Sistemas Distribuídos

Synchronization

António Rui Borges

Summary

- *Time concepts*
  - *Global time*
  - *Local time*
  - *Logical time*
- *Adjustment of local time*
  - *Problem characterization*
  - *Cristian method*
  - *Berkeley algorithm*
  - *Network Time Protocol*
- *Logical clocks*
  - *Scalar logical clock*
    - *Total ordering of events*
  - *Vector logical clocks*
- *Suggested reading*

Time concepts

Physical phenomena, and in particular human activity, take place in space and time. *Time* is the key element on event characterization, on their ordering and on determining possible causal relations that may exist between pairs, or groups, of events.

Time itself is thought of in multiple ways vis a vis how the *observer* faces reality

- *global time* – time as perceived by an external observer
- *local time* – time as perceived by each of the observed entities
- *logical time* – time as perceived by the flow of information.

Global time - 1

Time can not be directly measured.

Its measurement is done by observing the periodical motion of well-defined objects, taking advantage of the intimate connection of *time* with *space*

- *astronomical time* – based on the motion of astral bodies in heavens: specifically, the circular-like motion of Earth around the Sun (each cycle represents a *solar year*) and the Earth rotating motion (each cycle represents a *solar day*); the *solar day* is successively divided into hours, minutes and seconds (one *day* comprehends 24 hours, one hour 60 minutes and one minute 60 seconds); the *solar second*, while standard unit for time measurement, is defined as the fraction of 1/86400 of the solar day; and the *solar year* as being approximately 365 days and 6 hours

Global time - 2

- *atomic time* – the periodic motion of astral bodies is not constant enough to be used as standard when one considers very long time intervals; research carried out in the past decades has shown that the Earth rotating motion has become slower as time goes by due to the friction produced by the tides and the dragging effect of the atmosphere; furthermore, the Earth rotating motion is prone to small oscillations in its angular speed due to, is thought, turbulence at the core; for all these reasons, the *standard second* was redefined when atomic clocks came into being; present day definition states the *second* to be equal to *the duration of 9 192 631 770 periods of the radiation corresponding to the transition between two hyperfine levels of the state of minimum energy of the atom cesium 133, at rest and at the temperature of 0 K*.

Global time - 3

International Atomic Time (TAI) represents the practical standard computed from the weighted average of time measured by more than 200 atomic clocks located in national institutions around the world. This task is coordinated by the International Institute of Weights and Measures (BIPM), through its International Bureau of the Hour (BIH).

The duration of the *standard second* was established in order to ensure its coincidence with the *solar second* at the time of its introduction, January 1, 1958.

In January 1, 1977, a correction was introduced to the convention, based on general relativity, to minimize the time dilation effect produced by variations of the Earth gravitational field (the atomic clocks, used to compute the average, are located at different altitudes and, thus, tick at slightly different rates).

Global time - 4

Coordinated Universal Time (UTC), based on the *international atomic time*, constitutes the main standard that sets the time to human activities. The system is based on *standard seconds* and is kept as close as possible to the *astronomical time* by adding or subtracting (the latter has not happen yet) individual seconds to compensate for the slowing down, and occasional irregularities, of the Earth rotating motion.

In the UTC time scale, the *second* and its submultiples are always constant; its multiples, *days*, *hours* and *minutes*, however, are not. From time to time, in order to keep it in pace with the *astronomical time*, a second is added or subtracted to the last minute of a day, typically in the last day of June or December.

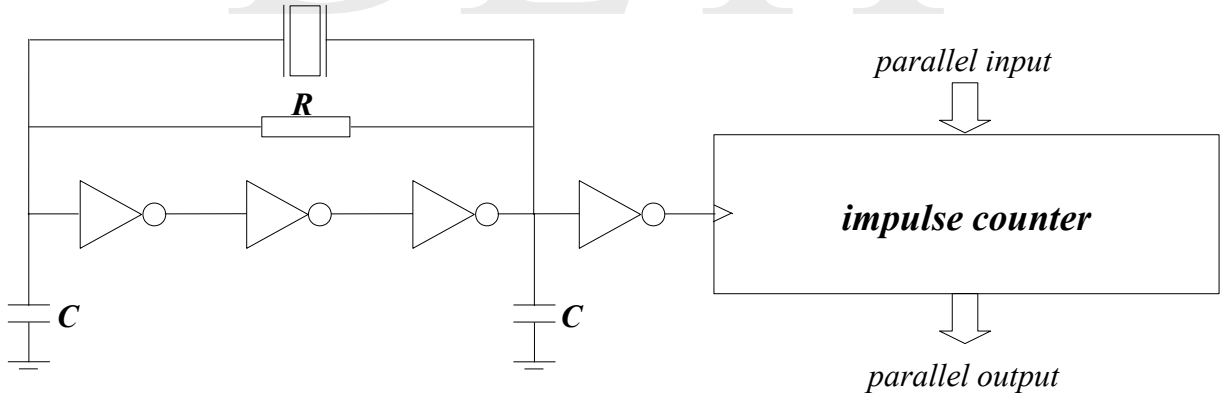
UTC time is made available from multiple sources, with different uncertainties. Among the most important, are

- short wave radio emitters: ±10 ms
- geostationary satellite systems (GEOS, GPS): ±0,5 ms
- internet (NTP): ±50 ms.

Local time – 1

The *clock* of a computer system consists of two elements: an oscillator circuit, controlled by a quartz crystal, and an impulse counter. The counting value at a given instant may be obtained by reading the *parallel output* and the counter may be set to a given state by feeding the *parallel input*.

The counting can be converted to a *time* provided an origin is defined (the convention in Unix, for instance, matches the origin to 0h 0m 0s of January 1, 1970).

