

Verteilte Betriebssysteme

5. Kapitel

Gruppenkommunikation und Ordnungen

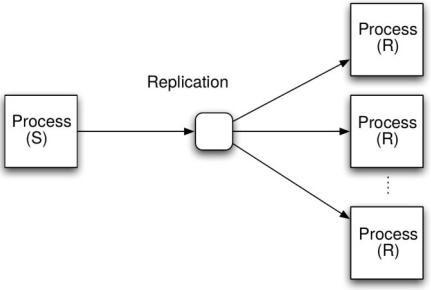
Matthias Werner

Professur Betriebssysteme

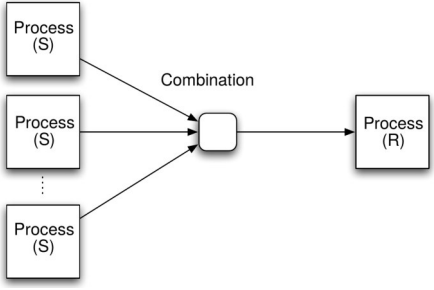
5.1 Einführung

- In verteilten Systemen sind an der Kommunikation häufig mehrere Partner beteiligt ➡ **Gruppenkommunikation**

- **Multicast**: Nachricht geht an mehrere



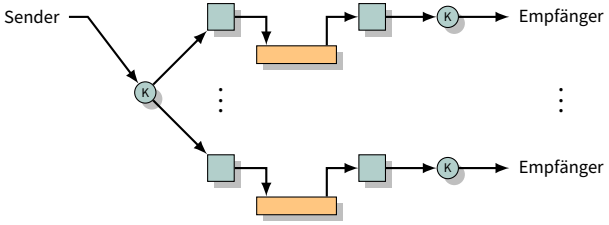
- **Combine**: Mehrere Nachrichten werden kombiniert



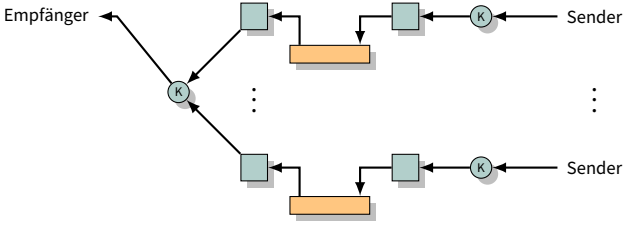
- Betrachten hier insbesondere Multicast

Realisierung durch Brücken

- **Multicast**: Empfänger auf mehreren Rechnern

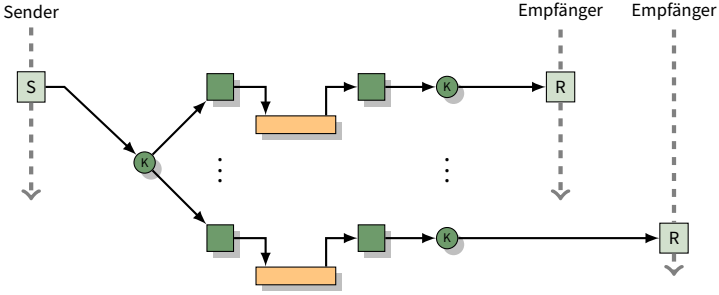


- **Combine**: Sender auf mehreren Rechnern



Semantische Varianten (bei Multicast)

- Die Verteiltheit des Kanals macht es schwierig, die Semantik des lokalen Falls zu erhalten
- Es eröffnen sich weitere semantische Varianten, welche
  - die **Zuverlässigkeit** der Auslieferung
    und
  - die **Synchronisation** der beteiligten Prozesse betreffen



