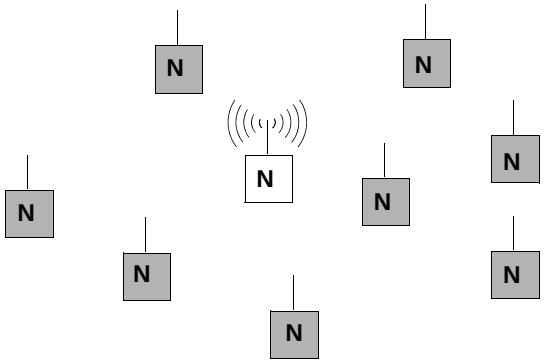
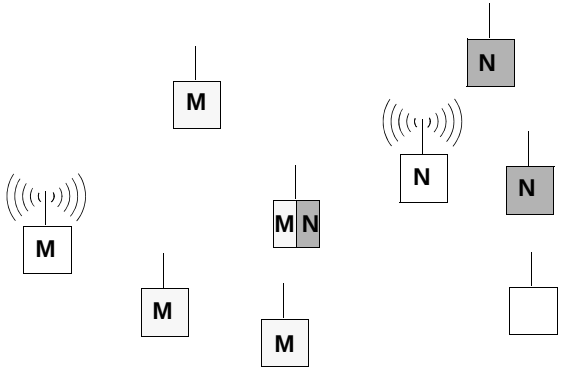
Gruppen-kommunikation

Gruppenkommunikation



Broadcast: Senden an die *Gesamtheit* aller Teilnehmer

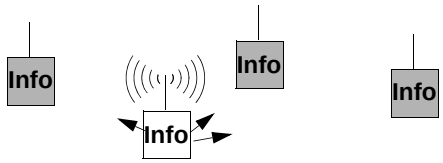


Multicast: Senden an eine *Untergruppe* aller Teilnehmer

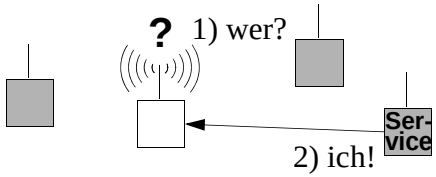
- entspricht Broadcast bezogen auf die Gruppe
- verschiedene Gruppen können sich evtl. überlappen
- jede Gruppe hat eine Multicastadresse

Anwendungen von Gruppenkommunikation

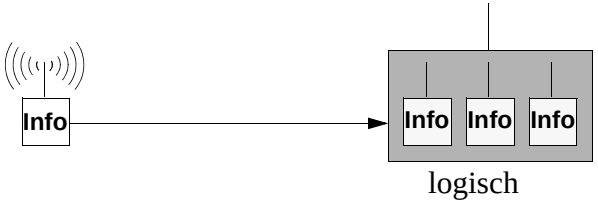
- *Informieren*
  - z.B. Newsdienste

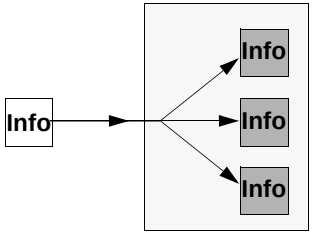


- *Suchen*
  - z.B. Finden von Objekten und Diensten



- *"Logischer Unicast"* an replizierte Komponenten

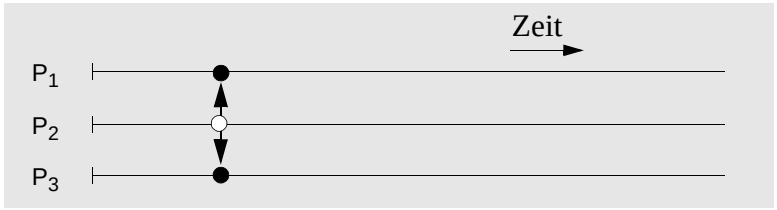




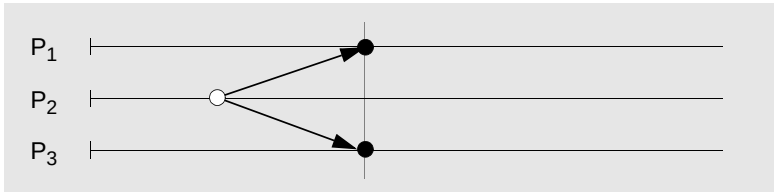
Typische Anwendungsklasse von Replikation: Fehlertoleranz

