

Systeme I: Betriebssysteme

Kapitel 6 **Deadlocks**

Wolfram Burgard

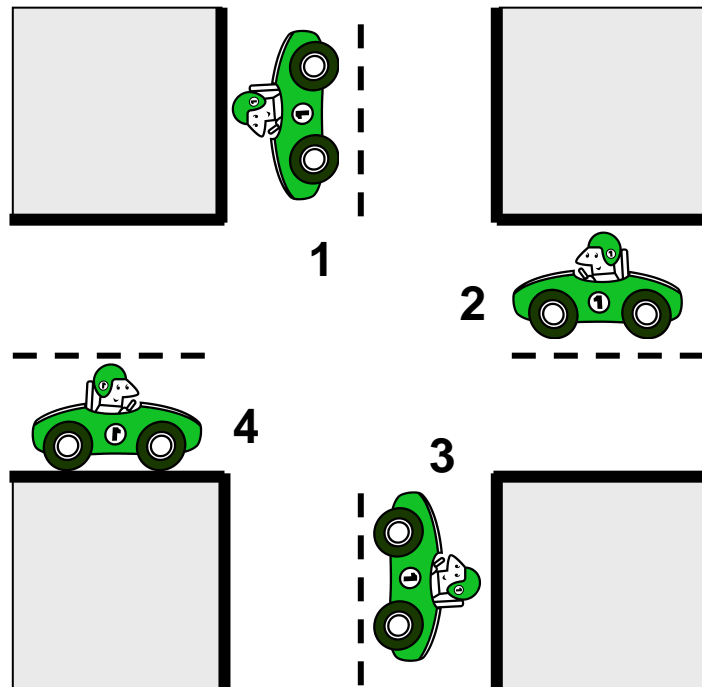


Inhalt Vorlesung

Verschiedene Komponenten / Konzepte von Betriebssystemen

- Dateisysteme
- Prozesse
- Nebenläufigkeit und wechselseitiger Ausschluss
- **Deadlocks**
- Scheduling
- Speicherverwaltung

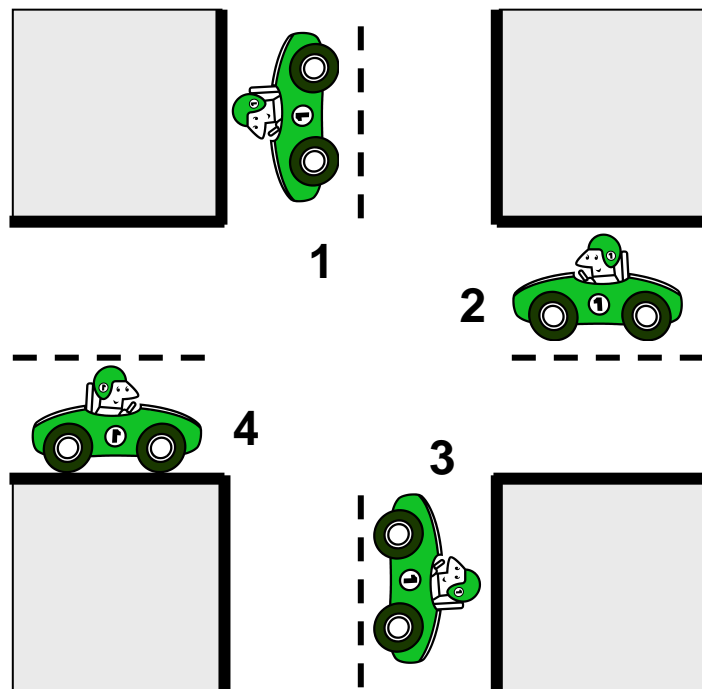
Deadlocks: Einführung (1)



Deadlocks: Einführung (1)

Eine Menge von Prozessen befindet sich in einem **Deadlock**, wenn jeder Prozess der Menge auf ein Ereignis wartet, das nur ein anderer Prozess der Menge auslösen kann

Beispiel:



Deadlocks: Einführung (2)

- Ressourcen, die zu einem Zeitpunkt jeweils nur ein Prozess benutzen kann
- Beispiele: Datensätze, Geräte, (Haupt-)Speichersegmente, E/A-Kanäle
- Typischerweise verlangen Prozesse auf mehrere Ressourcen alleinigen Zugriff

Deadlocks: Einführung (3)

- Die Benutzung einer Ressource besteht aus
 - Fordere Ressource an
 - Benutze die Ressource
 - Gib die Ressource frei
- Wenn Anforderung fehlschlägt, wartet der Prozess kurz und wiederholt sie dann

Deadlocks: Einführung (4)

- Idee: Ein binäres Semaphor (oder Mutex) für jede Ressource
- Bei mehreren Prozessen und Ressourcen kann es zum Deadlock kommen

Beispiel: Problemlose Ausführung

- Annahme: Exklusiver Zugriff auf Ressourcen

```
/* Prozess 1 */
```

```
...
```

```
Fordere Ressource 1 an
```

```
...
```

```
Fordere Ressource 2 an
```

```
...
```

```
Benutze beide Ressourcen
```

```
...
```

```
Gib beide Ressourcen frei
```

```
...
```

```
/* Prozess 2 */
```

```
...
```

```
Fordere Ressource 2 an
```

```
...
```

```
Fordere Ressource 1 an
```

```
...
```

```
Benutze beide Ressourcen
```

```
...
```

```
Gib beide Ressourcen frei
```

```
...
```

- Wenn Prozess 1 seine Arbeit rechtzeitig vor Anforderung von Ressource 2 von Prozess 2 beendet, tritt kein Deadlock auf.

Möglicher Deadlock

- Annahme: Exklusiver Zugriff auf Ressourcen

t_1

```
/* Prozess 1 */
...
Fordere Ressource 1 an
...
Fordere Ressource 2 an
 $t_4$ 
...
Benutze beide Ressourcen
...
Gib beide Ressourcen frei
...
```

t_2

```
/* Prozess 2 */
...
Fordere Ressource 2 an
...
Fordere Ressource 1 an
 $t_3$ 
...
Benutze beide Ressourcen
...
Gib beide Ressourcen frei
...
```

- Hier kommt es zu einer Deadlock-Situation!

Möglicher Deadlock

- Annahme: Exklusiver Zugriff auf Ressourcen

t_1

```
/* Prozess 1 */
...
Fordere Ressource 1 an
...
Fordere Ressource 2 an
 $t_4$ 
...
Benutze beide Ressourcen
...
Gib beide Ressourcen frei
...
```

t_2

```
/* Prozess 2 */
...
Fordere Ressource 2 an
...
Fordere Ressource 1 an
 $t_3$ 
...
Benutze beide Ressourcen
...
Gib beide Ressourcen frei
...
```

- Prozess 2 blockiert zum Zeitpunkt t_3 , weil Ressource 1 an Prozess 1 vergeben ist
- Prozess 1 blockiert zum Zeitpunkt t_4 , weil Ressource 2 an Prozess 2 vergeben ist

Hinweis

- Das Betriebssystem kann zu jedem Zeitpunkt jeden beliebigen nicht blockierten Prozess ausführen
- Streng sequentielle Ausführung ist nicht unbedingt optimal



Voraussetzungen für Ressourcen-Deadlocks (1)

- **Wechselseitiger Ausschluss:** Jede Ressource ist entweder verfügbar oder genau einem Prozess zugeordnet
- **Besitzen und Warten:** Prozesse, die schon Ressourcen reserviert haben, können noch weitere Ressourcen anfordern

Voraussetzungen für Ressourcen-Deadlocks (2)

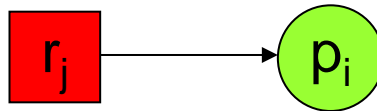
- **Kein Ressourcenentzug:** Ressourcen, die einem Prozess bewilligt wurden, können nicht gewaltsam wieder entzogen werden
- **Zyklisches Warten:** Es gibt eine zyklische Kette von Prozessen, von denen jeder auf eine Ressource wartet, die dem nächsten Prozess in der Kette gehört

Modellierung von Ressourcenbelegungen und Ressourcenanforderungen (1)

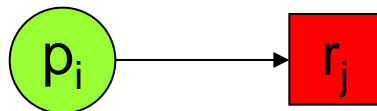
- **Belegungs-Anforderungs-Graph:** Zur Erkennung von Deadlock-Situationen
- Zwei Arten von Knoten:
 - Kreise repräsentieren Prozesse p_i : 
 - Quadrate repräsentieren Ressourcen r_j : 

Modellierung von Ressourcenbelegungen und Ressourcenanforderungen (2)

- Kante von einer Ressource r_j zu einem Prozess p_i : Ressource r_j wird von Prozess p_i belegt

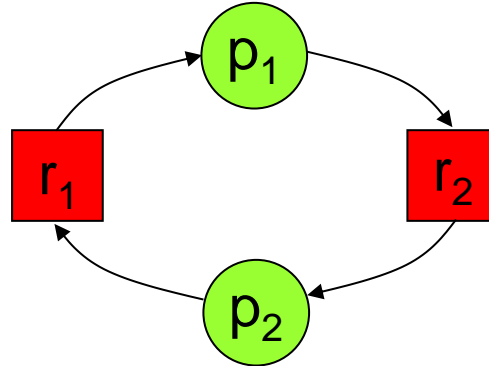


- Kante von einem Prozess p_i zu einer Ressource r_j : Prozess p_i hat Ressource r_j angefordert, aber noch nicht erhalten



Zyklen im Belegungs-Anforderungs-Graphen

- Zyklus: Wenn man von einem Knoten ausgehend über eine Folge von Kanten wieder zu dem Knoten zurückkommt



- Zyklen im Belegungs-Anforderungsgraphen repräsentieren Deadlocks!

