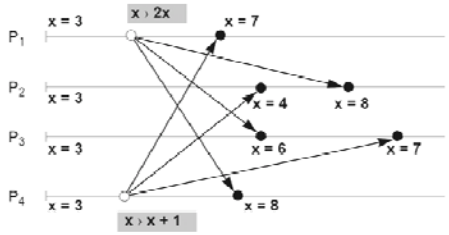
Probleme mit (kausalen) Broadcasts?

- Beispiel: Aktualisierung einer replizierten Variablen x:



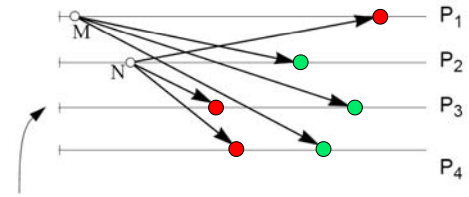
Konkrete Problemursache:
- Broadcasts werden nicht überall „gleich-zeitig“ empfangen
- dies führt lokal zu verschiedenen Emp-fangsreihenfolgen

- Abstrakte Ursache:
  - die Nachrichtenübermittlung erfolgt (erkennbar!) nicht-atomar
- → Auch kausale Broadcasts haben keine „ideale“ Semantik
  (im Sinne einer Illusion von speicherbasierter Kommunikation)

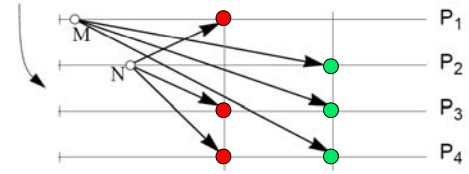
Atomarer Broadcast

- *Definition:* Wenn zwei Prozesse P_1 und P_2 beide die Nachrichten M und N empfangen, dann empfängt P_1 die Nachricht M vor N genau dann, wenn P_2 die Nachricht M vor N empfängt
- Anschaulich: Nachrichten eines Broadcasts werden „überall quasi gleichzeitig“ empfangen
- Beachte: „Atomar“ heisst hier nicht „alles oder nichts“
  (wie etwa beim Transaktionsbegriff von Datenbanken)

Atomarer Broadcast: Beispiel



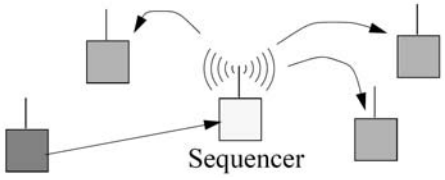
Äquivalent bzgl. „Gummiband-Transformation“



Beachte: das Senden wird nicht als Empfang der Nachricht beim Sender selbst gewertet

Realisierung von atomarem Broadcast

1. Lösung: Zentraler „Sequencer“, der Reihenfolge festlegt



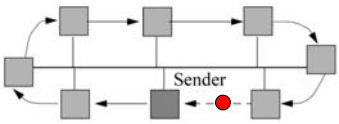
Sequencer ist allerdings ein potentieller Engpass!

- „Unicast“ vom Sender zum Sequencer
- Multicast vom Sequencer an alle
- Sequencer wartet jew. auf Acknowledgements von allen
  - oder genügt hierfür ein FIFO-Broadcast ohne Acknowledgements?

Realisierung von atomarem Broadcast

Verfahren berücksichtigt auch Nachrichtenverluste und gecrashte Prozesse

2. Lösung: Token, das auf einem (logischen) Ring kreist

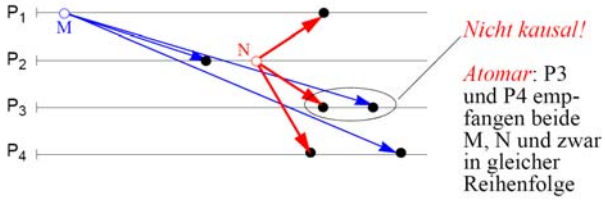


- Token = Senderecht (Token weitergeben!)
- Broadcast selbst z.B. über ein zugrunde liegendes broadcastfähiges Medium

