Chapter 11

Time

Events and Time

send/receive messages, change of machine state (change of memory content)

- An event is the occurrence of a single action that a process carries out as it executes.
- It is important that events are timestamped accurately:
  - NFS client checks its cache validity from timestamps of files on the server.
  - Merchant's computer and bank's computer timestamp events in an e-commerce transaction for auditing.
  - Etc.

Clock (1)

- Each computer has its own internal clock; there is no single global clock for every computer.
  - Clock counts oscillations in a quartz crystal at a definite frequency, divides the count, and stores the result in a counter register.
  - At real time, t, the OS reads the time on the computer's hardware clock $H_i(t)$.
  - It calculates the time on its software clock $C_i(t)$, e.g. a 64 bit number giving nanoseconds since some base time
- In general, clocks are not completely accurate; frequencies of oscillation differ:
  - Physical variation of quartz
  - Temperature

Clock (2)

- **Clock skew** – Difference between the time on two clocks (at any instant).
- **Clock drift** – Clocks count time at different rate.
- **Clock drift rate** – 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). seconds per second
  - High precision quartz clocks drift rate is about 10^{-7} or 10^{-8} secs/sec.
- Even if clocks on all computers in a DS are set to the same time, their clocks will eventually vary quite significantly unless corrections are applied.

$$\frac{10{,}000{,}000}{60 \times 60 \times 24 \times 365} \qquad \frac{4{,}000}{6 \times 6 \times 365} = 100 \text{ days}$$

External Clock

- Computer clocks can be synchronized with highly accurate external clocks.
- The most accurate are atomic clocks; they use atomic oscillators.
  - Drift rate is about 10^{-13} secs/sec.
- But the time units we use (e.g. seconds, days, years) are astronomical time.
  - They are based on rotation of the Earth on its axis and about the Sun.
  - Tidal friction within the Earth can increase or decrease rotation period.
- Adjustment to atomic time will keep it in step with astronomical time.
- **Coordinated Universal Time (UTC)** is the standard atomic time with a leap second inserted or deleted occasionally.
  - Timing signals are broadcast from land-based radio stations on shortwave frequencies and satellites (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.
  - Computers with receivers can synchronize their clocks with these timing signals.

Rotation is not uniform. Atomic time is more uniform.

NIST+USNO

Leap second is added to the last minute of 30 Jun or 31 Dec every 1-2 years, making that last minute have 61 seconds.

Clock Correctness

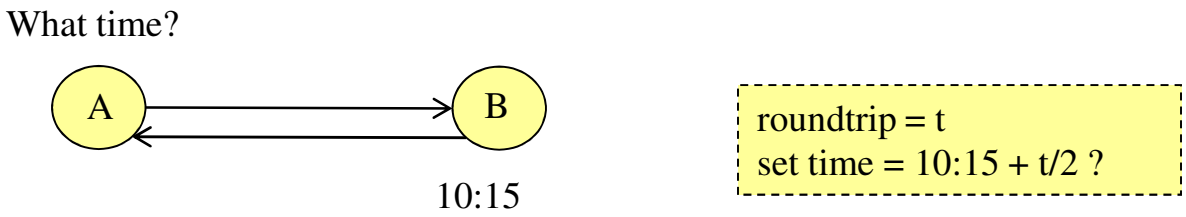
- Clock is faulty if it does not keep to correctness conditions.
- A hardware clock *H* is correct if its drift rate is within a bound $\rho > 0$ (e.g. 10^{-6} secs/sec (a value supplied by manufacturer)).
  - This forbids jump in the value of hardware clock.
  - A clock with very low battery is faulty; it has a large drift rate (c.f. arbitrary failure).
- Weaker condition of **monotonicity**; clocks will advance only.
  - E.g. required by UNIX *make* facility
  - A clock that stops ticking is faulty (c.f. crash failure).
  - A clock with Y2K bug has date changed from 31 Dec 1999 to 1 Jan 1900 (c.f. arbitrary failure).

Synchronizing physical clocks

- Synchronizing physical clocks has to obey clock correctness.
- *External synchronization* – Clocks C_i are synchronized with an authoritative external source of time.
- *Internal synchronization* – Clocks C_i are synchronized with one another, even though they are not necessarily synchronized with an external source.
- Clocks do not have to be accurate to be correct if the goal is internal synchronization, rather than external.

