Deadlock

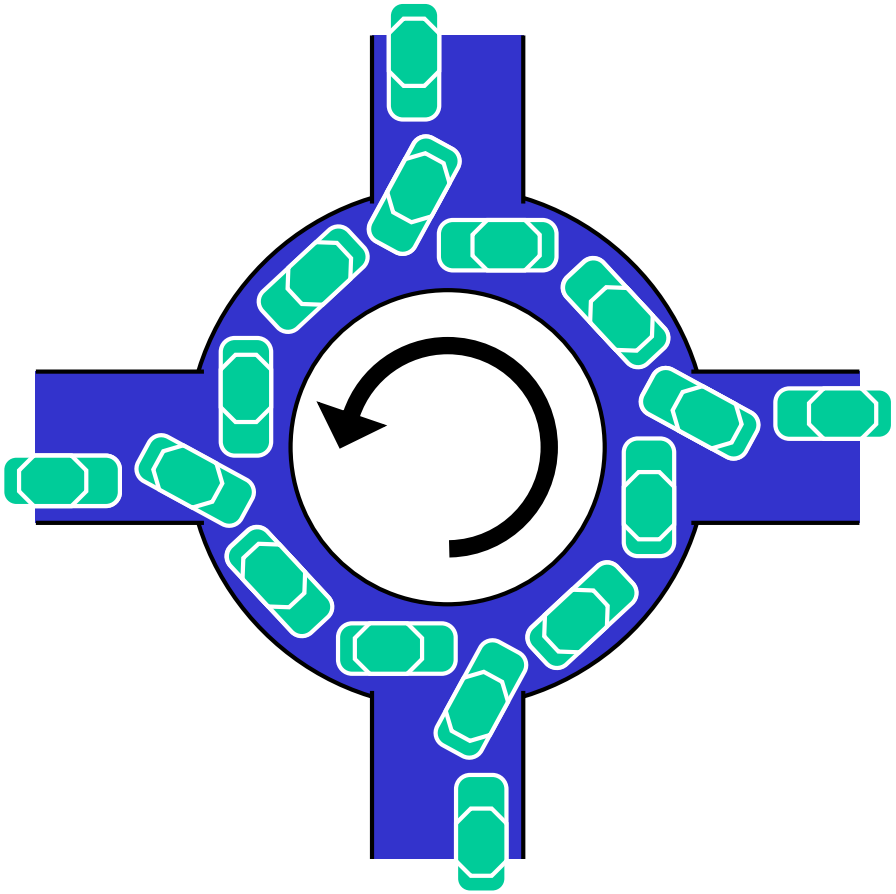
Peter Puschner
Institut für Technische Informatik
peter@vmars.tuwien.ac.at

Deadlock

- Permanentes Blockieren einer Menge von Prozessen, die um Ressourcen konkurrieren oder miteinander kommunizieren
- Zyklischer Ressourcenkonflikt zwischen zwei oder mehreren Prozessen:
  1. jeder Prozess hält eine Ressource und
  2. wartet auf eine Ressource, die gerade ein anderer Prozess hält
- erneuerbare vs. konsumierbare Ressourcen
- Es gibt keine universelle Lösung des Problems

Beispiel “Kreisverkehr”

- Kreisverkehr mit Rechtsvorrang
- Autos fahren nur vorwärts



Autos ⇔ Prozesse

Fahrbahnstücke ⇔ Ressourcen

Deadlock: jedes Auto blockiert ein Fahrbahnstück und möchte auf das Fahrbahnstück des vorderen Autos

Beispiel Nachrichten

Synchronisation über Nachrichten (konsumierbare Ressourcen)

