Universität Rostock

Traditio et Innovatio

Distributed Algorithms

Clocks

Univ.-Prof. Dr.-Ing. habil. Gero Mühl

Architecture of Application Systems
Faculty for Computer Science and Electrical Engineering
University of Rostock

Overview

> Time and Clocks

> Synchronization of physical clocks

> Order of events

> Logical clocks
  > Lamport clocks
  > Vector clocks
  > Application of vector clocks (causal multicast)

Time in Distributed Systems

The Importance of Time

- Determination of time and the measurement of time durations is indispensable for the coordination of human activities
- We have internalized the existence of a global time
- Therefore, our clocks have to be synchronized
  - Synchronization of clocks by means of church clock, telegraphy, radio, GPS
  - Clock synchronization made longitude determination in seafaring feasible

© NMM London, MoD Art Collection

Clocks

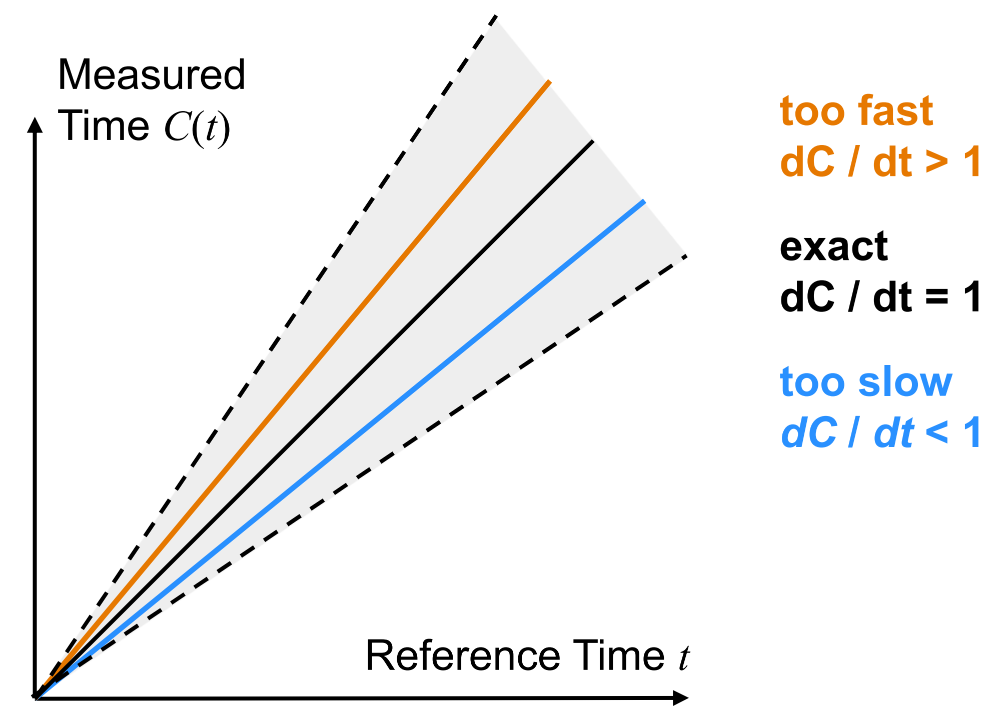
- Model: a clock maps the real time t to a time stamp $C(t)$
- Resolution
  - Smallest period of time by that two values of the clock can differ (e.g., 10 ms) → Tick duration
- Drift
  - Deviation of the speed of the clock from real time (ca. $\pm 10^{-6}$ with quartz clocks, ca. ±2sec per month)
- Offset
  - Deviation of the clock from real time at a point in time, i.e., $t - C(t)$
- A perfect clock has a drift and an offset of 0: $C(t) = t$

Correct Clocks

> A correct clock has a *limited maximal drift ρ*

$$(1+\rho)^{-1} \leq \frac{dC}{dt} \leq 1+\rho$$

„Speed“ of clock



Applications of Time Stamps

- Evaluate actuality of data
- Performance measurements
- Determine validity of access authorization
- Derive total order of events
  (e.g., for synchronization, debugging and audit)
- Evaluate sensor data and control actuators
  (real-time systems)
- …

Synchronization of Physical Clocks

Time in Distributed Systems

- Each computer has its own inaccurate digital clock
- The drifts of the clocks are different from each other
- Without synchronization, the values of the clocks can differ arbitrarily from each other → clock synchronization
- Clock synchronization in distributed systems is only possible through message exchange
- Hereby, the message delay plays an important role

Synchronization of Correct Clocks

- Two correct clocks with drift ρ should *not* deviate by more than time d
- How long is the maximum possible interval for synchronization?

- Assumptions
  - At $t = 0$, the clocks are synchronized: $C_1(0) = C_2(0)$
  - Worst case: One clock is as fast as drift allows, the other as slow as drift allows
  - Without loss of generality:
    - $C_1(t) / t = 1 + \rho$
    - $C_2(t) / t = 1 / (1 + \rho)$

Synchronization of Correct Clocks

> Therefore

$$(1+\rho)\ t - \frac{t}{1+\rho} \le d$$

$$\frac{(1+\rho)^2 t - t}{1+\rho} \le d$$

$$t\,\frac{2\rho+\rho^2}{1+\rho} \le d$$

$$t \le d\,\frac{1+\rho}{2\rho+\rho^2}$$

> Clocks must be synchronized again before $d\,\dfrac{1+\rho}{2\rho+\rho^2}$

> For very small ρ: synchronization before $\rho^{-1}\, d\,/\,2$

Synchronization Against a Perfect Clock

> Assumption: Clocks are synchronized at $t = 0$
> Two cases:

Clock as fast as possible

$$(1+\rho)\, t - t \leq d$$
$$\rho t \leq d$$
$$t \leq \frac{d}{\rho}$$

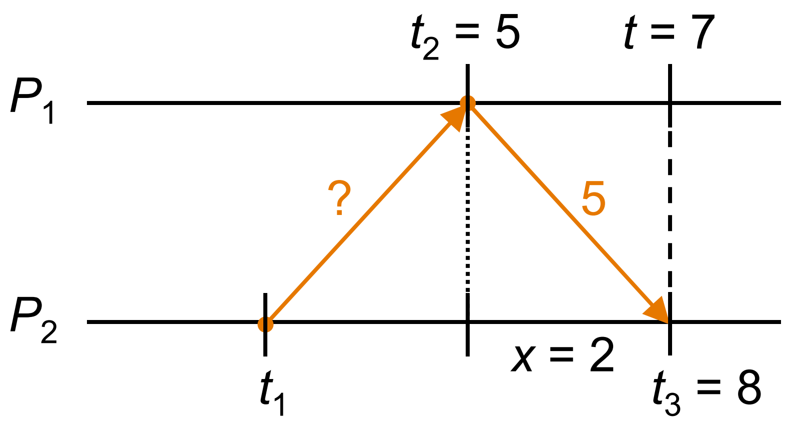
Clock as slow as possible

$$t - \frac{t}{1+\rho} \leq d$$
$$\frac{(1+\rho)t - t}{1+\rho} \leq d$$
$$\frac{\rho t}{1+\rho} \leq d$$
$$t \leq d\,\frac{1+\rho}{\rho}$$