Gruppenkommunikation - idealisierte Semantik

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



- Nachrichtenbasierte Kommunikation: Idealisierte Sicht
  - (verzögerter) *gleichzeitiger* Empfang
  - vollständige Zuverlässigkeit

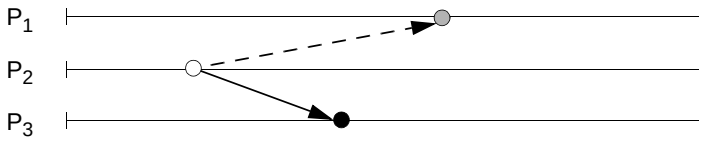


- Beide Ansichten sind aber in vert. Sys. nicht realistisch

Gruppenkommunikation - tatsächliche Situation

- Medium (Netz) ist oft nicht multicastfähig
  - LANs höchstens innerhalb von Teilstrukturen; WLAN als Funknetz a priori anfällig für Übertragungsstörungen
  - multicastfähige Netze sind typischerweise nicht verlässlich (keine Empfangsgarantie)
  - bei Punkt-zu-Punkt-Netzen: “Simulation” von Multicast durch ein Protokoll (z.B. Multicast-Server, der an alle einzeln weiterverteilt)

- Nachrichtenkommunikation ist nicht “ideal”
  - nicht-deterministische Zeitverzögerung → Empfang zu *unterschiedlichen* Zeiten
  - nur bedingte Zuverlässigkeit der Übermittlung



- Ziel von Broadcast / Multicast-Protokollen:
  - möglichst gute Approximation der Idealsituation
  - möglichst hohe Verlässlichkeit und Effizienz

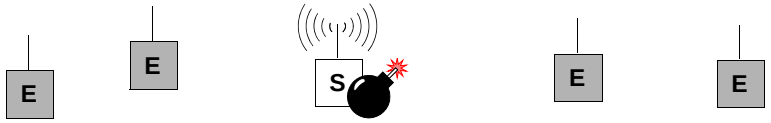
  - Beachte: Verlust von Nachrichten und sonstige Fehler sind bei Broadcast ein ernsteres Problem als beim “Unicast”! (Wieso?)
  - Hauptproblem bei der Realsierung von Broadcast: (1) Zuverlässigkeit und (2) garantierte Empfangsreihenfolge

Senderausfall beim Broadcast

Wenn Broadcast implementiert ist durch Senden vieler Einzelnachrichten (dann ist das Senden ein “nicht atomarer”, länger andauernder Vorgang)

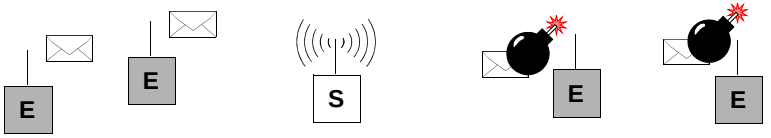
a) Sender fällt ganz aus: *kein* Empfänger erhält Nachricht

  - „günstiger“ Fall: zumindest *Einigkeit* unter den Empfängern!



b) Sender fällt *während* des Sendevorgangs aus: nur *einige* Empfänger erhalten u.U. die Nachricht

  - “ungünstiger” Fall, da *Uneinigkeit*



  - Uneinigkeit der Empfänger kann die Ursache für sehr ärgerliche Folgeprobleme sein! (Da wäre es manchmal besser, *gar kein* Prozess hätte die Nachricht empfangen!)
  - mögliche teilweise Abhilfe: Empfänger leiten zusätzlich die Nachricht auch untereinander weiter falls möglich (erhöhte Redundanz, mehr Aufwand)

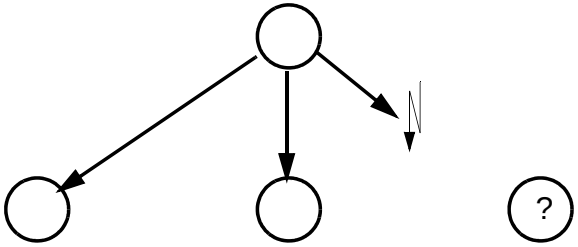
“Best effort”-Broadcast

- Fehlermodell: Verlust von Nachrichten (und evtl. temporärer Crash von Prozessen)
  - Nachrichten können aus unterschiedlichen Gründen verloren gehen (z.B. Netzüberlastung, Empfänger hört gerade nicht zu,...)
- Euphemistische Bezeichnung, da keine extra Anstrengung
  - typischerweise einfache Realisierung ohne Acknowledgements etc.
- Keinerlei Garantien
  - unbestimmt, wieviele / welche Empfänger eine Broadcastnachricht tatsächlich empfangen
  - unbestimmte Empfangsreihenfolge