P1:

```
   …
receive(MQ1, msg);
   …
send(MQ2, msg);
   ...
```

P2:

```
   …
receive(MQ2, msg);
   …
send(MQ1, msg);
   ...
```

Beispiel Mutual Exclusion

Prozesse greifen auf zwei gemeinsame Ressourcen zu; Mutual Exclusion

P1:

```
...
Get B
...
Get A
...
Release B
...
Release A
...
```

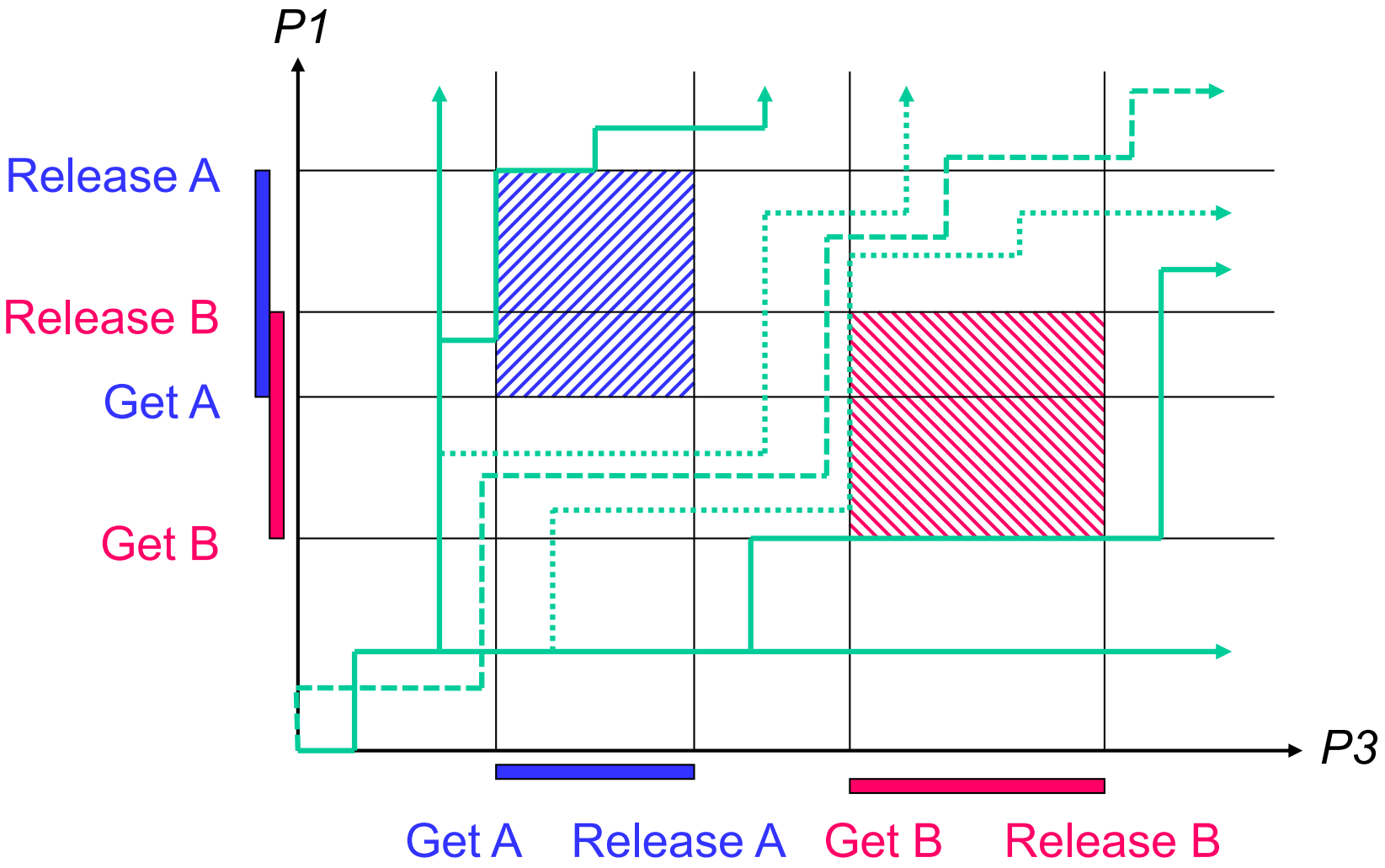
P2:

```
...
Get A
...
Get B
...
Release B
...
Release A
...
```

