

Distributed Operating Systems

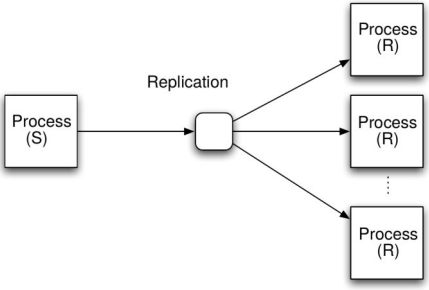
5th Chapter
Group Communication

Matthias Werner
Chair Operating Systems

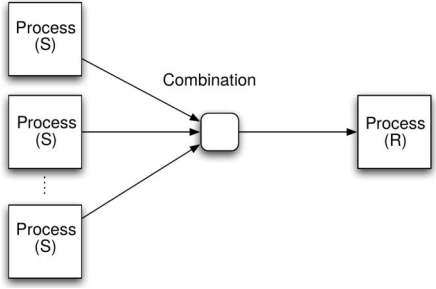
5.1 Introduction

- In distributed systems, often multiple partners take part in communication ➡ **Group Communication**

- **Multicast**: Message is sent to multiple receivers



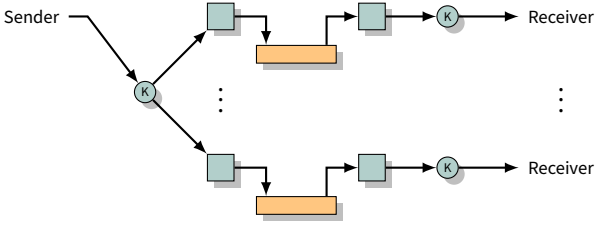
- **Combine**: Multiple messages are combined



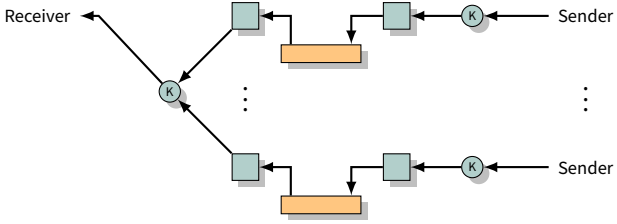
- We especially consider multicast

Realization through Bridges

- **Multicast**: Receivers on multiple computers

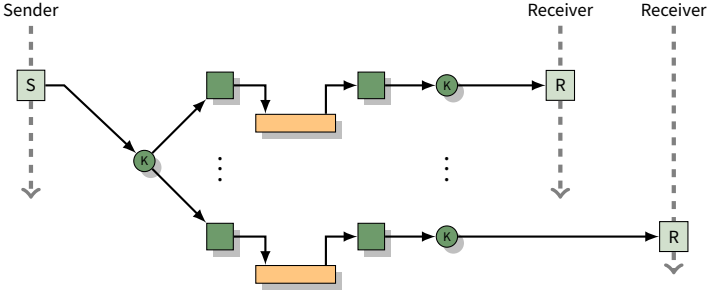


- **Combine**: Senders on multiple computers



Semantic Variants (for Multicast)

- The distribution of the channel complicates the preservation of the semantics of the local case
- Further semantic variants present themselves, which concern
  - the **reliability** of the delivery
    and
  - the **synchronization** of the involved processes



5.2 Reliability of the Delivery

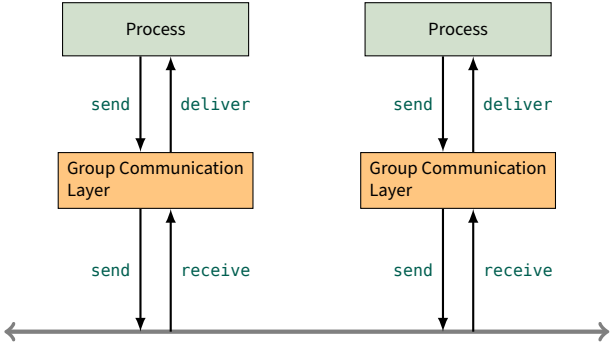
- **Unreliable Multicast**
  No promises are made about the success of the delivery
- **Reliable Multicast**
  The message is delivered to **at least** one receiver, but not necessarily to all
- **Atomic Multicast**
  The message is successfully delivered to **all** participants **or** to **none**

Attention

These definitions may vary in the literature; e.g. [CDK02 Chapter 11] defines the reliable broadcast as atomic, instead, there is a basic broadcast.

