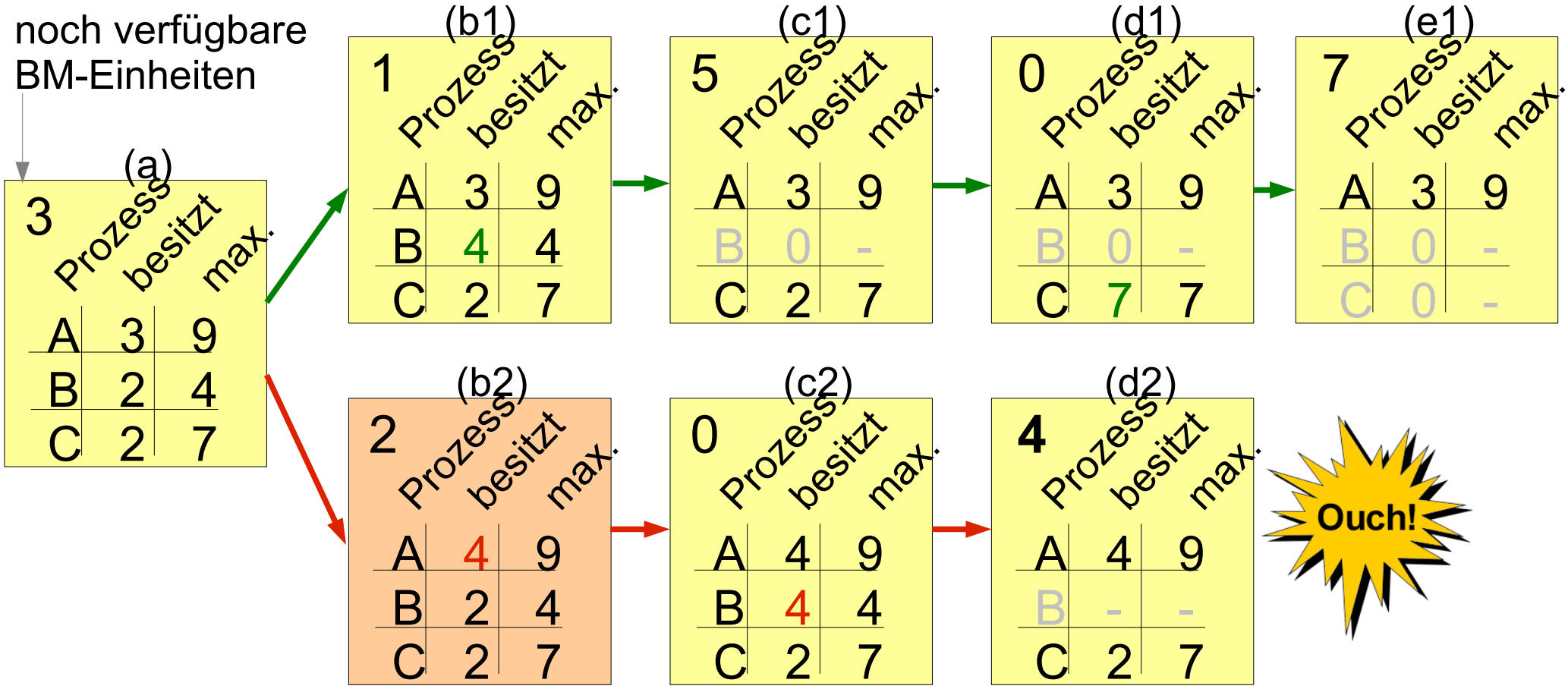
Was sagt uns das?

- Bei einem sicherem Zustand kann das System **garantieren**, dass alle Prozesse bis zum Ende durchlaufen können.
- Bei unsicherem Zustand ist das nicht garantierbar (aber auch nicht ausgeschlossen!).
  - Beispiel: Ein Prozess gibt ein BM zu einem „glücklichen Zeitpunkt“ kurzzeitig frei, wodurch eine Deadlock-Situation „zufällig“ vermieden wird. (→„Glück“ nicht vorhersehbar)
- „Unsicher“ bedeutet also *nicht* „Deadlock unvermeidlich“.