> Since $\frac{d}{\rho} \leq d\,\frac{1+\rho}{\rho}$, the clock must be synchronized again before $\rho^{-1}\, d$

External Clock Synchronization [5]

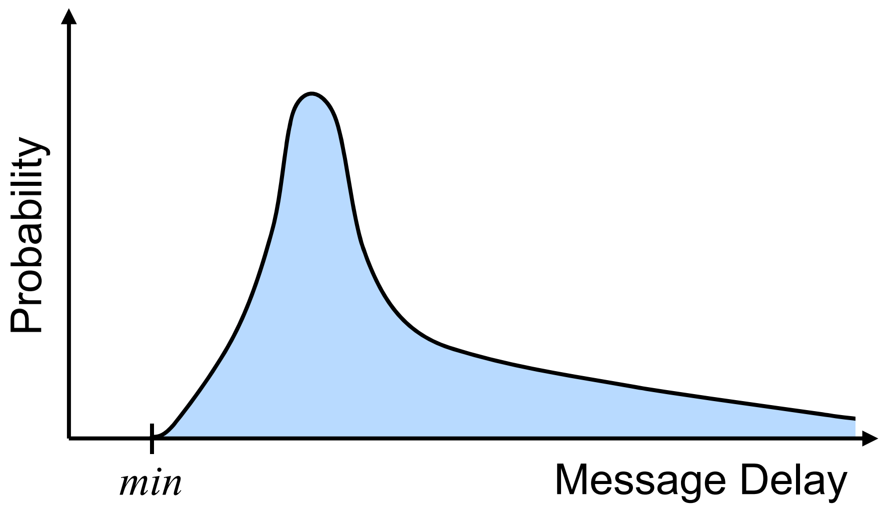
- Scenario: Process P_2 wants to adjust its clock on P_1
- Assumptions
  - P_1 has a clock with an assumed drift of 0
  - Message delay x is known
- P_2 adjusts its clock by $t_2 + x - t_3$
- Error of adjustment $e_{max} = 0$
- If the maximal deviation shall stay smaller than d,
  a new synchronization is at the latest necessary after $\rho^{-1} d$



→ P_2 sets its clock backwards by 1

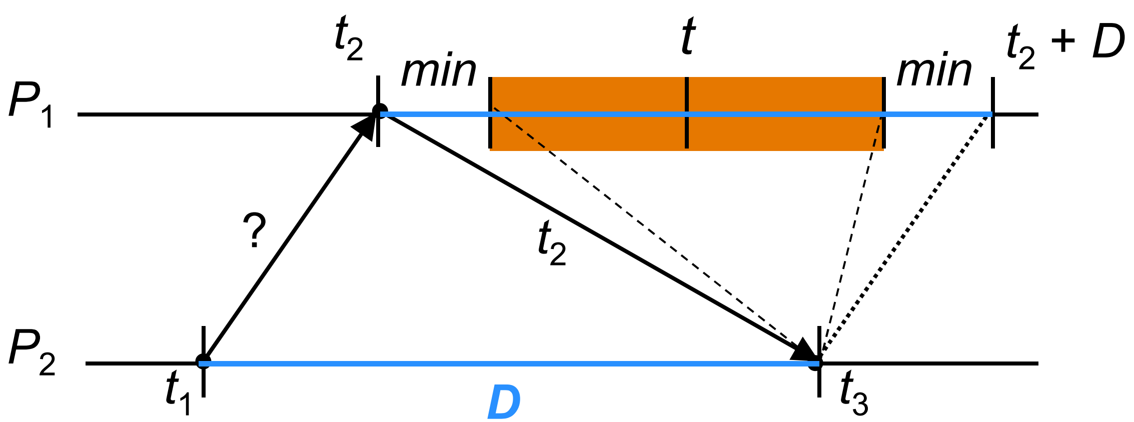
Unforeseeable Message Delay

- In reality, message delays are (in nearly all cases) load-dependent and unbounded
  - delay is higher with a high load than with a low load
  - delay can be arbitrarily long
- Then, the preceding procedure leads only to an approximate adjustment



Synchronization with Unforeseeable Delays

- Round Trip Time (RTT) $D := (t_3 - t_1)$
- Let t be the local time of P_1, when P_2 has the local time t_3
- t lies in the interval $[t_2 + min, t_2 + D - min]$
- Without further knowledge, the best prediction possible for t is the middle of the interval, i.e., $t_2 + D / 2$
- Thus, P_2 corrects its clock by $t_2 + D / 2 - t_3$
- The maximal adjustment error is $e_{max} = D / 2 - min$



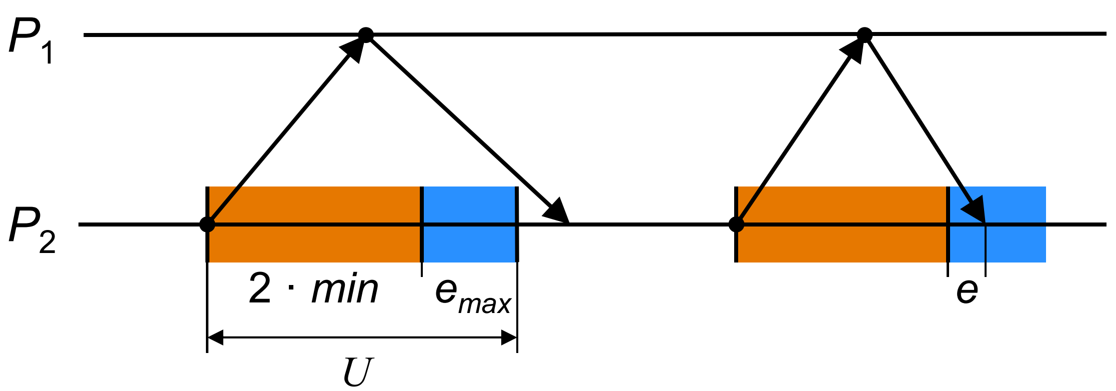
Probabilistic Limitation of the max. Error

> Idea: 1. Only accept the value of the reference clock if $D \leq U$ for a given bound $U > 2 \cdot min$
> 2. If that fails, repeat the attempt at most k-times, always after a waiting time W

