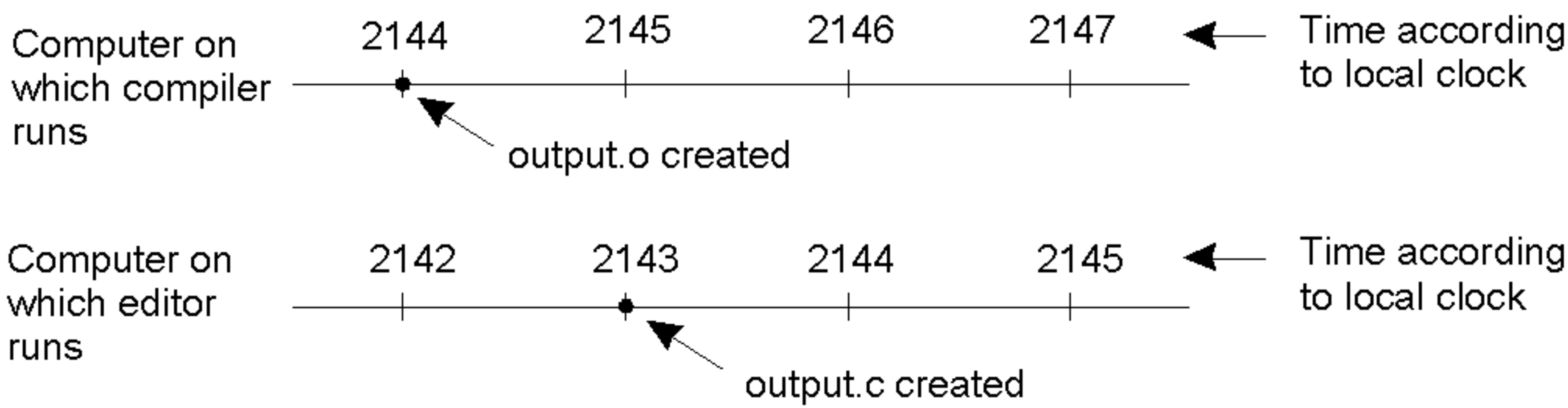
Time, Clocks, and Global State

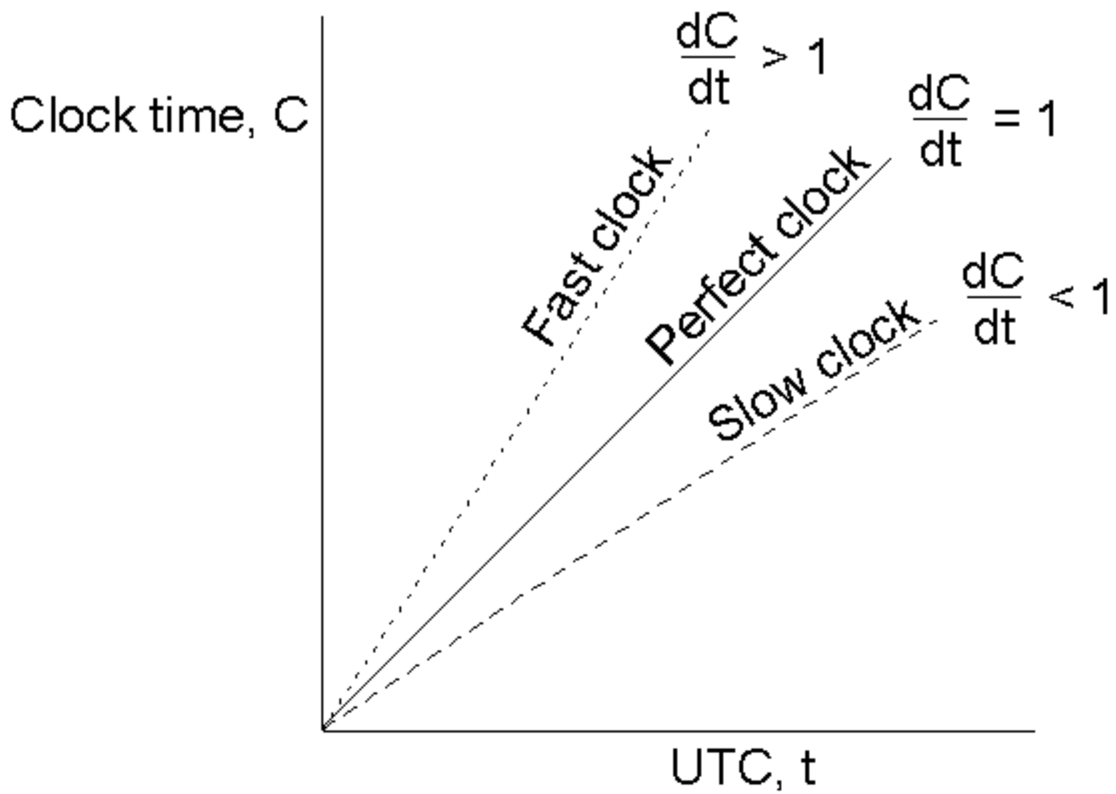
Part-2

Clock Synchronization



- When each machine has its own clock, an event that occurred after another event may nevertheless be assigned an earlier time.

