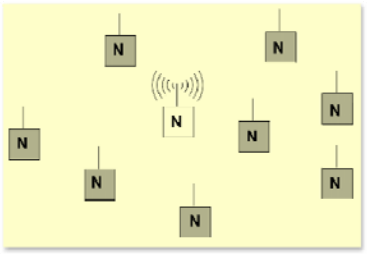
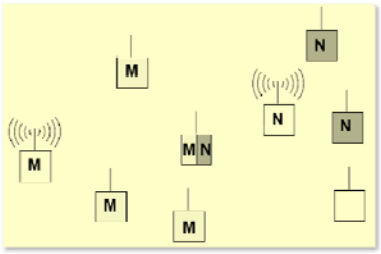
Gruppenkommunikation
(Broadcast / Multicast)



Broadcast: Senden an die Gesamtheit aller Teilnehmer

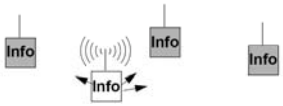


Multicast: Senden an eine Untergruppe aller Teilnehmer

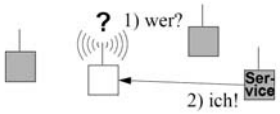
- Multicast entspricht Broadcast bezogen auf die Gruppe
  - verschiedene Gruppen können sich evtl. überlappen
  - jede Gruppe hat eine Multicastadresse

Anwendungen von Gruppenkommunikation

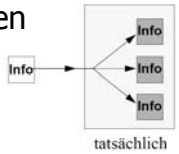
- Informieren
  - z.B. Newsdienste



- Suchen
  - z.B. Finden von Objekten und Diensten



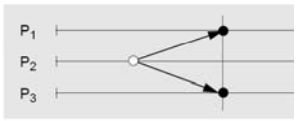
- „Logischer Unicast"
  - an replizierte Komponenten



Typische Anwendungsklasse von Replikation: Fehlertoleranz

Gruppenkommunikation – idealisierte Semantik

1. Modellhaftes Vorbild: Speicherbasierte Kommunikation
   - augenblicklicher „Empfang" in zentralistischen Systemen
   - vollständige Zuverlässigkeit (kein Nachrichtenverlust etc.)



2. Idealisierte Sicht bei nachrichtenbasierter Kommunikation:
   - gleichzeitiger Empfang
   - vollständige Zuverlässigkeit

- In verteilten Systemen ist aber beides nicht realistisch

  (2. kann allerdings in der Praxis bei Vorhandensein einer globalen Zeit evtl. durch eine „Sperrfrist" als Zeitpunkt in der Zukunft erreicht werden)

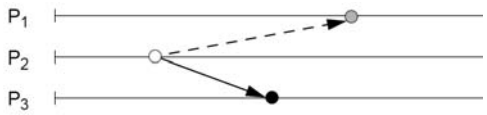
Gruppenkommunikation – tatsächliche Situation

- Zugrundeliegendes Medium (Netz) oft nicht multicastfähig
  - LANs höchstens innerhalb von Teilstrukturen; WLAN als Funknetz a priori anfällig für Übertragungsstörungen
  - multicastfähige Netze sind typischerweise nicht verlässlich (keine Empfangsgarantie)

- Daher typischerweise „Simulation" von Multicast durch ein Protokoll aus vielen Punkt-zu-Punkt-Einzelnachrichten
  - z.B. Multicast-Server, der die Information an alle Empfänger einzeln weiterverteilt

Gruppenkommunikation – tatsächliche Situation (2)

- Nachrichtenkommunikation entspricht nicht dem „Ideal"
  - nicht-deterministische Zeitverzögerung
    → Empfang zu unterschiedlichen Zeiten
  - keine garantierte Zuverlässigkeit der Nachrichtenübermittlung



- Ziel von Broadcast- / Multicast-Protokollen
  - möglichst gute Approximation der Idealsituation
- Hauptproblem bei der Realisierung von Broadcast also:
  1. Zuverlässigkeit
  2. garantierte Empfangsreihenfolge ←

