

Synchronisation in verteilten Systemen

Die Synchronisationsmethoden bei Einprozessorsystemen (z.B. Semaphore oder Monitore) setzen gemeinsame Speicher voraus. Also sind sie für verteilte Systeme so nicht anwendbar.

Dieser Teil behandelt die Synchronisation in verteilten Systemen. Hier speziell die Problematik der Synchronisation von Uhren, damit verteilte Applikationen mit derselben Zeit arbeiten.

1. Zeit in verteilten Systemen.....	2
1.1. Synchronisation physikalischer Uhren.....	2
1.1.1. Koordinierte Universalzeit.....	3
1.1.2. Generelle Voraussetzungen.....	4
1.1.3. Verfahren von Cristian.....	6
1.1.4. Berkley Algorithmus.....	7
1.1.5. Algorithmen ohne Zentralen Server mit Durchschnittsbildung.....	8
1.1.6. Das Network Time Protocol.....	11
1.2. Synchronisation logischer Uhren.....	14

1. Zeit in verteilten Systemen

Kausale Zusammenhänge von Ereignissen sind oft durch die zeitliche Reihenfolge gegeben. Das make-Werkzeug erkennt z.B. ob eine Quelldatei neu übersetzt werden muss daran, ob es eine abhängige Datei gibt, die jünger ist.

Allgemein kann man sagen, dass es oft relevant ist, ein Ereignis mit dem Zeitpunkt, an dem es stattgefunden hat, zu korrelieren.

Wenn die interne Konsistenz der Uhr ausreicht (wie bei make) und die Differenz zur absoluten Zeit unerheblich ist, so spricht man von **logischer Uhr**.

Wenn zusätzlich zur logischen Uhr dazukommt, dass die Zeit nur um einen bestimmten Betrag von der absoluten Zeit abweichen darf, spricht man von physischen Uhren.

1.1.Synchronisation physikalischer Uhren

Die rechnerinterne physikalische Uhr wird von einem Taktgeber abgetaktet und zählt ein Uhrenregister hoch. Eine Software berechnet daraus Zeit-Datum. Dies kann als Zeitstempel für ein Ereignis verwendet werden.

Dabei treten folgende Probleme auf:

- Uhren driften ca. 1 Sekunde in 10 Tagen, d.h. Uhrendriftrate von ca. 10^6 .
- Die beteiligten Rechner des verteilten Systems starten nicht zum gleichen Zeitpunkt.

Deshalb braucht man eine allgemein gültige, globale Zeit.

1.1.1. Koordinierte Universalzeit

Die Koordinierte Universalzeit (Universal Time Coordinated, kurz UTC) ist beschrieben durch:

- Die Dauer einer Sekunde ist astronomisch $1/86.400$ eines Solartages.
- Die Erdrotation verlangsamt sich jedoch stetig.
- Deshalb ist die Zeitangabe auf astronomischer Basis ungenau.
- 1958 wurde die Internationale Atomzeit (kurz TAI) eingeführt:
1 Sekunde ist die Dauer von 9.192.631.770 Übergangsperioden in Cäsium-133 Atomen.
- Die TAI ist die Anzahl der Takte dieser Cäsium-Uhr seit Mitternacht 1. Januar 1958 geteilt durch 9192631770.
- Ein Tag nach der TAI ist drei Millisekunden kürzer als ein Solartag, weshalb jeweils dann eine Schaltsekunde eingefügt wird, wenn die TAI und die Solarzeit um mindestens 800 Millisekunden auseinander liegen.
- 1967 wurde die Koordinierte Universalzeit (UTC) definiert: sie berücksichtigt die Schaltsekunden.
- UTC wird per Kurzwelle bzw. per Satellit ausgestrahlt.
- Die Übertragung mittels GEOS (Geostationary Environmental Operational Satellites) hat eine Genauigkeit von ca. 0,1 Millisekunden,
- über das GPS (Global Positioning System) eine Genauigkeit von ca. 1 Millisekunde.

Damit ist eine denkbare Lösung des Zeitproblems: **jeden Rechner** wird mit **Satellitenempfänger** für UTC ausgerüstet. Die ist aus Kostengründen nicht sinnvoll.

Nun werden Verfahren zu dieser Synchronisation beschrieben.

1.1.2. Generelle Voraussetzungen

Ein Rechner mit Satellitenempfänger ausgestattet und alle anderen Rechner werden auf die externe physikalische Zeit synchronisiert.

Dazu muss ein Rechner den Zeitdrift kompensieren:

- Läuft die interne Uhr zu langsam, so kann sie auf die UTC vorgestellt werden.
- Ist die Uhr zu schnell, ist ein **Zurückstellen problematisch**, da die Zeit auch für den Rechner ein Kontinuum sein sollte (ansonsten wird z.B. ein Backup doppelt durchgeführt). Eine Lösung dafür ist, die Uhr muss stetig verlangsamt werden.