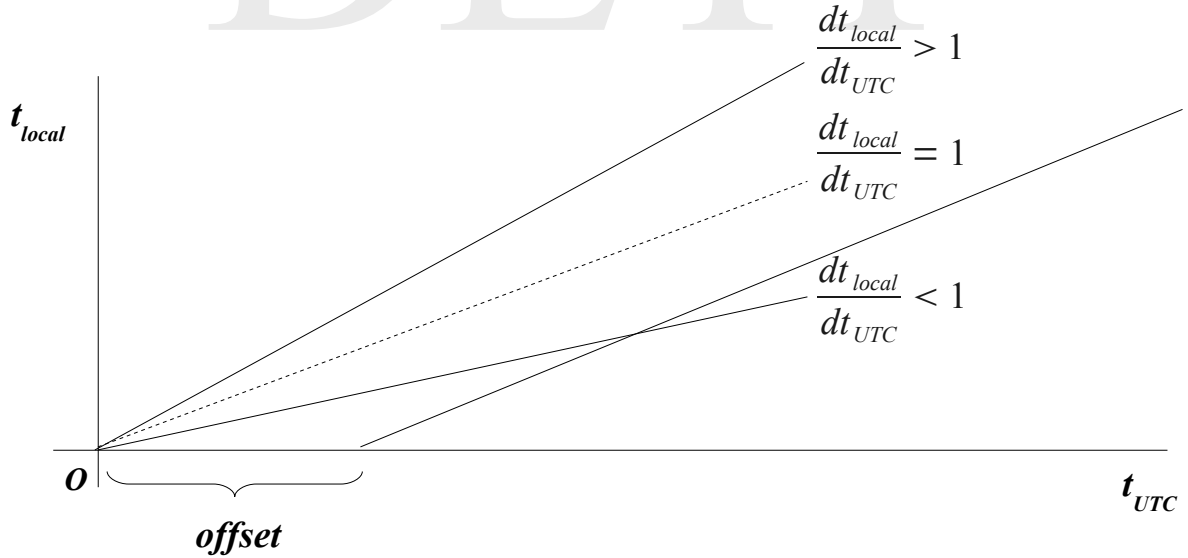


Local time – 2

The *clock* may display an incorrect time due to two different factors

- *offset* – the counting value differs from the correct value by a fixed number of impulses (error on defining the origin)
- *drift* – the oscillator frequency moves from the nominal one, changing its value erratically as time goes by due to the environment conditions, such as temperature and humidity (error on the counting rate).

Typically, an oscillator, controlled by a quartz crystal, has a *drift* of the order of magnitude of one part in 10^6 per second.



Time adjustment – 1

The problem of synchronizing the local clocks of the computer systems that make up the processing nodes of a parallel machine, can be thought of in two different ways

- *external synchronization* – given a known UTC source, $S(t)$, and a maximum interval of tolerated uncertainty, Δ, among the local clocks $Ck_i(t)$, with $i = 0,1,\ldots,$ N-1, and the UTC source $S(t)$, ensure for all time t that

$$\forall_{i \in \{0,1,\cdots,N-1\}} \; |S(t) - Ck_i(t)| < \Delta$$

- *internal synchronization* – given a maximum interval of tolerated uncertainty, Δ, among the local clocks $Ck_i(t)$, with $i = 0,1,\ldots,$ N-1, ensure for all time t that

$$\forall_{i,j \in \{0,1,\cdots,N-1\}} \; |Ck_i(t) - Ck_j(t)| < \Delta \quad .$$

Time adjustment – 2

One assumes that the processing nodes of the parallel machine are connected by some interconnection topology and that communication among them is carried out through message passing, with a finite transmission time, but without an upper limit, that is,

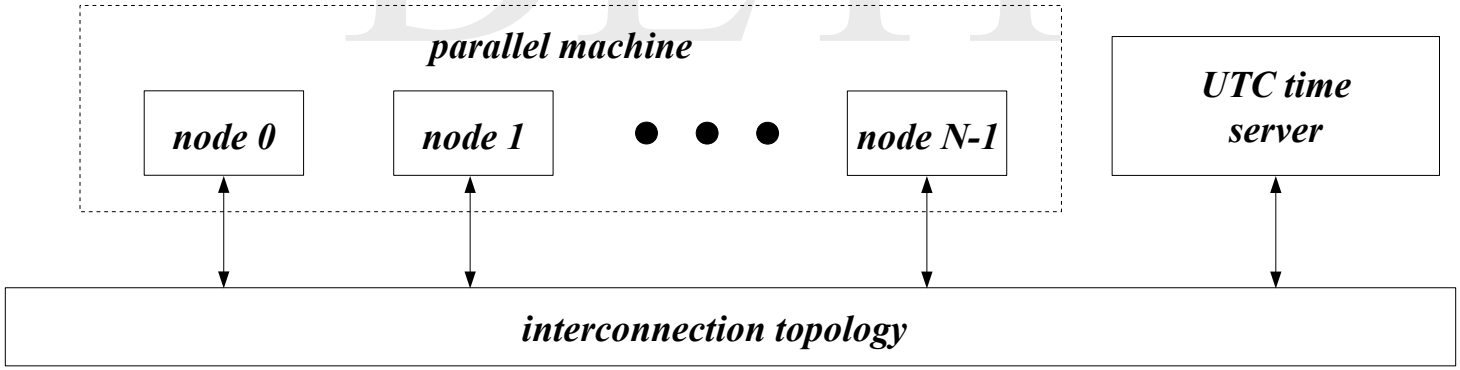
$$\forall_{L \in \mathbb{R}^+} \ \exists_{t_M} \ t_M > L$$

where t_M represents the message transmission time.

The adjustment itself should be made in a manner that the monotonicity of the local time is alwaays preserved. (***Why***?)

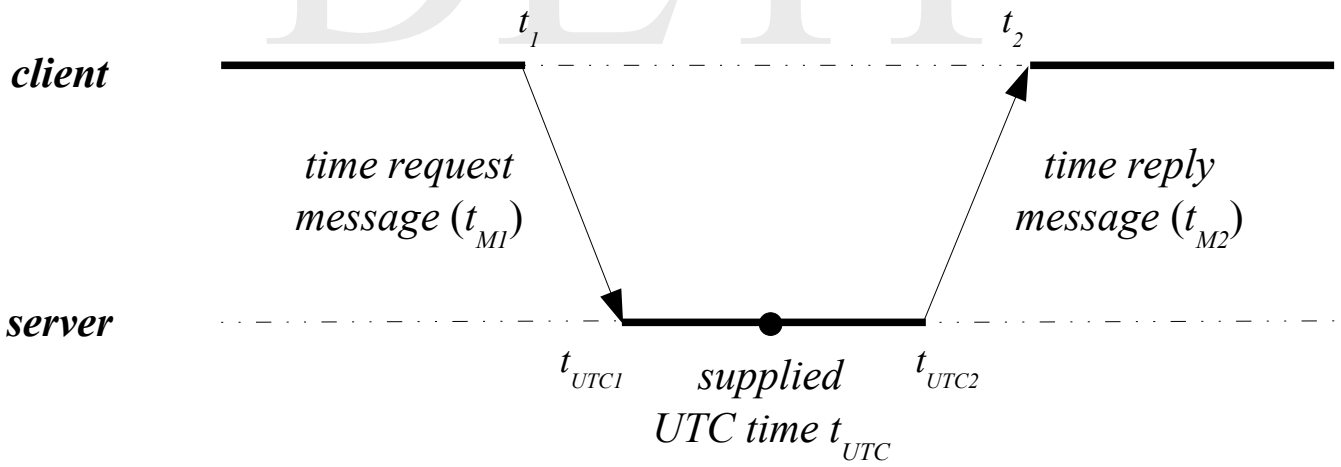
Cristian method – 1

It is an external synchronization method where one assumes the availability of a UTC time server. From time to time, in a proactive manner, in order to adjust its own local clock, each node (acting as a client) addresses the server through the sending of a message where the *right* time is requested. The server replies by sending it in a predefined format.



