Communication

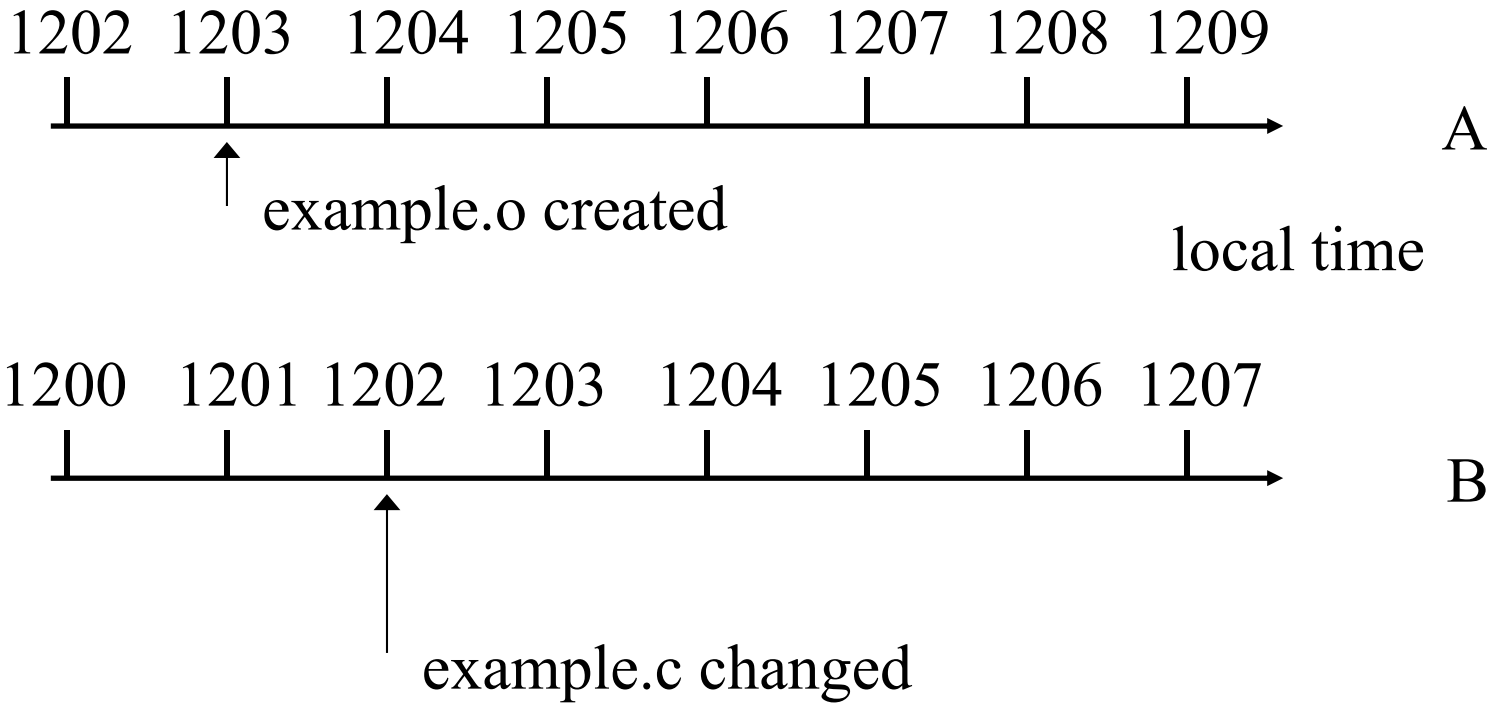
Clock Synchronization

Motivation

- Important to know the time a events occurred
- Consistent time often more important than precise time (group communication)
- Single processor:
  - global clock
- Distributed systems:
  - no common clock
  - different event succession (event horizons)
  - use of distributed algorithms

Examples

- Distributed make on a shared file system (NFS)