5.2 Zuverlässigkeit der Zustellung

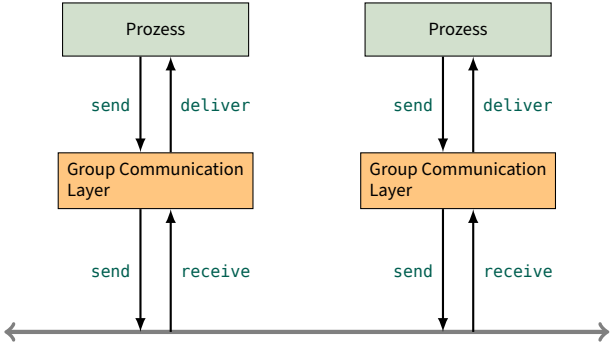
- **Unzuverlässiger Multicast**
  Es werden keinerlei Zusagen über den Erfolg der Zustellung gemacht
- **Zuverlässiger Multicast**
  Die Nachricht wird an **mindestens** einen Empfänger ausgeliefert, aber nicht notwendigerweise an alle
- **Atomarer Multicast**
  Die Nachricht wird entweder an **alle** Gruppenmitglieder erfolgreich ausgeliefert **oder** an **keinen**

Achtung

Mitunter sind die Definitionen in der Literatur anders. So definiert z.B. [CDK02 Kapitel 11] den zuverlässigen Broadcast als atomar, dafür gibt es dort einen Basis-Broadcast.

Zurückhalten beim Ersatzobjekt

- Um Atomarität (und Ordnung) beim Multicast zu erhalten, wird zwischen **Empfang** bei der empfängerseitigen Ersatzinstanz und der **Auslieferung** an den Empfänger unterschieden



Diskussion

Die Garantie der Atomarität setzt bestimmte Annahmen voraus, z.B.:

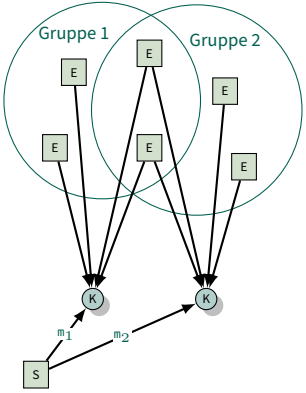
- Fehler treten ausschließlich in der Kommunikation vor Erreichen des Knotens auf.
- Prozess und Group Communication Layer fallen stets gemeinsam aus.

Synchronisation

- Der atomare Multicast impliziert (bei synchronem Empfang) eine **Barrierensynchronisation**
- Auslieferung der Nachrichten auf den Zielseiten und Deblockieren der Empfänger erst, wenn
  1. überall die Nachrichten eingetroffen
     und
  2. alle Empfänger die Auslieferungsoperation gerufen haben
- Ansonsten kann ein Deblockieren stattfinden, wenn die Nachricht beim jeweiligen Empfänger angekommen ist

5.3 Ordnung

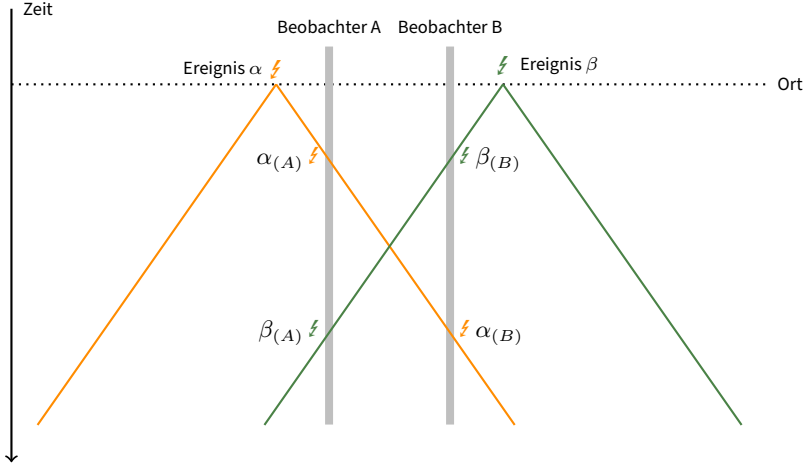
- Gegeben seien zwei überlappende Empfängergruppen $\mathbf{G}_1$ und $\mathbf{G}_2$, an die nacheinander jeweils eine Nachricht gesendet wird



- **Frage**:
  In welcher Reihenfolge kommen die Nachrichten bei den Empfängern an?
- Frage lässt sich verallgemeinern