Das soll dem Problem nicht-deterministischer Nachrich-tenverzögerungen begegnen

Typische Fehlerfälle bei der Gruppenkommunikation

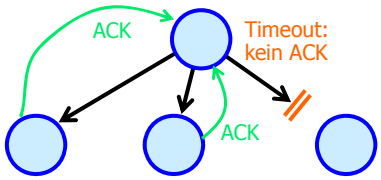
- Während des Protokolls: Verlust von Einzelnachrichten oder Ausfall des Senders
  - Nachrichten können aus unterschiedlichen Gründen verloren gehen (z.B. Netzüberlastung, Empfänger hört gerade nicht zu, Hilfsprozess in der Kommunikationsinfrastruktur abgestürzt,...)
- Problem dann: Empfänger sind sich nicht einig, da manche, aber nicht alle, informiert werden
  - Uneinigkeit der Empfänger kann die Ursache für sehr ärgerliche Folgeprobleme sein (da wäre es manchmal besser, gar kein Prozess hätte die Nachricht empfangen!)
  - partielle Abhilfe durch aufwändigere Protokolle und mehr Redundanz

„Best Effort“ Broadcast

- Euphemistische Bezeichnung, da keine extra Anstrengung
  - typischerweise einfache Realisierung ohne Ack etc.
- Keinerlei Garantien
  - unbestimmt, wie viele / welche Empfänger erreicht wurden
  - unbestimmte Empfangsreihenfolge
- Allerdings effizient (im Erfolgsfall)
- Geeignet für die Verbreitung unkritischer Informationen
  - z.B. Informationen, die Einfluss auf die Effizienz haben, nicht aber die Korrektheit betreffen
- Evtl. als Grundlage zur Realisierung höherer Protokolle
  - wenn Fehlerfall selten → aufwändiges Recovery auf höherer Ebene tolerierbar

„Reliable“ Broadcast

- Ziel: Unter gewissen (welchen?) Fehlermodellen einen „möglichst zuverlässigen“ Broadcast-Dienst realisieren



- 1. Idee: Quittung („positives Acknowledgement“: ACK) für jede Einzelnachricht
  - im Verlustfall z.B. einzeln nachliefern oder (bei broadcastfähigem Medium) einen zweiten Broadcast-Versuch (→ Duplikaterkennung!)
  - viele ACKs → teuer (d.h., das Prinzip skaliert schlecht)

„Reliable“ Broadcast mit negativen Acknowledements („NACK“)

- Alle broadcasts werden vom Sender nummeriert
- Empfänger erkennt so evtl. Lücke bei nächstem Empfang
- Bei fehlender Nachrichten wird ein NACK gesendet
- Fehlende Nachricht wird nachgeliefert
  - Sender muss daher Kopien von Nachrichten aufbewahren
  - aber wie lange?
- Broadcast einer „Nullnachricht“ ist evtl. sinnvoll
  - → schnelles Erkennen von Lücken
- Kombination von ACK / NACK mag sinnvoll sein

- Fehlermodell? Verfahren hilft gegen Verlust von Nachrichten, aber nicht gegen den Crash des Senders „mittendrin“

Ein weiterer Reliable-Broadcast-Algorithmus

- Zweck: Jeder nicht gecrashte und zumindest indirekt erreichbare Prozess soll die Broadcast-Nachricht erhalten
- Voraussetzung: zusammenhängendes „gut vermaschtes“ Punkt-zu-Punkt-Netz

Denkübungen:
- Bidirektionale Kommunikationskanäle notwendig?
- Wie effizient ist das Verfahren (Anzahl der Einzelnachrichten)?
- Wie fehlertolerant? (Wie viel darf kaputt sein / verloren gehen...?)

Sender s: Realisierung von broadcast(N)
- *send(N, s, sequ_num)* an alle Nachbarn
- sequ_num ++

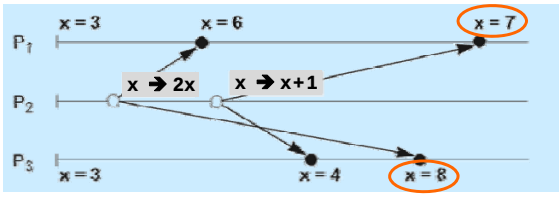
Beachte: receive ≠ deliver (unterscheide Anwendungsebene und Transportebene)