  Kann z.B. beim Software-update über Satellit zu einem ziemlichen Chaos führen
- Allerdings effizient (im Erfolgsfall)
- Geeignet für die Verbreitung unkritischer Informationen
  - z.B. Informationen, die evtl. Einfluss auf die Effizienz haben, nicht aber die Korrektheit betreffen
- Evtl. als Grundlage zur Realisierung höherer Protokolle
  - diese basieren oft auf der A-priori-Broadcastfähigkeit von Netzen
  - günstig bei zuverlässigen physischen Kommunikationsmedien (wenn Fehlerfall sehr selten → aufwändiges Recovery auf höherer Ebene tolerierbar)

“Reliable” Broadcast

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



- Quittung (“positives Acknowledgement”: ACK) für jede Einzelnachricht
  - im Verlustfall einzeln nachliefern oder (falls broadcastfähiges Medium vorhanden) einen zweiten Broadcast-Versuch? (→ Duplikaterkennung!)
  - viele ACKs → teuer / skaliert schlecht
- Skizze einer anderen Idee (“negatives Ack.”: NACK):
  - alle broadcasts werden vom Sender aufsteigend numeriert
  - Empfänger stellt beim *nächsten* Empfang evtl. eine Lücke fest
  - für fehlende Nachrichten wird ein “negatives ack” (NACK) gesendet
  - Sender muss daher Kopien von Nachrichten (wie lange?) aufbewahren und fehlende nachliefern
  - “Nullnachrichten” sind u.U. sinnvoll (→ schnelles Erkennen von Lücken)
  - Kombination von ACK / NACK mag sinnvoll sein
- Dies hilft aber nicht, wenn der Sender mittendrin crasht!

Ein 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
  - Fehlermodell: Prozesse und Verbindungen mit Fail-Stop-Charakteristik (“crash” ohne negative Seiteneffekte und ohne Neustart)

Sender *s*: Realisierung von **broadcast(*N*)**

- **send(*N, s, sequ_num*)** an alle Nachbarn (inklusive an *s* selber);
- sequ_num ++

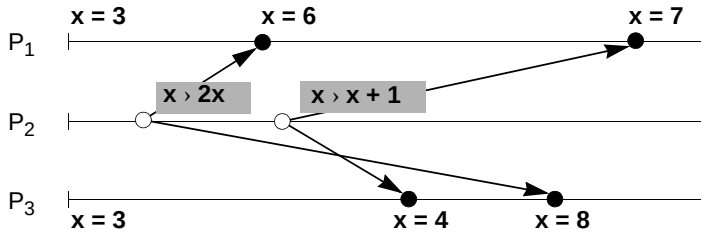
Empfänger *r*: Realisierung des Nachrichtenempfangs

- **receive(*N, s, sequ_num*);**
  wenn *r* noch kein **deliver(*N*)** für *sequ_num* ausgeführt hat, dann:
  wenn $r \neq s$ dann **send(*N, s, sequ_num*)** an alle Nachbarn von *r* ;
  Nachricht an die Anwendungsebene ausliefern (“deliver(N)”) ;

- Prinzip: “Fluten” des Netzes
  - vgl. dazu “Echo-Algorithmus” und Vorlesung “Verteilte Algorithmen”
- Beachte: receive ≠ deliver
  - unterscheide Anwendungsebene und Transportebene
- Denkübungen:
  - müssen die Kommunikationskanäle bidirektional sein?
  - wie effizient ist das Verfahren (Anzahl der Einzelnachrichten)?
  - wie fehlertolerant? (wieviel darf kaputt sein / verloren gehen...?)
  - kann man das gleiche auch anders / besser erreichen?

