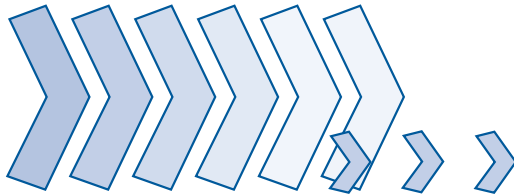
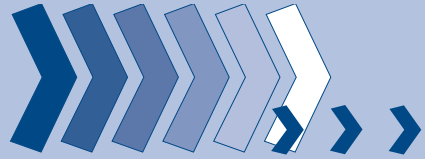




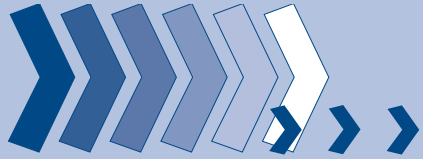
# Uhrensynchronisation

Dipl.-Inf. J. Richling  
Wintersemester 2003/2004

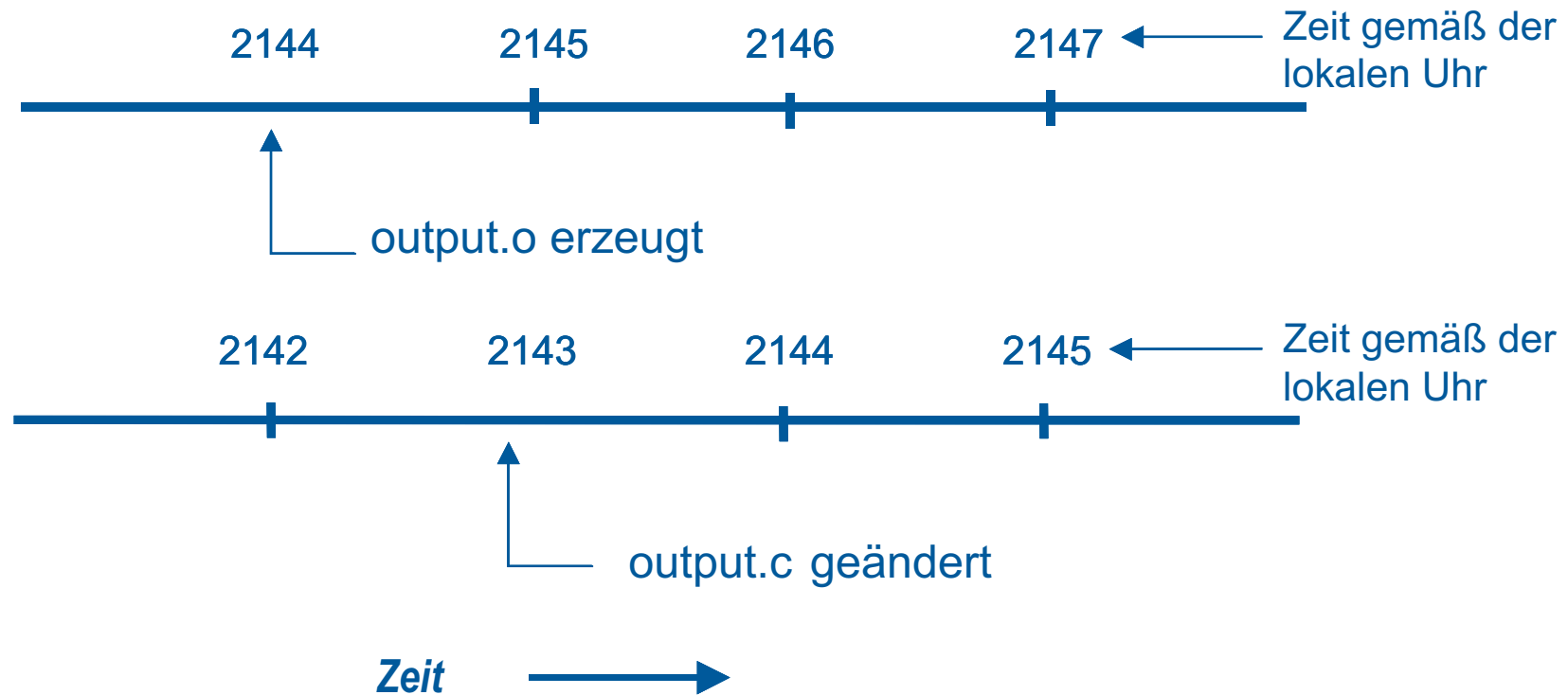




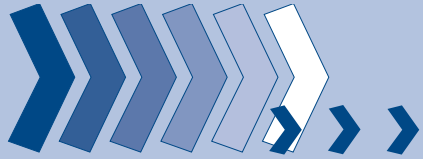
- Zeit kann in Anwendungen eine große Rolle spielen, insbesondere bei Echtzeitsystemen
- Häufig wichtiger noch als korrekte Zeit: konsistente Zeit
- Einprozessorsystem:  
Eine zentrale Uhr (shared memory bei mehreren Prozessen)
- Verteiltes System
  - meist kein gemeinsamer Speicher vorhanden
  - Problem mit der Ereignisreihenfolge (verschiedene Ereignishorizonte)
  - Verwendung von verteilten Algorithmen



## Beispiel: Verteiltes make

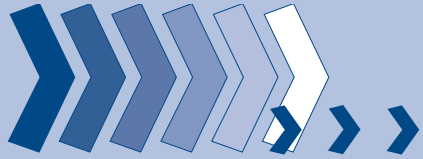


- Make ruft den Compiler trotz Änderung nicht auf.