- Compiler is not called in spite of change

Time Keeping

- Time in COTS based on hardware ticks
  - inexact quartz elements (battery charge)
  - Environmental influence e.g. temperature
- Single processor systems: event succession consistent
- Distributed systems:
  - Absence of global physical clock
  - Local clock deviation

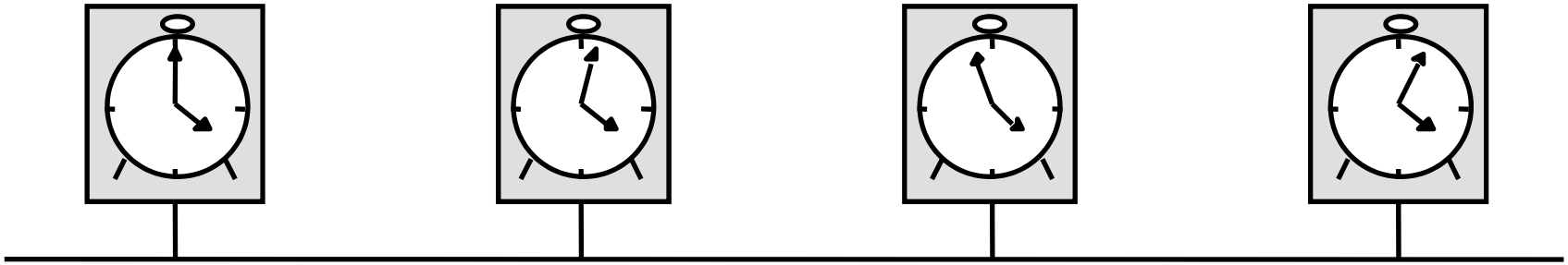
Time Measurement

- before 20th century astronomic calculation
  - solar day: time between 2 sun transitions
  - solar second: 1/84600 of solar day
- 1948 Atomic clock
  - 1 second = time for 9.192.631.770 atomic transition of the element caesium-133 (1967)
  - 1 atom second=average sun second in the year of introduction
- until 1972 Greenwich Mean Time (GMT)
- Bureau International de `Heure (BIH) determines international atomic time (TAI)

Coordinated **U**niversal **T**ime (UTC)

- International Atomic Time (TAI) is based on very accurate physical clocks (drift rate 10^{-13})
- Coordinated Universal Time (CUT) or Temps Universel Coordonné (TUC)
- Compromise abbreviation UTC by ITU organization
- International standard for time keeping
- Occasionally adjusted to astronomical time (UT1), if |UTC-UT1|>0.9s one-second change called a "leap second“
- It is broadcast from radio stations (WWV) on land and satellite (e.g. GPS)
- Signals from land-based stations are accurate to about 0.1-10 millisecond
- Signals from GPS are accurate to about 1 microsecond

Skew between computer clocks in a distributed system



Computer clocks are not generally in perfect agreement

- *Skew*: the difference between the times on two clocks (at any instant)
- Computer clocks are subject to *clock drift* (they count time at different rates)
- Clock *drift rate*: the difference per unit of time from some ideal reference clock
- Ordinary quartz clocks drift by about 1 sec in 11-12 days. (10^{-6} secs/sec).
- High precision quartz clocks drift rate is about 10^{-7} or 10^{-8} secs/sec

