EMES: Eigenschaften mobiler und eingebetteter Systeme

Gruppenkommunikation

Dr. Siegmar Sommer, Dr. Peter Tröger

Wintersemester 2009/2010

Motivation

- Häufig ist eine Aufgabe von einer Gruppe zu erledigen
- Gruppenmitglieder: Rechner, Prozesse
- Anwendung:
  - Fehlertoleranz
  - Client-Server-Anwendungen
  - ⋮
- Beispiele für Gruppen in EMES:
  - Prozessoren/verteilte Prozesse eines eingebetteten Systems
  - Anzahl mobiler Geräte in einer drahtlosen Umgebung (Handys, Verkehrsüberwachung)

Probleme

- Konsistente Sicht (alle Mitglieder sehen die Welt in gleicher Weise)
- Gruppenzustand kann sich ändern (ausscheidende und neue Mitglieder)
- Gruppenmitglieder können fehlerhaft sein

Gemeinsame Weltsicht

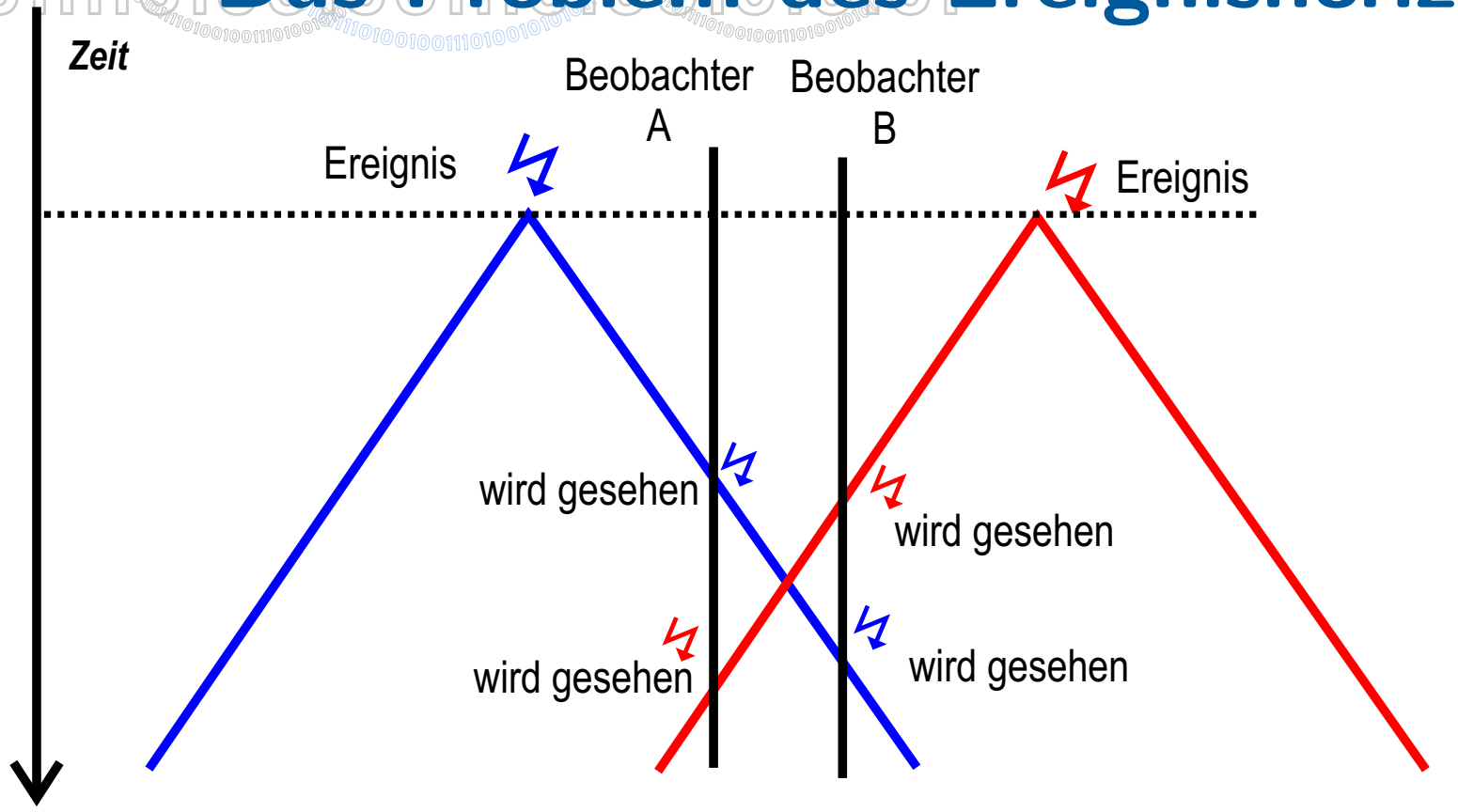
- Gemeinsame Weltsicht wird meist so verstanden:
  Zum gleichen Zeitpunkt werden die gleichen Ereignisse wahrgenommen.

