

Verteilte Systeme

Aufgabe 2

7

Beispiel: Verteilter Algorithmus

- ♦ **Satz von Euklid:** Der grösste gemeinsame Teiler (ggT) zweier positiver ganzer Zahlen x, y (mit $x \geq y > 0$) ist gleich dem ggT von y und dem Rest, der bei ganzzahliger Division von x durch y entsteht
- ♦ **Eigenschaften:**
 - Man setzt nun noch $\text{ggT}(x, 0) := x$ für alle x
 - Rekursive Realisierung: $\text{ggT}(x, y) := \text{ggT}(y, \text{mod}(x, y))$
 - Offenbar ist $\text{ggT}(x, x) = x$ für alle x
- ♦ **Erweiterung:** $\text{mod}^*(x, y) := \text{mod}(x-1, y)+1$
- ♦ **Verteilter Algorithmus:**
 - Jeder Prozeß P_i hat seine eigene Variable M_i .
 - ggT aller am Anfang bestehender M_i wird berechnet:


```

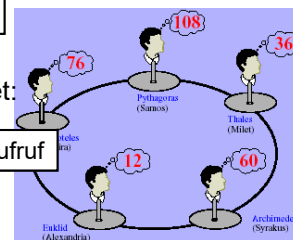
          {Eine Nachricht <y> ist eingetroffen}
          if y < Mi
          then Mi := mod(Mi, y)+1;
          send <Mi> to all neighbours;
          fi
          
```

Noch nicht geklärt:

- Wie wird der Algorithmus gestartet ?
- Wie erkennt man die Terminierung ?
- Wo steht das Ergebnis ?

Modifikation für die Verteilung

Entspricht rekursivem Aufruf



8

Beispiel: Verteilter Algorithmus

- ♦ Iterative Implementierung (für 2 Zahlen):

```
While b <> 0 do
  Ersetze (a,b) durch (b,(a mod b))
return a
```

(76,12) → (12,4)

- ♦ Rekursive Implementierung (für 2 Zahlen):

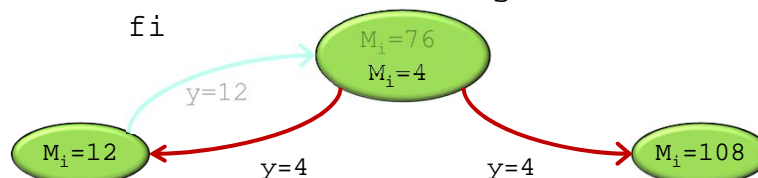
```
Procedure GGT (a,b)
  if b == 0 return a
  else return GGT(b,(a mod b))
```

GGT(76,12) → GGT(12,4)

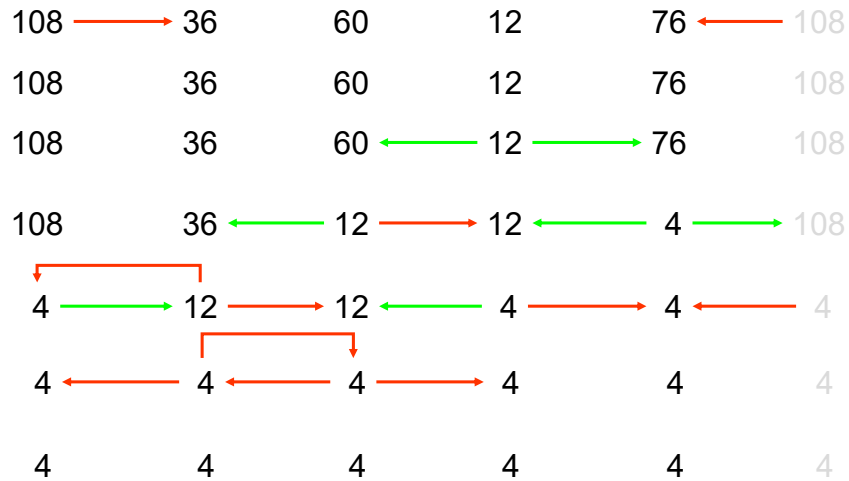
Beispiel: Verteilter Algorithmus

- ♦ Verteilte Implementierung (für n Zahlen):

```
{Eine Nachricht <y> ist eingetroffen}
  if y < Mi
    then Mi := mod(Mi-1,y)+1;
    send <Mi> to all neighbours;
  else do something else
fi
```