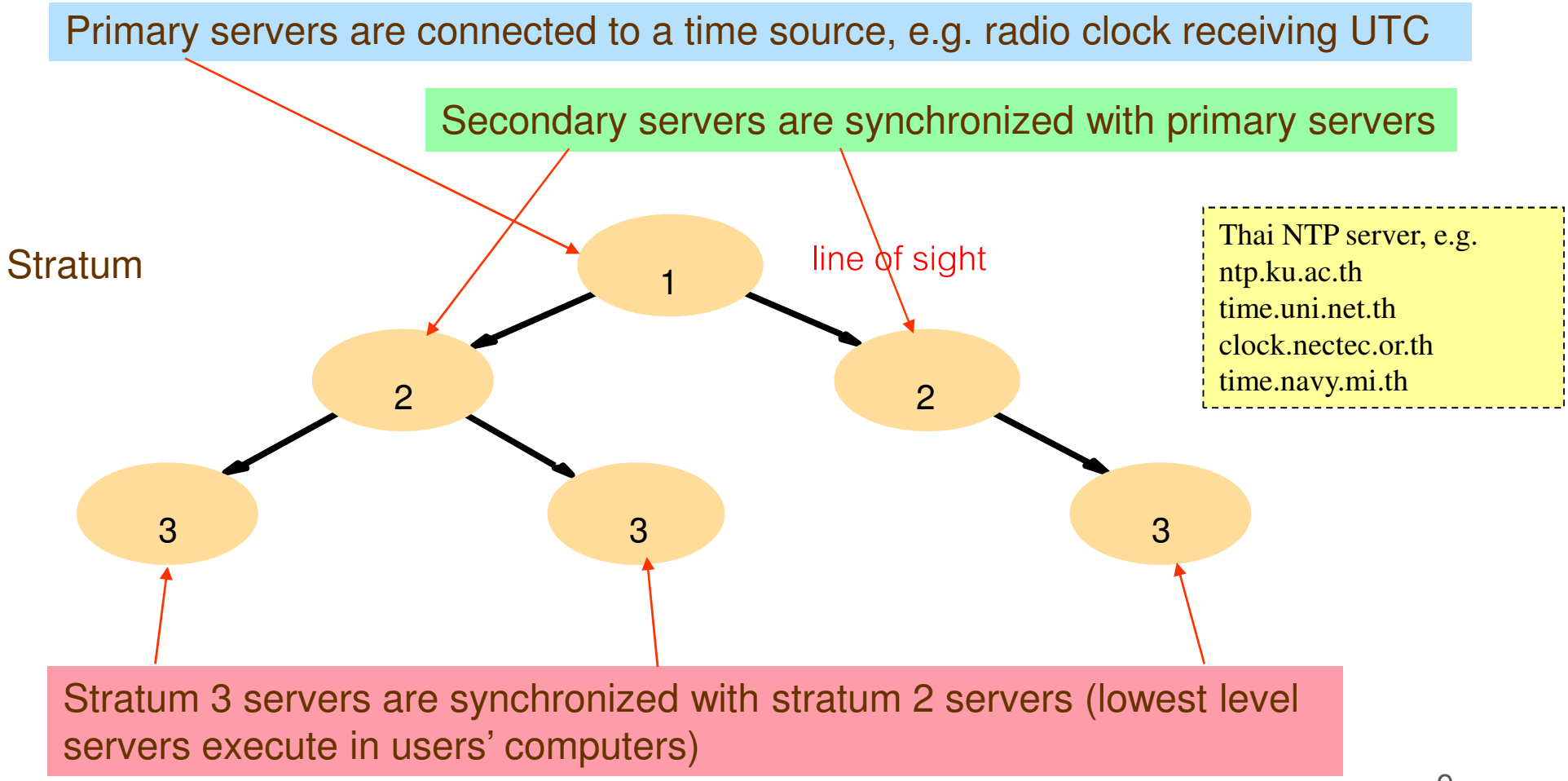
Network Time Protocol (NTP)

- A time service for the Internet that synchronizes clocks to UTC despite their drift and variability of message delays.
- It employs statistical techniques to calculate offset between two clocks and total transmission time for NTP request and reply messages.
- There are redundant servers and redundant paths between servers for reliability.
- The service scales to large number of clients and servers.
- NTP server authenticates time sources and return addresses of messages sent to it.