- Diese Art von gemeinsamer Weltsicht ist i.a. nicht möglich:
  - Es gibt keine gemeinsame Zeit
  - Durch räumliche Trennung gibt es unterschiedliche Ereignishorizonte

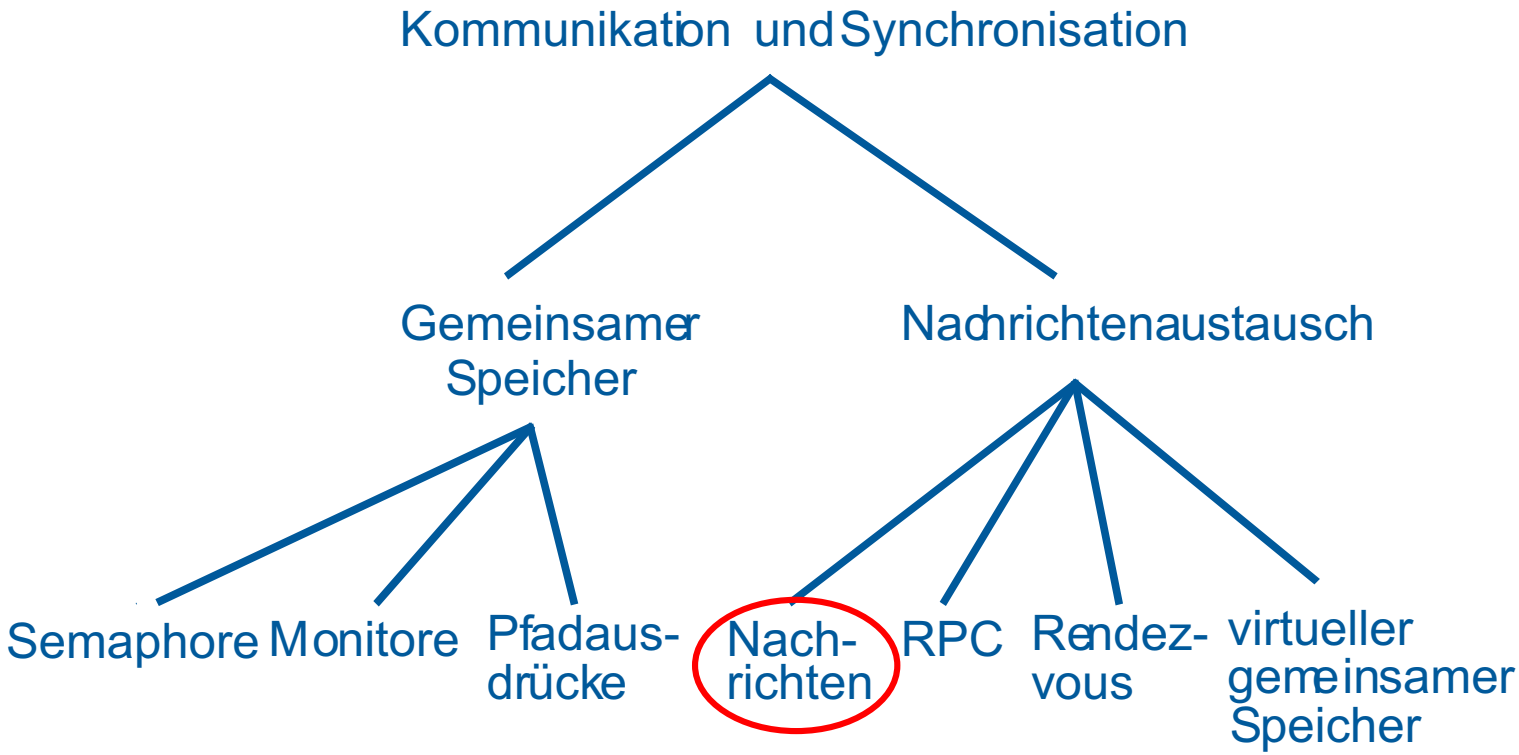
- Problem 1: Uhrensynchronisation (nächste Vorlesung)

- Problem 2: Virtuelle Zeiten (Ordnung von Ereignissen)

Das Problem des Ereignishorizontes



Ansätze zur Kommunikation

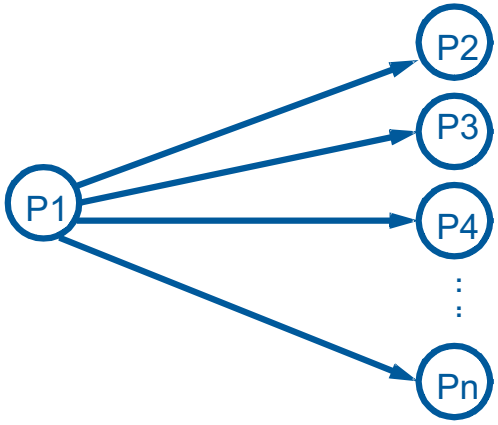


Modell

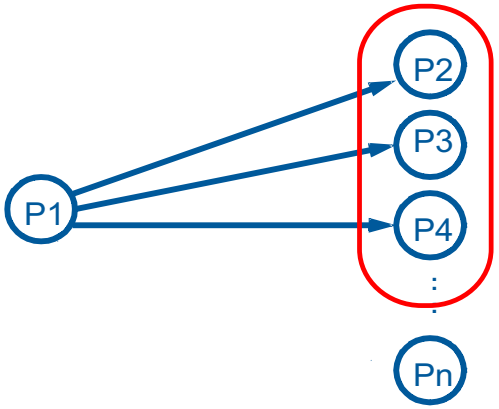
- Gruppenmitglieder kommunizieren ausschließlich über Nachrichten miteinander
- Ereignisse sind entweder das Senden/Empfangen von Nachrichten, oder mitgliedsintern
- Mitglieder können lokale Uhren besitzen; es existiert aber keine globale Uhr
- Über die Dauer einer Nachrichtenübertragung wird (zunächst) keine Annahme getroffen, aber
  - Nachricht wird **immer vor** ihrem Empfang gesendet
  - verschiedene Anforderungen an Konsistenz (Nachrichtenordnung) im folgenden diskutiert