> Let p be the probability that $D \geq U$ for one attempt

> Probability for k failures after another is then $q = p^k$

> Expected number of attempts until success is $E = 1 / (1 – p)$



Determination of the Synchronization Interval

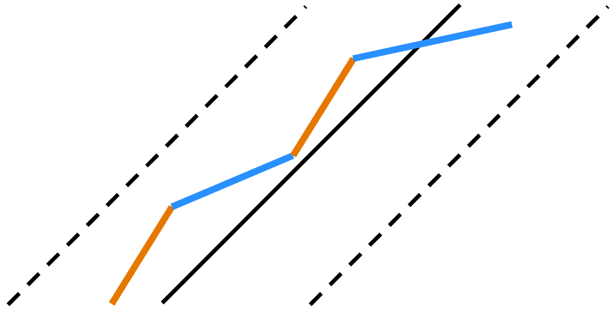
- The synchronization interval depends on the error of the last *successful* attempt
- To keep the deviation below d, the next success after a successful synchronization with error e must be after at most $\Delta = \rho^{-1} (d - e)$ time
  - Minimal synchronization interval $\Delta_{min} = \rho^{-1} (d - e_{max})$
  - Maximal synchronization interval $\Delta_{max} = \rho^{-1} d$
- First, the local clock is adjusted; then, the next attempt is started after $\Delta - k\,W$ at the latest
- Minimal maximal deviation with immediate start ($\Delta = k\,W$) is

$$d_{min} = k\,W\,\rho + e_{max}$$

Adjustment of the Local Clock Time

- Requirements
  - Great leaps of clock time must be avoided
  - Clock time must not decrease

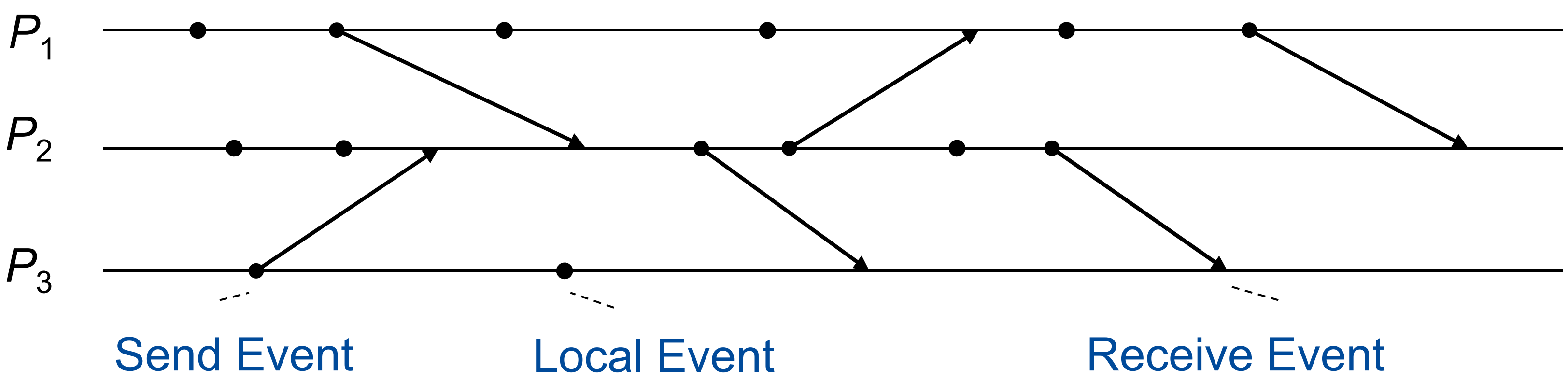
- Solution
  - Local clock is run slower or faster, until the offset is fully compensated



Order of Events

Order of Events in Distributed Systems

- Send and receive of a message are events
  → send event, receive event
- Additionally, there are local events



- In distributed systems, the absolute time of events are often not important; it often suffices to order the events

Order Relations

- An order relation $<$ is a
  - *irreflexive* — for no event a applies $a < a$
  - *asymmetrical* — $a < b \Rightarrow \neg(b < a)$
  - *transitive* — $a < b \land b < c \Rightarrow a < c$

  binary relation on the set of all events E

- Partial order:

  The order relation *is not* defined *for all* pairs of events

- Total order:

  The order relation is defined for *all* pairs of events, i.e.,

  $$e_1 \neq e_2 \Rightarrow e_1 < e_2 \lor e_2 < e_1$$

Possible Order Requirements

> FIFO (first in first out) order

> Causal order

> Total delivery order

> FIFO-total order

> Causal-total order

FIFO Order

> If a process sends message m_1 before message m_2, m_1 is delivered before m_2 to the receiver(s)

