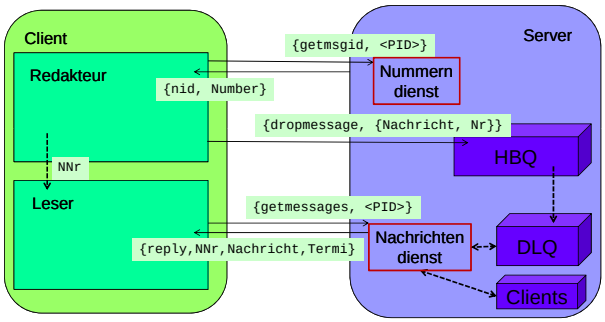
Verteilte Systeme

Aufgabe 1

Nachrichtendienst



Beispiel: server.log

Server Startzeit: 30.04 17:05:26,874| mit PID <0.2728.0>
Server: Nachrichtennummer 1 an <9447.9306.0> gesendet
Server: Nachrichtennummer 2 an <9447.9307.0> gesendet […]
4-client@Brummpa-<0.9310.0>-KLC: 5te_Nachricht. C Out: 30.04 17:05:31,797|(5); HBQ In: 30.04 17:05:31,796|-dropmessage
Server: Nachrichtennummer 6 an <9447.9310.0> gesendet
3-client@Brummpa-<0.9309.0>-KLC: 4te_Nachricht. C Out: 30.04 17:05:31,797|(4); HBQ In: 30.04 17:05:31,796|-dropmessage
Server: Nachrichtennummer 7 an <9447.9309.0> gesendet […]
0-client@Brummpa-<0.9306.0>-KLC: 1te_Nachricht. C Out: 30.04 17:05:31,797|(1); HBQ In: 30.04 17:05:31,796| DLQ In:30.04 17:05:34,812|.(1)-getmessages von <9447.9310.0>-false […]
1-client@Brummpa-<0.9307.0>-KLC: 2te_Nachricht. C Out: 30.04 17:05:31,797|(2); HBQ In: 30.04 17:05:31,796| DLQ In:30.04 17:05:34,812|.(2)-getmessages von <9447.9310.0>-false […]
2-client@Brummpa-<0.9308.0>-KLC: 3te_Nachricht. C Out: 30.04 17:05:31,797|(3); HBQ In: 30.04 17:05:31,796| DLQ In:30.04 17:05:34,812|.(3)-getmessages von <9447.9310.0>-false […]
4-client@Brummpa-<0.9310.0>-KLC: 25te_Nachricht. C Out: 30.04 17:05:43,859|(25); HBQ In: 30.04 17:05:43,858| DLQ In:30.04 17:05:43,858|.(25)-getmessages von <9447.9310.0>-true […]
Server: Nachrichtennummer 40 an <9447.9306.0> gesendet
Client <9447.9310.0> wird vergessen! ********** […]
***Fehlernachricht fuer Nachrichten 26 bis 35 um 30.04 17:05:54,093|.
QVerwaltung1>>> Nachrichten [60,59,58,57,56,55,54,53,52,51,50,49,48,47,46,45,44,43,42,41,40,39,38,37,36] von HBQ in DLQ transferiert.
QVerwaltung1>>> Nachrichten [9,8,7,6,5,4,3,2,1] von DLQ geloescht. […]
Downtime: 30.04 17:06:55,812| vom Nachrichtenserver <0.2728.0>; Anzahl Restnachrichten in der HBQ:5

Beispiel: client.log

2-client@Brummpa-<0.9308.0>-KLC Start: 30.04 17:05:28,781|.
2-client@Brummpa-<0.9308.0>-KLC: 3te_Nachricht. C Out: 30.04 17:05:31,797| gesendet
2-client@Brummpa-<0.9308.0>-KLC: 8te_Nachricht. C Out: 30.04 17:05:34,813| gesendet […]
28te_Nachricht um 30.04 17:05:43,859| vergessen zu senden ******
0-client@Brummpa-<0.9306.0>-KLC: 1te_Nachricht. C Out: 30.04 17:05:31,797|(1); HBQ In: 30.04 17:05:31,796| DLQ In:30.04 17:05:34,812|. ; C In: 30.04 17:05:43,859|
1-client@Brummpa-<0.9307.0>-KLC: 2te_Nachricht. C Out: 30.04 17:05:31,797|(2); HBQ In: 30.04 17:05:31,796| DLQ In:30.04 17:05:34,812|. ; C In: 30.04 17:05:43,875|
2-client@Brummpa-<0.9308.0>-KLC: 3te_Nachricht. C Out: 30.04 17:05:31,797|(3); HBQ In: 30.04 17:05:31,796| DLQ In:30.04 17:05:34,812|.*******; C In: 30.04 17:05:43,875| […]
..getmessages..Done...
Neues Sendeintervall: 2 Sekunden (3). […]
***Fehlernachricht fuer Nachrichten 61 bis 70 um 30.04 17:06:09,530|. ; C In: 30.04 17:06:09,625| […]
4-client@Brummpa-<0.9310.0>-KLC: 71te_Nachricht. C Out: 30.04 17:05:57,453|(71); HBQ In: 30.04 17:05:57,452| DLQ In:30.04 17:06:09,530|. ; C In: 30.04 17:06:09,641|