NTP Synchronization Subnet

- A logical hierarchy of NTP servers



Synchronization of NTP Servers

- Clocks in lower-level strata are less accurate than those in higher-level strata as errors are introduced at each level of synchronization.
- If failure occurs,
  - a primary that loses its UTC source can become a secondary.
  - a secondary that loses its primary can use another primary.
- Three modes of synchronization:
  - Multicast – A server within a high speed LAN periodically multicasts time to others which set their clocks assuming a small delay.
    - Not very accurate but sufficient for many purposes
  - Procedure call – A server accepts requests from other computers and replies with its current clock reading. sync on demand
    - Used when higher accuracy than multicast is required or if no hardware multicast.
  - Symmetric - Pairs of servers exchange messages containing time information.
    - Used in higher level strata where very high accuracy is needed.
  - All modes use UDP.

Logical time

- Since we cannot synchronize clocks perfectly, we cannot in general use physical time to find out the ordering of events that occur at different computers.
- Logical time is an alternative; it gives ordering of events, rather than exact timestamp of occurrence.
- **Happened-before relation** is a *partial order* on events that reflects a flow of information between them, and is defined as

HB1: If $\exists$ process p_i : $e \rightarrow_i e'$, $e \rightarrow e'$ — intraprocess

HB2: For any message m, $send(m) \rightarrow receive(m)$ — interprocess

HB3: If $e \rightarrow e'$ and $e' \rightarrow e''$ then $e \rightarrow e''$ — transitive

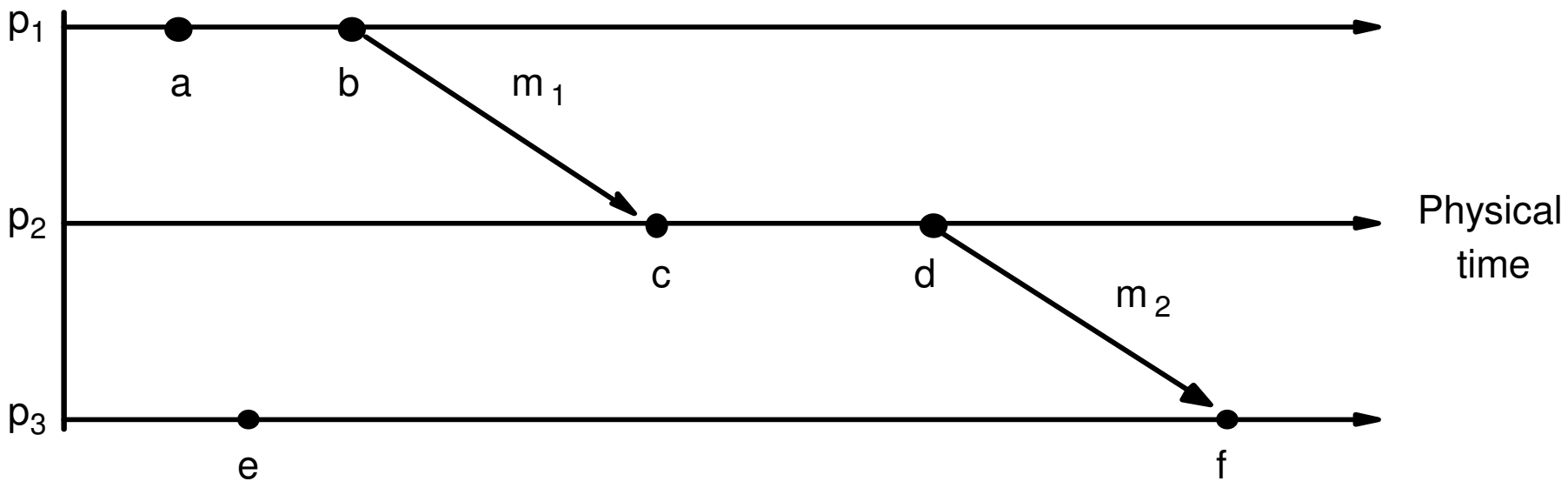
- Happened-before relation is a relation of **causal ordering** or **potential causal ordering.**

causal - cause-effect

casual - simple, easy

Timing Diagram

Logical Ordering



$a \rightarrow b$ (at $p1$) $c \rightarrow d$ (at $p2$)