Kommunikationsmuster

1-zu-1-Kommunikation

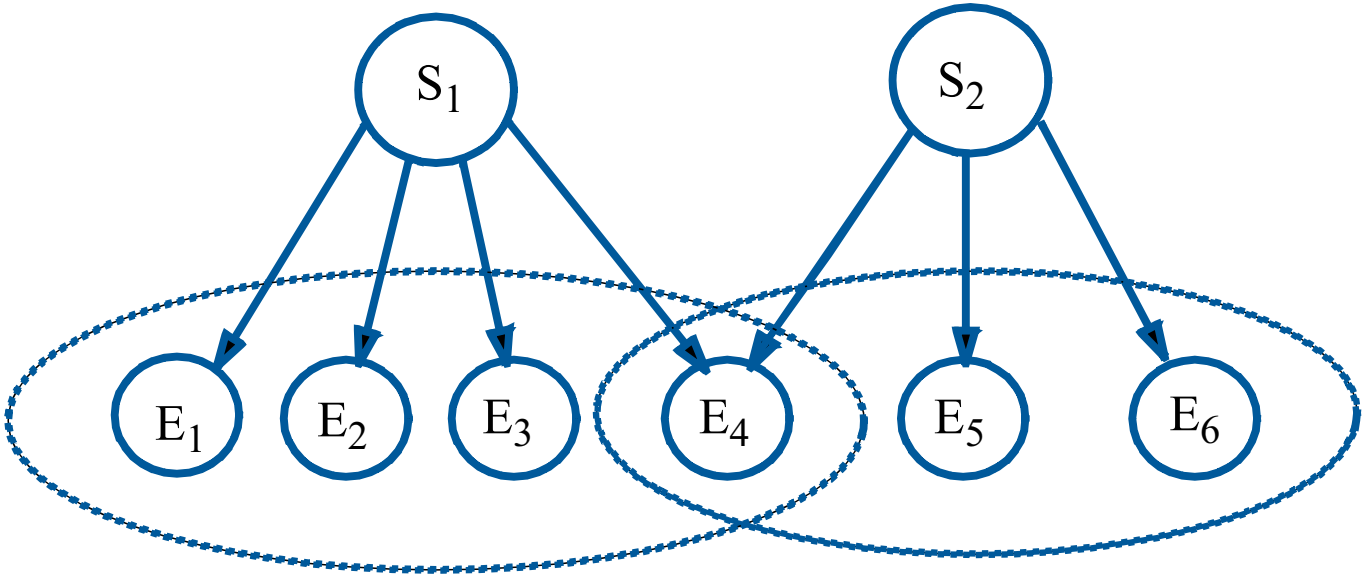


Broadcast (1 zu alle)



Multicast (1 zu mehrere)

Anforderungen in Kommunikationsgruppen



- Gruppen können kreiert und gelöscht werden
- Gruppenmitgliedschaft ist dynamisch
- Gruppen können sich überlappen

Typische Gruppenfunktionen

Create Group Erzeugen einer Gruppe

Join Group Hinzufügen eines Gruppenmitgliedes

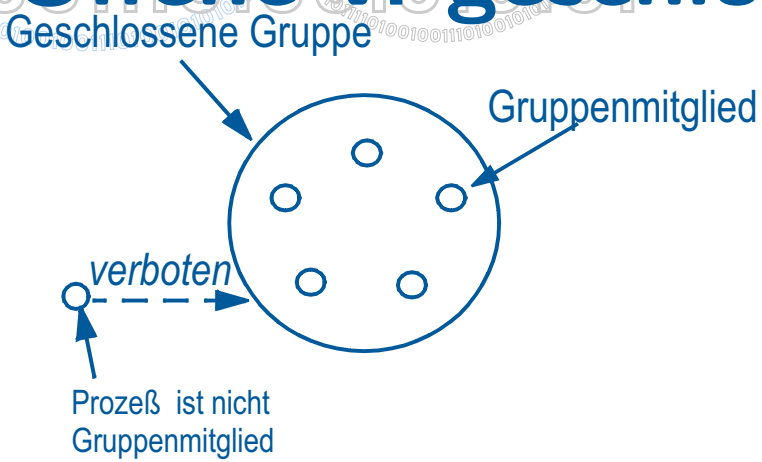
Leave Group Entfernen eines Gruppenmitgliedes

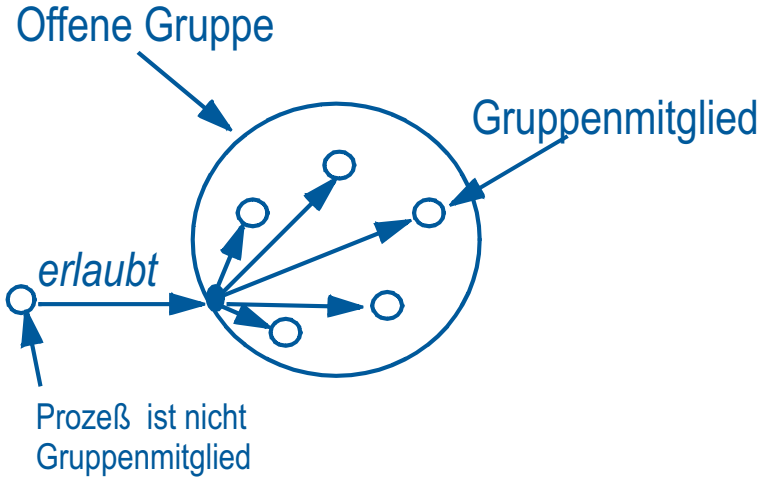
Destroy Group Explizites Löschen einer Gruppe