Broadcast: Empfangsreihenfolge

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

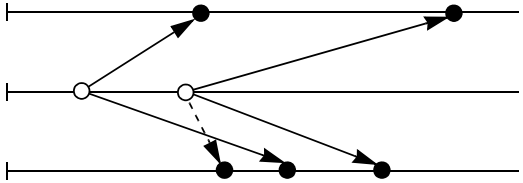


- Es sind verschiedene Grade des Ordnungserhalts denkbar
  - z.B. keine (ungeordnet), FIFO, kausal geordnet, total geordnet

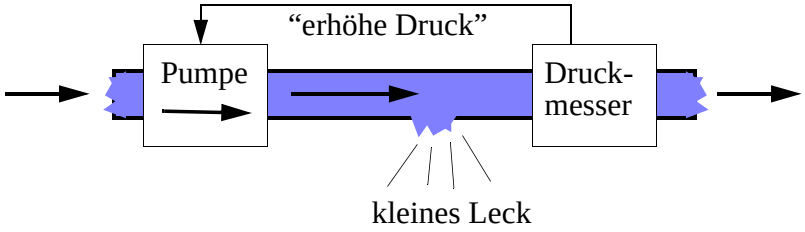
- *FIFO-Ordnung:*

  Alle Multicast-Nachrichten eines (d.h.: *ein und des selben*) Senders an eine Gruppe kommen bei allen Mitgliedern der Gruppe in FIFO-Reihenfolge an

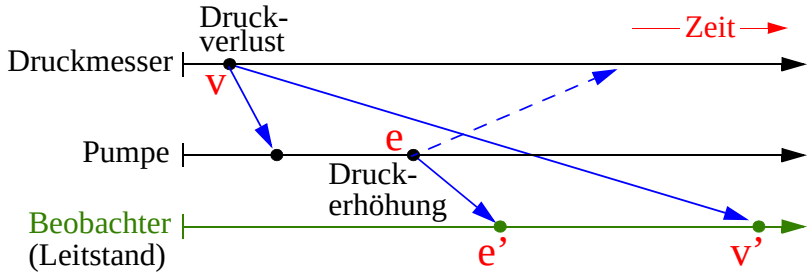
  - Denkübung: wie dies in einem Multicast-Protokoll garantieren?



Probleme mit FIFO-Broadcasts



- *Annahme*: Steuerelemente kommunizieren über FIFO-Broadcasts:



- “Irgendwie” kommt beim Beobachter die Reihenfolge durcheinander!

⇒ *Falsche Schlussfolgerung des Beobachters:*

“Aufgrund einer unbegreiflichen Pumpenaktivität wurde ein Leck erzeugt, wodurch schliesslich der Druck absank.”

Man sieht also:

- FIFO-Reihenfolge reicht oft nicht aus, um Semantik zu wahren
- eine Nachricht *verursacht* oft das Senden einer anderen

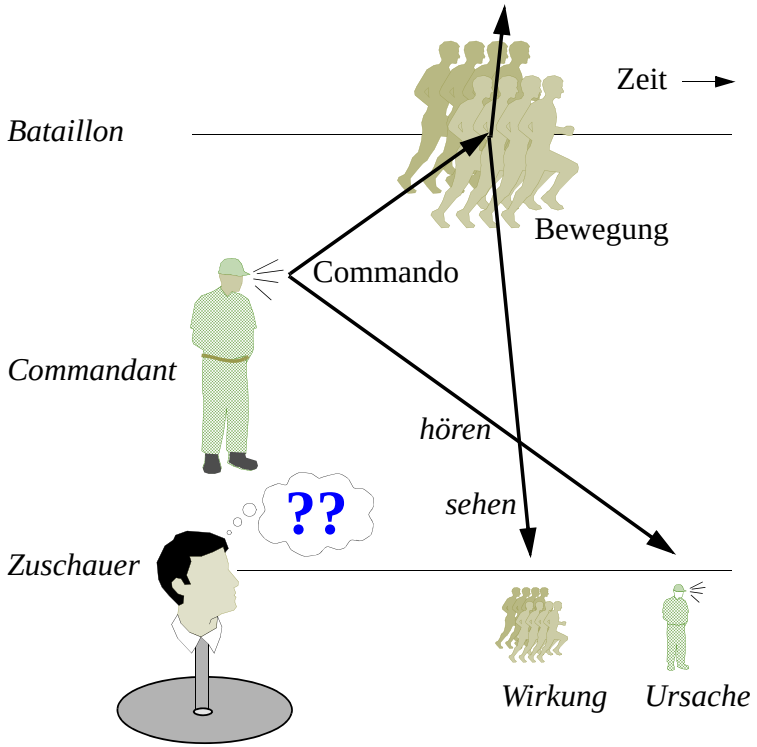
Kausalität

Das “Broadcastproblem” ist nicht neu

- Licht- und Schallwellen werden in natürlicher Weise per Broadcast verteilt
- Wann handelt es sich dabei um FIFO-Broadcasts?
- Wie ist es mit dem Kausalitätserhalt?

