

Distributed Operating Systems

7th Chapter
Logical Clocks

Matthias Werner
Chair Operating Systems

7.1 Motiviation

- Conclusion from previous chapter: Clock synchronization is costly, often failure-prone and only limited applicable

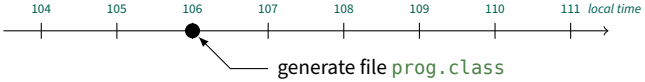
Friedemann Mattern

"The challenge consists in defining an abstract notion of time suitable for distributed systems which, on the one hand, is easily realizable without using physical clocks but, on the other hand, has enough interesting properties to justify the name 'time'."

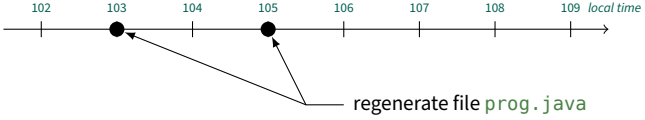
- In computer systems, its usually sufficient to describe the order of events
- Clock turns into event counter
- Related to orders of broadcast ➡ Chapter 4

Example Problem: Distributed `make`

- Computer where the compiler runs



- Computer where the editor runs



7.2 Lamport's Clocks

- Requirement to a logical clock (**Clock Condition**) (Lamport, 1978)
  - For all events a, b shall hold:

(Weak) Clock Condition

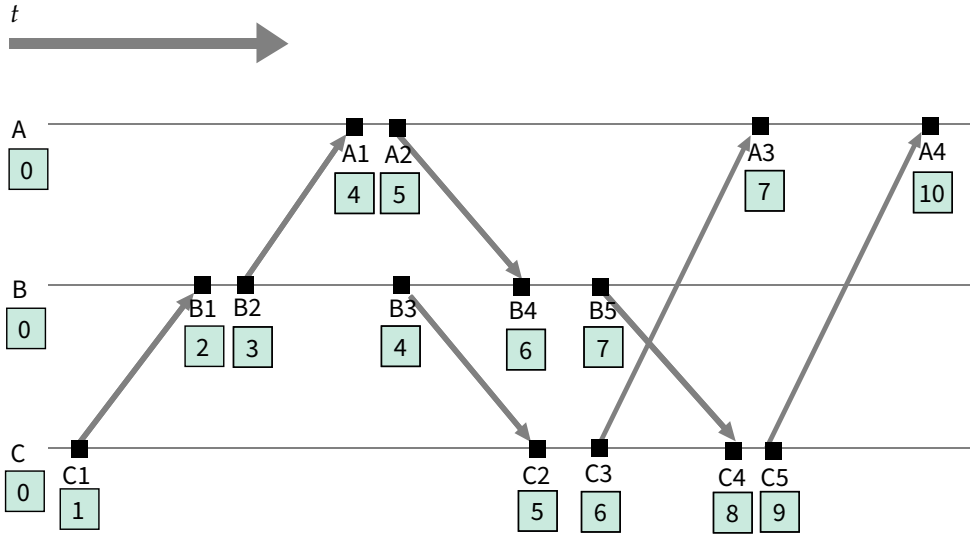
$$a \prec b \Rightarrow C(a) < C(b)$$

  - Meaning, the clock preserves the **causal** order of the events
- **Attention:** This implication holds in one direction only!
  - It holds only: $C(a) < C(b) \Rightarrow (a \prec b) \vee (a \parallel b)$
- From the clock condition follows: $C(a) = C(b) \Rightarrow a \parallel b$
- How can a logical clock, which fulfills the clock condition, be realized?