Jeder **Rechner** verfügt über eine **Stoppuhr**, die H mal in der Sekunde eine **Unterbrechung** auslöst. Dabei wird von einem Interrupthandler eine Softwareuhr inkrementiert. Diese Zahl ist die Anzahl der Unterbrechungen ab einem festen Bezugspunkt. Diese Zahl wird C genannt.

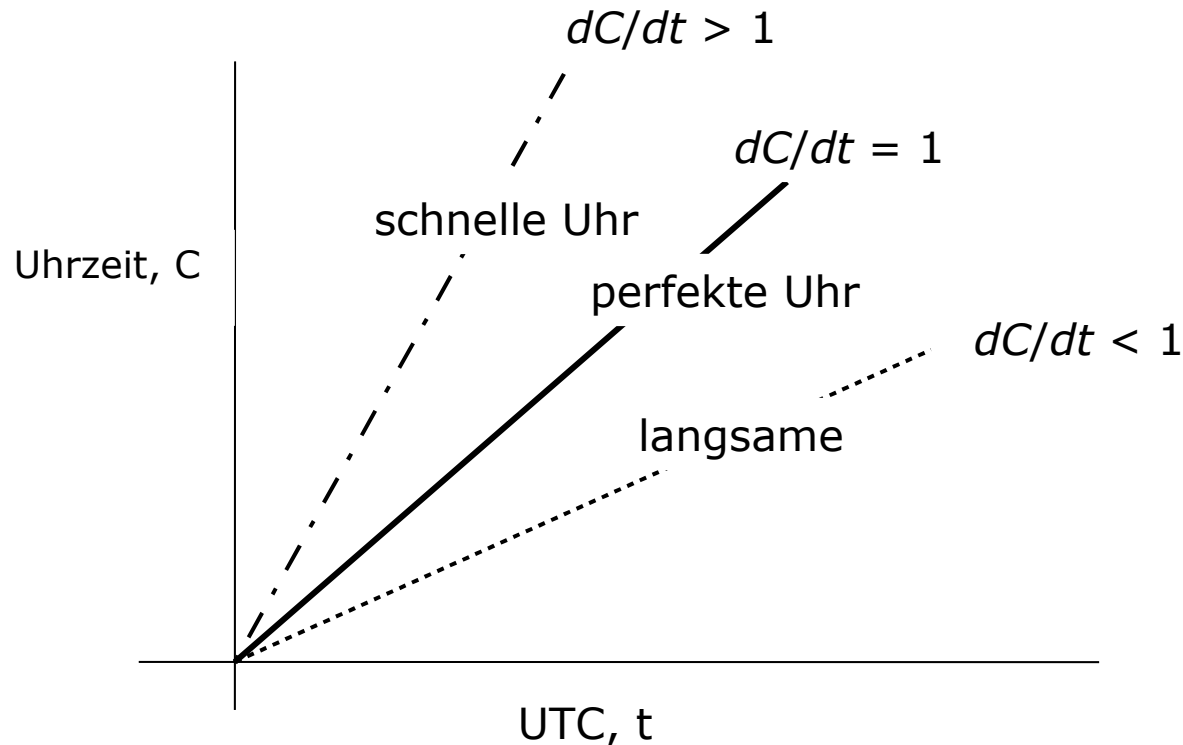
$C_p(t)$ sei die Uhr des Rechners p zur UTC t .

Ein Rechner läuft richtig, wenn gilt: $C_p(t) = t$. Anders ausgedrückt sollte $dC/dt = 1$ sein.

Die **Güte** einer Uhr wird bestimmt durch eine Konstante ρ (rho) mit: $1 - \rho \leq dC/dt \leq 1 + \rho$

Die Konstante ρ wird vom Hersteller angegeben und als **maximale Abweichung** bezeichnet.

Das folgende Bild zeigt den Bereich, in dem Uhren liegen dürfen:



Falls zwei Uhren im **schlimmsten Fall entgegengesetzt** driften, können sie Δt nach ihrer Synchronisation maximal um $2\rho\Delta t$ auseinander liegen.