Cristian method – 2

- operation decomposition
  - the request is made at local time t_1 through the sending of a message with transmission time t_{M1}, the message is received at server UTC time t_{UTC1}
  - the time t_{UTC}, which will be returned, is adjusted to match approximately the middle of the server processing interval
  - the reply is sent at UTC time t_{UTC2} through a message with transmission time t_{M2}, the message is received at local time t_2.



Cristian method – 3

- the client has at its disposal the times t_1 , t_2 and t_{UTC} to adjust the local clock
- the client assumes that the *drift* of the local clock produces a negligible variation of the time interval $t_2 - t_1$
- expressing the *offset* between UTC time and local time by $off = S(t) - Ck(t)$, the offset estimation is given by

$$off_{est} = t_{UTC} - \frac{t_1 + t_2}{2}$$

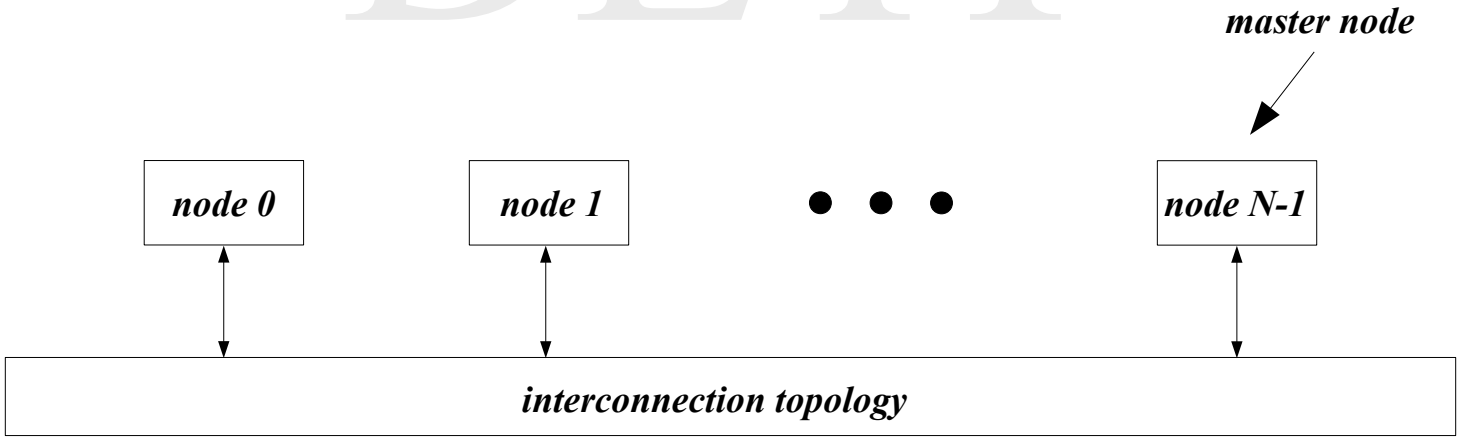
- the associated uncertainty to this value in the worst case, Δ_{est} , assumes that $t_2 - t_1 \approx t_{M1} + t_{MIN}$ (the processing time at the server is negligible when compared with message transmission time and one of them is transmitted in minimum time)

$$\Delta_{est} = \frac{t_2 - t_1}{2} - t_{MIN}$$

- if the estimated uncertainty is larger than the nominal accepted value, as in the case the communication channel endures random load variations or the server is very busy, the client may reject the *offset* estimation and try again later on.

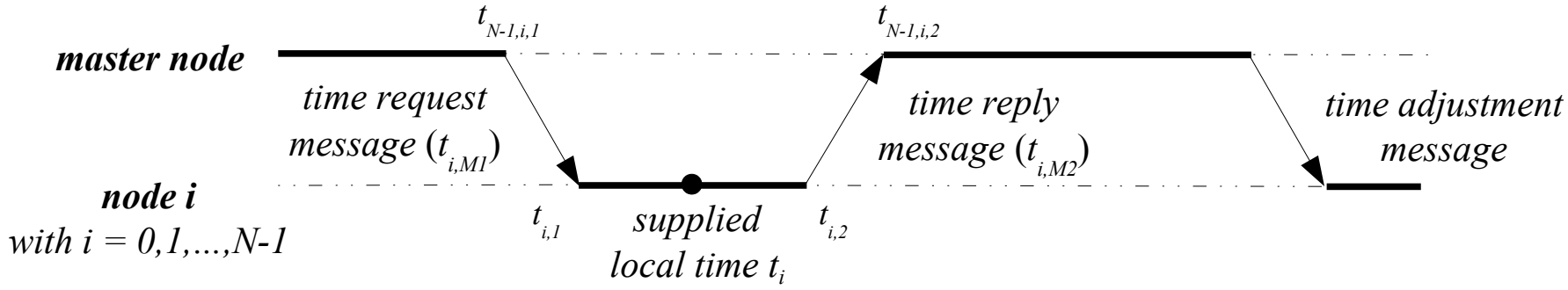
Berkeley algorithm – 1

It is an internal synchronization method to be applied when a UTC time server is not available. Thus, the sole goal is to keep synchronized among themselves the local clocks of the processing nodes of the parallel machine. From time to time, one of the nodes, called for this reason the *master* node, addresses in a proactive manner all the nodes of the parallel machine, itself included, for information about the local time. Then, it computes the average value of the offsets among the other local clocks and its own and sends to all the nodes, itself included, the correction that should be introduced to adjust the local time to a common value.