Synchronizing physical clocks

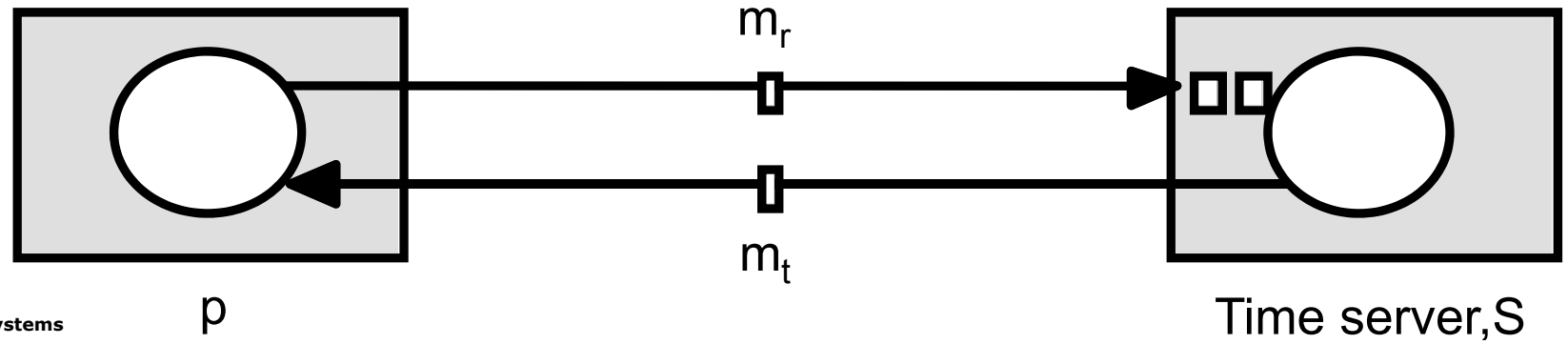
- Internal Synchronization:
  - $|C_i(t) - C_j(t)| < D$ for $D > 0$ and $i, j = 1 \ldots N$
  - the clocks C_i agree within the bound D
- External Synchronization:
  - $|S(t) - C_i(t)| < D$ for $D > 0$ and $i = 1..N$
  - the clocks C_i are accurate within the bound D with regard to a external reference S

Algorithms for clock synchronisation

- Cristian
- Berkeley
- NTP
- Lamport Logical Clocks
- Vector clocks

Cristian algorithm (1989)

- Time server S externally synchronized with UTC
- T_{round} is round trip time recorded by p
- T_{min} is an estimated minimum T_{round}
- Process *p* requests time in m_r and receives *t* in m_t from *S*
- *p* sets its clock to $t + T_{round}/2$
- Accuracy ± ($T_{round}/2$ - *min*) :
  - because the earliest time *S* puts *t* in message m_t is *min* after *p* sent m_r.
  - the latest time was *min* before m_t arrived at *p*
  - the time by *S*'s clock when m_t arrives is in the range [t+*min*, $t + T_{round}$ - *min*]



