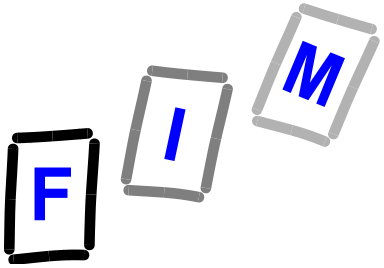
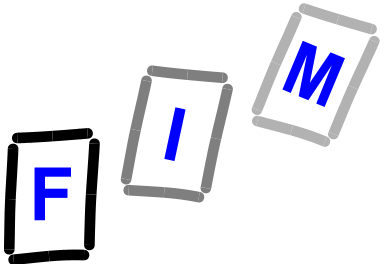


Betriebssysteme

H:

Deadlocks - Systemverklemmungen

(TeilB: Methoden zur Behandlung)

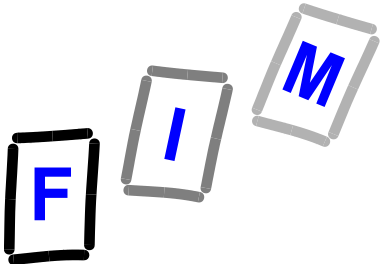


Deadlock-Behandlung

Im Prinzip gibt es 3 Ansätze

1) **Verwende ein Protokoll, welches garantiert, dass ein Deadlock nicht auftreten kann**
   - ➡ **Deadlock vorbeugen (deadlock prevention)**
     - » **Von vornherein unmöglich**
   - ➡ **Deadlock vermeiden (deadlock avoidance)**
     - » **So arbeiten, dass Deadlock-Gefahr erkannt wird**
2) **Ein Deadlock darf eintreten, dann erfolgt aber (automatisch) eine Befreiung davon**

   Erkennen und Wiederherstellen (detection and recovery)
3) **Gar nichts tun (!?)**



Methode 1

Vverwende ein Protokoll,
welches garantiert,
dass ein Deadlock
nicht auftreten kann

Deadlock prevention (1) Grundidee:

- **Die „Coffman- Bedingungen“**
  (1)-(4) sind *notwendige* Bedingungen
  - **(1) Mutual Exclusion**
  - **(2) Hold and Wait**
  - **(3) No Preemption**
  - **(4) Circular Wait**

- **Verwende ein Protokoll, sodass eine Bedingung (und damit alle 4 gleichzeitig) nie eintreten kann**

  insbesondere

- **verhindere direkt zirkuläres Warten (4)**

Deadlock prevention (2)
Hold and Wait sei nicht erlaubt

Verwende eine statische Betriebsmittel-Zuteilungsstrategie

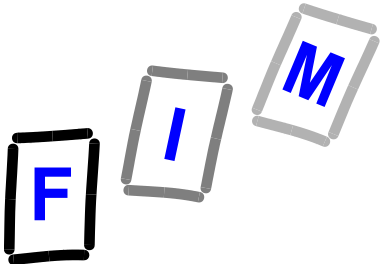
- ➡ **Jeder Prozess muss ALLE seine Betriebsmittel, die er braucht / brauchen könnte, im Voraus zu Prozessbeginn anfordern**
- ➡ **Also unabhängig davon, ob alle BM auch tatsächlich benötigt werden**

= Klar: Hold and Wait kann nicht eintreten

- Nachteil:

Flexible & dynamische BM-Zuteilung ist nicht mehr möglich

Man vgl.: Drucker, der vielleicht zu Prozessbeginn und am Schluss, oder auch gar nicht benötigt wird



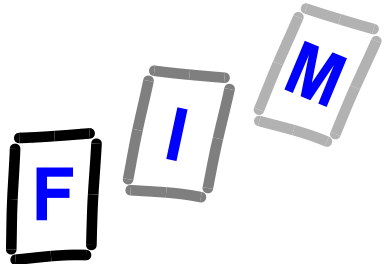
Deadlock prevention (3)
BM-Entzug erlauben

Vorgangsweise:

- **Angenommen: P hält R1, möchte R2**
- **Dann: R1 muss freigegeben werden (d.h. R1 wird entzogen) während P auf R2 wartet.**
- **Status von R1 muss gerettet werden, R1 wird in die Liste der Ressourcen eingehängt, auf die P wartet**

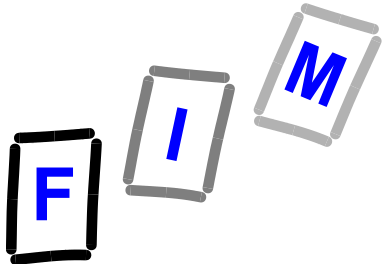
Methode nur anwendbar bei Ressourcen, deren Zustand gesichert werden kann und nachher wieder leicht restaurierbar ist

CPU, Puffer, Hauptspeicherbereiche, ...