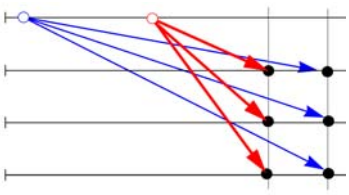
- Token führt eine Sequenznummer (inkrementiert beim Senden); so werden alle Broadcasts global nummeriert
- Empfänger wissen, dass Nachrichten entsprechend der (in den Nachrichten mitgeführten Sequenznummer) ausgeliefert werden müssen
- Bei Lücken in den Nummern: dem Token einen Wiederholungswunsch mitgeben (Sender erhält damit implizit ein NACK bzw. ACK)
- Tokenverlust (z.B. durch Prozessor-Crash) durch Timeouts feststellen (Vorsicht: Gefahr, dass dabei Token unabsichtlich verdoppelt wird!)
- Einen gecrashten Prozess (der z.B. das Token nicht entgegennimmt) aus dem logischen Ring entfernen
- Variante (z.B. bei vielen Teilnehmern): Token auf Anforderung direkt zusenden (broadcast: „Token bitte zu mir“), dabei aber Fairness beachten

Wie „gut“ ist atomarer Broadcast?

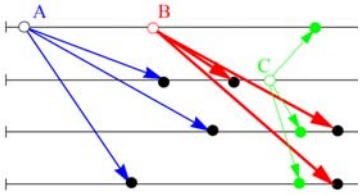
1) Ist atomar auch kausal?



2) Ist atomar wenigstens FIFO?



3) Ist atomar + FIFO evtl. kausal?



Fazit: Semantik von Broadcast

- Atomare Übermittlung ⇏ kausale Reihenfolge
- Atomare Übermittlung ⇏ FIFO-Reihenfolge
- Atomare Übermittlung + FIFO ⇏ kausale Reihenfolge
  - Bemerkung zu vorheriger Seite: nicht nur 3), sondern auch 1) ist ein (Gegen)beispiel, da M, N FIFO-Broadcast ist
- Vergleich mit speicherbasierter Kommunikation:
  - Kommunikation über gemeinsamen Speicher ist atomar (alle „sehen“ das Geschriebene gleichzeitig – so sie hinschauen)
  - Kommunikation über gemeinsamen Speicher wahrt Kausalität (Wirkung tritt unmittelbar mit dem Schreibereignis als Ursache ein)

Kausaler atomarer Broadcast

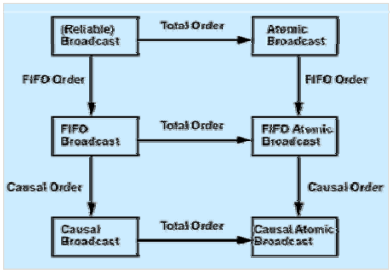
- Der speicherbasierten Kommunikation vergleichbares Kommunikationsmodell per Nachrichten: kausaler atomarer Broadcast
  - kausaler Broadcast + atomarer Broadcast
- Man nennt daher kausale, atomare Übermittlung auch virtuell synchrone Kommunikation
- Denkübung: realisieren die beiden Implementierungen „zentraler Sequencer“ bzw. „Token auf Ring“ die virtuell syn. Kommunikation?

Stichwort: Virtuelle Synchronität

- Idee: Ereignisse finden zu verschiedenen Realzeitpunkten statt, aber zur gleichen logischen Zeit
  - logische Zeit berücksichtigt nur die Kausalstruktur der Nachrichten und Ereignisse; die exakte Lage der Ereignisse auf dem „Zeitstrahl“ ist verschiebbar (Dehnen / Stauchen wie auf einem Gummiband)
- Innerhalb des Systems ist synchron (im Sinne von „gleichzeitig“) und virtuell synchron nicht unterscheidbar
  - identische Kausalbeziehungen
  - identische totale Ordnung aller Ereignisse
- Konsequenz: Nur mit Hilfe Realzeit / echter Uhr könnte ein externer Beobachter den Unterschied feststellen
- Den Begriff „logische Zeit“ werden wir später noch genauer fassen (mehr dazu dann wieder in der Vorlesung „Verteilte Algorithmen“)

Broadcast – schematische Übersicht

- Warum nicht einziger Broadcast, der alles kann? „Stärkere Semantik“ hat auch Nachteile:
  - Leistungseinbussen
  - weniger potentielle Parallelität
  - aufwändiger zu implementieren