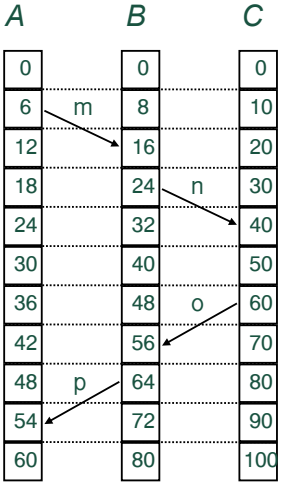
Lamport's Clocks – Realization

- Each process P_i owns a local logical clock L_i, whose value is adapted as soon as an event occurs
  - Local event at process P_i
    - Value of local clock L_i is increased by one
    - Event obtains the new value as timestamp
  - P_i sends message m
    - Value of local clock L_i is increased by one
    - Send event obtains new value as timestamp
    - m carries as timestamp t_m the timestamp of its send event
  - P_i receives a message
    - Value of local clock is adapted $L_i' = \max(L_i, t_m) + 1$
    - Receive event obtains new value as timestamp

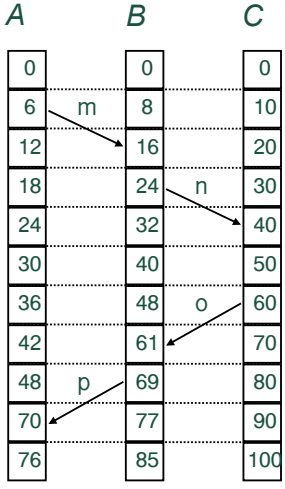
Lamport's Clocks – Example



Lamport's Clocks – Synchronization



without synchronization



with synchronization

7.3 Hybrid Clocks

- Lamport clocks fulfill the clock condition!
- The logical timestamps $L(e)$ hence define a partial order on the set of events
- Usually, we are interested in a causally correct, but usable time
  - The statement “The error occured at time 42” is only of limited usefullness
- **Solution**: **hybrid** or physical-logical[1] clocks $H(e)$

[1] [Kul+14]

7.3 Hybrid Clocks

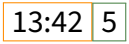
- Lamport clocks fulfill the clock condition!
- The logical timestamps $L(e)$ hence define a partial order on the set of events
- Usually, we are interested in a causally correct, but usable time
  - The statement “The error occured at time 42” is only of limited usefullness
- **Solution**: **hybrid** or physical-logical[1] clocks $H(e)$

13:42

[1] [Kul+14]

7.3 Hybrid Clocks

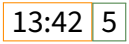
- Lamport clocks fulfill the clock condition!
- The logical timestamps $L(e)$ hence define a partial order on the set of events
- Usually, we are interested in a causally correct, but usable time
  - The statement “The error occured at time 42” is only of limited usefullness
- **Solution**: **hybrid** or physical-logical[1] clocks $H(e)$



[1] [Kul+14]

7.3 Hybrid Clocks

- Lamport clocks fulfill the clock condition!
- The logical timestamps $L(e)$ hence define a partial order on the set of events
- Usually, we are interested in a causally correct, but usable time
  - The statement “The error occured at time 42” is only of limited usefullness
- **Solution**: **hybrid** or physical-logical[1] clocks $H(e)$



  - Consists of physical timestamps t_P (dominant) and logical timestamps t_L, $H(e) = (t_P, t_L)$
  - Rules ensure that causality is preserved

[1] [Kul+14]

Hybrid Clocks – Rules

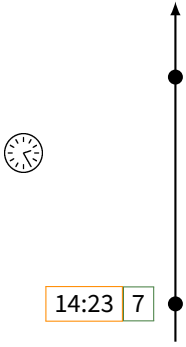
- **Assumption**: Each node owns a roughly synchronous clock
- As before, timestamps are send by messages
- If a new timestamp (t'_P, t'_L) shall be generated, the local time t_C is read
  - New local event:

$$(t'_P, t'_L) = \begin{cases} (t_C, 0) & \text{if } t_C > t_P \\ (t_P, t_L + 1) & \text{else} \end{cases}$$