Beispiel

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- 3 Prozesse A,B,C; jeweils mit BM-Besitz und max. Bedarf
- ein Betriebsmitteltyp, 10x vorhanden



- Zustand (a) ist sicher (es gibt eine DL-freie Lösung)
- (b2) ist **nicht** sicher (A und C brauchen je 5, frei sind nur 4)

7.5.3 Bankier-Algorithmus (1 BM-Klasse)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Dijkstra (wer sonst? 1965):

- Ein **Bankier kennt** die **Kreditrahmen** seiner Kunden.
- Er geht davon aus, dass **nicht alle** Kunden **gleichzeitig** ihre Rahmen **voll** ausschöpfen werden.
- Daher hält er **weniger Bargeld** bereit als die **Summe** der Kreditrahmen.
- Gegebenenfalls **verzögert** er die **Zuteilung** eines Kredits, bis ein anderer Kunde zurückgezahlt hat.
- **Zuteilung** erfolgt **nur**, wenn sie "**sicher**" ist (also letztlich alle Kunden bis zu ihrem Kreditrahmen bedient werden können).

- Bankier = Betriebssystem, Bargeld = Betriebsmitteltyp, Kunden = Prozesse, Kredit = BM-Anforderung, z

Bankier-Algorithmus (2)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Prüfe bei jeder Anfrage,
  ob die Bewilligung in einen sicheren Zustand führt:
  - **Teste** dazu, ob ausreichend Betriebsmittel bereitstehen, um **mindestens einen** Prozess **vollständig** zufrieden zu stellen.
  - Davon ausgehend, dass dieser Prozess nach Durchlauf seine Betriebsmittel freigibt: führe **Test** mit dem Prozess aus, der dann am nächsten am Kreditrahmen ist
  - usw., **bis alle** Prozesse positiv getestet sind;
  - Falls **ja**, kann die aktuelle Anfrage **bewilligt** werden.
  - **Sonst**: Anforderung **verschieben** (warten)

7.5.4 Verallgemeinerter Bankier-Alg.

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Mehrere Betriebsmittelklassen
- Daten wie bei „Deadlockerkennung“ (7.4.2)
- Matrizen mit belegten / angeforderten Betriebsmitteln
- Vektoren mit BM-Bestand, verfügbaren BM und belegten BM je Betriebsmitteltyp
  - E Betriebsmittelvektor
  - A Verfügbarkeitsvektor („Betriebsmittelrestvektor“)
  - C Belegungsmatrix
  - R Anforderungsmatrix

Verallgemeinerter Bankier-Algorithmus

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- **Verfahren:**
  - Zu Beginn sind alle Prozesse unmarkiert.
  - **Suche** Zeile aus Anforderungsmatrix, die kleiner oder gleich dem Verfügbarkeitsvektor ist (falls keine existiert, wird das System in einen Deadlock laufen)
  - **Markiere** zugehörigen Prozess und **addiere** seine Betriebsmittel (zugehörige Zeile in der Belegungsmatrix) zum Verfügbarkeitsvektor
  - **Wiederhole** die letzten beide Schritte,
    - bis alle Prozesse markiert sind (→Zustand ist sicher)
    - oder ein Deadlock entdeckt wurde (→Zustand unsicher)
  - Falls sicher → Teile BM zu
  - Falls unsicher → Anfordernden Prozess blockieren
  - Bei Beendigung eines Prozesses werden alle verbleibenden Prozesse geweckt und die Anforderungen gemäß Verfahren neu bearbeitet

Beispiel

$$E = (6\ 3\ 4\ 2) \quad \text{vorhanden}$$

$$C = \begin{pmatrix} 3 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad \text{zugewiesen}$$

$$A = (1\ 0\ 2\ 0) \quad \text{verfügbar}$$

$$P = (5\ 3\ 2\ 2) \quad \text{belegt}$$

$$R = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 2 \\ 3 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 2 & 1 & 1 & 0 \end{pmatrix} \quad \text{angefordert}$$

Sicher? Ja, Ausführungsfolge P4, P1, P5, ... ist möglich:

P4 -> A = (2 1 2 1)
P1 -> A = (5 1 3 2)
P5 -> A = (5 1 3 2)
P2 -> A = (5 2 3 2)
P3 -> A = (6 3 4 2)

