

4.3 Beispiel: Wahlalgorithmen (*election algorithms*)

Wird ein zentraler **Koordinator** benötigt (vgl. 4.2, S. 12), so müssen sich die Stationen entsprechend einigen.

Voraussetzungen:

- Stationen tragen verschiedene Nummern „ID“
- als Koordinatorin gesucht ist die „größte“ Station
- Stationen bilden einen *gerichteten Ring* wie in 4.2.2
- ID ist Initialwert einer Variablen max

→ [Chang/Roberts 1979]

Funktionsweise:

- ① Einige Stationen schicken unabhängig voneinander einen Aufruf (**Wählen!**, **ID**) an die jeweiligen Nachfolgerinnen und betrachten sich als **wählend**.
- ② Empfängerin eines Aufrufs (**Wählen!**, **x**) mit $x > \text{max}$ leitet diese Nachricht weiter (sie kommt als Koordinatorin nicht in Frage), setzt $\text{max} := x$ und betrachtet sich als **wählend**.
 - a) Ist aber $x < \text{max}$, dann *kommt sie in Frage*:
war sie **nicht wählend**, leitet sie die Nachricht
(Wählen!, max) weiter;
war sie **bereits wählend**, ist nichts zu tun
(weil max bereits gesendet wurde).

- b) Ist $x=\max$, dann ist $\max=ID$ durch den ganzen Ring gelaufen, alle Stationen haben das gleiche \max , und ID ist zur Koordinatorin gewählt.

Sie betrachtet sich als **nicht mehr wählend** und schickt die Nachricht (**fertig**) an ihre Nachfolgerin.

- ③ Empfängerin dieser Nachricht merkt sich \max als neue Koordinatorin, betrachtet sich als **nicht mehr wählend** und schickt die Nachricht weiter - sofern sie nicht selbst die Koordinatorin ist.

Analyse

Aufwand: $a(n) = \text{Anzahl der Nachrichten bei } n \text{ Stationen}$

$$2n \leq a(n) \leq 3n-1 \quad (\rightarrow \text{S. 5})$$

Korrektheit:

- ① Safety: nur eine wird gewählt
- ② Liveness: eine wird gewählt
(und alle sind informiert)
(\rightarrow S. 6)

Animation:

[http://visidia.labri.fr/examples/animations/Chang_roberts\(10\).html](http://visidia.labri.fr/examples/animations/Chang_roberts(10).html)

Aufwand :

am günstigsten: größte Station x startet die Wahl

- n Aufrufe (**Wählen!**, x) (1 Umlauf)
- n Meldungen (**Ergebnis:**, x) (1 Umlauf)

am ungünstigsten: Nachfolgerin y der größten Station x startet die Wahl

- $n-1$ Aufrufe (fast 1 Umlauf)
- n Aufrufe (**Wählen!**, x) (1 Umlauf)
- n Meldungen (**Ergebnis:**, x) (1 Umlauf)

Korrektheit:

① Safety: nur eine wird gewählt ?

Wäre mehr als eine gewählt worden, etwa x und y, dann wären beider Wahlaufrufe vollständig durch den Ring gelaufen, also der x-Aufruf durch y weitergeleitet - d.h. $x>y$ - und der y-Aufruf durch x - d.h. $y>x$ - Widerspruch!.

✓

② Liveness: eine wird gewählt und alle informiert ?

Die größte Station muss erfolgreich sein, denn ihre ID wird überall weitergeleitet, bis sie wieder bei ihr ankommt. Die ID ist dann als max bei allen bekannt.

✓

4.4 Beispiel: Verteilte Hash-Tabellen

Zur Erinnerung: **Hash-Tabelle** =

- Implementierung einer tabellierten Abbildung $\text{Key} \rightarrow \text{Data}$
- *Spezifikation:*
rechtseindeutige, endliche Relation $\subseteq \text{Key} \times \text{Data}$
mit Operationen $\text{put}(k,d)$ und $d=\text{get}(k)$
- *Implementierung:* offene Streuspeicherung unter
Verwendung einer *Hash-Funktion* h :
endlich viele Behälter (*buckets*),
 $\text{bucket}[i]$ enthält alle (k,d) mit $h(k) = i$
(„bei i kollidierende Einträge“)

Vorsicht mit dem Begriff Hash-Funktion!