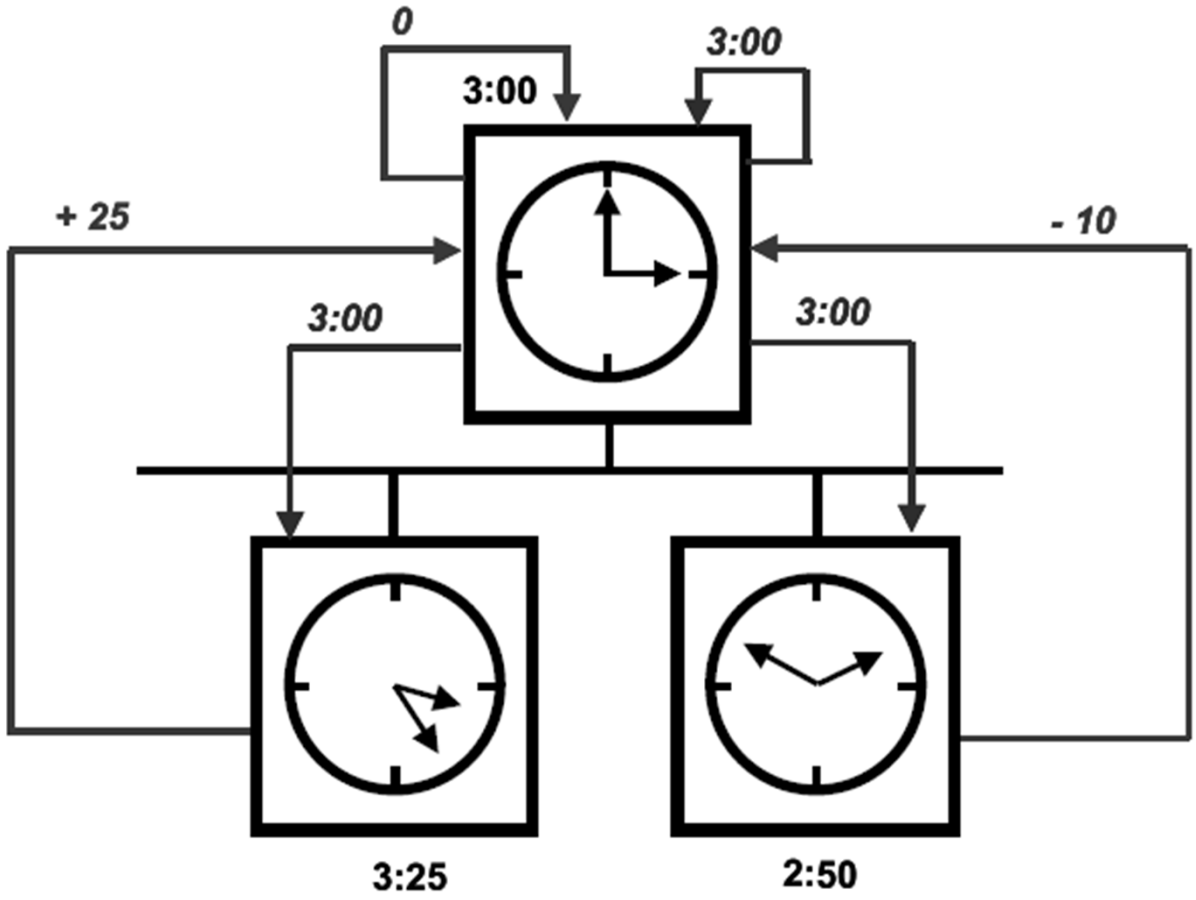
Problems with Cristian‘s algorithm

- Time must not go backwards
  - it doesn't deal with faulty time servers
- Possibly time jumps on large skews
- Single roundtrip are possibly inexact
  - multiple measurements with average forming
  - discard runaways

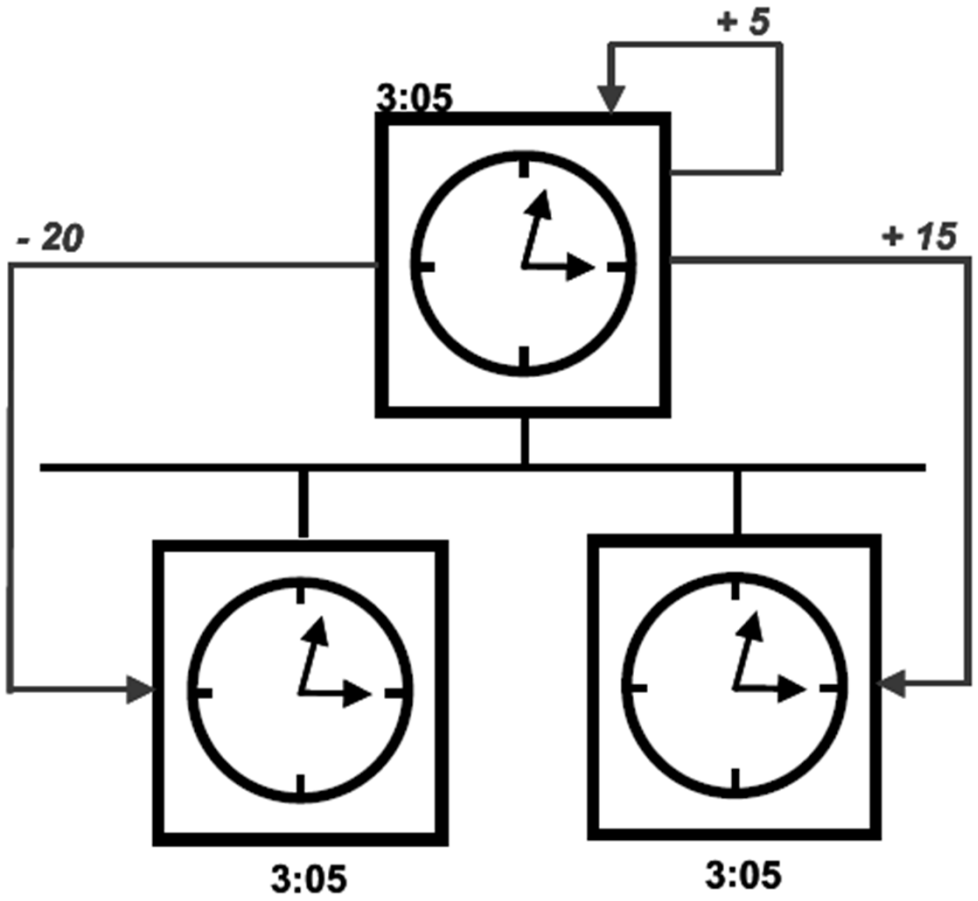
Berkeley algorithm (1989)

