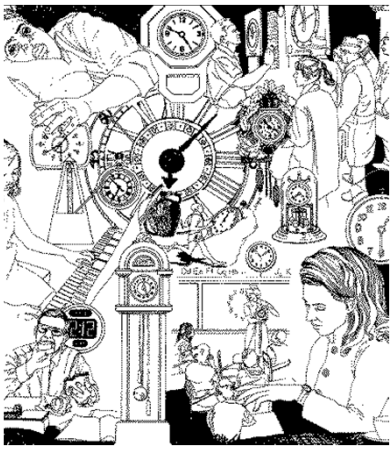
Logische Zeit

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

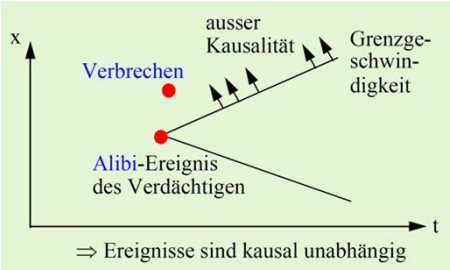


Zeit ist nützlich

Beispiele:

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



Zeit ist nützlich (2)

3. Fairer wechselseitiger Ausschluss
   - bedient wird derjenige, der am längsten wartet

4. Viele weitere nützliche Anwendungen von „Zeit“ in unserer verteilten realen Welt
   - z.B. kausaltreue Beobachtung durch „Zeitstempel“ der Ereignisse

---

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

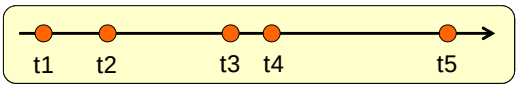
Eigenschaften der „Realzeit“

Struktureigenschaften eines „natürlichen“ Zeitpunktmodells:

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

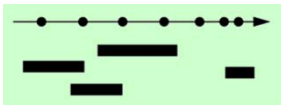
} lineare Ordnung („später als“)

- 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)

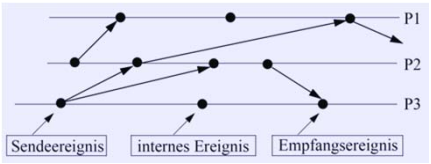
- Ist das Zeitpunktmodell adäquat? Oder sind Zeit-intervalle besser?



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

- Welche Eigenschaften benötigen wir überhaupt?
  - 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)
- 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 $C: E \rightarrow \mathbb{N}$ („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

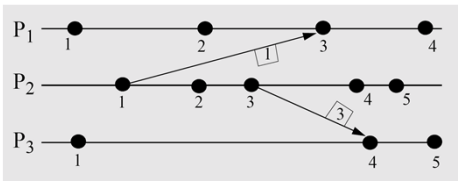
- Sinnvolle Forderung: Uhrenbedingung: $e \prec e' \Rightarrow C(e) < C(e')$
  - Interpretation („Zeit ist kausaltreu“):

    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

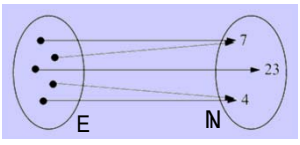
- $C: (E,\prec) \rightarrow (\mathbb{N},<)$
  Zeitstempel-Zuordnung
- $e \prec e' \Rightarrow C(e) < C(e')$
  Uhrenbedingung



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

Lamport-Zeit: injektive Variante

- Die durch Lamports Protokoll definierte 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 eine 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?

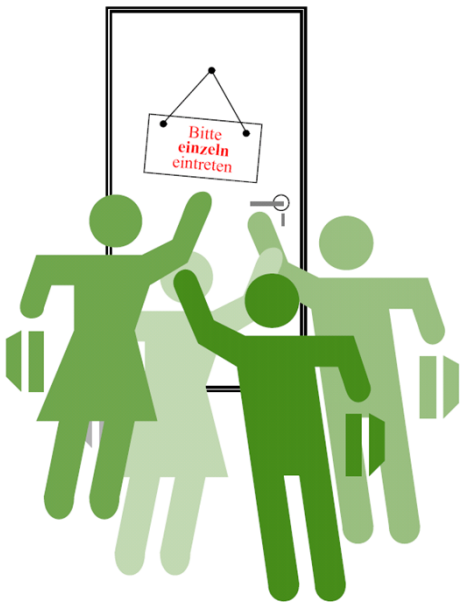
- Wieso gilt folgendes eigentlich nicht?