SendTo Group Multicast an eine Gruppe

ReceiveFrom Group Empfangen einer Nachricht

Offene v. geschlossene Gruppen





Synchronität und Auslieferung

- Synchronität von Senden und Empfangen
  - synchron: nur zu bestimmten Zeiten möglich → Blockierung
  - asynchron: jederzeit möglich
- Hier wird Asynchronität der Kommunikation angenommen
- Zwischenspeicherung von Nachrichten (Pufferung), bis sie an den eigentlichen Prozeß *ausgeliefert* werden (deliver())

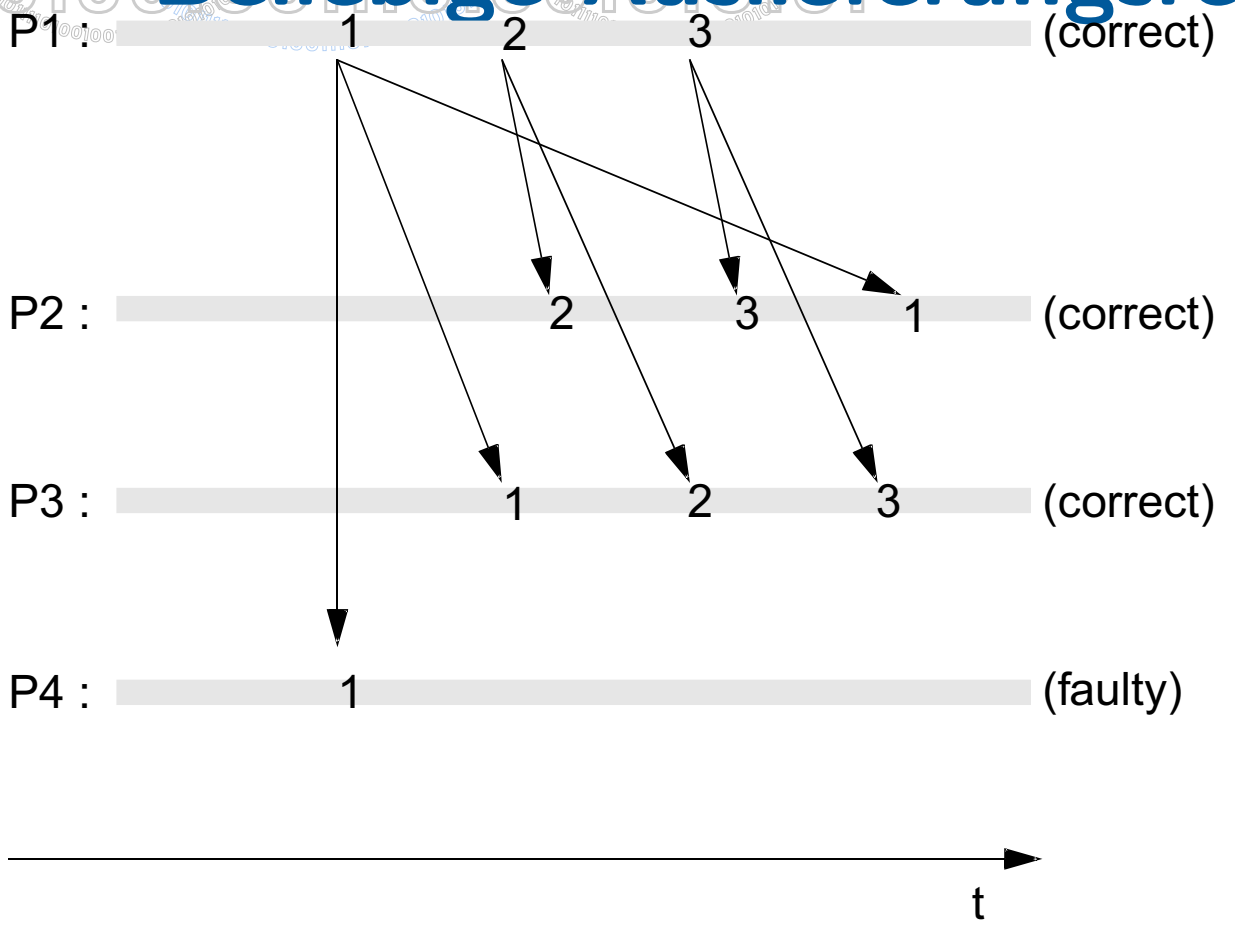
Zuverlässiger Multicast

- Gültigkeit (*validity*)
  Führt ein korrekter Prozeß multicast(msg) für eine Gruppe g aus, so führt in Folge jeder korrekte Prozeß deliver(msg) durch.

- Zustimmung (*Agreement*)
  Führt ein korrekter Prozeß einer Gruppe deliver(msg) durch, so führen irgendwann alle korrekten Prozesse der Gruppe deliver(msg) durch.

- Integrität (*integrity*)
  Für jede Nachricht msg gilt, daß sie jeder korrekte Prozeß höchstens einmal ausliefert und das auch nur, wenn irgendein Prozeß multicast(msg) durchgeführt hat.