Retention at Proxy

- To preserve atomicity (and order) for the multicast, we distinguish **receiving** at the receiver-sided proxy and **delivery** to the receiver



Diskussion

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

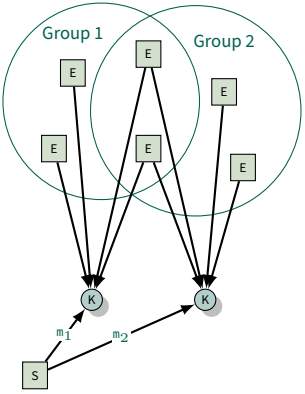
- Fehler treten ausschließlich in der Kommunikation vor Erreichen des Knotens auf.
- Prozess und Group Communication Layer fallen stets gemeinsam aus.

Synchronization

- The atomic multicast implies (for synchronous receiving) a **barrier synchronization**
- Delivery of the messages on the target sides and deblocking the receivers happens as recently as
  1. the messages have arrived everywhere
     and
  2. all receivers have called the delivery operation
- Otherwise, deblocking may happen as soon as the message has arrived the receiver

5.3 Order

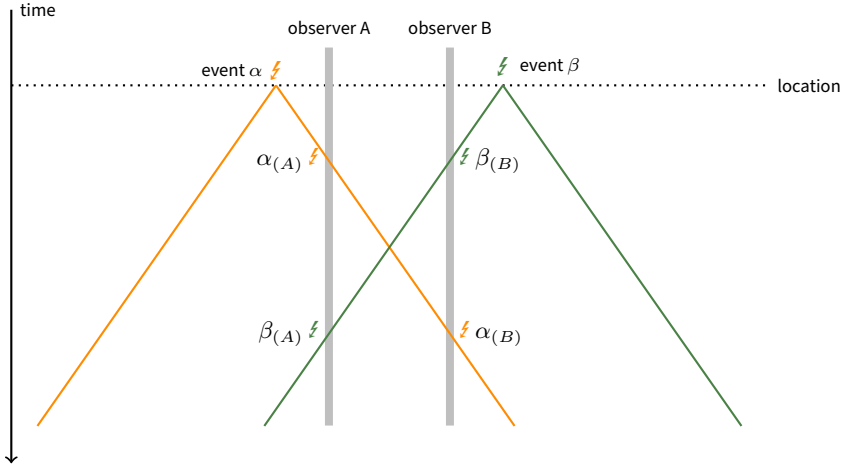
- Given are two overlapping groups of receivers $\mathbf{G}_1$ and $\mathbf{G}_2$, whereby a message is sent to each of them successively



- **Question**:
  In which order do the messages arrive the receivers?
- This question may be generalized

Event Horizon

- There is not necessarily a “right” order

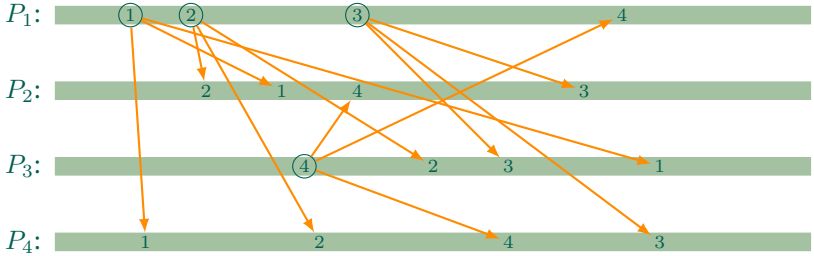


Possible Requirements on Orders

- There are different possible requirements on the delivery order
  - Non-ordered delivery
  - First In First Out (FIFO)
  - Causal order
  - Fully sorted (delivery) order
  - Fully sorted FIFO order
  - Fully sorted causal order (sometimes named “global order”)

