5. Betriebsmittelverwaltung

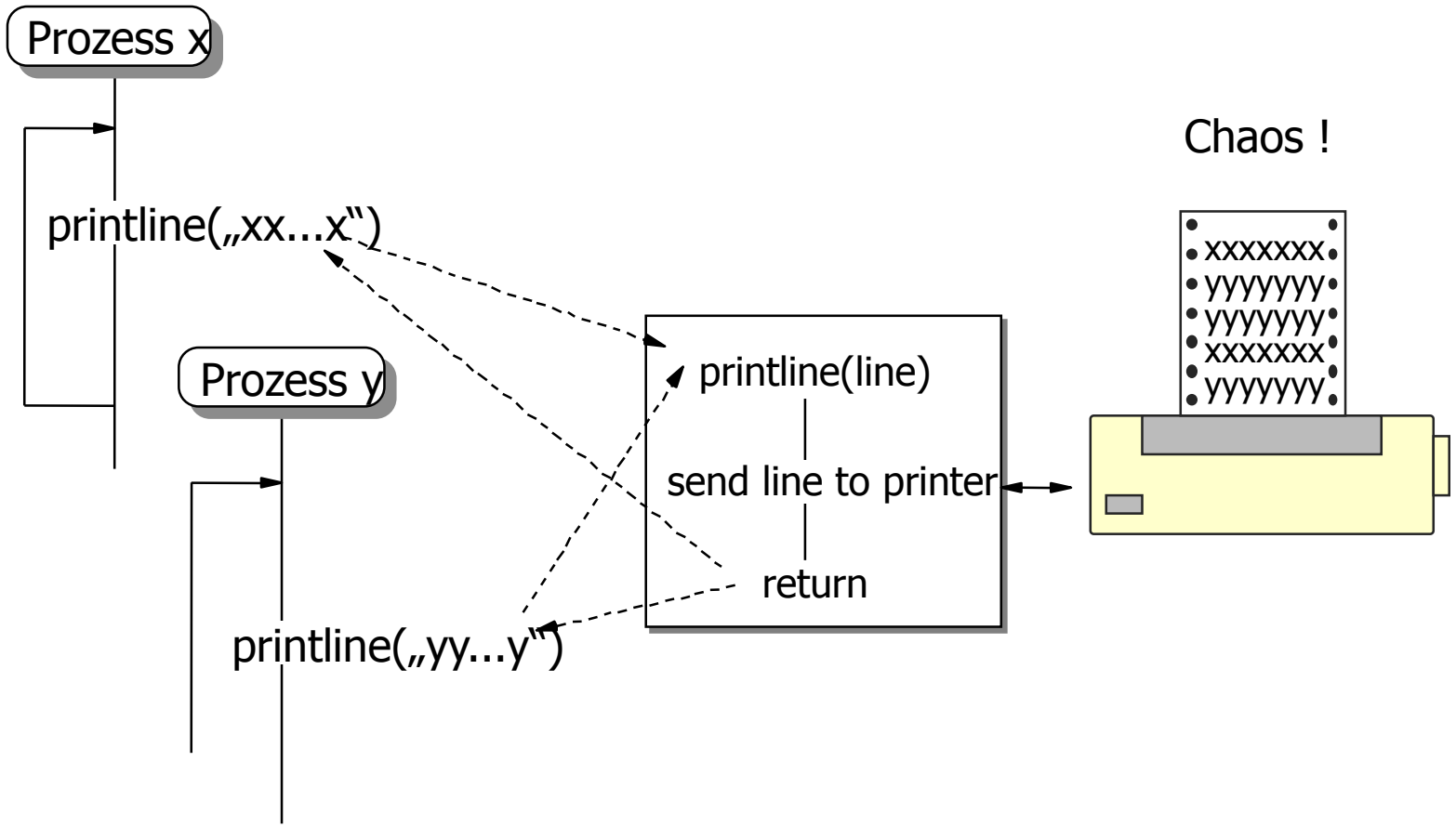
5.1 Einführung
5.2 Zentrale Betriebsmittelverwaltung
5.3 Auswahlstrategien
5.4 Verklemmungen
5.5 Behandlung von Verklemmungen
5.6 Betriebsmittelgraphen zur Modellierung von Belegungssituationen
5.7 Erkennung und Beseitigung von Verklemmungssituationen

5.1 Einführung

- Betriebsmittel (BM) oder Ressource (resource)
  - Alles, was ein Prozess als Aktivitätsträger in einem System zum Vorankommen benötigt
  - BM nur dann ein Problem, wenn sie nicht in beliebigem Umfang zur Verfügung stehen bzw. nicht simultan genutzt werden können
- Beispiel 1
  - Ein Prozess benötigt das Programm, das er ausführen soll
    - ⇒Das Programm ist ein BM des Prozesses
    - ⇒Aber auch andere Prozesse können gleichzeitig dasselbe Programm ausführen
    - ⇒BM muss nicht „bewirtschaftet" werden
- Beispiel 2
  - Prozesse benötigen Hauptspeicher zur Ablage ihrer Daten
  - Speicher steht nur begrenzt zur Verfügung und muss zugeteilt, d.h. verwaltet werden

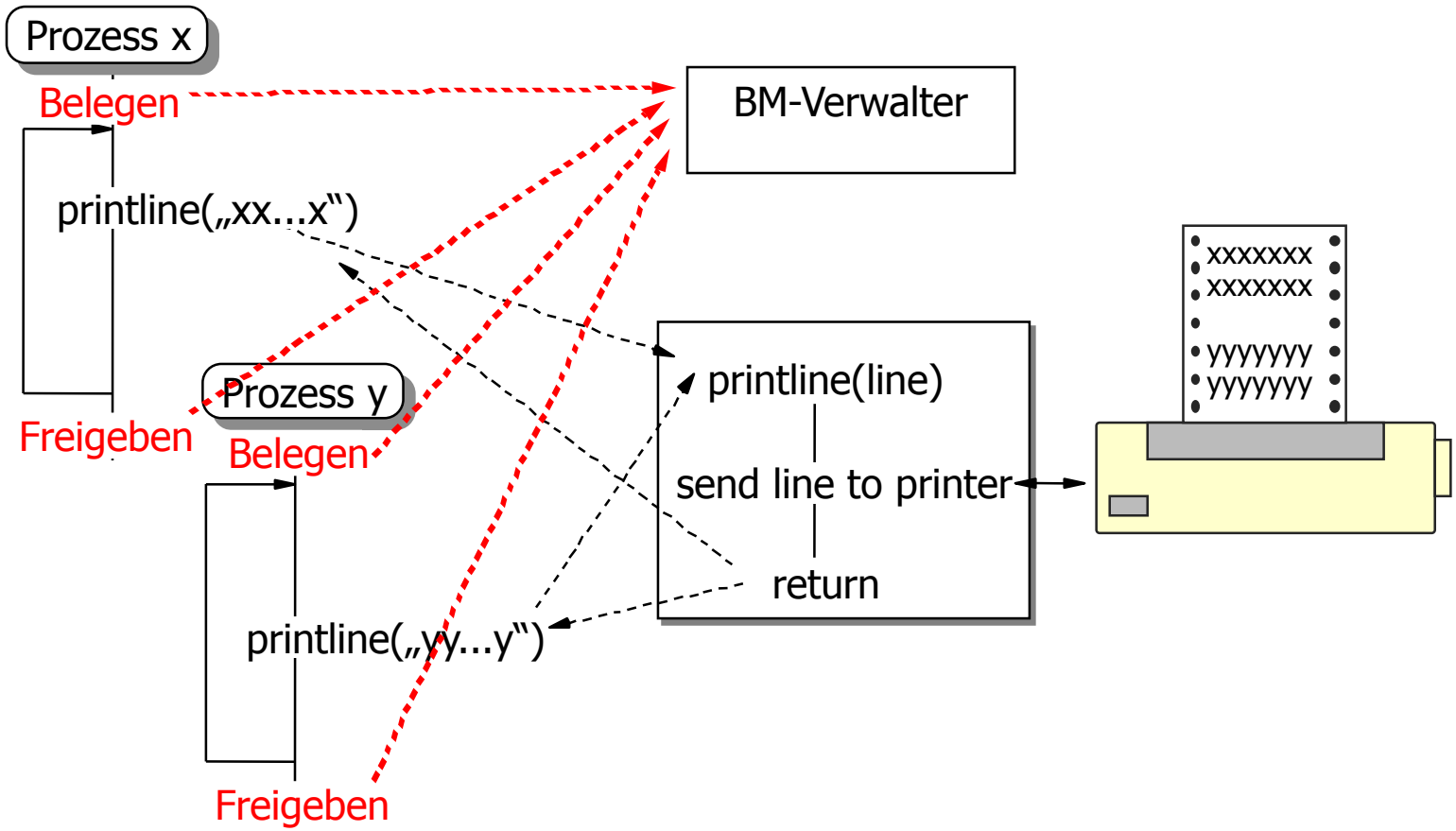
Unkoordinierte Nutzung

- Bei unkoordinierter Nutzung eines Betriebsmittels können unerwünschte Effekte auftreten



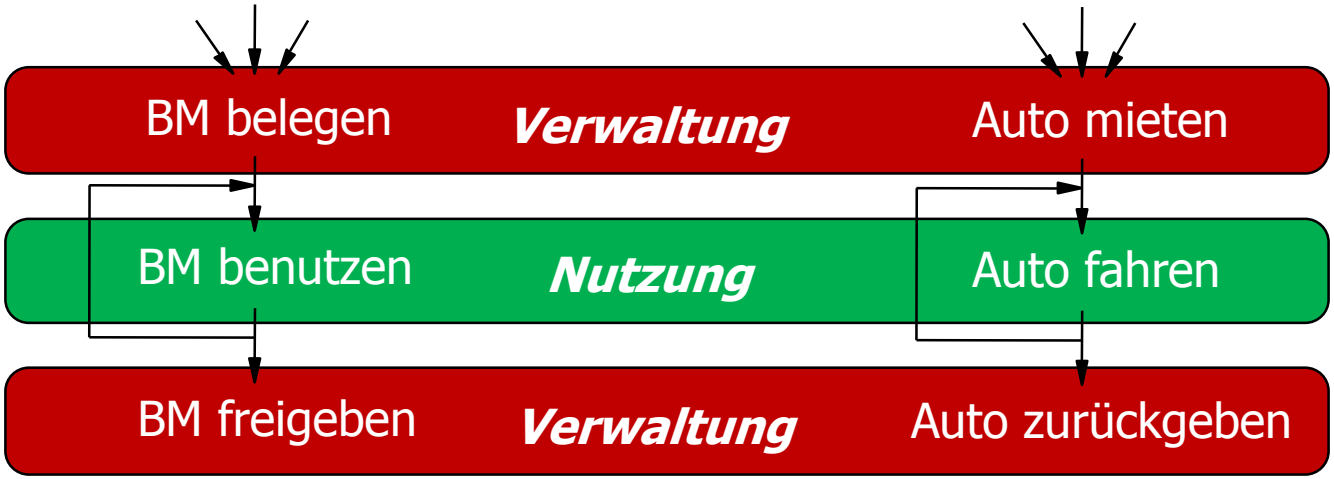
Koordination durch Verwaltungskomponente

- Durch den Einsatz eines BM-Verwalters kann z.B. eine exklusive Nutzung erreicht werden

