Dualität von Koordinierungstechniken
Theorie vs. Praxis

Problem	Methode
gegenseitiger Ausschluss	Schlossvariable, nicht blockierender Algor.
explizite Prozesssteuerung	Bedingungsvariable
bedingte Verzögerung	bedingter kritischer Abschnitt
Austausch von Zeitsignalen	Semaphor
Austausch von Daten	Nachrichtenpuffer (*bounded buffer*)

logisch betrachtet sind alle Methoden äquivalent, da jede von ihnen hilft, ein beliebiges Steuerungsproblem zu lösen

praktisch betrachtet sind die Methoden nicht äquivalent, da einige von ihnen für ein gegebenes Problem zu komplexen und ineffizienten Lösungen führen

Synchronisation
Koordination von Kooperation und Konkurrenz

- die Verfahren sind problemspezifisch und teils radikal unterschiedlich
  - einseitig oder mehrseitig
  - blockierend oder nicht-blockierend (wartend oder nicht-wartend)
- blockierende Verfahren erlauben die Wiederverwendung sequentieller Programme für nicht-sequentielle Ausführungsumgebungen
  - Schlossvariable, Bedingungsvariable, Semaphor, Monitor
  - die Gefahr von Verklemmungen ist stellenweise sehr hoch
- nicht-blockierende Verfahren sind frei von Verklemmungen, jedoch nicht unbedingt frei von Verhungerung
  - die Ansätze profitieren von Spezialbefehlen der CPU:
    - CISC `cas`, `cas2` (`dcas`), `cmpxchg`
    - RISC `ll/sc`
  - nicht-wartende Varianten beugen dem Verhungern von Prozessen vor
- nicht-sequentielle Programmierung ist nicht nur ein Betriebssystemfall

Überblick

Stillstand von Prozessen
Verklemmung mit passivem Warten durch Blockade

dead·lock **1** a standstill resulting from the action of equal and opposed forces; stalemate **2** a tie between opponents in the course of a contest **3** DEADBOLT — to bring or come to a deadlock

Der Begriff bezeichnet (in der Informatik)

> *[. . .] einen Zustand, in dem die beteiligten Prozesse wechselseitig auf den Eintritt von Bedingungen warten, die nur durch andere Prozesse in dieser Gruppe selbst hergestellt werden können. [2]*

- das „geringere Übel“ (im Vergleich zum *lifelock*), da dieser Zustand eindeutig erkennbar ist und so die Basis zur „Erholung“ gegeben ist
  - die verklemmten Prozesse sind im Einplanungszustand „blockiert“

Stillstand von Prozessen (Forts.)

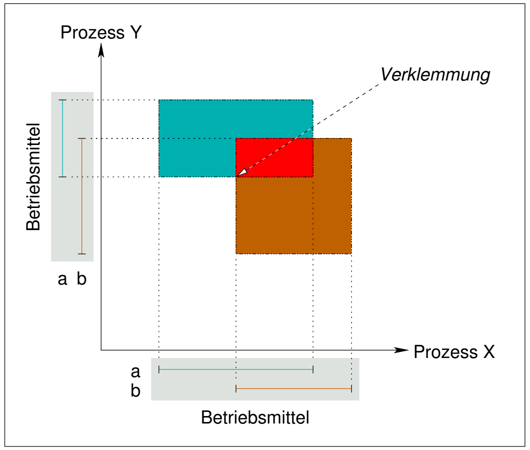
Verklemmung mit aktivem Warten

life·lock ist ...

- ein *deadlock*-ähnlicher Zustand, in dem die involvierten Prozesse zwar nicht blockieren, sie aber auch keine wirklichen Fortschritte in der weiteren Programmausführung erreichen
- wenn die an der Verklemmung beteiligten Prozesse wechselseitig aktiv auf die Bereitstellung von Betriebsmitteln warten:
  - ohne Prozessorabgabe $\mapsto$ *busy waiting*
  - mit Prozessorabgabe, in Laufbereitschaft bleibend $\mapsto$ *lazy waiting*
- das „größere Übel", da dieser Zustand nicht eindeutig erkennbar ist und damit die Basis zur „Erholung" fehlt
  - die verklemmten Prozesse haben die Einplanungszustände „laufend" oder „bereit" (d.h., jeden anderen außer „blockiert")
  - die Unterscheidung von unverklemmten Prozessen ist kaum möglich

Entstehung von Verklemmungen

Überlappender Zugriff auf gemeinsame unteilbare Betriebsmittel



Alles hängt davon ab,

1. ob sich die Prozesse überhaupt einander überlappen und
2. wie sich die Überlappung dann in Bezug auf die gemeinsamen Betriebsmittel zeigt.