$b \rightarrow c$ because of $m1$

$d \rightarrow f$ because of $m2$

also $a \rightarrow f$

Not all events are related by $\rightarrow$
Consider a and e (different processes and no chain of messages to relate them),
they are not related by $\rightarrow$; they are said to be concurrent; write as $a \parallel e$

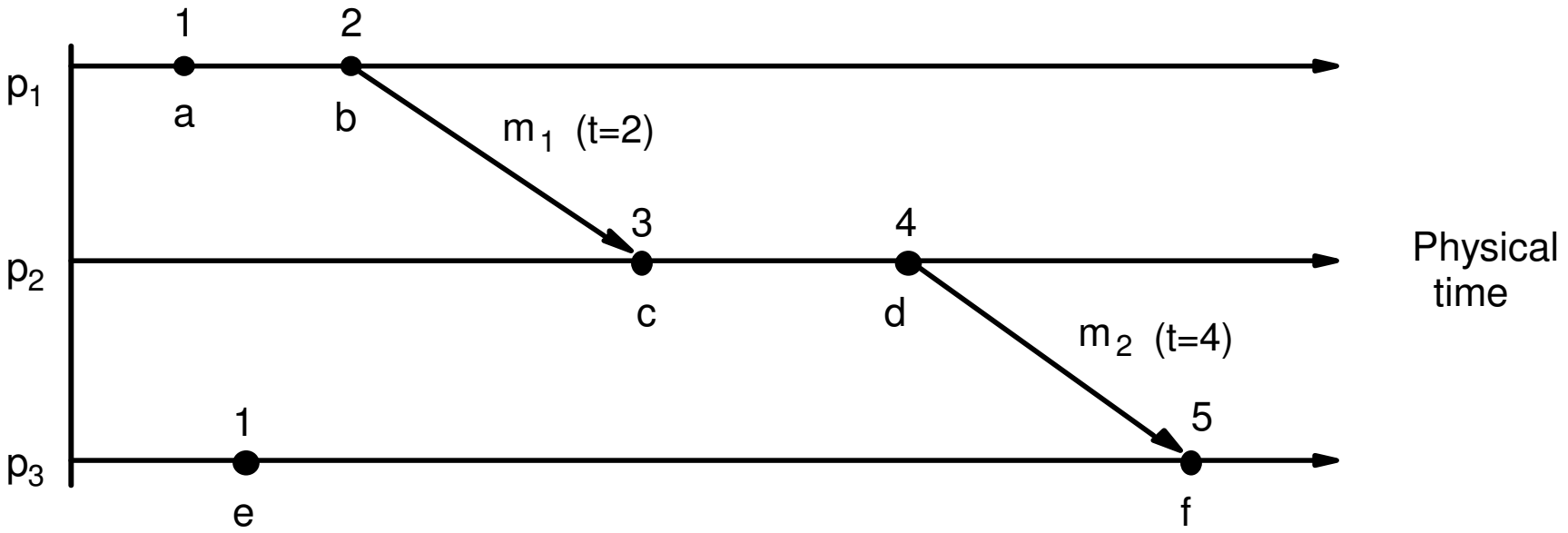
Logical Clock or Lamport Clock (1)

Leslie Lamport

- Logical clock is a monotonically increasing software counter that is updated according to the happened-before relationship between events.
  - It need not relate to a physical clock.
- Each process p_i has a logical clock L_i which can be used to apply logical timestamps to events.

LC1: L_i is incremented by 1 before each event at process p_i
LC2: (a) when process p_i sends message m, it piggybacks $t = L_i$
(b) when p_j receives (m,t) it sets $L_j := max(L_j, t)$ and applies LC1 before timestamping the event *receive* (m)

Logical Clock (2)



Each of p1, p2, p3 has its logical clock initialized to zero.
The clock values are those immediately after the event, e.g. 1 for a, 2 for b.