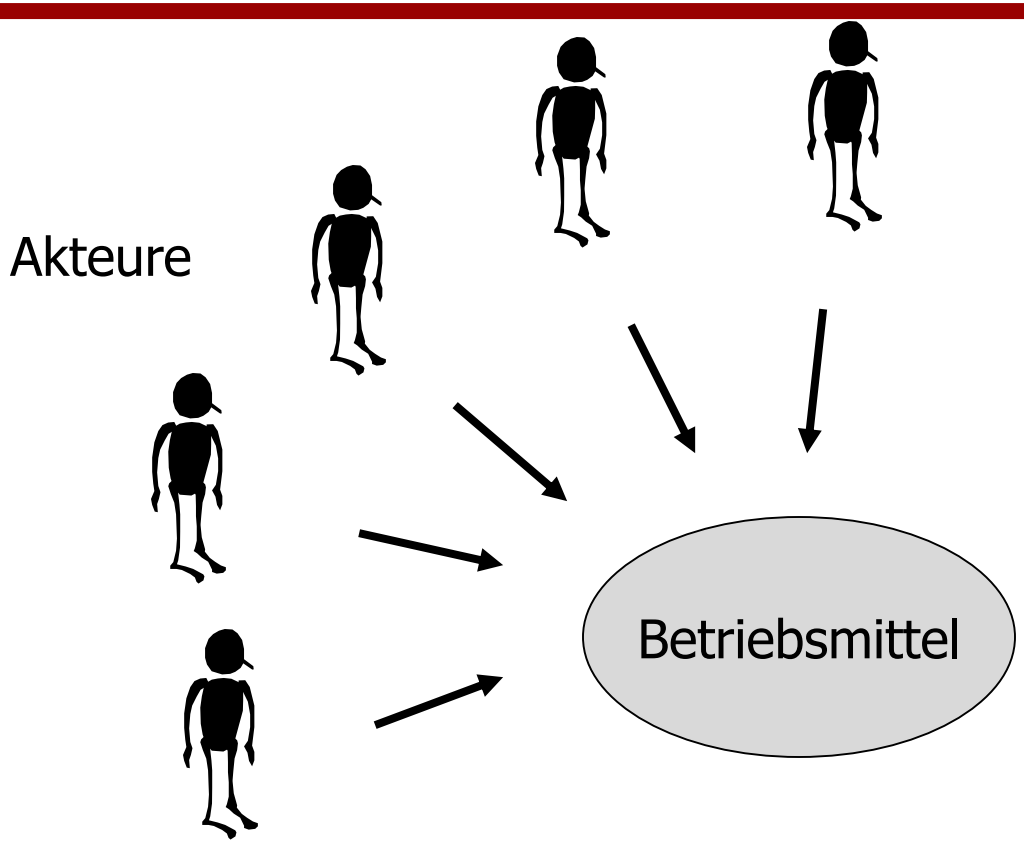


Betriebsmittelverwaltung

- Differenzierung zwischen Nutzung und Verwaltung!
- Die Nutzung wird von Verwaltungsoperationen geklammert!



Worum geht es?



- Betriebsmittel sind knapp
- Benutzung erfolgt exklusiv
- Verwaltung ist sinnvoll

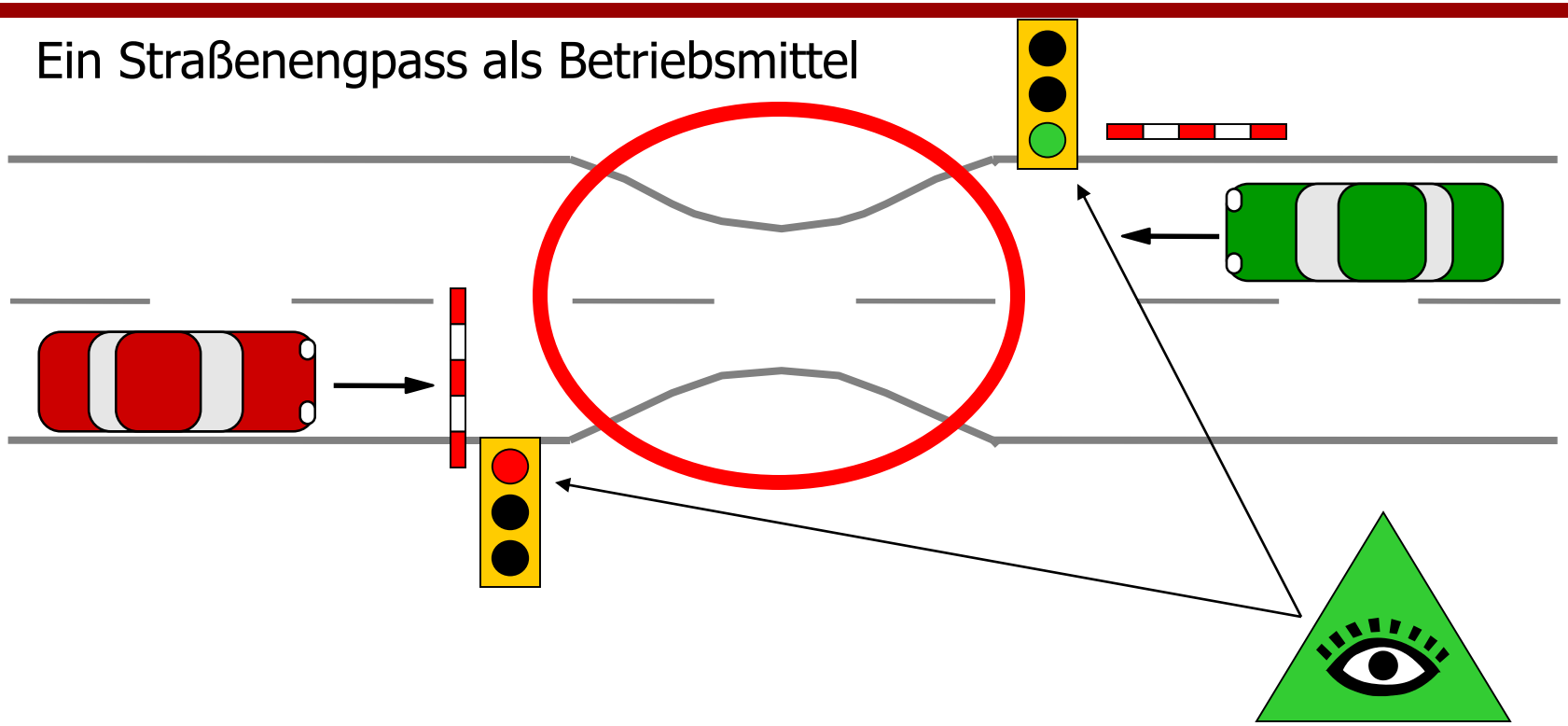
Akteure, z.B.
- Benutzer
- Prozess
- Thread
- Prozessor
- Netzwerkkarte

Betriebsmittel, z.B.
- Prozessor
- Speicher
- Bandbreite

- Datei
- Signal
- Nachricht
- Name
- Farbe

Betriebsmittelprobleme im Alltag

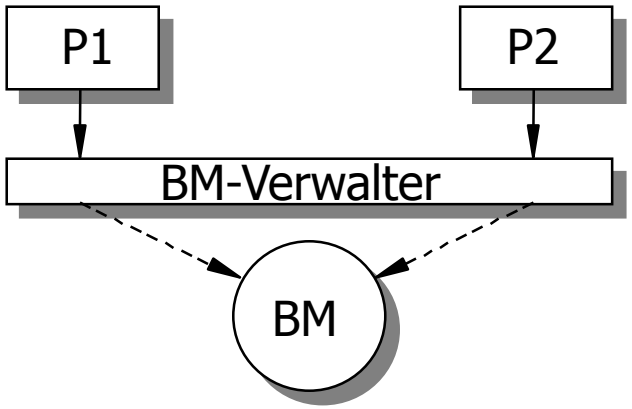
Ein Straßenengpass als Betriebsmittel



- Lösung 1
  - Zentrale Instanz entscheidet (Betriebsmittelverwalter)
  - Ampeln, Schranken

Lösung 1: Betriebsmittelverwalter

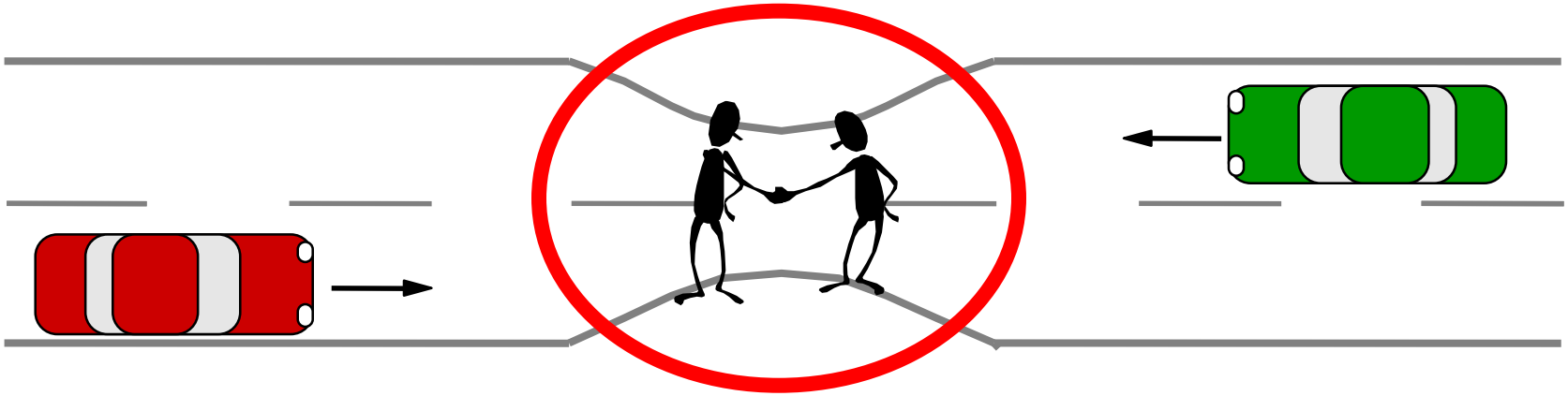
- Nutzung des BM nur nach vorheriger **Belegung** möglich
- Die vorherige Belegung wird durch eine zwischengeschaltete Instanz **erzwungen**



- Beispiele
  - Hauptspeicherverwaltung (Zugriffe sind nur innerhalb der zugewiesenen Segmente möglich)
  - Monitor (Aufruf einer Entry-Prozedur ist nur möglich, wenn der Monitor frei ist.)
  - Drucken (Zugriff auf den Drucker ist nicht unmittelbar möglich, sondern nur über spezielle Software, den Treiber, der als Verwalter fungiert)

Betriebsmittelprobleme im Alltag

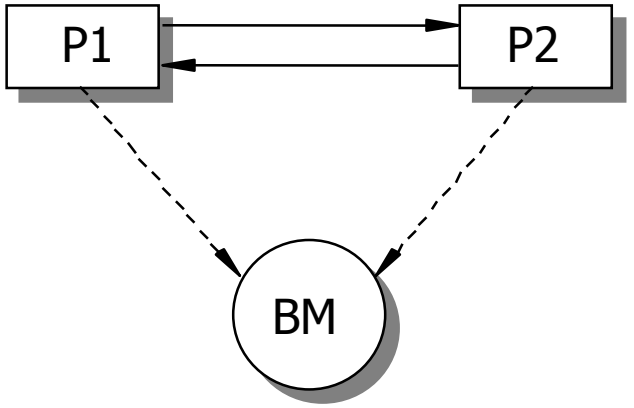
Ein Straßenengpass als Betriebsmittel



- Lösung 2
  - Verständigung der Teilnehmer (Regeln, Verhandlung, Protokoll)
  - Verkehrszeichen, „Berg- vor Talfahrt“, Handzeichen, Lichthupe

Lösung 2: Verständigung

- Die Bewerber um das Betriebsmittel stimmen sich ab (Protokoll)



- Beispiele für Protokolle / Regeln
  - Kritischer Abschnitt
    - Vereinbarung, den kritischen Abschnitt durch Nutzung von Sperren unter gegenseitigen Ausschluss zu stellen
  - Dezentrale Bus-Arbitrierung
    - Sendewillige Komponenten (bus request) legen im speziellen Koordinationsprotokoll (Bus-Arbitrierung) fest, wer als nächster senden darf
  - Verteilte Systeme
    - Knoten melden per Broadcast Bedarf an, Abstimmung basierend auf logischer Zeit legt Zugriffsreihenfolge fest