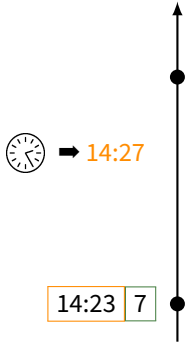
  - Receiving a message with timestamp $(t_{m,P}, t_{m,L})$:

$$(t'_P, t'_L) = \begin{cases} (t_C, 0) & \text{if } t_C > \max(t_P, t_{m,P}) \\ (t_{m,P}, t_{m,L} + 1) & \text{if } (t_{m,P} > t_P) \wedge (t_{m,P} \geq t_C) \\ (t_P, t_L + 1) & \text{if } (t_P > t_{m,P}) \wedge (t_P \geq t_C) \\ (t_P, \max(t_L, t_{m,L}) + 1) & \text{else} \end{cases}$$

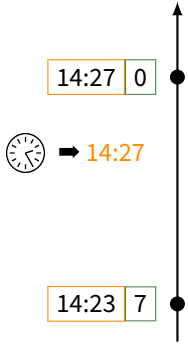
Example for hybrid clocks



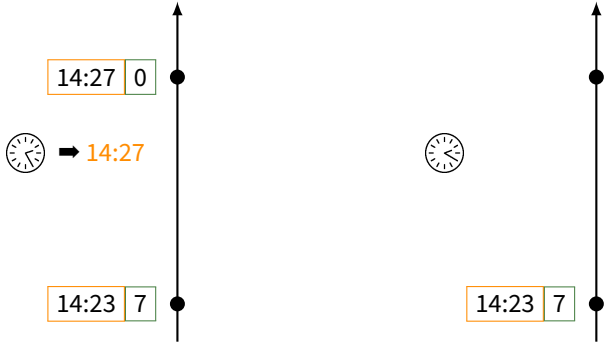
Example for hybrid clocks



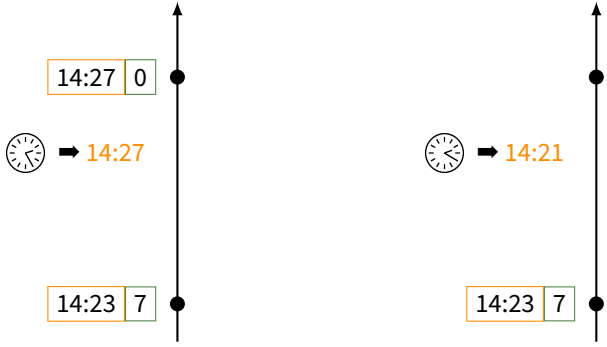
Example for hybrid clocks



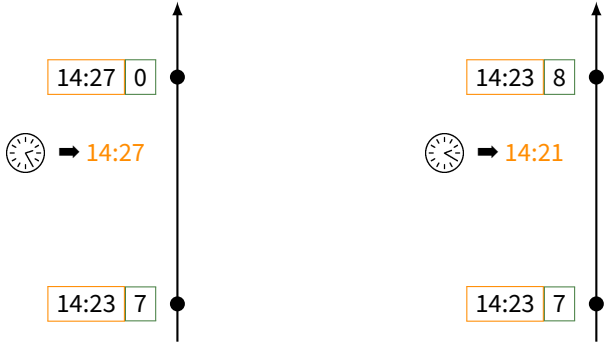
Example for hybrid clocks



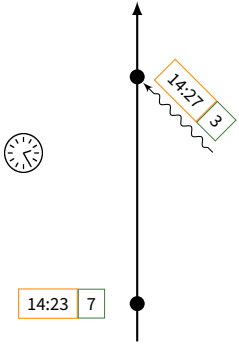
Example for hybrid clocks



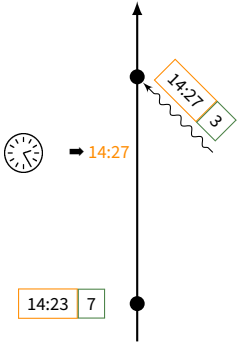
Example for hybrid clocks



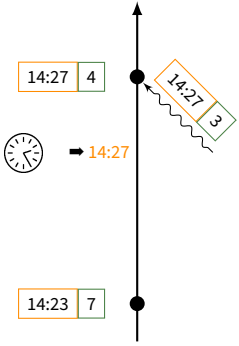
Example for hybrid clocks (cont.)