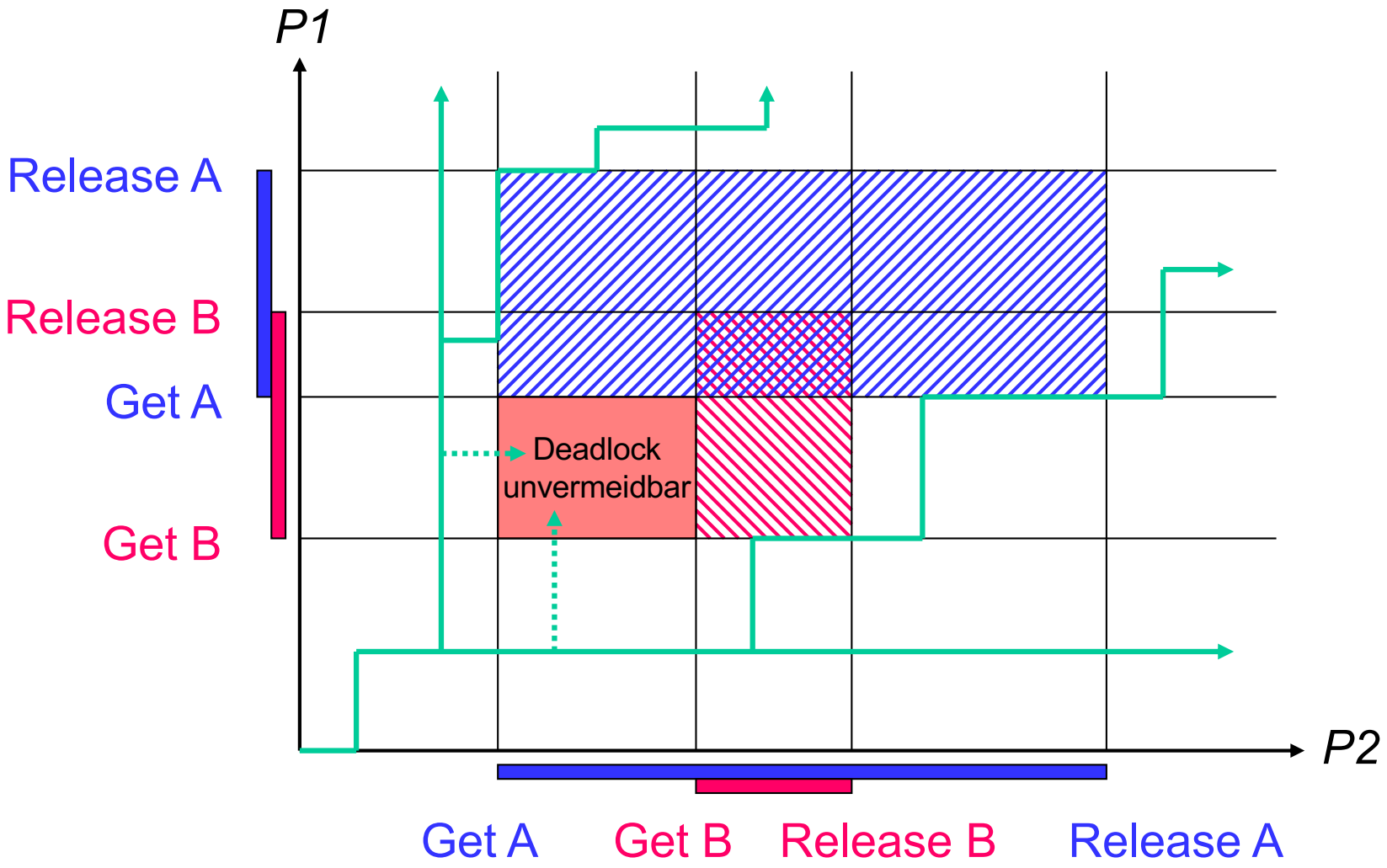
P3:

```
...
Get A
...
Release A
...
Get B
...
Release B
...
```