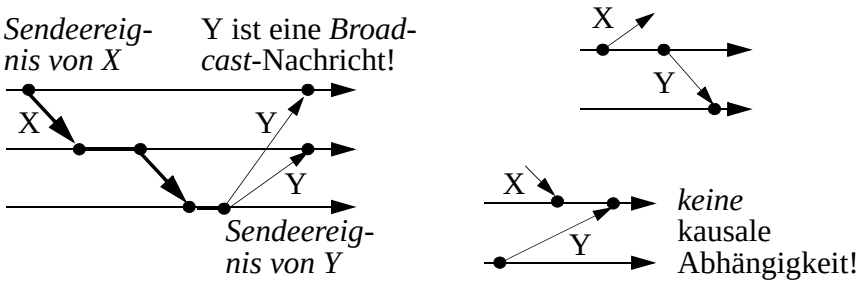
Wenn ein Zuschauer von der Ferne das Exercieren eines Bataillons verfolgt, so sieht er übereinstimmende Bewegungen desselben plötzlich eintreten, *ehe* er die Commandostimme oder das Hornsignal hört; aber aus seiner Kenntnis der *Causalzusammenhänge* weiß er, daß die Bewegungen die *Wirkung* des gehörten 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* (1889)



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")

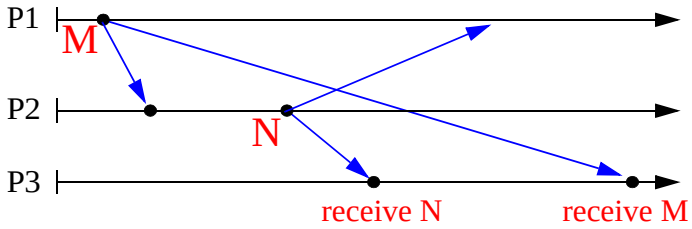
Kausaler Broadcast

Wahrung von Kausalität bei der Kommunikation:

- *Kausale Reihenfolge (Def.):* 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

"causal order"

- "Kausale Reihenfolge" (und "kausale Abhängigkeit") lassen sich insbes. auch auf *Broadcasts* anwenden:

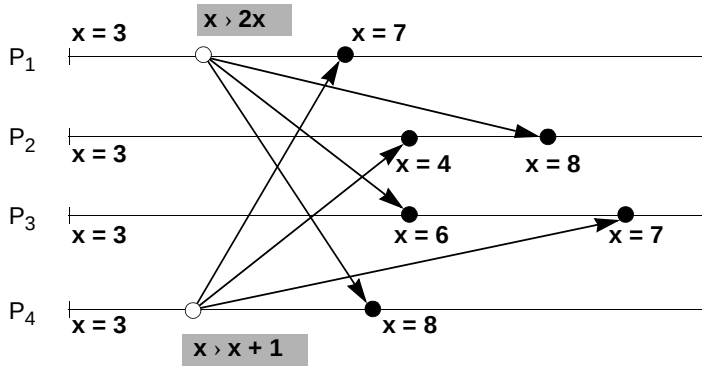


Gegenbeispiel: *Keine* kausalen Broadcasts

- 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:



Problem: Ergebnis statt *überall entweder 7 oder 8* hier nun “beides”!

Konkrete Ursache des Problems:

- Broadcasts werden nicht überall “gleichzeitig” empfangen
- dies führt lokal zu verschiedenen Empfangsreihenfolgen

Abstrakte Ursache:

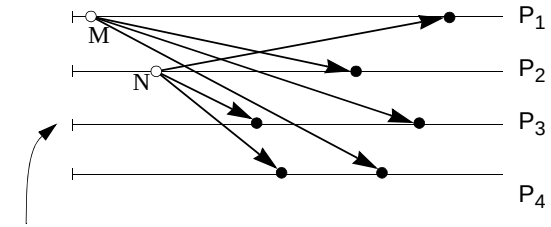
- die Nachrichtenübermittlung erfolgt (erkennbar!) *nicht atomar*

Also:

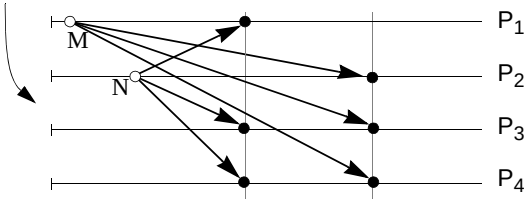
- auch kausale Broadcasts haben keine “perfekte” Semantik (d.h. Illusion einer speicherbasierten Kommunikation)

Atomarer bzw. “totaler” Broadcast

- *Totale Ordnung:* 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
  - das Senden wird dabei *nicht* als Empfang der Nachricht beim Sender selbst gewertet!
- Beachte: “Atomar” heisst hier *nicht* “alles oder nichts” (wie etwa beim Transaktionsbegriff von Datenbanken!)