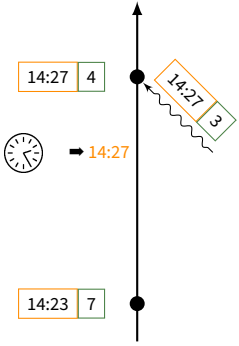
Example for hybrid clocks (cont.)

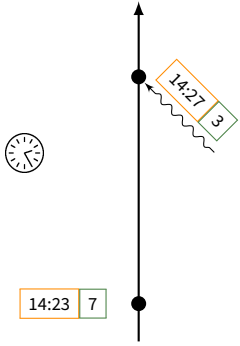


Example for hybrid clocks (cont.)

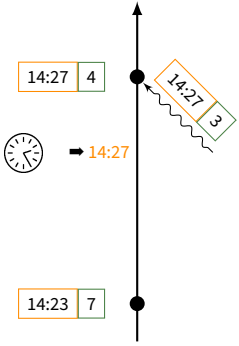


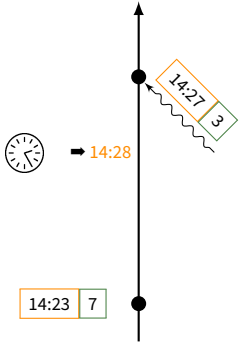
Example for hybrid clocks (cont.)



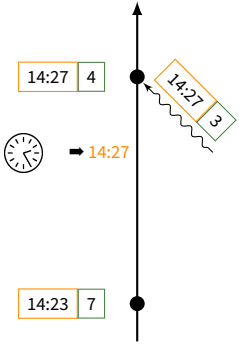


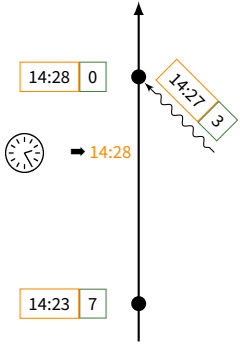
Example for hybrid clocks (cont.)



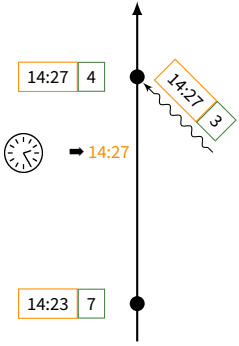


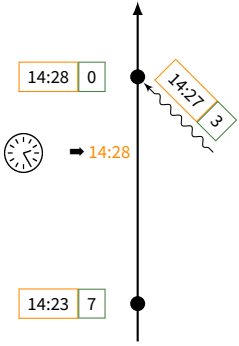
Example for hybrid clocks (cont.)

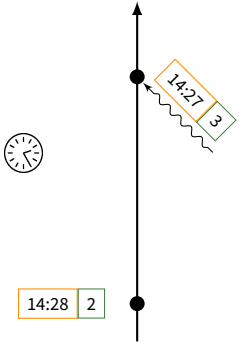




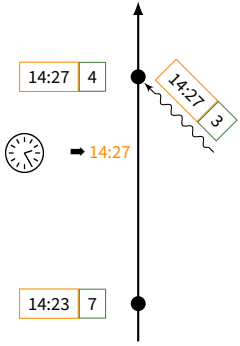
Example for hybrid clocks (cont.)

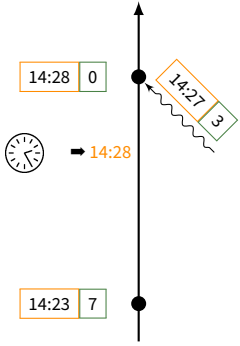


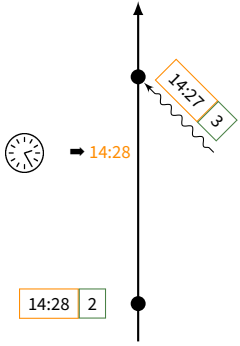




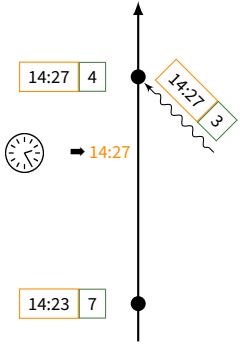
Example for hybrid clocks (cont.)