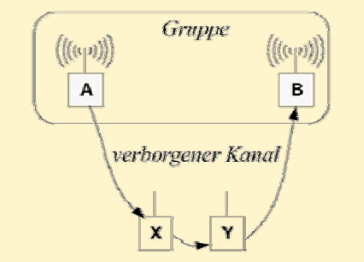


- Bekannte „Strategie“:
  - man begnügt sich daher, falls es der Anwendungskontext gestattet, oft mit einer billigeren, aber weniger perfekten Lösung
  - Motto: so billig wie möglich, so „perfekt“ wie nötig
  - man sollte aber die Schwächen einer Billiglösung kennen!
- ⇒ grössere Vielfalt ⇒ komplexer bzgl. Verständnis und Anwendung

Multicast

- Multicast = Broadcast an eine Teilmenge von Prozessen
  - diese Teilmenge wird „Multicast-Gruppe“ genannt
- Zweck von Multicast-Gruppen
  - „selektiver Broadcast“
  - Vereinfachung der Adressierung (z.B. statt Liste von Einzeladressen)
  - Verbergen der Gruppenzusammensetzung nach aussen
  - „logischer Unicast“: Gruppen ersetzen Individuen (z.B. für transparente Replikation)
- Alles, was zur Broadcastsemantik gesagt wurde, gilt (innerhalb der Gruppe) auch bzgl. Multicastsemantik:
  - zuverlässiger Multicast, FIFO-Multicast, kausaler Multicast, atomarer Multicast, kausaler atomarer Multicast

Problem der „Hidden Channels“



Kausalitätsbezüge verlassen (z.B. durch Gruppenüberlappung) die Multicastgruppe und kehren später wieder

- Soll nun das Senden von B als kausal abhängig vom Senden von A gelten?
- Global gesehen ist das der Fall, innerhalb der Gruppe ist eine solche Abhängigkeit jedoch nicht erkennbar

Dynamische Gruppen

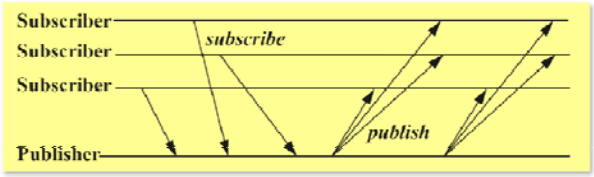
- Bei dynamischen Gruppen können Prozesse jederzeit der Gruppe beitreten oder aus der Gruppe austreten
  - Crash kann als eine besondere Austrittsform modelliert werden
- Und wenn dies während des Ablaufs einer Multicast-Operation geschieht?
  - haben verschiedene Sender an die Gruppe die gleiche Sicht bzgl. der Gruppenzusammensetzung?
- Man wünscht sich (Realisierung besprechen wir hier nicht!):
  - die Gruppe soll bei allen (potentiellen) Sendern an die Gruppe hinsichtlich der Ein- und Austrittszeitpunkte jedes Gruppenmitglieds übereinstimmen
  - Eintritt und Austritt sollen atomar erfolgen

Push-Paradigma / Publish & Subscribe

- „Push“ im Unterschied zum klassischen „Pull-“ (bzw. „Request / Reply“)-Paradigma, bei dem
  - Clients die gewünschte Information aktiv anfordern müssen,
  - sie aber nicht wissen, ob bzw. wann sich eine Information geändert hat,
  - dadurch periodisches Nachfragen („polling“) beim Server notwendig ist
- Push: „event driven“ ↔ pull: „demand driven“
- Subscriber (= Client) meldet sich für den Empfang des gewünschten Typs von Information („channel“) an

Push-Paradigma / Publish & Subscribe (2)

- Subscriber erhält automatisch (aktualisierte) Information, sobald diese zur Verfügung steht
  - „callback“ des Subscribers (= Client) durch den Publisher (= Server)



- „Publish“ entspricht Multicast
  - „subscribe“ entspricht dann einem Beitritt einer Multicast-Gruppe

Tupelräume

- Gemeinsam genutzter („virtuell globaler“) Speicher
- Blackboard- oder Marktplatz-Modell
  - Daten können von beliebigen Teilnehmern eingefügt, gelesen und entfernt werden
  - relativ starke Entkoppelung der Teilnehmer
- Tupel = geordnete Menge typisierter Datenwerte
- Entworfen 1985 von D. Gelernter (für die Sprache Linda)
- Operationen:
  - out (t): Einfügen eines Tupels t in den Tupelraum
  - in (t): Lesen und Löschen von t aus dem Tupelraum
  - read (t): Lesen von t im Tupelraum