Ereignishorizont

- Es gibt nicht notwendiger Weise eine „richtige“ Reihenfolge

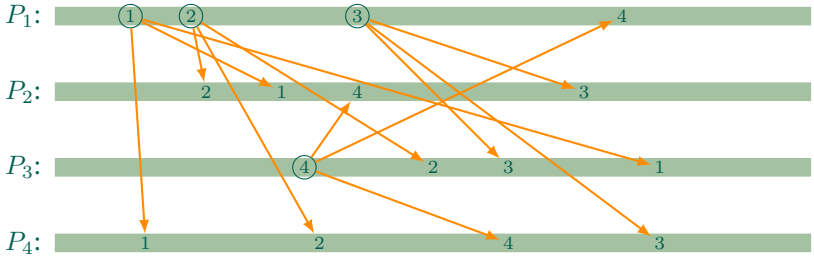


Mögliche Ordnungsanforderungen

- Es gibt verschiedene mögliche Anforderungen an die Auslieferungsreihenfolge
  - Ungeordnete Auslieferung
  - FIFO (first in first out)
  - Kausale Ordnung
  - Vollständig sortierte (Auslieferungs-)Ordnung
  - Vollständig sortierte FIFO-Ordnung
  - Vollständig sortierte Kausalordnung (mitunter als globale Ordnung bezeichnet)

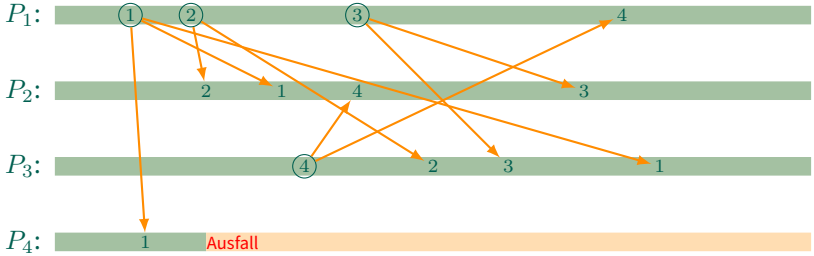
Beliebige Auslieferungsreihenfolge

- Der atomare Multicast allein verlangt keine bestimmte Auslieferungsreihenfolge
- Lediglich die Atomarität muss (für fehlerfreie Prozesse) gewährleistet sein



Beliebige Auslieferungsreihenfolge

- Der atomare Multicast allein verlangt keine bestimmte Auslieferungsreihenfolge
- Lediglich die Atomarität muss (für fehlerfreie Prozesse) gewährleistet sein



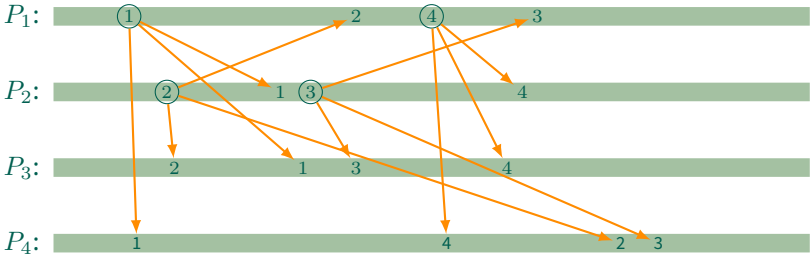
FIFO-Ordnung

- **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

FIFO-Ordnung

Wenn ein korrektes Mitglied `multicast(`m_1`)` vor `multicast(`m_2`)` ausführt, dann führt kein korrektes Mitglied `deliver(`m_2`)` durch, bevor es nicht m_1 ausgeliefert hat.

FIFO-Ordnung (Forts.)