Berkeley algorithm – 2

- operation decomposition
  - the request is made by the master node at local time $t_{N-1,i,1}$ through the sending of a message to each node of the parallel machine with transmission time $t_{i,M1}$, a message is received by each node at local time $t_{i,1}$, with i = 0,1, … ,N-1
  - the time t_i, which will be sent by each node, is adjusted to match approximately the middle of the processing interval
  - the reply is sent by each node at local time $t_{i,2}$, with i = 0,1, … ,N-1, through a message with transmission time $t_{i,M2}$, the message is received by the master node at local time $t_{N-1,i,2}$
  - the master node computes the deviations to its local time and sends a message to all the nodes with the adjustment that must be introduced in each case.



Berkeley algorithm – 3

- the *master* node has at its disposal the times $t_{N-1,i,1}$, $t_{N-1,i,2}$ and t_i, with i = 0,1,…,N-1, to estimate the offset, and the associated uncertainty, of each of the local clocks
- the *master* node assumes that the *drift* of its local clock produces a negligible variation of the time intervals $t_{N-1,i,2} - t_{N-1,i,1}$, with i = 0,1,…,N-1
- the individual estimations are computed, according to the method of Cristian, by

$$off_{est}(i) = t_i - \frac{t_{N-1,i,1} + t_{N-1,i,2}}{2} \qquad \Delta_{est}(i) = \frac{t_{N-1,i,2} - t_{N-1,i,1}}{2} - t_{MIN}$$

com $i = 0,1,\cdots,N-1$

- the *master* node computes next the average of the estimations of the *offset*, $off_{est\ med}$, rejecting in the computations the cases where the estimated uncertainties are larger than the nominal accepted value
- the *master* node sends in the end to each of the nodes the adjustment which must be introduced in each case, $off_{est\ med} - off_{est}(i)$
- one should remark, since the adjustment is a differential value, the transmission time of this last message does not introduce any further uncertainty.

Network Time Protocol (NTP) - 1

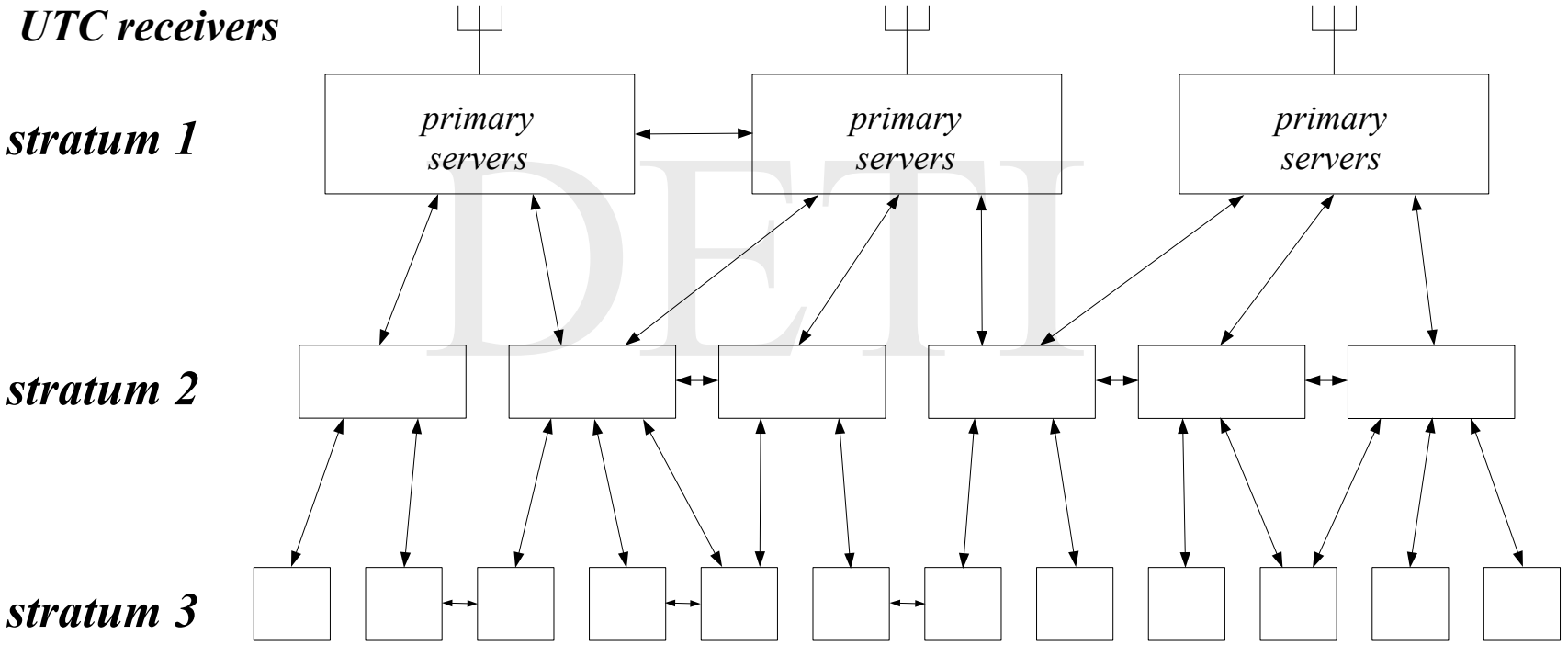
The methods just described aim the synchronization of the local clocks of computer systems integrated in local area networks. The *Network Time Protocol* (NTP), on the other hand, deals with the Internet itself.

The goal is

- to allow every computer system connected to the internet to adjust its local clock with reasonable accuracy and make the adjustment at a rate high enough to prevent serious time discrepancies due to *drift*
- to ensure that the provided service can outlive the more or less long losses of connectivity of specific servers, keeping a permanent availability
- to provide protection against particular interferences.