Entweder kryptographische Hash-Funktion, z.B. SHA-1:

Kollision $\text{hash}(x) = \text{hash}(y)$ mit $x \neq y$ ist so extrem unwahrscheinlich, dass ein Wert $\text{hash}(x)$ nur mit Kenntnis von x berechnet worden sein kann.

Salopp gesagt: hash ist „quasi-injektiv“.

Oder Streuspeicherungs-Funktion:

Kollisionen $h(k_1) = h(k_2)$ mit $k_1 \neq k_2$ kommen vor und werden speziell behandelt, z.B. mit buckets.
Wenn man möchte, kann man bei N buckets $h(k) = \text{hash}(k) \bmod N$ wählen.

Verteilte Hash-Tabelle: kryptographische Hash-Funktion hash :

- Die Tabelleneinträge (k, d) sind über N Stationen verteilt.
- $\text{hash}(k)$ entscheidet, wo $(\text{hash}(k), d)$ (!) gespeichert wird.

Motivation: Peer-to-Peer-Systeme im Netz

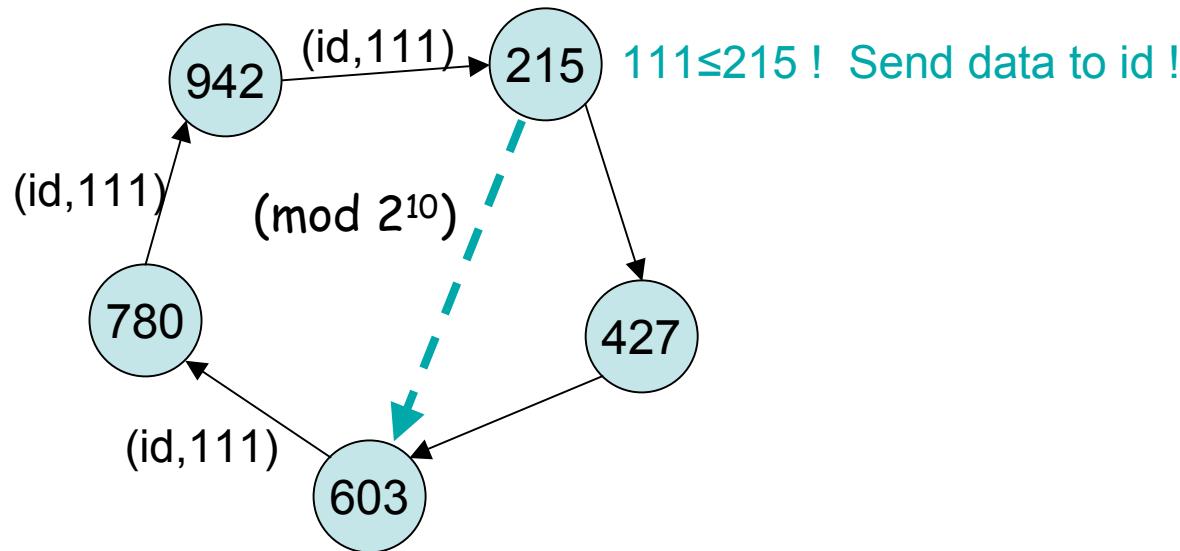
- Verteilte Speicherung großer Datenmengen
(*file sharing*, e.g., Gnutella, Napster, ...)
- beliebige umfangreiche Verzeichnis-Dienste

Das Beispiel **Chord** [I. Stoica et al. 2001]:

- Jede der N Stationen trägt eine Nummer
$$s = \text{hash}(\text{stationId}).$$
- Die Stationen kooperieren in einem *Ring* in Richtung aufsteigender Nummern (z.B. modulo 2^{160} mit SHA-1) („*overlay network*“ innerhalb des physischen Netzes):

$$s_0, s_1, s_2, \dots, s_{N-1}$$

- Eine Station s_i ist jeweils zuständig für alle k mit
$$s_{i-1} < \text{hash}(k) \leq s_i$$



Beispiel:

Station id mit Nummer 603 sucht
nach (k, d) mit $\text{hash}(k)=111$

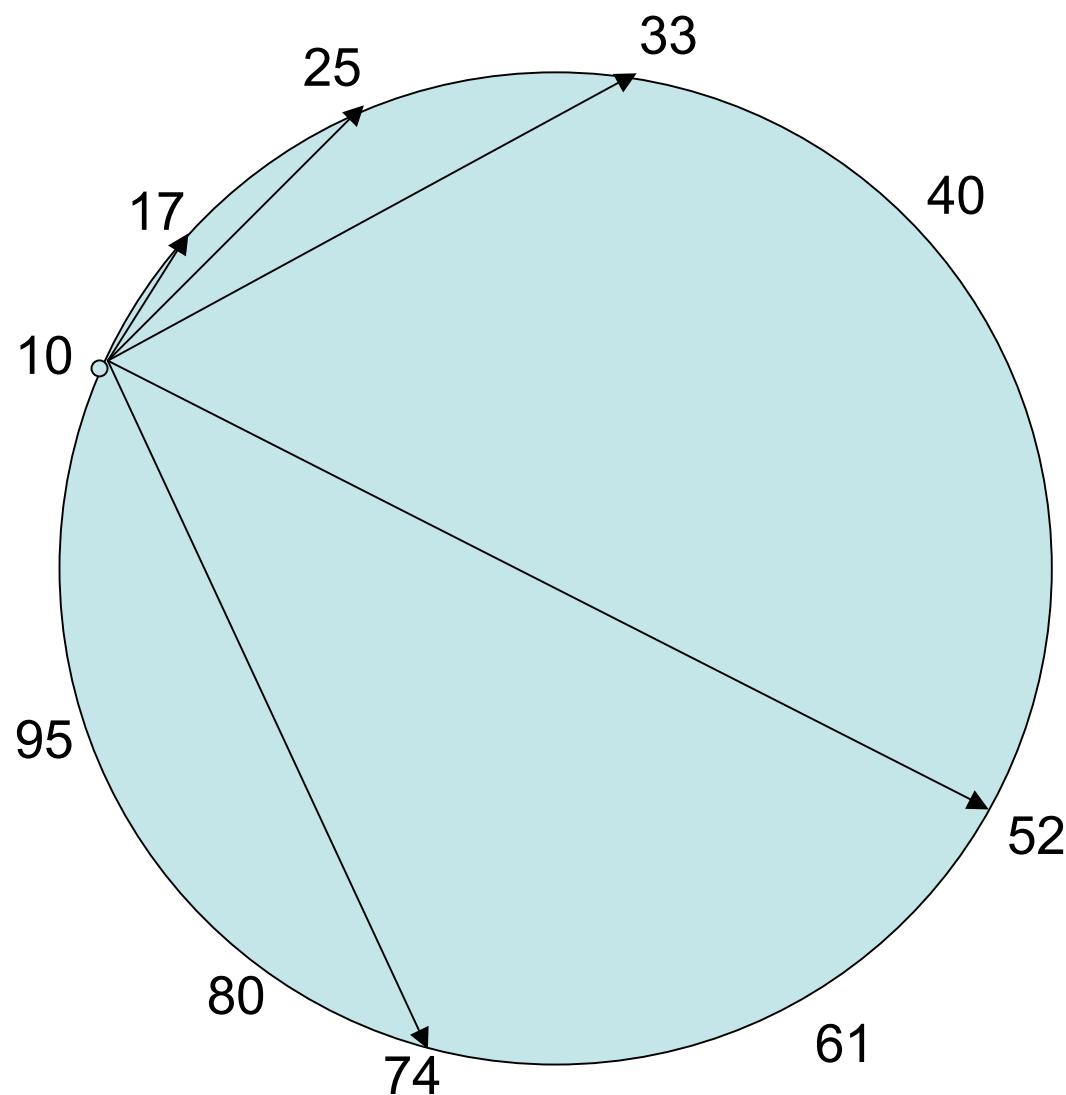
... benötigt im Mittel $N/2$ Schritte.

→ Chord macht es besser - logarithmisch.