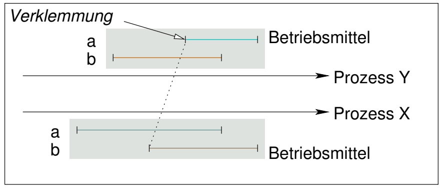
Keine Verklemmung, wenn...

P_X B_a erst nach P_Y oder

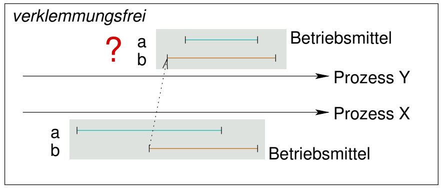
P_Y B_b erst nach P_X belegt.

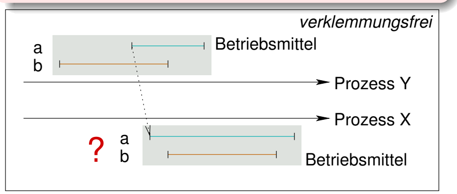
Entstehung von Verklemmungen (Forts.)

Nicht-deterministische Prozessabläufe und Betriebsmittelanforderungen



Vorbeugung oder Vermeidung von Verklemmungen ist durch geschickte Prozesseinplanung möglich, vorausgesetzt, die Prozesse wie auch ihre Betriebsmittelanforderungen sind alle bekannt.





☞ kooperative Ausführung von P_X und P_Y $\leadsto$ Verklemmungsfreiheit

Voraussetzungen für Verklemmungen

Notwendige und hinreichende Bedingungen

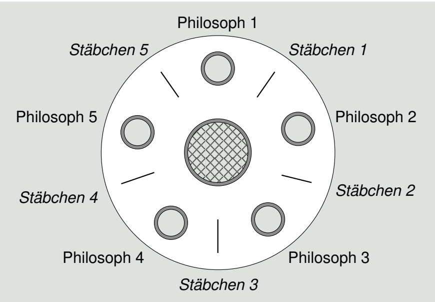
notwendige Bedingungen (müssen erfüllt sein, damit die Aussage zutreffen kann)

1. exklusive Belegung von Betriebsmitteln („*mutual exclusion*")
   - die umstrittenen Betriebsmittel sind nur unteilbar nutzbar
2. Nachforderung von Betriebsmitteln („*hold and wait*")
   - die umstrittenen Betriebsmittel sind nur schrittweise belegbar
3. kein Entzug von Betriebsmitteln („*no preemption*")
   - die umstrittenen Betriebsmittel sind nicht rückforderbar

hinreichende Bedingung (muss erfüllt sein, damit die Aussage zutrifft bzw. „bewiesen" ist)

4. zirkulares Warten (engl. *circular wait*)
   - Existenz einer geschlossenen Kette wechselseitig wartender Prozesse

Speisende Philosophen
Szenario



Fünf Philosophen, die nichts anderes zu tun haben, als zu denken und zu essen, sitzen an einem runden Tisch. Denken macht hungrig — also wird jeder Philosoph auch essen. Dazu benötigt ein Philosoph jedoch stets beide neben seinem Teller liegenden Stäbchen.

Prozess ↦ Philosoph
Betriebsmittel ↦ Stäbchen
- unteilbar

Verklemmung jeder Philosoph nimmt „gleichzeitig" das linke Stäbchen auf und greift anschließend auf das rechte zu...