Deadlock prevention (4) verhindere Circular Wait

- Definiere eine lineare Ordnung auf alle Betriebsmittel aller Typen (Klassen) R_i:
  $R_1 < R_2 < \ldots < R_m$
- Beachte: R_i ist eine Klasse von Ressourcen R_i
- Spezialfall:
  R_i enthält genau 1 Element



Deadlock prevention (5)
Verhindere Circular Wait

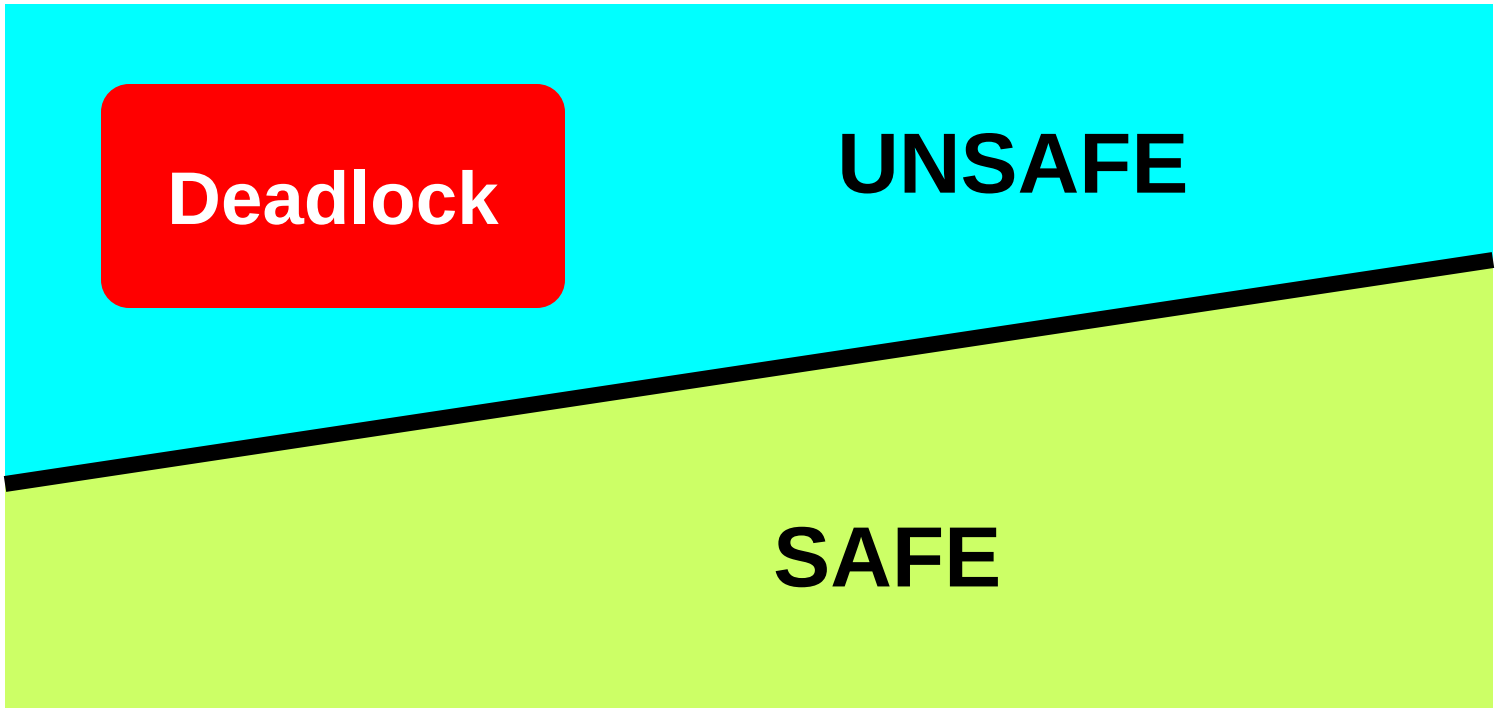
- **Also gegeben:**
  $R_1 < R_2 < \ldots < R_m$

- **Alle BM in einer Klasse müssen in einer einzigen Anforderung belegt werden**
- **Sobald P (die) Ressourcen der Klasse R_i, angefordert hat, können nur mehr Ressourcen von Klassen $R_k > R_i$ *belegt werden***
- **Umgekehrt: P muss R_k *freigeben*, ehe R_i mit $R_k > R_i$ angefordert werden kann**