Idee: größere Schritte in Richtung auf das Ziel machen

- ... erfordert Kenntnis weiterer Stationen, nicht nur des direkten Nachfolgers;
- damit: Anfrage für h an die *höchstnumerierte* lokal bekannte Station s mit $s < h$ schicken - und von dort entsprechend weiter.
- Präzisierung: „*finger table*“ mit $\log_2(N)$ Einträgen:
 $(s + 2^i, z), i=0, \dots, \log_2(N)-1$
 z ist die für $s+2^i$ zuständige Station,
 $\text{pred}(z) < s+2^i \leq z$

(für $s+1$ ist z der direkte Nachfolger im Ring)



$$S = 10$$

2^i	$s + 2^i$	z
1	11	17
2	12	17
4	14	17
8	18	25
16	26	33
32	42	52
64	74	74

Algorithmus zum Auffinden der für h zuständigen Station, ausgehend von nicht zuständiger Station s :

```
z = s
while succ(z) < h do
    t = fingertable of z
    z = farthest station in t with z < h od
return succ(z)
```

Bemerkung: Um der Klarheit willen ist das Protokoll als sequentieller Algorithmus auf den beteiligten *finger tables* formuliert. Die Umsetzung in den Code eines *helper*-Prozesses ist einfach.

Quasi-Animation:

www.csg.uzh.ch/teaching/fs12/p2p/lectures/Intern/M04-1up-animated.pdf, S.29ff

Was fehlt für den Einsatz in der Realität?

Verwaltung der *finger tables* angesichts

- hinzukommender Stationen
 - ausscheidender Stationen
 - zusammenbrechender Stationen
 - zusammenbrechender Kommunikation
- [I. Stoica et al. 2001] und Vorlesung Verteilte Systeme 2013

4.5 Physische Verteilung

... erfordert erheblich komplexere Protokolle

- einzelne Stationen können zusammenbrechen
- Dienstgüte des Nachrichtensystems:
 - Verlust, Duplikate, Reihenfolgetreue
- keine globale Zeit
- absolute Zeit interessiert wenig
- aber relative Zeitpunkte einzelner Ereignisse
 - können wichtig sein
- Gruppenkommunikation mit Rundsendungen
 - (*multicast*) bietet zusätzliche Möglichkeiten

4.5.1 Beispiel Wahlalgorithmus

Voraussetzungen zusätzlich zu 4.3:

- Stationen können zusammenbrechen
- Jede Station kennt *alle* ihre Partner
- Jede Station kennt die Konfiguration des Rings.

Modifikation des Algorithmus aus 4.3:

- Stationen bestätigen den Empfang einer Nachricht durch Zurücksenden einer Quittung.
- Nicht funktionsfähige Stationen werden durch Zeitüberschreitung (*timeout*) bei den Quittungen erkannt.
- Eine nicht funktionsfähige Station wird beim Herumschicken des Wahlaufrufs übersprungen.
- Im umlaufenden Wahlaufruf werden die Nummern der funktionsfähigen Stationen akkumuliert; daraus ergibt sich die aktuelle Konfiguration des Rings und die Koordinatorin.

4.5.2 Zeit und Kausalität

Def.: Ereignis (event) - drei verschiedene Arten:

local - eine prozessinterne Aktion (z.B. „Zuweisung“)

send - Abschicken einer Nachricht (als Folge von **send**)

recv - Empfangen einer Nachricht (als Folge von **recv**)

- ☞ Dienstgüte des Nachrichtensystems bleibt dabei offen

Ereignismenge, nicht näher spezifiziert: E

Def.: Kausale Abhängigkeit

Zwei Ereignisse $a, b \in E$ stehen in der Beziehung $a \rightarrow b$
(„a vor b“, „a happened before b“, „b ist kausal abhängig von a“)

zueinander, wenn gilt:

entweder

- ① a und b gehören zum selben Prozess
und geschehen in dieser Reihenfolge

oder

- ② a ist das Abschicken einer Nachricht,
b ist das Empfangen dieser Nachricht

oder

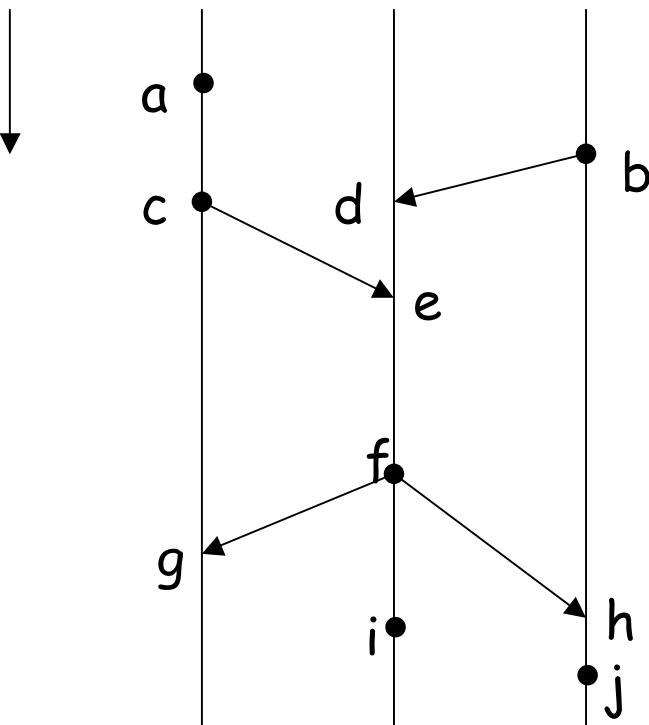
- ③ es gibt ein Ereignis c mit $a \rightarrow c$ und $c \rightarrow b$
(Transitivität von \rightarrow)

Bemerkung: Die Relation „gleich oder vor“
ist partielle Ordnung auf E: Kausalordnung

Def.: Zwei Ereignisse a,b sind voneinander unabhängig
(auch *nebenläufig, concurrent, causally unrelated*), wenn
weder $a \rightarrow b$ noch $b \rightarrow a$

Veranschaulichung der partiellen Ordnung durch Zeitdiagramm (ähnlich dem Hasse-Diagramm für partielle Ordnungen)

Zeit



z.B.

$$a \rightarrow c \rightarrow e$$

$$b \rightarrow d \rightarrow e$$

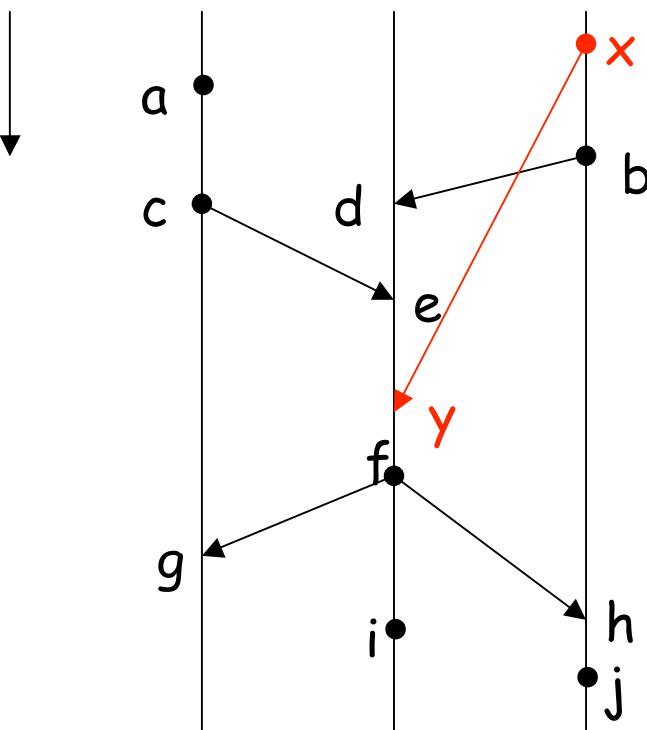
$c \rightarrow h !$

nicht

$$a \rightarrow d, g \rightarrow j !$$

Veranschaulichung der partiellen Ordnung durch Zeitdiagramm (ähnlich dem Hasse-Diagramm für partielle Ordnungen)

Zeit



z.B.

$$a \rightarrow c \rightarrow e$$

$$b \rightarrow d \rightarrow e$$

$c \rightarrow h$!