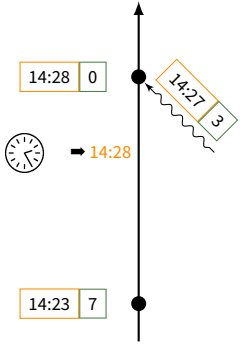


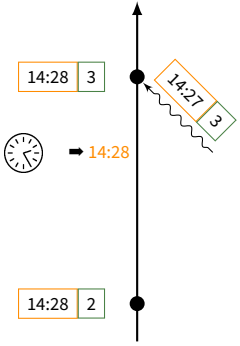




Example for hybrid clocks (cont.)







Error

- Distributed Clocks are often not completely shut down ➡ remaining components still run
- Problem in case of synchronization error:
  - **Assumption**: instead of

19/Nov/2015:14:23:56	5

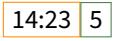
    a local clock is set to

19/Nov/2051:14:23:56	5

  - First message propagates the error and thereby infects all other nodes
  - **Result**: until 2051 this hybrid clock degenerates to a pure lamport clock.
- Without complete shutdown of the distributed system this error is **not** correctable.

Epochs

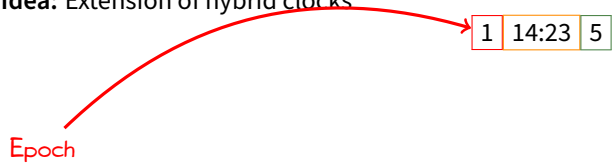
- **Idea:** Extension of hybrid clocks



- In case a correction is necessary, simply the epoch is incremented
  - Allows a “Reset”

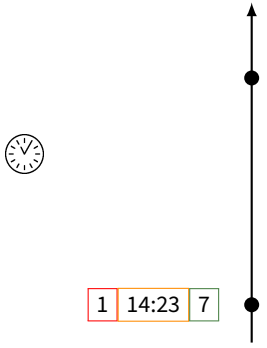
Epochs

- **Idea:** Extension of hybrid clocks

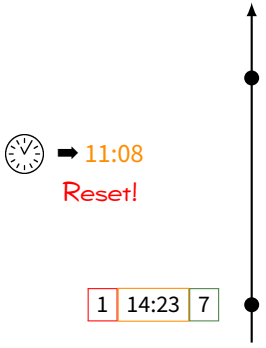


- "Epoch" is the most significant information
  - It is taken into account for the computation of the maximum
- In case a correction is necessary, simply the epoch is incremented
  - Allows a "Reset"

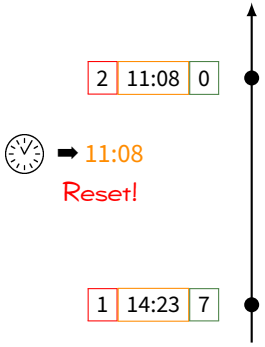
Reset and Epochs



Reset and Epochs



Reset and Epochs



- Causality is maintained
- User can still read extended hybrid clocks

Discussion

- Hybrid clocks fulfill the clock condition (as did Lamport clocks) ➡ Causality is respected with these clocks
- **Problem:** The timestamp does not contain information whether two events are causally dependent
  ➡ for this the reversal of the clock condition would have to be true

7.4 Vector Clocks

- Assume that there is a clock that makes an equivalence from the implication in the clock condition.