Deadlock avoidance (1) Grundidee

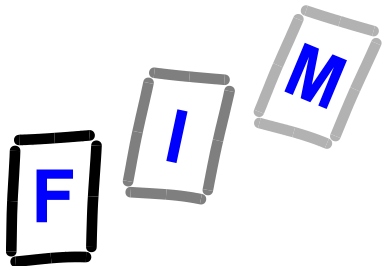
- **Ein Zustand heißt sicher (safe), wenn ein Deadlock nicht eintreten kann oder vermieden werden kann**
- **Ein Zustand heißt unsicher (unsafe), wenn ein Deadlock passieren kann und keine Zustandsfolge zur Verhinderung existiert**
  - ➡ **Ob ein Deadlock eintritt oder nicht, hängt von der ((un)glücklichen) Verzahnung der Operationen ab, zeitabhängiges Ereignis!**
  - ➡ ***Unsafe* heißt: Deadlock kann eintreten, aber wir haben keine Information darüber, ob das tatsächlich passieren wird oder nicht!**

Deadlock vermeiden (2) Schema



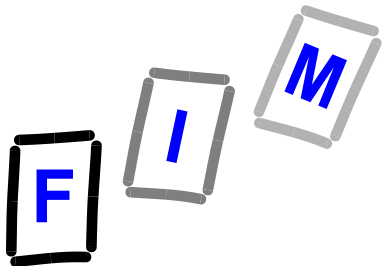
Deadlock vermeiden (3)

- Gehe von einem sicheren Zustand aus
- Prüfe, ob eine Ressourcenanforderung in einen unsicheren Zustand führen würde
- Wenn nein, dann gestatte die Ressourcenbelegung
- Andernfalls werde sie nicht erlaubt



Beispiel: Zuteilung von Plattenlaufwerken (1)

- Seien 12 Plattenlaufwerke D vorhanden
  - ➡ Jedes muss unter mutual exclusion zugeteilt werden
- Seien 3 Prozesse gegeben P0, P1, P2
  - ➡ Jeder gibt seinen maximalen Bedarf bekannt P0: 10; P1: 4; P2: 9
- Diese Situation ist sicher, denn z.B.:
  - ➡ P0 erhält 10 Ds zuerst, verwendet sie und gibt sie dann alle zusammen frei
  - ➡ Dann kommt P2 dran ...



Beispiel: Zuteilung von Plattenlaufwerken (2)

- **Dieser Zustand ist *sicher*:**

	max Bedarf	momentane Zuordnung	
P0	10	5	5
P1	4	2	2
P2	9	2	7
		9	

9 Ds sind in Verwendung
3 noch frei

P1 bekommt 2, gibt 4 zurück, ist dann fertig, es sind dann 5 Ds frei, jetzt kann P0 5 bekommen, ...

Beispiel: Zuteilung von Plattenlaufwerken (3)

- Wie ist dieser Zustand: safe / unsafe????

	max Bedarf	momentane Zuordnung
P0	10	5
P1	4	2
P2	9	3
		10

5 10 Ds sind in Verwendung

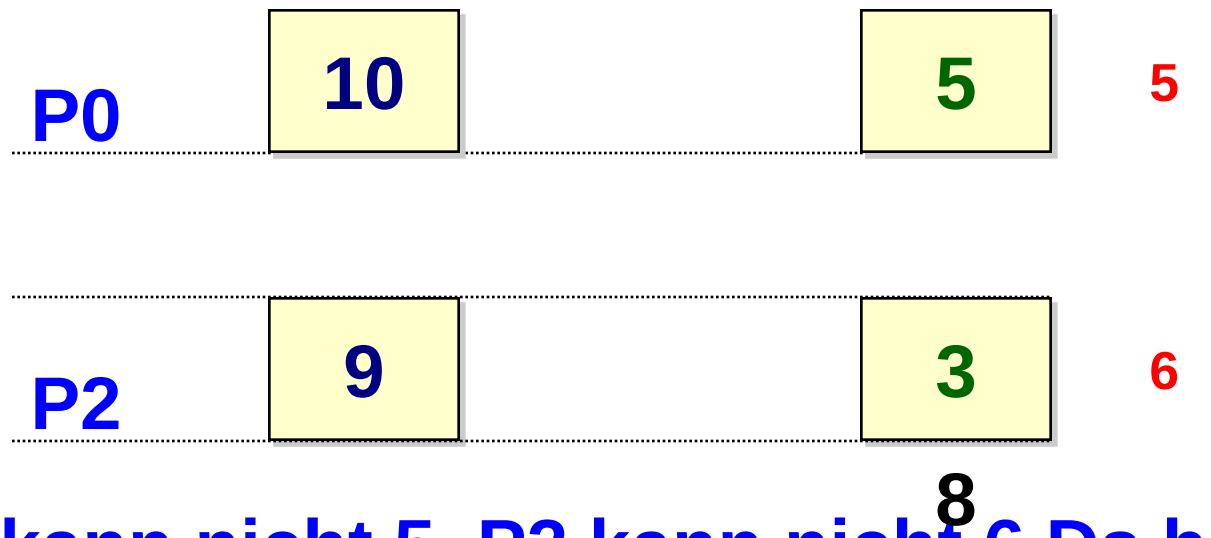
2 2 noch frei

6

P0 kann nicht 5 anfordern, wenn er sie braucht
P2 kann nicht 6 anfordern
P1 kann 2 bekommen, gibt 4 zurück, worauf aber nur 4 verbleiben → das reicht nicht mehr!