Äquivalent bzgl. “Gummiband-Transformation”



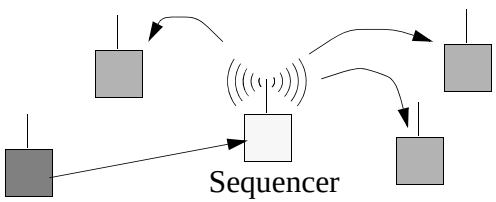
Anschaulich:

- Nachrichten eines Broadcasts werden “überall gleichzeitig” empfangen

Realisierung von atomarem Broadcast

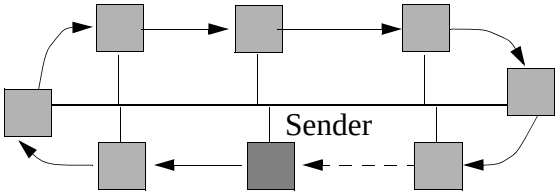
1) Zentraler „Sequencer“, der Reihenfolge festlegt

- ist allerdings ein potentieller Engpass!



- “Unicast” vom Sender zum Sequencer
- Multicast vom Sequencer an alle
- Sequencer *wartet* jew. auf alle Acknowledgements (oder genügt hierfür FIFO-Broadcast?)

2) Token, das auf einem (logischen) Ring kreist



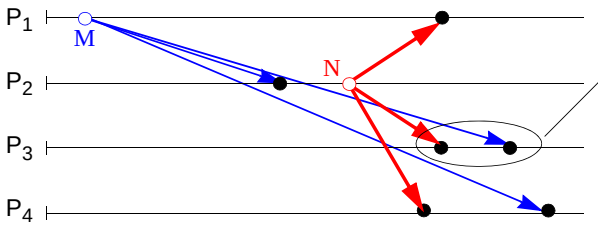
- Token = Senderecht (Token weitergeben!)
- Broadcast selbst könnte z.B. über ein zugrundeliegendes broadcastfähiges Medium erfolgen

- Token führt eine Sequenznummer (inkrementiert beim Senden), dadurch sind alle Broadcasts *global numeriert*
- Empfänger wissen, dass Nachrichten entsprechend der (in den Nachrichten mitgeführten Nummer) ausgeliefert werden müssen
- bei Lücken in den Nummern: dem Token einen Wiederholungswunsch mitgeben (Sender erhält damit implizit ein Acknowledgement)
- Tokenverlust (z.B. durch Prozessor-Crash) durch Timeouts feststellen (Vorsicht: Token dabei nicht versehentlich verdoppeln!)
- einen gecrashten Prozessor (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

Denkübung: Geht es auch ohne zentrale Elemente (Sequencer, Token)?

Wie “gut” ist atomarer Broadcast?

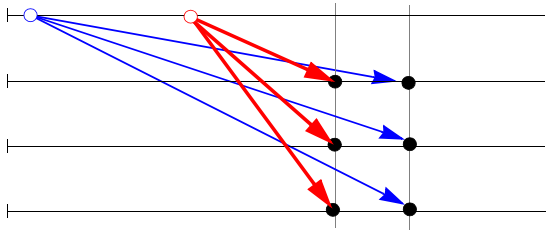
1) Ist atomar auch kausal?



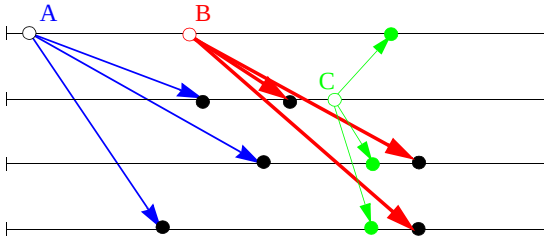
Nicht kausal!

Atomar: P3 und P4 empfangen beide M, N und zwar in gleicher Reihenfolge

2) Ist atomar wenigstens FIFO?



3) Ist atomar + FIFO vielleicht kausal?