Clock Synchronization Algorithms



- The relation between clock time and UTC when clocks tick at different rates.

Synchronization Protocols

The simplest case of clock synchronization involves two processes in a synchronous system.

Here, bounds are known for:

- drift rate of clocks
- maximum transmission delay
- time for each step in the process

p_i can then send $C_i(t)$ in a message *m* to other processes p_j. The receiving process p_j sets its clock to $C_i(t) + T_{trans}$, where T_{trans} is the time taken to transmit message m.

Unfortunately, T_{trans} cannot be static and is subject to variation.
In general, T_{trans} is not known.

In a synchronous systems, we have an upper and lower bound on *transmission delay* T_{trans}. Hence, the uncertainty in T_{trans} is *u=(max – min).*

Setting the clock to *t + min* will result in clock skew as much as *u*. Similarly, if the clock is set to *t + max,* the skew may be as large as *u.*

If, however, we set the clock to *t + (min + max)/2,* the skew is at most *u/2.*

more Synchronization

Lundelius and Lynch have shown that the optimal bound that can be achieved on clock skew when synchronizing N clocks is *u(1-1/N)*.

Most DS found in practice are asynchronous:

•factors leading to message delays are not bounded;

•there is no *max* on T_{trans}

➔see the Internet !!

Here, T_{trans} = *min + x where x is* $x \geq 0$ *unknown.*

External Synchronization as proposed by Cristian (1989).....

He suggested the use of time servers, connected to a device that receives signals from a UTC source.

Upon request, server process S supplies the time according to its clock.

A process p requests the time via a message m_r and receives time value t via a message m_t. p records the total round-trip time T_{round}. p can do so with reasonable precision if its rate of clock drift is small.

Cristian's approach

For example: the round-trip delay in a LAN is on the order of 1 – 10 ms. A clock drift rate of 10^{-6} sec/sec will cause a drift of at most 10^{-5}ms.

p should set its clock to $t + T_{round}/2$, which assumes that delay is split equally in both directions.

If *min* is known or can be estimated conservatively, the clock accuracy can be computed as follows:

The earliest time that S could have placed t into m_t was *min* after p send m_r.

The latest point this could have been done was *min* before m_t arrived at p.

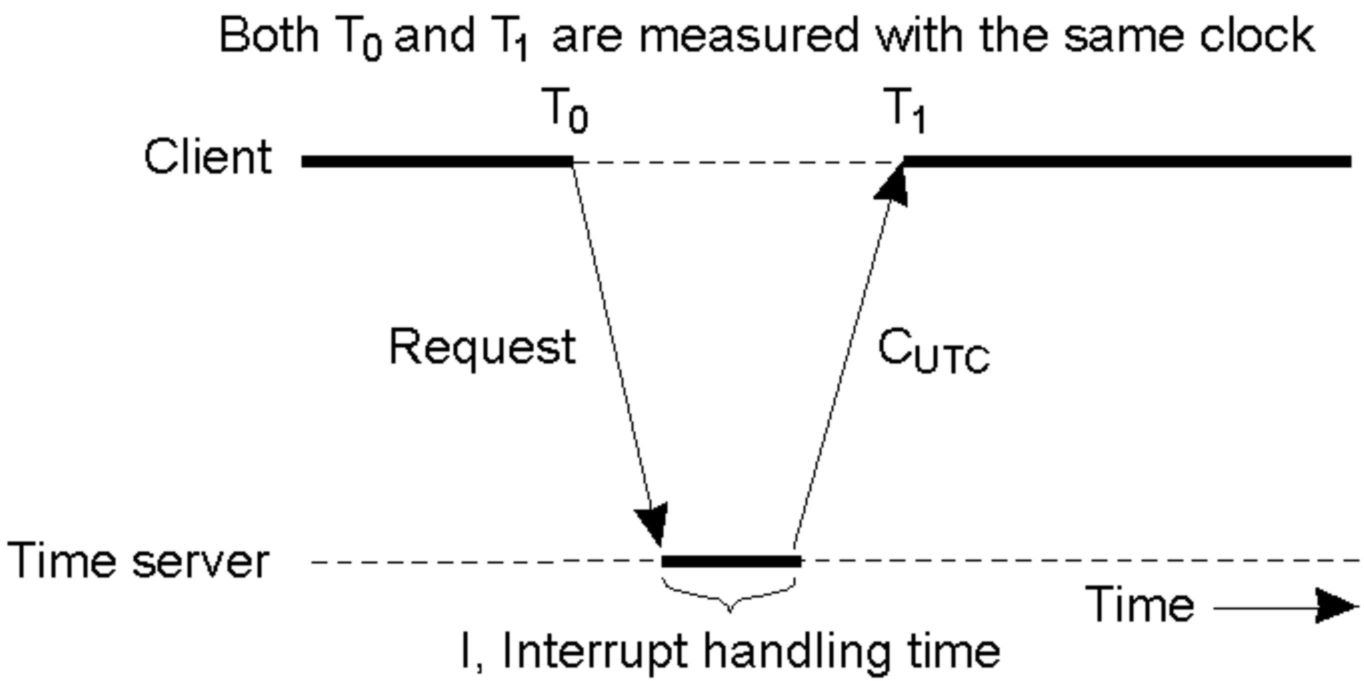
The uncertainty is hence:

$[\, t + min,\ t + T_{round} - min]$

➔ *accuracy is thus* $\pm(T_{round}/2 - min)$

Cristian's Algorithm

- Getting the current time from a time server.



Discussion

Of course, Cristian's approach suffers from several disadvantages including:

- Single point of failure ➔ if S fails, no synchronization is possible !
- Faulty or corrupt time servers may reply with spurious time values !
- An imposter may deliberately reply with incorrect times and wreak havoc.

Cristian advocated he use of groups of time servers to avoid some of these problems. However, this would require the coordination of time servers, i.e., internal synchronization among S_i.

Imposters and faulty time servers are beyond the scope of clock synchronization. They are, however, addressed in the context of the Byzantine Generals problem, which deals with the ability to compute correct values in a DS even in the presence of faulty nodes.

The Berkeley Approach

Gusella and Zatti (1989) developed an algorithm for internal synchronization.

In it, one node is chosen as coordinator to act as *master*. The master periodically contacts nodes and requests their current time.

Upon receiving their responses, the *master* estimates their corresponding $C_i(t)$ by observing round-trip delays.

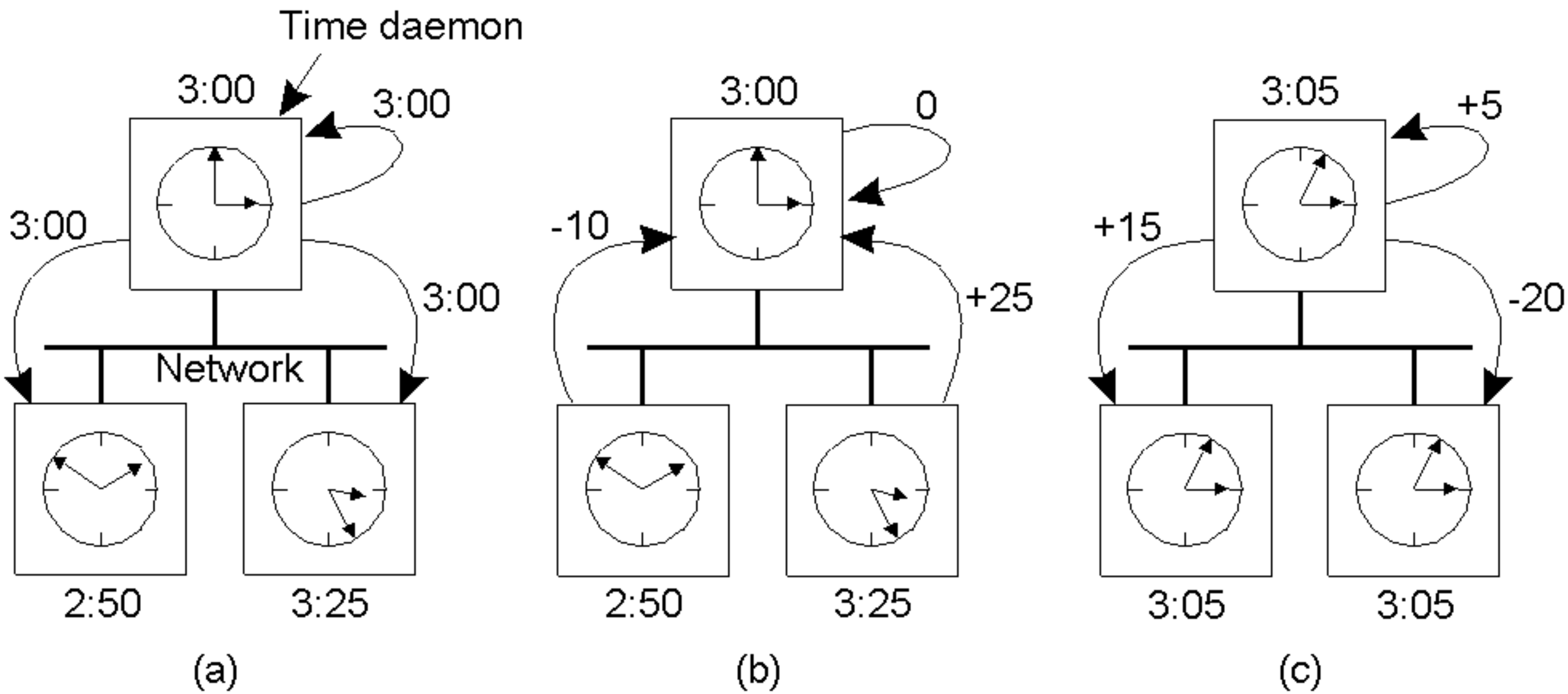
It then averages the values of all nodes (including its own). This averaging cancels out the individual clock drifts.