Beispiel – Abarbeitungsfolge 1

Prozess p_1 : ¹Anforderung R, ⁴Anforderung S, Freigabe R, Freigabe S

Prozess p_2 : ²Anforderung S, ⁵Anforderung T, Freigabe S, Freigabe T

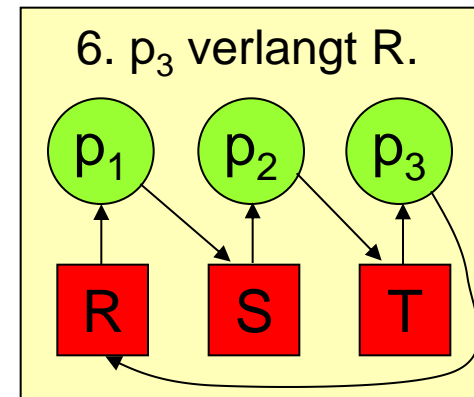
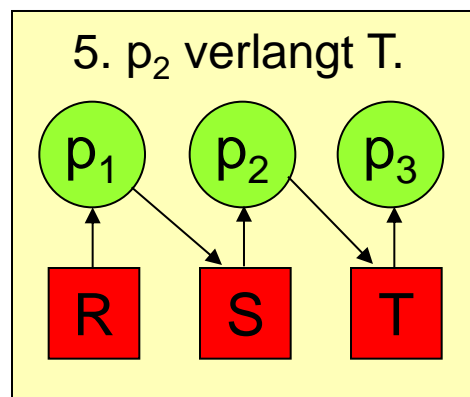
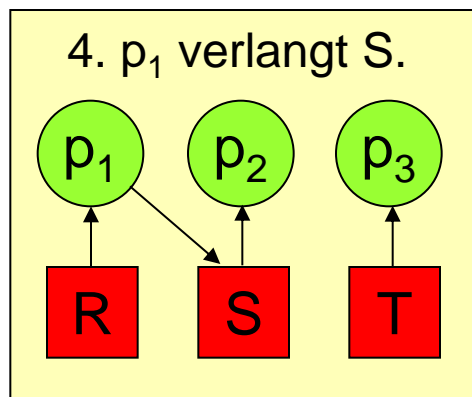
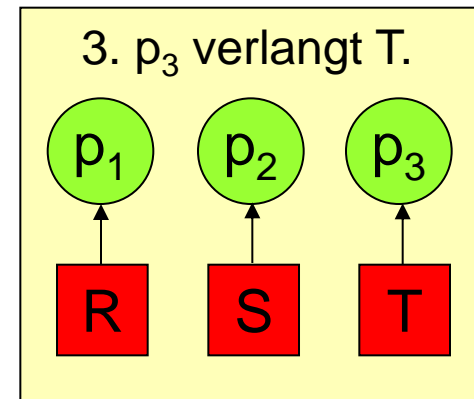
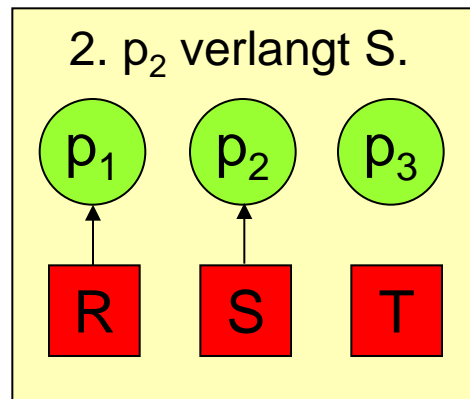
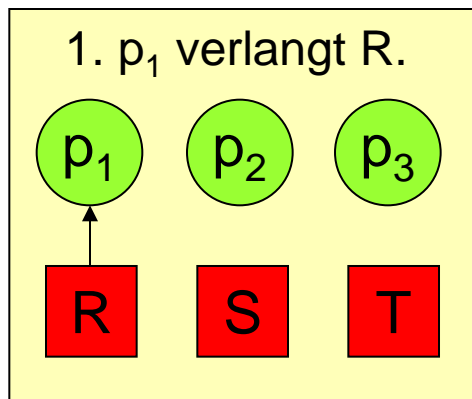
Prozess p_3 : ³Anforderung T, ⁶Anforderung R, Freigabe T, Freigabe R

Beispiel – Abarbeitungsfolge 1

Prozess p_1 : ¹Anforderung R, ⁴Anforderung S, Freigabe R, Freigabe S

Prozess p_2 : ²Anforderung S, ⁵Anforderung T, Freigabe S, Freigabe T

Prozess p_3 : ³Anforderung T, ⁶Anforderung R, Freigabe T, Freigabe R



Deadlock!

Beispiel – Abarbeitungsfolge 2

Prozess p_1 : ¹Anforderung R, ³Anforderung S, ⁵Freigabe R, ⁶Freigabe S

Prozess p_2 : Anforderung S, Anforderung T, Freigabe S, Freigabe T

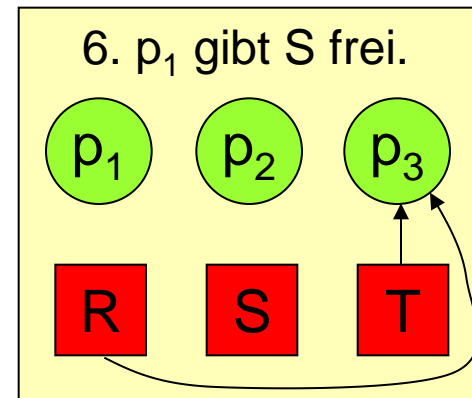
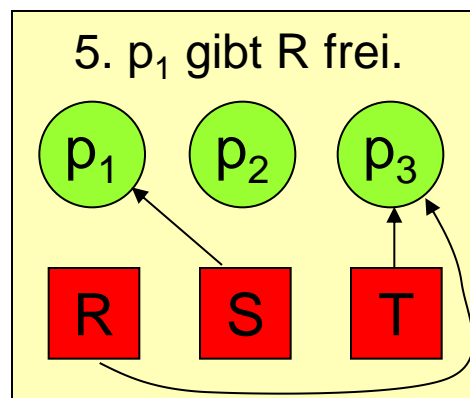
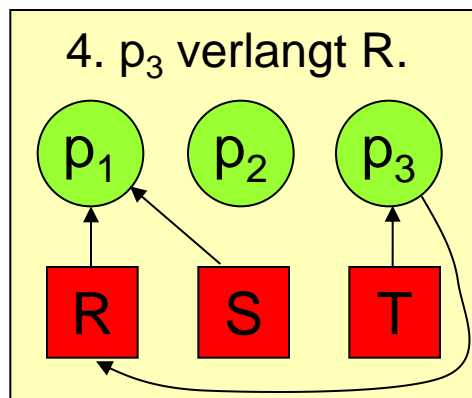
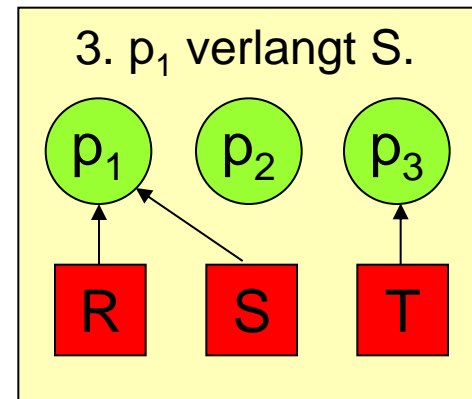
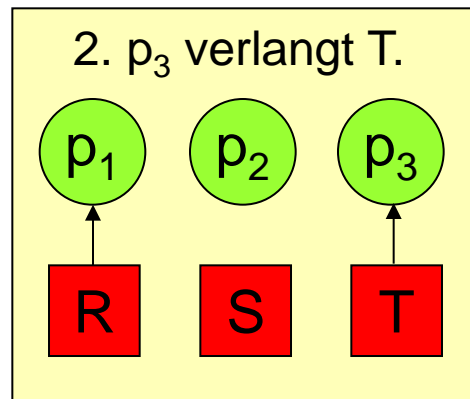
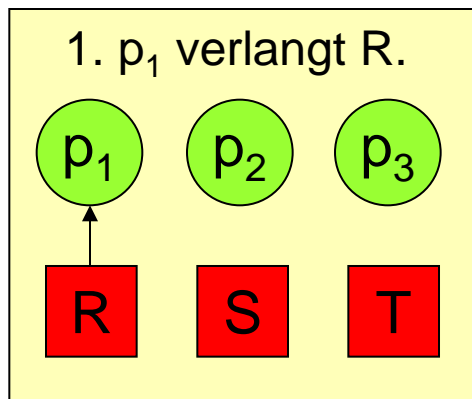
Prozess p_3 : ²Anforderung T, ⁴Anforderung R, Freigabe T, Freigabe R

Beispiel – Abarbeitungsfolge 2

Prozess p_1 : ¹Anforderung R, ³Anforderung S, ⁵Freigabe R, ⁶Freigabe S

Prozess p_2 : Anforderung S, Anforderung T, Freigabe S, Freigabe T

Prozess p_3 : ²Anforderung T, ⁴Anforderung R, Freigabe T, Freigabe R



kein
Deadlock!

Ressourcendiagramm

- Diagramm zur Visualisierung der Ressourcenanforderungen über die Zeit
- Dient zur Erkennung von potentiellen Deadlocks
- Zeitachsen: Prozessfortschritt
- Ressourcenspur: Eine mögliche Ausführungsreihenfolge der Anweisungen (nur nach rechts oder nach oben)
- Enthält ein Rechteck für jedes Zeitintervall, in dem eine Ressource von zwei Prozessen beansprucht werden kann
- Wir suchen eine Ausführungsreihenfolge, die jedes Rechteck umgeht

Ressourcendiagramm

Prozess 1

1: ...

2: ...

3: Anforderung B

4: Anforderung A

5: Freigabe B

6: ...

7: Freigabe A

Prozess 2

1: ...