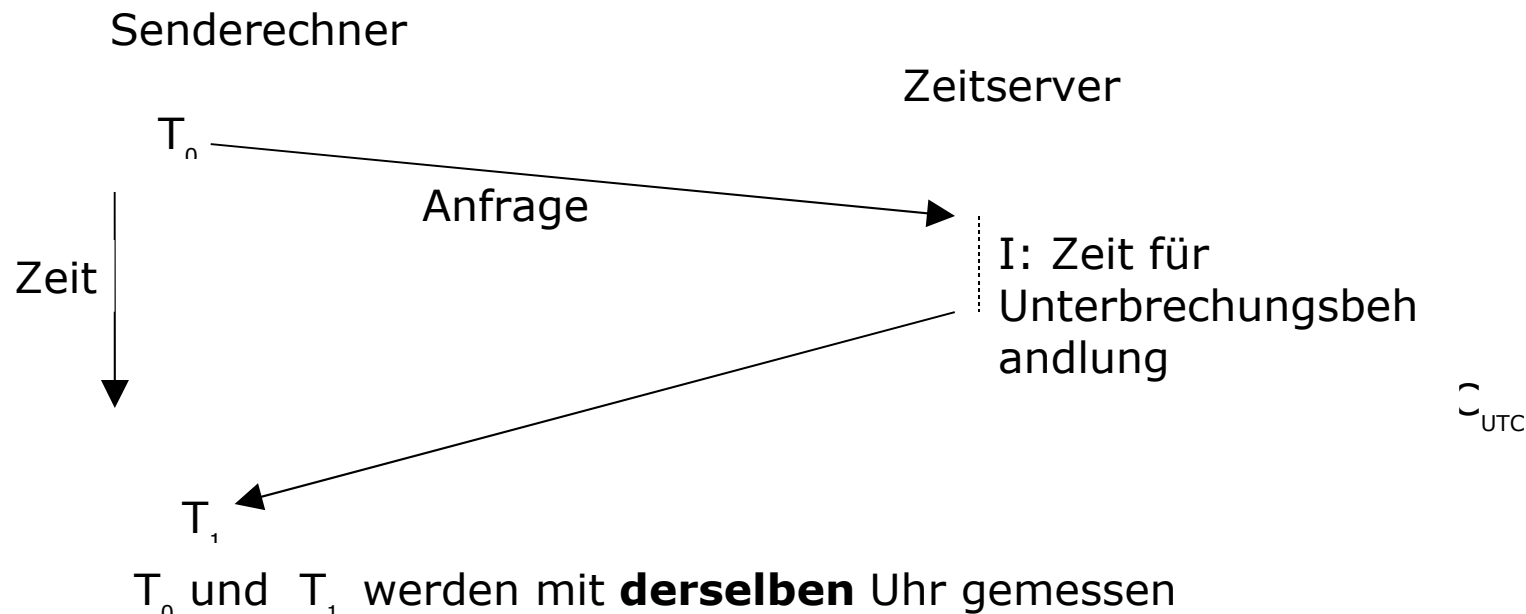
Wenn also sichergestellt sein muss, dass **zwei Uhren** nicht mehr als δ (delta) auseinander laufen, so müssen sie **mindestens** alle $\delta/2\rho$ **Sekunden synchronisiert** werden.

Die im Folgenden betrachteten Verfahren unterscheiden sich darin, wie präzise diese Synchronisation durchgeführt wird.

1.1.3. Verfahren von Christian

Verwendet wird ein Zeitserver, der mit Satellitenempfänger für UTC ausgestattet ist. Alle Rechner synchronisieren sich mit dem Zeitserver durch folgenden Algorithmus:

- regelmäßig ($\geq \delta/2\rho$ Sekunden) wird Nachricht an Zeitserver gesendet.
- der Zeitserver antwortet mit C_{UTC} .
- Die Abfolge ist wie folgt:



- Der Sender setzt in erster Näherung seine Uhr auf C_{UTC}
- Ist $C_{UTC} >$ eigene Zeit, dann wird C_{UTC} übernommen.
- Ist $C_{UTC} <$ eigene Zeit, dann wird die eigene Uhr stetig verlangsamt: dies wird durch die

Behandlungsroutine vollzogen, die weniger als den normale Wert inkrementiert bis die Angleichung vollzogen ist.

- Jetzt wird die eigene Uhr um die Hälfte der Zeit angepasst, die die Nachricht zum und vom Zeitserver gebraucht hat (Roundtrip-Delay): $(T_1 - T_0)/2$.

Das größte **Problem** dieses Verfahrens ist die Erfordernis eines **zentralen Zeitserver**s. Bei Ausfall des Servers kann keine Synchronisation der Zeit stattfinden.