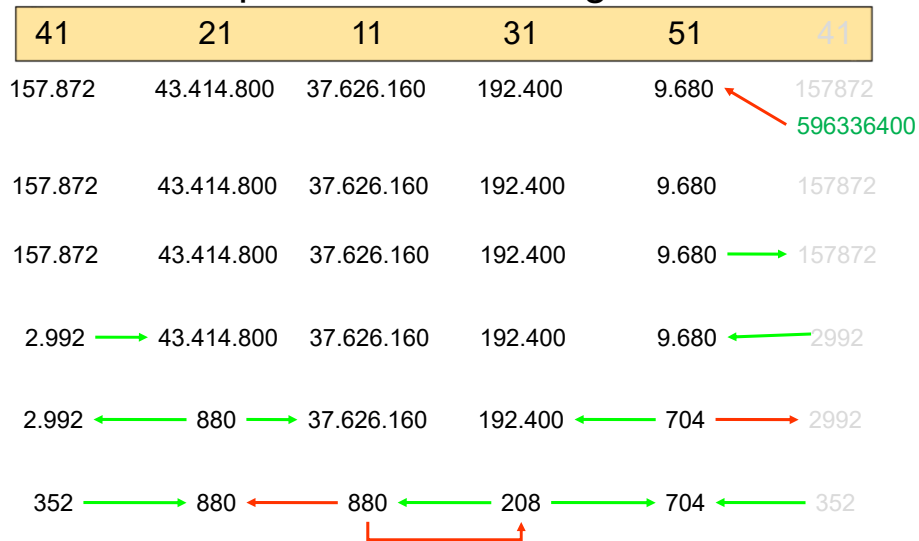


Beispiel 1: Verteilter Algorithmus



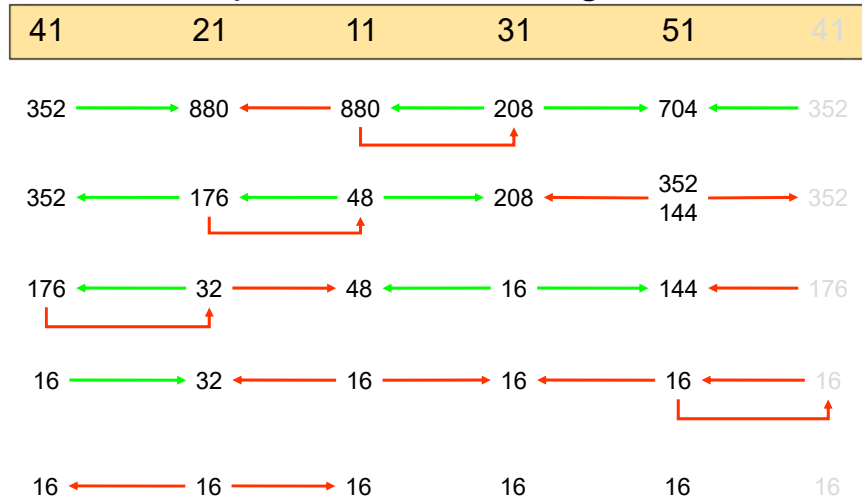
11

Beispiel 2: Verteilter Algorithmus



12

Beispiel 2: Verteilter Algorithmus



13

Verteilter Algorithmus

Ausschnitt aus der log-Datei

Koordinator-ko@Brummpa-KLC Startzeit: 17.06 10:13:52,969< mit PID <0.37.0>
 [...] Alle ggT-Prozesse gebunden.
 [...] Ring wird/wurde erstellt, Koordinator geht in den Zustand 'Bereit für Berechnung'.
 [...] Beginne eine neue ggT-Berechnung mit Ziel 16.
 ggT-Prozess 48851 (ggT@Brummpa) initiales Mi 9680 gesendet.
 [...] ggT-Prozess 48851 (ggT@Brummpa) startendes y 596336400 gesendet.
 [...] 48851 meldet Terminierung mit ggT 9680 um 17.06 10:15:52,000< (17.06 10:15:52,000<).
 [...] **Per Hand** 48841 die Zahl 9680 **gesendet** { , 48841, ggT@Brummpa } ! { sendy, 9680 }
 48841 meldet neues Mi 2992 um 17.06 10:18:36,499< (17.06 10:18:36,499<).
 48821 meldet neues Mi 880 um 17.06 10:18:39,100< (17.06 10:18:39,100<).
 48851 meldet neues Mi 704 um 17.06 10:18:39,100< (17.06 10:18:39,100<).
 48841 meldet neues Mi 352 um 17.06 10:18:42,700< (17.06 10:18:42,700<).
 48831 meldet neues Mi 208 um 17.06 10:18:42,700< (17.06 10:18:42,700<).
 48811 meldet neues Mi 880 um 17.06 10:18:42,700< (17.06 10:18:42,700<).