The *master* then returns to each node the amount of time by which each individual $C_i(t)$ should be adjusted. (i.e., a + or – number).

In order to address the issue of faulty clocks, which could have adverse effects on the average, a *fault-tolerant* average is computed.

For this, only a subset of nodes with $C_i(t)$ values close to each other are considered.

The Berkeley Algorithm



a) The time daemon asks all the other machines for their clock values
b) The machines answer
c) The time daemon tells everyone how to adjust their clock

The Network Time Protocol

Cristian's and Berkley algorithms are designed for use in small, delineated network (DS) environments.

NTP defines and architecture for time services and a protocol for the distribution of time information across the Internet.

NTP has the following design aims:

- to provide services that enables clients across the Internet to synch. accurately.
- to provide reliable service that can overcome lengthy losses of connectivity.
- to enable client to frequently to resynchronize to offset the drift rates.
- to protect against interference with the time service, both malicious and accidental.

this is too much for this course but you can read more about NTP at http://www.ntp.org also, check out RFCs 1305 & 2030.

Events and Logical Clocks

- Lamport's 1978 paper: *Time, Clocks, and the Ordering of Events in Distributed Systems.*
  - Theoretical Foundation
  - Logical Clocks
  - Partial and Total Event Ordering
  - Towards distribute mutual exclusion

Theoretical Foundations

- Inherent limitations of a distributed system:
  - Absence of a global clock:
    - Global clock is available to all the processes: two processes can observe a global clock value at different instants due to unpredictable message delay; therefore, may perceive two different instants in physical time to be a single instant in physical time.
    - A physical clock for each computer: these clocks can drift from the physical time and the drift rate may vary from clock to clock; therefore, may perceive two different instants in physical time as a single instant.
  - Impact: Due to the absence of global clock, it is difficult to reason about the temporal order of events in distributed system, e.g. scheduling is more difficult.

Inherent Limitation -- cont…

- Absence of shared memory: an up-to-date state of the entire system is not available to any process.
  - A view is **coherent** if all the observations of different processes (computers) are made at the same physical time.
  - A **complete view** (global state) encompasses the local views (local states) at all the processes (computers) and any messages that are in transit.
  - A process in a distributed system can obtain a *coherent* but partial view of the system or a complete but *incoherent* view of the system.

Lamport's Logical Clocks

- The execution of processes is characterized by a sequence of events; e.g. execution of an instruction or a procedure, sending or receiving messages.
- Lamport proposed a scheme to order events in a distributed system.
- Note that due to the absence of perfectly synchronized clocks and global time in distributed systems, the order in which two events occur at two different computers cannot be determined based on the local time at which they occur.

Happened Before Relation

- The **happened before** relation captures the causal dependencies between events, i.e. whether two events are causally related or not.
- a→b if a and b are events in the same process and a occurred before b.
- a→b if a is the event of sending a message m in a process and b is the event of receipt of the same message m by another process.
- If a->b and b->c, then a→c, i.e. happened before relation is transitive.
- That is, past events causal affects future events.