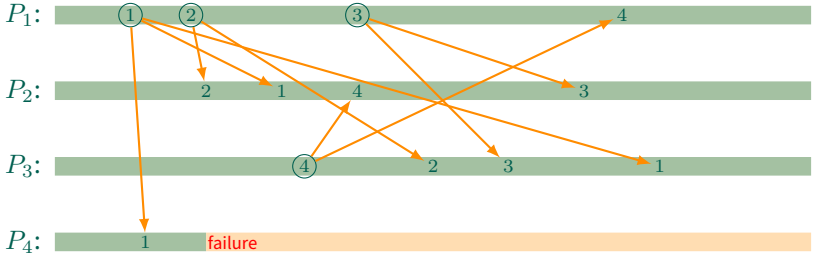
Arbitrary Delivery Order

- The atomic multicast itself does not demand a specific delivery order
- Only atomicity needs to be guaranteed (for faultless processes)



Arbitrary Delivery Order

- The atomic multicast itself does not demand a specific delivery order
- Only atomicity needs to be guaranteed (for faultless processes)



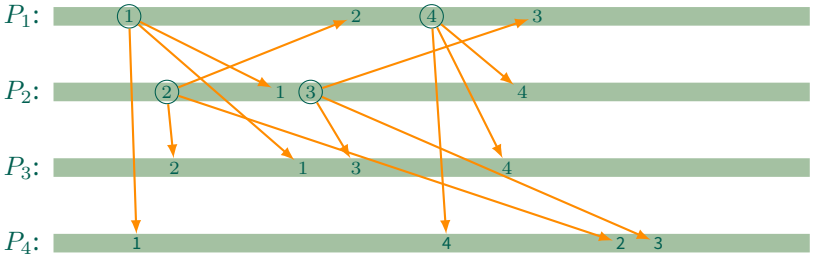
FIFO Order

- **Idea**:
  - Message is usually processed in a **context**
  - A message sent via multicast should find the same context at each receiver (with reference to the sender)
  - Example: Open file before write file

FIFO Order

If a correct member executes `multicast(`m_1`)` before `multicast(`m_2`)`, then no correct member executes `deliver(`m_2`)` before it has not delivered m_1.

FIFO Order (cont.)



Causal Order

- FIFO order refers to single senders only
- **But:**
  Not sufficient for more complex contexts (dependencies of multiple senders)
- **Example:** A receiver could receive a message m_1 and send a message m_2 in consequence of this message. Message m_2 could be delivered to a third member before m_1 is delivered (FIFO).
- To describe such cases, (potential) causality is described by means of the **Happened-before-Relation**

Happened-before-Relation

- Two events e_1 and e_2 fullfill the **Happened-before-Relation**, denoted as $e_1 \prec e_2$, if the following holds:
  1. e_1 and e_2 occur on one member (meaning locally) in exactly this order
     or
  2. e_1 is a `multicast`-event of a message and e_2 the dedicated delivery
     or
  3. there is an event e_0 with the result that $e_1 \prec e_0$ and $e_0 \prec e_2$
- Spoken: e_2 is **causally dependent** from e_1
- Alternative notation: $e_1 \rightarrow e_2$

Happened-before-Relation (cont.)

- The Happened-before-Relation is:
  - **Transitive:** $(a \prec b) \wedge (b \prec c) \Rightarrow a \prec c$
  - **Irreflexiv:** There is no a with $a \prec a$
  - **Asymmetric:** $(a \prec b) \Rightarrow \neg(b \prec a)$

- Two events a and b, $a \neq b$ are **causally independent**, if neither $a \prec b$ nor $b \prec a$ holds.
  - Notation: $a \parallel b$
  - The relation "$\parallel$" is reflexive, but not transitive

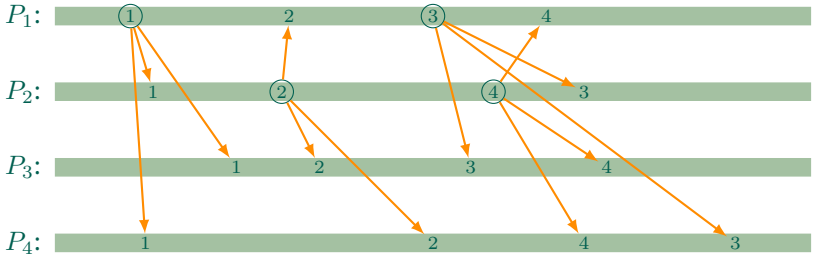
Causal Order

Causal Order

If $\texttt{multicast}(m_1) \prec \texttt{multicast}(m_2)$ holds, then no correct member executes $\texttt{deliver}(m_2)$ before it has delivered m_1

- Alias: **causal order**
- Causality is here hypothetic:
  - Every **possible** causality is preserved
  - It is possible that causal ordered events are not really (semantically) dependent on eachother.