- Gusella and Zatti 1989
- An algorithm for internal synchronization of a group of computers
- A *master* polls to collect clock values from the others (*slaves*)
- The master uses round trip times to estimate the slaves' clock values
- It takes an average (eliminating any above some average round trip time or with faulty clocks)
- It sends the required adjustment to the slaves (better than sending the time which depends on the round trip time)

Berkeley algorithm (II)

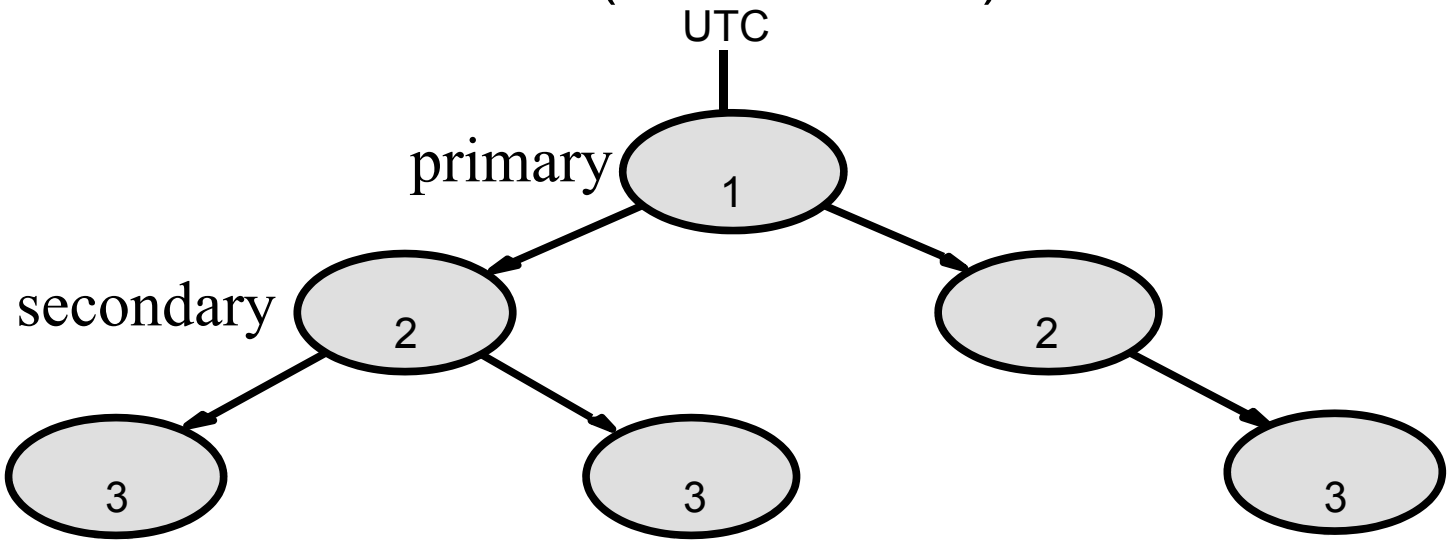


Berkeley algorithm (III)