Beispiel

$$E = (6\ \ 3\ \ 4\ \ 2) \quad \text{vorhanden}$$

$$C = \begin{pmatrix} 3 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad \text{zugewiesen}$$

$$A = (1\ \ 0\ \ 1\ \ 0) \quad \text{verfügbar}$$

$$P = (5\ \ 3\ \ 3\ \ 2) \quad \text{belegt}$$

$$R = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 2 \\ 3 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 2 & 1 & 1 & 0 \end{pmatrix} \quad \text{angefordert}$$

P2 fordere ein BM 3 an (rot)

Sicher? Ja (P4, P1, P5, P2, P3)

P4 -> A = (2 1 1 1)
P1 -> A = (5 1 2 2)
P5 -> A = (5 1 2 2)
P2 -> A = (5 2 3 2)
P3 -> A = (6 3 4 2)

also erhält P2 ein BM 3

Beispiel

$E = (6\ \ 3\ \ 4\ \ 2)$ vorhanden

$$C = \begin{pmatrix} 3 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & \mathbf{1} & 0 \end{pmatrix}$$ zugewiesen

$A = (1\ \ 0\ \ \mathbf{0}\ \ 0)$ verfügbar

$P = (5\ \ 3\ \ \mathbf{4}\ \ 2)$ belegt

$$R = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 2 \\ 3 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 2 & 1 & 1 & 0 \end{pmatrix}$$ angefordert

Nun fordere auch P5 ein BM 3 an
→ dann würde A = (1 0 0 0)

Sicher? *Nein!*
→ daher Anfrage von P5 blockieren

Ist der Bankier-Algorithmus praktikabel?

- In der Praxis gibt es mehrere **Probleme** beim Einsatz:
  - Prozesse können "maximale Ressourcenanforderung" selten im Voraus angeben
  - Anzahl der Prozesse ändert sich ständig
  - Ressourcen können verschwinden (z.B. durch Ausfall)

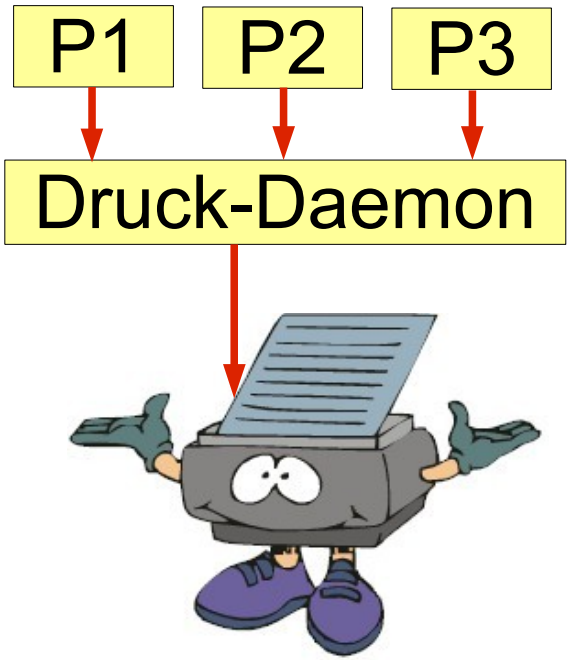
7.6 Deadlock-Vermeidung

- Deadlock-Verhinderung ist wenig praktikabel :-(
- Ansatz: **Vermeidung** mindestens einer der vier Deadlock-**Voraussetzungen** (vgl 7.2)
  - Wechselseitiger Ausschluss
  - Belegungs-/Anforderungsbedingung („Hold-and-Wait“, d.h. zu reservierten BM weitere anforderbar)
  - Ununterbrechbarkeit (kein erzwungener BM-Entzug)
  - zyklisches Warten

7.6.1 Wechselseitiger Ausschluß?

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Falls es keine exklusive Zuteilung eines Betriebsmittels an einen Prozess gibt, gibt es auch keine Deadlocks.
- **Beispiel**: Zugriff auf Drucker
- Einführung eines **Spool-Systems**, das
  - Druckaufträge von Prozessen (schnell) entgegennimmt
  - ggf. zwischenspeichert
  - und der Reihe nach auf dem Drucker ausgibt
- **Entkopplung** zwischen (konkurrierenden) Prozessen und dem (langsamen) Betriebsmittel
- **Vermeidung** einer exklusiven Zuteilung des Betriebsmittels „Drucker“