Network Time Protocol (NTP) – 2

Architecture of the service



Network Time Protocol (NTP) – 3

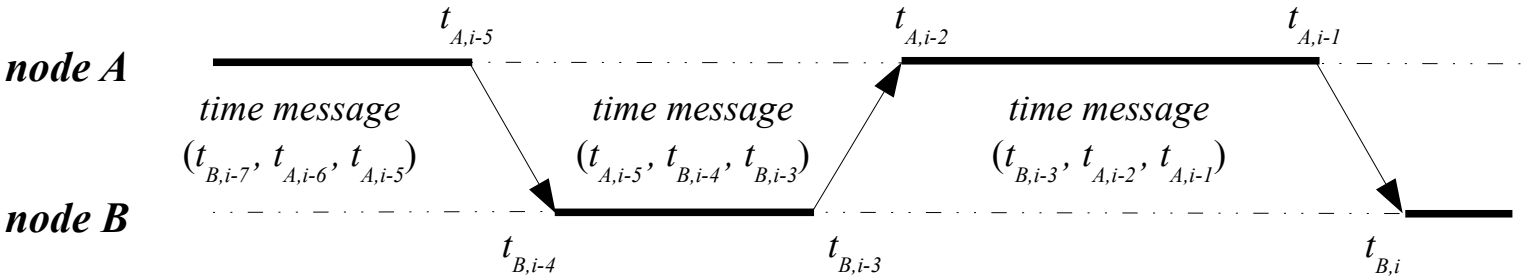
- the service assumes a hierarchical organization of computer systems in different levels, called *strata*
- the computer systems in the first level, *stratum* 1, are connected directly to a UTC source, so they are called *primary servers*
- the remaining computer systems, located in the other *strata*, are called *secondary servers* and adjust their local clocks with servers belonging to a *stratum* immediately above in the hierarchy
- the computer systems in each *stratum* can also coordinate its time information with other servers belonging to the same *stratum* to provide a globally more stable and robust information

Network Time Protocol (NTP) – 4

- as one goes down in the hiearchy, the degree of uncertainty of the produced time information increases because the errors introduced in each synchronization stage are cumulative
- any time adjustment sub-tree is dynamically reconfigured
  - whenever a *primary server* is unable to access its UTC source, it passes to the *stratum* 2, becoming a *secondary server*
  - whenever a server used to adjust the local clock of a computer system in a given *stratum* becomes unavailable, another server is looked for.

Network Time Protocol (NTP) – 5

- an exchange of time messages takes place continuously between pairs of nodes (A e B, for instance) for adjustment of the local clock of node B, if this belongs to a lower *stratum* in the hierarchy, or for mutual adjustment of the local clocks of both nodes, if they belong to the same *stratum*
  - each forward message includes three time stamps: the local times of transmission and reception of last received message, $t_{B,i-3}$ e $t_{A,i-2}$, and the local time of transmission of present message, $t_{A,i-1}$
  - upon message reception, the destination node saves the local reception time, $t_{B,i}$
  - the four times, $t_{B,i-3}$, $t_{A,i-2}$, $t_{A,i-1}$ e $t_{B,i}$, are then used to compute an estimation of the *offset* and its uncertainty.



Network Time Protocol (NTP) – 6

- the node B assumes that the *drifts* of its local clock and the local clock of node A produce negligible variations of the time intervals $t_{B,i} - t_{B,i-3}$ and $t_{A,i-1} - t_{A,i-2}$, respectively
- expressing the *offset* between the local times of both nodes as $off = Ck_A(t) - Ck_B(t)$, aits estimation is given by

$$off_{est} = \frac{t_{A,i-1} + t_{A,i-2}}{2} - \frac{t_{B,i} + t_{B,i-3}}{2}$$

- the associated uncertainty to this value in the worst case, Δ_{est}, assumes that the transmission time of one of two successive messages is minimum

$$\Delta_{est} = \frac{t_{A,i-2} - t_{B,i-3} + t_{B,i} - t_{A,i-1}}{2} - t_{MIN}$$

- the node B applies next an algorithm of statistical filtering to the successive pairs (off_{est}, Δ_{est}) which are computed to produce a final estimation
- the procedure is carried out with more than a server, the results are compared and may lead to a change of the server(s) being addressed.

Process synchronization

The variability introduced in the readings of the local clocks of the processing nodes of the parallel machine, by the time adjustment algorithms, makes impossible to use time information to synchronize the activities of the different processes which form a distributed application. The best one may hope to achieve is to generate a degree of uncertainty in the millisecond range between the time readings of pairs of local clocks, which means that about a million of instructions may be executed by each processor in a time interval of this magnitude.

On the other hand, Lamport (1978) has shown that if two processes residing in distinct nodes do not interact, it is not strictly necessary that their local clocks tick in pace: the mismatch is not observable and, therefore, conflicts will not arise. Thus, what really matters is not that all concerned processes agree on the value of *actual* time, but they order the relevant events that take place in the same manner, that is, those that are involved in some kind of interaction.

The exploitation of this concept leads to the so called *logic clocks* which only portray the flow of information that occurs.

Scalar logic clock - 1

One defines an *event* as any relevant activity which occurs during the execution of a process. Among all activities that take place, the communication activities are specially significant as far as process synchronization is concerned, that is, message *sending* and message *receiving*.

The ordering of events that occur in a distributed application, is based on two almost trivial observations

- when two events occur in the same process, they take place in the order perceived by the process
- when a message is exchanged between processes, the message *sending* event takes necessarily place *before* the *receiving* event of the same message.

Pairs of events are classified for ordering purposes as

- *sequential* – if it is possible to assert which one has occurred *before*
- *concurrent* – otherwise.