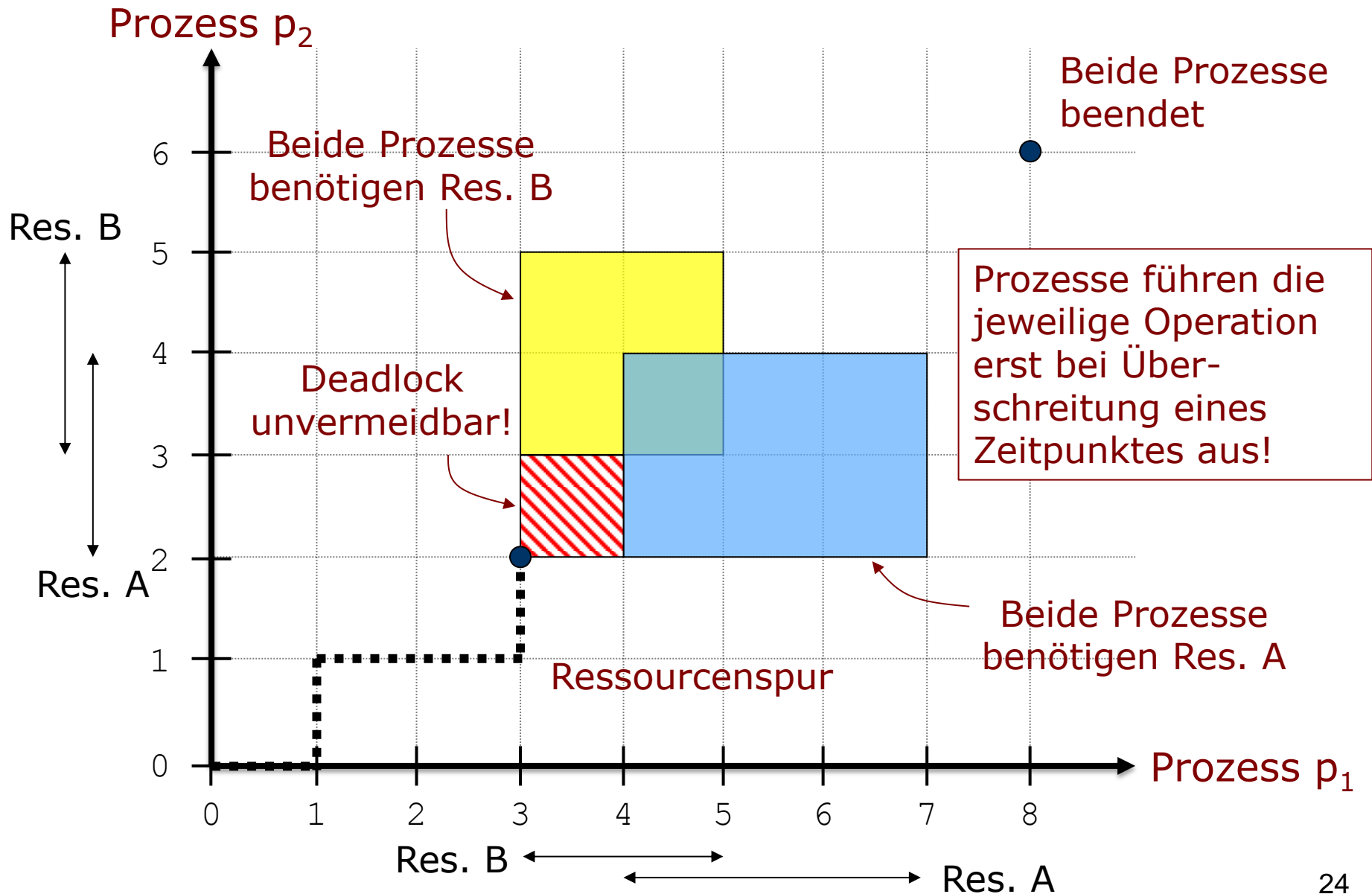
2: Anforderung A

3: Anforderung B

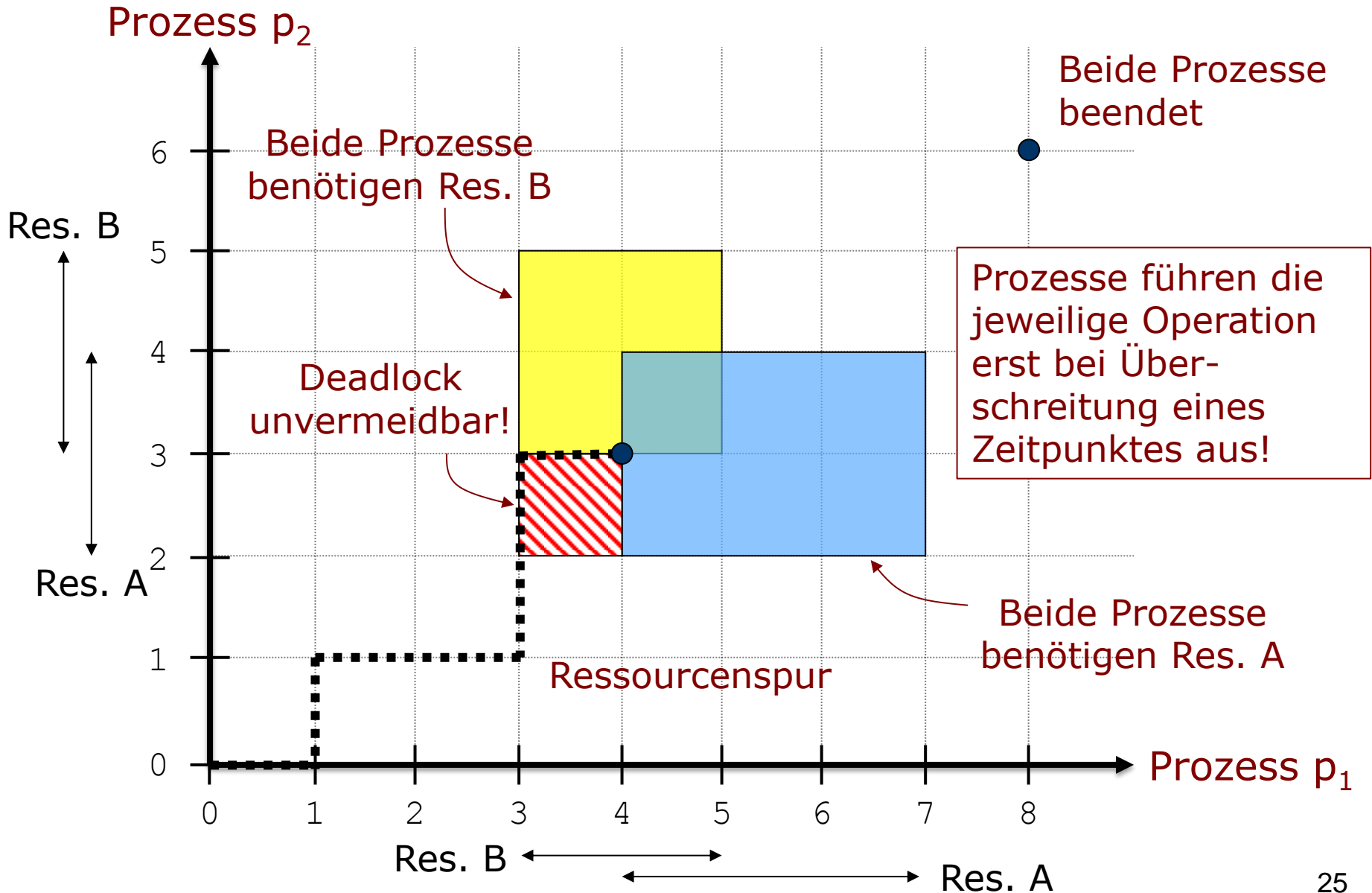
4: Freigabe A

5: Freigabe B

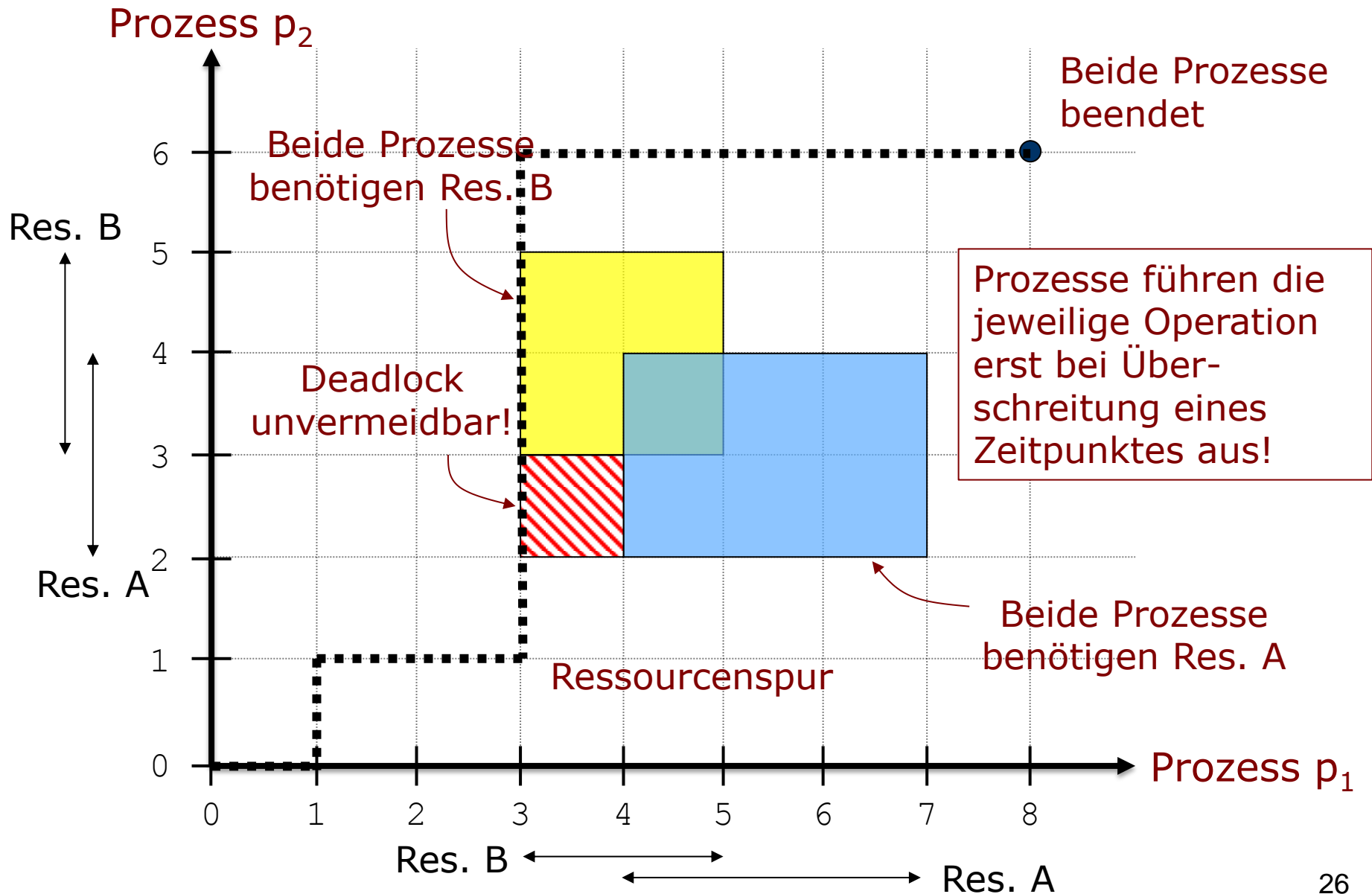