Verteilte Systeme

Aufgabe 2

Beispiel: Verteilter Algorithmus

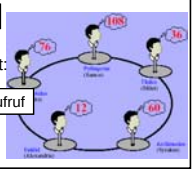
- **Satz von Euklid**: Der grösste gemeinsame Teiler (ggT) zweier positiver ganzer Zahlen x, y (mit x≥y>0) ist gleich dem ggT von y und dem Rest, der bei ganzzahliger Division von x durch y entsteht.
- **Eigenschaften**:
  - Offenbar ist **ggT(x,x) = x** für alle x
  - Man setzt nun noch **ggT(x,0) := x** für alle x
  - Rekursive Realisierung: **ggt(x,y) := ggt(y,mod(x,y))**
- **Erweiterung**: **mod*(x,y) := 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-1,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



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):

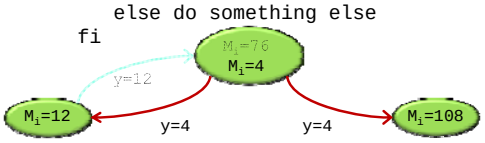
```
Procedure GGT (a,b)
   if b == 0 return a
      else return GGT(b,(a mod b))
```

GGT(76,12) → GGT(12,4)

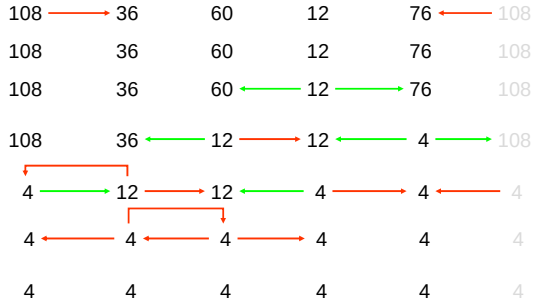
Beispiel: Verteilter Algorithmus

- Verteilte Implementierung (für n Zahlen):

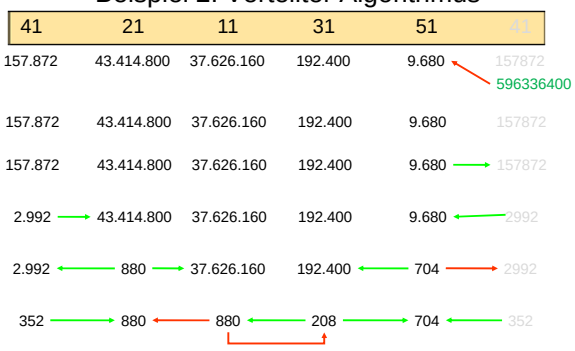
```
{Eine Nachricht <y> ist eingetroffen}
     if y < Mi
        then Mi := mod(Mi-1,y)+1;
             send <Mi> to all neighbours;
        else do something else
     fi
```



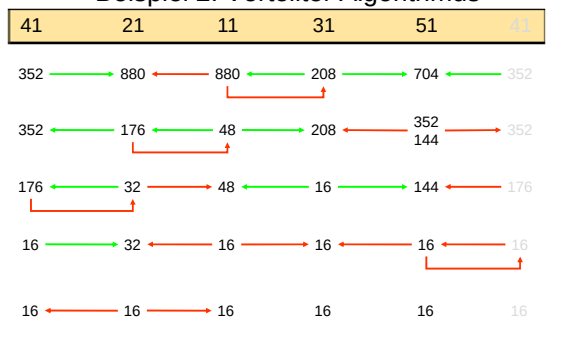
Beispiel 1: Verteilter Algorithmus



Beispiel 2: Verteilter Algorithmus



Beispiel 2: Verteilter Algorithmus



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<).
…

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<).
[...]