 48811 meldet neues Mi 48 um 17.06 10:18:45,200< (17.06 10:18:45,200<).
 48821 meldet neues Mi 176 um 17.06 10:18:45,200< (17.06 10:18:45,200<).
 ...

14

Verteilter Algorithmus

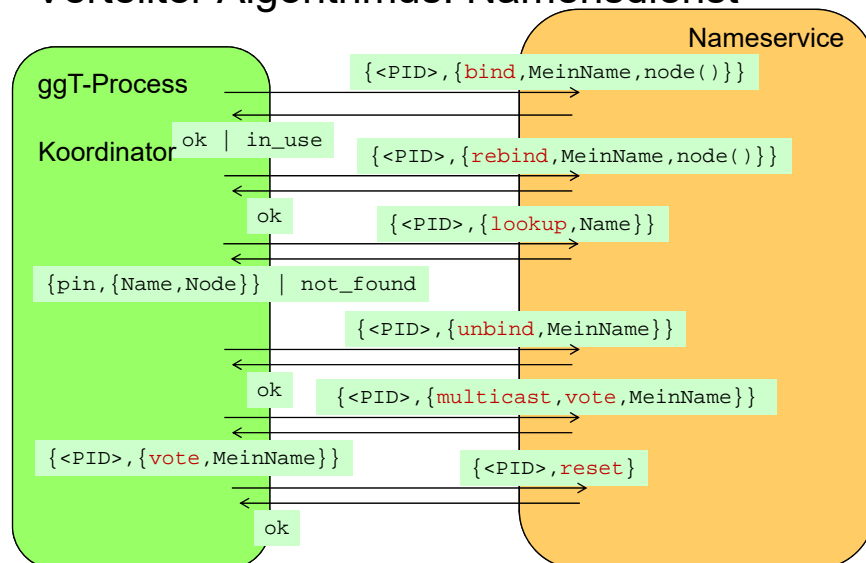
Ausschnitt aus der log-Datei

```

...
48851 meldet neues Mi 352 um 17.06 10:18:45,200< (17.06 10:18:45,200<).
48831 meldet neues Mi 16 um 17.06 10:18:48,800< (17.06 10:18:48,800<).
48821 meldet neues Mi 32 um 17.06 10:18:48,800< (17.06 10:18:48,800<).
48851 meldet neues Mi 144 um 17.06 10:18:48,801< (17.06 10:18:48,800<).
48841 meldet neues Mi 176 um 17.06 10:18:48,802< (17.06 10:18:48,800<).
48841 meldet neues Mi 16 um 17.06 10:18:51,300< (17.06 10:18:51,299<).
48811 meldet neues Mi 16 um 17.06 10:18:51,300< (17.06 10:18:51,300<).
48851 meldet neues Mi 16 um 17.06 10:18:51,300< (17.06 10:18:51,300<).
48821 meldet neues Mi 16 um 17.06 10:18:54,090< (17.06 10:18:54,090<).
48811 meldet Terminierung mit ggT 16 um 17.06 10:18:54,090< (17.06 10:18:54,090<).
48811 meldet Terminierung mit ggT 16 um 17.06 10:18:57,090< (17.06 10:18:57,090<).
48821 meldet Terminierung mit ggT 16 um 17.06 10:18:57,090< (17.06 10:18:57,090<).
48841 meldet Terminierung mit ggT 16 um 17.06 10:18:57,090< (17.06 10:18:57,090<).
[...]
```

