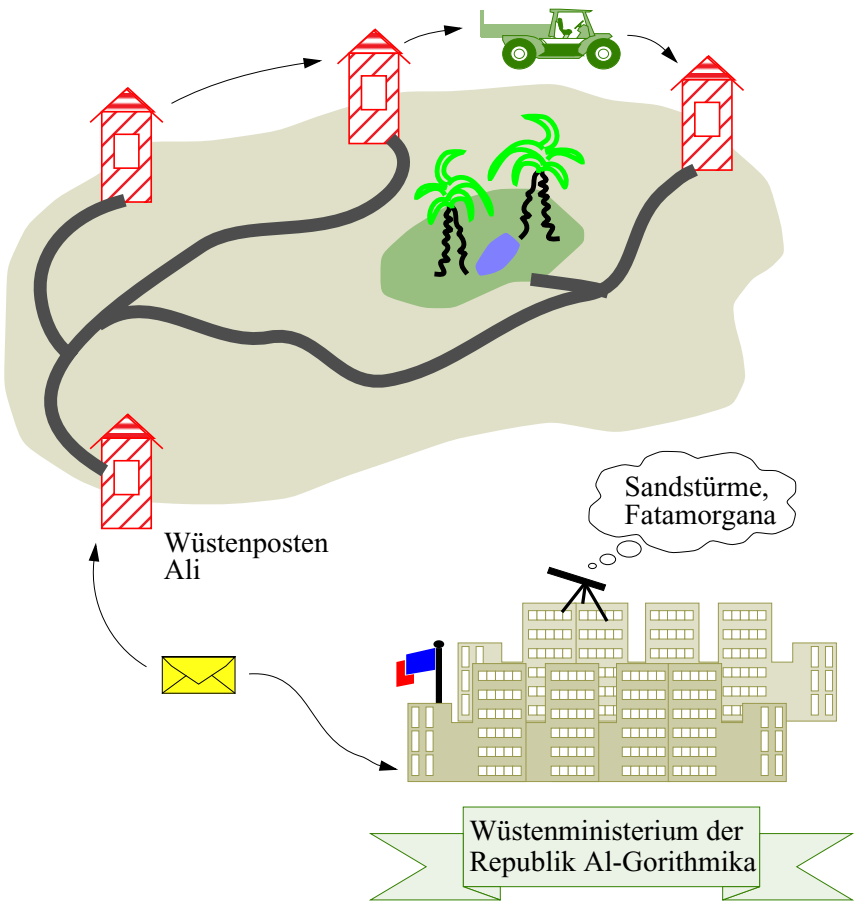
Das Problem der Wüstenposten



Vorschriften und Probleme...

- Reisende dürfen die Wüste nur "kontrolliert" (bei einem "wachen" Posten) betreten

- Nur ein wacher Posten lässt Personen in die Wüste einreisen

- Nur aus der Wüste kommende Reisende können einen Posten wecken

- Kontrolleure sollen die Beobachtungen der Wüstenposten an das Ministerium melden

Problem: Wann ist die Wüste leer?

- Fatamorgana und Sandstürme trüben die globale Sicht des Ministeriums

- Zählen der Reisende geht oft "schief", wie früher bereits eingesehen

Eine orientalische Lösung

- Reisende erhalten *Eintrittskarten*
- Rückgabe bei Austritt aus der Wüste
- Ministerium gibt feste Zahl von Tickets aus
- Ministerium sammelt Tickets wieder ein

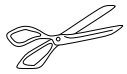
Invariante: Gesamtzahl der Tickets

- Aktiver Posten soll einen nichtleeren Vorrat an Tickets haben

- Wenn das Ministerium alle Tickets wieder eingesammelt hat, ist kein Reisender in der Wüste unterwegs und sind alle Posten passiv --> *terminiert*

Was tut ein Posten, wenn er sein letztes Ticket verkaufen müsste?