Prozessfortschrittsdiagramm



Prozessfortschrittsdiagramm



Die 4 Deadlock-Bedingungen

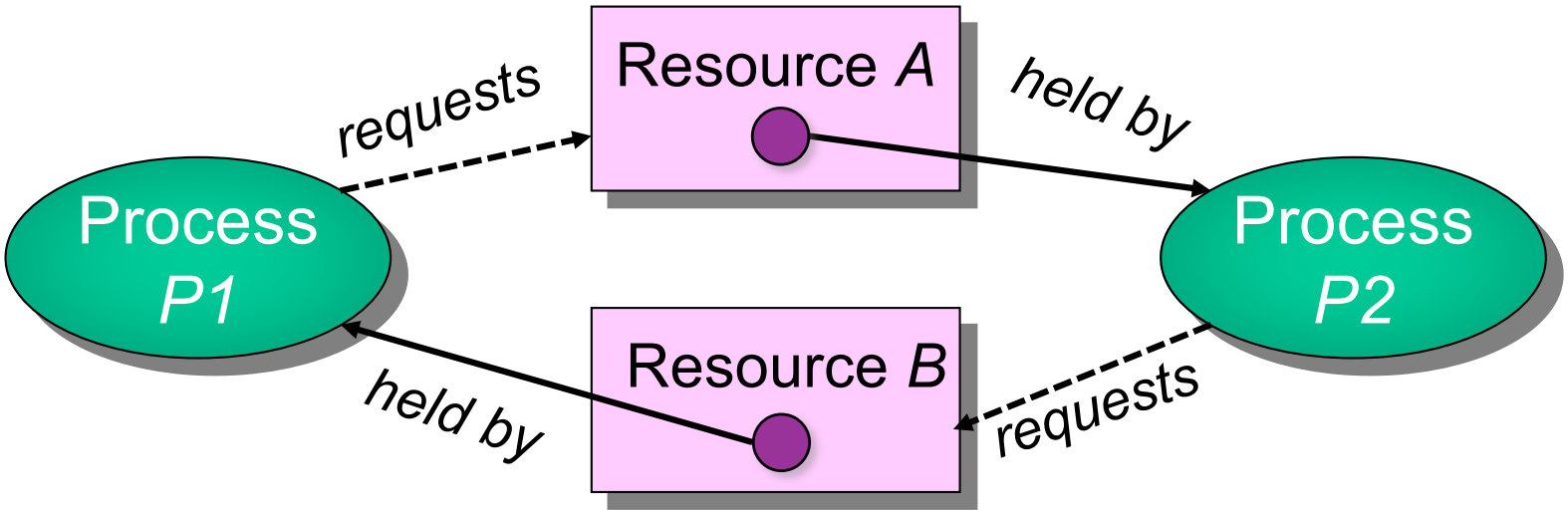
Drei Voraussetzungen des Systems

1. *Mutual Exclusion*
   - exklusiver Zugriff auf Ressourcen
2. *Hold and Wait*
   - Prozess kann Ressourcen halten, während er auf andere Ressourcen wartet
3. *No Preemption*
   - zugewiesene Ressourcen werden den Prozessen nicht weggenommen

Die 4 Deadlock-Bedingungen

Auftreten einer bestimmten Ereignisfolge

4. *Circular Wait*
   - Geschlossene Kette von Prozessen, von denen jeder Prozess mindestens eine Ressource hält, die von einem anderen Prozess benötigt wird



Deadlock-Bedingungen

- Ein Deadlock tritt genau dann auf, wenn das *Circular Wait* nicht aufgelöst werden kann
- *Circular Wait* kann nicht aufgelöst werden, wenn Bedingungen 1 bis 3 gelten

⇨ Bedingungen 1 bis 4 sind notwendig und hinreichend für das Auftreten eines Deadlocks

Behandlung von Deadlocks

- *Deadlock Prevention*
  - verhindern einer der 4 Deadlock-Bedingungen
- *Deadlock Avoidance*
  - Ressourcenreservierungen, die zu Deadlock führen könnten, werden nicht gewährt
- *Deadlock Detection*
  - Ressourcenanforderungen werden immer gewährt, sofern Ressourcen vorhanden sind
  - Periodisches Überprüfen, ob ein Deadlock vorliegt und ggf. Recovery