Strong clock condition

$$a \prec b \Leftrightarrow C(a) < C(b)$$

- With such a clock the timestamps can be used to derive the dependency of events.
  - $C(a) < C(b) \Rightarrow a \prec b$
  - $C(a) > C(b) \Rightarrow b \prec a$
  - $\neg\left((C(a) < C(b)) \vee (C(b) < C(a))\right) \Rightarrow a \parallel b$

- **Hints**:
  - For all three equations the reverse is also true.
  - $\neg\left((C(a) < C(b)) \vee (C(b) < C(a))\right)$ can be abbreviated to $C(a) \parallel C(b)$

How does one realize such a clock?

Vector clocks by MATTERN and FIDGE, 1988

- Each process P_i maintains a vector timestamp $\vec{V}_i$ consisting of n counters
- Initially all vector timestamps are null-vectors
- If there is an event in process P_i then this process increments the ith component of its vector
- With each message from P_i the **new** version of $\vec{V}_i$ is also sent
- If P_i receives a message with vector timestamp $\vec{T}$ it uses the componentwise maximum of $\vec{V}_i$ and $\vec{T}$ to construct the **new** version of $\vec{V}_i$

- This approach is obviously related to the causal order (➡ chapter 4)

Vector clocks – Operations

- Componentwise maximum of two vectors:

$$\max(\vec{V}_i, \vec{V}_j) \stackrel{def}{=} (\max(v_{i,1}, v_{j,1}), \ldots, \max(v_{i,n}, v_{j,n}))^T$$

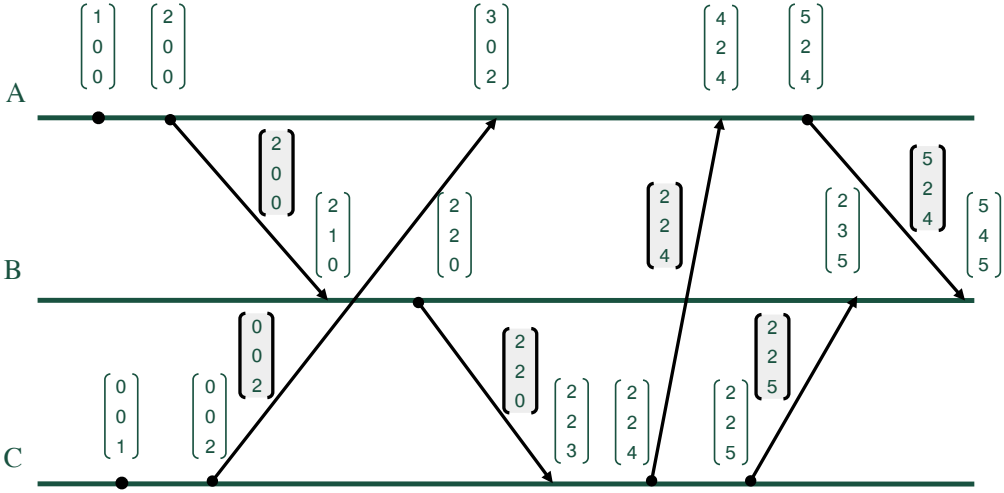
- Componentwise comparision of two vectors:

$$\vec{V}_i < \vec{V}_j \Leftrightarrow \vec{V}_i \neq \vec{V}_j \wedge \forall k, 1 \leq k \leq n, v_{i,k} \leq v_{j,k}$$

- Vector clocks define just a **partial** order

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

Vector clocks – Example



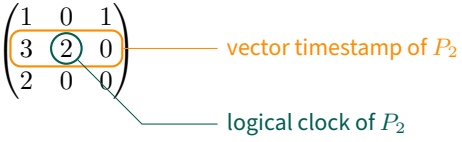
Connection between timestamps

- $R(e)$: real (global) time of event e
- $L(e)$: lamport time of e
- $V(e)$: vector time of e

$$\begin{array}{ccc} R(a) < R(b) & \Leftarrow & V(a) < V(b) \\ \Uparrow & \not\nearrow & \Downarrow \\ a \prec b & \Rightarrow & L(a) < L(b) \end{array}$$