- Neue anfordern --> bürokratischer Aufwand!
- Orientalischer Trick: *Ticket halbieren* und nur das halbe Ticket verkaufen...



Puzzle-Abteilung im Ministerium...

Die Kreditmethode

- Idee:

  Verallgemeinerung des beim Integrationsbeispiel gefundenen Prinzips

  Also: Akkumulation eines Wertes, bis dieser =1

Bedingungen:

(1) Urprozess startet die verteilte Berechnung
(2) Prozesse und Nachrichten haben einen Kreditanteil $\in Q^+$
(3) Summe aller Kreditanteile stets = 1 ← Invariante
(4) Aktiver Prozess hat Kreditanteil > 0
(5) Nachricht hat Kreditanteil > 0

Theorem: Dann Berechnung *terminiert*, wenn der Urprozess die Kreditsumme 1 wiedererlangt hat

---

Man muss also die Bedingungen erfüllen ("*safety*") und dafür sorgen, dass der Urprozess die gesamte Kreditsumme wiedererhält ("*liveness*")

Kreditmethode - Realisierung

(1) Wird ein Prozess passiv, übermittelt er seinen Kreditanteil an den Urprozess.

(2) Der Kreditanteil einer ankommenden Nachricht wird dem Empfänger zugeschlagen.

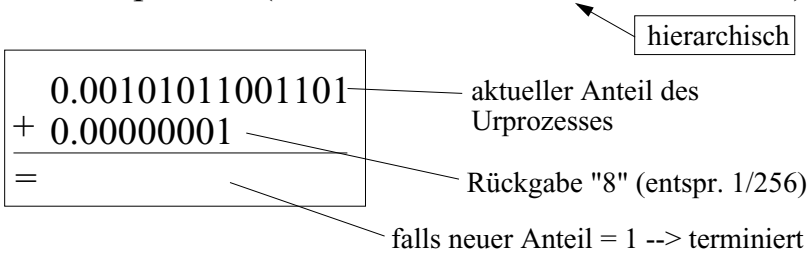
(3) Ausgesandte Nachricht erhält Hälfte des Anteils des Senders.

Implementierung:

(1) Gleitpunktzahlen für Kreditanteile unbrauchbar
Lösung: Bruchdarstellung

(2) Problem: Nenner schnell zu gross
Lösung: Da stets nur negative 2er Potenzen
--> nur Exponent des Nenners ("negativer Logarithmus")
--> halbieren: KREDIT := KREDIT + 1

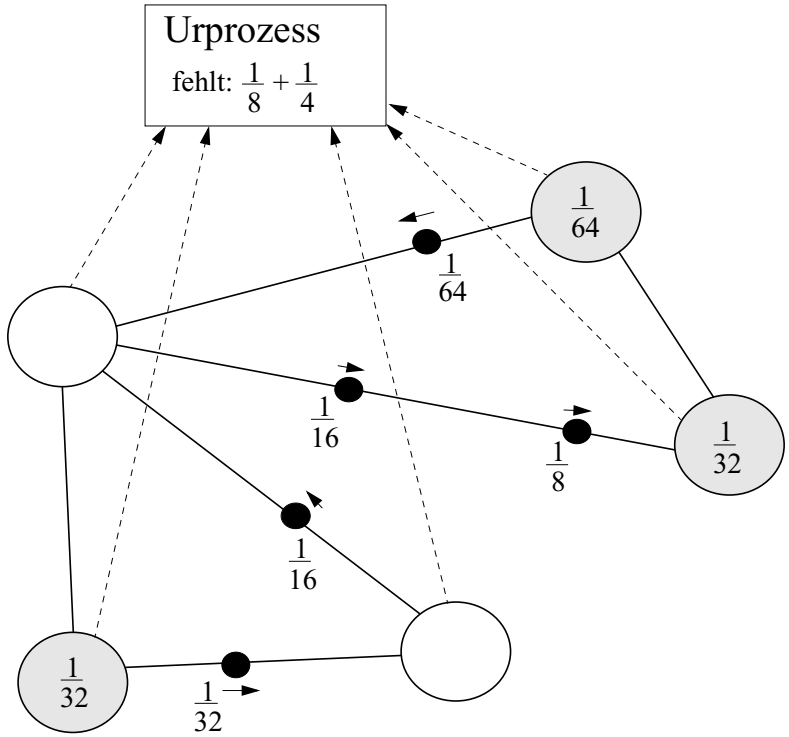
(3) Problem: Rekombination (Addition) der Anteile beim Urprozess (bzw. auch den anderen Prozessen)