Network Time Protocol (Mills 1985)

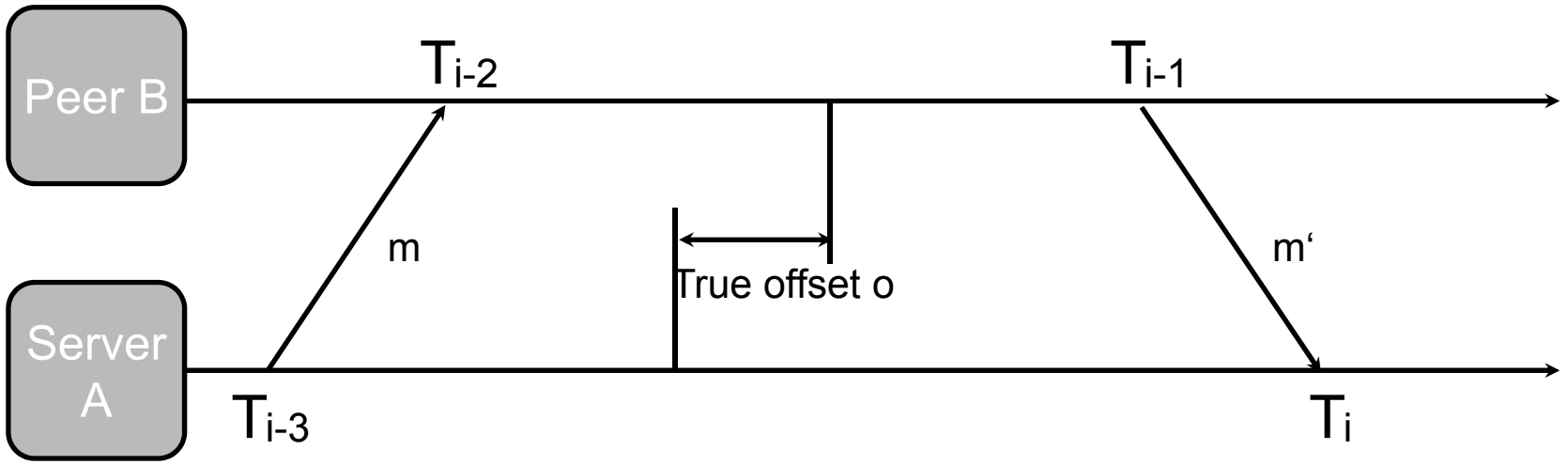
- A time service for the Internet - synchronizes clients to UTC
- Version 3 in RFC 1305, 1992 D.L. Mills
- Version 4: RFC 5905, 2010 D.L. Mills et. al.
  - www.ntp.org
- Hierarchical architecture (*stratum level*)



NTP Algorithm

- Exchange of timestamps to estimate offset o_i between the clocks at T_i and roundtrip delay $d_i=m+m'$; Errors due network delays dominate
- $T_{i-2}=T_{i-3}+m+o_i$ and $T_i=T_{i-1}+m'-o_i$, therefore $d_i=T_{i-2}-T_{i-3}+T_i-T_{i-1}$
- Estimated offset $o_i=(T_{i-2}-T_{i-3}+T_{i-1}-T_i)/2$
- $o <= T_{i-2}-T_{i-3}$ and $o >= T_{i-1}-T_i$ since $m, m' > 0$,