Beispiel: Zuteilung von Plattenlaufwerken (4)

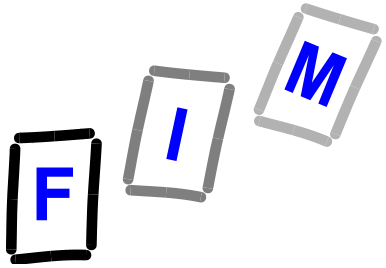
- Der vorige Zustand ist unsafe!!!!



8 Ds sind in Verwendung
4 noch frei

P0 kann nicht 5, P2 kann nicht 6 Ds bekommen
Falls beide, P0 und P2, auf Ihren Anforderungen 5 bzw. 6 bestehen, müssen beide warten:

Deadlock

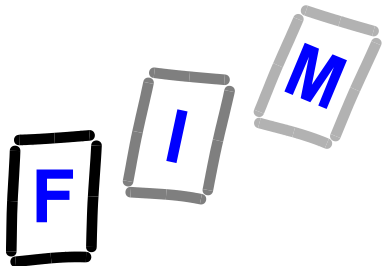


Banker's algorithm Dijkstra, 1965

- **Mit dem Konzept „sicherer Zustand“ kann man einen Algorithmus zur Vermeidung angeben:**

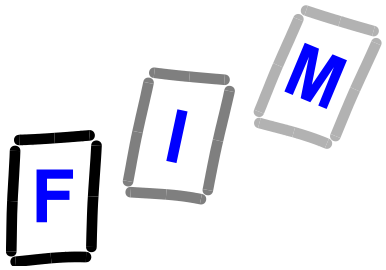
<u>Basis-Idee:</u>

- **Beginne in einem sicheren Zustand**
- **Simuliere die Anforderung**
- **Prüfe, ob Coffman-Bedingung „circular wait“ eintritt**
- **Verwende dazu einen BM-Belegungs-Graph**
- **Prüfe, ob dieser einen Kreis enthält**



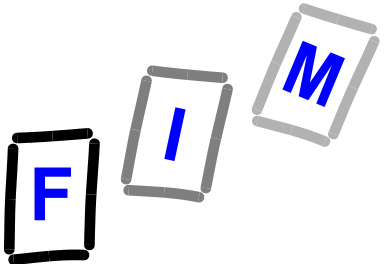
Banker's Algorithm
woher der Name kommt

- Wird kein Kreis gefunden, dann führt die BM Zuteilung wieder in einen sicheren Zustand
- Zur Bezeichnung: Eine Bank darf ihr verfügbares Geld nicht so verleihen, dass sie die Kreditzusagen an ihre Kunden bei deren Bedarf nicht einhalten kann
  - ➡ n: Anzahl der Kunden
  - ➡ m: Anzahl der Ressourcen (Geldeinheiten)