Kausale Ordnung

- FIFO-Ordnung bezieht sich nur auf einzelnen Sender
- **Aber:**
  Bei komplexeren Kontexten (Abhängigkeit von mehreren Sendern) nicht hinreichend
- **Beispiel:** Ein Empfänger kann aufgrund der empfangenen Nachricht $\mathtt{m}_1$ selbst wieder eine Nachricht $\mathtt{m}_2$ senden, die bei einem dritten Mitglied (bei FIFO) vor $\mathtt{m}_1$ ausgeliefert wird
- Um solche Fälle beschreiben zu können, wird (potentielle) Kausalität über die **Happened-before-Relation** beschrieben

Happened-before-Relation

- Zwei Ereignisse e_1 und e_2 erfüllen die **Happened-before-Relation** (auch: „happens before“ oder „happen before“), bezeichnet mit $e_1 \prec e_2$, wenn Folgendes gilt:
  1. e_1 und e_2 treten bei einem Gruppenmitglied (d.h. lokal) genau in dieser Reihenfolge auf; oder
  2. e_1 ist ein `multicast`-Ereignis einer Nachricht und e_2 die zugehörige Auslieferung; oder
  3. Es gibt ein Ereignis e_0, so dass $e_1 \prec e_0$ und $e_0 \prec e_2$
- **Sprachregelung**: e_2 ist von e_1 **kausal abhängig**
- Alternative Schreibweise: $e_1 \rightarrow e_2$

Happened-before-Relation (Forts.)

- Die Happened-before-Relation ist:
  - **Transitiv:** $(a \prec b) \wedge (b \prec c) \Rightarrow a \prec c$
  - **Irreflexiv:** Es gibt kein a mit $a \prec a$
  - **Asymmetrisch:** $(a \prec b) \Rightarrow \neg(b \prec a)$

- Zwei Ereignisse a und b, $a \neq b$ sind voneinander **kausal unabhängig**, wenn weder $a \prec b$ noch $b \prec a$ gilt
  - Man schreibt: $a \parallel b$
  - Die Relation „$\parallel$“ ist reflexiv, aber nicht transitiv

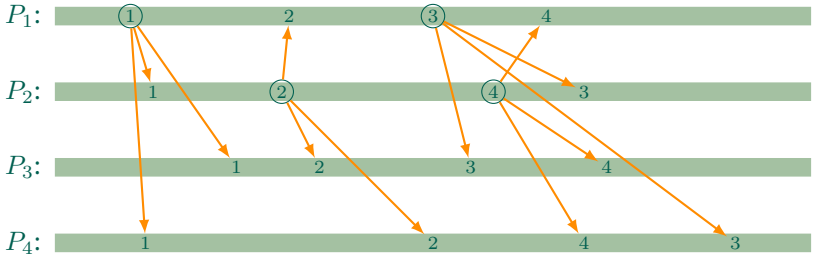
Kausale Ordnung

Kausale Ordnung

Gilt $\texttt{multicast}(m_1) \prec \texttt{multicast}(m_2)$, dann führt kein korrektes Gruppenmitglied $\texttt{deliver}(m_2)$, ehe es m_1 ausgeliefert hat.

- Auch: **Kausalordnung**
- Kausalität ist hier hypothetisch:
  - Jede **mögliche** Kausalität wird bewahrt
  - Es ist möglich, dass kausal geordnete Ereignisse gar nicht tatsächlich (semantisch) voneinander abhängig sind

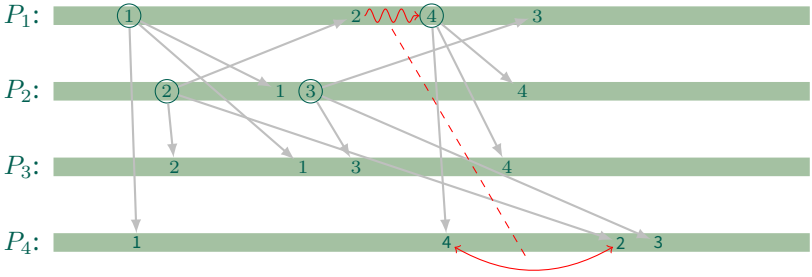
Kausale Ordnung (Forts.)



- Kausale Ordnung wird bei logischen Uhren genutzt (➡ nächstes Kapitel).

Kausale Ordnung (Forts.)