Betriebsmittelprobleme im Alltag

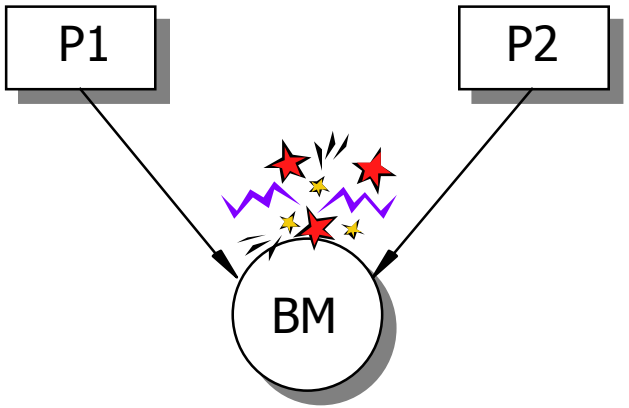
Ein Straßenengpass als Betriebsmittel

- Lösung 3
  - Keinerlei Maßnahmen
  - Kollisionsgefahr

Lösung 3: Unkoordinierte Nutzung

- Ohne Abstimmung der Interessenten kann es zu **Kollisionen** kommen

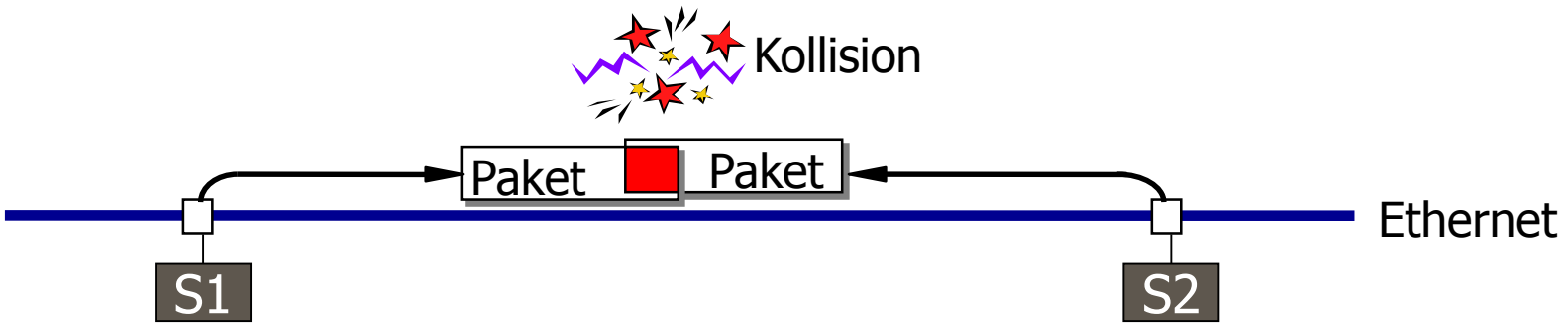
  ⇒Kollisionen müssen geeignet aufgelöst werden



- Aufwand für seltene Kollisionsauflösung kann geringer sein als der permanente Aufwand für eine vorherige Abstimmung
- Einsatz dort, wo
  - eine Kollision unwahrscheinlich, d.h. selten und
  - der durch die Kollision entstandene „Schaden" „reparabel" ist.

Lösung 3: Beispiele für unkoordinierte Nutzung

- Optimistische Synchronisation von Transaktionen (Validierung)
  - Transaktionen setzen keine Sperren, sondern greifen einfach zu
  - Zugriffe werden protokolliert (Log)
  - Am Ende (Commit) wird überprüft, ob Operationen in Konflikt zu anderen standen (Validierung)
  - Falls ja, wird die Transaktion abgebrochen (und neu gestartet)
- Lokale Netze: Kollisionsbehaftete Protokolle (Ethernet)
  - Sendewillige Station sendet, nachdem sie vorher kurz die Leitung abgehört hatte
  - Senden zwei Stationen gleichzeitig, so kollidieren die Datenpakete und werden zerstört ⇒ beide Stationen müssen nach Wartezeit erneut die Daten schicken



Klassifikation

- Existenzform: real / logisch / virtuell
- Reales Betriebsmittel: physisch existent
  - Voraussetzung für virtuelle/logische BM
  - Beispiel: Hauptspeicher, Platte, Prozessor

- Virtuelles Betriebsmittel = eine größere Anzahl eines BM als real vorhanden wird vorgespiegelt
  - Virtuelle BM werden nur für kurze Zeiten auf das reale BM abgebildet (Multiplexing)
  - Beispiel: Virtueller Speicher, Virtuelle Verbindung

- Logisches Betriebsmittel = Abstraktion des realen BM
  - Benutzer hat im Vergleich zum realen eine komfortablere, funktional angereicherte Schnittstelle
  - Beispiel: Datei = Abstraktion der Platte, Fenster = Abstraktion des Bildschirms

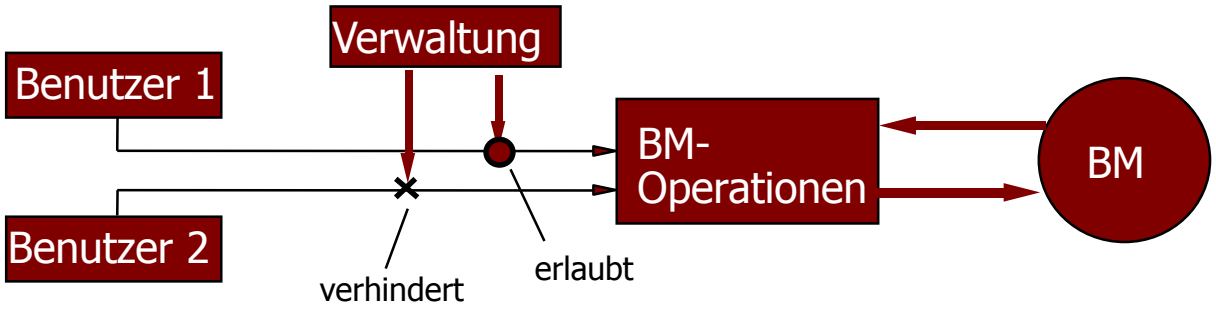
Klassifikation

- Persistenz
  - Wiederverwendbar
    - BM werden nach Nutzung freigegeben und können von anderen Prozessen genutzt werden
  - Verbrauchbar
    - Einige logische BM werden durch die Nutzung verbraucht, d.h. sie werden erzeugt und sind nach ihrer einmaligen Nutzung nicht mehr vorhanden
    - Beispiele: Signale, Nachrichten, Zeitstempel

- Kapazität
  - Begrenzt
    - BM muss bewirtschaftet werden (explizites Belegen / Freigeben)
  - Unbegrenzt
    - BM-Verwaltung weitgehend verzichtbar, höchstens An-/Abmelden einer Nutzung

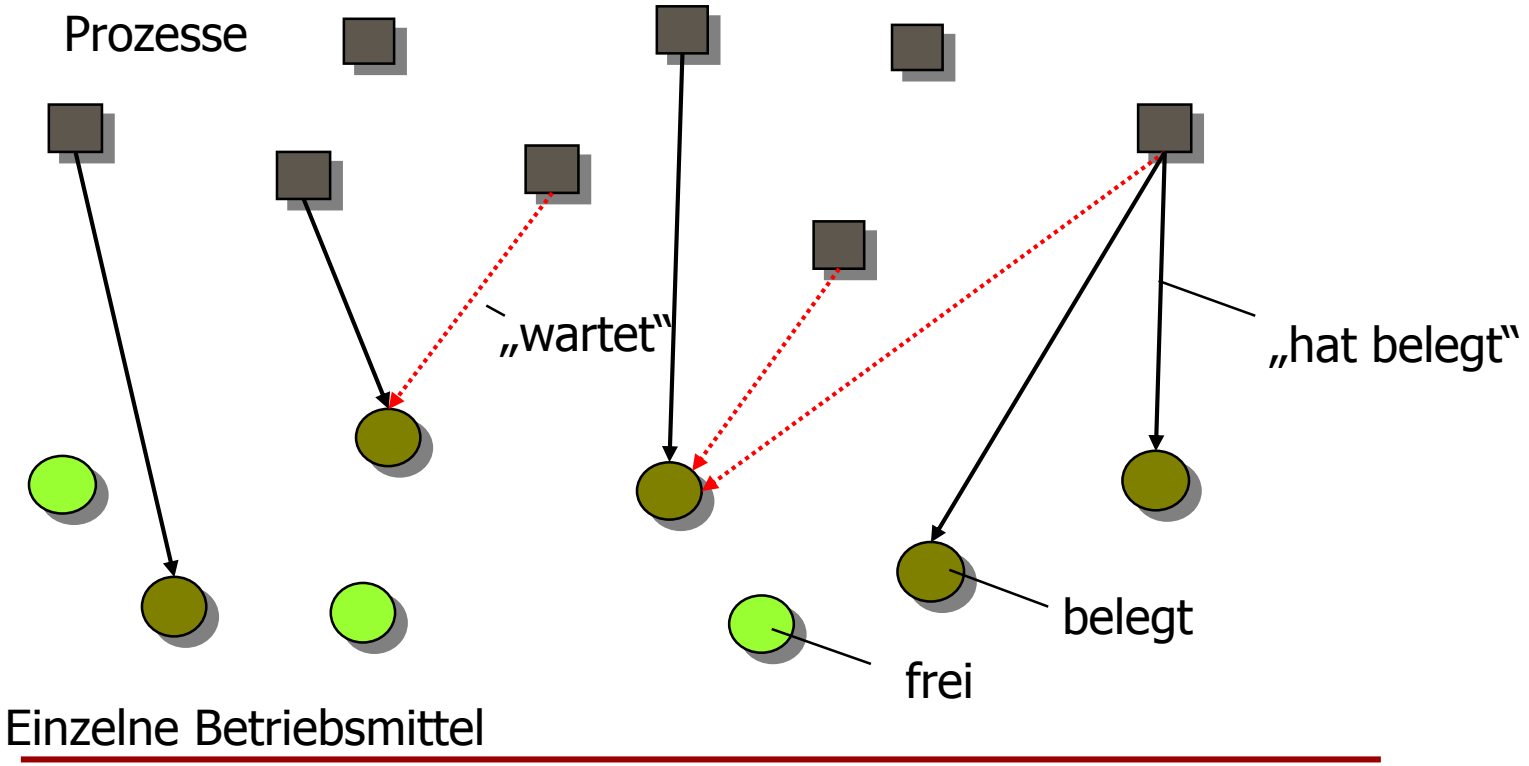
Zwischenbilanz

- BM = Umfassender Begriff für alles, was ein Prozess benötigt
- BM-Verwaltung
  - Summe aller Aufgaben, die organisatorisch vor und nach der Nutzung von BM erforderlich sind, um einen reibungsfreien Betrieb zu gewährleisten
- Ziele einer BM-Verwaltung
  - korrekte Abläufe
  - keine Verklemmung
  - kein Verhungern
  - hohe Nebenläufigkeit
  - hohe Auslastung des Betriebsmittels
- Praxis
  - oft mit Lösung komplexer Optimierungsprobleme verbunden



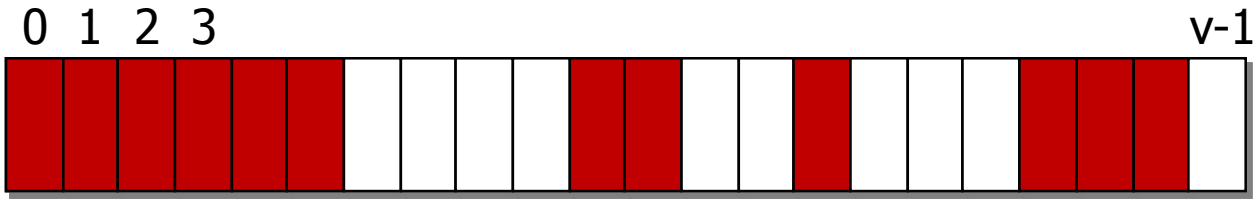
5.2 Zentrale Betriebsmittel-verwaltung: Einexemplar