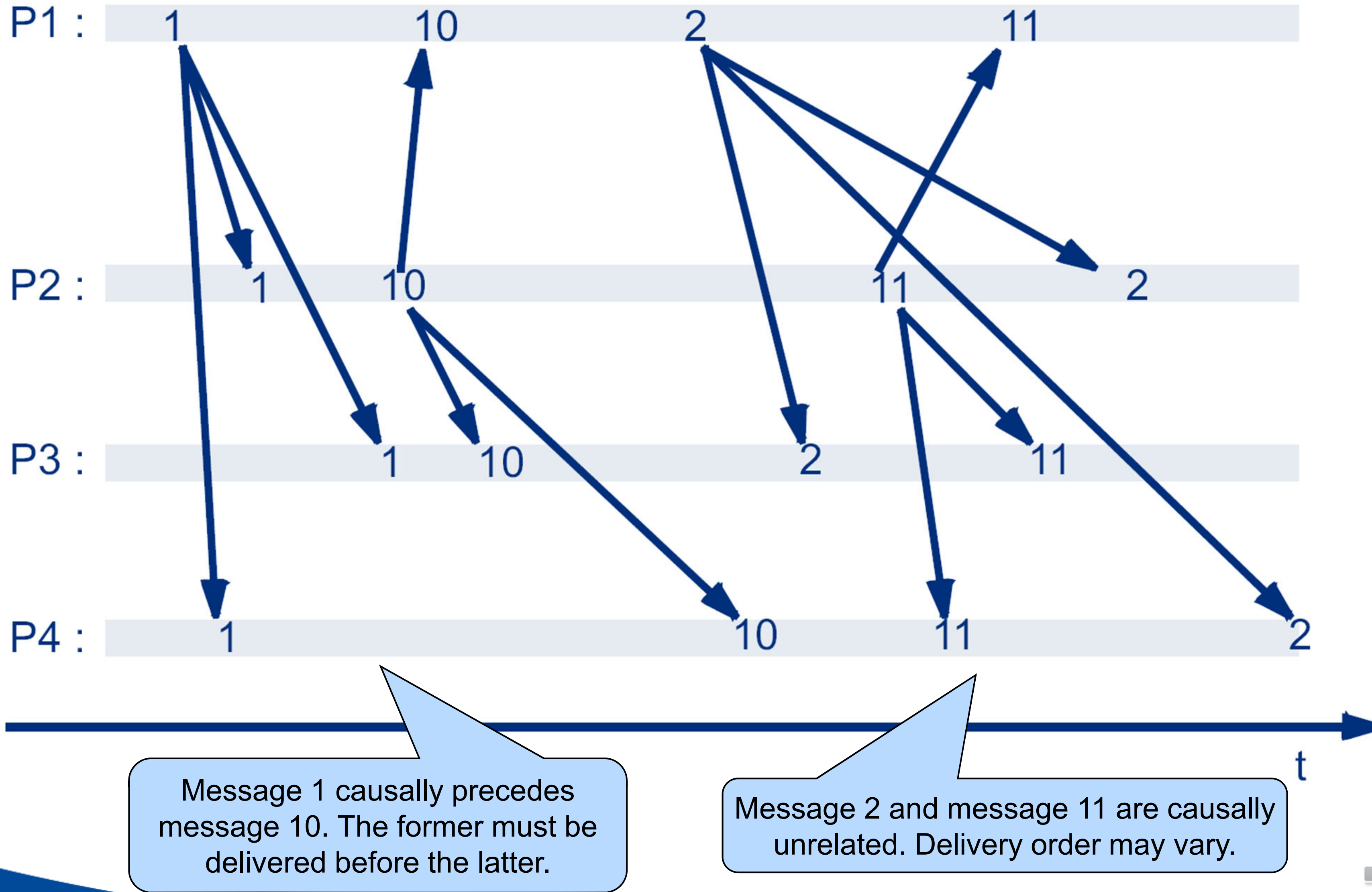
> Since messages can overtake each other, we distinguish between "receive" and "deliver"

> Messages are, then, delayed if necessary to enforce an order

Causal Order

- If the sending of message m_2 causally depends on the sending of the message m_1, then m_1 is delivered before m_2 to any receiver (getting both messages)
- Causal order is *stronger* than FIFO
  - Every causal order is also FIFO ordered
  - The inverse does not apply
- Causality is *hypothetical*
  - Every possible causality is preserved
  - It is, thus, possible that causally ordered events do not depend on each other in reality

Example: Causally Ordered Multicast

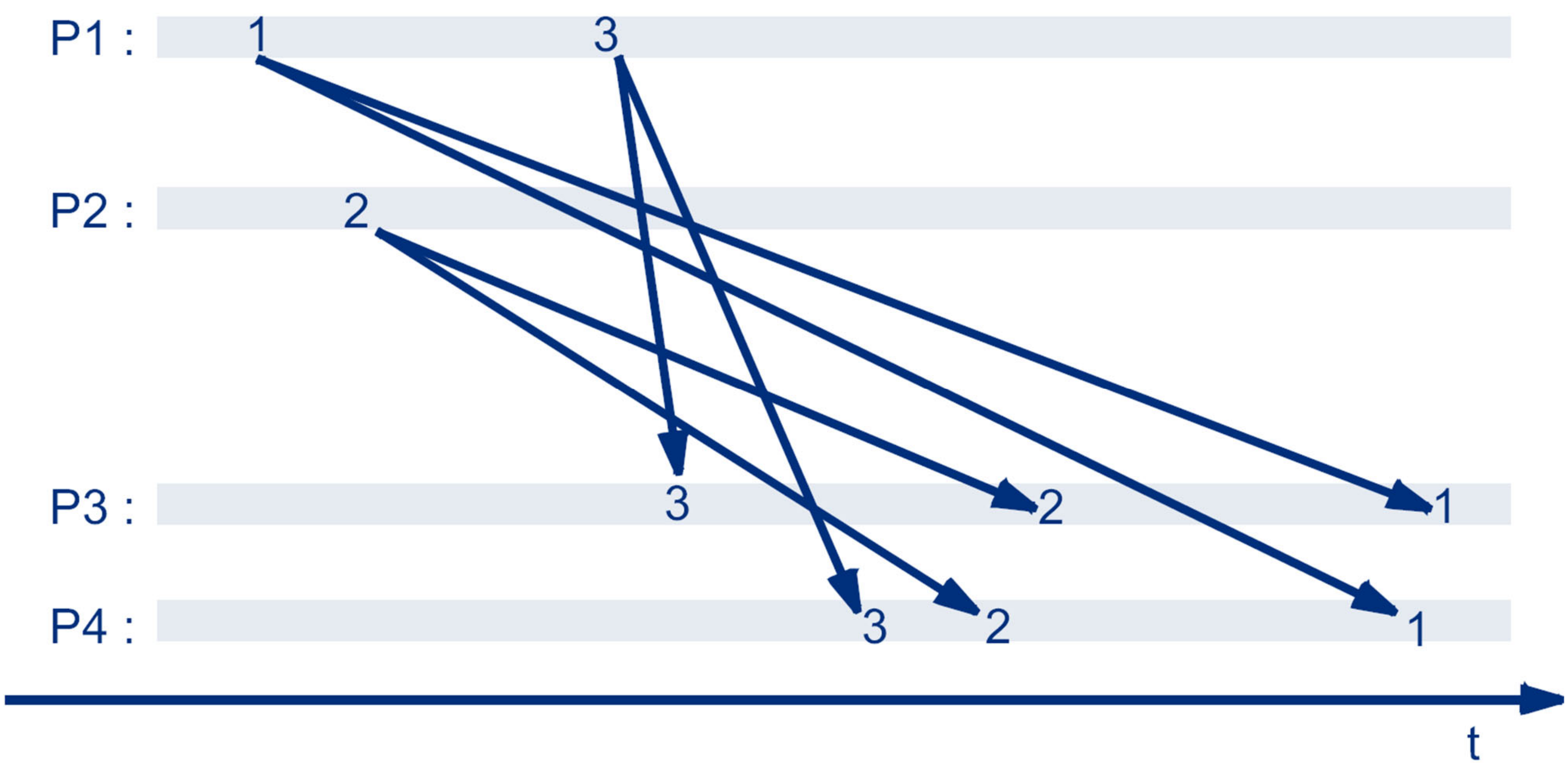


Total Delivery Order

> Only useful, if a message is delivered to more than one process → multicast communication
> If two processes *P* and *Q* both deliver the messages m_1 and m_2, then *P* delivers m_1 before m_2 if and only if *Q* also does.
> This means that all processes receiving m_1 *and* m_2, deliver the messages in the same order
> Caution: The order itself is not specified, especially, causality can be violated