(4) Problem: Länge der Bitleiste
Lösung: Anzahl *fehlender* Kreditanteile stets beschränkt ("klein") --> speichere *Komplement* als Menge

Kreditmethode - ein Beispiel

Alle Kreditanteile sind von der Form $\frac{1}{2^k}$

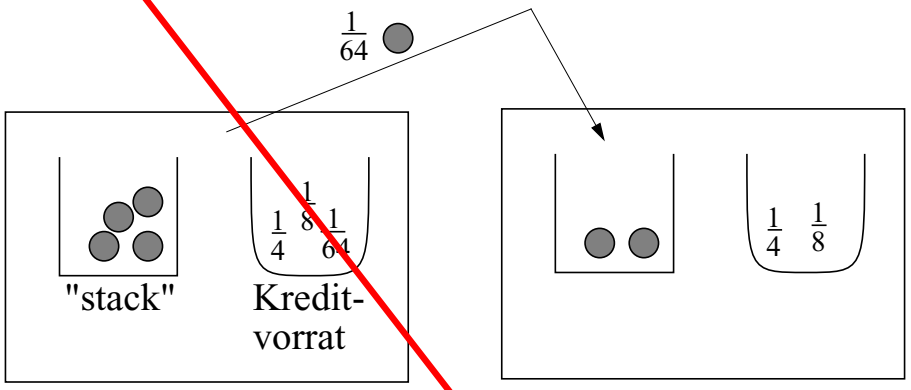


Der Urprozess muss nicht mehr "Krümel" halten, als davon im System (d.h. in aktiven Prozessen, in Basisnachrichten oder in Kontrollnachrichten zum Urprozess) sind.

(Der Urprozess kann die ihm noch fehlenden Krümel sogar oft kompakter speichern!)

Kreditmethode beim parallelen Berechnungsschema

1) Kreditanteile werden weitgehend dezentral von den Prozessoren verwaltet; lokale Arbeitseinheiten benötigen keinen Kredit.



2) Bei Versenden einer Arbeitseinheit: Diese bekommt einen Anteil aus dem lokalen Kreditvorrat

3) Lokaler Kreditvorrat darf bei einem "aktiven" Prozessor nie leer werden: ggf. muss ein Kreditanteil (der letzte) halbiert werden

4) Anteil eines ankommenden Arbeitseinheit wird dem Empfänger zugeschlagen; ggf. (falls gleicher Anteil dort schon vorhanden): iterativ rekombinieren (d.h. soweit möglich aufaddieren)

5) Wenn stack leer (oder auf explizite Anforderung): Kreditvorrat an den zentralen Prozess übermitteln

Rekombination der Kreditanteile

- Einsammeln der Kredite auch durch beliebigen Wellenalgorithmus möglich (initiiert durch Urprozess)
  - dabei Rekombination eingesammelter Kredite ggf. hierarchisch

---

- Rekombination der Kreditanteile beim *Urprozess*:
  - Initial: D = {0};
  - Bei Empfang von <KRÜMEL>:

steht für $2^{-KREDIT}$

```
K := KRÜMEL;
while K ∉ D begin
   D := D ∪ {K};
   K := K-1;
end;
D := D-{K}
if D = ∅ then terminated fi;
```

  - Wieso klappt dieses Schema?