7.5 Matrix clocks

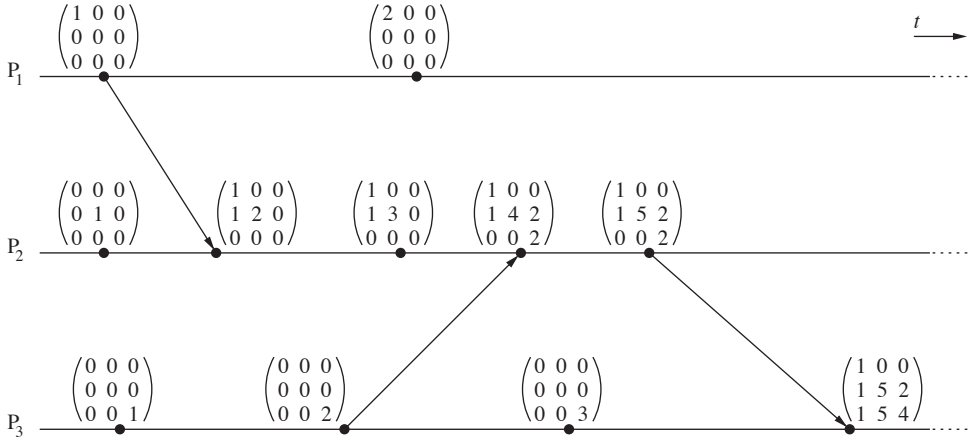
- There are more complex clocks: **Matrix clocks** (FISCHER and MICHAEL (1982))
- Each process P_i maintains a $n \times n$-matrix $\mathbb{M}_i$, representing its view on global distributed time of the system
- $m_{i,i,i}$ is the logical time of process P_i
- Row vector $\vec{m}_{i,i} = (v_{i,1}, \ldots, v_{i,n})$ contains all vector timestamps of process P_i
- Entry $m_{i,j,k}$ of matrix $\mathbb{M}_i$ of process P_i describes P_is knowledge about what P_j knows about the logic clock $m_{k,k,k}$ of P_k



Rules for generating matrix time

- Before the execution of an event in P_i the local logic clock is incremented: $m'_{i,i,i} = m_{i,i,i} + 1$
- By sending a message the matrix timestamp $\mathbb{M}T_i$ of the sender is also transmitted
- If P_i receives a message from P_j the matrix timestamp of P_i is updated:
  - $\forall k, 1 \leq k \leq n, m_{i,i,k} = \max(m_{i,i,k}, mt_{j,j,k})$
  - $\forall k, l, 1 \leq k, l \leq n, m_{i,k,l} = \max(m_{i,k,l}, mt_{j,k,l})$
  - $m_{i,i,i} = m_{i,i,i} + 1$

Matrix clocks, Example



And what is all this for?

- We can do everything thats possible with vector clocks ➡ strong clock condition
- Furthermore we have:

$\min_k(m_{i,k,l}) \geq t \Rightarrow$ Process P_i knows that each process P_k knows that the time in P_i is at least t

- Useful for
  - consistent error recognition in our system
  - elimination of redundant information e.g.:
    - in replicated data (e.g LYNCH and SARIN 1987)
    - retention of return values of RPC in case of errors (see chapter 3)

Literature

[Mat89] Mattern, F.: *Verteilte Basisalgorithmen*. Springer, 1989 , Kapitel 5: Virtuelle Zeit in verteilten Systemen

[Lam78] Lamport, L.: “Time, clocks, and the ordering of events in a distributed system”. *Commun. ACM*, 21(7)1978, 558–565

[Kul+14] Kulkarni, S. S., M. Demirbas, D. Madappa, B. Avva and M. Leone: “Logical Physical Clocks”, in *Principles of Distributed Systems*, edited by M. K. Aguilera, L. Querzoni and M. Shapiro, 17–32, Springer International Publishing