> A multicast with total delivery order is also called atomic multicast
> Caution: A total delivery order is not always total in the sense of a total order relation (slide 29)

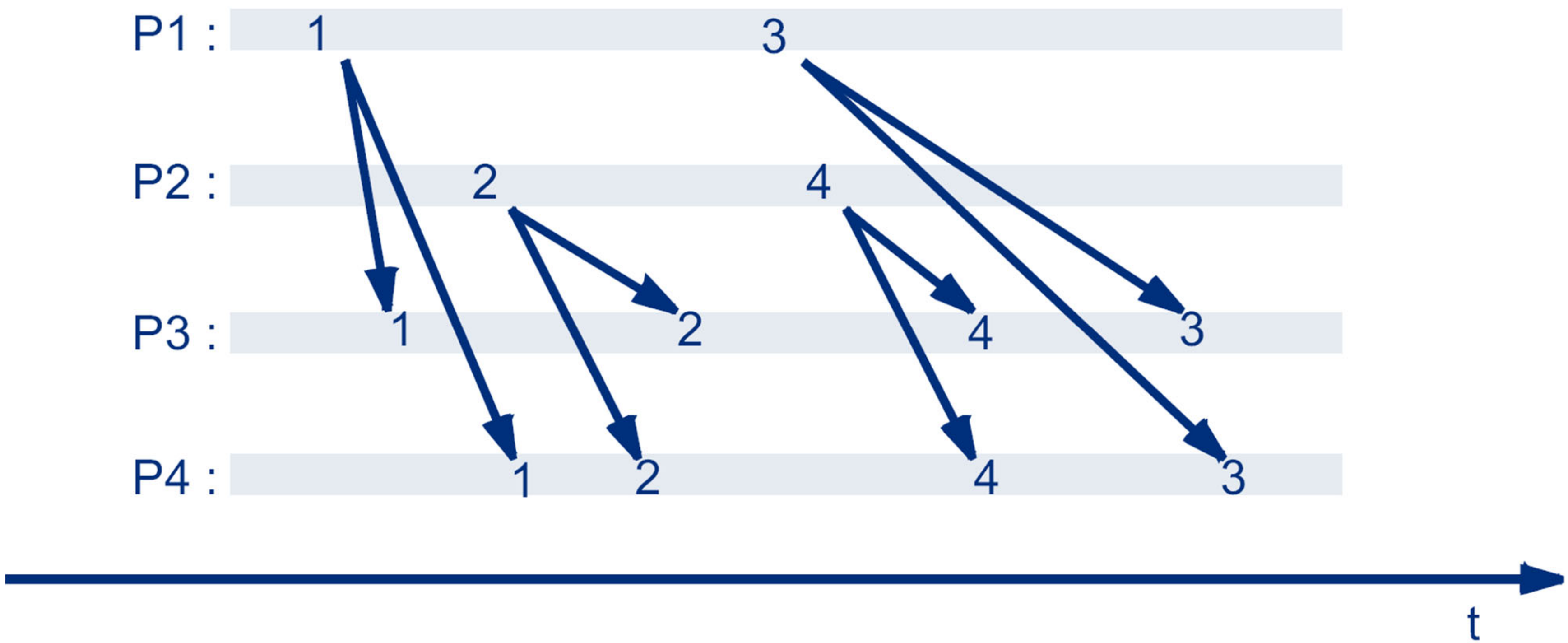
Example for Total Delivery Order



FIFO-Total and Causal-Total Order

- FIFO-Total order
  - An order that is both FIFO and total with respect to delivery
- Causal-Total order
  - An order that is both causal and total with respect to delivery
- As causal order implies FIFO order, a causal-total order is also a FIFO-total order

Example for FIFO-Total Order



Logical Clocks

A Simple Logical Time

- A logical clock assigns a time stamp $C(e)$ to each event e
- Each process P_i manages a counter C_i that is increased by 1 when an event e occurs
- The event e gets the new value of the counter as logical time stamp
- The logical time stamps $C(e)$ define a partial order on the set of events $e_1 < e_2 \Leftrightarrow C(e_1) < C(e_2)$
- It is partial as some events might get the same timestamp

A Simple Logical Time

- It is possible to supplement the simple logical time to a total order through the usage of unique process IDs as tiebreaker
- The time stamp $C'(e_i)$ of an event e_i is a pair (C_i, P_i)

$$e_1 < e_2 \Leftrightarrow C'(e_1) < C'(e_2)$$
$$\Leftrightarrow C_1 < C_2 \vee C_1 = C_2 \wedge P_1 < P_2$$

- Since process identities are unique, for two arbitrary events

$$e_1 \neq e_2 \Rightarrow e_1 < e_2 \vee e_2 < e_1$$

A Simple Logical Time – Problem

- Problem: The simple logical time does not take the causal relation between events into account

- Example: Replicated article data base
  - Editor adds a new article
  - Chief editor redacts the article afterwards
  - If the second action gets a smaller time stamp than the first one, the actions are applied in the wrong order and the data base contains the article which was not redacted

Happened-Before Relation (Lamport, 1978)

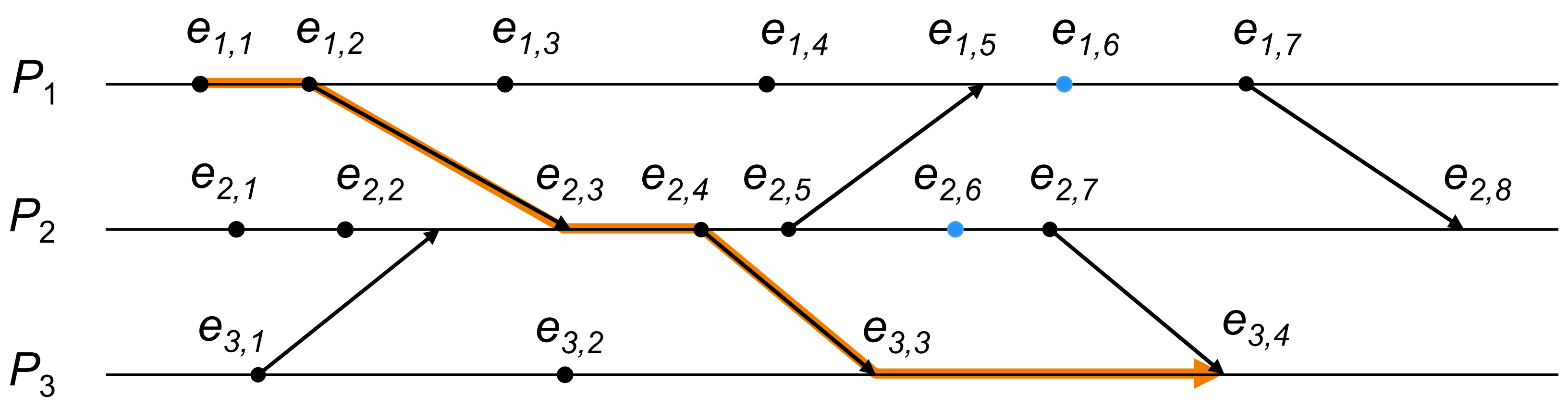
- The relation → („happened before") on the set of events is the *smallest* order relation that fulfills the following 3 conditions
  1. If *a* and *b* are two events in a process and *a* occurs before *b*, then $a \rightarrow b$
  2. If *a* is the sending of a message in a process and *b* the receipt of the same message in another process, then $a \rightarrow b$
  3. If $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$
- An event $b \neq a$ causally depends on *a*, if $a \rightarrow b$
- Two events $a \neq b$ are causally independent, written $a \,||\, b$, if neither $a \rightarrow b$ nor $b \rightarrow a$