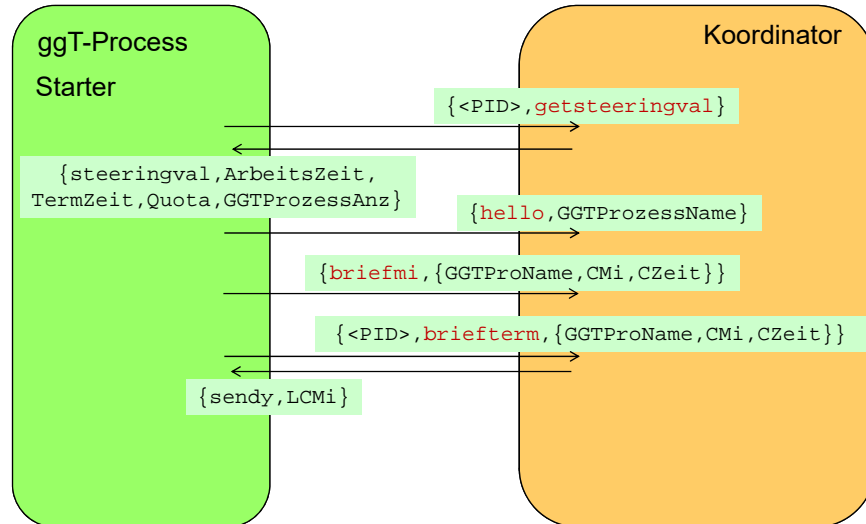
15

Verteilter Algorithmus: Namensdienst



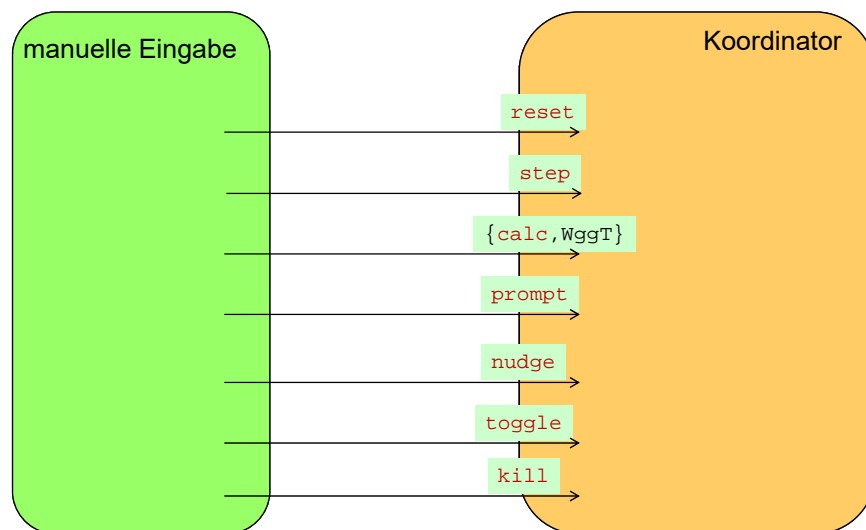
16

Verteilter Algorithmus: Koordinator



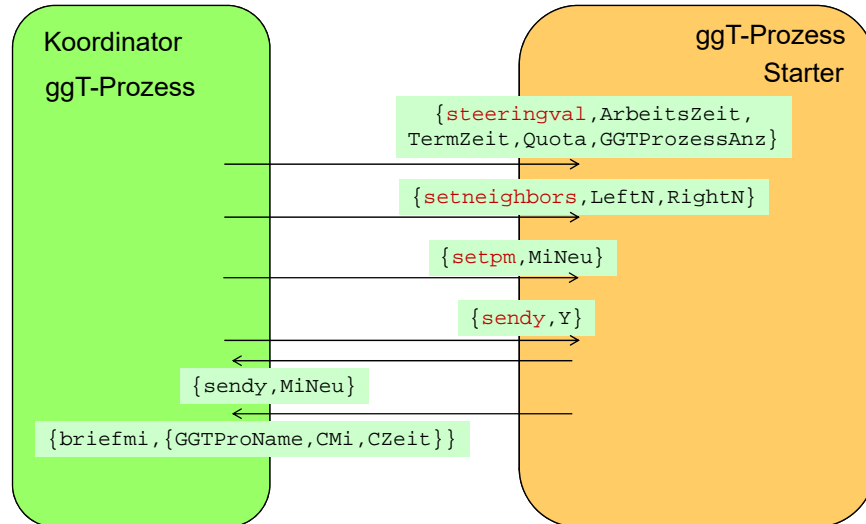
17

Verteilter Algorithmus: Koordinator



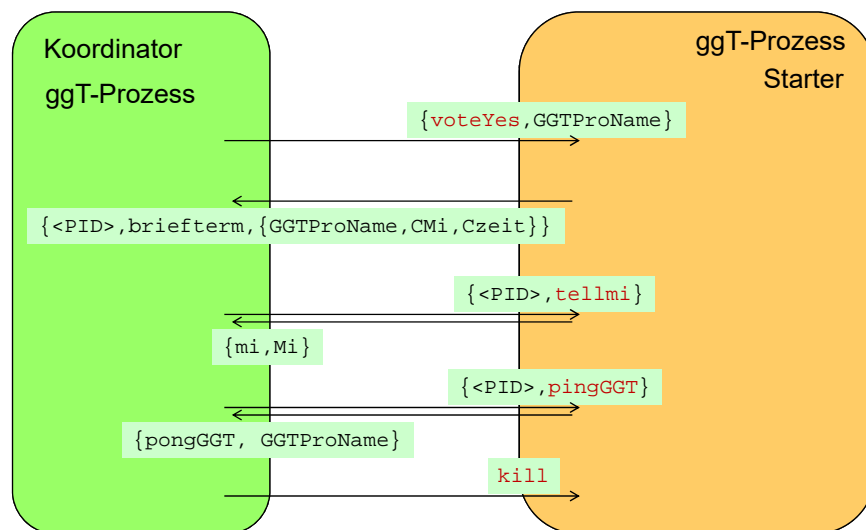
18

Verteilter Algorithmus: ggT-Prozess



19

Verteilter Algorithmus: ggT-Prozess



20

Namensdienst:
lab22

Anfang

```
(w)erl -(s)name ns (-setcookie zummsel)
1>nameservice:start( ).
%global:register_name(nameservice,NServerPid)
```

Namensdienst:
lab22

Anfang

NameS ! {self(),{rebind,KoName,KoNode}}

Koordinator
lab24

```
(w)erl -(s)name ko (-setcookie zummsel)
1>koordinator:start( ).
%net_adm:ping(NameServiceNode)
%register(KoName,KoPID)
```


Durchzuführende Tests

The screenshot shows a file explorer on the left with a list of files including `ggt.cfg`, `ggt_process.beam`, `ggt_starter.beam`, `ko_send.beam`, `koordinator.beam`, `koordinator.cfg`, `nameservice.beam`, `ns_send.beam`, `Readme.txt`, and `start_starter.beam`. To the right, three terminal windows are open, each running Erlang code. The first terminal shows the command `(auskunft@Qigong42)1> nameservice:start().`. The second terminal shows `(central@Qigong42)1> koordinator:start().`. The third terminal shows `(kaunto@Qigong42)1> start_starter().`. The output of the third terminal shows the start of the `start_starter` process, including file loading and registration details.

23

Anfang

Namensdienst:
lab22Koordinator
lab24

$$\text{NameS} = \{ \text{self()}, \{ \text{rebind}, \text{Clientname}, \text{self()} \} \}$$

Starter 1, Gruppe 08

Starter 2, Gruppe 08

ggT488311

```
(w)erl -(s)name ggt (-setcookie zummse1)
1>start_starter:go(Anzahl,Start).
%net_adm:ping(NameServiceNode)
%register(Clientname,self())
```

24

Durchzuführende Tests

[illegible]

Anfang