Ressourcendiagramm



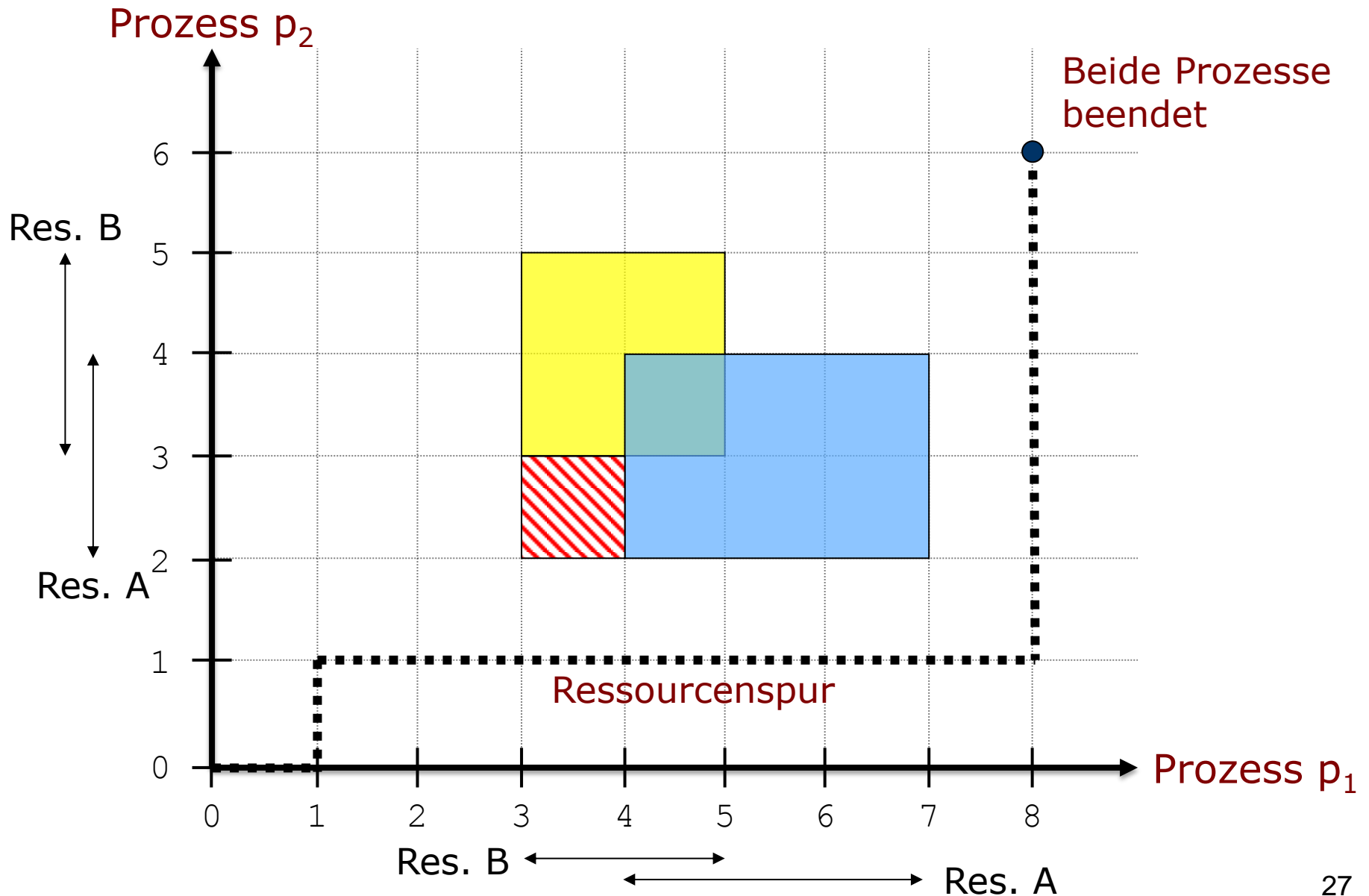
Beispiel Deadlock



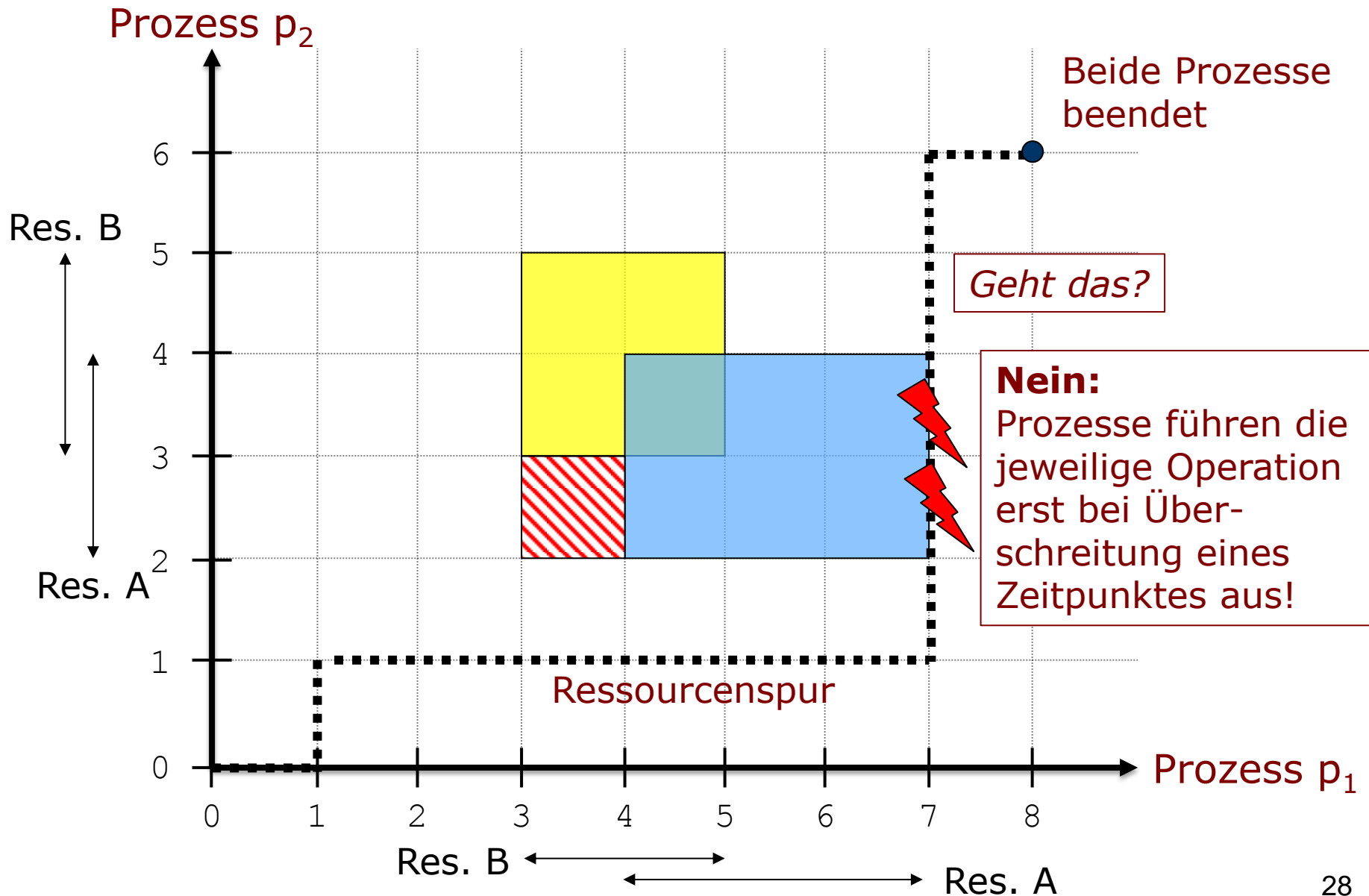
Beispiel ohne Deadlock (1)



Beispiel ohne Deadlock (2)