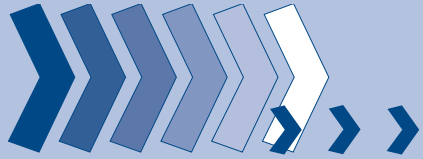
## *Zeitmessung und Konsequenzen*

- Lokale Zeitmessung erfolgt durch einen Timer (Zähler = Speicheradresse + Ticker)
- Problem: Unterschiedliche Tickfrequenzen führen zu Abweichungen sowohl von der realen Zeit, als auch untereinander
- Ursachen
  - Ungenaue Quarze
  - Verbotene Interrupts
  - Nichtkonsistente Umweltbedingungen
- Im Einprozessorsystem meist unkritisch, da Ereignisreihenfolge konsistent bleibt
- Problem im verteilten System  $\Rightarrow$  Ziel: ein einziger, eindeutiger Zustand

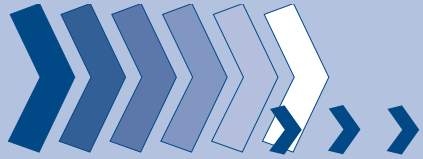


## *Kurze Geschichte der Zeit(messung)*

- Ab 17. Jahrhundert: astronomische Zeitmessung
  - Sonnentag: Intervall zwischen zwei Sonnendurchgängen
  - Sonnensekunde:  $\frac{1}{86400}$  eines Sonnentages
- Sonnentage sind nicht gleich lang  $\Rightarrow$  Mittelwertbildung über eine größere Zahl von Tagen
- 1948 Atomuhr
  - 1 Sekunde = Zeit für 9.192.631.770 Zustandsübergänge des Cäsium-133-Atoms
  - 1 Atomsekunde = mittlere Sonnensekunde im Jahr der Einführung
- Bureau International de l'Heure (BHI) bestimmt die internationale Atomzeit (TAI)



- Problem: Atomzeit nicht genau Sonnenzeit
- Einführung einer *Universellen Koordinierten Zeit* (UTC) mit Schaltsekunden
- Schaltsekunden werden immer dann eingefügt (herausgenommen), wenn die Differenz zwischen Sonnenzeit und UTC größer als 800 ms wird.
- UTC wird von Kurzwellensender (WWV) ausgestrahlt (Fort Colling, ca. 100km nördlich von Denver)

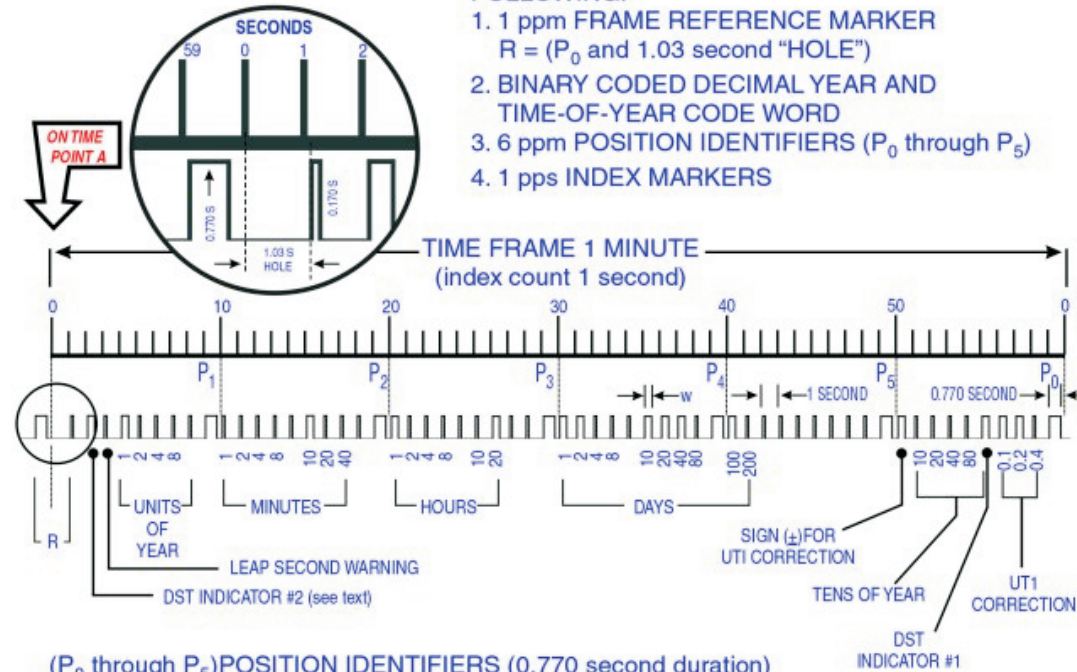


# WWV Time Code Format

## WWV and WWVH TIME CODE FORMAT

MODIFIED IRIG H FORMAT IS COMPOSED OF THE FOLLOWING:

1. 1 ppm FRAME REFERENCE MARKER  
R = ( $P_0$  and 1.03 second "HOLE")
2. BINARY CODED DECIMAL YEAR AND TIME-OF-YEAR CODE WORD
3. 6 ppm POSITION IDENTIFIERS ( $P_0$  through  $P_5$ )
4. 1 pps INDEX MARKERS



( $P_0$  through  $P_5$ ) POSITION IDENTIFIERS (0.770 second duration)

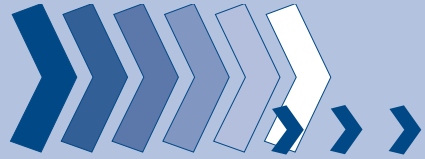
W WEIGHTED CODE DIGIT (0.470 second duration)

DURATION OF INDEX MARKERS, UNWEIGHTED CODE, AND UNWEIGHTED CONTROL ELEMENTS = 0.170 SECONDS

NOTE: BEGINNING OF PULSE IS REPRESENTED BY POSITIVE-GOING EDGE.