Kausaler atomarer Broadcast

- Fazit:
  - atomare Übermittlung $\not\Rightarrow$ kausale Reihenfolge
  - atomare Übermittlung $\not\Rightarrow$ FIFO-Reihenfolge
  - atomare Übermittlung + FIFO $\not\Rightarrow$ 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 - wenn sie hinschauen)
  - Kommunikation über gemeinsamen Speicher *wahrt Kausalität* (die Wirkung tritt unmittelbar mit der Ursache, dem Schreibereignis, ein)

- Vergleichbares Kommunikationsmodell per Nachrichten: *Kausaler atomarer Broadcast*
  - kausaler Broadcast + totale Ordnung
  - man nennt daher kausale, atomare Übermittlung auch *virtuell synchrone Kommunikation*
  - Denkübung: realisieren die beiden Implementierungen “zentraler Sequencer” bzw. “Token auf Ring” die virtuell synchrone 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)

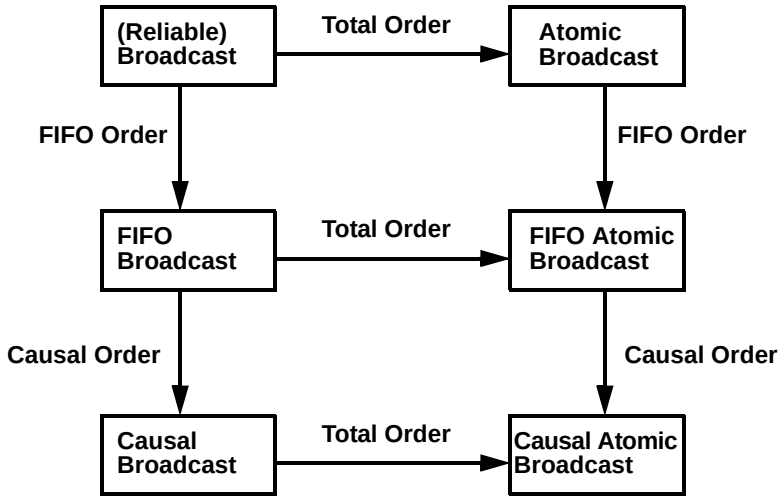
- *Innerhalb* des Systems ist synchron (im Sinne von “gleichzeitig”) und virtuell synchron *nicht unterscheidbar*
  - identische totale Ordnung aller Ereignisse
  - identische Kausalbeziehungen

- Folge: 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 ein einziger Broadcast, der alles kann?
  "Stärkere Semantik" hat auch Nachteile:
  - Performance-Einbussen
  - Verringerung der potentiellen Parallelität
  - aufwändiger zu implementieren

- Bekannte "Strategie":
  - man begnügt sich daher, falls es der Anwendungsfall 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!

$\Rightarrow$ grössere Vielfalt $\Rightarrow$ komplexer bzgl. Verständnis und Anwendung

Multicast

- Definition von Multicast (informell): "*Multicast ist ein Broadcast an eine Teilmenge von Prozessen*"
  - diese Teilmenge wird "*Multicast-Gruppe*" genannt

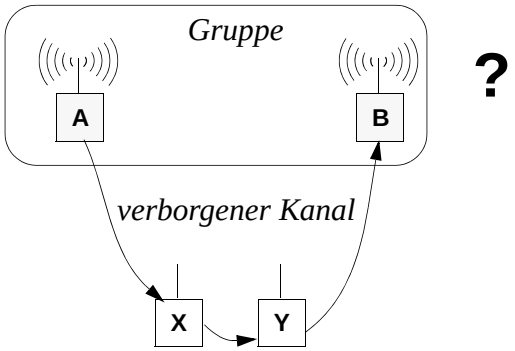
- Daher: Alles, was bisher über Broadcast gesagt wurde, gilt (innerhalb der Teilmenge) auch weiterhin:
  - zuverlässiger Multicast
  - FIFO-Multicast
  - kausaler Multicast
  - atomarer Multicast
  - kausaler atomarer Multicast

Unterschied: Wo bisher "alle Prozesse" gesagt wurde, gilt nun "alle Prozesse innerhalb der Teilmenge"

- Zweck Multicast-Gruppe
  - "Selektiver Broadcast"
  - Vereinfachung der Adressierung (z.B. statt Liste von Einzeladressen)
  - Verbergen der Gruppenzusammensetzung (vgl. Port-Konzept)
  - "Logischer Unicast": Gruppen ersetzen Individuen (z.B. für transparente Replikation)