Banker's Algorithmus Zeitkomplexität

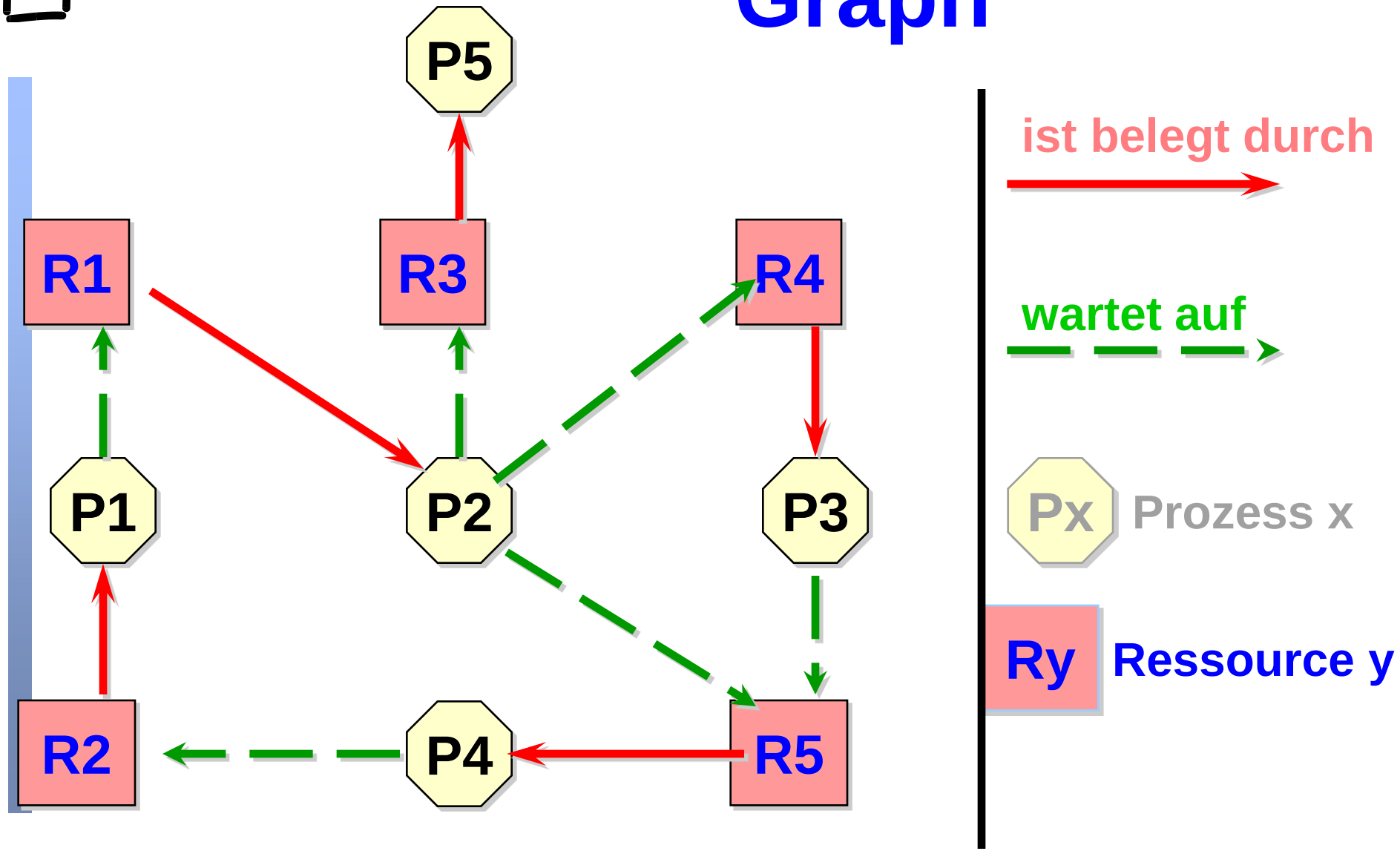
- **Unterscheide zwischen**
  - **Einfacher Fall: Jede der m Ressourcen gibt es genau k=1 mal**
  - **Allgemeiner Fall: Ressourcen sind Klassen und enthalten mehrfache (k>1) Elemente**
- **Auch für k=1: $O(m*n^2)$ Operationen !**
  - **R.C.Holt, 1971: O(m*n), aber komplizierter**
- **Theoretisch nett, aber nicht praxistauglich**
  - **Jeder Prozess müsste seine Anforderungen im Voraus bekannt geben**