UTC AT POINT A = 2001, 173 DAYS, 21 HOURS, 10 MINUTES

UT1 AT POINT A = 2001, 173 DAYS, 21 HOURS, 10 MINUTES, 0.3 SECONDS



Wie sind Uhren in verteilten Systemen miteinander synchronisierbar?

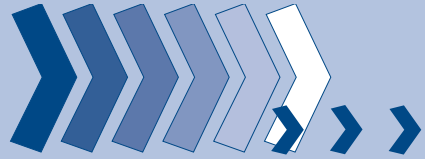
Zwei Aspekte:

- **Logische Uhren**

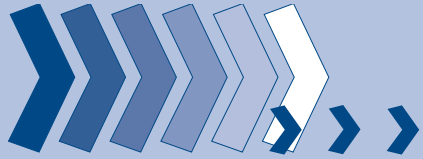
- relative Zeit im verteiltem System
- interne Konsistenz
- keine Berücksichtigung der Realzeit

- **Physikalische Uhren**

- Verwendung realer Zeit bzw. Abbildungen realer Zeit

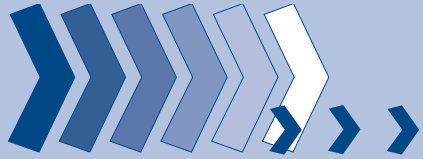


- Eine Reihe von Problemen ist unabhängig von der realen Zeit
- Wichtig: Konsistente Zeitsicht
- Zeit wird als Abfolge von Ereignissen betrachtet, die für alle Prozesse die gleiche Ordnung haben (siehe Gruppenkommunikation)
- Beispiellösung: Algorithmus von LAMPORT



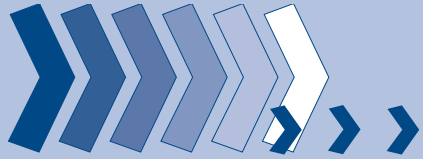
## Wiederholung: Happens-before-Relation

- LAMPORT benutzt die Relation „*happens before*“ ( $\prec$ )
- Zwei Ereignisse  $e_1$  und  $e_2$  erfüllen die Happens-before-Relation ( $e_1 \prec e_2$ ), wenn folgendes gilt:
  - $e_1$  und  $e_2$  treten bei einem Prozeß genau in dieser Reihenfolge auf, oder
  - $e_1$  ist das Senden einer Nachricht und  $e_2$  ihr Empfang, oder
  - es gibt ein Ereignis  $e'$ , so daß  $e_1 \prec e'$  und  $e' \prec e_2$  gilt
- Wenn für zwei Ereignisse  $e_1$  und  $e_2$  weder  $e_1 \prec e_2$  noch  $e_2 \prec e_1$  gilt, so heißen  $e_1$  und  $e_2$  *nebenläufig* (Notation:  $e_1 || e_2$ )



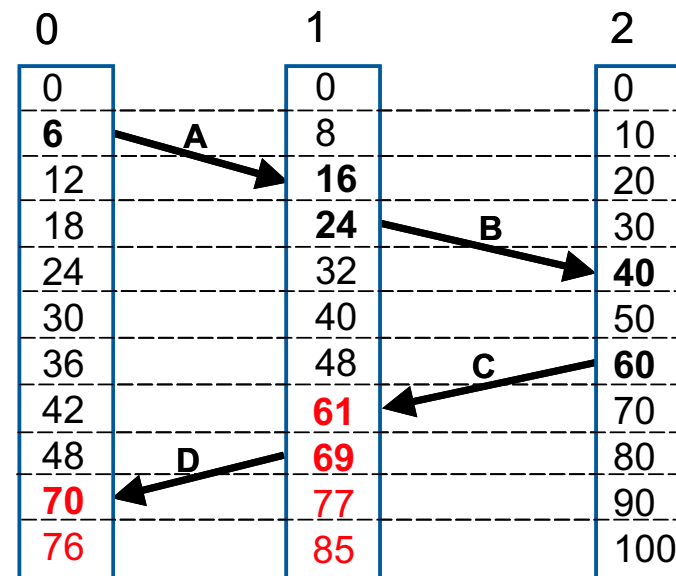
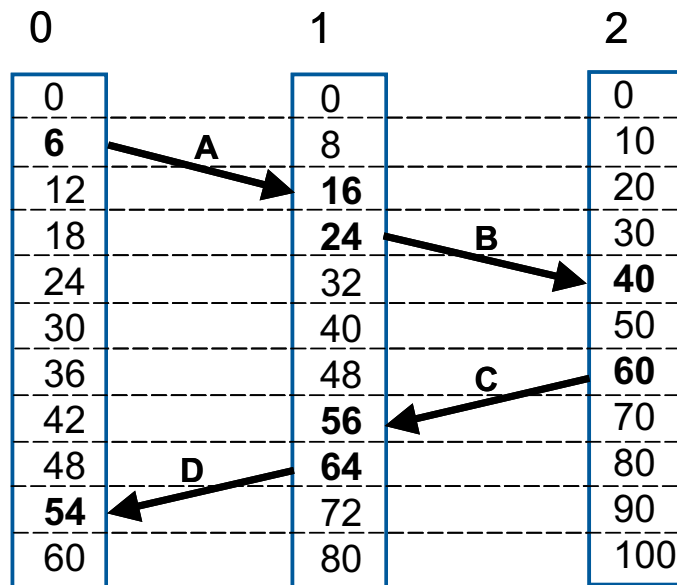
# LAMPORT-Algorithmus (I)