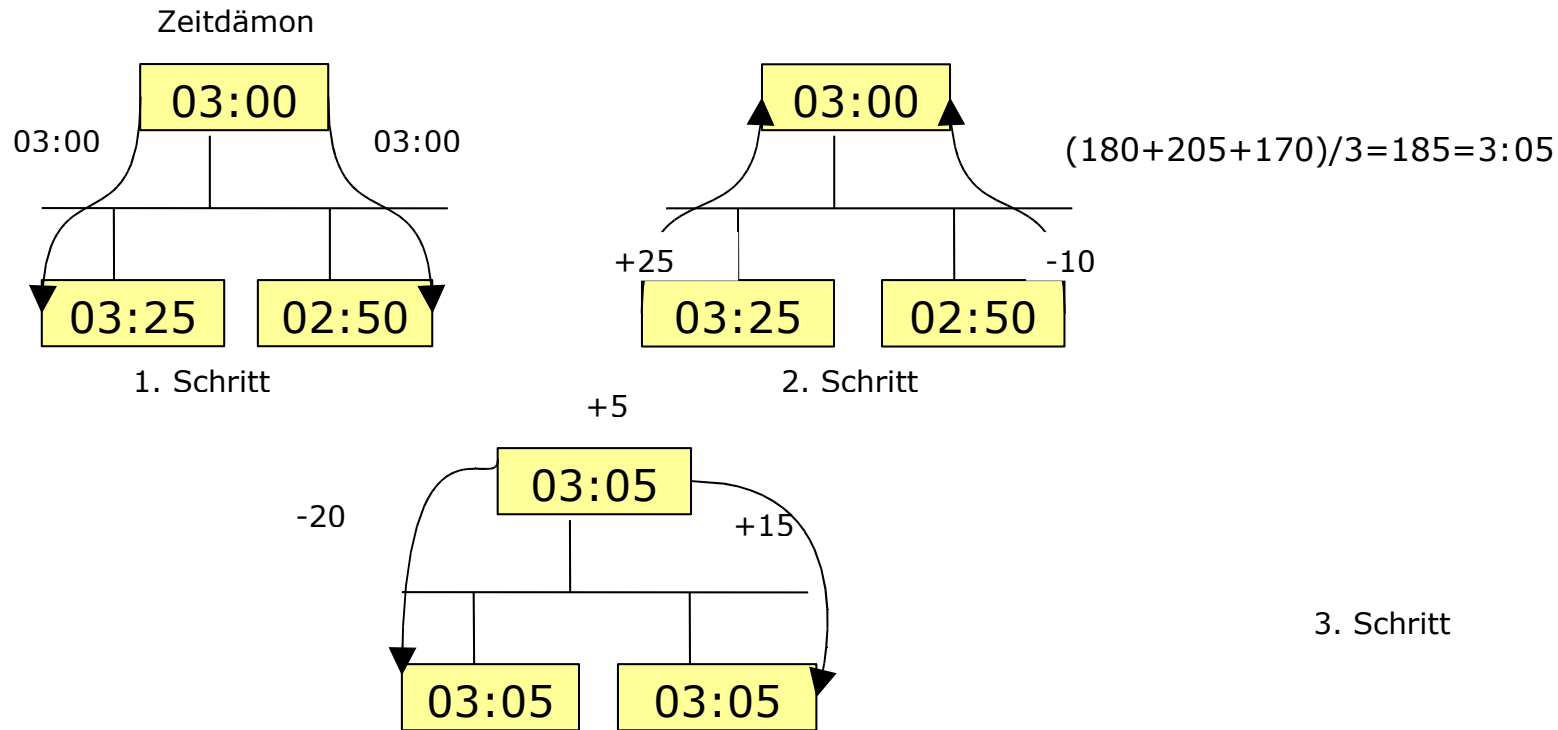
1.1.4. Berkeley Algorithmus

Hier wird ein aktiver **Zeitserver** eingesetzt. Dieser Dämon **fragt** periodisch **alle Rechner** nach ihrer Zeit. Aus allen Antworten und seiner eigenen Zeit wird die durchschnittliche Zeit ermittelt. Diese Durchschnittszeit wird allen Rechnern mitgeteilt.

Alle Rechner (auch der Zeitdämon) stellen dann ihre Uhr: sie erhöhen die Zeit oder verlangsamen die Uhr bis eine Angleichung erfolgt ist.

Diese Methode bedarf keiner Funkuhr.

Das Verfahren kann wie folgt veranschaulicht werden:



Das größte **Problem** dieses Verfahrens ist die Erfordernis eines **zentralen Zeitserverns**. Bei Ausfall des Servers kann keine Synchronisation der Zeit stattfinden.

1.1.5.Algorithmen ohne Zentralen Server mit Durchschnittsbildung

Die Nachteile der o.a. Verfahren (zentraler Server) können vermieden werden, wenn man dezentrale Methoden einsetzt.

Eine Methode ist, die Zeit in **Synchronisationsintervalle** mit **fester Länge** einzuteilen. Das i -te Intervall beginnt um T_0+iR und dauert bis $T_0+(i+1)R$ wobei T_0 ein festgelegter Startzeitpunkt und R ein Systemparameter ist.

Am **Anfang** eines jeden **Intervalls** sendet **jeder** Rechner seine lokale Zeit an alle anderen Rechner.

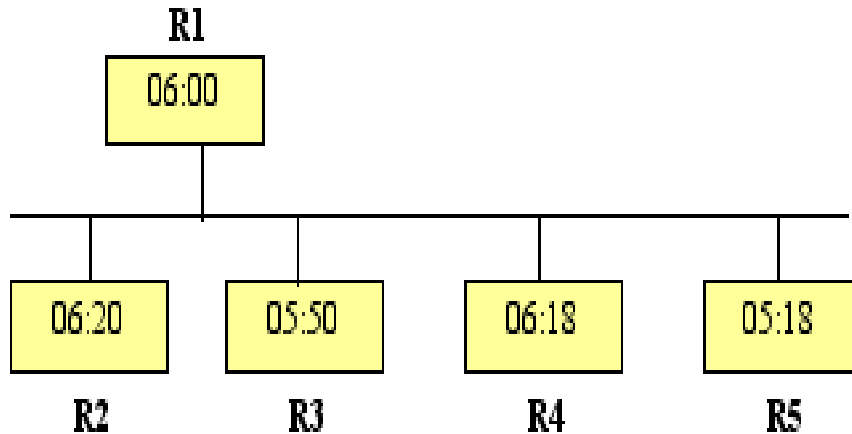
Da die Uhren der Rechner nicht exakt synchron laufen, gibt es i.A. keine Kollisionen durch das „gleichzeitige“ Senden.