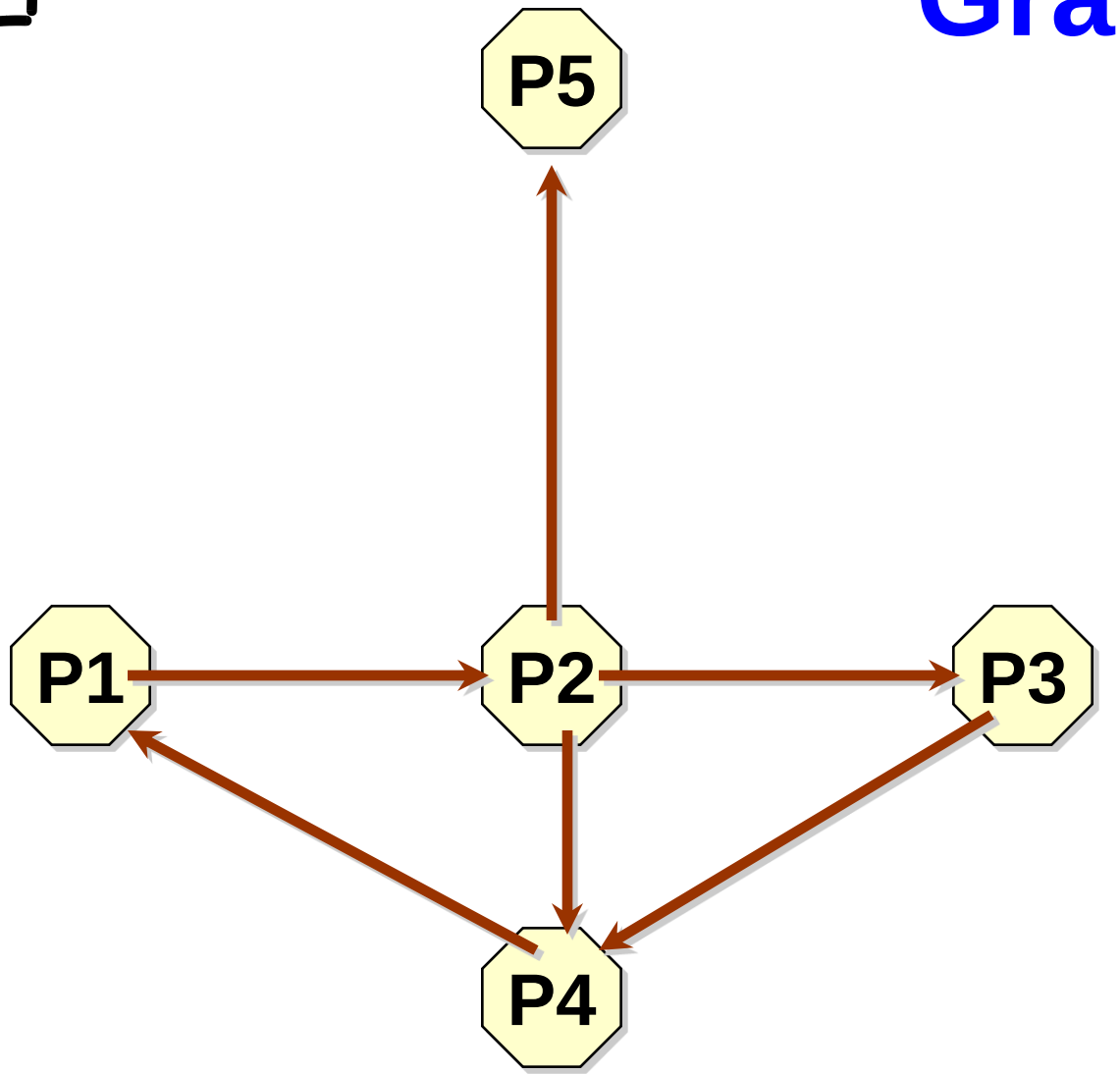
Methode 2

- **Ein Deadlock darf eintreten,**
  aber dann erfolgt eine Befreiung davon
  - ➡ **Umfasst daher 2 Schritte**
    - » **Erkennen (detection)**
    - » **Wiederherstellen (recovery)**

Ressource Zuordnungs-Graph



„Warte auf“ Graph

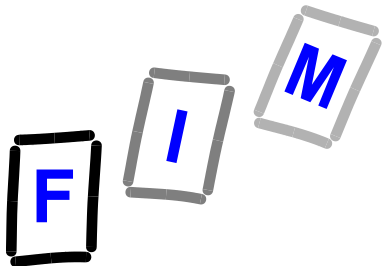


wartet auf

Deadlock existiert, besteht aus: {P1, P2, P3, P4} oder/und {P1, P2, P4}

Erkennen - Wiederherstellen

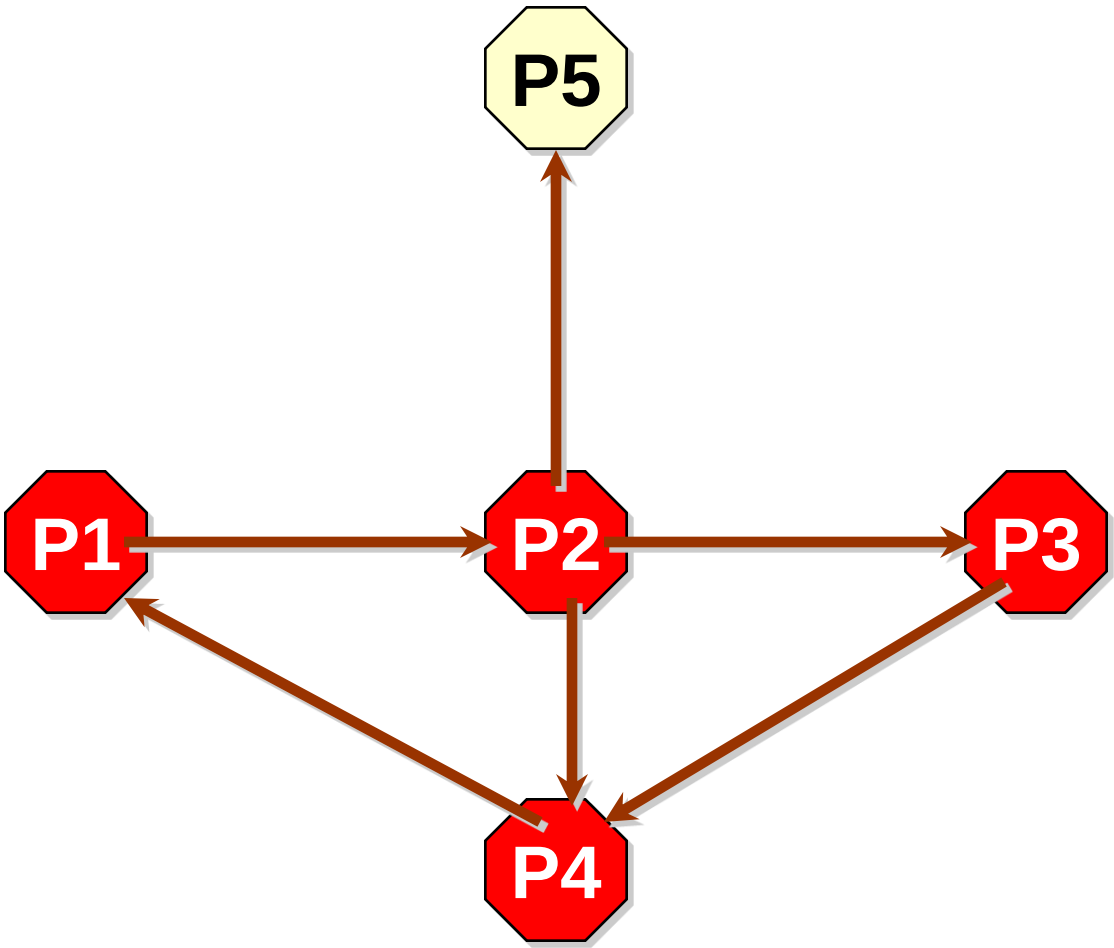
- **Der Systemzustand wird darauf überprüft, ob ein Deadlock eingetreten ist**
  - **z.B.: Bestimmen von Kreisen im Betriebsmittel-Zuteilungsgraph (Zuordnungsgraph) oder Warte-auf-Graph**
- **Ein Algorithmus sorgt für „recovery“ (Befreiung aus dem Deadlock)**
  - **z.B. mittels: Stoppe alle jene Prozesse, die in den Kreisen aufgespürt worden sind, bis kein Kreis mehr vorhanden ist.**
  - **z.B. mittels: Betriebsmittelentzug, bis kein Kreis mehr vorhanden ist**