Deadlock Prevention

- Betriebssystemdesign, das Deadlocks ausschließt

- *Indirect Deadlock Prevention*
  - Verhinderung einer der Bedingungen 1 bis 3
- *Direct Deadlock Prevention*
  - Verhinderung des *Circular Wait*

Indirect Deadlock Prevention

- Mutual Exclusion
  - Zielsetzung; kann nicht unterbunden werden
- Hold and Wait
  - Prozesse fordern alle Ressourcen auf einmal an
  - Blockieren, bis alle Ressourcen vorhanden sind

  ⇨ Lange Verzögerungen der Prozesse möglich

  ⇨ Schlechte Nutzung allokierter Ressourcen

  ⇨ Prozess braucht Wissen über Ressourcen, die er verwenden wird

Indirect Deadlock Prevention

- No Preemption

  a) Prozess gibt Ressourcen frei, wenn er eine weitere Ressource nicht bekommt;
  fordert Ressourcen zu einem späteren Zeitpunkt wieder an

  b) Anforderung eines Prozesses führt dazu, dass ein anderer Prozess Ressourcen freigeben muss

  Anwendbar für Ressourcen, deren Zustand leicht gespeichert und später wieder hergestellt werden kann (z.B. Prozessor, beim Blockieren bei *wait*)

Direct Deadlock Prevention

- Protokoll: Verhinderung des *Circular Wait*
  - Strikte lineare Ordnung O für Ressourcenarten, z.B. O(Tape Drives) = 2, O(Disk Drives) = 4
  - Erste Anforderung: Prozess fordert eine Anzahl von Ressourcen der Art R_i an (alle müssen in einem Schritt angefordert werden)
  - In der Folge werden für den Prozess nur Anforderungen von Ressourcen der Art R_k mit $O(R_k) > O(R_i)$ zugelassen

Verhinderung des Circular Wait

ind. Beweis: das Protokoll verhindert *Circular Wait*

- Ann: es gibt ein Circular Wait von $P_0 \ldots P_n$
  d.h.: für alle i gilt:
  P_i belegt R_i und wartet auf $R_{(i+1) \bmod n}$
- Entsprechend dem Protokoll gilt daher:
  $O(R_0) < O(R_1) < \ldots < O(R_n) < O(R_0)$
  ⇨ lt. Protokoll unmöglich!

- Das Protokoll verhindert Deadlock
- Ineffizienz: unnötiges Zurückweisen von Anforderungen aufgrund der Ordnung

Deadlock Avoidance

- Bedingungen 1 bis 3 erlaubt, selektives Vergeben von Ressourcen
  - *Process Initiation Denial*: ein Prozess wird nicht gestartet, wenn seine Anforderungen zu einem Deadlock führen könnten
  - *Resource Allocation Denial*: eine Ressourcen-anforderung eines Prozesses wird verwehrt, wenn dadurch ein Deadlock entstehen könnte
- Höhere Parallelität als Deadlock Prevention
- Voraussetzung: Ressourcenbedarf der Prozesse muss bekannt sein

Deadlock Avoidance Notation

- *n* Prozesse
- *m* Ressourcenkategorien
- Vektoren
  - *Resource* = (R_1, R_2, …, R_m)
    Gesamtanzahl der Ressourcen im System
  - *Available* = (V_1, V_2, …, V_m)
    Anzahl der nicht vergebenen Ressourcen

Deadlock Avoidance Notation

- Matrizen
  - *Claim* = $\begin{pmatrix} C_{11}, C_{12}, \ldots, C_{1m} \\ \ldots \\ C_{n1}, C_{n2}, \ldots, C_{nm} \end{pmatrix}$ Maximal-anforderungen an Ressourcen
  - *Allocation* = $\begin{pmatrix} A_{11}, A_{12}, \ldots, A_{1m} \\ \ldots \\ A_{n1}, A_{n2}, \ldots, A_{nm} \end{pmatrix}$ gegenwärtige Ressourcen-belegung