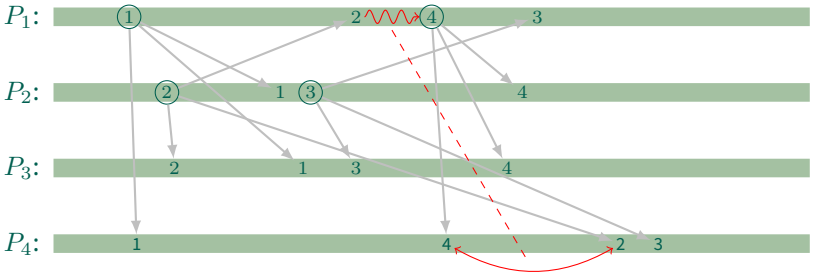
Causal Order (cont.)



- Causal order is used for logical clocks (➡ next chapter).

Causal Order (cont.)

- Causal order is stronger than FIFO: Every causal order is FIFO ordered, but the inversion does not hold
- E.g., the run at Slide 13 does **not** satisfy the causal order
  - $e_2 \prec e_4$, but at P_4, e_4 is delivered before e_2



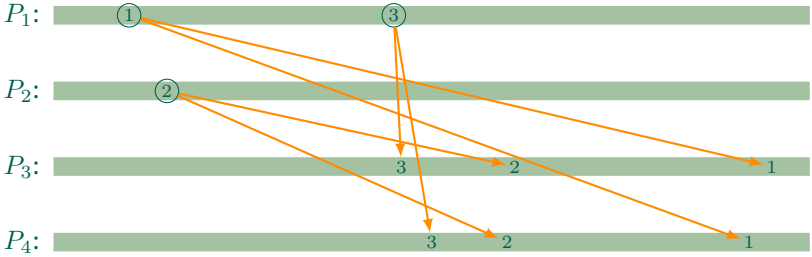
Fully Sorted Multicast

TECHNISCHE UNIVERSITÄT CHEMNITZ
Distributed Operating Systems – Group Communication
5.3 Order

- In other words: correct processes deliver the same sequence of messages
- **Please note**: No statement is made about the sequence itself; especially causality may be violated
- A multicast with fully sorted order is sometimes named “multicast with total delivery order” or “atomic multicast”
  - The latter is somewhat misleading, since the non-ordered multicast is already atomic

Fully Sorted Multicast (cont.)

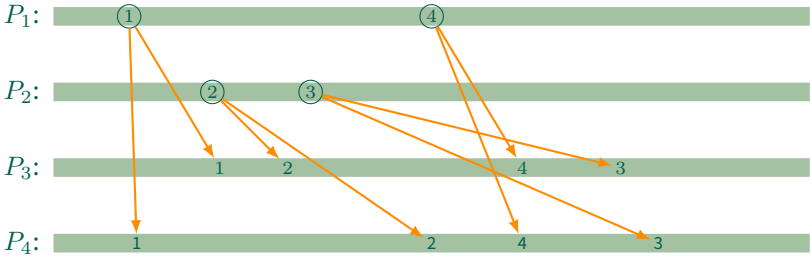
- Example:



Fully sorted FIFO and Causal Orders

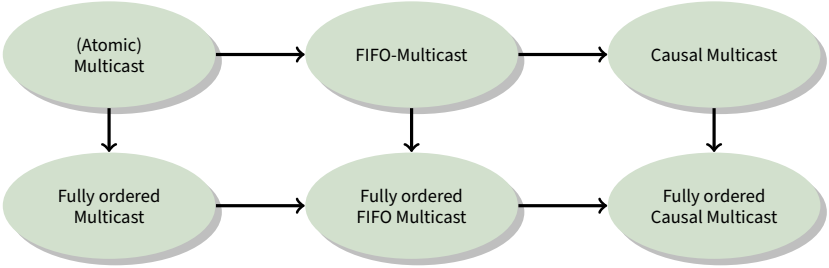
- Properties can be combined:
- **Fully sorted FIFO order**
  - An order that is not only FIFO ordered, but also fully sorted
- **Fully sorted causal order**
  - An order that is not only causally ordered, but also fully sorted
  - Commonly named “total order” (not only at delviery) or “global order”
  - Closely related to the **Snapshot Problem** ➡ later chapter