Beliebige Auslieferungsreihenfolge



Die Bedingungen des zuverlässigen Multicast verlangen keine bestimmte Reihenfolge der Auslieferung

Mögliche Ordnungsanforderungen

- FIFO (*first in first out*)
- kausale Ordnung
- totale Ordnung
- totale FIFO-Ordnung
- totale Kausalordnung
- globale Ordnung

FIFO

FIFO-Ordnung Wenn ein korrekter Prozeß $\text{multicast}(m_1)$ vor $\text{multicast}(m_2)$ ausführt, dann führt kein korrekter Prozeß $\text{deliver}(m_2)$ durch, bevor er nicht m_1 ausgeliefert hat.

- Idee: Nachricht wird in der Regel in einem Kontext abgearbeitet
- Eine per Multicast gesendete Nachricht sollte bei jedem Empfänger den gleichen Kontext (senderbezogen) vorfinden
- Beispiel: File-Öffnen *vor* File-Schreiben

Kausale Ordnung (I)

- FIFO-Ordnung bezieht sich nur auf *einen* Sender
- Aber: Ein Empfänger kann aufgrund der empfangenen Nachricht m_1 selbst wieder eine Nachricht m_2 senden, die bei einem dritten Mitglied *vor* m_1 ausgeliefert wird.
- Bei komplexeren Kontexten (Abhängigkeit von mehreren Sendern) nicht erwünscht

Happens-before-Relation

Zwei Ereignisse e_1 und e_2 erfüllen die Happens-before-Relation ($e_1 \prec e_2$), wenn folgendes gilt:

- e_1 und e_2 treten bei einem Prozeß genau in dieser Reihenfolge auf, oder
- e_1 ist $\mathrm{multicast}$ einer Nachricht und e_2 ihre Auslieferung, oder
- es gibt ein Ereignis e', so daß $e_1 \prec e'$ und $e' \prec e_2$ gilt

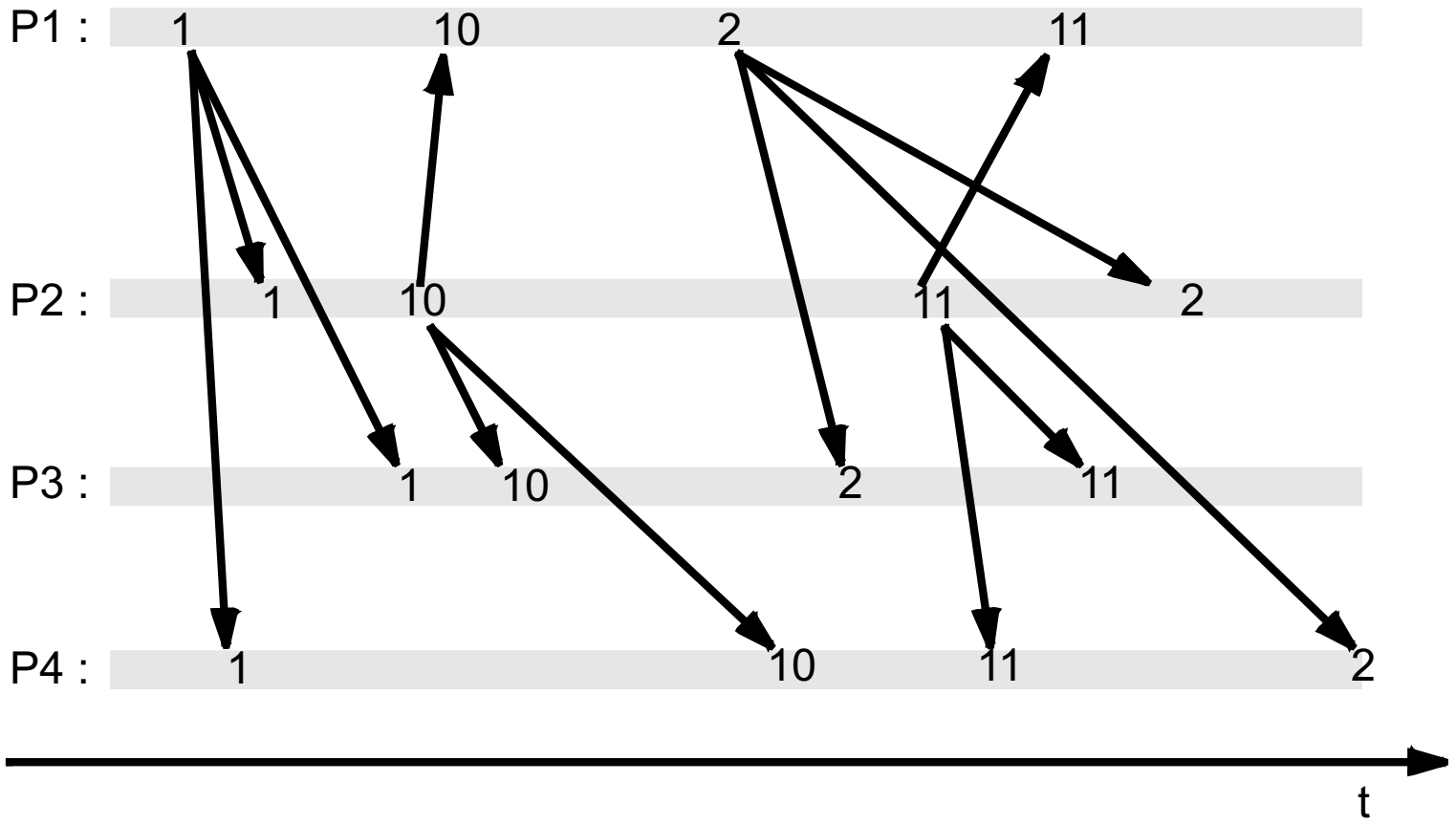
Man sagt dann: e_2 ist von e_1 *kausal abhängig*

Kausale Ordnung (II)

Kausale Ordnung Gilt $\text{multicast}(m_1) \prec \text{multicast}(m_2)$, dann führt kein korrekter Prozeß $\text{deliver}(m_2)$ durch, ehe er m_1 ausgeliefert hat.