- Kausale Ordnung ist stärker als FIFO
  ➡ Jede kausale Ordnung ist auch FIFO geordnet, Umkehrung gilt jedoch nicht
- Beispielsweise erfüllt der Ablauf von Folie 13 die Kausalordnung **nicht**
  - $e_2 \prec e_4$, aber bei P_4 wird e_4 vor e_2 ausgeliefert



Vollständig sortierter Multicast

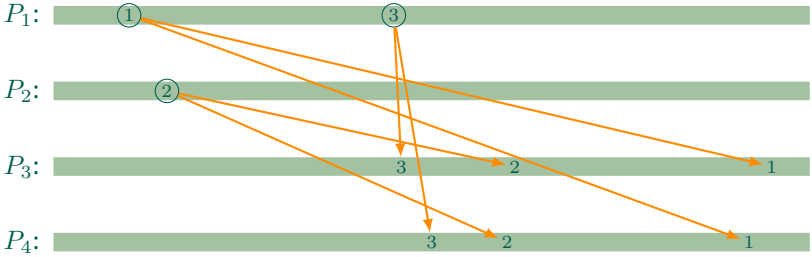
Vollständig sortierter Multicast

Wenn die beiden korrekten Mitglieder **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 Multicast mit vollständig sortierter Ordnung wird manchmal auch „Multicast mit totaler (Auslieferungs-)ordnung“ oder „Atomarer Multicast“ genannt
  - Letzteres ist etwas irreführend, da schon der ungeordnete Multicast atomar ist (Ganz-oder-gar-nicht-Semantik)

Vollständig sortierter Multicast (Forts.)

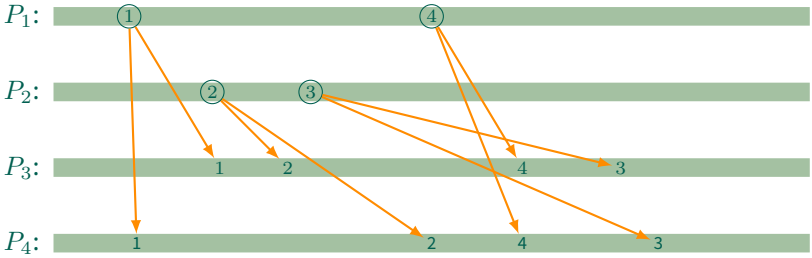
- Beispiel:



Vollständig sortierte FIFO- und Kausal-Ordnungen

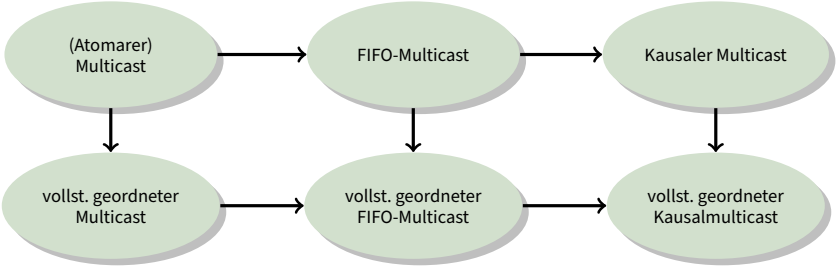
- Eigenschaften können kombiniert werden:
- **Vollständig sortierte FIFO-Ordnung**
  - Eine Ordnung, die sowohl first-in-first-out geordnet als auch vollständig sortiert ist
- **Vollständig sortierte Kausal-Ordnung**
  - Eine Ordnung, die sowohl kausal geordnet als auch vollständig sortiert ist
  - Häufig wird von **totaler Ordnung** (nicht nur bei der Auslieferung) oder globaler Ordnung gesprochen
  - Eng verwandt mit dem **Schnappschussproblem** ➡ späteres Kapitel

Beispiel für vollständig sortierte FIFO-Ordnung



Ordnung der Ordnungen

- Die Ordnungen können selbst nach der Stärke ihrer Anforderungen geordnet werden