Empfänger r: Realisierung des Nachrichtenempfangs
- *receive(N, s, sequ_num);*
  wenn r noch kein *deliver(N)* für sequ_num ausgeführt hat, *dann*:
  wenn r ≠ s *dann* *send(N, s, sequ_num)* an alle Nachbarn von r ;
  Nachricht an die Anwendungsebene ausliefern („*deliver(N)*“) ;

Prinzip: „Fluten“ des Netzes (→ Vorlesung „Verteilte Algorithmen“)

Broadcast: Empfangsreihenfolge

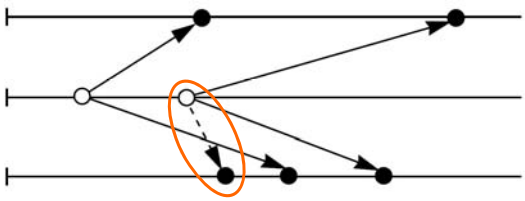
- Wie ist die Empfangsreihenfolge von Gruppennachrichten?
  - problematisch wegen der i.Allg. ungleichen Übermittlungszeiten
  - Bsp.: Update einer replizierten Variablen mittels „function shipping“:



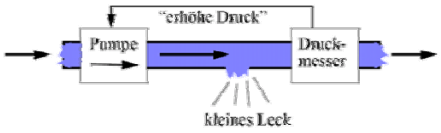
- Es sind verschiedene Grade des Ordnungserhalts denkbar
  - z.B. keine (d.h. ungeordnet), FIFO, kausal geordnet, total geordnet

FIFO-Ordnung bei Multicast

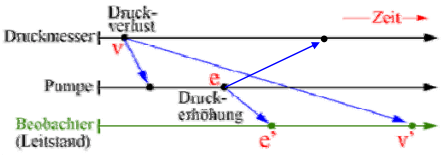
- Alle Broadcast-Nachrichten eines (d.h.: ein und des selben) Senders an eine Gruppe kommen bei allen Mitgliedern der Gruppe in FIFO-Reihenfolge an
  - Denkübung: wie dies mittels eines Protokolls garantieren?



FIFO-Broadcasts sind aber nicht immer „gut genug“



Annahme: Steuerelemente kommunizieren über FIFO-Broadcasts:



Falsche Schlussfolgerung des Beobachters:

„Aufgrund einer unbegründeten Pumpenaktivität wurde ein Leck erzeugt, wodurch schliesslich der Druck absank.“

„Irgendwie“ kommt beim Beobachter die Reihenfolge durcheinander!

Man sieht also:

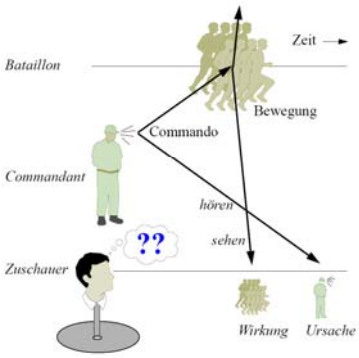
- FIFO-Reihenfolge nicht immer ausreichend, um Semantik zu wahren
- eine Nachricht verursacht oft das Senden einer anderen (→ Kausalität!)

Das „Broadcastproblem“ ist nicht neu

wenn ein Zuschauer von der Ferne das Exercieren eines Bataillons verfolgt, so sieht er übereinstimmende Bewegungen desselben plötzlich eintreten, ehe er die Commandostimme oder das Hornsignal hört; aber aus seiner Kenntnis der Causalzusammenhänge weiss er, dass die Bewegungen die Wirkung des gehörten Commandos sind, dieses also jenen objectiv vorangehen muss, und er wird sich sofort der Täuschung bewusst, die in der Umkehrung der Zeitfolge in seinen Perceptionen liegt.

Das „Broadcastproblem“ ist nicht neu

- Natürlicher Broadcast bei Licht- und Schallwellen
  - wann handelt es sich dabei um FIFO-Broadcasts?
  - wie ist es mit dem Kausalitätserhalt?