Scalar logic clock - 2

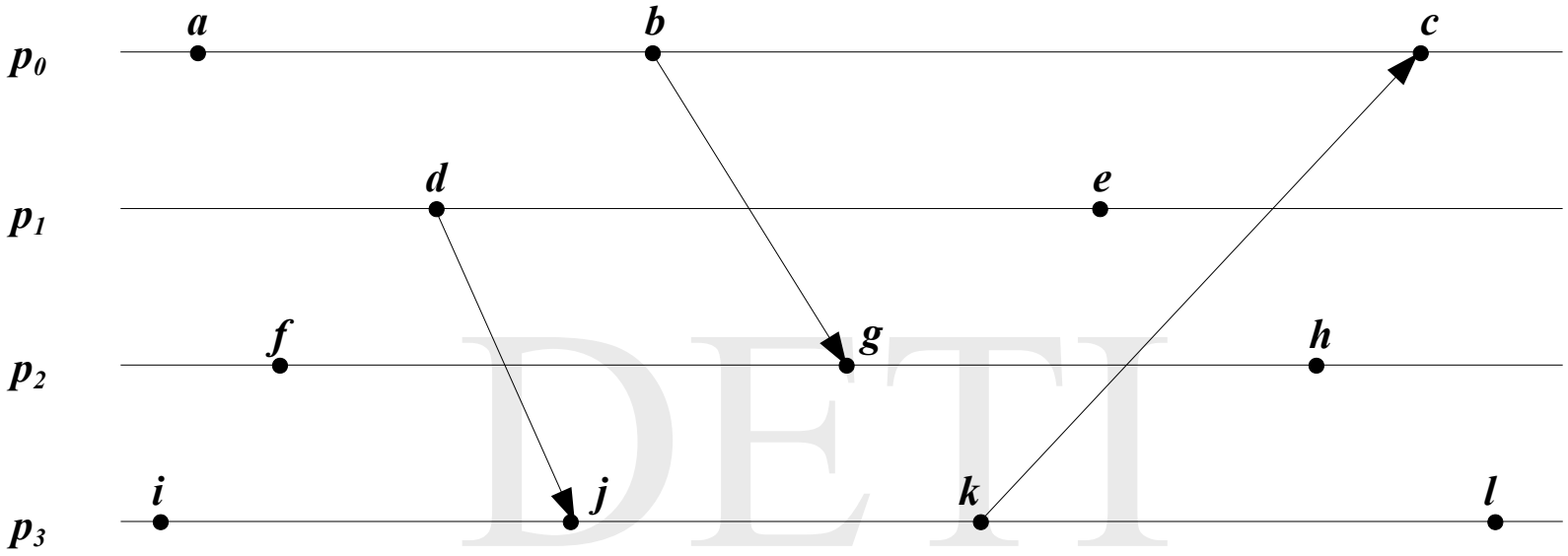
Lamport (1978) named *occurred before*, and denoted it by $\rightarrow$, the partial ordering of events which comes from generalizing the observations that were just made.

The relation *occurred before* can be formally defined in the following way

- $\exists_{p_i \in \{p_0, p_1, \cdots, p_{N-1}\}} \; e \rightarrow_i e' \Rightarrow e \rightarrow e'$
- $\forall_{message} \; \text{send}(m) \rightarrow \text{receive}(m)$
- $e \rightarrow e' \wedge e' \rightarrow e'' \Rightarrow e \rightarrow e''$

where p_i , with i = 0,1,…,N-1, are the processes that coexist in the distributed application and that are executed in distinct nodes of the parallel machine, and e, e' and e'' are generic events.

Scalar logic clock - 3



sequential events

$$f \rightarrow g \ \wedge \ g \rightarrow h \ \Rightarrow \ f \rightarrow h$$

$$d \rightarrow j \ \wedge \ j \rightarrow k \ \wedge \ k \rightarrow c \ \Rightarrow \ d \rightarrow c$$

concurrent events

$$\neg(f \rightarrow c) \ \wedge \ \neg(c \rightarrow f) \ \Rightarrow \ f \parallel c$$

$$\neg(i \rightarrow e) \ \wedge \ \neg(e \rightarrow i) \ \Rightarrow \ i \parallel e$$

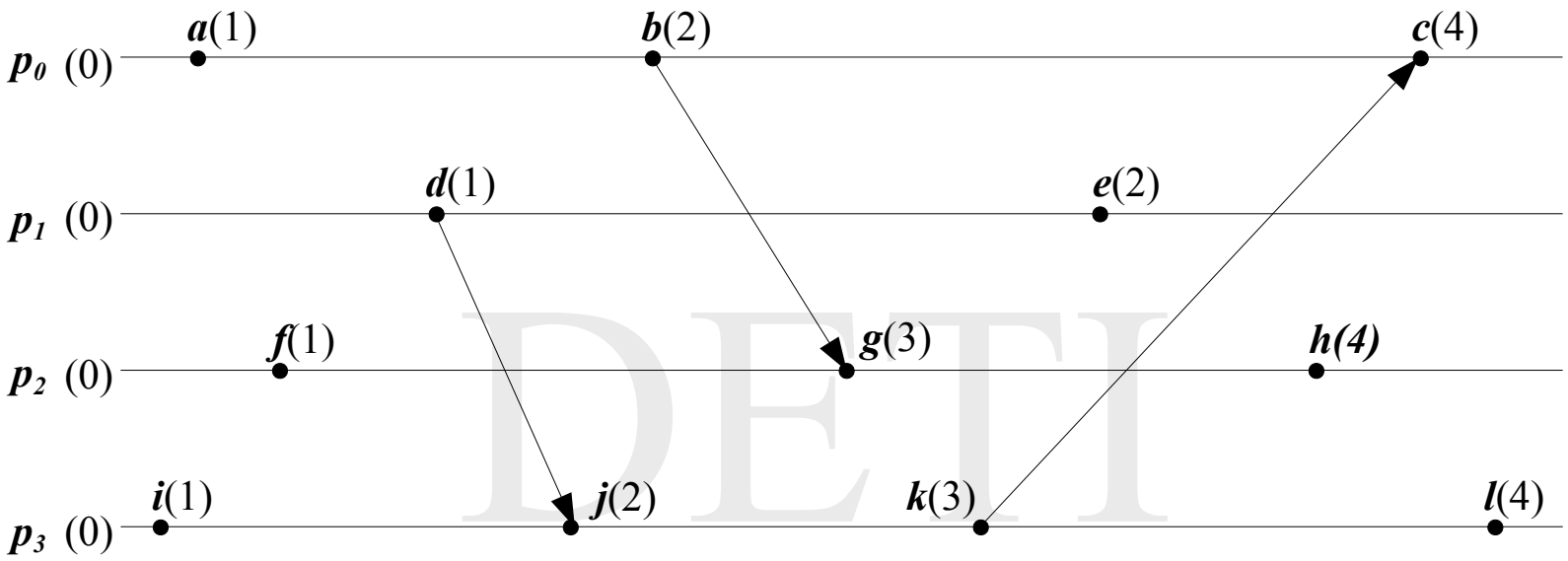
Scalar logic clock - 4

Lamport suggested a device, which he called *logic clock*, to explicit numerically the ordering resulting from the relation *occurred before*. A *scalar logic clock* is essentially a local counter of events, monotonically increasing, which has no direct association with the *actual* time.

In a distributed application, each process p_i, with $i = 0,1,\ldots,N-1$, uses a logical clock as its own local clock, Ck_i, and marks the events according to the following rules

- *initialization*: $Ck_i = 0$
- *occurrence of a locally relevant event*: updating of the local clock, $Ck_i = Ck_i + \alpha_i$, where α_i is a numeric constant usually equal to 1
- *message sending*: insertion of a time stamp *ts*, of value equal to Ck_i, in the message to be sent after the updating takes place
- *message reception*: adjustment of the local clock to the value *max* (Ck_i , *ts*) before updating its value with the reception event.