Tupelräume (2)

- Inhaltsadressiert („Assoziativspeicher“)
  - Vorgabe eines Zugriffsmusters (bzw. „Suchmaske“) beim Lesen, damit Ermittlung der restlichen Datenwerte eines Tupels („wild cards“)
  - Beispiel: int i,j; in(„Buchung“, ?i, ?j) liefert ein „passendes“ Tupel
  - analog zu einigen relationalen Datenbankabfragesprachen
- Synchrone und asynchrone Leseoperationen
  - ‘in’ und ‘read’ blockieren, bis ein passendes Tupel vorhanden ist
  - ‘inp’ und ‘readp’ blockieren nicht, sondern liefern als Prädikat („passendes Tupel vorhanden?“) ‘wahr’ oder ‘falsch’ zurück

Tupelräume (3)

- Mit Tupelräumen sind auch die üblichen Kommunikationsmuster realisierbar, z.B. Client-Server:

```
/* Client */
...
out(“Anfrage” client_Id, Parameterliste);
in(“Antwort”, client_Id, ?Ergebnisliste);
...
/* Server*/
...
while (true)
{  in(“Anfrage”, ?client_Id, ?Parameterliste);
   ...
   out(“Antwort”, client_Id, Ergebnisliste);
}
```

Zuordnung des „richtigen“ Clients über die client_Id

Tupelräume (4)

- Kanonische Erweiterungen des Modells
  - Persistenz (Tupel bleiben nach Programmende erhalten)
  - Transaktionseigenschaft (wichtig, wenn mehrere Prozesse parallel auf den Tupelraum bzw. gleiche Tupel zugreifen)
- Problem: effiziente, skalierbare Implementierung?
  1) zentrale Lösung: Engpass
  2) replizierter Tupelraum (jeder Rechner hat vollständige Kopie des Tupelraums; schnelle Zugriffe, jedoch hoher Synchronisationsaufwand)
  3) aufgeteilter Tupelraum (jeder Rechner hat einen Teil des Tupelraums; 'out'- Operationen können z.B. lokal ausgeführt werden, 'in' evtl. mit Broadcast)
- Kritik: globaler Speicher ist der strukturierten Programmierung und der Verifikation abträglich
  - unüberschaubare potentielle Seiteneffekte

JavaSpaces

- „Tupelraum" für Java
  - gespeichert werden Objekte → neben Daten auch „Verhalten"
  - Tupel entspricht Gruppen von Objekten
- Operationen
  - write: mehrfache Anwendung erzeugt verschiedene Kopien
  - read
  - readifexists: blockiert (im Gegensatz zu read) nicht; liefert u.U. 'null'
  - takeifexists
  - notify: Benachrichtigung (mittels eines Ereignisses), wenn ein passendes Objekt in den JavaSpace geschrieben wird

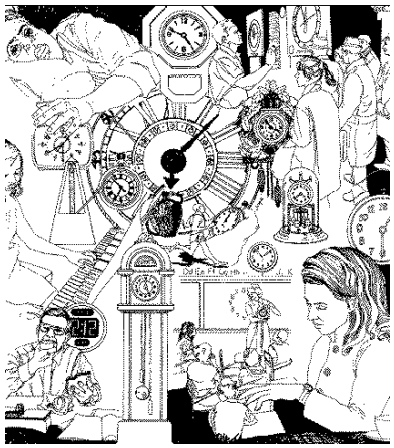
JavaSpaces (2)

- Teil von Jini
  - Kommunikation zwischen entfernten Objekten
  - Transport von Programmcode vom Sender zum Empfänger
  - gemeinsame Nutzung von Objekten
  - persistente Speicherung von Objekten (aber keine Festlegung, ob eine Implementierung fehlertolerant ist und einen Crash überlebt)
- Semantik: Reihenfolge der Wirkung von Operationen verschiedener Threads ist nicht festgelegt
  - selbst wenn ein write vor einem read beendet wird, muss read nicht notwendigerweise das lesen, was write geschrieben hat

Logische Zeit und wechselseitiger Ausschluss

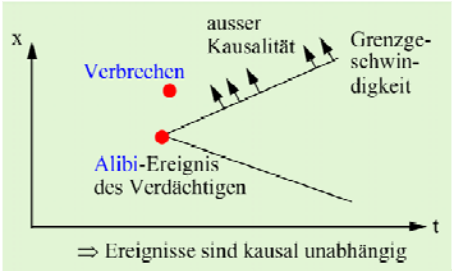
Zeit?