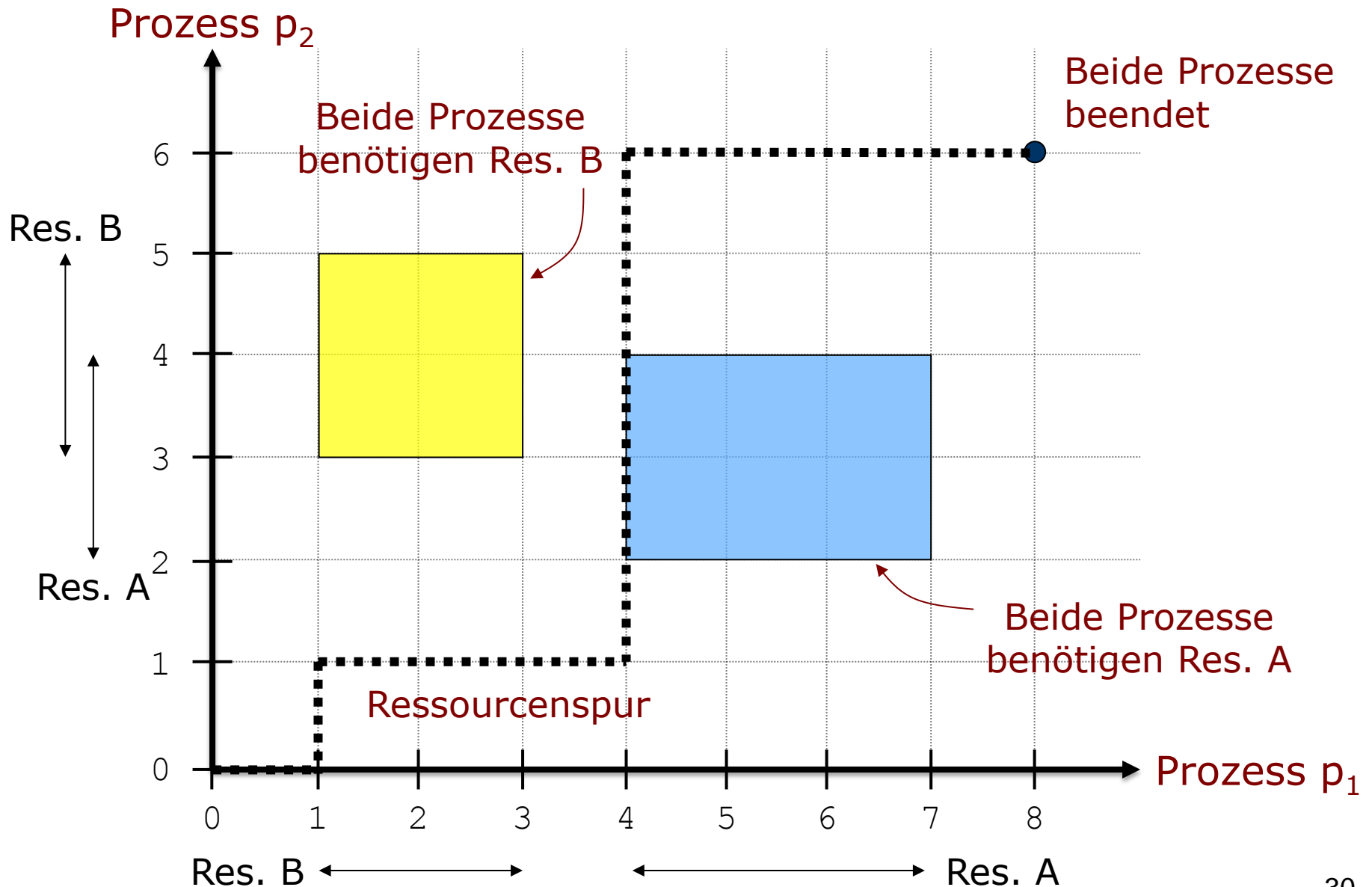
Beispiel (ungültige Reihenfolge)



Beispiel 2: Nie Deadlock (1)

- Ob ein Deadlock eintreten kann oder nicht, hängt von der Situation ab
- Beispiel: Prozess 1 benötigt **nicht gleichzeitig** beide Ressourcen
- Dann kann unabhängig von der relativen Ausführung der beiden Prozesse **kein Deadlock** eintreten

Beispiel 2: Nie Deadlock (2)



Verhindern von Deadlocks (1)

- Ressourcen werden nach und nach von den Prozessen angefordert
- Ziel: Konstruiere einen Mechanismus, so dass Deadlocks **garantiert** verhindert werden können
- Teile einem Prozess eine Ressource nur zu, wenn dies „ungefährlich“ ist

Verhindern von Deadlocks (2)

- **Bankier-Algorithmus** (Dijkstra, 1965):
Ressourcenzuteilung an Prozesse, so dass Deadlocks verhindert werden
- **Voraussetzungen:**
 - Im Voraus bekannt: Welche und wie viele Ressourcen die einzelnen Prozesse maximal anfordern werden
 - Anforderung übersteigt für keinen Prozess die zur Verfügung stehenden Ressourcen

Verhindern von Deadlocks (3)

- Immer möglich: Ausführen aller Prozesse nacheinander (ineffizient!)
- Nach Ablauf eines Prozesses: Freigabe aller Ressourcen und der nächste Prozess kann ausgeführt werden

Verhindern von Deadlocks (4)

Grundidee des Bankier-Algorithmus:

- Versuche möglichst viel (Pseudo-) Parallelismus zu erreichen
- Riskiere dabei aber zu keinem Zeitpunkt eine potentielle Deadlock-Situation
- Ressourcenanforderung wird nur gewährt, wenn sie garantiert nicht zu einem Deadlock führen kann

Bankier-Algorithmus (1)

Ein Zustand ist **sicher**, wenn

- es auf jeden Fall eine deadlock-freie „Restausführung“ aller Prozesse gibt,
- unabhängig davon, wann die Prozesse in Zukunft ihre Ressourcenanforderungen und -freigaben durchführen, und
- auch dann, wenn gilt:
 - Prozesse stellen ihre restlichen Anforderungen jeweils auf einen Schlag und
 - Freigaben finden erst bei Prozessbeendigung statt (Worst Case).

Bankier-Algorithmus (2)

- Wenn Deadlock-freie Restausführung nicht garantiert werden kann: Zustand ist **unsicher**
- Beachte: Ein unsicherer Zustand muss nicht notwendigerweise zu einem Deadlock führen

Bankier-Algorithmus (3)

- Strategie: Überführe das System immer nur in **sichere** Zustände
- Nach Voraussetzung ist der Startzustand sicher
- **Bei jeder Ressourcenanforderung** eines Prozesses: Prüfe, ob das System bei Erfüllung der Anforderung in einen **sicheren** Zustand kommt
- **Falls nein**: Erfülle Anforderung **nicht**, stelle Prozess p_i zurück und mache mit einem anderen Prozess weiter; **sonst erfülle sie**

Bankier-Algorithmus – Überprüfung auf sicheren Zustand

- Gibt es einen Prozess, dessen verbliebene Anforderungen alle mit den verfügbaren Ressourcen erfüllt werden können?
- Nimm an, dass dieser Prozess ausgeführt wird und alle seine Ressourcen danach freigibt
- Gibt es einen weiteren Prozess, dessen Ressourcenanforderung alle erfüllt werden können? Wenn ja, verfahren mit diesem Prozess gleichermaßen
- Zustand sicher: Alle Prozesse können so zu Ende gebracht werden

Bankier-Algorithmus

Eine einzige Ressourcenklasse (1)

- Beispiel: Es gibt 10 Instanzen einer einzigen Ressourcenklasse
- Wenn alle 10 durch Prozesse belegt sind, dann wird kein weiterer vergeben
- Ein Prozess kann mehrere Ressourcen anfordern, aber nur bis zu einer bestimmten Maximalanzahl ≤ 10

Bankier-Algorithmus

Eine einzige Ressourcenklasse (2)

- Gegeben: Prozesse p_1, \dots, p_n , die Ressourcen aus einer einzigen Klasse anfordern
- Gegeben: Anzahl zur Verfügung stehender Ressourcen: $V \in \mathbb{N}$

Bankier-Algorithmus

Eine einzige Ressourcenklasse (3)

- Für jeden Prozess p_i gibt es
 - eine maximale Anzahl M_i von Ressourcen, die der Prozess noch **anfordern** wird
 - eine Anzahl von Ressourcen E_i , die der Prozess zu einem bestimmten Zeitpunkt schon **erhalten** hat
 - eine Anzahl A_i von Ressourcen, die der Prozess nach diesem Zeitpunkt noch maximal **anfordern** wird: $A_i = M_i - E_i$
 - Anzahl F der **freien** Ressourcen zu diesem Zeitpunkt $F = V - \sum_{i=1}^n E_i$
- Es gilt $\forall 1 \leq i \leq n : M_i \leq V$ und $E_i \leq M_i$

Bankier-Algorithmus Beispiel (1)

- Es gibt $V = 10$ Instanzen einer Ressource
- Drei Prozesse p_1, p_2, p_3
- Maximale Anforderungen:

	M_i
p_1	9
p_2	4
p_3	7

- Zustand zum Zeitpunkt t :

	E_i	A_i
p_1	3	6
p_2	2	2
p_3	2	5

Bankier-Algorithmus Beispiel (1)

- Es gibt $V = 10$ Instanzen einer Ressource
- Drei Prozesse p_1, p_2, p_3
- Maximale Anforderungen:

	M_i
p_1	9
p_2	4
p_3	7

- Zustand zum Zeitpunkt t :

	E_i	A_i
p_1	3	6
p_2	2	2
p_3	2	5

- Freie Ressourcen
 $F = 10 - 7 = 3$

- Ist dies ein sicherer Zustand?