- Anforderungen:
  - Für jedes Ereignis  $e$  wird ein Zeitwert  $C(e)$  angegeben, über den sich alle Prozesse einig sind ( $C(e)$  soll unabhängig vom Prozeß gelten)
  - Es soll gelten:  $e_1 \prec e_2 \Rightarrow C(e_1) < C(e_2)$
  - $C$  soll (streng) monoton steigend sein
  - $\rightarrow$  Korrektur der Zeit nur durch Addition eines positiven Wertes möglich
- Ansatz:
  - Jedes Ereignis  $e$  bekommt einen Zeitstempel (bzgl. der lokalen Uhr)
  - Bearbeitet ein Prozeß ein Ereignis, dessen Zeitstempel  $s_t$  größer oder gleich der lokalen Zeit  $c_l$  ist, so wird zunächst die lokale Zeit verstellt:  $c_l := s_t + 1$



# Beispiel für LAMPORT-Zeit

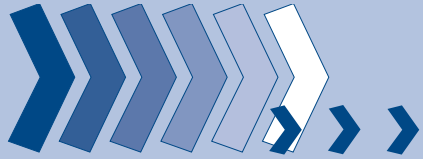
- Drei Prozesse mit lokalen Uhren
- Uhren laufen mit unterschiedlichen Raten





## Physikalische Uhren

- Auch physikalische Uhren werden in der Regel nur „ungefähr“ die genaue Zeit anzeigen
- Aber: Im Gegensatz zu logischen Uhren werden strengere Anforderungen an die Beziehung der lokalen Uhrzeit zur tatsächlichen Uhrzeit gestellt
- Abweichung einer lokalen Zeit von der tatsächlichen Zeit: *Drift*
- Angezeigte Uhrzeit:  $C(t)$



## Korrekte und perfekte Uhren

- Idealerweise hat eine Uhr keine Drift  
⇒ perfekte Uhr:  $C(t) = t$

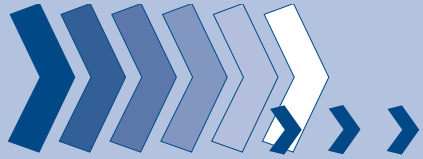
- Echte Uhren sind nicht perfekt, Driftrate:  $p$

- Nichtideale Uhren können *korrekt* sein

- Korrekte Uhr:

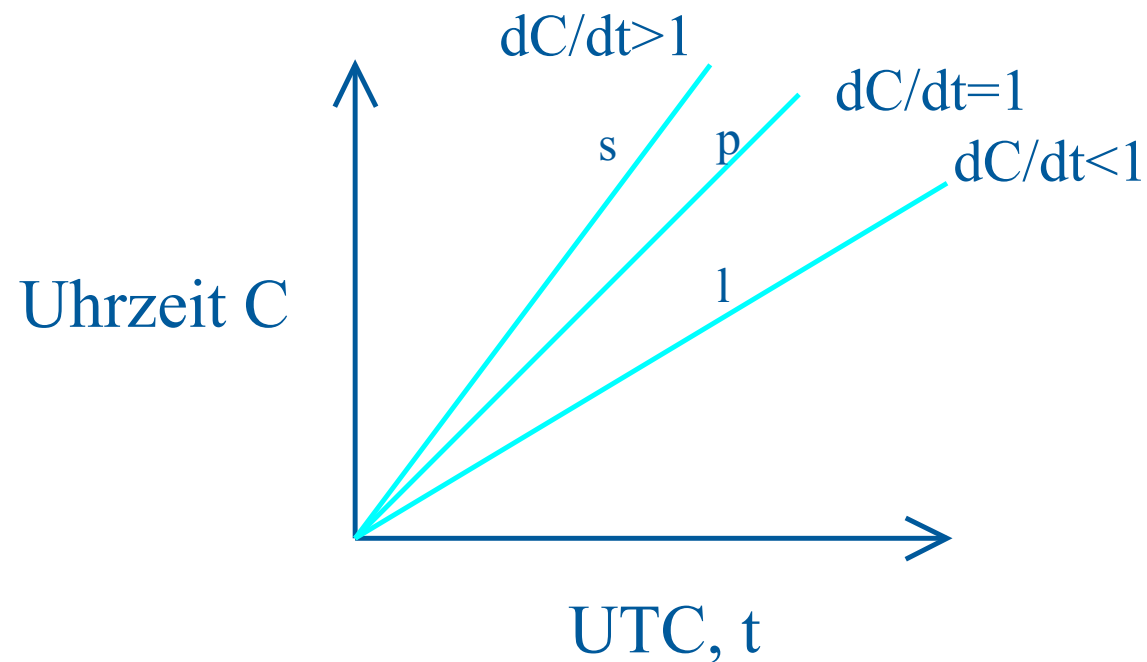
$$\frac{1}{1+p} \leq \frac{dC(t)}{dt} \leq 1 + p$$

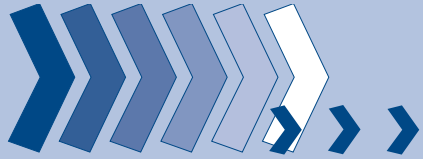
- Achtung: Eine korrekte Uhr kann stark falsch gehen (Korrektheitsannahme sagt nur etwas über die Drift, nicht über die angezeigte Zeit)
- Unkorrekte Uhren verletzen z.B. die Driftannahme, gehen rückwärts oder zeigen willkürliche Zeiten an



## Vor- und nachgehende Uhren

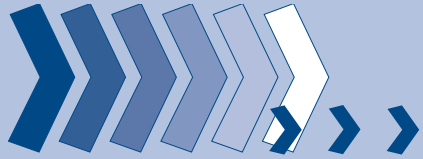
- Selbst wenn eine korrekte Uhr zu einem Zeitpunkt die richtige Zeit anzeigt, wird sie später vor oder nachgehen
- Synchronisation ist nötig