Namensdienst:
lab22

Anfang

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

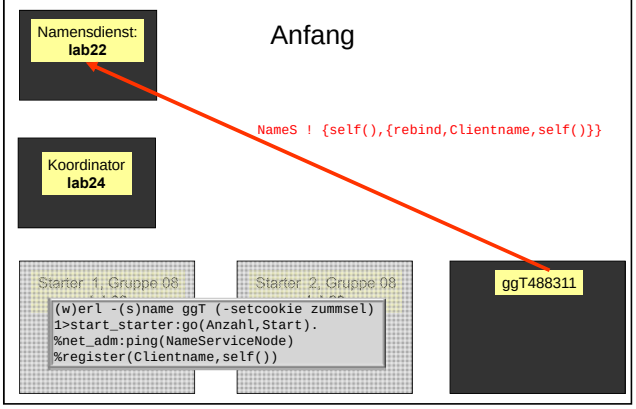

Namensdienst:
lab22

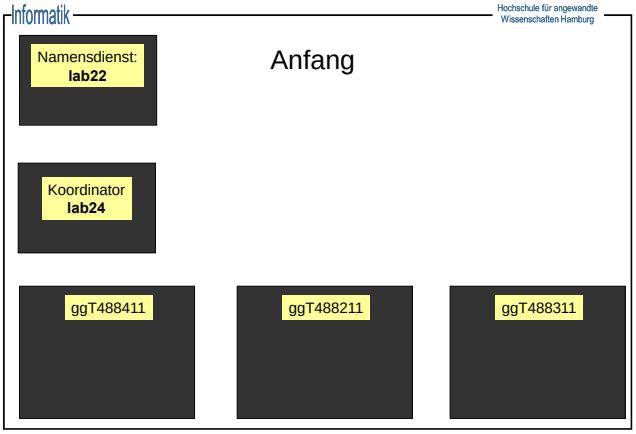
Anfang

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

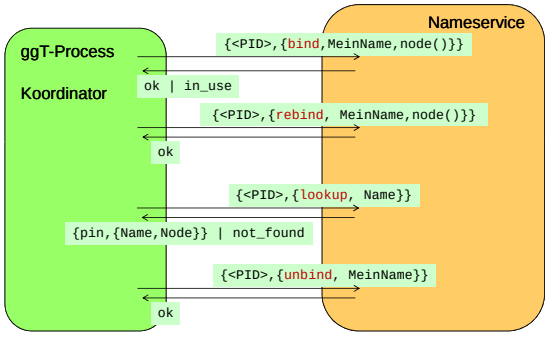
Koordinator
lab24

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

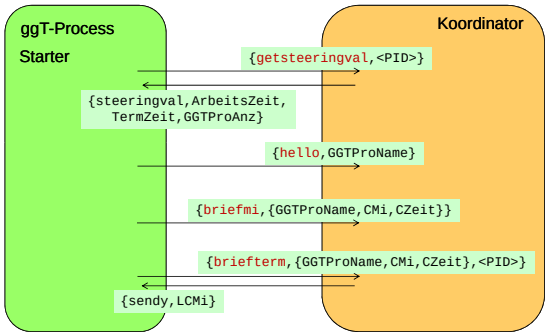




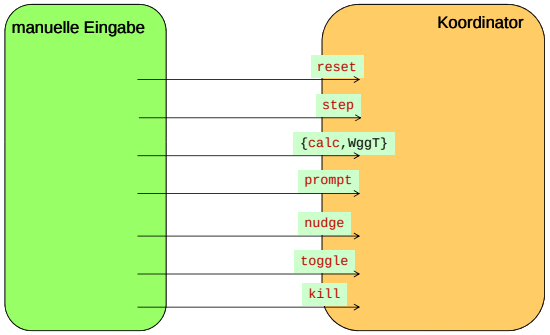
Verteilter Algorithmus: Namensdienst



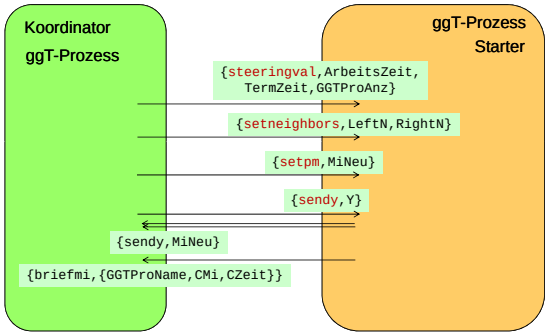
Verteilter Algorithmus: Koordinator



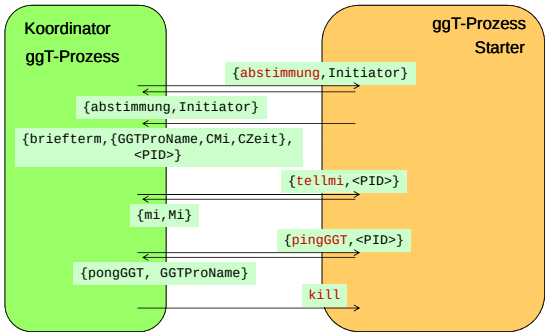
Verteilter Algorithmus: Koordinator



Verteilter Algorithmus: ggT-Prozess



Verteilter Algorithmus: ggT-Prozess



Verteilte Systeme

Aufgabe 3

Beispielcode: Corba-Server

- Name Binding (Server):

```
ServerImpl server = newServerImpl();
orb.connect(server);  //POA Aktivierung
org.omg.CORBA.Object nameservice =
     orb.resolve_initial_references("NameService");
NamingContext namingcontext =
     NamingContextHelper.narrow(nameservice);
NameComponent name = newNameComponent("Datum","");
NameComponent path[] = {name};
namingcontext.rebind(path,server);
```