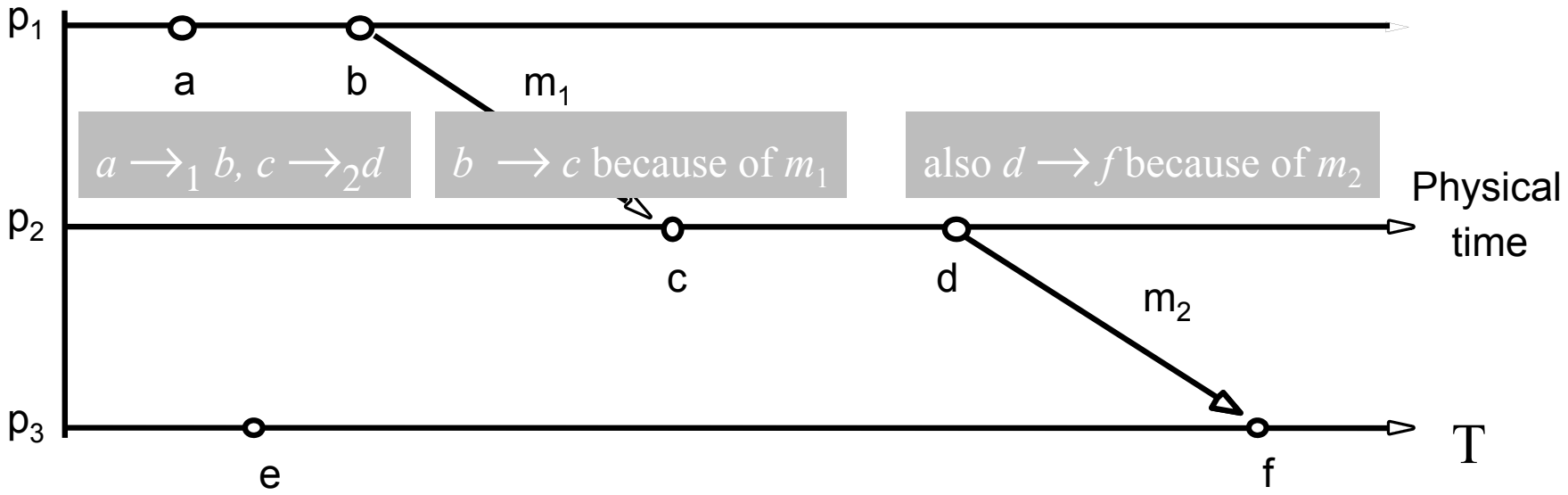
  leads to $\mathbf{o_i-d_i/2 <= o <= o_i+d_i/2}$



Logical time and logical clocks

- Lamport 1978
- Instead of synchronizing clocks, event ordering can be used

1. If two events occurred at the same process p_i (i = 1, 2, … N) then they occurred in the order observed by p_i, that is $\rightarrow_i$
2. when a message, m is sent between two processes, *send(m)* happened before ($\rightarrow$) *receive(m)*



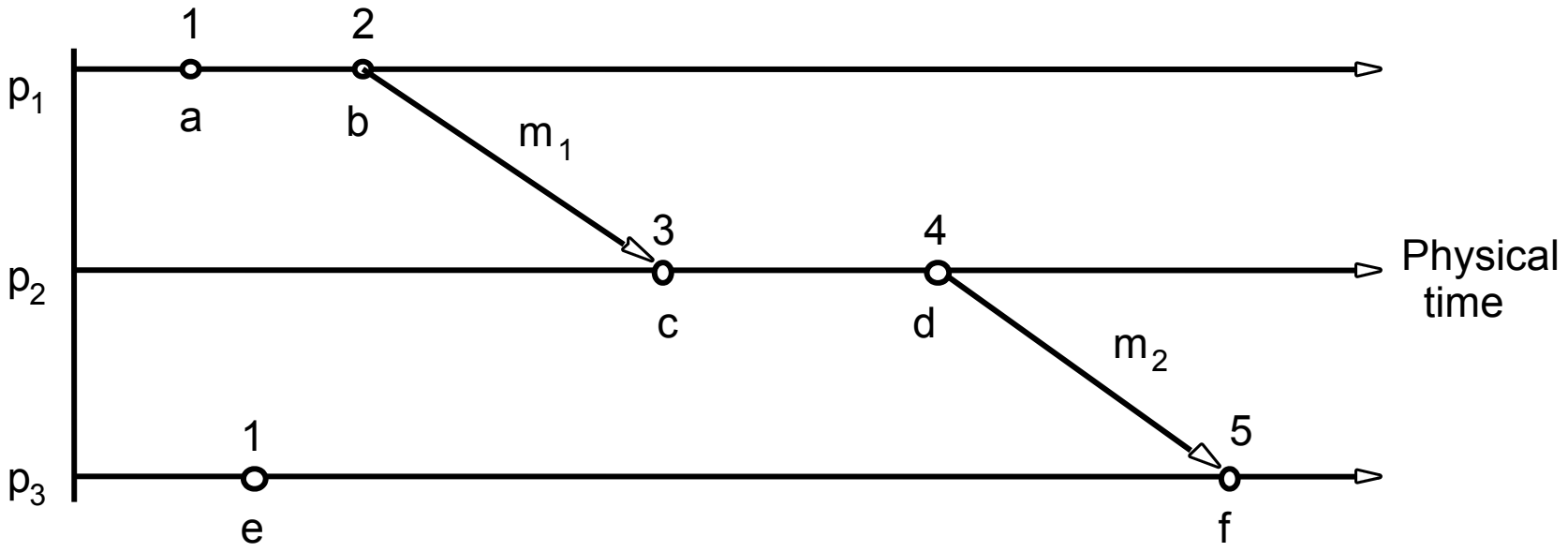
- a and e are no related by $\rightarrow$, a and e are *concurrent* $\Rightarrow$ a||e