- Nutzung von Einexemplar-Betriebsmitteln kann als kritischer Abschnitt angesehen werden
  - ⇒ BM-Verwaltung: Koordinationsproblem im erweiterten Sinn, *Belegen (allocate)* und *Freigeben (release)* mit gleicher Struktur wie Sperr-Operationen *lock* und *unlock*

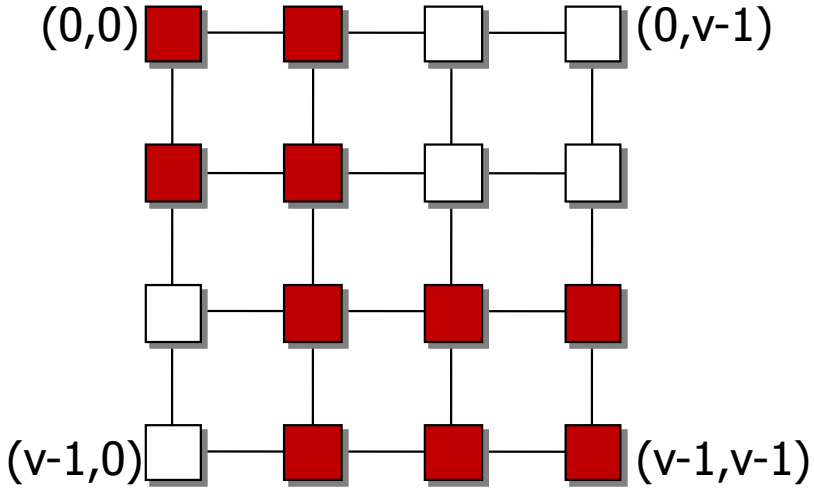


Mehrexemplar-BM: Teilbare Betriebsmittel

- Speicher (eindimensionales teilbares BM)



- Prozessoren (Parallelrechner) als zweidimensionales teilbares BM



Daten im Betriebsmittelverwalter

- Minimale Daten
  - Belegungszustand (frei, belegt)
  - Wartende Prozesse (die beim Belegungsversuch blockiert wurden)
- Ergänzende Daten (optional)
  - Belegender Prozess (aktueller Besitzer)
  - Anzahl Belegungen
  - Mittlere Belegungsdauer
  - Nutzungsgrad
  - Beginn der aktuellen Belegung
  - ....
- Diese Daten benötigt man u.U. für bestimmte Strategien, z.B. auch für BM-Entzug

Belegungsdarstellung

- Wie speichert man den Belegungszustand in einer Datenstruktur?
- Einfachster Fall: Bitliste

- Operationen zum Zugriff auf den Belegungszustand
  - **boolean** `free(num)` Prüft, ob eine Anforderung nach einem Stück der Länge *num* erfüllbar
  - **void** `set_occupied(start,num)` Setzt die entsprechenden Bits
  - **void** `set_free(start,num)` Setzt die entsprechenden Bits zurück

Kandidatenauswahlstrategien

- Für ein BM existieren mehrere Interessenten
- Auswahl durch zentrale Instanz so, dass gute BM-Auslastung und faire Behandlung der Interessenten erreicht wird
  - $n_f(t)$: Anzahl der zum Zeitpunkt t freien BM-Einheiten
  - $n(i)$: Anzahl der vom Prozess i geforderten BM-Einheiten
  - $W(t)$: Warteschlange der angemeldeten Anforderungen / Prozesse

- Strategie FCFS/FIFO (*First-In-First-Out*)
  - Betrachte die erste Anforderung i in der Warteschlange
  - Gilt $n(i) \leq n_f(t)$, belege die Einheiten
  - Andererseits, warte bis $n(i)$ Einheiten frei sind
  - Auslastung u.U. schlecht, wenn die erste Anforderung groß ist

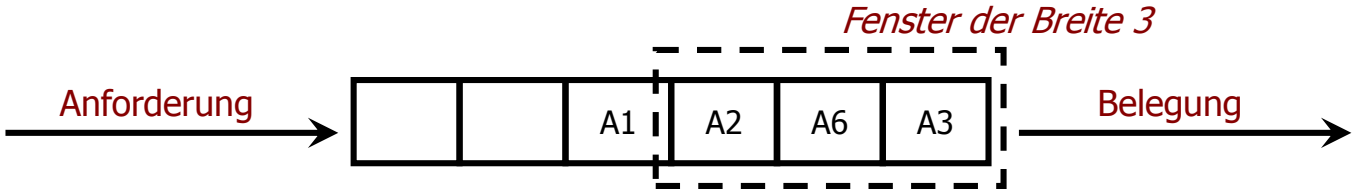
    $\Rightarrow$Nachfolgende kleinere und erfüllbare Anforderungen bleiben unberücksichtigt

5.3 Auswahlstrategien

- Strategie *First-Fit-Request*
  - Durchsuche die Warteschlange (vorne beginnend), bis die erste erfüllbare Anforderung *i* gefunden ist, d.h. $n(i) \leq n_f(t)$
- Strategie *Best-Fit-Request*
  - Durchsuche die Warteschlange vollständig und finde die Anforderung *i*, welche die Restkapazität minimiert, d.h.

$$\min_{j \in W(t) \wedge n(j) \leq n_f(t)} \{n_f(t) - n(j)\}$$

- Iterative Anwendung
  - Wende die Strategien an, bis keine Belegung mehr möglich
  - Aufwandreduktion durch Betrachtung der ersten L Anforderungen (Fenster der Breite L)

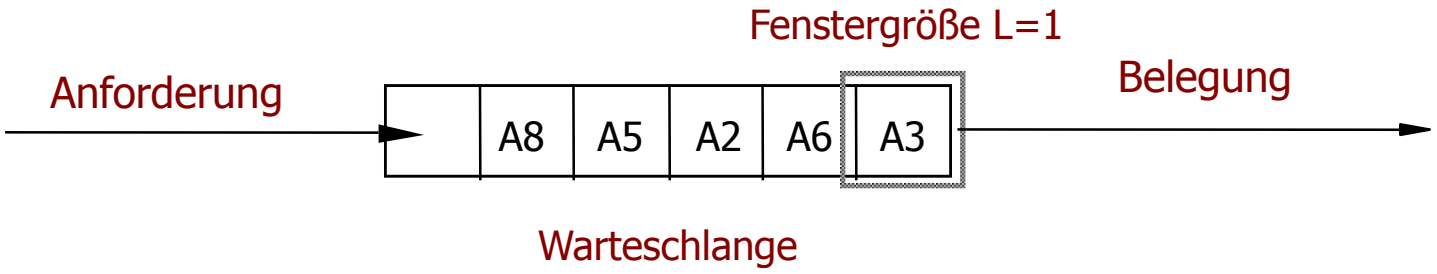


Problem des Verhungerns (Starvation)

- Bei First/Best-Fit besteht die Gefahr, dass Prozesse mit großen Anforderungen sehr lange warten müssen (Verhungern)
- Idee: Verwende Fenster dynamischer Größe
  - Initialbreite Lmax
  - Nach jeder erfolgreichen Belegung wird die Fensterbreite folgendermaßen reduziert

$$L = \begin{cases} L - 1, \text{ falls } L > 1 \text{ und erste Anforderung nicht berücksichtigt} \\ L_{max}, \text{ sonst} \end{cases}$$

  - Nach spätestens Lmax-1 Zugriffe gilt Fenstergröße = 1, d.h. die vorderste Anforderung muss berücksichtigt werden

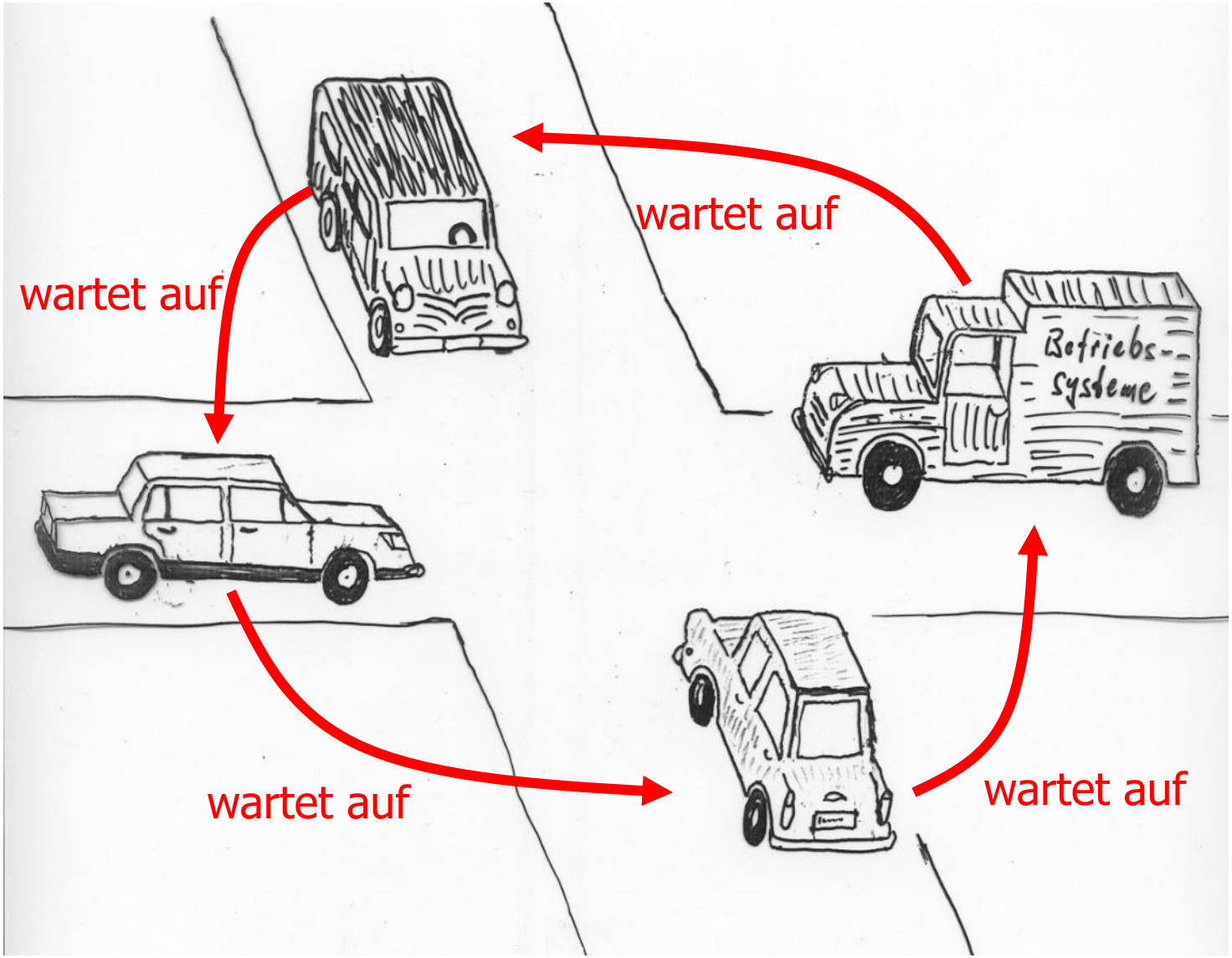


5.4 Verklemmung (Deadlock)

- Konzept
  - Verklemmung (Deadlock) bewirkt, dass die Abarbeitung bestimmter Operationen dauerhaft gestoppt wird $\Rightarrow$ kein Fortschritt im System möglich
- Praxis
  - Blockierte Prozesse und Threads
- Erkennung und Behandlung
  - Vier notwendige und hinreichende Bedingungen vorhanden
- Ziel
  - Vermeidung von Deadlocks: Systeme so entwerfen, dass eine Verklemmung erst gar nicht auftreten kann