"Smallest" here means that the relation contains exactly those pairs satisfying the conditions.

Condition 3 is implicitly implied by requiring that the relation is an order.

Happened-Before Relation – Interpretation

- a → b ⇒ "*b causally depends on a*"
- a || b ⇒ „*a* and *b* have *not* influenced each other causally"
- a → b ⇔ „One can get from *a* to *b* in the space-time diagram by following the process lines and message lines from *a* in the direction of *ascending* time"
- For the example below applies, e.g., $e_{1,1} \rightarrow e_{3,4}$ and $e_{1,6} \parallel e_{2,6}$



Clock Condition (Lamport, 1978)

- Requirement for a logical clock (clock condition)
  - For all events a, b: $\boldsymbol{a \rightarrow b \;\Rightarrow\; C(a) < C(b)}$
  - Thus, the clock *preserves* the causal order of the events
- Attention: implication only holds in one direction!
  - It only applies $C(a) < C(b) \;\Rightarrow\; a \rightarrow b \;\lor\; a \,||\, b$
- From the clock condition follows: $C(a) = C(b) \;\Rightarrow\; a \,||\, b$
- How can a logical clock be realized that fulfills the clock condition?

Lamport's Clocks – Realization

- Each process P_i has a local logical clock L_i, whose value is adapted when an event occurs
- A local event occurs at process P_i
  - Value of the local clock L_i is increased by one
  - Event gets the new value as time stamp
- P_i sends a message
  - Value of local clock L_i is increased by one
  - Send event gets new value as time stamp
  - Message sent carries time stamp of its send event t_m

Lamport's Clocks – Realization

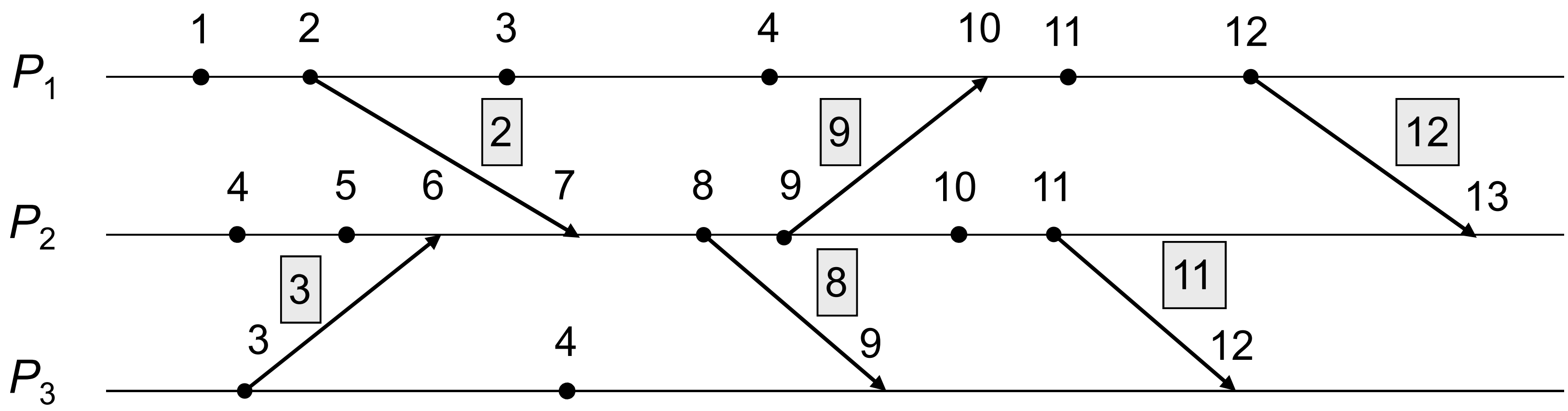
- P_i receives a message m with time stamp t_m
  - Value of the local clock is adapted to

$$L_i := \max(L_i, t_m) + 1$$

  - Receive event gets the new value as time stamp

- Incrementing after maximum calculation ensures that the receive event gets a higher time stamp than both the send event and the preceding local event at the receiver

Lamport's Clocks – Example



Lamport's Clocks – Characteristics

- Lamport's clocks fulfill the clock condition!
- The logical time stamps $L(e)$, thus, define a *partial* order on the set of events maintaining the causal relation
- Complementing it to a *total* order is possible again

- Problem
  - By means of a time stamp one can not tell for sure, whether two events are causally related
  - For that purpose also the converse of the clock condition would have to apply

Vector Clocks – Motivation

- Assume we had a clock that turns the implication of the clock condition into an equivalence
  - $a \rightarrow b \;\Leftrightarrow\; C(a) < C(b)$
- With such a clock, we could determine how events are related by means of the time stamps of the events
  - $C(a) < C(b) \Rightarrow a \rightarrow b$
  - $C(b) < C(a) \Rightarrow b \rightarrow a$
  - $\neg\,(C(a) < C(b) \;\vee\; C(b) < C(a)) \quad \Rightarrow \quad a \,||\, b$
- For all three equations also the converse applies
- For $\neg(C(a) < C(b) \;\vee\; C(b) < C(a))$, we write $C(a) \,||\, C(b)$
- How can a clock with such characteristics be realized?