- Kausale Ordnung ist stärker als FIFO:
  - Jede kausale Ordnung ist auch FIFO geordnet
  - Umkehrung gilt nicht
- Kausalität ist hier hypothetisch:
  - Jede mögliche Kausalität wird bewahrt
  - Es ist möglich, daß kausal geordnete Ereignisse gar nicht voneinander abhängig sind

Beispiel für kausale Ordnung

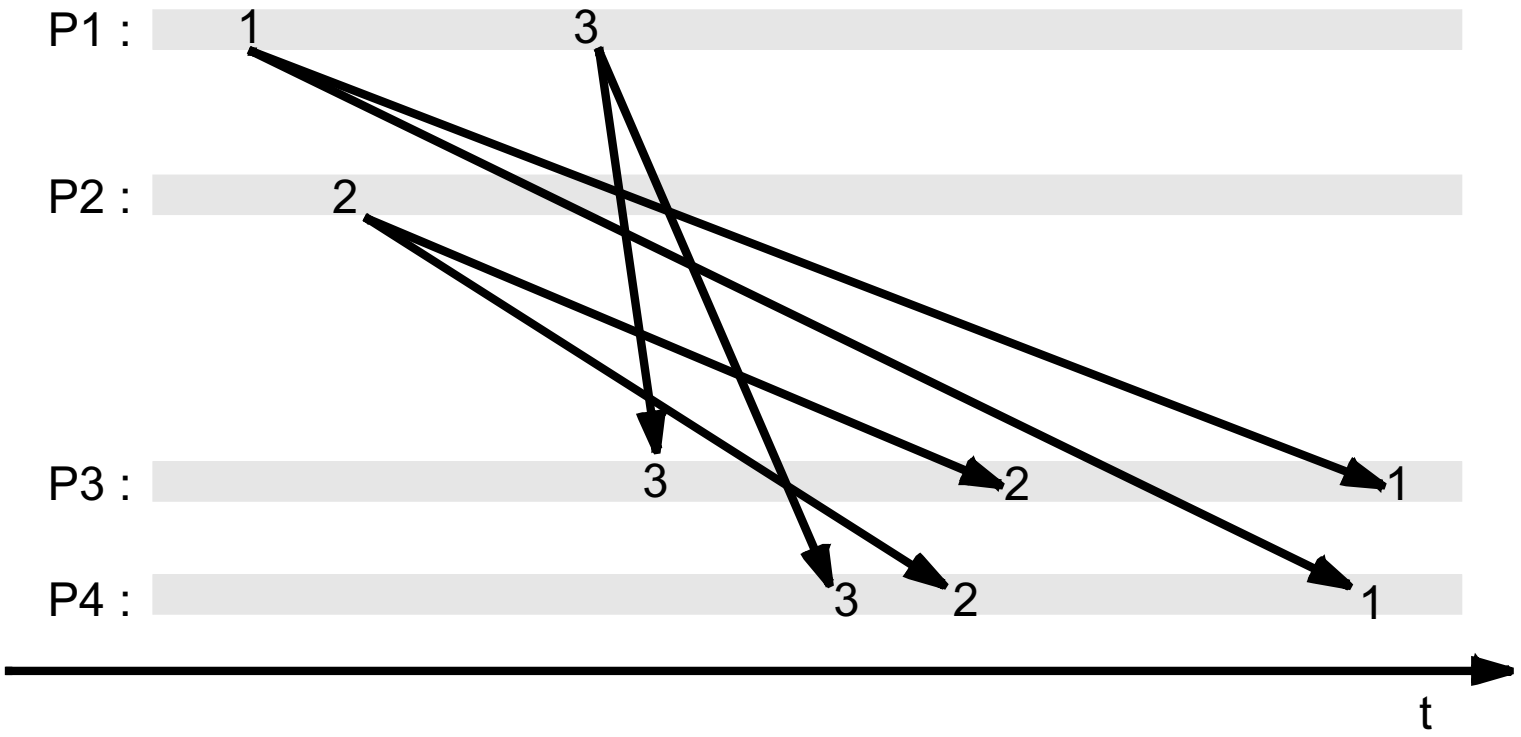


Totale Ordnung

Totale Ordnung Wenn die beiden korrekten Prozesse P und Q die Nachrichten m_1 und m_2 ausliefern, dann liefert P die Nachricht m_1 genau dann vor m_2 aus, wenn Q dies auch tut.