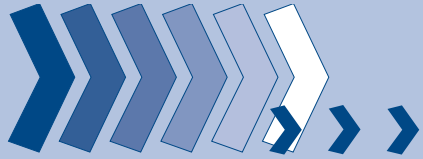
## Synchronisationsintervall

- Zwei korrekte Uhren mit der Drift  $p$  sollen nicht mehr als  $\Delta$  auseinanderlaufen. Wie groß ist das maximal erlaubte Synchronisationsintervall?
- Annahmen:  
zu  $t = 0$  seien die Uhren synchron, worst case:  $\frac{C_1(t)}{t} = 1 + p$ ,  $\frac{C_2(t)}{t} = \frac{1}{1+p}$
- Wenn zu  $t = 0$  synchron, dann  $C_1(t) = (1 + p)t$  und  $C_2(t) = \frac{t}{1+p}$
- $(1 + p)t - \frac{t}{1+p} \leq \Delta$
- $\frac{(1+p)^2 t - t}{1+p} \leq \Delta$
- $t \frac{2p+p^2}{1+p} \leq \Delta$
- $t \leq \Delta \frac{1+p}{2p+p^2}$
- Die Uhren müssen nach spätestens  $\Delta \frac{1+p}{2p+p^2}$  synchronisiert werden
- Für sehr kleine  $p$ :  $t \approx \frac{1}{2p} \Delta$

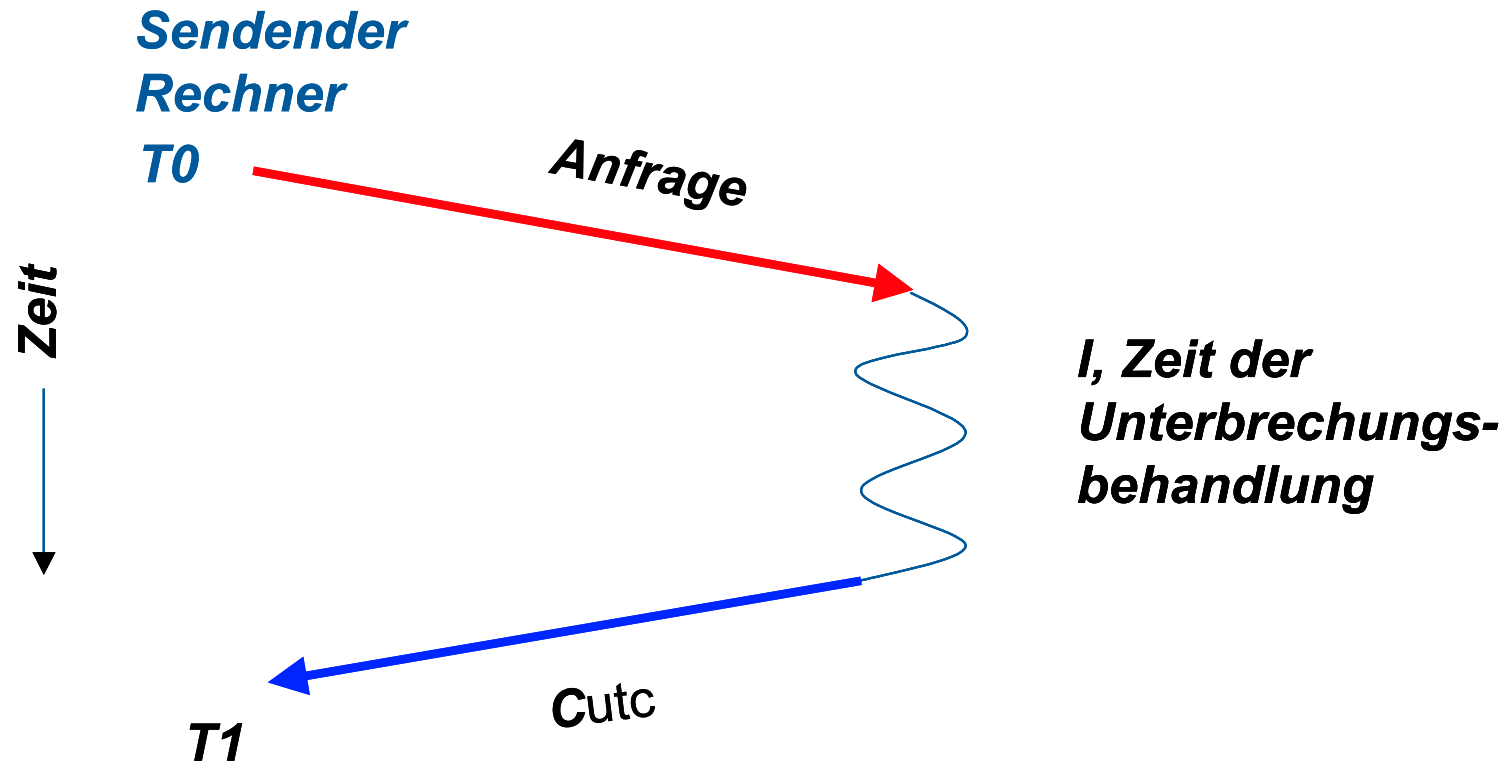


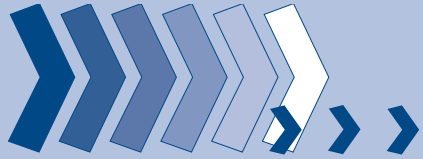
## Uhrenynchronisation nach CRISTIAN (I)

- Algorithmus von CRISTIAN (1989)
- Annahme: Ein Rechner ist mit einem WWV-Empfänger ausgestattet
- Ziel: alle Rechner sollen mit dem Zeit-Server synchronisiert werden
- Es werden korrekte Uhren angenommen
- Ansatz: Eine Uhr schickt genügend häufig einen Request an den Zeitserver. Dieser antwortet so schnell wie möglich mit der Zeit  $C_{UTC}$
- Erste Näherung:  $C = C_{UTC}$