Scalar logic clock - 5



It is straightforward to prove by mathematical induction that

$$e \rightarrow e' \;\Rightarrow\; Ck_i(e) < Ck_j(e') \quad \text{, with } i, j = 0,1, \cdots, N-1 \;\; .$$

The converse, however, is not true.

Total ordering of groups of events - 1

Lamport has shown, however, that groups of related events may be subjected to an operation of *total ordering* and be perceived in the same order by processes residing in different nodes of a parallel machine, if logic clocks of the type he has prescribed are used in the generation of time stamps included in the exchanged messages.

Let e_j, with $j = 0,1,\ldots,K\text{-}1$, be events associated with the exchange of messages m_j among the processes p_i, with $i = 0,1,\ldots,N\text{-}1$, then the events e_j can be *totally ordered* if and only if one can establish a one-to-one correspondence between each event and a point in the numeric [straight] line through an associated property (the time stamp inserted in each message).

Total ordering of groups of events - 2

Since nothing prevents the time stamps associated with two distinct messages to be equal, $ts(m_p) = ts(m_q)$, with $p, q = 0,1,\ldots,K\text{-}1$ and $p \neq q$, Lamport defined a data structure he called *extended time stamp*, $(ts(m_p),id(m_p))$, which consists of the ordered pair formed by the message time stamp and the sender identification, and prescribed the following rule to order the *time stamps*

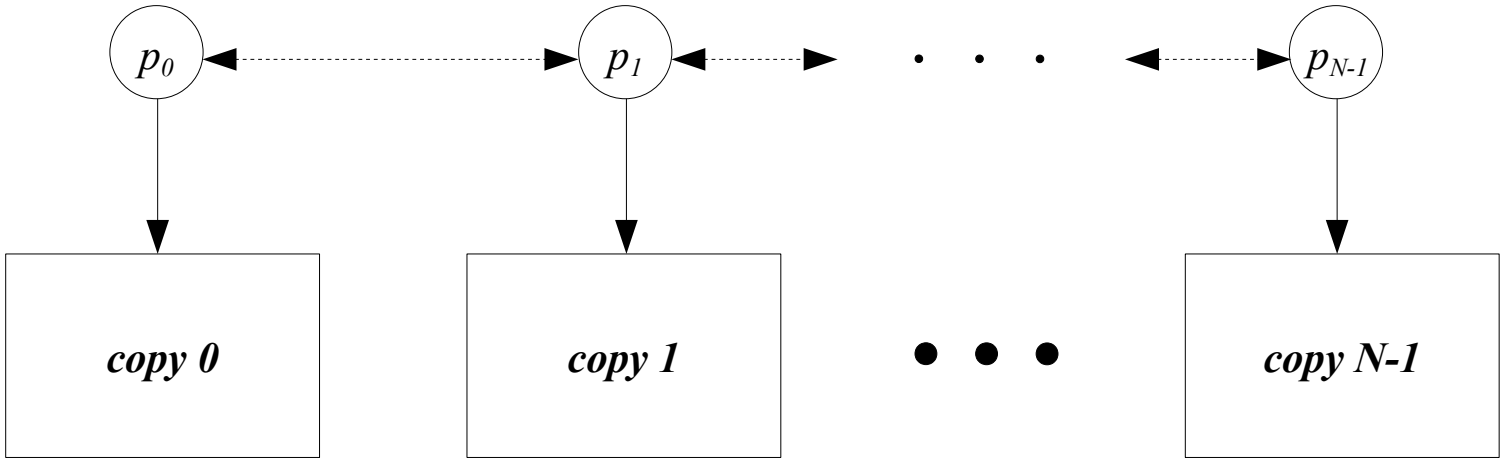
$$(ts(m_p), id(m_p)) < (ts(m_q), id(m_q)) \Leftrightarrow ts(m_p) < ts(m_q) \vee ts(m_p) = ts(m_q) \wedge id(m_p) < id(m_q) \ .$$

Total ordering of groups of events - 3

Assume that, in a given distributed application, there are N copies of the same data region located in geographically distinct places. Each copy is accessed by a specific process p_i, with $i = 0,1,\ldots,N\text{-}1$. Each process p_i performs operations of *writing* and *reading* over the registers of its copy leading to the change of its contents.

Question

How to organize operations so that the different copies are kept permanently synchronized, that is, they always present the same value in all its registers?



Total ordering of groups of events - 4

The permanent synchronization of the copies requires that

- whenever a process wants to modify the value of a register of its local copy, the operation must be propagated first to the processes that manage the access to all the other copies
- the order of execution of these operations must be the same everywhere.

In order for this to be done successfully, one must assume that

- the processes p_i are kept in correct operation, that is, no *catastrophic failures* occur
- there is no message loss.

Total ordering of groups of events - 5

Lamport proposed the following algorithm to solve the problem

- each process p_i, upon asserting that next operation will modify the value of a register of its local copy, builds a message with all the data concerning the operation and attaches a time stamp with the value of its local clock marking the event
- the message is sent to all the group members, itself included
- upon receiving the message, each process p_i adjusts its local clock according to the rules prescribed by Lamport and inserts the message in a local queue in increasing order of its *extended time stamp*
- a message of *acknowledge* is sent to all the group members, itself included
- the operations described in the messages stored in each local queue are executed by each process p_i in the pre-established order when all group members have acknowledged the operation.

Vector logic clock - 1

Mattern (1989) and Fidge (1991) have introduced another type of logic clock with the goal of surpassing the limitation present in Lamport scalar logic clock

$$\neg[\forall_{e_i, e'_j} \; Ck_i(e_i) < Ck_j(e'_j) \;\Rightarrow\; e_i \rightarrow e'_j] \quad \text{, with } i, j = 0,1, \cdots, N-1 \text{ and } i \neq j \text{ ,}$$

that is, when two events e_i and e'_j occur in distinct processes, the fact that the value of the time stamp associated with the first be less than the value of the time stamp associated with the second does not mean the first event *has occurred before* the second.

Their idea was to keep information about events, not only retrieved from their own logic clock, but also all available information retrieved from the logic clocks of the other processes in the group, even if it were not updated. In this way, one can capture the *potential causality* that may exist between events occurring in processes residing in distinct nodes of the parallel machine.