Process Initiation Denial

- Ein Prozess P_{n+1} wird nur gestartet, wenn seine Ressourcenanforderungen keinen Deadlock hervorrufen können, d.h. wenn:

$$R_i >= C_{(n+1)i} + \sum_{k=1}^{n} C_{ki}$$

Nachteil: Annahme, dass alle Prozesse ihre Ressourcen gleichzeitig anfordern

⇨ Einschränkung der Parallelität

Resource Allocation Denial - Banker´s Algorithm

- *State*: Ressourcenbelegung der Prozesse, charakterisiert durch Vektoren u. Matrizen
- *Safe State*: es existiert eine Folge von Schritten, mit der der Algorithmus alle Prozesse als “finished” markiert in den Endzustand bringt.
- *Unsafe State*: es existiert keine solche Schrittfolge.

Banker´s Algorithm - Notation

- Zusätzliche Matrix
  - *Required* = $\begin{pmatrix} N_{11}, N_{12}, \ldots, N_{1m} \\ \ldots \\ N_{n1}, N_{n2}, \ldots, N_{nm} \end{pmatrix}$ $N_{ki} = C_{ki} - A_{ki}$

Vor einer Ressourcenzuweisung: Test, ob die Zuweisung zu Safe State führt

Safe State ⇨ Zuteilung der Ressourcen
Unsafe State ⇨ keine Ressourcenzuweisung

Banker´s Algorithm

Initialisierung:

markiere alle Prozesse als „unfinished“

Work Vector $W_i = V_i$, für alle i

loop

suche P_k mit *unfinished* (P_k) und $N_{ki} <= W_i$ für alle i

kein Prozess gefunden ⇨ goto END

andernfalls ⇨ markiere Prozess als „finished“ und

gib seine Ressourcen frei: $W_i = W_i + A_{ki}$ für alle i

end loop

END: alle Prozesse mit „finished“ markiert ⇨ Safe State

sonst ⇨ Unsafe State

Banker´s Alg. - Anwendung

- Test, ob aktuelle Ressourcenanforderungen Q_{ki} von Prozess P_k erfüllt werden sollen

1. Test: Q_{ki} <= N_{ki} für alle i
   - true ⇨ weiter
   - false ⇨ Fehler (Anforderung zu hoch)
2. Test: Q_{ki} <= V_i für alle i
   - true ⇨ weiter
   - false ⇨ Warte (Ressourcen nicht verfügbar)
3. Provisorisches „Gewähren“ der Anforderung und Untersuchen auf Safe State (siehe Fortsetzung)

Banker´s Alg. - Anwendung

Fortsetzung 3.

für alle *i*:

$V_i := V_i - Q_{ki}$

$A_{ki} := A_{ki} + Q_{ki}$

$N_{ki} := N_{ki} - Q_{ki}$

4. Test: Ist der vorliegende State ein Safe State?

true ⇨ weise Ressourcen Q_{ki} an P_k zu

false ⇨ Aufschieben des Requests

Banker´s Algorithm - Beispiele

- 4 Prozesse, 3 Ressourcenklassen
- Ausgangszustand:

$$R = (9\ \ 3\ \ 6)$$

$$C = \begin{pmatrix} 3 & 2 & 2 \\ 6 & 1 & 3 \\ 3 & 1 & 4 \\ 4 & 2 & 2 \end{pmatrix} \quad A = \begin{pmatrix} 1 & 0 & 0 \\ 5 & 1 & 1 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{pmatrix} \quad V = (1\ \ 1\ \ 2)$$

Banker´s Algorithm - Beispiel1

- Anforderung durch P_2: $Q_2 = (1\ \ 0\ \ 1)$

$$C = \begin{pmatrix} 3 & 2 & 2 \\ 6 & 1 & 3 \\ 3 & 1 & 4 \\ 4 & 2 & 2 \end{pmatrix} \quad A = \begin{pmatrix} 1 & 0 & 0 \\ 6 & 1 & 2 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{pmatrix} \quad V = (0\ \ 1\ \ 1)$$