# Uhrenynchronisation nach CRISTIAN (II)

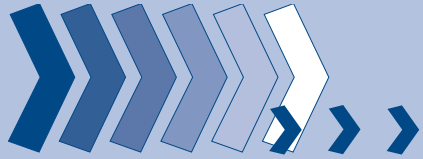




## Uhrenynchronisation nach CRISTIAN (III)

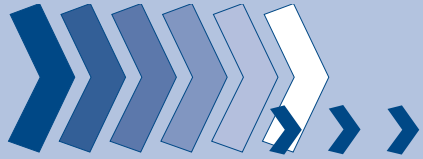
Probleme:

- **Problem 1:** Zeit darf niemals rückwärts laufen
  - Es könnte sein, daß die Uhr des Klienten vorgeht
- Lösung:
  - Der Klient ändert seine Uhr in mehreren Schritten
  - Z.B.: Normalerweise wird pro Tick um 10 ms erhöht, nun nur um 9 ms
- **Problem 2:** Es vergeht eine gewisse Zeit, bis der Zeitserver antwortet
- Lösung: Einrechnen der Verzögerung
  - Wenn Verzögerung bekannt, berücksichtigen
  - Sonst:  $(T_1 - T_0)/2$  als Verzögerung annehmen
  - Mehrere Messungen; Mittelwertbildung (Verwerfen von Ausreißern)

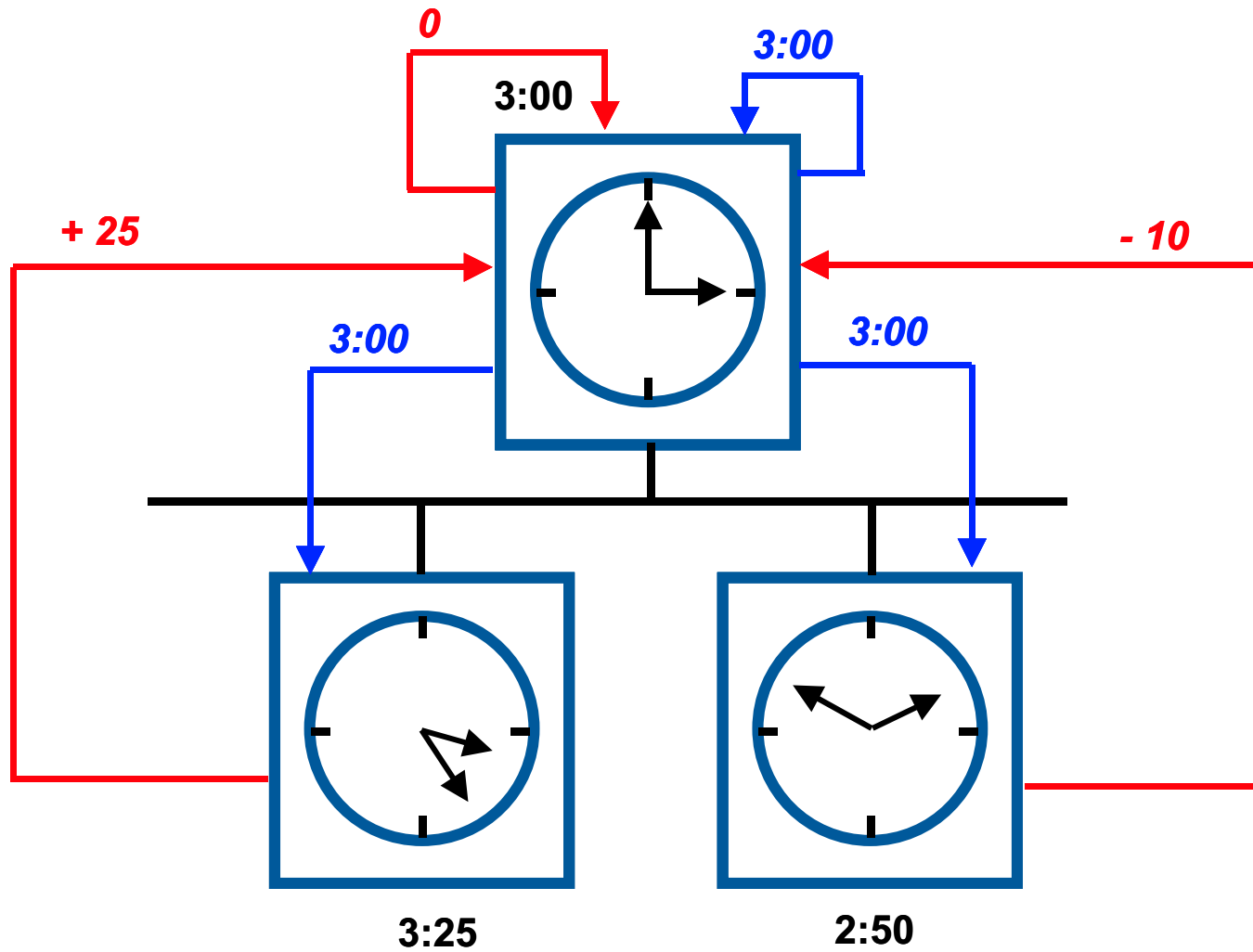


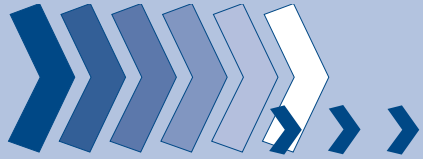
## *Berkley-Algorithmus (I)*

- 1988 entwickelt an der Berkley University
- verzichtet auf WWV-Empfänger
- Entgegengesetzter Ansatz: statt passiven Zeitserver aktiver Zeit-Daemon, der alle Zeiten abfragt
- Daemon ändert die Zeit im System, auch bei sich selbst