Problem der “Hidden Channels”

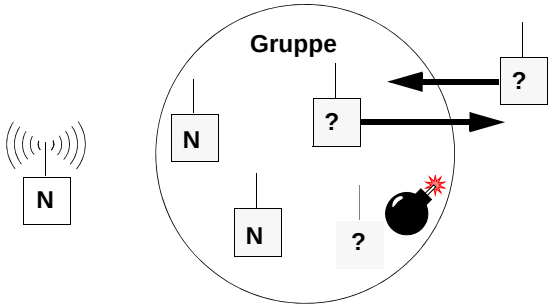
- Kausalitätsbezüge verlassen (z.B. durch Gruppenüberlappung) die Multicast-Gruppe und kehren später wieder



- 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

Gruppen-Management / -Membership



- Dynamische Gruppe: wie sieht die Gruppe “momentan” aus?

- Haben alle Mitglieder (gleichzeitig?) die gleiche Sicht?

- Was bedeutet “alle Gruppenmitglieder”?

  - *Beitritt* (“join”) zu einer Gruppe
  - *Austritt* (“leave”) aus einer Gruppe } *während* eines Multicasts?
  - *Crash:“korrekt”* → *“fehlerhaft”*

- Beachte:

  - “Zufälligkeiten” (z.B. Beitrittszeitpunkt kurz vor / nach dem Empfang einer Einzelnachricht) sollten (soweit möglich) vermieden werden (→ Nichtdeterminismus; Nicht-Reproduzierbarkeit)

- Folge:

  - Zu jedem Zeitpunkt soll *Übereinstimmung* über *Gruppenzusammensetzung* und *Fehlerzustand* (“korrekt”, “ausgefallen” etc.) aller Mitglieder erzielt werden

- Problem: Wie erzielt man diese Übereinkunft?

Wechsel der Gruppenmitgliedschaft

"...während..." gibt es nicht (→ "virtuell synchron")

- Forderungen:
  - Eintritt und Austritt sollen *atomar* erfolgen:
    - Die Gruppe muss bei allen (potentiellen) Sendern an die Gruppe hinsichtlich der Ein- und Austrittszeitpunkte jedes Gruppenmitglieds übereinstimmen
  - *Kausalität* soll gewahrt bleiben

- Realisierungsmöglichkeit:
  - konzeptuell führt jeder Prozess eine Liste mit den Namen aller Gruppenmitglieder
    - Realisierung als zentrale Liste (Fehlertoleranz und Performance?)
    - oder Realisierung als verteilte, replizierte Liste
  - massgeblich ist die zum Sendezeitpunkt gültige Mitgliederliste
  - Listenänderungen werden (virtuell) synchron durchgeführt:
    - bei einer zentralen Liste kein Problem
    - bei replizierten Listen: verwende *kausalen atomaren Multicast*

      Schwierigkeit: *Bootstrapping-Problem* (mögliche Lösung: Service-Multicast zur dezentralen Mitgliedslistenverwaltung löst dies für sich selbst über einen zentralen Server)

Behandlung von Prozessausfällen

- Forderungen:
  - *Ausfall* eines Prozesses soll *global atomar* erfolgen:
    - Übereinstimmung über Ausfallzeitpunkt jedes Gruppenmitglieds
  - *Reintegration* nach einem vorübergehenden Ausfall soll *atomar* erfolgen:
    - Übereinstimmung über Reintegrationszeitpunkt

- Realisierungsmöglichkeit:
  - Ausfallzeitpunkt:
    - Prozesse dürfen nur Fail-Stop-Verhalten zeigen: *"Einmal tot, immer tot"*
    - Gruppenmitglieder erklären Opfer per kausalem, atomarem Multicast übereinstimmend für tot: *"Ich sage tot – alle sagen tot!"*
    - Beachte: *"Lebendiges Begraben"* ist nicht ausschliessbar! (Irrtum eines "failure suspectors" aufgrund zu langsamer Nachrichten)
    - Fälschlich für tot erklärte Prozesse sollten unverzüglich Selbstmord begehen
  - Reintegration:
    - Jeder tote (bzw. für tot erklärte) Prozess kann der Gruppe nur nach dem offiziellen Verfahren ("Neuaufnahme") wieder beitreten

- Damit erfolgen Wechsel der Gruppenmitgliedschaft und Crashes in "geordneter Weise" für *alle* Teilnehmer