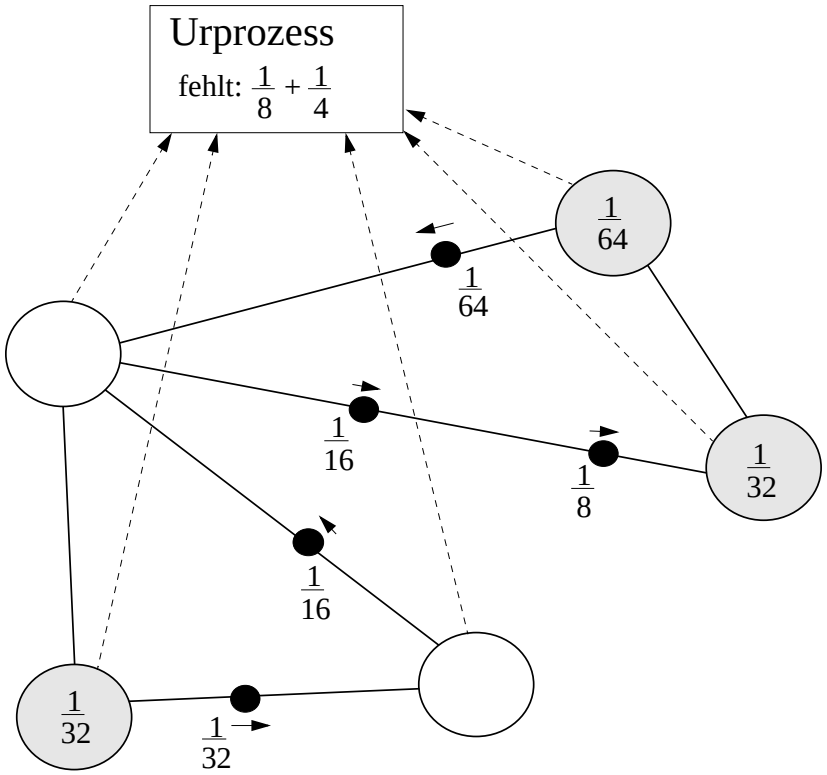
Kreditmethode - ein Beispiel

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

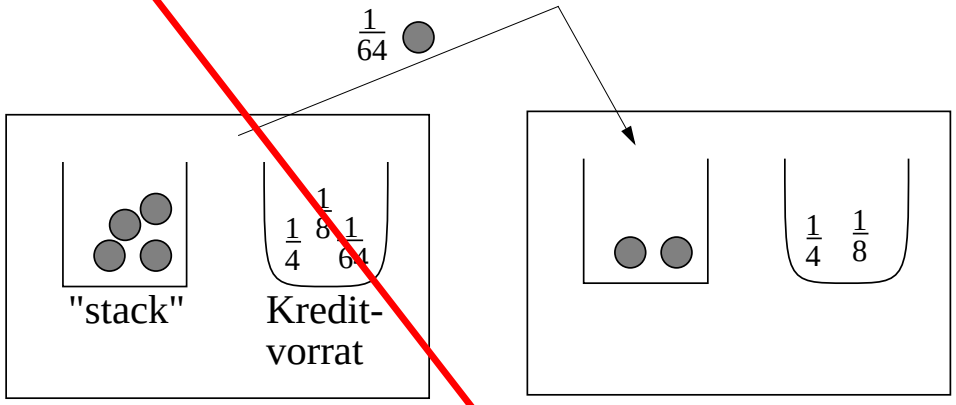


Der Urprozess muss nicht mehr "Krümel" halten, als davon im System (d.h. in aktiven Prozessen, in Basisnachrichten oder in Kontrollnachrichten zum Urprozess) sind.

(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 $2^{-\text{KREDIT}}$

```
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!

Denkübung: Wie genau sieht die Variante des Echo-Algorithmus aus?

- Tip: dynamischer Graph mit ggf. parallelen Kanten
- Wann Echo-Nachricht versenden? Gesamtzahl?