Safe State: Abarbeitungsfolge P_2, P_1, P_3, P_4 erlaubt Fertigstellung aller Prozesse (nach Ausführung von P_2 ist $W = (6\ \ 2\ \ 3)$)
Ressourcenzuweisung an P_2 wird gewährt

R: (9 3 6)

V: (1 1 2)

Q_2 : (1 0 1)

C: $\begin{pmatrix} 3 & 2 & 2 \\ 6 & 1 & 3 \\ 3 & 1 & 4 \\ 4 & 2 & 2 \end{pmatrix}$ A: $\begin{pmatrix} 1 & 0 & 0 \\ 5 & 1 & 1 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{pmatrix}$ N: $\begin{pmatrix} 2 & 2 & 2 \\ 1 & 0 & 2 \\ 1 & 0 & 3 \\ 4 & 2 & 0 \end{pmatrix}$

W: (0 1 1)

C: $\begin{pmatrix} 3 & 2 & 2 \\ 6 & 1 & 3 \\ 3 & 1 & 4 \\ 4 & 2 & 2 \end{pmatrix}$ A: $\begin{pmatrix} 1 & 0 & 0 \\ 6 & 1 & 2 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{pmatrix}$ N: $\begin{pmatrix} 2 & 2 & 2 \\ 0 & 0 & 1 \\ 1 & 0 & 3 \\ 4 & 2 & 0 \end{pmatrix}$ ⬅

W: (6 2 3)

C: $\begin{pmatrix} 3 & 2 & 2 \\ 6 & 1 & 3 \\ 3 & 1 & 4 \\ 4 & 2 & 2 \end{pmatrix}$ A: $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{pmatrix}$ N: $\begin{pmatrix} 2 & 2 & 2 \\ 0 & 0 & 0 \\ 1 & 0 & 3 \\ 4 & 2 & 0 \end{pmatrix}$ ⬅

W: (7 2 3)

C: $\begin{pmatrix} 3 & 2 & 2 \\ 6 & 1 & 3 \\ 3 & 1 & 4 \\ 4 & 2 & 2 \end{pmatrix}$ A: $\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{pmatrix}$ N: $\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 0 & 3 \\ 4 & 2 & 0 \end{pmatrix}$ ⬅ ...

Banker´s Algorithm - Beispiel2

- Anforderung durch P_1: $Q_1 = (1\ \ 0\ \ 1)$

$$C = \begin{pmatrix} 3 & 2 & 2 \\ 6 & 1 & 3 \\ 3 & 1 & 4 \\ 4 & 2 & 2 \end{pmatrix} \quad A = \begin{pmatrix} 2 & 0 & 1 \\ 5 & 1 & 1 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{pmatrix} \quad V = (0\ \ 1\ \ 1)$$

Unsafe State: Jeder Prozess benötigt noch mindestens eine Einheit von Ressource R_1
keine Zuweisung der Ressourcen an P_1

R: (9 3 6)

V: (1 1 2)

Q_1 : (1 0 1)

C: $\begin{pmatrix} 3 & 2 & 2 \\ 6 & 1 & 3 \\ 3 & 1 & 4 \\ 4 & 2 & 2 \end{pmatrix}$ A: $\begin{pmatrix} 1 & 0 & 0 \\ 5 & 1 & 1 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{pmatrix}$ N: $\begin{pmatrix} 2 & 2 & 2 \\ 1 & 0 & 2 \\ 1 & 0 & 3 \\ 4 & 2 & 0 \end{pmatrix}$

W: (0 1 1)

C: $\begin{pmatrix} 3 & 2 & 2 \\ 6 & 1 & 3 \\ 3 & 1 & 4 \\ 4 & 2 & 2 \end{pmatrix}$ A: $\begin{pmatrix} 2 & 0 & 1 \\ 5 & 1 & 1 \\ 2 & 1 & 1 \\ 0 & 0 & 2 \end{pmatrix}$ N: $\begin{pmatrix} 1 & 2 & 1 \\ 1 & 0 & 2 \\ 1 & 0 & 3 \\ 4 & 2 & 0 \end{pmatrix}$

Unsafe State !!!