Anforderung des initialen Namensraum

Name

Art

narrow: findet zu Objektreferenz die Klasse

Helper: vom Schnitt-stellencompiler erzeugt

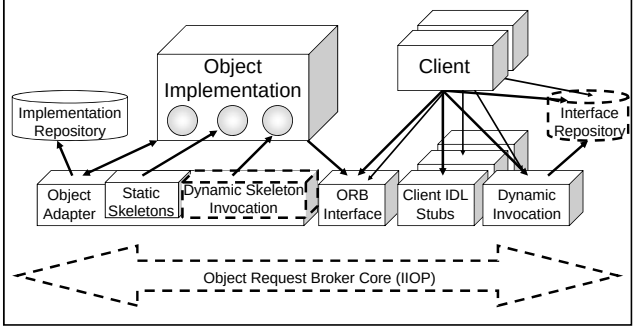
rebind: stellt das Serverobjekt der Middleware vor.

Beispielcode: CORBA-Client

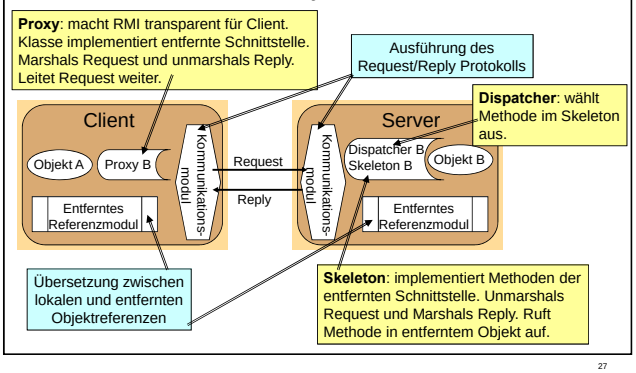
- Name Resolution (Client):

```
Server server;
org.omg.CORBA.Objectnameservice =
  orb.resolve_initial_references("NameService");
NamingContext namingcontext =
  NamingContextHelper.narrow(nameservice);
NameComponent name = newNameComponent("Datum","");
NameComponent path[] = {name};
server = ServerHelper.narrow(namingcontext.resolve(path));
```

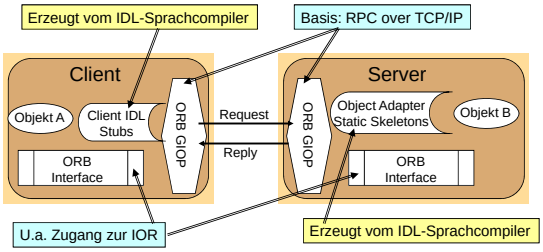

Struktur eines CORBA-2.*-ORB's I



Rolle von Proxy und Skeleton



Rolle von Proxy und Skeleton



Beispiel: CORBA-IOR

module ggt{
interface unit{ … }; };

Inhalt der Schnittstellendefinition

Abgelegte Referenz im Namensdienst

IOR:010000001100000049444c3a6767742f756e69743a312e3000000000200000000000003c0000000101000019000000 6c616232322e6370742e6861772d68616d627572672e646500000b70e1400000002f31313930312f313231333237363837322f5f3001000000240000000100000001000000014000000010000000100010000000000090101000000000