- Mit anderen Worten: korrekte Prozesse liefern die gleiche Sequenz von Nachrichten aus
- **Beachte:** Es wird keine Aussage über die Sequenz an sich gemacht. Insbesondere kann die Kausalität verletzt sein
- Ein Broadcast mit totaler Ordnung wird manchmal auch *atomarer* Broadcast genannt

Beispiel für totale Ordnung

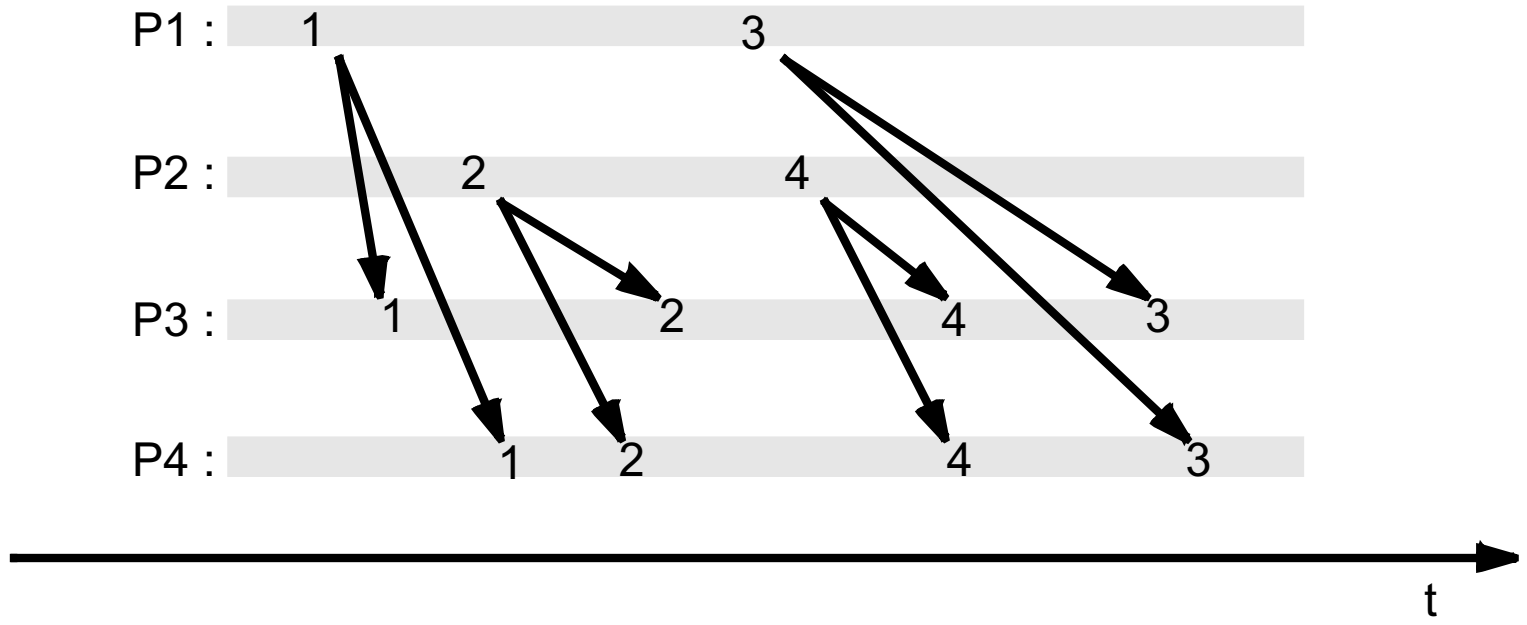


Totale FIFO- und Kausalordnung

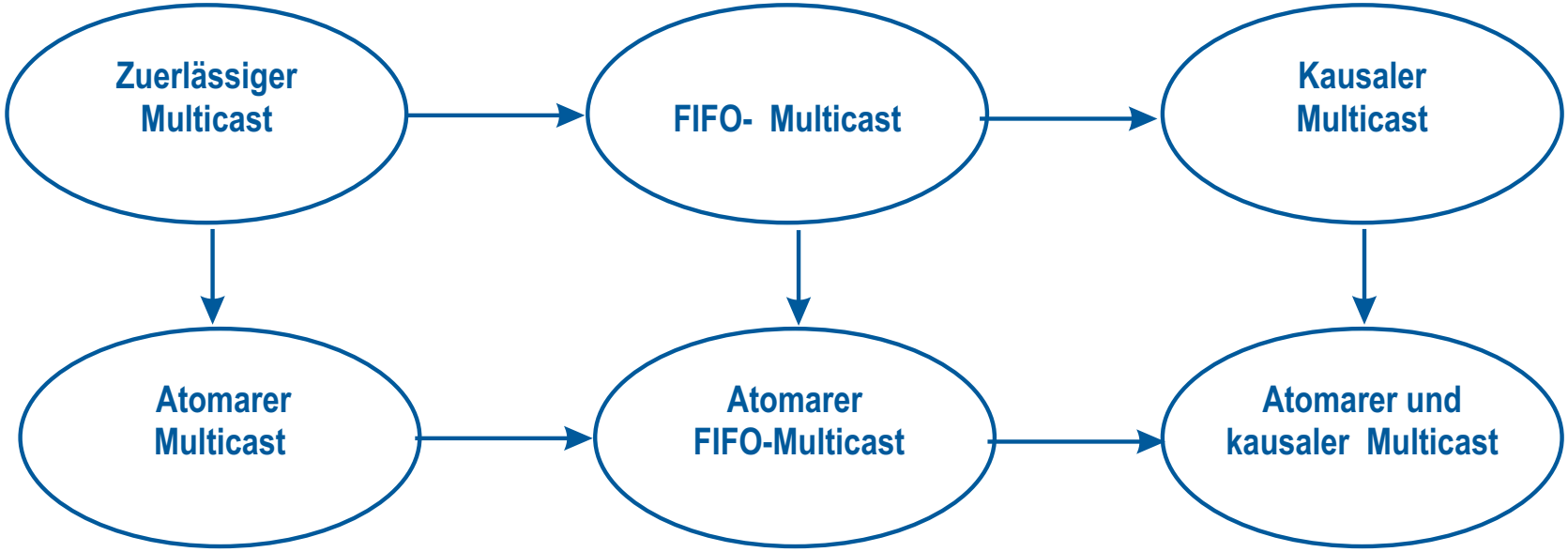
Totale FIFO-Ordnung Eine Ordnung, die sowohl first-in-first-out als auch total geordnet ist.