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

  - Was kann man überhaupt über die beiden Ereignisse e und e' sagen, wenn man die Zeitstempel C(e) und C(e') vergleicht?

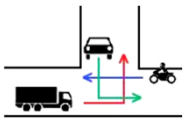
- Kann man eine andere Art von Zeitstempeln finden, für die die Umkehrung der Uhrenbedingung gilt?
  - wofür wäre das nützlich?
  - → Vorlesung „verteilte Algorithmen“

Wechselseitiger Ausschluss



Wechselseitiger Ausschluss
(Mutual Exclusion, „Mutex“)

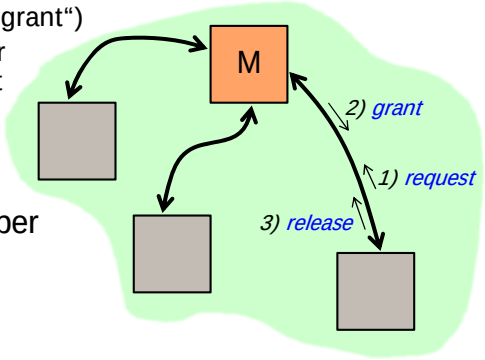
- Koordination, wenn viele wollen, aber nur einer darf
- „Streit“ um exklusives Betriebsmittel, z.B.:
  - konkrete Ressource (z.B. gemeinsamer Datenbus)
  - abstrakte Ressource (z.B. ein „Termin“ in einem verteilten Terminkalendersystem)
  - „kritischer Abschnitt“ in einem nebenläufigen Programm



- Es gibt klassische Lösungen bei shared memory
  - z.B. Semaphore und Monitore (→ Betriebssystemtheorie)
  - sind in unserem verteilten Kontext aber nicht relevant

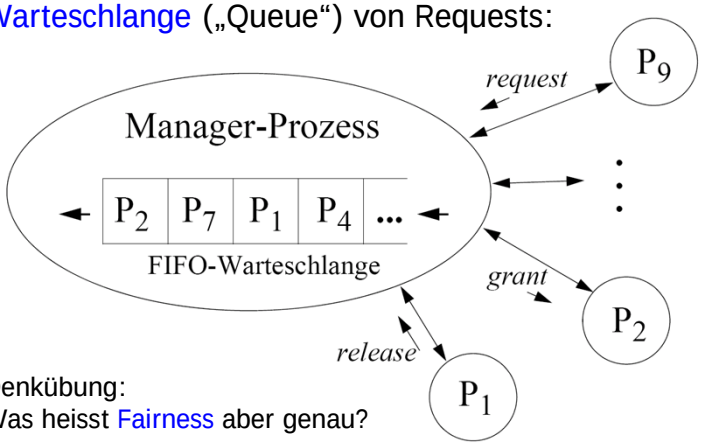
Zentraler Manager?

- Hier: Nachrichtenbasiertes System konkurrierender Prozesse
- Idee: Manager, der Ressourcen (exklusiv aber fair!) zuordnet
  - ein Prozess bewirbt sich um die Ressource mit „request“
  - wartet dann auf Erlaubnis („grant“)
  - teilt schliesslich Freigabe der Ressource dem Manager mit „release“ mit
- Vergleichsweise einfach und wenige Nachrichten, aber
  - potentieller Engpass
  - single point of failure



Globale Warteschlange garantiert Fairness

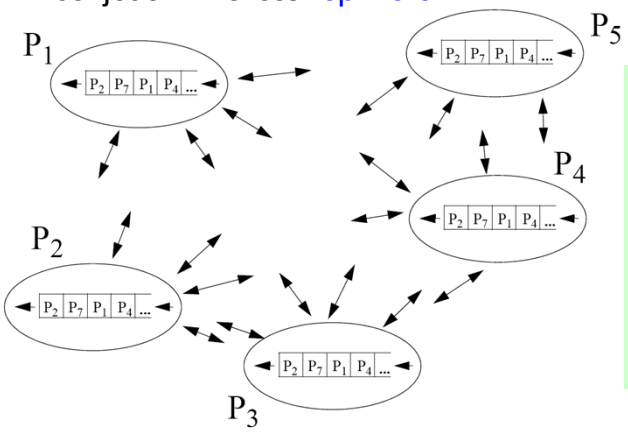
- Zentraler Manager-Prozess hält eine (zeitlich geordnete) Warteschlange („Queue“) von Requests:



- Denkübung:
  Was heisst Fairness aber genau?