Vector Clocks – (Mattern, Fidge, 1988)

- Each process P_i holds a vector time stamp V_i consisting of n counters that are initially all zero

- Local event
  - If an event occurs in a process P_i , it increments the i-th component of its vector
- Sending a message
  - If P_i sends a message, the new version of V_i is sent along with the message
- Receiving a message
  - If P_i receives a message with vector time stamp T, it assigns to V_i the component-wise maximum of the T and the new version of V_i

- The vector time can (also) be extended to a *total* order by using process identities as tiebreakers

Vector Clocks

> Component-wise maximum of two vectors

$$max(V_i, V_j) := (max(V_i[1], V_j[1]), \ldots, max(V_i[n], V_j[n]))$$

> Component-by-component comparison of two vectors

$$V_i < V_j \Leftrightarrow V_i \neq V_j \wedge \forall\, 1 \leq k \leq n : V_i[k] \leq V_j[k]$$

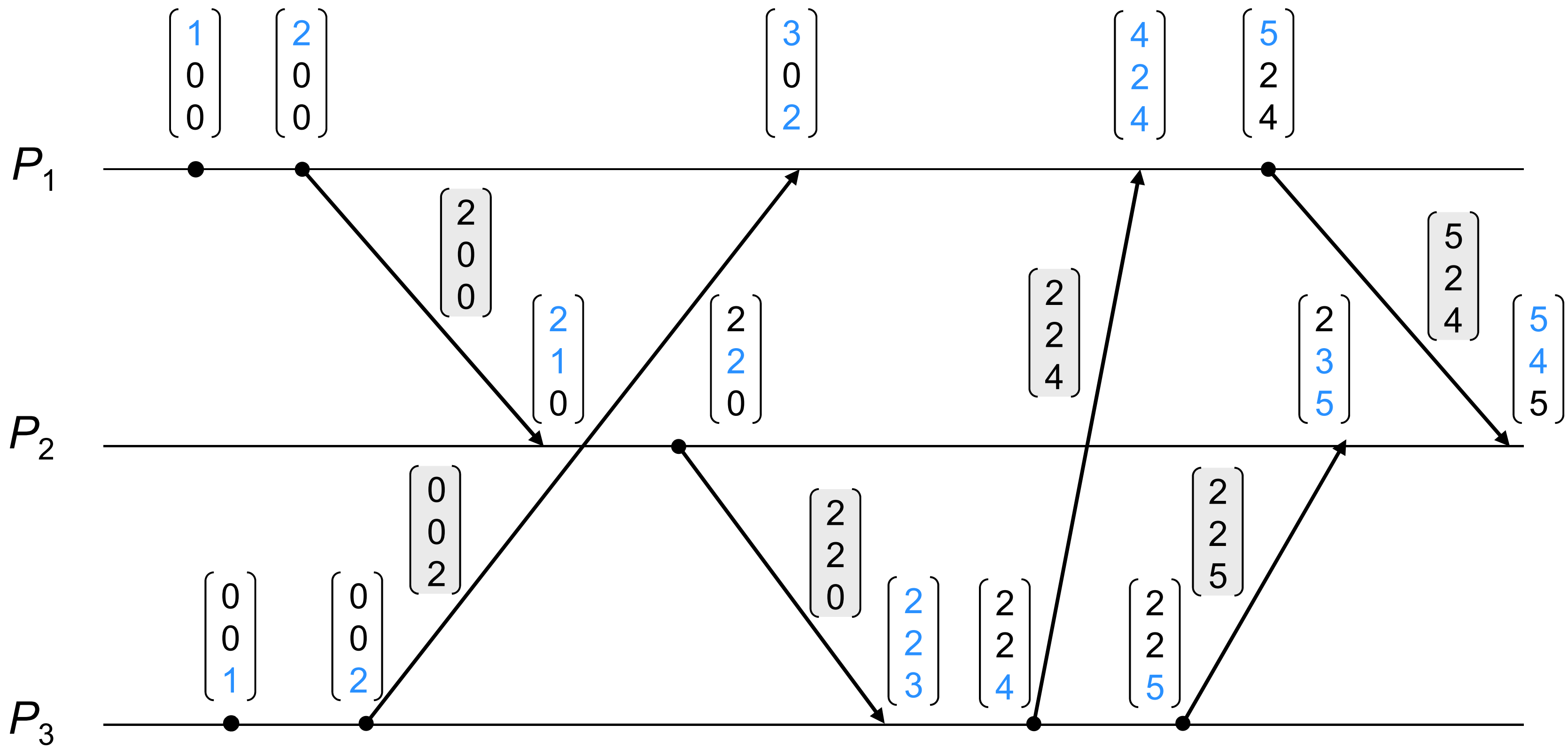
> Vector clocks only define a *partial* order!

$$\begin{bmatrix} 2 \\ 2 \\ 4 \\ 2 \\ 8 \\ 6 \end{bmatrix} < \begin{bmatrix} 2 \\ 3 \\ 5 \\ 6 \\ 8 \\ 9 \end{bmatrix}$$

$$\begin{bmatrix} 1 \\ 2 \\ 4 \\ 2 \\ 5 \\ 6 \end{bmatrix} \,||\, \begin{bmatrix} 3 \\ 5 \\ 3 \\ 1 \\ 8 \\ 9 \end{bmatrix}$$

$$max\left(\begin{bmatrix} 1 \\ 2 \\ 4 \\ 2 \\ 5 \\ 6 \end{bmatrix}, \begin{bmatrix} 3 \\ 5 \\ 3 \\ 1 \\ 8 \\ 9 \end{bmatrix} \right) = \begin{bmatrix} 3 \\ 5 \\ 4 \\ 2 \\ 8 \\ 9 \end{bmatrix}$$

Vector Clocks – Example



Relation Between the Time Stamps

- $R(e)$ exact real time of the event e
- $L(e)$ Lamport-time of e
- $V(e)$ Vector-time of e

$$\begin{tikzcd} R(a) < R(b) & V(a) < V(b) \arrow[l, Rightarrow] \arrow[d, Rightarrow] \\ a \rightarrow b \arrow[u, Rightarrow] \arrow[r, Rightarrow] \arrow[ur, Leftrightarrow] & L(a) < L(b) \end{tikzcd}$$

Exemplary Applications of Logical Clocks

Application of Vector Clocks: Causal Multicast

- Each message is delivered to all processes
- Messages must be delivered in an order satisfying causality
- Update policy of the vector clocks is changed such that P_i *only* increments $V_i[\,i\,]$ if it sends a message

- Delivery condition
  - A message with time stamp T sent by P_i is not delivered to P_j before T fulfills the following conditions:

$$\underbrace{T[\,i\,] = V_j[\,i\,] + 1}_{\text{Next message awaited from } P_i} \;\wedge\; \underbrace{\forall k \neq i\text{: } T[\,k\,] \leq V_j[\,k\,]}_{\text{No message from arbitrary } P_k \text{ is missing}}$$

- 1st condition violated → a message from P_i is missing
- 2nd condition violated → a message from P_k is missing that P_i had already received at the time it sent the message received yet