For *m*1, 2 is piggybacked and *c* gets *max*(0,2)+1 = 3

$e \rightarrow e'$ implies $L(e) < L(e')$

The converse is not true, that is $L(e) < L(e')$ does not imply $e \rightarrow e'$
e.g. $L(b) > L(e)$ but $b \parallel e$

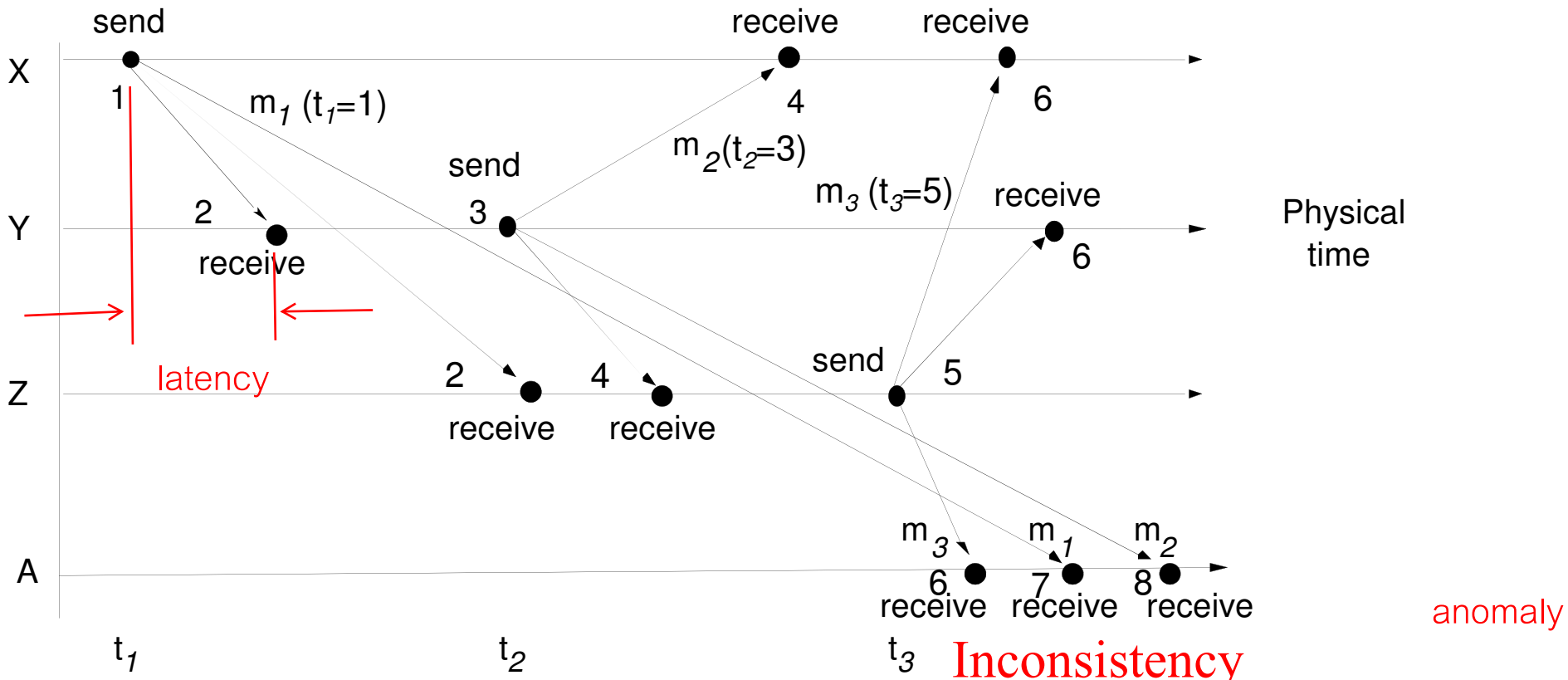
Ordering of events by logical time (1)

- User X sends a message ‘Meeting’ to users Y, Z, A.
- User Y reads and sends ‘Re: Meeting’.
- User Z reads X’s and Y’s messages and sends ‘Re: Meeting’.
- User A could see all messages in the wrong order:

Inbox:		
Item	From	Subject
23	Z	Re: Meeting
24	X	Meeting
25	Y	Re: Meeting

- Messages from X, Y, and Z have time order in the real world.
- If the clocks on X’s, Y’s, and Z’s computers could be synchronized, each message could carry a timestamp based on the local computer’s clock.
  - Messages could be displayed for A in the correct time order of the real world.
- Since clocks cannot be synchronized perfectly, we may use logical time.