Detection and Recovery: Spezielle Probleme

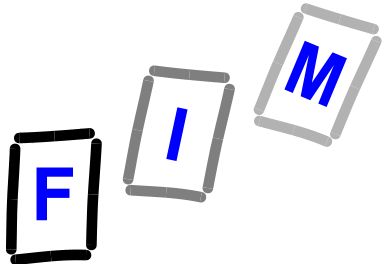
- Welcher Prozess soll abgebrochen werden??
- Nicht jeder Prozess kann zu jedem Zeitpunkt gestoppt und wieder aufgesetzt werden
- Welche Ressource soll (temporär) freigegeben werden ?
  - ➡ Es gibt Ressourcen, die können/dürfen nicht zu einem beliebigen Zeitpunkt freigegeben und später wieder verwendet werden.
- Aufruf des Deadlock-Erkennungs-Algorithmus:
  - ➡ Wann und in welchen Abständen?

Recovery (1) mit Prozessabbruch



Der Warte-auf-Graph bestimmt nur die in einen Deadlock involvierten Prozesse.

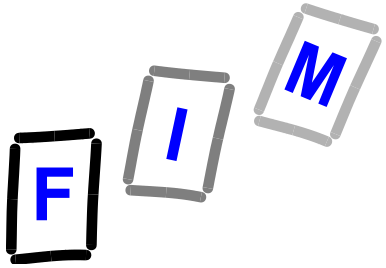
Die betroffenen Ressourcen werden nicht bestimmt.



Recovery (2) mit Prozessabbruch

Prozess-Abbruch schaut leichter aus als er bewerkstelligt werden kann:

- **Alle Prozesse abbrechen?:**
  - ➡ **Teuer, alle bislang erfolgten (Teil-)Berechnungen müssen verworfen und später völlig neu begonnen werden.**
- **Nur einen Prozess abbrechen?:**
  - ➡ **Mit welchem Prozess soll begonnen werden?**
  - ➡ **Overhead: Nach Abbruch eines jeden P muss erneut geprüft werden, ob nicht noch immer ein Deadlock vorliegt und weitere Prozesse zu stoppen sind.**