Ich halte ja eine Uhr für überflüssig. Sehen Sie, ich wohne ja ganz nah beim Rathaus. Und jeden Morgen, wenn ich ins Geschäft gehe, da schau ich auf die Rathausuhr hinauf, wie viel Uhr es ist, und da merke ich's mir gleich für den ganzen Tag und nütze meine Uhr nicht so ab. -- Karl Valentin



Kommt Zeit, kommt Rat

1. Volkszählung: Stichzeitpunkt in der Zukunft
   - liefert eine gleichzeitige „Beobachtung" im Nachhinein
2. Kausalitätsbeziehung zwischen Ereignissen („Alibi-Prinzip")
   - Ausschluss potentieller Kausalität
   - wurde Y später als X geboren, dann kann Y unmöglich Vater von X sein



Kommt Zeit, kommt Rat (2)

3. Fairer wechselseitiger Ausschluss
   - bedient wird derjenige, wer am längsten wartet
4. Viele weitere nützliche Anwendungen von „Zeit“ in unserer verteilten realen Welt
   - z.B. kausaltreue Beobachtung durch „Zeitstempel“ der Ereignisse

---

- Zeit ist vor allem natürlich auch dann wichtig, wenn es um Interaktionen mit der realen Welt geht
  - Prozesssteuerung, Realzeitsysteme, Cyber Physical Systems,...

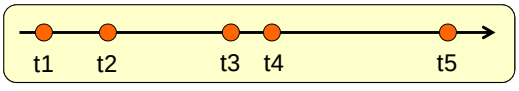
Eigenschaften der „Realzeit“

Struktureigenschaften eines „natürlichen“ Zeitpunktmodells:

- asymmetrisch (Zeit ist „gerichtet“)
- transitiv
- irreflexiv
- linear

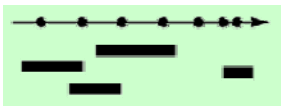
} lineare Ordnung („später als“)

- unbeschränkt („Zeit ist ewig“: Kein Anfang oder Ende)
- dicht (es gibt immer einen Zeitpunkt dazwischen)
- kontinuierlich
- metrisch
- vergeht „von selbst“
  → jeder Zeitpunkt wird schliesslich erreicht



Eigenschaften der „Realzeit“ (2)

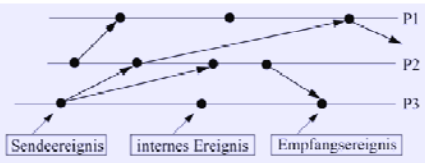
- Ist das Zeitpunktmodell adäquat? Oder sind Zeitintervalle besser?



  - wann tritt das Ereignis (?) „Sonne wird rot“ am Abend ein?

- Welche Eigenschaften benötigen wir wirklich?
  - Idee: „billigeren“ Ersatz für fehlende globale Realzeit konstruieren (z.B.: sind die reellen / rationalen / ganzen Zahlen gute Modelle?)
  - wann genügt welche Form „logischer“ statt „echter“ Zeit?
  - dazu vorher klären: was wollen wir mit „Zeit“ anfangen?

Raum-Zeitdiagramme und die Kausalrelation



Interessant dabei: von links nach rechts verlaufende „Kausalitätspfade“

Bezeichnung oft: „happened before“
- eingeführt von Leslie Lamport (1978)
- aber Vorsicht: damit ist nicht direkt eine „zeitliche“ Aussage getroffen!