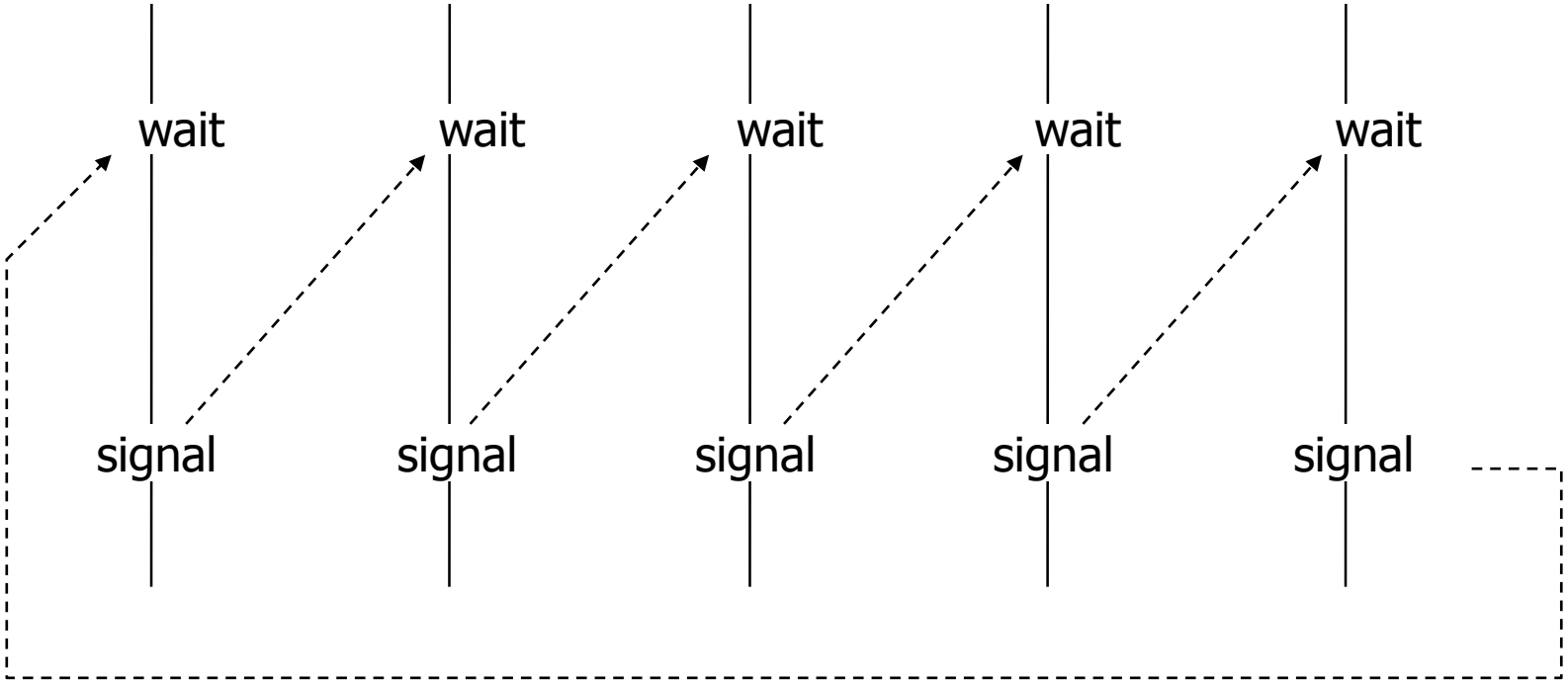
$\Rightarrow$ Unter einer **Verklemmung** wird eine Situation verstanden, in der Prozesse sich gegenseitig behindern und blockieren und deshalb nicht weiter ausgeführt werden können

Verklemmung im Alltag



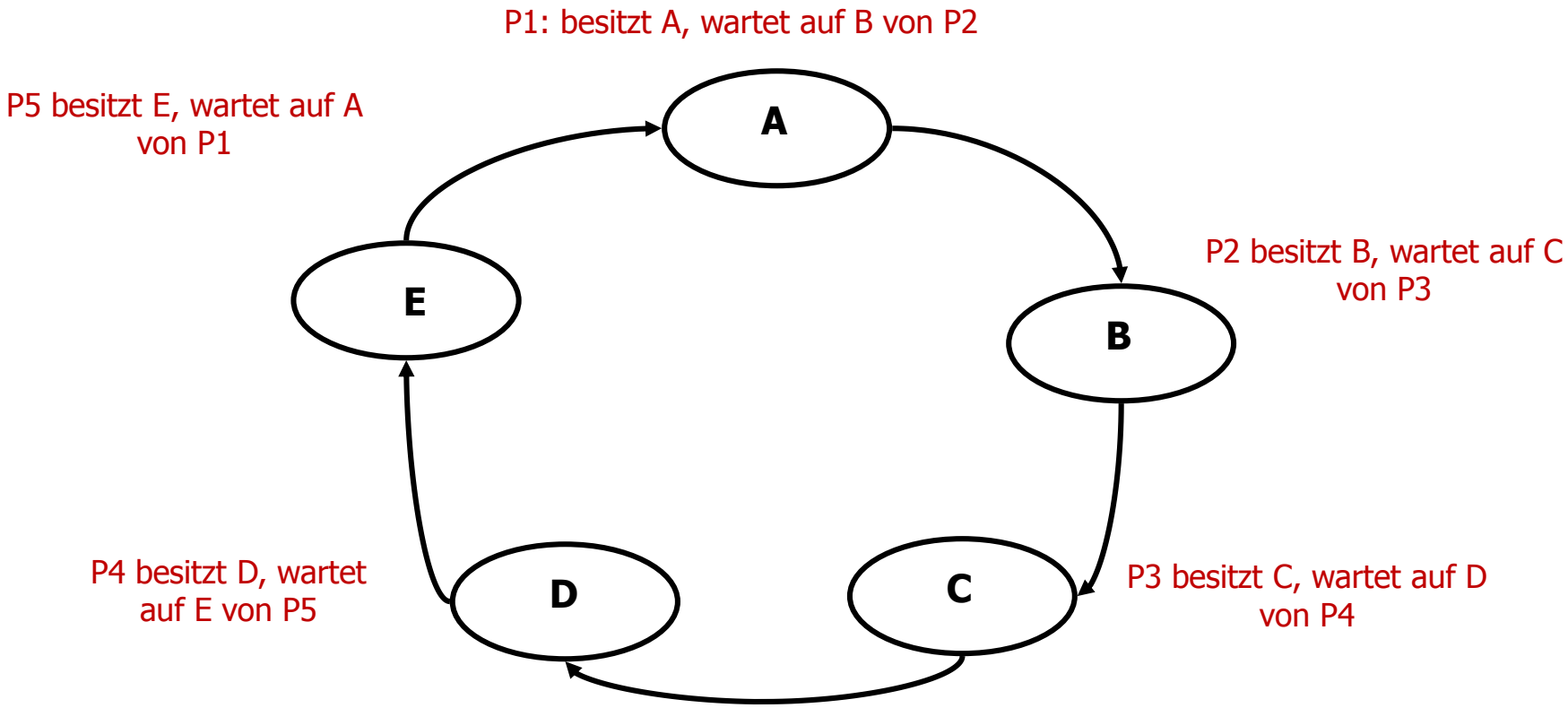
Verklemmung

- Beispiel (mehrere Prozesse beteiligt)



Wartegraph

- Wartegraph (*wait-for graph*) = gerichteter Graph mit den Prozessen als Knoten und Wartebeziehungen als Pfeile
- Verklemmung = charakterisiert durch Zyklus im Wartegraphen



Notwendige Bedingungen für Deadlocks

- Im Zusammenhang mit Betriebsmitteln sind die folgenden drei Bedingungen **notwendig** für das Auftreten einer Verklemmung:
  1. Beschränkte Belegung (*mutual exclusion*): Jedes involvierte BM ist entweder exklusiv belegt oder frei
  2. Zusätzliche Belegung (*hold-and-wait*): Die Prozesse haben bereits BM belegt, wollen zusätzlich weitere BM belegen und warten darauf, dass sie frei werden $\Rightarrow$ notwendige BM werden nicht auf einmal angefordert
  3. Keine vorzeitige Rückgabe (*no pre-emption*): Die bereits belegten BM können den Prozessen nicht wieder entzogen werden, sondern müssen von den Prozessen selbst explizit zurückgegeben werden

Hinreichende Bedingung für Deadlocks

- Unter diesen Bedingungen kann dann die folgende Bedingung eintreten, die - zusammen mit den anderen drei Bedingungen - **hinreichend** ist für die Existenz einer Verklemmung

4. Gegenseitiges Warten (*circular wait*): Eine geschlossene Kette von zwei oder mehr wartenden Prozessen muss existieren, wobei ein Prozess BM vom nächsten haben will, die dieser belegt hat und die deshalb nicht mehr frei sind

5.5 Behandlung von Verklemmungen

- Zur Behandlung von Verklemmungen können folgende Maßnahmen eingesetzt werden
  1. Vorbeugung (*Prevention*)

     Vergabe der BM so restriktiv gestalten, dass mindestens eine der vier Bedingungen für eine Verklemmung nicht erfüllt sein kann

  2. Vermeidung (*Avoidance*)

     In einer aktuellen Belegungssituation werden die Restanforderungen der Prozesse analysiert und so umgesetzt, dass eine unsichere Situation nicht auftreten kann

  3. Entdeckung und Auflösung (*Detection & Recovery*)

     In regelmäßigen Abständen/bei jeder Belegung wird die aktuelle Belegungssituation analysiert $\Rightarrow$ Bei erkannter Verklemmung werden Maßnahmen zur Auflösung des Wartezyklus ergriffen

Verklemmungsvorbeugung

- Bedingung *Mutual Exclusion* wird nicht erfüllt
  - Abschaffung der Konkurrenz für BM durch einen speziellen Verwaltungsprozess, der alle Anfragen annimmt/ausführt
- Beispiel: Verwaltung eines Druckers (Druckerdämon)
  - Druckerdämon erhält dauerhaft den Drucker
  - Alle anderen Prozesse reihen die auszudruckenden Daten in die Warteschlange des Druckerdämons (*spooling*)
  - Druckerdämon arbeitet stellvertretend für die ihn beauftragenden Prozesse die Warteschlange ab

  $\Rightarrow$ Vermeidung von Verklemmungen
- Diese Strategie ist allerdings nicht für alle BM möglich
  - Beispiel Speicher: mehrere Tausend Anforderungen müssten pro Sekunde bearbeitet werden