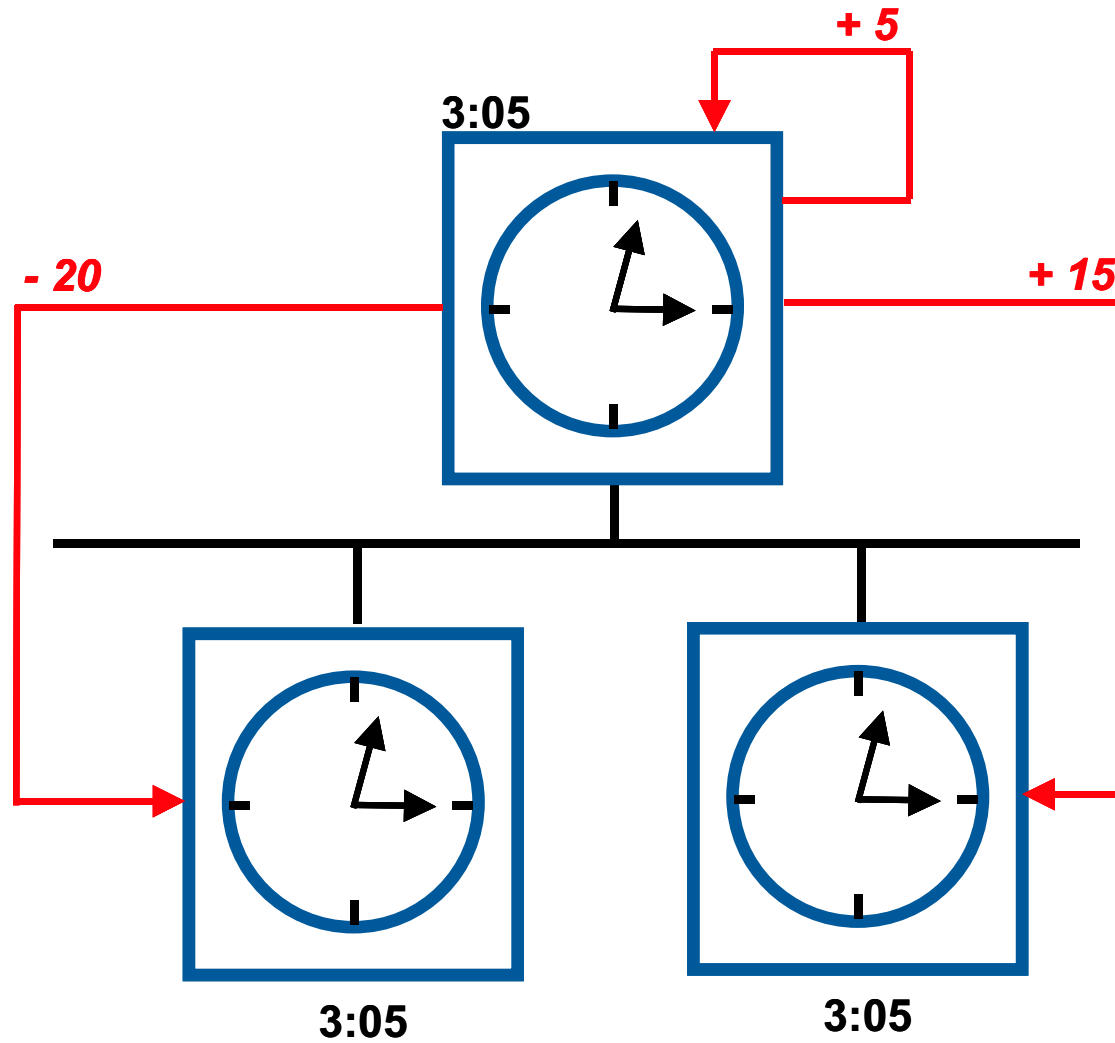


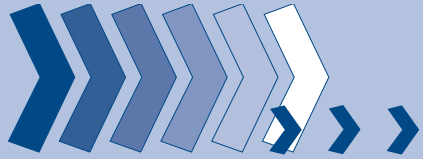
# Berkeley-Algorithmus (II)