Application of Vector Clocks: Causal Multicast

- The maximum of V_j und T is derived after the delivery of the message
- Due to the second delivery condition:

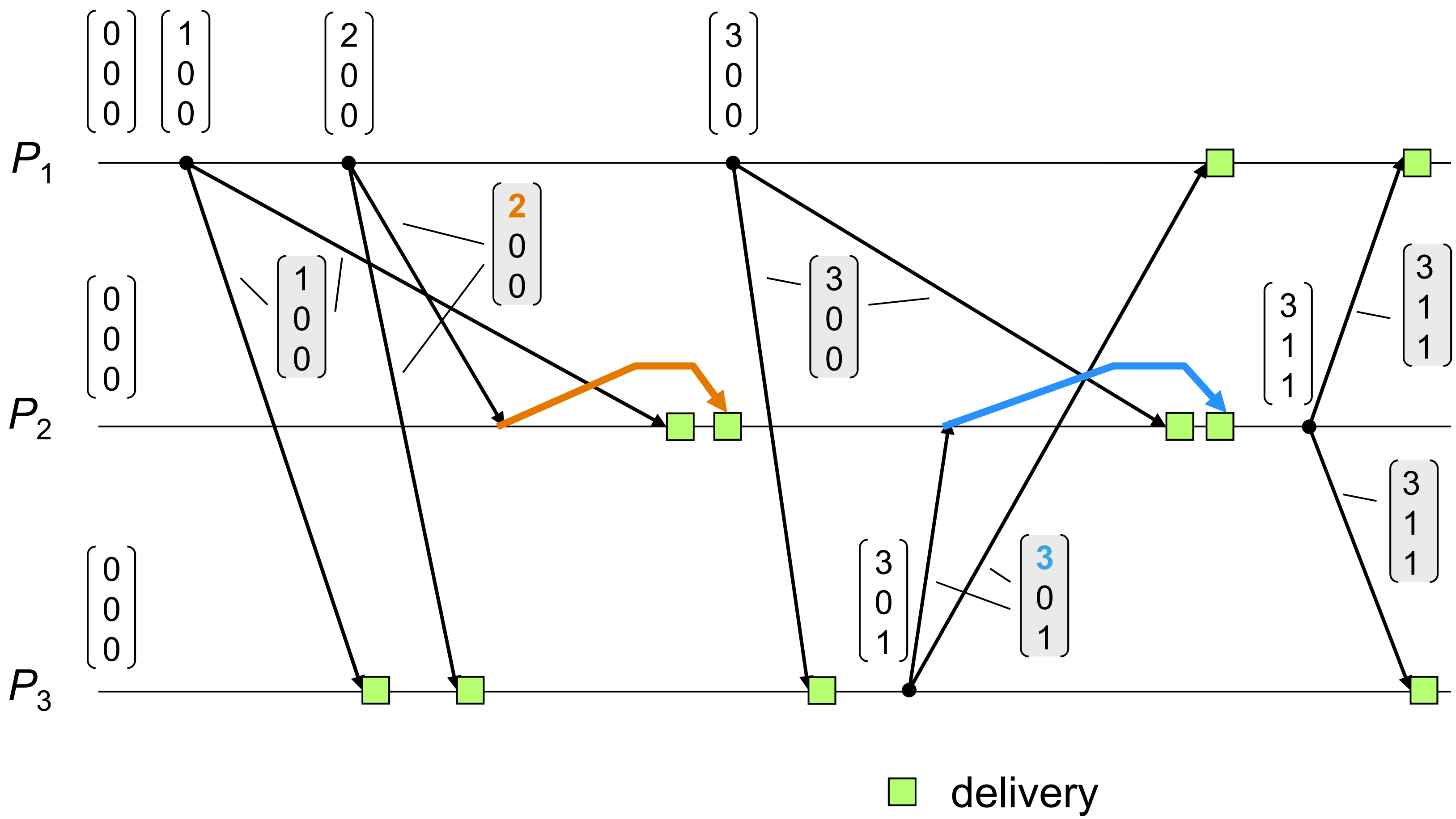
  $$V_j[\,k\,] \geq T[\,k\,] \text{ for all } k \neq i$$

- This means that except for the component $V_j[\,i\,]$, V_j only contains components that are larger or equal to those in T
- Due to the first delivery condition, for $V_j[\,i\,]$ holds:

  $$V_j[\,i\,] = T[\,i\,] - 1$$

- Therefore, calculating the maximum is not necessary
- Instead, it is sufficient to increase $V_j[\,i\,]$ by one at delivery

Causal Multicast – Example



Exemplary Exam Questions

1. Explain the external clock synchronization algorithm!
2. What is Lamport’s clock condition?
3. How the Happened-Before-Relation is defined?
4. Describe the functioning of Lamport clocks!
5. Can events be arranged by Lamport clocks in such a way that causally related events are properly ordered?
6. Explain the functionality of vector clocks!
7. What additional opportunities vector clocks can provide compared to Lamport clocks?
8. Describe how a causal multicast can be realized using vector clocks!

Literature

1. A. S. Tanenbaum and M. van Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall, 2002. Chapter 5.1 + 5.2
2. G. Coulouris, J. Dollimore, and T. Kindberg. *Distributed Systems: Concepts and Design*. Addison-Wesley, 4th edition, 2005. Chapter 11.1 – 11.4
3. N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996. Chapter 18
4. F. Mattern. *Verteilte Basisalgorithmen*. Springer-Verlag, 1989. Kapitel 5: Virtuelle Zeit in verteilten Systemen
5. F. Cristian. *Probabilistic clock synchronization*. Distributed Computing, 3(3):146--158, September 1989.

Literature

6. D. L. Mills. *Network Time Protocol (Version 3)*. Internet Request for Comments RFC 1305, March 1992.
7. L. Lamport. *Time, Clocks, and the Ordering of Events in a Distributed Environment*. Communications of the ACM, 21:558--564, July 1978.
8. F. Mattern. *Virtual Time and Global States of Distributed Systems*. In C. M. et al., editor, Proc. Workshop on Parallel and Distributed Algorithms, pages 215--226, North-Holland / Elsevier, 1989. (Reprinted in: Z. Yang, T. A. Marsland (Eds.), "Global States and Time in Distributed Systems", IEEE, 1994, pp. 123-133.).
9. Michel Raynal. *About logical clocks for distributed systems*. SIGOPS Oper. Syst. Rev. 26, pp. 41-48, 1992.

Thank you for your kind attention!

Univ.-Prof. Dr.-Ing. habil. Gero Mühl

`gero.muehl@uni-rostock.de`
`http://wwwava.informatik.uni-rostock.de`