Ordering of events by logical time (2)



Totally ordered logical clocks (1)

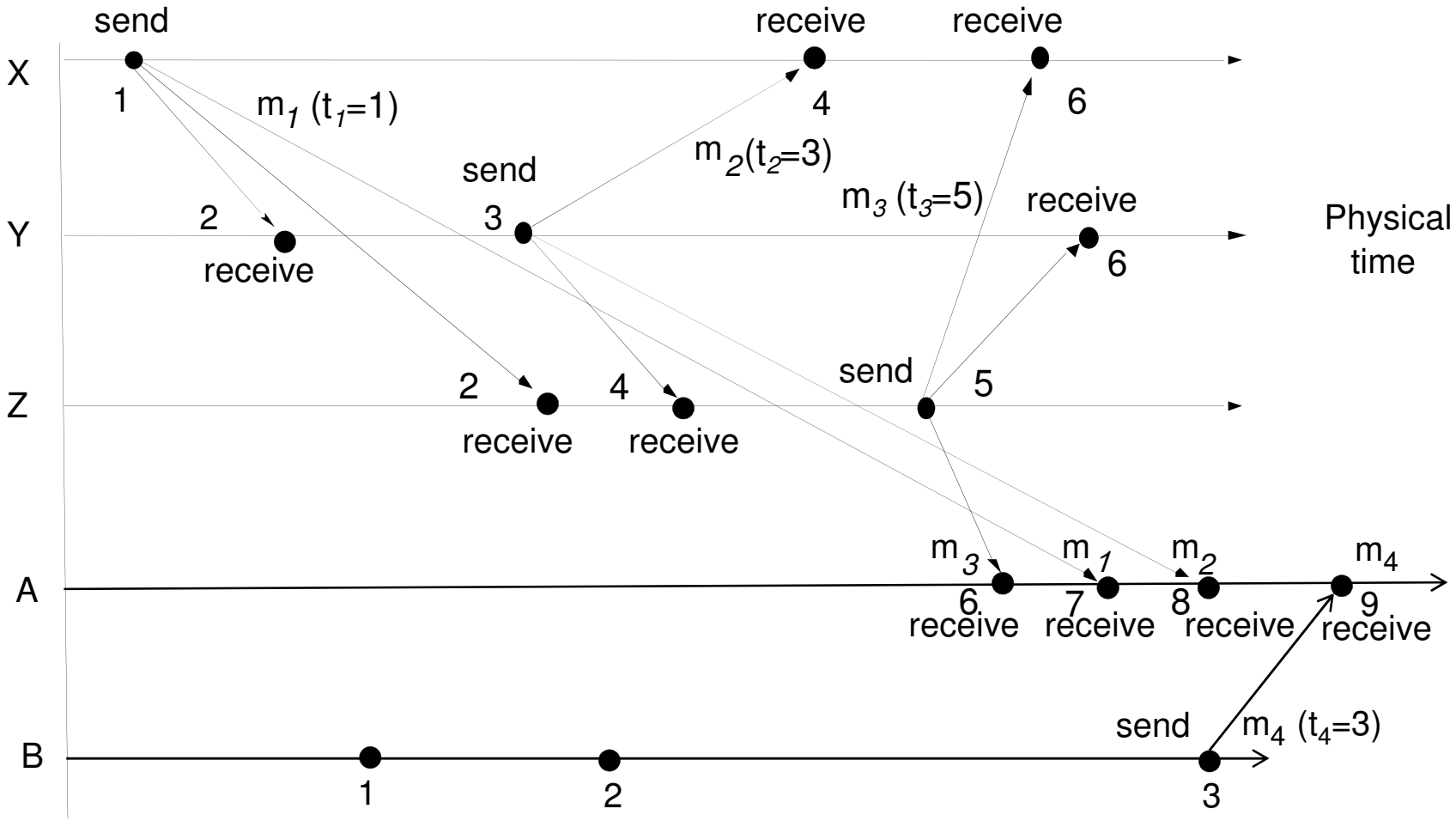
- We can create a **total order** on events (i.e. order all distinct events whether or not they are related by happened-before relation).
- The identifiers of the processes at which the events occur are taken into account.
- Timestamp for an event e at process p_i is (T_i, i).