Concurrent Events

- Two distinct events a and b are **concurrent** (a||b) if not (a→b or b→a). In other words, concurrent events do not causally affect each other.
- For any two events a and b in a distributed system, either a→b, b→a or a||b.

Logical Clocks

- There is a clock Ci at each process Pi in the distributed system.
- The clock Ci can be thought of as a function that assigns a number Ci(a) to any event a, called the **timestamp** of event a, at Pi.
- These clocks can be implemented by counters and have no relation to physical time.

Conditions Satisfied by the System of Clocks

- For any events a and b: if $a \rightarrow b$, then $C(a) < C(b)$.

- The happened before relation can now be realized by using the logical clock if the following two conditions are met:
  - [C1] For any two events a and b in a process Pi, if a occurs before b, then $Ci(a) < Ci(b)$.
  - [C2] If a is the event of sending a message m in process Pi and b is the event of receiving the same message m at process Pj, then $Ci(a) < Cj(b)$.

Implementation Rules

- [IR1] Clock Ci is incremented between any two successive events in process Pi: Ci:=Ci+d (d>0). Note that if a and b are two successive events in Pi and a->b, then Ci(b):=Ci(a)+d. Note: d is usually 1.

- [IR2] If event a is the sending of message m by process Pi, then message m is assigned a timestamp tm=Ci(a) (note that the value of Ci(a) is obtained after applying rule IR1). On receiving the same message m by process Pj, Cj is set to a value greater than or equal to its present value and greater than tm. Cj:=max(Cj, tm+d) (d>0).

Total Ordering of Events

- Lamport's happened before relation defines an irreflexive partial order among the events.
- The set of all events in a distributed computation can be totally ordered (denoted by =>) using the above system of clocks as follows: if a is any event at process Pi and b is any event at process Pj then a=>b iff either
  - $C_i(a) < C_j(b)$ or
  - $C_i(a) = C_j(b)$ and $P_i < P_j$.

Virtual Time

- Lamport's system of logical clocks implements an approximation to global/physical time, which is referred to as virtual time.

- Virtual time advances along with the progression of events and is therefore discrete.

- If no events occur in the system, virtual time stops, unlike physical time which continuously progresses.

Limitation of Lamport's Clocks

- Note that in Lamport's system of clocks, if a→b then C(a)<C(b).

- However, the reverse is not necessary true if the events have occurred in different processes: if a and b are events in different processes and C(a)<C(b), then a->b is not necessary true; events a and b may be causally related or may not be causally related.

Simple Solution to DME

- A site, called the **control site**, is assigned the task of granting permission for the CS execution.
- To request the CS, a site sends a request message to the control site, which queues up the requests and grants them one by one.
- Requires 3 messages per CS execution.
- Drawbacks:
  - single point of failure
  - control site may be overloaded and nearby communication links may be congested
  - low system throughput.

Lamport's Algorithm

- Based on Lamport's clock synchronization scheme.
- For all i, the request set Ri={S1,S2,.......,Sn}
- Every site Si keeps a request queue, which contains requests ordered by their timestamp.
- Assume that messages to be delivered in FIFO order between every pair of sites.

Requesting CS:

- To request CS, a site send a REQUEST(tsi,i) message to all sites in Ri and places the request on its request queue
- When a site Sj receives the REQUEST message from Si, it returns a timestamped REPLY message to Si and places the REQUEST in its request queue.

Executing CS: Site Si enters CS when

- Si has received a message with timestamp larger than (tsi,i) from all other sites, and
- Si's request is at the top of its own request queue

Releasing CS:

- Remove its request and sends timestamped RELEASE
- Other sites will remove the REQUEST accordingly.

Correctness Proof

By contradiction: suppose two sites S_i and Sj are executing the CS concurrently. Then both the conditions for executing CS must hold at both sites, i.e. both S_i and S_j have its own requests at the top of their request queues. WLOG, assume that S_i's request have a smaller timestamp than that of S_j.

It is clear that the request of Si must be present in S_j's request queue, when Sj is executing in CS. This provides the contradiction that S_j's own request is at the top of the request queue when a smaller timestamp request is present.

Performance and Optimization

- Number of messages required: 3(N-1) message per CS invocation.

- Synchronization delay: T

- Optimization: By suppressing REPLY messages in certain condition:
  - For example, suppose site Sj receives a REQUEST message from site Si after it has sent its own REQUEST message with timestamp higher than the timestamp of site Si's request. In this case site Sj need not send a REPLY message to site Si.