Inhalt der IOR

Repo Id: IDL:ggt/unit:1.0

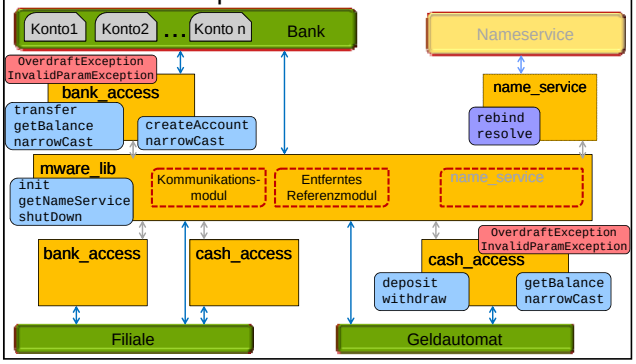
IIOP Profile
Version: 1.0
Address: lab22.cpt.haw-hamburg.de:3767
Key: 2f 31 31 39 30 31 2f 31 32 31 33 32 37 36 38 37 /11901/121327687
32 2f 5f 30 2/_0

Multiple Components:
Codesets Tag
Native char CS: ISO 8859-1:1987; Latin Alphabet No. 1
Native wchar CS: ISO/IEC 10646-1:1993; UTF-16, UCS Transformation Format 16-bit form

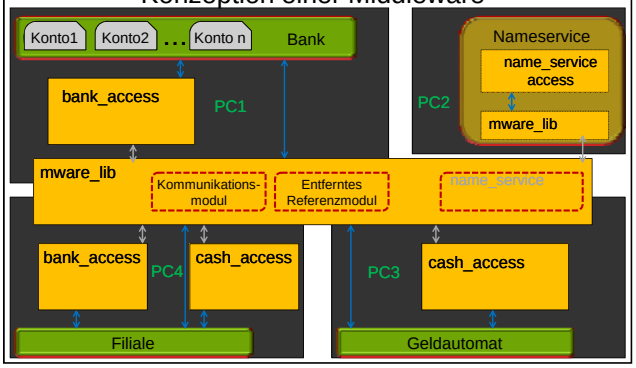
Namen und Binden

- Zuordnung Name ---> Adresse
- Bindezeitpunkt:
  - beim Übersetzen (**statisches Binden**)
    (Call-by-Value)
    z.B. bei Programmiersprachen
  - beim Starten (**"halb"-dynamisches Binden**)
    z.B. moderne Binder (SunOS), nach dem Start in der Regel nicht änderbar
  - beim Zugriff (**dynamisches Binden**)
    (Call-by-Refernce/Call-by-Name)
    in verteilten Systemen angebracht:
    - Neue Dienste
    - Verlagerung existierender Dienste

Konzeption einer Middleware



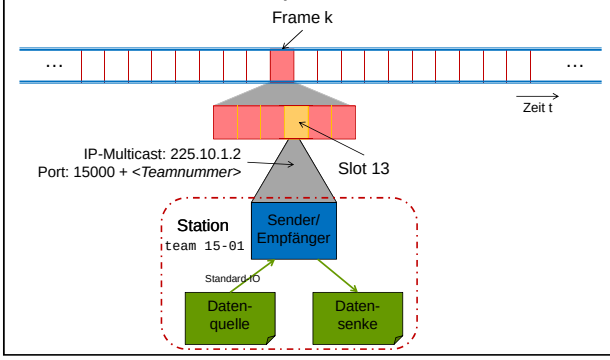
Konzeption einer Middleware



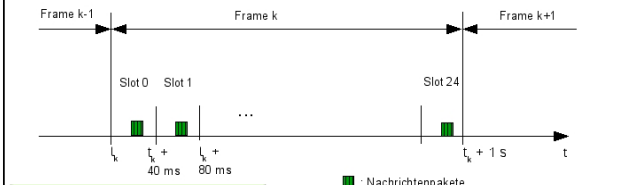
Verteilte Systeme

Aufgabe 4

Zeitmultiplexverfahren



Zeitmultiplexverfahren



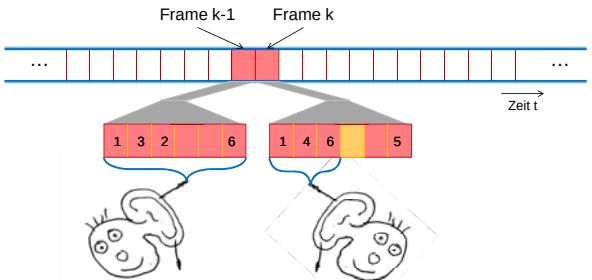
Eine **Kollision** besteht, wenn die Sendezeitpunkte von zwei Nachrichten innerhalb eines Slotzeitintervalls liegen!