Rekombination - Ein Beispiel

Initial: D = {0}

$\frac{1}{4}$ ("2") --> $\cup${2}, $\cup${1}, -{0} --> D = {2, 1}

$\frac{1}{2}$ ("1") --> -{1} --> D = {2}

$\frac{1}{64}$ ("6") --> --> D = {6, 5, 4, 3}

$\frac{1}{8}$ ("3") --> --> D = {6, 5, 4}

$\frac{1}{64}$ ("6") --> --> D = {5, 4}

$\frac{1}{16}$ ("4") --> --> D = {5}

$\frac{1}{32}$ ("5") --> --> D = {}
--> *Terminierung!*

$\Sigma = 1$

==> Prinzip: Binäre Subtraktion!

Kreditmethode - Bewertung

- Topologievoraussetzungen?
- Zentraler Urprozess: Engpass?
  - ggf. weitere Hierarchiestufen einführen
  - Kreditanteile durch einen Wellenalgorithmus einsammeln
- (Worst-case) Nachrichtenkomplexität? (= Overhead)
  - wenn passive Prozesse ihren Kreditanteil stets zurücksenden
  - wenn nicht mehr benötigte Anteile von einer Welle eingesammelt werden
- lok. Speicheraufwand? (Insbes. beim Urprozess)
  - wie gross kann die Menge dort gehaltener "Krümel" werden?
- lok. Berechnungsaufwand?
- "Detection Delay" (nach erfolgter Terminierung)?
- Qualitativer Vergleich mit anderen Verfahren?
- Varianten?

- Es gilt: Es gibt keinen Algorithmus zur Feststellung der verteilten Terminierung, der eine bessere Worst-case-Nachrichtenkomplexität als O(m+n) hat
  - m = Anzahl der Basisnachrichten; n = Anzahl der Prozesse
  - wie könnte man eine solche Aussage beweisen?
  - Kreditmethode ist daher "worst-case-optimal"!

Variante: "Nachlaufverfahren"

- Idee: Kontrollnachrichten laufen auf "benutzten" Kanälen den Basisnachrichten hinterher, um die Kredite zurückzufordern
- Überholen dabei keine Basisnachrichten (wie realisieren?)
- Spalten sich bei Prozessen "geeignet" auf (Kreditanteile)
- Kehren (direkt oder indirekt) zum Urprozess zurück, wenn kein benutzter Kanal mehr existiert

---

- Eine spezielle Ausprägung davon: hinter *jeder* Nachricht *direkt* herlaufen, um den Kredit wieder zurückzuholen
  - *Diesen* Typ von Kontrollnachrichten kann man sich sparen! ("Verheiraten" mit der zugehörigen Basisnachricht)
  - Acknowledgement wird sofort generiert, wenn der Empfänger aktiv ist.
  - Aber wenn der Empfänger passiv ist, wann dann?
    - Empfänger wartet mit der Rückgabe des Kredits (d.h. mit dem Acknowledgement), bis er selbst Acknowledgements zu allen von ihm selbst ausgesendeten Nachrichten erhalten hat
  - Kreditwerte zu versenden, kann man sich sparen: Man bekommt schliesslich genau den Wert zurück, den man versendet hat!
  - Was bleibt vom Verfahren also übrig? (--> Neuformulierung: Jeder Prozess zählt gesendete Nachrichten und empfangene Acknowledgements...)

Diese Idee *indirekter Acknowledgements* ("diffusing computations"-Verfahren von Dijkstra und Scholten, 1980) erinnert sehr an den *Echo-Algorithmus!*

Man könnte also überhaupt verteilte Berechnungen als Instanzen (einer einfachen *Variante*) des Echo-Algorithmus hinsichtlich der Explorer-Nachrichten auffassen. Die Echo-Nachrichten übernehmen dann wie gehabt die Erkennung der Terminierung!