nicht

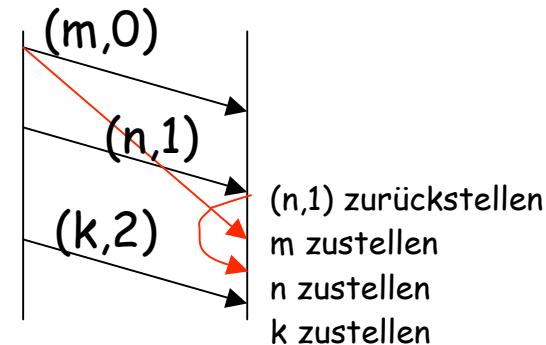
$$a \rightarrow d, g \rightarrow j$$
 !

Aus $x \rightarrow b$, $d \rightarrow y$ folgt,
daß die beiden Nachrichten
nicht FIFO übertragen wurden!

Dienstgüte des Nachrichtensystems:

Reihenfolgetreue (FCFS, FIFO) bei Sender/Empfänger-Paar:

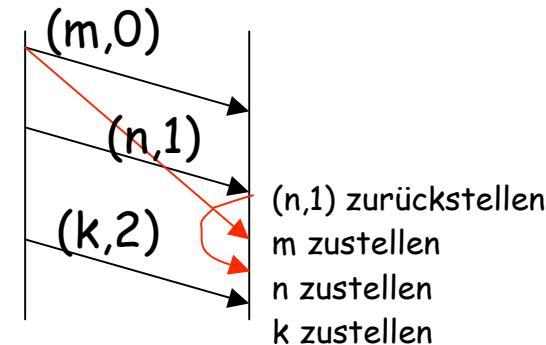
gegebenenfalls erzwingen mit
Durchnumerieren der Nachrichten
(vgl. TCP versus UDP)



Dienstgüte des Nachrichtensystems:

Reihenfolgetreue (FCFS, FIFO) bei Sender/Empfänger-Paar:

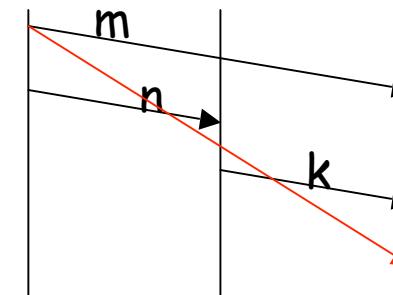
gegebenenfalls erzwingen mit
Durchnumerieren der Nachrichten
(vgl. TCP versus UDP)



Kausalitätstreue bei mehreren Beteiligten:

m wird vor k gesendet,
also auch vor k empfangen

wie erzwingen ?



4.5.3 Logische Uhren

in Anlehnung an die Kausalitätsbeziehung

Ziel: Jedem Ereignis $e \in E$ wird eine „Zeit“ $C(e) \in T$ mit gewissem T zugeordnet.

Die Zeiten sind partiell geordnet, und die Ordnung sollte isomorph zur Kausalordnung sein:

$$(E, \rightarrow) \cong (T, <)$$

Die Abbildung $C: E \rightarrow T$ heißt **logische Uhr**.

1. Versuch: Skalare Zeit [Lamport 1978]

$T = \text{natürliche Zahlen}$

(totale Ordnung - daher zum Scheitern verurteilt)

- Jeder Prozess führt in einer lokalen Uhr eine *lokale Zeit c* (anfangs 0).
- Jeder Prozess versieht jede versendete Nachricht mit einem *Zeitstempel (timestamp) t = c*.
- Zwischen je zwei Ereignissen wird c um 1 erhöht.
- Zusätzlich wird nach einem *recv-Ereignis* mit Zeitstempel t die Zeit c auf $\max(c, t+1)$ gesetzt.

Für beliebige Ereignisse $a, b \in E$ gilt

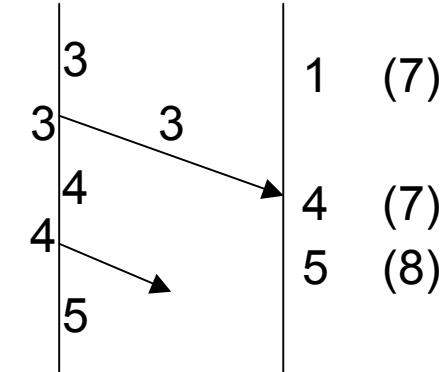
$$a \rightarrow b \Rightarrow C(a) < C(b),$$

aber nicht die Umkehrung!