Recovery (3) mit Prozessabbruch

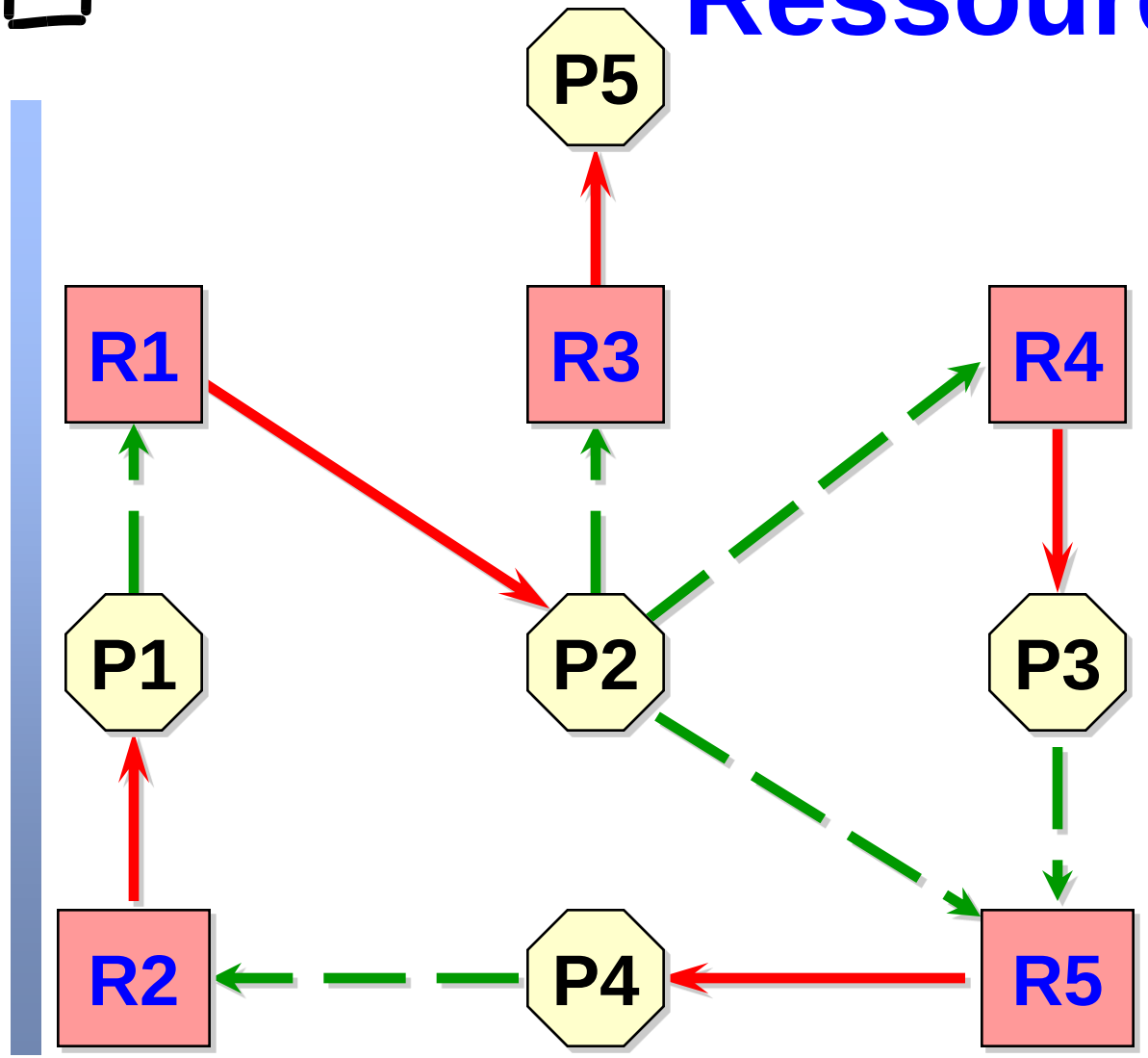
Abbruch eines Prozesses

- **Mit welchem Prozess soll begonnen werden?**

hängt ab von

- ➡ **Priorität von P?**
- ➡ **Welche Rechenzeit hat P schon verbraucht?**
- ➡ **Welche Betriebsmittel R und welcher Art hat P bislang ge- und verbraucht?**
- ➡ **Wie viele und welche R wird/würde P bis zur Fertigstellung noch benötigen?**
- ➡ **Wie viele Ps müsste man stoppen, würde man P weiterlaufen lassen?**
- ➡ **Ist P ein Batchprozess oder ein interaktiver Prozess?**

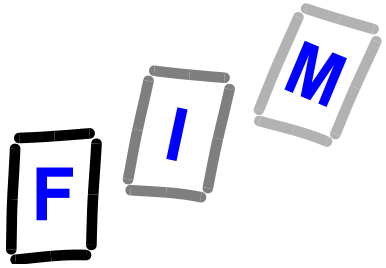
Wiederherstellen (4) Ressourcen-Entzug



Bestimmung und Speicherung des „resource allocation graph“ ist kompliziert

Verwendet man „BM-Entzug“ müssen 3 Aspekte berücksichtigt werden :

- **Opfer wählen**
- **„roll back“**
- **„starvation“**



Wiederherstellen (5) Ressourcen-Entzug

- **Opfer auswählen**
  - ➡ **Welche Ressourcen und damit welche P kommen in Betracht? Kosten minimieren!**
- **Zurückrollen: Rollback**
  - ➡ **Wenn eine Ressource einem Prozess P weggenommen wird, dann muss P zurückgebracht (rolled back) werden in einen sicheren Zustand, von dem er später neu gestartet werden kann.**
- **Verhungern: Starvation**
  - ➡ **Wie kann man sicherstellen, dass R_s nicht immer dem selben P entzogen werden?**