5.4 Implementierung

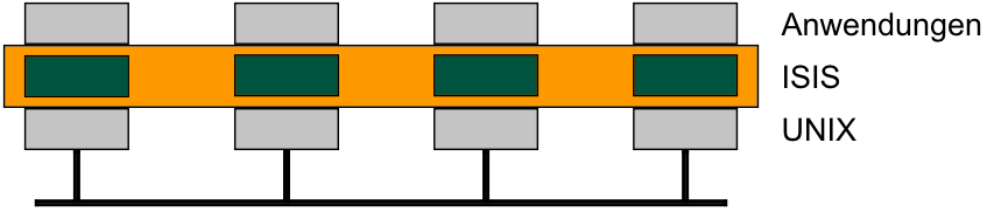
- Es gibt viele Systeme, die Gruppenkommunikation nutzen/realisieren
  - **ISIS**, Horus, Ensemble – Cornell University
  - **Transis** – Hebrew University
  - **Cactus** – Arizona State University
  - **JavaGroups** – open source
  - **Phoenix** – EPFL
  - **DCS** – IBM Haifa

5.4 Implementierung

- Es gibt viele Systeme, die Gruppenkommunikation nutzen/realisieren
  - **ISIS**, Horus, Ensemble – Cornell University
  - **Transis** – Hebrew University
  - **Cactus** – Arizona State University
  - **JavaGroups** – open source
  - **Phoenix** – EPFL
  - **DCS** – IBM Haifa

ISIS

- Entwickelt an der Cornell University Ihaka 1987 (Kenneth P. Birman)
- Kommunikationschicht (Toolkit) oberhalb UNIX, später Mach, Chorus und anderer OS



- Kommerzielle Anwender des Isis Toolkits:
  - New York Stock Exchange, Swiss Exchange
  - French Air Traffic Control System
  - AEGIS radar control and communications

CBCAST-Protokoll

- Ordnung durch (Sende-)**Ereigniszähler** (Zeitstempel)
  - Jeder Prozess P_i ($i = 1, \ldots, n$) besitzt einen Vektor $\vec{v}_i$
  - $\vec{v}_i = (v_{i,1}, \ldots, v_{i,n})$
  - Alle Prozesse starten mit einem Nullvektor

- **Senden**

Senden durch P_i

- Lokalen Zähler inkrementieren: $v_{i,i} := v_{i,i} + 1$
- $\vec{v}_i$ wird an Nachricht angehangen

CBCAST-Protokoll (Forts.)

- **Empfang**

Empfangen einer Nachricht von P_i durch P_j

- Lokale Nachrichten (d.h. $i = j$) werden sofort ausgeliefert
- Alle anderen Nachrichten werden solange zurückgehalten, bis folgende Kriterien erfüllt sind:
  1. Nachricht muss die nächste in einer Sequenz von P_i sein, ➡ $v_{i,i} \stackrel{!}{=} v_{j,i} + 1$
  2. Alle kausal vorhergehenden Nachrichten die an P_i gesendet wurden, sollten bei P_j ausgeliefert sein, ➡ $\forall k, v_{j,k} \stackrel{!}{\geq} v_{i,k}$
- Der lokale Zähler für den Sender wird inkrementiert
  ➡ $v_{j,i} := v_{j,i} + 1$

ABCAST-Protokoll

- Ursprünglich nur totale Ordnung, in späteren Versionen totale Kausalordnung
- **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
  - Vom 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

ABCAST-Protokoll (Forts.)

- Mitglieder, die keine Token-Halter sind, verzögern ABCAST-Nachrichten, bis
  - sie die entsprechende Sets-order-Nachricht empfangen haben
  - sie alle ABCAST-Nachrichten, auf die in der Sets-order-Nachricht bezug genommen wird, empfangen haben
  - sie alle kausal vorhergehenden CBCAST-Nachrichten ausgeliefert haben
- Das Token kann weitergegeben werden
  - **Vorteil**: Wenn (fast) nur Token-Halter ABCAST-Nachrichten versenden, sind keine (wenig) Sets-order-Nachrichten notwendig

Literatur

[Mul93] Mullender, S. J.: *Distributed Systems*. ACM Press, 1993 , Abschnitte 5.3 und 5.4

[BSS91] Birman, K., A. Schiper und P. Stephenson: „Lightweight causal and atomic group multicast“. *ACM Transactions on Computer Systems*, 9(3)1991, 272–314

[CDK02] Coulouris, G., J. Dollimore und T. Kindberg: *Distributed Systems: Concepts and Design*. Prentice Hall, 2002