7.6.2 Belegungs-/Anforderungsbed.?

- Vermeiden, dass neue Betriebsmittel-Anforderungen zu bereits bestehenden hinzukommen.
- **„Preclaiming“**: Alle Anforderungen zu Beginn der Ausführung stellen („alles oder nichts“)
- **Vorteil**: Wenn Anforderungen erfüllt werden, kann der Prozess bestimmt bis zum Ende durchlaufen
  (er hat ja dann alles, was er braucht)
- **Nachteil**:
  - Anforderungen müssen **zu Beginn bekannt** sein
  - Betriebsmittel werden unter Umständen **lange blockiert**
  - und können zwischenzeitlich nicht (sinnvoll) anders genutzt werden.
- **Beispiel**: Batch-Jobs bei Großrechnern.

7.6.3 Ununterbrechbarkeit?

- Hängt vom Betriebsmittel ab, aber
- „gewaltsamer“ Entzug ist in der Regel nicht akzeptabel
  - Drucker?
  - CD-Brenner?

7.6.4 Zyklische Wartebedingung?

Hochschule **RheinMain**
University of Applied Sciences
Wiesbaden Rüsselsheim

- Wenn es kein zyklisches Auf-einander-warten gibt, entstehen auch keine Deadlocks
- Idee:
  - Betriebsmitteltypen **linear ordnen** und
  - nur in aufsteigender Ordnung Anforderungen annehmen (wenn mehrere Exemplare eines Typs gebraucht werden: alle Exemplare auf einmal anfordern)
  - „Drucker *vor* Scanner *vor* CD-Brenner *vor* ...“
- Dadurch entsteht **automatisch** ein **zyklenfreier** Belegungs-Anforderungs-Graph,
- wodurch Deadlocks ausgeschlossen sind.
- Tatsächlich praktikables Verfahren.

DL-Vermeidung im Überblick

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Deadlock-Vermeidung durch Verhinderung (mindestens) einer der 4 Vorbedingungen eines Deadlocks ist möglich:

- Wechselseitiger Ausschluß → Spooling
- Belegungs-/Anforderungsbed. → Preclaiming
- Ununterbrechbarkeit *(BM-Entzug... besser nicht)*
- Zyklisches Warten → Betriebsmittel ordnen

7.7 Verwandte Fragestellungen

- Deadlocks bei der Benutzung von Semaphoren (vgl. Kap. 3)
- Zwei-Phasen-Locking in Datenbanken
- Verhungern (Starvation), kein Deadlock, aber auch kein Fortschritt für einen Prozess (vgl. Philosophen-Problem)

7.8 Zusammenfassung

- Deadlocks sind ein Problem in jedem Betriebssystem aber auch in nebenläufigen Anwendungssystemen.
- Deadlocks treten auf, wenn einer Menge von Prozessen jeweils exklusiv ein Betriebsmittel zugeteilt ist, und alle ein Betriebsmittel anfordern, das bereits von einem anderen Prozess der Menge belegt ist. Alle Prozesse sind blockiert, keiner kann jemals fortgeführt werden.
- Deadlocks können durch vier notwendige Bedingungen charakterisiert werden:
  1. Bedingung des wechselseitigen Ausschlusses
  2. Belegungs- / Anforderungs-Bedingung
  3. Ununterbrechbarkeitsbedingung
  4. Zyklische Wartebedingung

Zusammenfassung (2)

Hochschule RheinMain
University of Applied Sciences
Wiesbaden Rüsselsheim

- Deadlocks können vermieden werden (7.5), indem überprüft wird, ob der nachfolgende Zustand sicher ist oder nicht.
  - Zustand ist sicher, wenn es Folge von Ereignissen gibt, so dass alle Prozesse ihre Ausführung beenden können.
  - In unsicherem Zustand gibt es keine Garantie dafür.
  - Der Bankier-Algorithmus vermeidet Deadlocks, in dem er Anforderungen zurückstellt, die das System in einen unsicheren Zustand überführen würden.
- Deadlocks können durch konstruktive Verfahren verhindert werden (7.6)
  - z.B. durch Vorabbelegen aller Betriebsmittel
  - durch die geordnete Betriebsmittelbenutzung.