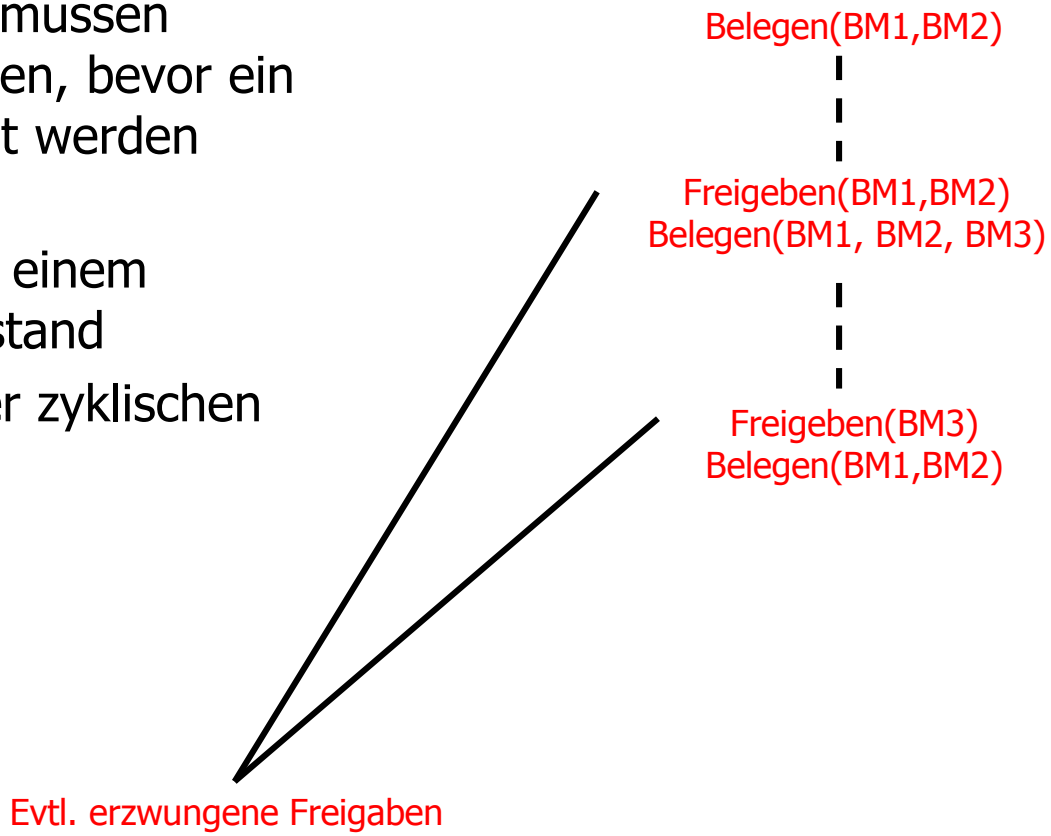
Summenbelegung (pre-claiming)

- Sämtliche benötigten BM werden einmalig zu Beginn angefordert
  - ⇒ Bedingung *Hold-and-Wait* wird nicht erfüllt
  - ⇒ Zusätzlicher Programmieraufwand, da die BM nur dem Entwickler, aber nicht der Systemsoftware bekannt sind

- Anmerkung
  - In dynamischen Systemen ist der Gesamtbedarf eines Prozesses schwierig abzuschätzen
  - Verfahren ist unökonomisch, da BM viel zu lange belegt
- Alternativ
  - Notwendige BM werden phasenweise angefordert und wieder frei gegeben ⇒ Hoher Belegungsaufwand

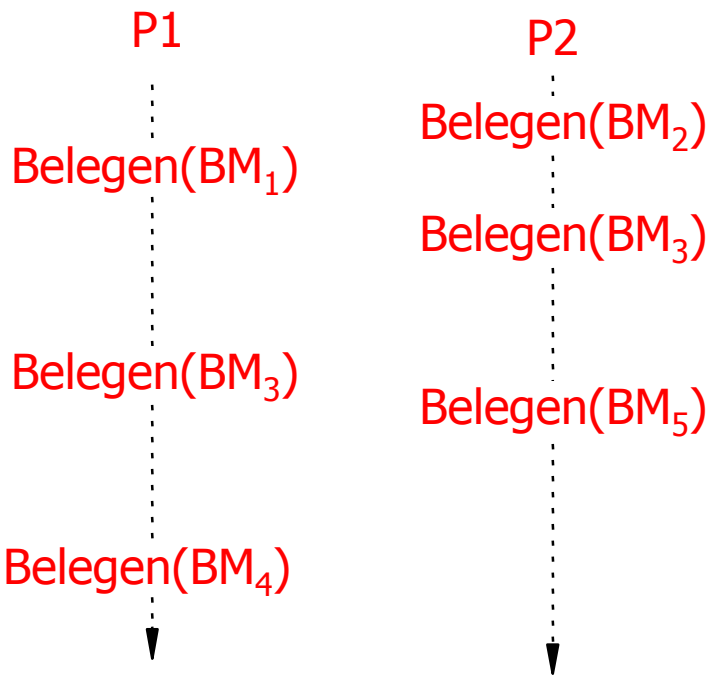
Totalfreigabe

- Totalfreigabe
  - ➢ Alle belegten BM müssen freigegeben werden, bevor ein (neues) BM belegt werden kann
  - ⇒ Anforderung aus einem „besitzlosen" Zustand
  - ⇒ Vermeidung einer zyklischen Wartestellung



Belegung gemäß vorgegebener Ordnung

- Vermeidung vom gegenseitigen Warten durch Belegung gemäß vorgegebener Ordnung
  - BM werden geordnet und durch-nummeriert (BM1, BM2, BM3, …)
  - BM dürfen nur gemäß dieser Ordnung angefordert werden
  - Zunächst wird das benötigte BM niedrigster Ordnung (z.B. BM1) reserviert, danach das nächste BM (z.B. BM3) angefordert usw.
    - ⇒Durch die Ordnungsrelation für die BM-Anforderung werden Zyklen durchbrochen
    - ⇒Linearisierung der BM-Anforderung



5.6 Beschreibung der BM-Situation

- Die **Betriebsmittelsituation** definiert den aktuellen Anforderungs- und Belegungszustand
- Sie ist vollständig beschrieben mit dem Quintupel $(P, BM, \vec{v}, B, A)$ mit

  P Menge der Prozesse, $|P| = m$

  BM Menge der BM-Typen, $|BM| = n$

Vorhandene Betriebsmittel $\vec{v} := (v_1, v_2, \ldots, v_n)$

Belegungen B Anforderungen A

$$B := \begin{pmatrix} b_{11} & \ldots & b_{1n} \\ : & & : \\ b_{m1} & \ldots & b_{mn} \end{pmatrix} \qquad A := \begin{pmatrix} a_{11} & \ldots & a_{1n} \\ : & & : \\ a_{m1} & \ldots & a_{mn} \end{pmatrix}$$

Gesamtanforderungen G (Maximalanforderungen)

$$G := \begin{pmatrix} g_{11} & \ldots & g_{1n} \\ : & & : \\ g_{m1} & \ldots & g_{mn} \end{pmatrix}$$

Bedingungen

1. Es kann nicht mehr belegt sein, als vorhanden ist

$$\forall j \in \{1,\ldots,n\} : \sum_{i=1}^{m} b_{ij} \leq v_j$$

2. Es kann nicht mehr angefordert werden, als verfügbar ist

$$\forall i \in \{1,\ldots,m\} \quad \forall j \in \{1,\ldots,n\} : a_{ij} + b_{ij} \leq v_j$$

$$g_{ij} \leq v$$

3. Ein Prozess, der eine Anforderung gestellt hat, wird bis zur Belegung blockiert

4. Anforderungen können nur von nicht blockierten Prozessen gestellt werden (Konsequenz aus 3)

Zweckmäßige Hilfsgrößen

- Freie Betriebsmittel $\vec{f} := (f_1, f_2, ..., f_n)$

mit $f_j := v_j - \sum_{i=1}^{m} b_{ij}$ „Freies = Vorhandenes - Belegtes“

- Restanforderungen

$$R := \begin{pmatrix} r_{11} & \cdots & r_{1n} \\ \vdots & & \vdots \\ r_{m1} & \cdots & r_{mn} \end{pmatrix}$$

mit $r_{ij} := g_{ij} - b_{ij}$ „Restanforderung = Gesamtanforderung - Belegtes“

Schreibweisen

- Statt Matrix manchmal Zeilenvektoren

Anforderung von Prozess *i*	$\vec{a}_i := (a_{11}, a_{12}, ..., a_{1n})$
Belegung von Prozess *i*	$\vec{b}_i := (b_{11}, b_{12}, ..., b_{1n})$
Gesamtanforderung von Prozess *i*	$\vec{g}_i := (g_{11}, g_{12}, ..., g_{1n})$
Restanforderung von Prozess *i*	$\vec{r}_i := (r_{11}, r_{12}, ..., r_{1n})$

- Vergleichsoperatoren

$$\vec{x} \leq \vec{y} \quad \Leftrightarrow \quad \forall k : x_k \leq y_k$$

$$\vec{x} \not\leq \vec{y} \quad \Leftrightarrow \quad \exists k : x_k > y_k$$

Definitionen

- Ein Prozess P_i heißt **blockiert** $\Leftrightarrow \quad \vec{a}_i \not\leq \vec{v} - \sum_{k=1}^{m} \vec{b}_k = \vec{f}$,

  d.h. wenn seine aktuelle Anforderung derzeit nicht erfüllbar ist
- Eine Prozessmenge $\boldsymbol{P} = \{P_1, P_2, \ldots, P_m\}$ heißt **verklemmt** $\Leftrightarrow$

$$\exists I \subseteq \{1,2,\ldots,m\} : \forall k \in I : \vec{a}_k \not\leq \vec{v} - \sum_{k \in I} \vec{b}_k$$

  d.h. es gibt eine Teilmenge von Prozessen, deren Anforderungen nicht in allen Komponenten erfüllbar sind durch den Vorrat, der nicht von Prozessen dieser Teilmenge belegt ist
- Beispiel

$\vec{a}_1 = (0,1) \quad \vec{a}_2 = (2,0) \quad \vec{b}_1 = (2,3) \quad \vec{b}_2 = (1,1)$

$I = \{1,2\},\ n = 2 \qquad \vec{v} = (4,4)$

$\vec{v} - \sum_{k \in I} \vec{b}_k = (4,4) - (2,3) - (1,1) = (1,0)$

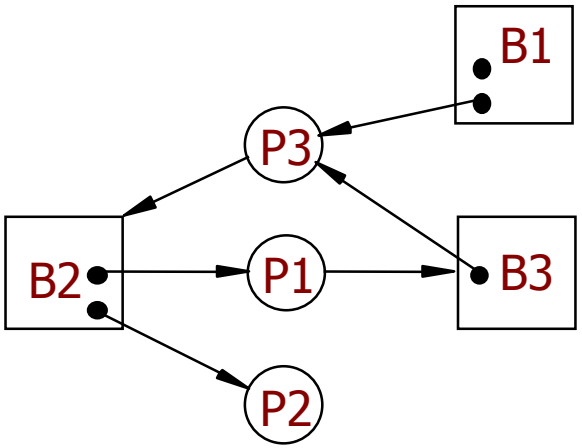
$\vec{a}_1 = (0,1) \not\leq (1,0) \qquad \vec{a}_2 = (2,0) \not\leq (1,0)$

Betriebsmittelgraph

- Formale Darstellung von Anforderungs- und Belegungssituationen
- Definition Betriebsmittelgraph
  - Sei P die Menge der Prozesse, BM die Menge der Betriebsmitteltypen
  - Ein gerichteter Graph (V, E) mit $V = P \cup BM$ und der folgenden Pfeilsemantik heißt **Betriebsmittelgraph**:

    $(p,b) \in E \Leftrightarrow$ Prozess p fordert eine Einheit von BM-Typ b

    $(b,p) \in E \Leftrightarrow$ Prozess p besitzt eine Einheit von BM-Typ b

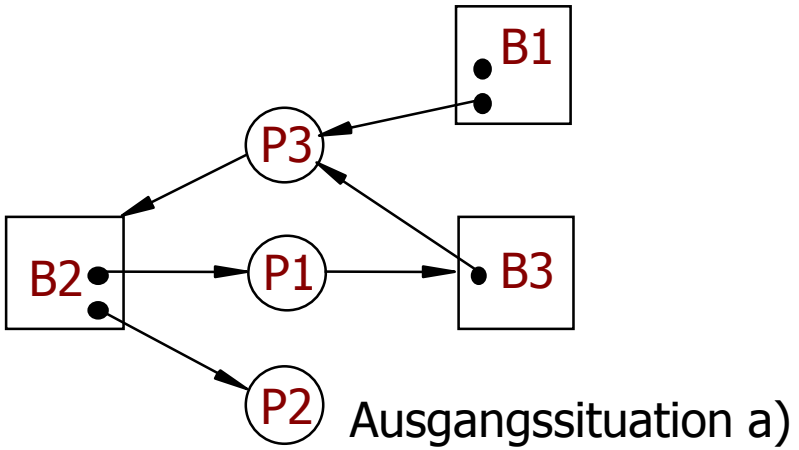


Betriebsmittelgraph: Eigenschaften und Operationen

- Der BM-Graph ist bezüglich der Knotenmengen P (Kreise) und BM (Rechtecke) **bipartit**
  - ⇒ es gibt nur Kanten von P nach B oder umgekehrt
- Menge von Punkten im Knoten = Anzahl der insgesamt verfügbaren Einheiten eines Betriebsmitteltyps
  - ⇒ bestimmt den maximalen Ausgangsgrad des BM-Knotens
- Ein Zyklus im BM-Graph weist auf eine potentielle Verklemmungssituation hin
  - ⇒ nur eine notwendige, aber keine hinreichende Bedingung für die Existenz einer Verklemmung (siehe Beispiel)
- Jede Operation (Anfordern, Belegen, Freigeben) bedeutet eine Graphtransformation (Hinzufügen bzw. Entfernen von Kanten)
- Beendigung eines Prozesses = Freigabe aller belegter BM