Lamport Logical Clocks (1978)

- is a monotonically increasing counter L_i i=1...N
- $L_i(t=0):=0$
- for every event *e* exists a local $L_i(e)$ and a common L(*e*) Lamport timestamp
  1. $L_i:=L_i+1$ before a event is issued at p_i
  2.1 $t=L_i$ is piggybacked on each sent message *m* of p_i
  2.2 p_j computes $L_j:=\max(L_j,t)$ on receiving (*m*,*t*) and applies 1. before timestamping the receipt of *m*
- $e \rightarrow e' \Rightarrow L(e)<L(e')$

Lamport Timestamps

- L(b)>L(e) but b || e



Vector Clocks

- Mattern 1989 and Fidge 1991
- To overcome the shortcoming: L(e)<L(e‘) does not imply e$\rightarrow$e‘ ´with Lamport
  - Vector clock V_i at process p_i is an array of N integers
  - VC1: initially $V_i[j] = 0$ for $i, j = 1, 2, \ldots N$
  - VC2: before p_i timestamps an event it sets $V_i[i] := V_i[i] + 1$
  - VC3: p_i piggybacks $t = V_i$ on every message it sends
  - VC4: when p_i receives (m,t) it sets $V_i[j] := \max(V_i[j], t[j])$ $j = 1, 2, \ldots N$

Vector Timestamps

- V=V‘ if V[j]=V‘[j] for j=1...N
- V≤V‘ if V[j]≤V‘[j] for j=1...N
- V<V‘ if V≤V‘ ∧ V≠V‘

Summary: Clock Synchronization

- Accurate timekeeping is important for distributed systems.
- Algorithms (e.g. Cristian's and NTP) synchronize clocks in spite of their drift and the variability of message delays.
- For ordering of an arbitrary pair of events at different computers, clock synchronization is not always practical.
- The happened-before relation is a partial order on events that reflects a flow of information between them.
- Lamport clocks are counters that are updated according to the happened-before relationship between events.
- Vector clocks are an improvement on Lamport clocks,

References

- Distributed Systems: Concepts and Design, Addison Wesley, 2001, 2005
- David L. Mills: Network Time Protocol V3, RFC 1305 www.ntp.org, 1992
- J. Burbank et al: Network Time Protocol V4, RFC 5905 www.ntp.org, 2010
- David L. Mills: Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI, RFC 2030, 1996
- Bureau International des Poids et Mesures www.bipm.fr
- Lesslie Lamport: Time, Clocks, and the Ordering of Events in a Distributed System, Comm. of the ACM Vol. 21 Numb. 7, 1978