Vector logic clock - 2

The suggested device operating in a N process system is basically a collection of monotonically increasing event counters, which again have no connection to the *actual* time.

The counter collection is organized as an array V, where each element represents a Lamport-like local clock.

In a distributed application, each process p_i, with $i = 0,1,\ ,N\text{-}1$, has its own vector logic clock V_i. Array V_i elements are interpreted in the following way

- $V_i[i] = Ck_i$, is the local clock of process p_i
- $V_i[j] = Ck_j$, with $j \neq i$, represents the perception process p_i has about the evolution of the local clock of process p_j (process p_j may meanwhile have marked more events, but process p_i has not yet received any information about them in the message time stamps so far received).

Vector logic clock - 3

The updating of the local vector clock is carried out according to the following rules

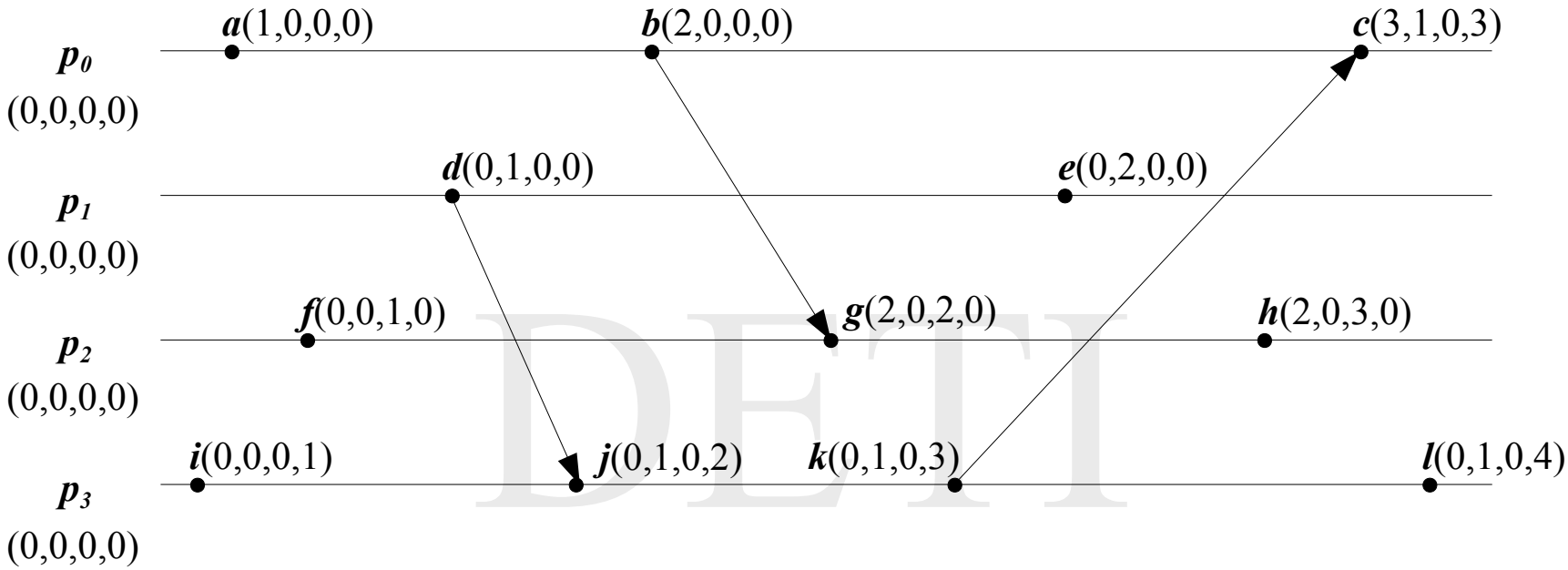
- *initialization*: $V_i[j] = 0$, with $j = 0,1,\ldots,N-1$
- *occurrence of a locally relevant event*: updating of the local clock $V_i[i] = V_i[i]+\alpha_i$, where α_i is a numeric constant usually equal to 1
- *message sending*: insertion of a time stamp *ts*, of value equal to V_i, in the message to be sent after the updating takes place
- *message receiving*: adjustment of each element of the local vector clock to the value *max* ($V_i[j]$, *ts*[*j*]), with $j = 0,1,\ldots,N-1$ and $i \neq j$, before updating its own value $V_i[i]$ with the reception event.

Vector logic clock - 4

Let V e V' be vector time stamps, then their values are compared according to the following rules

- $V = V' \Leftrightarrow \forall_{0 \le j < N} \; V[j] = V'[j]$
- $V \le V' \Leftrightarrow \forall_{0 \le j < N} \; V[j] \le V'[j]$
- $V < V' \Leftrightarrow V \le V' \wedge V \ne V'$.

Vector logic clock - 5



sequential events

$$f \rightarrow h \;\Rightarrow\; V_2(f) < V_2(h)$$

$$d \rightarrow c \;\Rightarrow\; V_1(d) < V_3(c)$$

concurrent events

$$f \parallel c \;\Rightarrow\; \neg[V_2(f) < V_0(c)] \;\wedge\; \neg[V_0(c) < V_2(f)]$$

$$i \parallel e \;\Rightarrow\; \neg[V_3(i) < V_1(e)] \;\wedge\; \neg[V_1(e) < V_3(i)]$$

Vector logic clock - 6

It is straightforward to prove by mathematical induction that

$$e \rightarrow e' \;\Rightarrow\; V_i(e) < V_j(e') \quad \text{, with } i, j = 0,1, \cdots, N-1 \;\; .$$

The converse is now also true.

$$V_i(e) < V_j(e') \;\Rightarrow\; e \rightarrow e' \quad \text{, with } i, j = 0,1, \cdots, N-1 \;\; .$$

Challenge

Prove it!

It is precisely this fact that allows, by comparing the associated time stamps to capture the *potential causality* which may exist between events that occurs in processes located in distinct nodes of a parallel machine and order them.

Suggested reading

- *Distributed Systems: Concepts and Design, 4th Edition, Coulouris*, Dollimore, Kindberg, Addison-Wesley
  - Chapter 11: *Time and global states*
    - Sections 11.1 to 11.4
- *Distributed Systems: Principles and Paradigms, 2nd Edition,* Tanenbaum, van Steen, Pearson Education Inc.
  - Chapter 6: *Synchronization*
    - Sections 6.1 and 6.2