: Nachrichtenpakete

Stationsklasse	Nutzdaten	reservierte Slotnummer	Sendezeitpunkt ms, 8-Byte Integer, Big Endian
Byte 0	Byte 1..24	Byte 25	Byte 26..33

A | -***-team 4711-***- | **4** | **77394825**

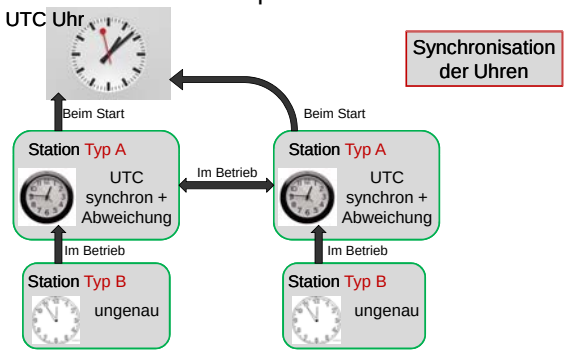
Zeitmultiplexverfahren



Reservierte Slots in Frame k:
1,2,3,6
Per **Zufall** ausgewählt für eigenes Senden: Slot 4

Bisher reservierte Slots in Frame k+1:
1,4,6
Per **Zufall** ausgewählt für eigenes Senden: Slot 3

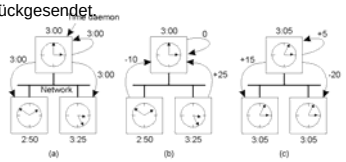
Zeitmultiplexverfahren



Relative (interne) Uhrensynchronisation

- Falls eine einheitliche Zeit benötigt wird (**ohne UTC-Empfänger**)
- **Berkeley (UNIX) Algorithmus** (1989): Ein Rechner ist der *Koordinator*:
  - Zeit-Server (= Koordinator) fragt periodisch alle Rechner nach ihrer Uhrzeit
  - Aus den erhaltenden Antworten werden die lokalen Zeiten durch Schätzung der Nachrichtenlaufzeiten ermittelt. Antworten, die zu lange auf sich warten lassen, werden ignoriert.
  - Aus den geschätzten lokalen Zeiten wird das arithmetische Mittel gebildet.
  - Die jeweiligen Abweichungen vom Mittel werden als neuer aktueller Uhrenwert den Rechnern zurückgesendet.

Hier: „jeder ist Koordinator“ und die Uhrzeit wird aus den Nachrichten extrahiert.



Tipp

- **Empfängereinheit**: hört zu und entscheidet, ob es eine Kollision gab oder nicht; reicht die Nachrichten (wenn es keine Kollision gab) an die Auswertung weiter.
- **Uhrensynchronisation**: wertet die Zeitstempel der Typ A Stationen (abzüglich/zuzüglich der Slotzeit im Frame) per arithmetischem Mittel aus und bestimmt die eigene Abweichung (eigene Uhr: Sytemzeit +/- Abweichung).
- **Slotreservierung**: ermittelt die reservierten Slots im nächsten Frame und stellt einen (zufällig gewählten) freien Slot zu jedem Zeitpunkt (z.B. nach Slot 2 oder nach Slot 24) zur Verfügung.
- **Sendeeinheit**: besorgt sich einen für das nächste Frame zu reservierenden Slot von der Slotreservierung; prüft, ob die aktuelle Zeit (nach eigener Uhr!) mit dem vorgesehenen Slot bzw. der damit verbundenen Sendezeit übereinstimmt und fügt ggf. der Nachricht die Sendezeit an und sendet die Nachricht. Hat man die Sendezeit verpasst, muss am Ende des Frames ein Slot zum Wiedereinstieg gewählt werden.
- **Nachrichtengenerierung**: bereitet die zu sendende Nachricht vor (bis auf die Sendezeit); bestimmt auf Grund des reservierten (oder ggf. bei Neueinstieg gewähltem) Slot die Sendezeit und sorgt dafür, dass die Sendeeinheit rechtzeitig aktiviert wird.