Bankier-Algorithmus Beispiel (2)

Nachweis:

Es handelt sich um einen sicheren Zustand

	E_i	A_i	M_i
p_1	3	6	9
p_2	2	2	4
p_3	2	5	7


$$F = 10 - 7 = 3$$

Es muss gelten: Es gibt **auf jeden Fall** eine deadlockfreie „Restausführung“ aller Prozesse, auch wenn die Prozesse jeweils ihre restlichen Anforderungen auf **einen Schlag stellen** und Freigaben **erst bei Prozessbeendigung** durchführen

Bankier-Algorithmus Beispiel (2)

Nachweis:

Es handelt sich um einen sicheren Zustand



	E_i	A_i	M_i		E_i	A_i	M_i	
p_1	3	6	9	Führe zunächst ausschließlich Prozess p_2 aus 	p_1	3	6	9
p_2	2	2	4		p_2	4	0	4
p_3	2	5	7		p_3	2	5	7

$F = 10 - 7 = 3$ $F = 10 - 9 = 1$

Bankier-Algorithmus Beispiel (2)

Nachweis:

Es handelt sich um einen sicheren Zustand

	E_i	A_i	M_i		E_i	A_i	M_i		E_i	A_i	M_i		
p_1	3	6	9	Führe zunächst ausschließlich Prozess p_2 aus 	p_1	3	6	9	Freigabe durch Prozess p_2 	p_1	3	6	9
p_2	2	2	4		p_2	4	0	4		p_2	0	-	-
p_3	2	5	7		p_3	2	5	7		p_3	2	5	7
$F = 10 - 7 = 3$					$F = 10 - 9 = 1$					$F = 10 - 5 = 5$			

Bankier-Algorithmus Beispiel (2)

Nachweis:

Es handelt sich um einen sicheren Zustand

	E_i	A_i	M_i			E_i	A_i	M_i			E_i	A_i	M_i
p_1	3	6	9	Führe zunächst ausschließlich Prozess p_2 aus	p_1	3	6	9	Freigabe durch Prozess p_2	p_1	3	6	9
p_2	2	2	4		p_2	4	0	4		p_2	0	-	-
p_3	2	5	7		p_3	2	5	7		p_3	2	5	7
$F = 10 - 7 = 3$					$F = 10 - 9 = 1$					$F = 10 - 5 = 5$			

Führe ausschließlich Prozess p_3 aus

	E_i	A_i	M_i
p_1	3	6	9
p_2	0	-	-
p_3	7	0	7
$F = 10 - 10 = 0$			

Bankier-Algorithmus Beispiel (2)

Nachweis:

Es handelt sich um einen sicheren Zustand

	E_i	A_i	M_i			E_i	A_i	M_i			E_i	A_i	M_i
p_1	3	6	9	Führe zunächst ausschließlich Prozess p_2 aus	p_1	3	6	9	Freigabe durch Prozess p_2	p_1	3	6	9
p_2	2	2	4		p_2	4	0	4		p_2	0	-	-
p_3	2	5	7		p_3	2	5	7		p_3	2	5	7
$F = 10 - 7 = 3$					$F = 10 - 9 = 1$					$F = 10 - 5 = 5$			

Führe ausschließlich Prozess p_3 aus

	E_i	A_i	M_i			E_i	A_i	M_i
p_1	3	6	9	Freigabe durch Prozess p_3	p_1	3	6	9
p_2	0	-	-		p_2	0	-	-
p_3	7	0	7		p_3	0	-	-
$F = 10 - 10 = 0$					$F = 10 - 3 = 7$			

Jetzt kann Prozess 1 zu Ende gebracht werden!

Bankier-Algorithmus Beispiel (3)

Kann Prozess p_1 eine weitere Ressource zugewiesen werden?

	E_i	A_i	M_i
p_1	3	6	9
p_2	2	2	4
p_3	2	5	7


$$F = 10 - 7 = 3$$

Organisatorisches

- Klausur: 9.3.2018 (Raum TBA)
- Vorlesung am 20.12. findet statt

Bankier-Algorithmus Beispiel (3)


Kann Prozess p_1 eine weitere Ressource zugewiesen werden?

	E_i	A_i	M_i		E_i	A_i	M_i
p_1	3	6	9		4	5	9
p_2	2	2	4		2	2	4
p_3	2	5	7		2	5	7

$F = 10 - 7 = 3$ $F = 10 - 8 = 2$

Bankier-Algorithmus Beispiel (3)

Kann Prozess p_1 eine weitere Ressource zugewiesen werden?

	E_i	A_i	M_i		E_i	A_i	M_i	
p_1	3	6	9		p_1	4	5	9
p_2	2	2	4		p_2	2	2	4
p_3	2	5	7		p_3	2	5	7

$F = 10 - 7 = 3$ $F = 10 - 8 = 2$

Frage: Ist dieser Zustand immer noch sicher?

Bankier-Algorithmus Beispiel (4)

Antwort: Der Zustand ist nicht sicher


	E_i	A_i	M_i
p_1	4	5	9
p_2	2	2	4
p_3	2	5	7

$$F = 10 - 8 = 2$$

- Annahme: Prozesse fordern künftig ihre Ressourcen jeweils auf einen Schlag an
- Prozesse p_1 und p_3 können unter dieser Annahme nicht ausgeführt werden
- Also: Versuche, ob Ausführen von Prozess p_2 zum Ziel führt

Bankier-Algorithmus Beispiel (5)

Nachweis: Der Zustand ist unsicher

	E_i	A_i	M_i		E_i	A_i	M_i	
p_1	4	5	9	Führe Prozess p_2 bis zum Ende aus 	p_1	4	5	9
p_2	2	2	4		p_2	4	0	4
p_3	2	5	7		p_3	2	5	7

$F = 10 - 8 = 2$ $F = 10 - 10 = 0$

Bankier-Algorithmus Beispiel (5)

Nachweis: Der Zustand ist unsicher

	E _i	A _i	M _i			E _i	A _i	M _i			E _i	A _i	M _i
p ₁	4	5	9	Führe Prozess p ₂ bis zum Ende aus 	p ₁	4	5	9	Freigabe durch Prozess p ₂ 	p ₁	4	5	9
p ₂	2	2	4		p ₂	4	0	4		p ₂	0	-	-
p ₃	2	5	7		p ₃	2	5	7		p ₃	2	5	7
$F = 10 - 8 = 2$					$F = 10 - 10 = 0$					$F = 10 - 6 = 4$			

Bankier-Algorithmus Beispiel (5)

Nachweis: Der Zustand ist unsicher

	E_i	A_i	M_i		E_i	A_i	M_i		E_i	A_i	M_i		
p_1	4	5	9	Führe Prozess p_2 bis zum Ende aus	p_1	4	5	9	Freigabe durch Prozess p_2	p_1	4	5	9
p_2	2	2	4		p_2	4	0	4		p_2	0	-	-
p_3	2	5	7		p_3	2	5	7		p_3	2	5	7
$F = 10 - 8 = 2$					$F = 10 - 10 = 0$					$F = 10 - 6 = 4$			

- Es stehen nun 4 freie Ressourcen zur Verfügung
- Damit lassen sich weder Prozess p_1 noch Prozess p_3 ausführen, wenn sie laut Annahme ihre Ressourcenanforderungen sofort stellen und vor Prozessbeendigung nichts freigeben
- Also: Der Zustand ist unsicher!

Bemerkungen (1)

- Im Allgemeinen kann es mehrere Prozesse geben, die virtuell ausgeführt werden können
- Das Endergebnis (sicherer oder unsicherer Zustand) ist aber immer das Gleiche
- Nach virtueller Prozessausführung und Freigabe aller Ressourcen eines Prozesses: Es können stets nur gleich viele oder mehr Ressourcen zur Verfügung stehen

Bemerkungen (2)

- Bankier-Algorithmus führt im Allgemeinen zu quasi-paralleler Ausführung
- Die Tests auf Sicherheit von Zuständen beschränken die Ausführung allerdings
- Vorteil: Mögliche Deadlocks werden erkannt und können garantiert verhindert werden
- Allerdings sind die Annahmen konservativ und können ggf. sichere, nicht überlappende Anforderung von Betriebsmitteln verhindern

Bankier-Algorithmus

Mehrere Ressourcenklassen (1)

- Mehrere Ressourcenklassen
(z.B. verschiedene Geräte, E/A-Kanäle)
- Prozesse p_1, \dots, p_n , die Ressourcen aus Klassen K_1, \dots, K_m anfordern
- Anzahl der zur Verfügung stehenden Ressourcen aus Klasse K_k :
$$V_k \in \mathbb{N}, 1 \leq k \leq m$$
- Vektor verfügbarer Ressourcen (V_1, \dots, V_m)

Bankier-Algorithmus

Mehrere Ressourcenklassen (2)

- M_{ik} : Maximalanzahl von Ressourcen der Klasse K_k , die der Prozess p_i **anfordern** wird
- **Maximalanforderungsmatrix**

	K_1	...	K_m			
	M_{11}	M_{12}	M_{13}	...	M_{1m}	p_1
	M_{21}	M_{22}	M_{23}	...	M_{2m}	
	\vdots	\vdots	\vdots		\vdots	
	M_{n1}	M_{n2}	M_{n3}	...	M_{nm}	p_n

Zeile i gibt Maximalanforderungen von Prozess i an

Bankier-Algorithmus

Mehrere Ressourcenklassen (3)

- E_{ik} : Anzahl von Ressourcen der Klasse K_k , die der Prozess p_i zu einem bestimmten Zeitpunkt **schon erhalten** hat
- Belegungsmatrix

	K_1	...	K_m		
	E_{11}	E_{12}	E_{13} ...	E_{1m}	p_1
	E_{21}	E_{22}	E_{23} ...	E_{2m}	\vdots
	\vdots	\vdots	\vdots	\vdots	\vdots
	E_{n1}	E_{n2}	E_{n3} ...	E_{nm}	p_n

Zeile i gibt an, welche Ressourcen Prozess i schon erhalten hat

Bankier-Algorithmus

Mehrere Ressourcenklassen (4)

- A_{ik} : Anzahl von Ressourcen der Klasse K_k , die der Prozess p_i nach diesem Zeitpunkt maximal noch anfordern wird:

$$A_{ik} = M_{ik} - E_{ik}$$

- Restanforderungsmatrix

	K_1	...	K_m			
	A_{11}	A_{12}	A_{13}	...	A_{1m}	p_1
	A_{21}	A_{22}	A_{23}	...	A_{2m}	
	\vdots	\vdots	\vdots		\vdots	\vdots
	A_{n1}	A_{n2}	A_{n3}	...	A_{nm}	p_n

Zeile i gibt an, welche Ressourcen Prozess i maximal noch anfordern wird

Bankier-Algorithmus

Mehrere Ressourcenklassen (5)

- Die Anzahl der freien Ressourcen der Klasse K_k zu diesem Zeitpunkt ist

$$F_k = V_k - \sum_{i=1}^n E_{ik}$$

- Ressourcenrestvektor (F_1, \dots, F_m)
- Es gilt weiterhin:

$$\forall 1 \leq i \leq n, \forall 1 \leq k \leq m : E_{ik} \leq M_{ik} \leq V_k$$