Speisende Philosophen (Forts.)
Synchronisationsaspekte

mehrseitige Synchronisation gegenseitiger Ausschluss beim Gebrauch wiederverwendbarer Betriebsmittel
- keine zwei benachbarten Philosophen können gleichzeitig dasselbe Stäbchen gemeinsam benutzen

einseitige Synchronisation
- ein Philosoph muss warten, bis seine beiden Nachbarn ihm ein Stäbchen zur Verfügung gestellt haben

Randbedingungen
- jeder Philosoph fordert die Stäbchen nacheinander an
  - kein Philosoph legt ein Stäbchen zurück, wenn er feststellt, dass das andere bereits vom Nachbarn aufgenommen worden ist
- ein Philosoph kann seinem Nachbarn ein bereits aufgenommenes Stäbchen nicht entreissen

Speisende Philosophen (Forts.)
Umsetzung als nebenläufiges Programm

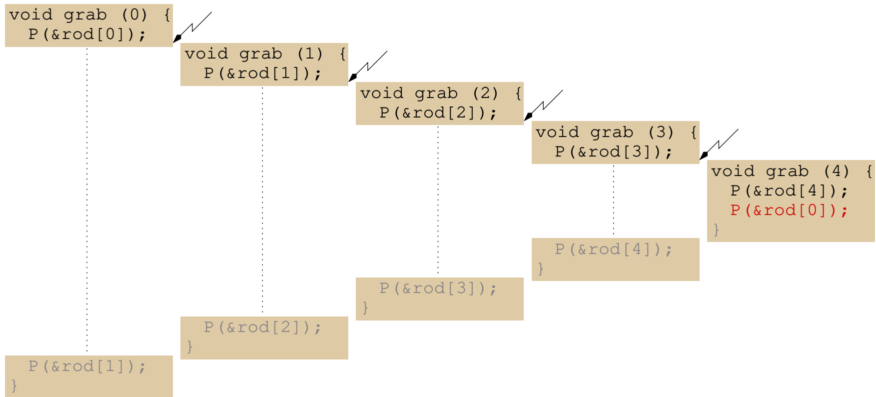
```
void phil (int who) {
    for (;;) {
        think();
        grab(who);
        eat();
        drop(who);
    }
}

void think () {}
void eat   () {}
```

```
semaphore rod[5] = {
    {1, 0}, {1, 0}, {1, 0}, {1, 0}, {1, 0}
};

void grab (int who) {
    P(&rod[who]);
    P(&rod[(who + 1) % NPHIL]);
}

void drop (int who) {
    V(&rod[who]);
    V(&rod[(who + 1) % NPHIL]);
}
```

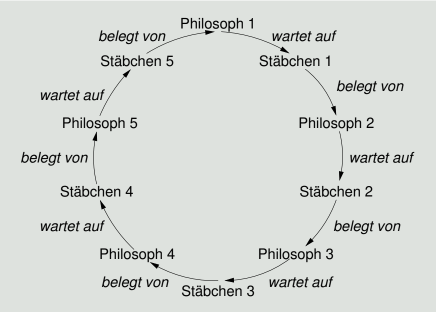
`P()` fordert zu einem Zeitpunkt nur ein Stäbchen (engl. *rod*) an

`V()` gibt ein Stäbchen frei

Speisende Philosophen (Forts.)
Philosoph 4 überlappt Philosoph 3 und nimmt Stäbchen 4 auf... GAU



Speisende Philosophen (Forts.)
Zirkulares Warten



Betriebsmittelgraph zeigt für jedes Betriebsmittel, welcher Prozess es belegt

Wartegraph verbucht für jeden einzelnen Prozess, auf welches Betriebsmittel er wartet

- ein geschlossener Kreis (im Wartegraphen) erfasst all die Prozesse, die sich zusammen im *Deadlock* befinden.
- es muss sichergestellt sein, dass ein solcher Kreis entweder nicht entstehen oder dass er erkannt und „durchbrochen" werden kann

Speisende Philosophen (Forts.)
Kritischer Abschnitt — Verklemmungsfreiheit bei minimaler Nebenläufigkeit

```
semaphore mutex = {1, 0};

void grab (int who) {
    P(&mutex);
    P(&rod[who]);
    P(&rod[(who + 1) % NPHIL]);
    V(&mutex);
}
```

Philosoph$_{who}$ greift auf die von ihm zum Essen benötigten beiden Stäbchen unteilbar zu

- ein binärer Semaphor (mutex) sorgt für gegenseitigen Ausschluss

Problem Philosoph$_{who}$ wird von seinen Nachbarn reihum unterbrochen

- Philosoph$_{who+1}$ muss auf Stäbchen$_{who}$ warten
  - er blockiert im kritischen Abschnitt, ohne diesen freizugeben
- Philosoph$_{who+2}$ muss auf die Freigabe des KA warten
  - ebenso ergeht es den beiden anderen Philosophen
- schlimmstenfalls kann immer nur ein Philosoph essen

Speisende Philosophen (Forts.)
Ein in der Praxis nicht selten anzufindendes Problem. . .