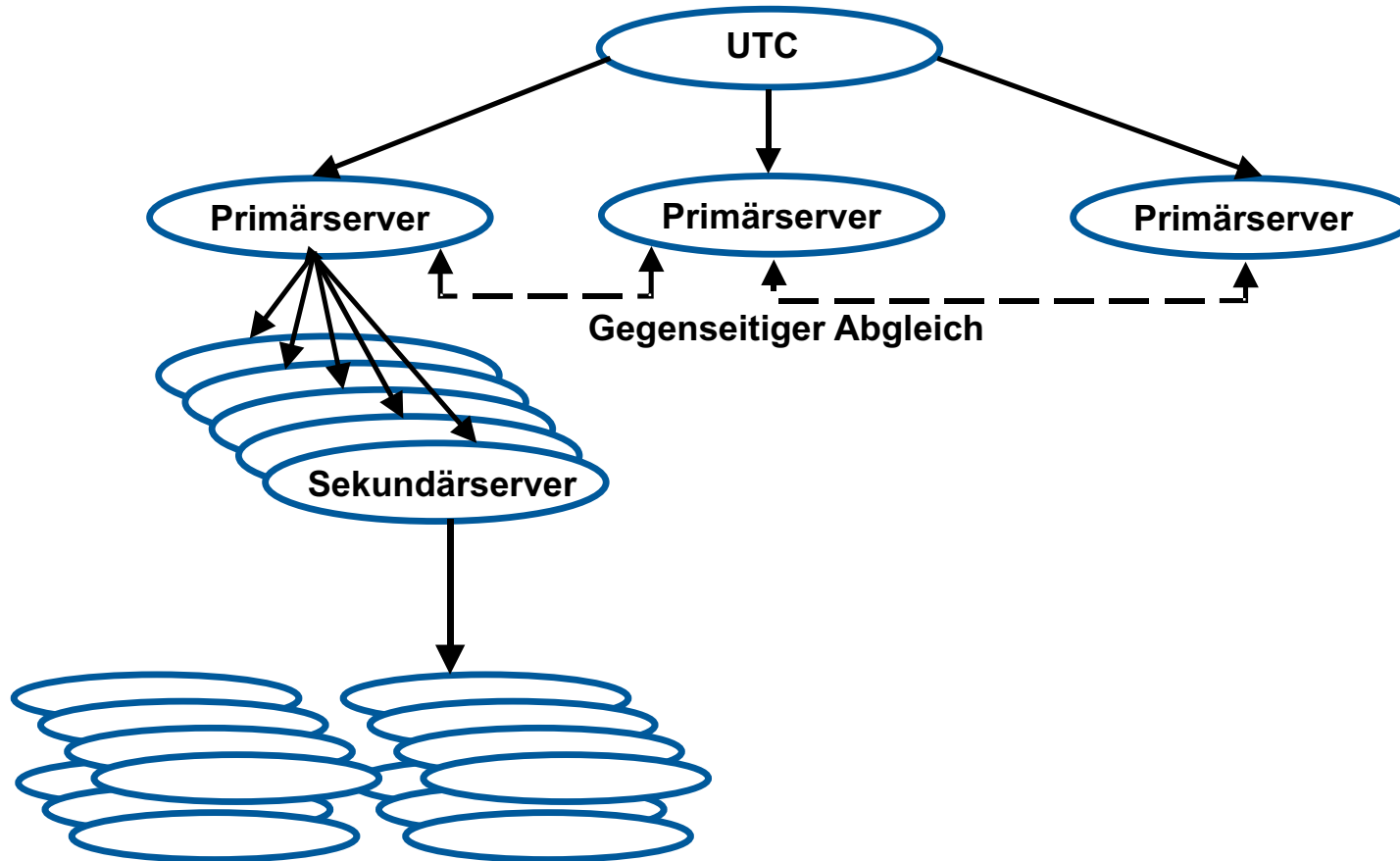


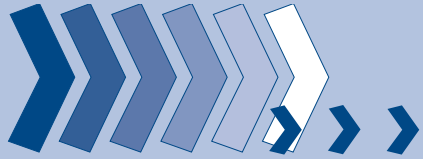
# Berkeley-Algorithmus (III)





# Network Time Protokoll - NTP



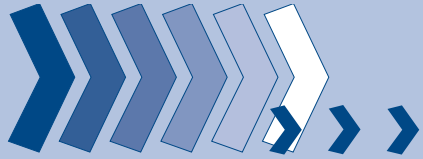


## Grenze für Uhrensynchronisation

- Wie genau lassen sich  $n$  Uhren synchronisieren?
- Vereinfachte Annahmen:
  - Perfekte Uhren (keine Drift)
  - Schwankende Verzögerung der Nachrichtenübermittlung:  
Unsicherheit  $\epsilon$
  - Stand der einzelnen Uhren anfangs unbekannt

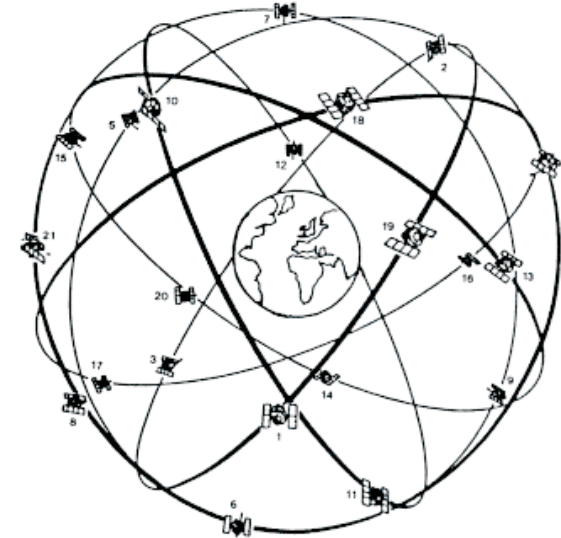
**Theorem 0.1.** *Es gibt keinen Algorithmus, der  $n$  Uhren enger als auf ein Intervall der Größe  $\epsilon \left(1 - \frac{1}{n}\right)$  garantiert synchronisiert.*

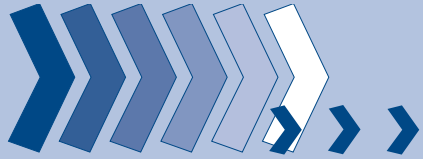
- Diese Grenze ist hart, da es Algorithmen gibt, die sie erreichen
- Beweis siehe LUDELIUS und LYNCH, *Information and Control*, Vol. 62, 1984 (Semesterapparat)



## Mobile Zeitsynchronisation - GPS

- GPS - Global Positioning System dient normalerweise zur Positionsbestimmung
- Bei bekannter Zeit kann Entfernung zum Satelliten bestimmt werden
- 3 Entfernungen legen Punkt fest
- Bei unbekannter Zeit: 4 Entfernungen
- Durch Überbestimmung kann Zeit ermittelt werden
- Problem: Willkürlich falsche Zeiten aus taktischen Gründen
- Mögliche Lösung: Differenzen-GPS





## *Inkorrekte Uhren*

- Inkorrekte Uhren können jede beliebige Zeit anzeigen - und das bei jeder Abfrage!
- Mittelwertbildung schlägt fehl
- Lösung: Byzantinisches Agreement
- $\Rightarrow$  weniger als ein Drittel aller Uhren im System dürfen inkorrekt sein