Bankier-Algorithmus

Mehrere Ressourcenklassen (6)

- Bankier-Algorithmus für mehrere Ressourcenklassen analog zum Algorithmus für eine Ressourcenklasse
- Einziger Unterschied: Vergleich natürlicher Zahlen ersetzt durch Vergleich von Vektoren natürlicher Zahlen
- Verwendete Vergleichsoperation für zwei Vektoren (v_1, \dots, v_m) und $(w_1, \dots, w_m) \in \mathbb{N}^m$:
$$(v_1, \dots, v_m) \leq (w_1, \dots, w_m) \Leftrightarrow v_i \leq w_i \forall 1 \leq i \leq m$$

Bankier-Algorithmus

Mehrere Ressourcenklassen (7)

Überprüfung auf sicheren Zustand:

- Prüfe, ob es einen Prozess p_i gibt, dessen Anforderungen **alle mit den verfügbaren Ressourcen erfüllt** werden können
- Gibt es eine Zeile i in Anforderungsmatrix, die kleiner ist als der Ressourcenrestvektor:

$$(A_{i1}, \dots, A_{im}) \leq (F_1, \dots, F_m)$$

- Wenn ja
 - Addiere (A_{i1}, \dots, A_{im}) zu (E_{i1}, \dots, E_{im})
 - Aktualisiere den Restvektor (F_1, \dots, F_m)

$$F_k = F_k - A_{ik}$$

Bankier-Algorithmus

Mehrere Ressourcenklassen (8)

Überprüfung auf sicheren Zustand:

- Nimm an, dass p_i ausgeführt wird und danach alle seine Ressourcen freigibt

- Aktualisiere den Restvektor (F_1, \dots, F_m) d.h.

$$F_k = F_k + E_{ik}$$

- Teste, ob es nun einen anderen Prozess p_j gibt, dessen Ressourcenanforderung erfüllt wird, also

$$(A_{j1}, \dots, A_{jm}) \leq (F_1, \dots, F_m)$$

und verfare mit diesem Prozess gleichermaßen

Bankier-Algorithmus

Mehrere Ressourcenklassen (9)

- Der Zustand ist **sicher**, wenn auf diese Weise alle Prozesse „virtuell“ zu Ende gebracht werden können
- Ansonsten ist der Zustand unsicher
- Prozesse, die so nicht zu Ende geführt werden können, sind an dem **potentiellen** Deadlock beteiligt

Bankier-Algorithmus

Mehrere Ressourcenklassen (10)

Beispiel:

- Vektor verfügbarer Ressourcen $V = (4 \quad 2 \quad 3 \quad 1)$
- Maximalanforderungsmatrix M

		Ress. 1	Ress. 2	Ress. 3	Ress. 4	
	$V =$	4	2	3	1	
	$\left($	2	0	1	1	Prozess 1
		3	0	1	1	Prozess 2
	$\left. \right)$	2	2	2	0	Prozess 3
- Aktuelle Belegungsmatrix E

	$\left($	0	0	1	0	Prozess 1
		2	0	0	1	Prozess 2
		0	1	2	0	Prozess 3
	$\left. \right)$	2	1	3	1	
- Aktuelle Restanforderungsmatrix A

$$A_{ik} = M_{ik} - E_{ik}$$

	$\left($	2	0	0	1	Prozess 1
		1	0	1	0	Prozess 2
	$\left. \right)$	2	1	0	0	Prozess 3
- Ressourcenrestvektor $F = (2 \quad 1 \quad 0 \quad 0)$

$$F_k = V_k - \sum_{i=1}^n E_{ik}$$

Bankier-Algorithmus

Mehrere Ressourcenklassen (11)

Frage:

Befinden wir uns in einem **sicheren** Zustand?

Bankier-Algorithmus

Mehrere Ressourcenklassen (12)

Beispiel:

- Vektor verfügbarer Ressourcen

$$V = (4 \quad 2 \quad 3 \quad 1)$$

- Maximalanforderungsmatrix M

$$\begin{pmatrix} 2 & 0 & 1 & 1 \\ 3 & 0 & 1 & 1 \\ 2 & 2 & 2 & 0 \end{pmatrix} \begin{array}{l} \text{Prozess 1} \\ \text{Prozess 2} \\ \text{Prozess 3} \end{array}$$

- Aktuelle Belegungsmatrix E

$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{pmatrix} \begin{array}{l} \text{Prozess 1} \\ \text{Prozess 2} \\ \text{Prozess 3} \end{array}$$

- Aktuelle Restanforderungsmatrix A

$$\begin{pmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{pmatrix} \begin{array}{l} \text{Prozess 1} \\ \text{Prozess 2} \\ \text{Prozess 3} \end{array}$$

Nur 3. Zeile $A_3 = (2 \quad 1 \quad 0 \quad 0)$ der Restanforderungsmatrix ist kleiner gleich Ressourcenrestvektor $F = (2 \quad 1 \quad 0 \quad 0)$

- Ressourcenrestvektor

$$F = (2 \quad 1 \quad 0 \quad 0)$$

Bankier-Algorithmus

Mehrere Ressourcenklassen (13)

Beispiel:

- Vektor verfügbarer Ressourcen

$$V = (4 \quad 2 \quad 3 \quad 1)$$

- Maximalanforderungsmatrix M

$$\begin{pmatrix} 2 & 0 & 1 & 1 \\ 3 & 0 & 1 & 1 \\ 2 & 2 & 2 & 0 \end{pmatrix} \begin{array}{l} \text{Prozess 1} \\ \text{Prozess 2} \\ \text{Prozess 3} \end{array}$$

- Aktuelle Belegungsmatrix E

$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 2 & 2 & 2 & 0 \end{pmatrix} \begin{array}{l} \text{Prozess 1} \\ \text{Prozess 2} \\ \text{Prozess 3} \end{array}$$

- Aktuelle Restanforderungsmatrix A

virtuelle Ausführung von Prozess 3

$$\begin{pmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{array}{l} \text{Prozess 1} \\ \text{Prozess 2} \\ \text{Prozess 3} \end{array}$$

- Ressourcenrestvektor

$$F = (0 \quad 0 \quad 0 \quad 0)$$

Bankier-Algorithmus

Mehrere Ressourcenklassen (14)

Beispiel:

- Vektor verfügbarer Ressourcen

$$V = (4 \quad 2 \quad 3 \quad 1)$$

- Maximalanforderungsmatrix M

$$\begin{pmatrix} 2 & 0 & 1 & 1 \\ 3 & 0 & 1 & 1 \\ 2 & 2 & 2 & 0 \end{pmatrix} \begin{array}{l} \text{Prozess 1} \\ \text{Prozess 2} \\ \text{Prozess 3} \end{array}$$

- Aktuelle Belegungsmatrix E

$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{array}{l} \text{Prozess 1} \\ \text{Prozess 2} \\ \text{Prozess 3} \end{array}$$

- Aktuelle Restanforderungsmatrix A

$$\begin{pmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ - & - & - & - \end{pmatrix} \begin{array}{l} \text{Prozess 1} \\ \text{Prozess 2} \\ \text{Prozess 3} \end{array}$$

Ressourcenfreigabe
nach Beendigung

- Ressourcenrestvektor

$$F = (2 \quad 2 \quad 2 \quad 0)$$

Bankier-Algorithmus

Mehrere Ressourcenklassen (15)

Beispiel:

- Vektor verfügbarer Ressourcen

$$V = (4 \quad 2 \quad 3 \quad 1)$$

- Maximalanforderungsmatrix M

$$\begin{pmatrix} 2 & 0 & 1 & 1 \\ 3 & 0 & 1 & 1 \\ 2 & 2 & 2 & 0 \end{pmatrix} \begin{array}{l} \text{Prozess 1} \\ \text{Prozess 2} \\ \text{Prozess 3} \end{array}$$

- Aktuelle Belegungsmatrix E

$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{array}{l} \text{Prozess 1} \\ \text{Prozess 2} \\ \text{Prozess 3} \end{array}$$

- Aktuelle Restanforderungsmatrix A

$$\begin{pmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ - & - & - & - \end{pmatrix} \begin{array}{l} \text{Prozess 1} \\ \text{Prozess 2} \\ \text{Prozess 3} \end{array}$$

Nur 2. Zeile $A_2 = (1 \quad 0 \quad 1 \quad 0)$ der Restanforderungsmatrix kleiner gleich

Ressourcenrestvektor $F = (2 \quad 2 \quad 2 \quad 0)$

- Ressourcenrestvektor

$$F = (2 \quad 2 \quad 2 \quad 0)$$

Bankier-Algorithmus

Mehrere Ressourcenklassen (16)

Beispiel:

- Vektor verfügbarer Ressourcen

$$V = (4 \quad 2 \quad 3 \quad 1)$$

- Maximalanforderungsmatrix M

$$\begin{pmatrix} 2 & 0 & 1 & 1 \\ 3 & 0 & 1 & 1 \\ 2 & 2 & 2 & 0 \end{pmatrix} \begin{array}{l} \text{Prozess 1} \\ \text{Prozess 2} \\ \text{Prozess 3} \end{array}$$

- Aktuelle Belegungsmatrix E

$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 3 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{array}{l} \text{Prozess 1} \\ \text{Prozess 2} \\ \text{Prozess 3} \end{array}$$

- Aktuelle Restanforderungsmatrix A

virtuelle Ausführung von Prozess 2

$$\begin{pmatrix} 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ - & - & - & - \end{pmatrix} \begin{array}{l} \text{Prozess 1} \\ \text{Prozess 2} \\ \text{Prozess 3} \end{array}$$

- Ressourcenrestvektor

$$F = (1 \quad 2 \quad 1 \quad 0)$$

Bankier-Algorithmus

Mehrere Ressourcenklassen (17)

Beispiel:

- Vektor verfügbarer Ressourcen

$$V = (4 \quad 2 \quad 3 \quad 1)$$

- Maximalanforderungsmatrix M

$$\begin{pmatrix} 2 & 0 & 1 & 1 \\ 3 & 0 & 1 & 1 \\ 2 & 2 & 2 & 0 \end{pmatrix} \begin{array}{l} \text{Prozess 1} \\ \text{Prozess 2} \\ \text{Prozess 3} \end{array}$$