Banker´s Alg. - Bemerkungen

- Safe State ⇨ kein Deadlock möglich
- Unsafe State ⇨ Deadlock möglich, muss aber nicht eintreten
  - Prozesse benötigen Ressourcen nicht während der gesamten Ausführungsdauer
  - Maximalanforderungen in den Ressourcenkategorien treten nicht gemeinsam auf
- Strategien zur Deadlock Avoidance nehmen Unabhängigkeit der Prozesse an (keine Bedingungssynchronisation zwischen Prozessen)

Deadlock Detection

- Ressourcenanforderungen werden immer gewährt, sofern Ressourcen vorhanden sind
- Notwendige BS-Vorkehrungen
  - Algorithmus, um Deadlock zu erkennen
  - Strategie zur Deadlockbehebung (Recovery)
- Deadlocküberprüfung z.B. bei jeder Ressourcenanforderung ⇨ CPU-intensiv

Deadlock Detection Algorithm

- Markieren der deadlock-freien Prozesse

Initialisierung:
    setze alle Prozesse P_k auf unmarkiert

markiere alle Prozesse P_k mit $A_{ki} = 0$ für alle i
setze $W_i := V_i$ für alle i
loop
    suche unmarkierten P_k mit $Q_{ki} <= W_i$ für alle i
    Test: existiert so ein unmarkierter Prozess P_k?
        true ⇨ markiere P_k ; setze $W_i := W_i + A_{ki}$ für alle i
        false ⇨ **exit loop**
end loop
-- unmarkierte Prozesse befinden sich in einem Deadlock

Deadlock Detection - Beispiel

$$Q = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 \end{pmatrix} \quad A = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad V = (0\ \ 0\ \ 0\ \ 0\ \ 1)$$

markiere P_4 (belegt keine Ressourcen)

W:= (0 0 0 0 1)

Q_{3i} (Anforderung von P_3) <= W_i , daher markiere P3 und setze W := W + (0 0 0 1 0) → (0 0 0 1 1)

Algorithmus terminiert, liefert Deadlock von P_1 und P_2

Deadl. Detection - Bemerkungen

- Optimistische Annahme, dass Prozesse keine zusätzlichen Ressourcen für ihre Fertigstellung benötigen
- Wenn diese Annahme nicht stimmt, kann es später zu einem Deadlock kommen
  - ⇨ dieser Deadlock wird beim nächsten Aufruf des Deadlock Detection Algorithmus erkannt

Deadlock Recovery

Auflösen des erkannten Deadlocks

- *Abbrechen aller* betroffenen Prozesse (häufig angewandte Strategie)
- *Rollback aller* beteiligten Prozesse bis zu einem definierten Aufsetzpunkt (Deadlock kann sich wiederholen)
- *Abbrechen einzelner* beteiligter Prozesse, solange bis der Deadlock beseitigt ist (wiederholtes Aufrufen des Detection Alg.)

Deadlock Recovery (2)

- *Ressourcen* werden Prozessen *schrittweise entzogen* und an andere Prozesse vergeben, bis kein Deadlock mehr vorliegt
  - Rücksetzen der Prozesse, denen Ressourcen entzogen werden, bis zum Punkt der Zuweisung

Auswahl des zu unterbrechenden Prozesses:

- wenigste bisher verbrauchte CPU-Zeit?
- geringste Anzahl an bisher belegten Ressourcen?
- geringster Fortschritt?

Integrierte Deadlock-Strategie

- Kombinieren der genannten Ansätze
  - Gruppieren der Ressourcen in Klassen und Ordnen der Klassen
    - Swappable Space (Sekundärspeicher)
    - Prozessressourcen (I/O-Geräte, Files, etc.)
    - Hauptspeicher
  - Circular Wait Prevention zwischen Klassen
  - Verwendung der best-geeigneten Strategie in den einzelnen Klassen
    (z.B. Hauptspeicher - Prevention: Preemption, Processressourcen - Avoidance)

Zusammenfassung

- 4 Bedingungen:
  - Mutual Exclusion
  - Hold and Wait
  - No Preemption
  - Circular Wait
- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection und Recovery