Damit gilt

$$C(a) = C(b) \Rightarrow \neg (a \rightarrow b \vee b \rightarrow a),$$

aber nicht $C(a) = C(b) \Rightarrow a = b$



Feststellung: Unabhängige Ereignisse können gleiche Zeiten haben!

Skalarzeit kann in verteilten Algorithmen anstelle der
- nicht vorhandenen - globalen Zeit eingesetzt werden:

Hängt man an die Skalarzeit die *Stationsnummer* an,
so kann man die Ereignisse gemäß dieser Kennung
linear anordnen - verträglich mit ihrer Kausalordnung,
z.B. $31 < 41 < 42 < \dots$ („topologisches Sortieren“).

Damit ist wenigstens garantiert, dass alle Zeitstempel
verschieden sind und dass der Fall „a vor b“ in den
zugehörigen Zeitstempeln mit $C(a) < C(b)$ reflektiert wird.

2. Versuch: **Vektorzeit** [Fidge, Mattern]

$T = n$ -Tupel natürlicher Zahlen (bei n Prozessen 1,...,n)

$t \leq s : t_i \leq s_i$ für alle $i=1,\dots,n$ \rightarrow Halbordnung !

Z.B. weder $(1,0,2) < (2,1,1)$ noch umgekehrt

Jeder Prozess p

- führt in einer lokalen Uhr eine *lokale Vektorzeit* c (anfangs $(0,0,\dots)$),
- versieht jede versendete Nachricht mit *Zeitstempel* $t = c$.
- Vor jedem Ereignis wird c_p um 1 erhöht.
- Nach recv mit Zeitstempel t werden für alle $i=1,\dots,n$ die c_i auf $\max(c_i, t_i)$ gesetzt.

Mit der *Vektorzeit C* erreichen wir die gewünschte *Isomorphie*:

$$(E, \rightarrow) \cong (T, <)$$

Skalarzeit und Vektorzeit sind vielfältig einsetzbar
→ Verteilte Systeme SS 2013

Zusammenfassung

- Verteilte Algorithmen sind i.d.R. schwierig
- ... insbesondere bei physischer Verteilung
- i.d.R. nicht Teil von Anwendungssoftware
- Hilfreich sind
 - virtuelle Ringe
 - logische Uhren

Quellen

Erlang: www.erlang.org/doc.html

Einführung: queue.acm.org/detail.cfm?id=1454463

AB-Protokoll: en.wikipedia.org/wiki/Alternating_bit_protocol

G. Ricart, A.K. Agrawala: *An Optimal Algorithm for Mutual Exclusion in Computer Networks.* CACM 24.1, January 1981,
<http://dl.acm.org/citation.cfm?id=358537> ;
Ferner: CACM 24.9, September 1981

E. Chang, R. Roberts: *An Improved Algorithm for Decentralized Extrema-finding in Circular Configurations of Processes.*
<http://dl.acm.org/citation.cfm?id=359108> ;
CACM 22.5, May 1979

DHTs: http://en.wikipedia.org/wiki/Distributed_hash_table

I. Stoica et al.: *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications.* Proc. ACM SIGCOMM 2001,
<http://pdos.csail.mit.edu/papers/ton:chord/paper-ton.pdf>

L. Lamport: *Time, clocks, and the ordering of events in a distributed system.* CACM 21.7, July 1978
<http://dl.acm.org/citation.cfm?doid=359545.359563>

- C. J. Fidge: *Timestamps in Message-Passing Systems That Preserve the Partial Ordering.* Proc. 11. Australian Computer Science Conf, February 1988
- F. Mattern: *Virtual Time and Global States of Distributed Systems.* Proc. Workshop on Parallel and Distributed Algorithms, Elsevier, October 1988

... und die Hinweise in

www.inf.fu-berlin.de/lehre/WS12/alp5/literatur.html