Nachdem ein Rechner seine Zeit gesendet hat, wartet er innerhalb eines Zeitintervalls auf den Empfang von Zeitnachrichten der anderen Rechner.

Dann stellt er seine Uhr auf den Durchschnittswert der empfangenen Zeiten und seiner eigenen Zeit ein, wobei die kleinste und größte Zeit weggelassen wird (defekte Uhren bleiben unberücksichtigt, auch u.U. seine eigene).

Hörsaalübung

Gegeben seien 5 Rechner mit jeweils folgenden Uhrzeinstellungen:



a) Gehen Sie davon aus, dass Rechner R1 der Zeitserver ist. Welche Zeit hat jeder Rechner gemäß des Berkeley Algorithmus nachdem die Uhren einmal synchronisiert wurden.

R1:

R2:

R3:

R4:

R5:

b) Welche Uhrzeit stellt sich gemäß des Algorithmus „ohne Zentralen Server mit Durchschnittsbildung“ ein, wenn einmal synchronisiert wurde.

R1:

R2:

R3:

R4:

R5:

1.1.6. Das Network Time Protocol

Das freie Software-Paket **NTP** (Network Time Protocol) ist eine Implementierung des gleichnamigen TCP/IP-Protokolls zur Zeitsynchronisierung von Geräten im Netzwerk. NTP wurde in den 1980er Jahren von [Dave L. Mills](#) in den USA entwickelt, der versuchte, die Systemzeiten mehrerer Rechner im Netzwerk mit möglichst hoher Genauigkeit zu synchronisieren.

Das Protokoll selbst unterstützt eine **Zeitgenauigkeit** bis in den **Nanosekundenbereich** hinein. Die tatsächlich erreichbare Genauigkeit hängt jedoch in großem Maße von den verwendeten Betriebssystemen und der Qualität der Netzwerkverbindungen ab.

Hierarchische Zeitsynchronisierung

Ein NTP-Daemon kann nicht nur die Zeit seines eigenen Rechners synchronisieren, sondern kann auch Client, Server, oder Peer für andere NTP-Daemons im Netzwerk sein:

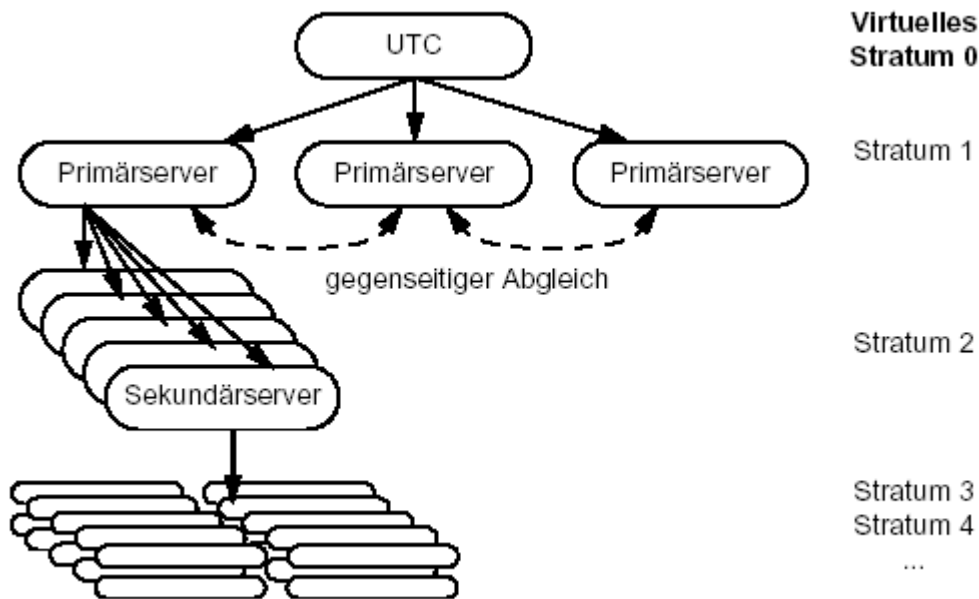
- Als **Client** holt er sich die Referenzzeit von einem oder mehreren Servern.
- Als **Server** stellt er seine eigene Zeit anderen NTP-Clients im Netzwerk zur Verfügung.
- Als **Peer** vergleicht er seine eigene Zeit mit mehreren anderen NTP-Peers, die sich schließlich auf eine gemeinsame Zeit einigen, nach der sich alle richten.