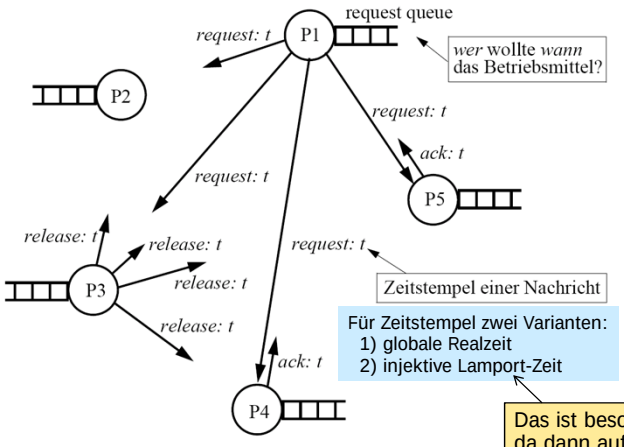
Replizierte Warteschlange?

- Idee für eine dezentrale Lösung: globale Warteschlange bei jedem Prozess replizieren



- Alle Prozesse sollen die gleiche Sicht der „virtuell globalen“ Warteschlange haben
- Konsistenz wird mit (vielen) Nachrichten und logischer Zeit erreicht (→ nächste zwei slides)

Synchronisation der Warteschlangen mit Zeitstempeln



- Voraussetzung: FIFO-Kommunikationskanäle
- Alle Nachrichten tragen (eindeutige!) Zeitstempel
- Request- und Release-Nachrichten immer an alle senden (FIFO-Broadcast)
- Requests werden bestätigt („ack“)

Der Algorithmus (Lamport 1978)

1) Bewerbung um Betriebsmittel: Request mit Zeitstempel und Absender an alle senden und in eigene Queue einfügen
2) Bei Empfang eines Request: Request in eigene Queue einfügen, ack versenden
3) Bei Freigabe des Betriebsmittels: Aus eigener Queue entfernen und Release an alle versenden
4) Bei Empfang eines Release: Zugehörigen Request aus eigener Queue entfernen
5) Ein Prozess darf das Betriebsmittel nutzen, wenn:
   a) der eigene Request der früheste in seiner Queue ist
   b) und er bereits von jedem anderen Prozess (irgendeine) spätere Nachricht bekommen hat

(Frühester Request ist global eindeutig ⇒ die beiden Bedingungen garantieren, dass kein früherer Request mehr kommt (wieso?))

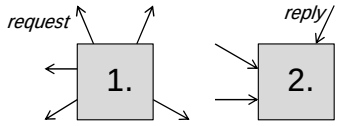
Denkübungen:
- Wieso ist FIFO notwendig?
- Wo geht (bei Lamport-Zeit) die Uhrenbedingung ein?

Wieso sind garantiert:
1) Safety (zu jedem Zeitpunkt höchstens einer),
2) Fairness (jeder Request wird „schliesslich“ erfüllt)?

3(n-1) Nachrichten pro Bewerbung (n = Zahl der Prozesse)

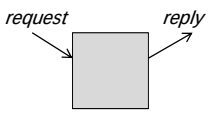
Ein anderer verteilter Mutex-Algorithmus (Ricart / Agrawala, 1981)

- Nur 2(n-1) Nachrichten (Reply übernimmt Rolle von Release und ack)



1. Sende Request (mit log. Zeitstempel!) an alle n-1 anderen
2. Dann auf n-1 Replies warten, danach Betriebsmittel nutzen

- Bei Eintreffen einer Request-Nachricht:
  - wenn nicht selbst beworben oder der Sender „ältere Rechte“ (bzgl. logischer Zeit!) hat, dann Reply sofort schicken
  - ansonsten Reply erst später (im Sinne von Release, s.o.) schicken, nach Erfüllen des eigenen Requests (d.h. dem exklusivem Zugriff)



Nur älteste Bewerbung setzt sich überall durch!

Denkübungen: Safety? Fairness? Deadlockfreiheit? FIFO notwendig?