Betriebsmittelgraph: Reduktionen

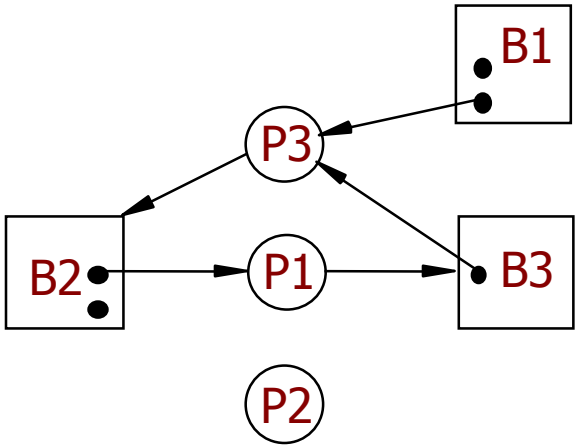
- BM-Graph **reduzierbar** ⇔ es gibt einen Prozess, dessen Anforderungen sofort erfüllbar sind und alle seine Kanten entfernt werden können
- BM-Graph **vollständig reduzierbar** ⇔ es gibt eine Folge von Reduktionen, so dass am Ende alle Kanten entfernt sind
- Verklemmungstheorem für BM-Graphen:
  - Aktuelle BM-Situation verklemmt ⇔ dazugehöriger BM-Graph ist nicht vollständig reduzierbar
- Beispiel für eine Reduktion (Ausgangsgraph)



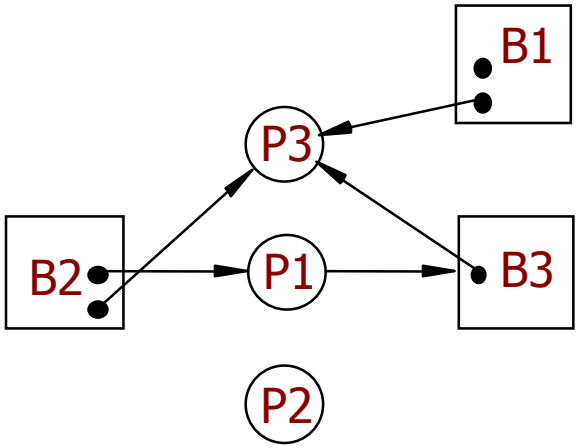
Ausgangssituation a)

Beispiel (Fortsetzung)

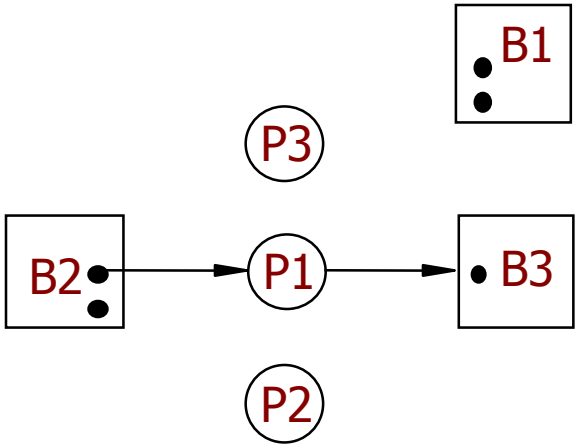
b)



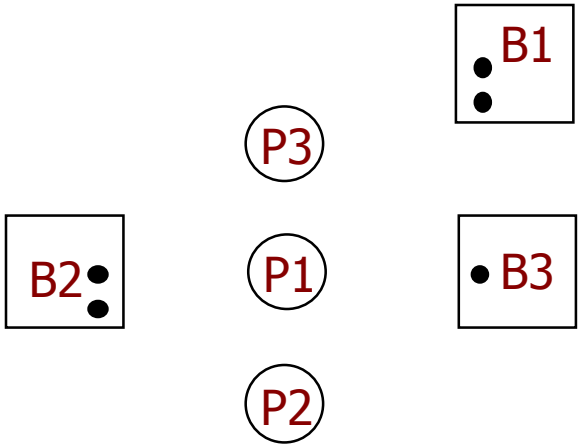
c)



d)



e)



5.7 Verklemmungsvermeidung (deadlock avoidance)

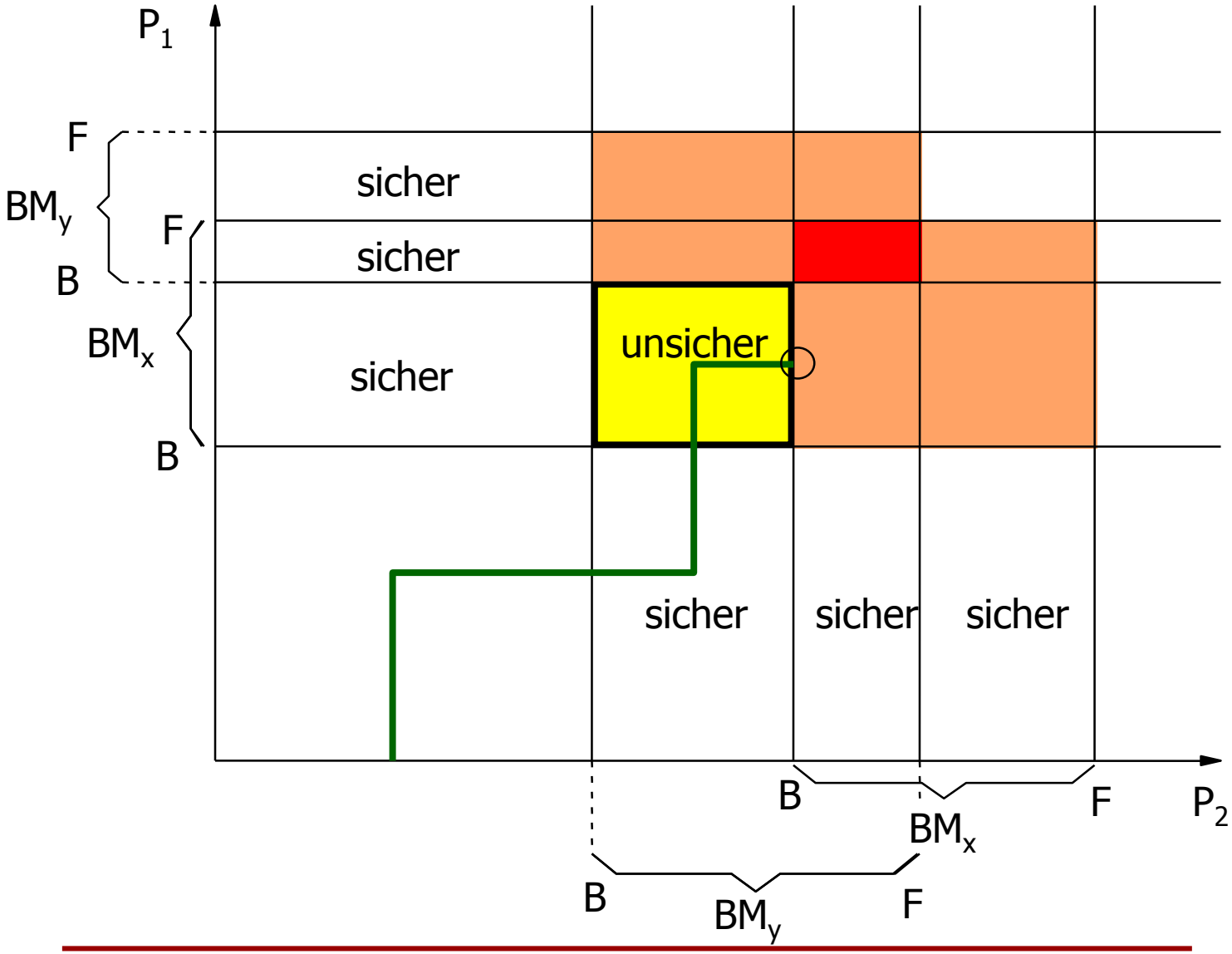
- Wie kann man eine Verklemmung vermeiden?
  - Aktuelle Situation kennen ⇒ BM-Graph
  - Restanforderungen der Prozesse müssen bekannt sein
  - Eine Belegungsstrategie anwenden, so dass kein Wartezyklus entsteht

- Problem
  - Restanforderungen und Zeitpunkt der Bekanntgabe meistens unbekannt
  - Im ungünstigsten Fall werden alle Restforderungen augenblicklich gestellt

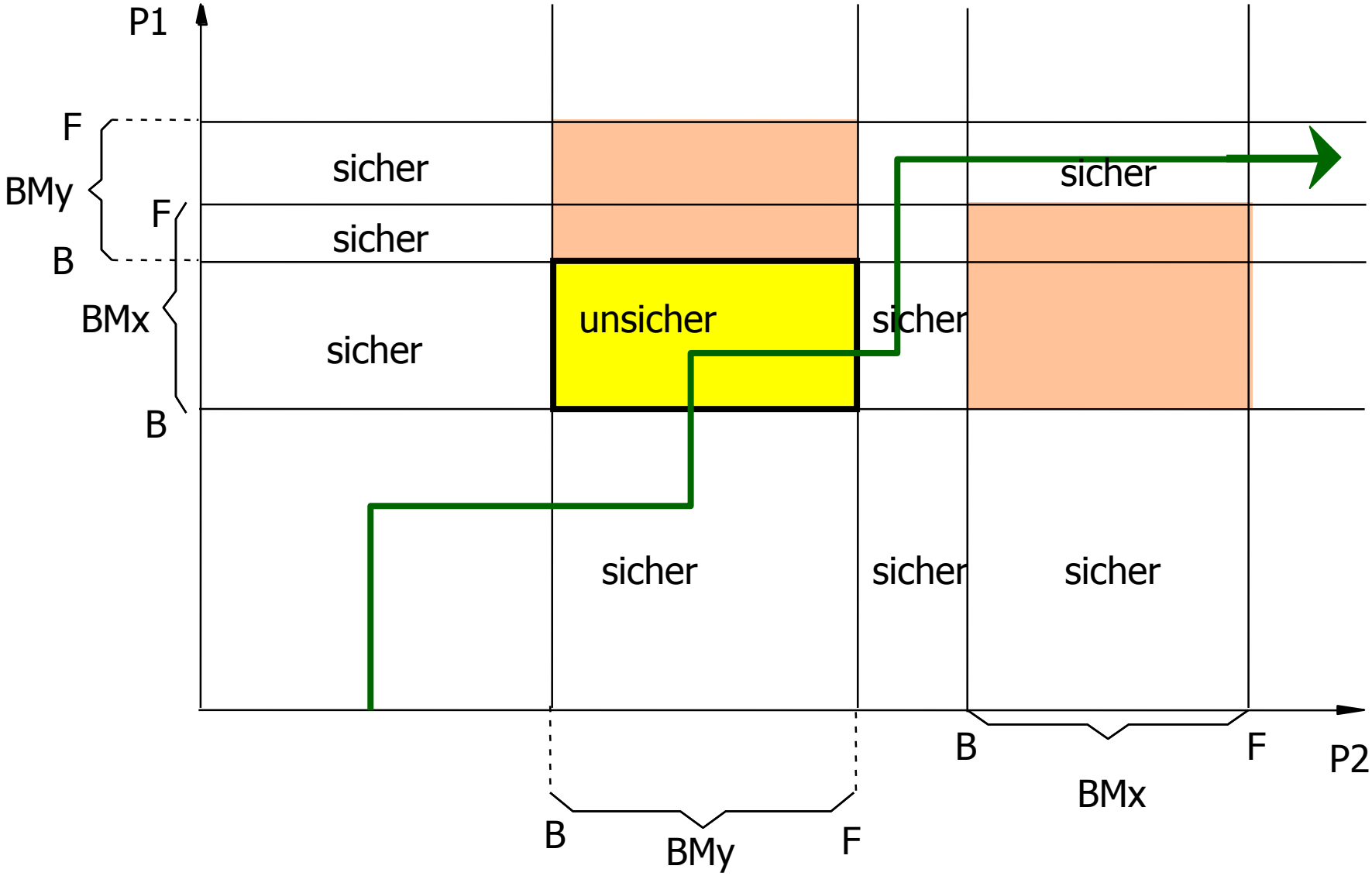
- Definition unsichere Betriebsmittelsituation
  - Es existiert eine Teilmenge von Prozessen, deren Restanforderungen nicht alle erfüllbar sind