Totale Kausalordnung Eine Ordnung, die sowohl kausal als auch total geordnet ist.

Beispiel für totale FIFO-Ordnung



Ordnung der Ordnungen



Beispiel: ISIS (I)

- Programmier-Toolkit, entwickelt an der Cornell Universität (Ithaka, USA)
- Ursprünglich als UNIX-Bibliothek implementiert, später für Mach und Chorus
- ISIS bietet an:
  - Multicast-Protokolle
  - Erhaltung der Gruppensicht
  - Zustandsübertragung
- Die Multicast-Protokolle sind:
  - FBCAST (ungeordneter Multicast)
  - CBCAST (kausal geordneter Multicast)
  - ABCAST (total geordneter Multicast)
  - GBCAST (global synchronisationsgeordneter Multicast)

Beispiel: ISIS (II)

- Als Kommunikationsgrundlage wird UDP/IP benutzt
- Nachrichten werden bestätigt (*acknowledged*) und gegebenenfalls erneut gesendet $\Rightarrow$ zuverlässige Übertragung
- Wenn IP-Multicast vorhanden ist (z.B. Ethernet), dann wird es genutzt, sonst Punkt-zu-Punkt-Kommunikation
- Ordnung(en) werden mit Hilfe von Ereigniszählervektoren (Zeitstempelvektoren) erzielt:
  - Jeder Prozeß P_i $(i = 1...n)$ besitzt einen Vektor $V_i = \{V_i[1], ..., V_n[n]$
  - Alle Prozesse starten mit einem Nullvektor

ISIS: CBCAST-Protokoll

- Update eines Nachrichtenvektors:
  1. Wenn P_i eine neue Nachricht senden will, wird zunächst $V_i[i]$ inkrementiert und dann V_i mit der Nachricht mitgesendet
  2. Wenn ein Prozeß P_j eine Nachricht von P_i mit dem Vektor V_i empfängt, so wird sein Vektor V_j wie folgt aktualisiert:
     $\forall k = 1...n : P_j[k] = \max(P_j[k], P_i[k])$
- Auslieferung einer Nachricht von P_i bei P_j
  - Lokale Nachrichten werden sofort ausgeliefert (d.h. $i = j$)
  - Alle anderen Nachrichten werden solange zurückgehalten, bis:
    1. Die Nachricht muß die nächste in einer Sequenz von P_i sein, d.h. $V_i[i] = V_j[i] + 1$
    2. Alle kausal vorhergehenden Nachrichten die an P_i gesendet wurden, sollten bei P_j ausgeliefert sein, d.h. $V_j[k] \geq V_i[k]$ für $k \neq i$

ISIS: ABCAST-Protokoll (I)

- Idee: Zentraler Sequenzer
- ABCAST nutzt CBCAST-Nachrichten, die aber als ABCAST-Nachrichten gekennzeichnet sind
- In der Gruppe wird ein Token-Halter gewählt
  - Der Token-Halter empfängt ABCAST-Nachrichten und liefert sie bei sich kausal geordnet aus
  - Von Token-Halter gesendete Nachrichten enthalten die Ordnungsnummer der Nachricht innerhalb der totalen Ordnung
  - Von Zeit zu Zeit sendet der Token-Halter eine sogenannte *sets-order* Nachricht, die die Sequenz-Nummer für eine oder mehrere von ihm empfangene ABCAST-Nachrichten enthält

ISIS: ABCAST-Protokoll (II)

- Alle anderen Mitglieder verzögern ABCAST-Nachrichten, bis:
  - Sie die entsprechende Sets-order-Nachricht empfangen haben
  - Sie alle ABCAST-Nachrichten, auf die in der Set-order-Nachricht bezug genommen wird, empfangen haben
  - Sie alle kausal vorhergehenden CBCAST-Nachrichten ausgeliefert haben

- Das Token kann weitergegeben werden:
  - Vorteil: Wenn nur Token-Halter ABCAST-Nachrichten versenden, sind keine Sets-order-Nachrichten notwendig

Beispiel: TOTEM

- Entwickelt an der University of California at Santa Barbara (USA)
- TOTEM = *to*tal ordered and *tem*poral predictability (außerdem: Protokollstack hat Ähnlichkeit mit Totempfahl)
- Bietet zwei zuverlässige Multicastdienste mit totaler Ordnung
  - Agreed delivery
  - Safe delivery
- Im Gegensatz zu ISIS hält TOTEM die gewünschten Ordnungseigenschaften auch bei Gruppenpartitionierung und Verschmelzung bei
- Die Autoren behaupten, daß TOTEM echtzeitfähig ist

Agreed und safe delivery

Agreed delivery garantiert, daß eine Nachricht nur ausgeliefert wird, wenn alle vorhergehenden Nachrichten (in dieser Konfiguration) bereits ausgeliefert wurden

Safe delivery garantiert darüber hinaus, daß diese Nachricht von jedem anderen Prozeß empfangen wurde, ehe sie lokal ausgeliefert wird.

- Safe delivery ist z.B. in Transaktionssystemen nützlich, in denen eine Transaktion von allen oder keinen Prozeß ausgeführt werden muß

TOTEM und Echtzeit

- Bedingt echtzeitfähig im fehlerfreien Fall
- Bei Vorhandensein von Fehlern nur stochastische Zeitgarantien