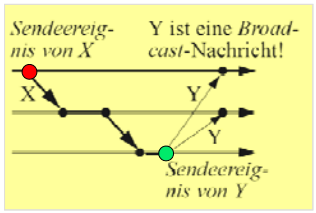


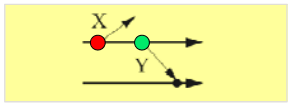
„Wenn ein Zuschauer von der Ferne das Exercieren eines Bataillons verfolgt, so sieht er übereinstimmende Bewegungen desselben plötzlich eintreten, ehe er die Commandostimme oder das Hornsignal hört; aber aus seiner Kenntnis der Causalzusammenhänge weiss er, dass die Bewegungen die Wirkung des gehörten Commandos sind, dieses also jenen objectiv vorangehen muss, und er wird sich sofort der Täuschung bewusst, die in der Umkehrung der Zeitfolge in seinen Perceptionen liegt.“

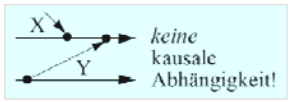
Christoph von Sigwart (1830-1904): *Logik. Zweiter Band. Die Methodenlehre* (1878)

Kausale Nachrichtenabhängigkeit

- *Definition:* Nachricht Y hängt kausal von Nachricht X ab, wenn es im Raum-Zeit-Diagramm einen von links nach rechts verlaufenden Pfad gibt, der vom Sendeereignis von X zum Sendeereignis von Y führt.





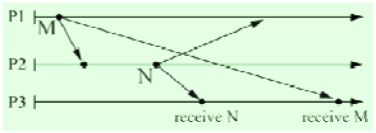


- *Beachte:* Dies lässt sich bei geeigneter Modellierung auch abstrakter fassen (→ „logische Zeit" später in der Vorlesung und auch Vorlesung „Verteilte Algorithmen")

Kausaler Broadcast

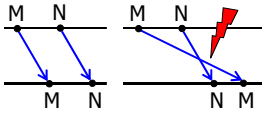
Zweck: Wahrung von Kausalität bei der Kommunikation

- *Definition:* Kausale Reihenfolge („causal order"): Wenn eine Nachricht N kausal von einer Nachricht M abhängt, und ein Prozess P die Nachrichten N und M empfängt, dann muss er M vor N empfangen haben
- „Kausale Reihenfolge" (bzw. „kausale Abhängigkeit") lassen sich insbesondere auch auf Broadcasts anwenden



Gegenbeispiel: *Keine* kausalen Broadcasts!

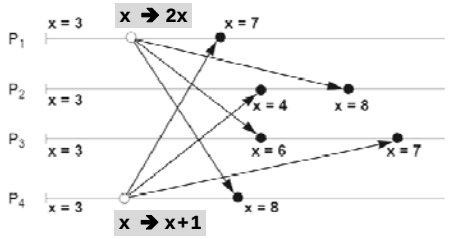
- Kausale Reihenfolge impliziert FIFO-Reihenfolge: kausale Reihenfolge ist eine Art „globales FIFO"



Das Erzwingen der kausalen Reihenfolge ist mittels geeigneter Algorithmen möglich (→ Vorlesung „Verteilte Algorithmen", z.B. Verallgemeinerung der Sequenzzählermethode für FIFO)

Probleme mit (kausalen) Broadcasts?

- Beispiel: Aktualisierung einer replizierten Variablen x:



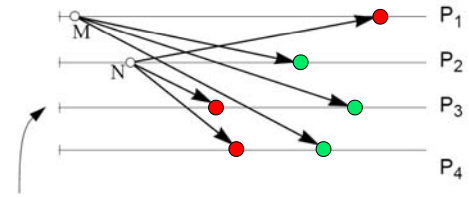
Konkrete Problemursache:
- Broadcasts werden nicht überall „gleich-zeitig" empfangen
- dies führt lokal zu verschiedenen Emp-fangsreihenfolgen

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

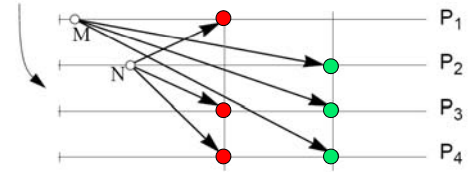
Atomarer Broadcast

- *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



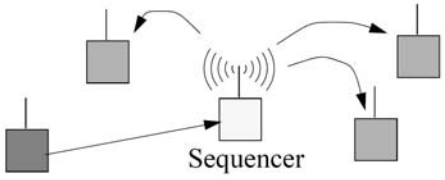
Äquivalent bzgl. „Gummiband-Transformation"



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?