  ⇒ Aktuelle Anforderungen mögen zwar noch erfüllt werden, aber bei **ungünstiger Reihenfolge** der Anforderungen und Freigaben kann es zur Verklemmung kommen

Belegungstrajektorie (mit Verklemmung)



Belegungstrajektorie (ohne Verklemmung)



Verklemmungsvermeidung

- Vermeidung = BM-Vergabe so, dass keine unsichere Situation eintritt
  ⇒ Anforderungen nur dann gewährt, wenn sie in eine sichere Situation führen
- Eine Prozessmenge P = {P1, P2, ... , Pm} heißt **sicher** ⇔

$$\exists \text{ Permutation } P_{k_1}, P_{k_2}, \ldots, P_{k_m} \quad \text{mit} \quad \forall r \in \{1,2,\ldots, m\} : \vec{r}_{k_r} \leq \vec{f} + \sum_{s=1}^{r-1} \vec{b}_{k_s}$$

bzw. $$\forall r \in \{1,2,\ldots, m\} : \vec{r}_{k_r} \leq \vec{v} - \sum_{s=r}^{m} \vec{b}_{k_s}$$

- Ausgangssituation = „Alle Restforderungen kommen gleichzeitig an"
  - Können alle Anforderung erfüllt werden ⇒ Situation ist *sicher* (kommt nur in unterausgelasteten Systemen vor)
  - Üblicherweise entsteht eine *unsichere* Situation
    ⇒es gibt eine Teilmenge von Prozessen, bei denen die aktuellen Anforderungen zwar erfüllt sind, aber ein Problem bei den Restforderungen entstehen **kann**
    - Bei ungünstiger Reihenfolge der Anforderungen und Freigaben, kann eine Verklemmung entstehen

Banker-Algorithmus

- Banker einer Kleinstadt gewährt Kredite (*max. Höhe bekannt*)
  - Kredite werden nicht auf einmal angefordert ⇒ Reserviert nur einen Teil des notwendigen Geldes (hier 4 Kunden / 10 Einheiten)
  - Anforderungen so erfüllen, dass der Banker durch Verzögerung der Zahlung alle Wünsche berücksichtigen kann

  ⇒ Genug Geld für die erste Anforderung bereithalten

  ⇒ Mit zurückgezahltem Kredit wird ein anderer Kredit gedeckt, usw.

	Akt.	*Max*
P1	0	6
P2	0	5
P3	0	4
P4	0	7

Frei: 10

Sicher im Sinne des Banker Alg.

	Akt.	*Max*
P1	1	6
P2	1	5
P3	2	4
P4	4	7

Frei: 2

Sicher im Sinne des Banker Alg.

	Akt.	*Max*
P1	1	6
P2	2	5
P3	2	4
P4	4	7

Frei: 1

Unsicher

Banker Algorithmus

1. Wähle eine Zeile aus der Matrix *R* so aus, dass die
   - notierten Restforderungen kleiner/ gleich der freien Ressourcen in *f* sind
   - Falls eine solche Zeile nicht vorliegt, wird das System u. U. in ein Deadlock enden, da kein Prozess beendet werden kann
2. Nehme an, der zugehörige Prozess ist terminiert und gebe seine belegten Ressourcen frei

   $\Rightarrow$ aktualisiere den Vektor *f*
3. Wiederhole die Schritte 1 und 2, bis alle Prozesse terminiert sind (*sicherer Zustand*) oder eine Verklemmung auftritt

- Anmerkungen
  - Sind mehrere Prozesse für die Auswahl geeignet, spielt die Reihenfolge keine Rolle, da die Anzahl freier Ressourcen schlimmstenfalls gleich bleibt $\Rightarrow$ Einsatz einer Hilfsstrategie
  - Problematisch ist die Ermittlung maximal benötigter Ressourcen sowie die dynamische Änderung der Prozessanzahl während der Laufzeit

Beispiel

- Gegeben sei ein System mit m=4 Prozessen (i = 1,..,4) und n=2 BM-Typen (j = 1,2) . Die aktuelle Situation sei durch die folgenden Größen beschrieben:

Belegungen:

$$B = \begin{pmatrix} 0 & 4 \\ 1 & 0 \\ 3 & 0 \\ 5 & 4 \end{pmatrix}$$

Gesamtanforderungen:

$$G = \begin{pmatrix} 4 & 6 \\ 2 & 4 \\ 3 & 1 \\ 10 & 6 \end{pmatrix}$$

Freie Betriebsmittel:

$$f = \begin{pmatrix} 3 & 2 \end{pmatrix}$$

Daraus berechnet man Restanforderungen R und die vorhandenen BM v:

$$R = \begin{pmatrix} 4 & 2 \\ 1 & 4 \\ 0 & 1 \\ 5 & 2 \end{pmatrix}$$

$$v = \begin{pmatrix} 12 & 10 \end{pmatrix}$$

Beispiel Banker-Algorithmus (4 Prozesse, 2 BM-Typen)

Vorhandene BM **v**

BM1	*BM2*
12	10

Belegungen **B**

	BM1	*BM2*
P1	0	4
P2	1	0
P3	3	0
P4	5	4

Gesamtforderung **G**

	BM1	*BM2*
P1	4	6
P2	2	4
P3	3	1
P4	10	6

Restforderung **R**

	BM1	*BM2*
P1	4	2
P2	1	4
P3	0	1
P4	5	2

Freie BM **f**

BM1	*BM2*
3	2

- Ist das System im sicheren Zustand?
  1. P3 kann beendet werden: R3=(0 1) ≤ f = (3 2)
     f = f + B3 = (3 2)+(3 0) = (6 2)
  2. P1 kann beendet werden: R3=(4 2) ≤ f = (6 2)
     f = f + B1 = (6 2)+(0 4) = (6 6)
  3. P2 kann beendet werden: R2=(1 4) ≤ f = (6 6)
     f = f + B2 = (6 6)+(1 0) = (7 6)
  4. P4 kann beendet werden: R4=(5 2) ≤ f = (7 6), f = (12 10)=v

⇒ Sichere Reihenfolge P3 P1 P2 P4

Verklemmungsvermeidung: Banker-Algorithmus

```
enum state {safe, unsafe, undefined}
state deadlock_avoidance(set_of_processes P, vector
  v, matrix r, matrix b, set_of_processes DP){
  vector f;
  state answer = undefined;
  DP = P;

  f: = v − Σ_{i=1}^{m} b_i ;

  while (answer == undefined) {
   if (∃ P_i ∈ DP: r_i ≤ f){
         DP = DP − {P_i};
         f = f + b_i  ;
         if (DP == ∅) answer= safe;
   }
   else answer = unsafe;
   return answer;
}
```

5.7 Verklemmungsentdeckung (deadlock detection)

- Restanforderungen nicht bekannt
  - ⇒ keine Verklemmungsvermeidung möglich
  - ⇒ Es kann zu Verklemmungen kommen
- Mindestreaktion: Erkennung einer Verklemmungssituation
- Indirekter Ansatz
  - ➢ Bestimmung der Prozesse, die nicht in einer Verklemmung involviert sind

- Definition
  - ➢ Eine Prozessmenge P1, P2, ... , Pm heißt **verklemmungsfrei** ⇔

$$\exists \text{ Permutation } P_{k_1}, P_{k_2}, .., P_{k_m} \quad \text{mit} \quad \forall r \in \{1,2,..., m\} : \vec{a}_{k_r} \leq \vec{f} + \sum_{s=1}^{r-1} \vec{b}_{k_s}$$

$$\text{bzw.} \quad \forall r \in \{1,2,..., m\} : \vec{a}_{k_r} \leq \vec{v} - \sum_{s=r}^{m} \vec{b}_{k_s}$$

d.h. es gibt eine Beendigungsreihenfolge der Prozesse derart, dass jede Anforderung durch das erfüllt werden kann, was von früher beendeten Prozessen freigegeben wird

Verklemmungsentdeckung

- Vergleich „verklemmungsfrei" und „sicher" zeigt weitgehende Übereinstimmung
  - Sicherheit= Permutation gesucht, die ausgeführt werden kann, wenn alle Restanforderungen auf einmal gestellt werden
  - Verklemmungsfreiheit = das gleiche bezüglich der aktuellen Anforderungen
  - ⇒ Banker-Algorithmus kann auch für Verklemmungs**entdeckung** verwendet werden
  - ⇒ Wir müssen nur die Restanforderungen durch die aktuellen Anforderungen ersetzen
- Der Algorithmus muss dabei alle Prozesse einbeziehen
- Der Entdeckungsalgorithmus kann aufgerufen werden
  - bei jeder Belegung
  - periodisch
  - im Leerlaufprozess
  - bei Verdacht (evtl. manuell ausgelöst)

Algorithmus zur Verklemmungsentdeckung

```
enum state {deadlock, no_deadlock, undefined};
state deadlock_detection(set_of_processes P, vector v,
   matrix a, matrix b, set_of_processes DP){
   vector f;
   state answer = undefined;
   DP = P;
   f: = v − ∑_{i=1}^{m} b_i ;
   while (answer == undefined) {
        if (∃ P_i ∈ DP: a_i ≤ f){
                DP = DP − {P_i};
                f = f + b_i;
                if (DP == ∅) answer= no_deadlock;
        }
        else answer = deadlock;
    return answer;
}
```

Beseitigung von Verklemmungen

- Nach der Erkennung einer Verklemmung können folgende Aktionen ausgeführt werden
  - Prozesse abbrechen: ein unbeteiligter Prozess, der das benötigte BM hat, wird anhand einer Strategie selektiert und abgebrochen
    - Hoffnung, alles arrangiert sich selbst wieder
    - Auswahlstrategien basierend auf Größe der Anforderung, Umfang belegter BM, Dringlichkeit, Restbedienzeit, Aufwand des Abbruchs
  - Prozesse zurücksetzen: Ein oder mehrere Prozesse werden auf den letzten gültigen Checkpoint zurückgesetzt und nach einer Wartespanne wieder gestartet
  - BM entziehen: einem Prozess werden – falls möglich – das von anderen Prozessen benötigte BM entzogen
    - nur für bestimmte BM-Arten anwendbar
- Allerdings: mögliche Datenverluste und -inkonsistenzen

Weiterführende Literatur

- Stallings,W.: *Operating Systems 6th ed*, Prentice Hall, 2009, Chapter 6
- Tanenbaum, A.: *Moderne Betriebssysteme*, 2.Aufl., Pearson, 2002, Kapitel 3
- Holt, R.: *Some Deadlock Properties of Computer Systems.* Computer Surveys, Sept. 1972
- Isloor,S.; Marsland, T.: *The Deadlock Problem: An Overview.* Computer, Sept. 1980