Überall dort, wo eine von mehreren Prozessen benutzte Routine (mindestens) zwei wiederverwendbare Betriebsmittel zugleich benötigt, diese aber nur nacheinander anfordert werden können:

- auf mehreren Magnetbändern vorliegende Daten sortieren
- einen kontinuierlichen Strom kodierter Informationen umkodieren
- Nachrichten von einem Eingangsport auf einen Ausgangsport leiten
- Pakete auf einem Ringnetz zur übernächsten Station durchschleusen
- Daten von dem einen *Backup Medium* auf ein anderes transferieren

Verklemmungsvorbeugung
(engl. *deadlock prevention*)

indirekte Methoden entkräften eine der Bedingungen 1–3

1. nicht-blockierende Verfahren verwenden
2. Betriebsmittelanforderungen unteilbar (atomar) auslegen
3. Betriebsmittelentzug durch Virtualisierung ermöglichen
   - virtueller Speicher, virtuelle Geräte, virtuelle Prozessoren

direkte Methoden entkräften Bedingung 4

4. lineare/totale Ordnung von Betriebsmittelklassen einführen:
   - Betriebsmittel B_i ist nur dann erfolgreich vor B_j belegbar, wenn i linear vor j angeordnet ist (d.h. $i < j$).

- Regeln, die das Eintreten von Verklemmungen verhindern
- Methoden, die zur Entwurfs- bzw. Implementierungszeit greifen

Verklemmungsvermeidung
(engl. *deadlock avoidance*)

Verhinderung von Wartezyklen durch strategische Maßnahmen:

- keine der drei notwendigen Bedingungen wird entkräftet
- fortlaufende Bedarfsanalyse schließt zirkulares Warten aus

Prozesse und ihre Betriebsmittelanforderungen sind zu steuern:

- das System wird (laufend) auf „unsichere Zustände" hin überprüft
- Zuteilungsablehnung im Falle nicht abgedeckten Betriebsmittelbedarfs
- anfordernde Prozesse nicht bedienen bzw. frühzeitig suspendieren
- Betriebsmittelnutzung einschränken $\leadsto$ „sichere Zustände"

☞ *à priori* Wissen über den maximalen Betriebsmittelbedarf ist erforderlich

Sicherer/Unsicherer Zustand
Speisende Philosophen

Ausgangspunkt fünf Stäbchen sind insgesamt vorhanden

- jeder der fünf Philosophen braucht zwei Stäbchen zum Essen

Situation P_1, P_2 und P_3 haben je ein Stäbchen; zwei Stäbchen sind frei

- P_4 fordert ein Stäbchen an $\rightarrow$ ein Stäbchen wäre dann noch frei
  - sicherer Zustand: einer von drei Philosophen könnte essen
  - die Anforderung von P_4 wird akzeptiert
- P_5 fordert ein Stäbchen an $\rightarrow$ kein Stäbchen wäre dann mehr frei
  - unsicherer Zustand: keiner der Philosophen könnte essen
  - die Anforderung von P_5 wird abgelehnt, P_5 muss warten
- haben vier Philosophen je ein Stäbchen, wird der fünfte gestoppt

Sicherer/Unsicherer Zustand (Forts.)
Leitungsvermittlung

Ausgangspunkt ein Vermittlungsrechner mit 12 Kommunikationskanälen

- Prozess P_1 benötigt max. 10 Kanäle, P_2 vier und P_3 neun

Situation: P_1 belegt fünf Kanäle, P_2 und P_3 je zwei; drei Kanäle sind frei

- P_3 fordert einen Kanal an, zwei blieben frei $\longrightarrow$ unsicherer Zustand
  - P_3 könnte noch sechs Kanäle anfordern: $6 > 2$
  - die Anforderung von P_3 wird abgelehnt, P_3 muss warten
- P_1 fordert zwei Kanäle an, einer bliebe frei $\longrightarrow$ unsicherer Zustand
  - P_1 könnte noch drei Kanäle anfordern: $3 > 1$
  - die Anforderung von P_1 wird abgelehnt, P_1 muss warten
- sichere Prozessfolge: $P_2 \rightarrow P_1 \rightarrow P_3$