Definiere eine Kausalrelation $\prec$ auf der Menge aller Ereignisse:

- Es sei $x \prec y$ genau dann, wenn:
  1) x und y auf dem gleichen Prozess stattfinden und x vor y kommt, oder
  2) x ist ein Sendeereignis und y korrespondierendes Empfangsereignis, oder
  3) $\exists$ z mit $x \prec z \ \wedge \ z \prec y$ (Transitivität)

links von

zur gleichen Nachricht gehörend

Logische Zeitstempel von Ereignissen

- Zweck: Ereignissen E eine Zeit geben
  - welche Zeit *zwischen* Ereignissen herrscht, ist irrelevant
  - gesucht ist also eine Abbildung $C: E \to \mathbb{N}$ („C“ steht für „Clock“)
  - $\mathbb{N}$ genügt hier, $\mathbb{Z}$ oder $\mathbb{R}$ ist nicht nötig, wie wir sehen werden
- C(e) nennt man den Zeitstempel von e
  - wenn C(e) < C(e'), dann nennt man e früher als e'

Ordnungsho-momorphismus

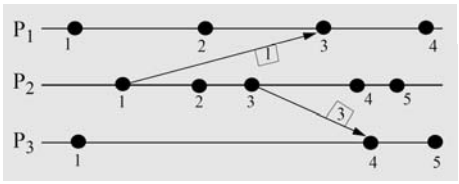
- Sinnvolle Forderung: Uhrenbedingung: $e \prec e' \Rightarrow C(e) < C(e')$
  - Interpretation („Zeit ist kausaltreu“):

    Kann ein Ereignis e ein anderes Ereignis e' beeinflussen, dann muss e einen kleineren Zeitstempel als e' haben

Communications of the ACM 21(7), 558-565, 1978: *Time, Clocks, and the Ordering of Events in a Distributed System*

Logische Uhren von Lamport

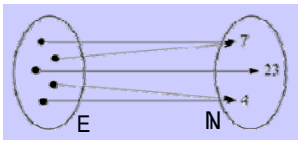
- $C: (E,\prec) \to (\mathbb{N},<)$
  Zeitstempel-Zuordnung
- $e \prec e' \Rightarrow C(e) < C(e')$
  Uhrenbedingung



- Protokoll zur Implementierung der Uhrenbedingung:
  - bei jedem Ereignis tickt die lokale Uhr (= „Zähler“)
  - Sendeereignis: Uhrwert mitsenden (→ Zeitstempel der Nachricht)
  - Empfangsereignis: Uhr = max(Uhr, Zeitstempel der Nachricht)
    (zuerst max, danach erst tickt die Uhr)
- *Behauptung:* Protokoll respektiert Uhrenbedingung
  - *Beweis:* Entlang von Kausalitätspfaden wächst logische Zeit monoton...

Lamport-Zeit: Nicht-Injektivität

- Abbildung ist nicht injektiv
  - wäre wichtig z.B. für: „Wer die kleinste Zeit hat, der gewinnt“



- Lösung: lexikographische Ordnung (C(e),i), wobei i die Prozessnummer bezeichnet, auf dem e stattfindet
  - Def.: $(a,b) < (a',b') \Leftrightarrow a<a' \vee a=a' \wedge b<b'$ (ist lineare Ordnung!)
- Alle Ereignisse haben nun verschiedene Zeitstempel
  - Abbildung ist injektiv
  - jede (nicht-leere) Menge von Ereignissen hat nun ein „frühestes“
- Abb. $(E,\prec) \rightarrow (\mathbb{N}\times\mathbb{N},<)$ respektiert die Uhrenbedingung

Umkehrung der Uhrenbedingung?

- Wieso gilt folgendes eigentlich nicht?

  $e \prec e' \Leftrightarrow C(e) < C(e')$

  - Was kann man überhaupt über die beiden Ereignisse e und e’ sagen, wenn man die Zeitstempel C(e) und C(e’) vergleicht?
- Kann man eine andere Art von Zeitstempeln finden, für die die Umkehrung der Uhrenbedingung gilt?
  - wofür wäre das nützlich?

Resümee (7a)

- Atomare Broadcasts
  - logisch gleichzeitiger Empfang der Einzelnachrichten eines Broadcasts
  - Realisierung über zentralen Sequencer bzw. Token auf einem logischen Ring
- Kausal atomare Broadcasts
  - virtuelle Synchronität
- Multicast
  - dynamische Gruppen, „hidden channels“

Resümee (7b)

- Push-Prinzip und Publish & Subscribe
- Tupelräume
  - Linda-Modell
  - JavaSpaces
- Logische Zeit
  - Raum-Zeitdiagramme, Ereignisse, Kausalrelation
  - Zeitstempel von Ereignissen
  - Uhrenbedingung (als Ordnungshomomorphismus)
- Logische Uhren von Lamport
  - Definition
  - Realisierung
  - injektive Abbildung, eindeutige Zeitpunkte