Example for fully sorted FIFO order



Order of Orders

- Orders may be ordered by the strength of their requirements



5.4 Implementation

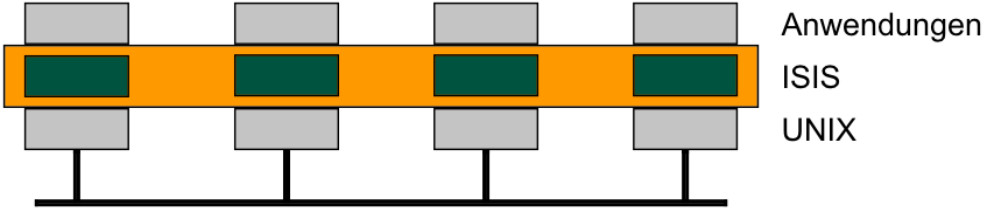
- There are many systems that use or realize group communication:
  - **ISIS**, Horus, Ensemble – Cornell University
  - **Transis** – Hebrew University
  - **Cactus** – Arizona State University
  - **JavaGroups** – open source
  - **Phoenix** – EPFL
  - **DCS** – IBM Haifa

5.4 Implementation

- There are many systems that use or realize group communication:
  - **ISIS**, Horus, Ensemble – Cornell University
  - **Transis** – Hebrew University
  - **Cactus** – Arizona State University
  - **JavaGroups** – open source
  - **Phoenix** – EPFL
  - **DCS** – IBM Haifa

ISIS

- Developed at Cornell University Ihaka 1987 (Kenneth P. Birman)
- Communication layer (toolkit) above UNIX, later Mach, Chorus and other OS



- Commercial users of ISIS:
  - New York Stock Exchange, Swiss Exchange
  - French Air Traffic Control System
  - AEGIS radar control and communications

CBCAST Protocol

- Order through (Send-)**Eventcounter** (time stamp)
  - Each process P_i ($i = 1, \ldots, n$) owns a vector $\vec{v}_i$
  - $\vec{v}_i = (v_{i,1}, \ldots, v_{i,n})$
  - All processors start with a zero vector
- **Senden**

Senden durch P_i

- Increment local counter: $v_{i,i} := v_{i,i} + 1$
- $\vec{v}_i$ is appended to the message

CBCAST Protocol (cont.)

- **Empfang**

Received of a message from P_i by P_j

- Local messages are delivered immediatly (i.e. $i = j$)
- All the other messages are restrained, until the following criteria are fullfilled
  1. Message has to be the next in a sequence of P_i, ➡ $v_{i,i} \stackrel{!}{=} v_{j,i} + 1$
  2. All the other causallly preceding messages which where sent to P_i should be delivered at P_j, meaning $v_{j,k} \stackrel{!}{\geq} v_{i,k}$
- The lokal conter for the sender will be incremented
  ➡ $v_{j,i} := v_{j,i} + 1$

ABCAST Protocol

- Originally total order only, in later versions total causal order
- **Idea: Central Sequencer**
  ABCAST uses CBCAST messages which are labelled as ABCAST messages
- A **token holder** is chosen in the group
  - Token holder receives ABCAST messages and delivers them locally in causal order
  - Messages sent by the token holder contain the order number of the message within the total order
  - Occasionally, the token holder sends a so-called **sets order message** which contains the sequence number of one or multiple ABCAST messages it received

ABCAST Protocol (cont.)

- Members which are not token holders, delay ABCAST messages, until
  - they have received the corresponding sets order message
  - they have received all ABCAST messages which are referenced in the sets order message
  - they have delivered all causally preceding CBCAST messages
- Token may be passed on
  - **Advantage**: If (almost) only token holders send ABCAST messages, then no (only a few) sets order messages are necessary

Literature

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

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

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