HostID + PID

$(T_i, i) < (T_j, j)$ if and only if either $T_i < T_j$, or $T_i = T_j$ and $i < j$

tie breaker

Totally ordered logical clocks (2)



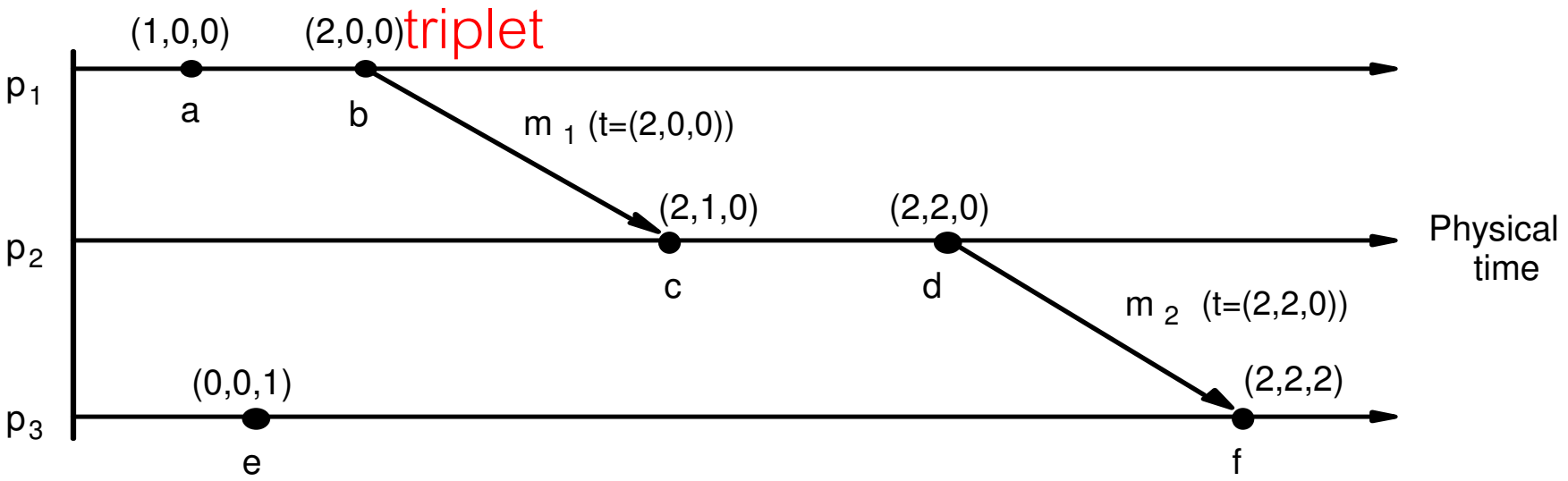
If B < Y: m1, m4, m2, m3
If Y < B: m1, m2, m4, m3

Vector Clock (1)

- An improvement on Lamport clock.
  - We can tell whether two events are ordered by happened-before relation or are concurrent, by comparing their vector timestamps.
  - It is applied in schemes for replication of data, e.g. Gossip (you will see that later in the course).
- Vector clock V_i at process p_i in a system of N processes 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 $V_i[j] = \max(V_i[j], t[j])$, for $j = 1, 2, \ldots N$
(This is a *merge* operation; And before next event, apply VC2 to its own element)

Vector Clock (2)



$V_i[i]$ is the number of events that p_i has timestamped.
$V_i[j]$ ($j \neq i$) is the number of events at p_j that p_i has been affected by.
(p_j may have timestamped more events but no info about them has flowed to p_i yet.)

Comparison of vector timestamps:
$V = V'$ *iff* $V[j] = V'[j]$ for $j = 1, 2, \ldots N$.
$V \leq V'$ *iff* $V[j] \leq V'[j]$ for $j = 1, 2, \ldots N$.
$V < V'$ iff $V \leq V' \wedge V \neq V'$.

$e \rightarrow e'$ *iff* $V(e) < V(e')$

$V(a) < V(f)$ implies $a \rightarrow f$

$a \parallel e$ as neither $V(a) \leq V(e)$ *nor* $V(e) \leq V(a)$