Verklemmungsfreiheit
Verhinderung unsicherer Zustände

sicherer Zustand ist, wenn eine Folge der Verarbeitung vorhandener Prozesse existiert, in der alle Betriebsmittelanforderungen erfüllbar sind

unsicherer Zustand ist, wenn eine solche Folge nicht existiert; Erkennung dieses Zustands z.B. durch:

- Betriebsmittelbelegungsgraph (engl. *resource allocation graph*)
  - damit Vorhersage über das Eintreten von Zyklen treffen $\leadsto$ $O(n^2)$
  - bei jeder Betriebsmittelanforderung den Graphen überprüfen
- Bankiersalgorithmus (engl. *banker's algorithm* [54])
  1. Prozesse beenden ihre Operationen in endlicher Zeit
  2. Betriebsmittelbedarf aller Prozesse übersteigt nicht den Gesamtvorrat
  3. Prozesse definieren einen verbindlichen Kreditrahmen
  4. Betriebsmittelzuteilung erfolgt variabel innerhalb dieses Rahmens
- die Verfahrensweisen führen Prozesse dem *long-term scheduling* zu

Verklemmungserkennung
(engl. *deadlock detection*)

Verklemmungen werden (stillschweigend) in Kauf genommen. . .
- nichts ist im System verhindert das Auftreten von Zyklen
- keine der vier Bedingungen wird entkräftet

Ansatz: Wartegraph erstellen und auf Zyklen hin untersuchen $\rightsquigarrow$ $O(n^2)$
- zu häufige Überprüfung verschwendet Betriebsmittel/Rechenleistung
- zu seltene Überprüfung lässt Betriebsmittel brach liegen

Zyklensuche geschieht zumeist in großen Zeitabständen, wenn. . .
- Betriebsmittelanforderungen zu lange andauern
- die Auslastung der CPU trotz Prozesszunahme sinkt
- die CPU bereits über einen sehr langen Zeitraum untätig ist

Verklemmungsauflösung
Erholungsphase nach der Erkennungsphase

Prozesse abbrechen und dadurch Betriebsmittel frei bekommen
- verklemmte Prozesse schrittweise abbrechen (gr. Aufwand)
  - mit dem „effektivsten Opfer“ (?) beginnen
- alle verklemmten Prozesse terminieren (gr. Schaden)

Betriebsmittel entziehen und mit dem „effektivsten Opfer“ (?) beginnen
- der betreffende Prozess ist zurückzufahren bzw. wieder aufzusetzen
  - Transaktionen, *checkpointing*/*recovery* (gr. Aufwand)
- ein Aushungern der zurückgefahrenen Prozesse ist zu vermeiden

Gratwanderung zwischen Schaden und Aufwand:
- Schäden sind unvermeidbar und die Frage ist, wie sie sich auswirken

Nachlese . . .

Verfahren zum Vermeiden/Erkennen sind eher praxisirrelevant
- sie sind kaum umzusetzen, zu aufwändig und damit nicht einsetzbar
- zudem macht die Vorherrschaft sequentieller Programmierung diese Verfahren wenig notwendig

Verklemmungsgefahr ist lösbar durch Virtualisierung von Betriebsmitteln
- Prozesse beanspruchen/belegen ausschließlich logische Betriebsmittel
- der Trick besteht darin, in kritischen Momenten den Prozessen (ohne ihr Wissen) physische Betriebsmittel entziehen zu können
- dadurch wird die Bedingung der Nichtentziehbarkeit entkräftet

☞ eher praxisrelevant/verbreitet sind die Vorbeugungsmaßnahmen

Zusammenfassung

- Verklemmung bedeutet „*deadlock*“ oder „*lifelock*“
  - „[. . .] einen Zustand, in dem die beteiligten Prozesse wechselseitig auf den Eintritt von Bedingungen warten, die nur durch andere Prozesse in dieser Gruppe selbst hergestellt werden können“ [2]
  - dabei ist der *Lifelock* das größere Problem beider Verklemmungsarten
- für eine Verklemmung müssen vier Bedingungen gleichzeitig gelten
  - exklusive Belegung, Nachforderung, kein Entzug von Betriebsmitteln
  - zirkulares Warten der die Betriebsmittel beanspruchenden Prozesse
- Verklemmungsbekämpfung meint: Vorbeugen, Vermeiden, Erkennen
  - die Verfahren können im Mix zum Einsatz kommen