Mit Hilfe dieser Möglichkeiten kann auf einfache Weise eine hierarchische Zeitverteilung in einem Netzwerk eingerichtet werden. Die einzelnen Hierarchie-Ebenen werden **Stratum** genannt. Eine kleinerer *Stratum*-Wert bedeutet dabei eine höhere Ebene in der Hierarchie-Struktur

Jeder NTP-Daemon hat einen Stratum-Wert, der um eins höher ist als der Stratum-Wert seiner Referenzzeitquelle. Auf der obersten Hierarchie-Ebene befindet sich der NTP-Daemon mit der genauesten Zeit und dem kleinsten *Stratum*-Wert, der normalerweise als Referenzzeit eine Funkuhr verwendet.

Funkuhren haben normalerweise den Stratum-Wert 0. Daher wird der NTP-Daemon, der die Funkuhr direkt als Referenz benutzt, zum **Stratum-1-Timeserver**, der die höchste Priorität in der NTP-Hierarchie hat. In der Praxis wird man in größeren Netzwerken einen oder mehrere Stratum-1-Timeserver einrichten, die ihre Zeit den Servern in den einzelnen Abteilungen bereitstellen. Diese werden dadurch zu Stratum-2-Servern, auf die sich weitere Geräte und Arbeitsplatzrechner im Netzwerk synchronisieren können.

Dadurch entsteht eine baumartige Struktur:



Redundante Zeitsynchronisierung

Jeder **NTP-Daemon** kann so konfiguriert werden, dass er **mehrere unabhängige Referenzzeitquellen** verwenden kann. Von allen möglichen Zeitquellen sucht er sich dann die beste aus. Kriterien für die Auswahl sind Erreichbarkeit, der beste *Stratum*-Wert, sowie die Größenordnung der Antwortzeiten bei der Abfrage (Delay) und die Schwankungsbreite der Antwortzeiten (Jitter). Wenn die ausgewählte Referenzzeitquelle nicht mehr erreichbar sein sollte, wählt sich der NTP-Daemon die nächst beste Referenzzeitquelle aus. Wenn diese einen anderen *Stratum*-Wert besitzt als die vorherige Quelle, dann ändert sich beim Wechsel der Zeitquelle auch der *Stratum*-Wert des NTP-Daemons selbst.

Der NTP-Dämon wird gestartet durch:

```
# ntpd
```

wobei die Konfigurationsdatei

```
/etc/ntp.conf
```

die Statum-Rechner spezifiziert.

Einen konfigurierten Server (ntp.conf) kann man abfragen, ohne dass die lokale Uhr verändert wird, durch:

```
# ntpdate -q
```

1.2.Synchronisation logischer Uhren

Für die **Feststellung** der **zeitlichen Reihenfolge** von Ereignissen ist oft die **Ordnung** der **Ereignisse ausreichend**, d.h. die Uhren der Rechner können ungleich laufen, sowohl bzgl. der Zeit als auch der Geschwindigkeit.

Zur Synchronisation von logischen Uhren wird oft der **Lamport Algorithmus** verwendet.

Dazu wird die **Relation „happened before“** verwendet: der Ausdruck $a \rightarrow b$ wird als „**a** tritt vor **b** ein“ und bedeutet, dass alle Prozesse darin übereinstimmen, dass zuerst das Ereignis **a** eintritt und danach das Ereignis **b**.

Definition („happened before“, „ \rightarrow “)

- Sind a und b Ereignisse im **gleichen** Prozess und a ereignet sich vor b , dann gilt: $a \rightarrow b$ (a happened before b).
- Ist a ein Sendeereignis in **einem Prozess** und b das zugehörige Empfangsereignis in einem **anderen Prozess**, dann gilt: $a \rightarrow b$.

Damit entspricht die Relation \rightarrow der kausalen Ordnung der Ereignisse.

Die Relation \rightarrow besitzt folgende Eigenschaften:

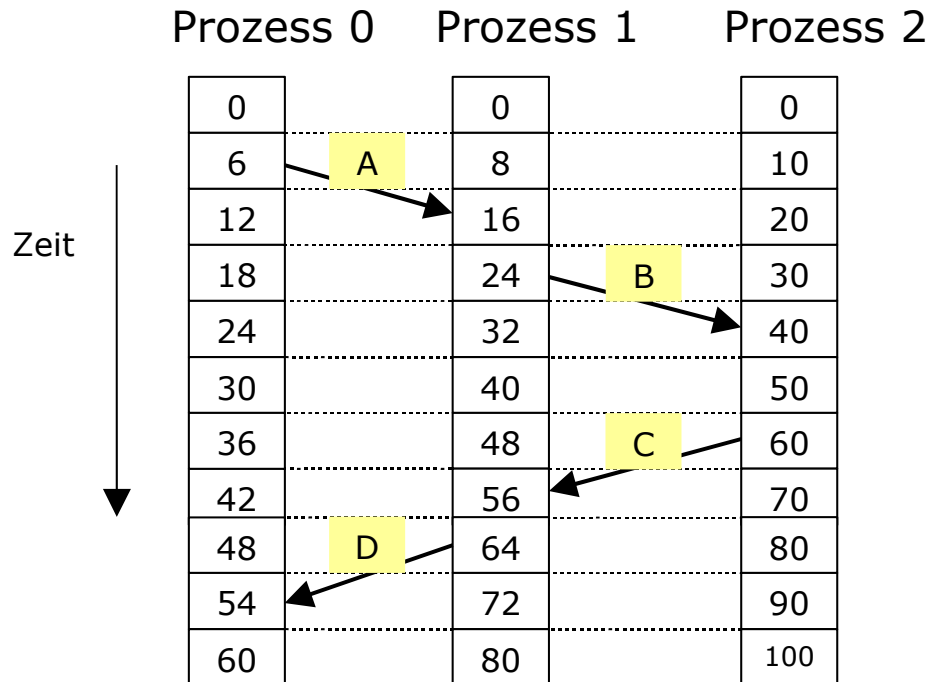
- **Transitivität**, d.h. wenn $a \rightarrow b$ und $b \rightarrow c$ gilt, dann gilt auch $a \rightarrow c$
- unabhängige Ereignisse (es gilt weder $a \rightarrow b$ noch $b \rightarrow a$) heißen nebenläufig (engl.: concurrent) und werden als $a || b$ geschrieben
- Happened-before definiert **eine partielle Ordnung** auf den Ereignissen

Benötigt wird eine Möglichkeit der Zeitmessung, so dass dem Ereignis a eine Zeit $C(a)$ zugewiesen werden kann, mit der alle Prozesse einverstanden sind.

Die Idee der logischen Uhr ist:

- **Jeder Prozess** p führt seine **logische Uhr** C_p als eine Variable vom Typ natürliche Zahl
- Bei **jedem Ereignis** wird diese **Variable verändert**, um die Ereignisse mit Zeitstempeln zu versehen.
- $C_p(a)$ bezeichnet den Zeitstempel des Ereignisses a im Prozess p .
- Die logische Uhr soll der happened-before Relation genügen.

Betrachten wir ein Beispiel mit drei Prozessen, die alle unterschiedliche Uhren haben:



Jede **Nachricht** soll den **Sendezeitpunkt enthalten**. Die Dauer der Nachrichtenübermittlung hängt von der Uhr ab, die man verwendet. Zum Zeitpunkt 6 sendet Prozess 0 eine Nachricht A. Aus Sicht von Prozess 1 hat die Nachricht 10 Zeiteinheiten gebraucht ($16-6$); ein plausibler Wert aus Sicht Prozess 1.

Die Nachricht C von Prozess 2 an Prozess 1 wird zum Zeitpunkt 60 gesendet und kommt zum Zeitpunkt 56 bei Prozess 1 an ($56-60=-4$); dies ist kein plausibler Wert.

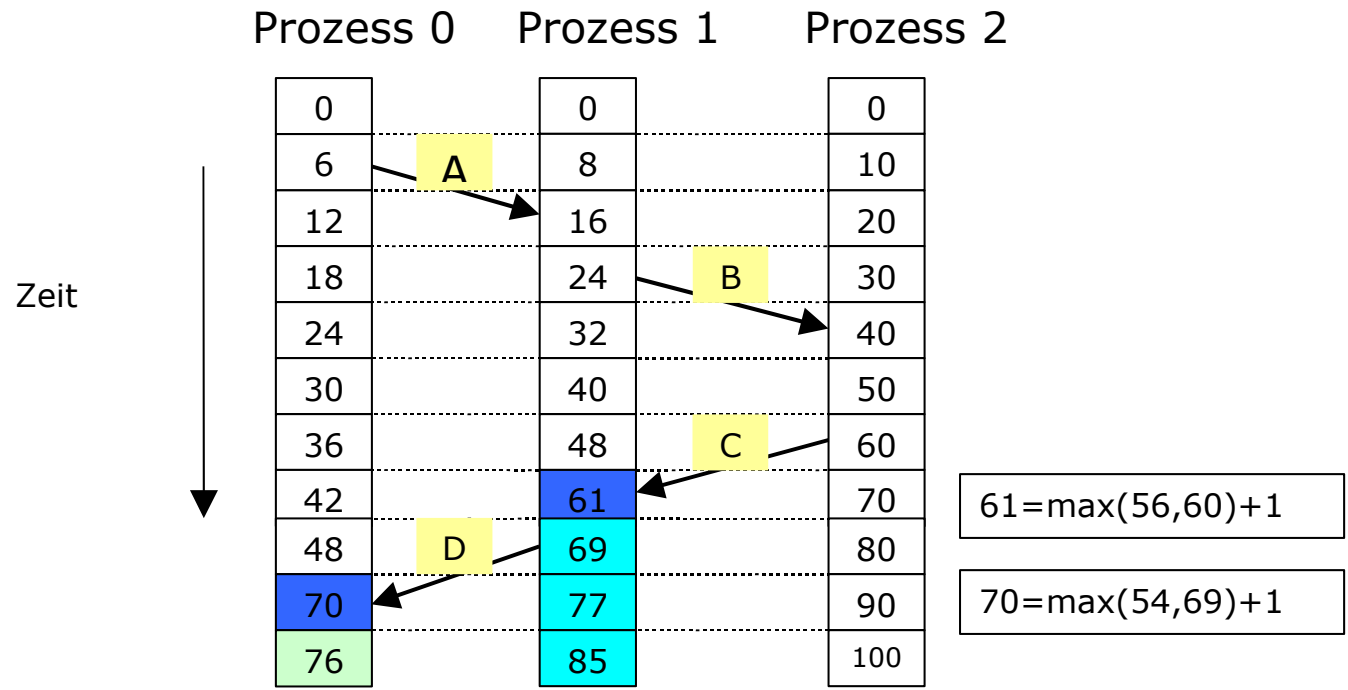
Der **Algorithmus von Lamport** verhindert diese Situation:

Verfahren aus Sicht Prozess p

- lokales Ereignis a :
 $C_p(a) := C_p(a) + 1;$
- Sendeereignis a :
 $C_p(a) := C_p(a) + 1;$
send (message, $C_p(a)$);
- Empfangsereignis a :
receive (message, C_Q);
 $C_p(a) := \max(C_p(a), C_Q) + 1;$

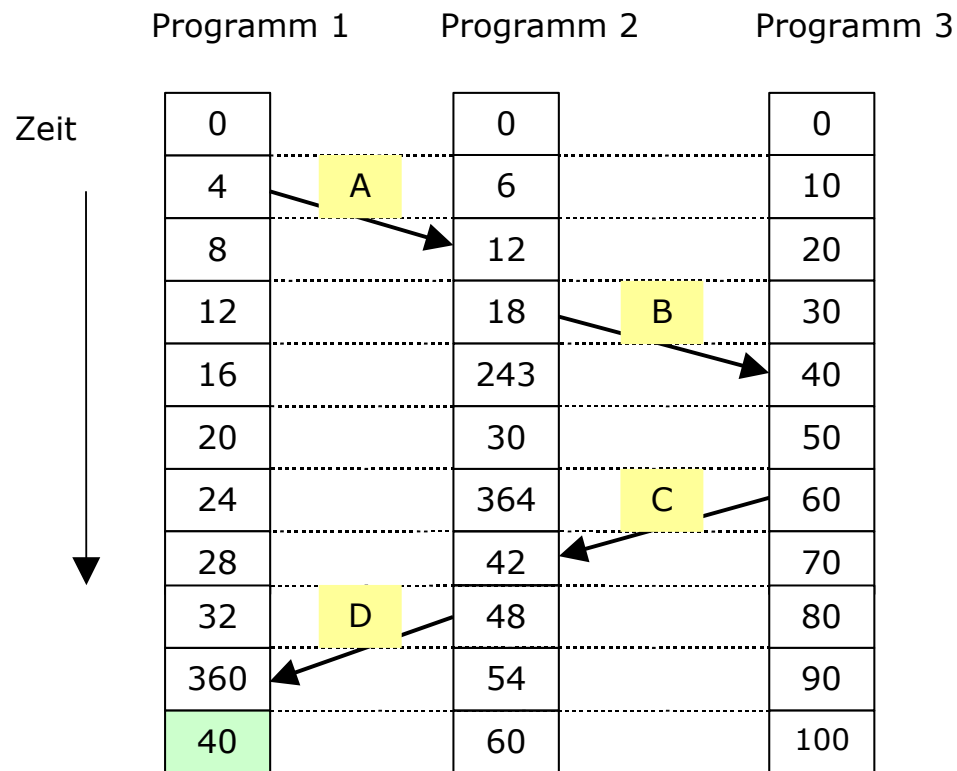
D.h. bei jedem Ereignis wird die Zeit erhöht; die Sendezeit wird mit übertragen und bei Nachrichtenempfang wird das Maximum von eigener Zeit und übertragener Zeit verwendet.

Somit stellt sich das o.a. Beispiel mit Lamport Algorithmus wie folgt dar:



Hörsaalübung

Gegeben seien 3 Programme, die jeweils eine eigene logische Uhr gemäss des Algorithmus von Lamport implementiert haben. Die Uhren der Programme „ticken“ unterschiedlich schnell, Programm 1 im 4er Taktung, Programm 2 in 6er Taktung und Programm 3 in 10er Taktung. Die Programme tauschen Nachrichten (A-D) gemäss der u.a. Abbildung aus.



Zeigen Sie in der o.a. Abbildung, wie sich die Zeit der logischen Uhr pro Programm verändert.