- Aktuelle Belegungsmatrix E

$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{array}{l} \text{Prozess 1} \\ \text{Prozess 2} \\ \text{Prozess 3} \end{array}$$

- Aktuelle Restanforderungsmatrix A

Ressourcenfreigabe
nach Beendigung

$$\begin{pmatrix} 2 & 0 & 0 & 1 \\ - & - & - & - \\ - & - & - & - \end{pmatrix} \begin{array}{l} \text{Prozess 1} \\ \text{Prozess 2} \\ \text{Prozess 3} \end{array}$$

- Ressourcenrestvektor

$$F = (4 \quad 2 \quad 2 \quad 1)$$

Bankier-Algorithmus

Mehrere Ressourcenklassen (18)

Beispiel:

- Vektor verfügbarer Ressourcen

$$V = (4 \quad 2 \quad 3 \quad 1)$$

- Maximalanforderungsmatrix M

$$\begin{pmatrix} 2 & 0 & 1 & 1 \\ 3 & 0 & 1 & 1 \\ 2 & 2 & 2 & 0 \end{pmatrix} \begin{array}{l} \text{Prozess 1} \\ \text{Prozess 2} \\ \text{Prozess 3} \end{array}$$

- Aktuelle Belegungsmatrix E

$$\begin{pmatrix} 2 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{array}{l} \text{Prozess 1} \\ \text{Prozess 2} \\ \text{Prozess 3} \end{array}$$

- Aktuelle Restanforderungsmatrix A

virtuelle Ausführung von Prozess 1

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ - & - & - & - \\ - & - & - & - \end{pmatrix} \begin{array}{l} \text{Prozess 1} \\ \text{Prozess 2} \\ \text{Prozess 3} \end{array}$$

- Ressourcenrestvektor

$$F = (2 \quad 2 \quad 2 \quad 0)$$

Bankier-Algorithmus

Mehrere Ressourcenklassen (19)

Beispiel:

- Vektor verfügbarer Ressourcen

$$V = (4 \quad 2 \quad 3 \quad 1)$$

- Maximalanforderungsmatrix M

$$\begin{pmatrix} 2 & 0 & 1 & 1 \\ 3 & 0 & 1 & 1 \\ 2 & 2 & 2 & 0 \end{pmatrix} \begin{array}{l} \text{Prozess 1} \\ \text{Prozess 2} \\ \text{Prozess 3} \end{array}$$

- Aktuelle Belegungsmatrix E

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \begin{array}{l} \text{Prozess 1} \\ \text{Prozess 2} \\ \text{Prozess 3} \end{array}$$

- Aktuelle Restanforderungsmatrix A

$$\begin{pmatrix} - & - & - & - \\ - & - & - & - \\ - & - & - & - \end{pmatrix} \begin{array}{l} \text{Prozess 1} \\ \text{Prozess 2} \\ \text{Prozess 3} \end{array}$$

Ressourcenfreigabe nach Beendigung,
alle Prozesse abgearbeitet

- Ressourcenrestvektor

$$F = (4 \quad 2 \quad 3 \quad 1)$$

Bankier-Algorithmus

Mehrere Ressourcenklassen (20)

Beispiel:

- Vektor verfügbarer Ressourcen

$$V = (4 \quad 2 \quad 3 \quad 1)$$

- Maximalanforderungsmatrix M

$$\begin{pmatrix} 2 & 0 & 1 & 1 \\ 3 & 0 & 1 & 1 \\ 2 & 2 & 2 & 0 \end{pmatrix} \begin{array}{l} \text{Prozess 1} \\ \text{Prozess 2} \\ \text{Prozess 3} \end{array}$$

- Aktuelle Belegungsmatrix E

Darf man jetzt dem Prozess 2 eine weitere Forderung nach Ressource 1 erfüllen?

$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 2 & 0 \end{pmatrix} \begin{array}{l} \text{Prozess 1} \\ \text{Prozess 2} \\ \text{Prozess 3} \end{array}$$

- Aktuelle Restanforderungsmatrix A

$$\begin{pmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{pmatrix} \begin{array}{l} \text{Prozess 1} \\ \text{Prozess 2} \\ \text{Prozess 3} \end{array}$$

- Ressourcenrestvektor

$$F = (2 \quad 1 \quad 0 \quad 0)$$

Bankier-Algorithmus

Mehrere Ressourcenklassen (21)

Beispiel:

- Vektor verfügbarer Ressourcen

$$V = (4 \quad 2 \quad 3 \quad 1)$$

- Maximalanforderungsmatrix M

	Ress. 1	Ress. 2	Ress. 3	Ress. 4	
2	0	1	1	1	Prozess 1
3	0	1	1	1	Prozess 2
2	2	2	0	0	Prozess 3

- Aktuelle Belegungsmatrix E

Darf man jetzt dem Prozess 2 eine weitere Forderung nach Ressource 1 erfüllen?

0	0	1	0	1	Prozess 1
3	0	0	1	1	Prozess 2
0	1	2	0	0	Prozess 3

- Aktuelle Restanforderungsmatrix A

Nein! Das führt zu einem unsicheren Zustand

2	0	0	1	1	Prozess 1
0	0	1	0	1	Prozess 2
2	1	0	0	0	Prozess 3

- Ressourcenrestvektor

$$F = (1 \quad 1 \quad 0 \quad 0)$$

Bankier-Algorithmus – Analyse (1)

- Hauptvorteil: Mögliche Deadlocks werden erkannt und können garantiert verhindert werden
- Ist mit dem Bankier-Algorithmus das Deadlock-Problem restlos gelöst?

Bankier-Algorithmus – Analyse (2)

Leider nein

- Prozesse können meist nicht im Voraus eine verlässliche Obergrenze für ihre Ressourcenanforderungen geben
- Garantierte Obergrenzen würden häufig die Anzahl der verfügbaren Ressourcen übersteigen
- Anzahl der Prozesse ist nicht fest, sondern ändert sich ständig

Lösung durch Aufhebung der Deadlock-Voraussetzungen

- Versuche, die Bedingungen ganz oder teilweise außer Kraft zu setzen, um Deadlocks zu vermeiden
 1. **Wechselseitiger Ausschluss**: Jede Ressource ist entweder verfügbar oder genau einem Prozess zugeordnet
 2. **Besitzen und Warten**: Prozesse, die schon Ressourcen reserviert haben, können noch weitere Ressourcen anfordern
 3. **Kein Ressourcenentzug**: Ressourcen, die einem Prozess bewilligt wurden, können nicht gewaltsam wieder entzogen werden
 4. **Zyklisches Warten**: Es gibt eine zyklische Kette von Prozessen, von denen jeder auf eine Ressource wartet, die dem nächsten Prozess in der Kette gehört
- Reale Systeme tun genau das

Wechselseitiger Ausschluss

- Braucht man meistens, z.B. Schreiben in Datei
- In gewissen Situationen lässt sich wechselseitiger Ausschluss aber einschränken
- Beispiel: Drucken mit Spooling-Verzeichnis und Drucker-Dämon, der als einziger den Drucker reserviert und Dateien aus dem Spooling-Verzeichnis der Reihe nach druckt

Besitzen und Warten

- Prozesse müssen benötigte Ressourcen **immer auf einmal und im Voraus** anfordern
- Funktioniert nicht, Prozesse kennen Ressourcenbedarf nicht im Voraus
- Und außerdem ineffizient, da Prozesse u.U. lange aufgehalten werden und Ressourcen unnötig lange belegt sind

Kein Ressourcenentzug

- Ressourcenentzug und Unterbrechung schwer zu realisieren
- Status **muss gespeichert und wiederhergestellt** werden
- Oft nicht möglich

Zyklisches Warten (1)

Lösungsansatz

- Ressourcen werden **durchnummeriert**
- Ressourcenanforderungen dürfen nur in **aufsteigender** Reihenfolge erfolgen
- Damit wird ein Deadlock verhindert

Zyklisches Warten (2)

- Ressource R_i vor Ressource R_j , wenn $i < j$
- Annahme: Prozesse A und B verklemmt: A besitzt R_i und hat R_j angefordert, B besitzt R_j und hat R_i angefordert
- Dies ist nicht möglich, weil dann $i < j$ und $j < i$
- Also: Belegungs-Anforderungsgraph kann nicht zyklisch werden
- Aber schwierig, eine Ordnung zu finden
- Kann zu ineffizientem Verhalten führen

Schlussfolgerungen (1)

- Deadlock-Verhinderung ist schwierig
- Deadlock-Freiheit kann zwar prinzipiell erreicht werden, führt jedoch häufig zu starken Effizienzverlusten
- In der Praxis verzichtet man daher auf absolute Garantien für Deadlock-Freiheit

Schlussfolgerungen (2)

- Die meisten Betriebssysteme ignorieren das Deadlock-Problem
- Systeme werden nicht an ihrem Ressourcenlimit betrieben
- Häufig angewendet werden das Spooling und die Anforderung gemäß einer vorgegebenen Ordnung

Beheben von Deadlocks

- Am weitesten verbreitet: Abbruch aller verklemmten Prozesse
- Rückführung aller verklemmten Prozesse auf einen festgelegten Kontrollpunkt und Neustart
- Schrittweiser Abbruch aller verklemmten Prozesse, bis die Verklemmung nicht mehr existiert
- Schrittweiser Ressourcenentzug, bis die Verklemmung nicht mehr existiert
(Zurückführung eines/mehrerer Prozesse auf Zeitpunkt vor Ressourcenerhalt)

Zusammenfassung

- Ressourcen werden nach und nach von den Prozessen angefordert
- Es kann passieren, dass eine Menge von Prozessen sich in einem Deadlock befindet
- Belegungs-Anforderungsgraph und Ressourcendiagramm zum Erkennen von Deadlock-Situationen
- Bankier-Algorithmus: Ressourcenaufteilung, so dass Deadlocks verhindert werden