ICS 362 Distributed Systems

Distributed Systems: Part 12

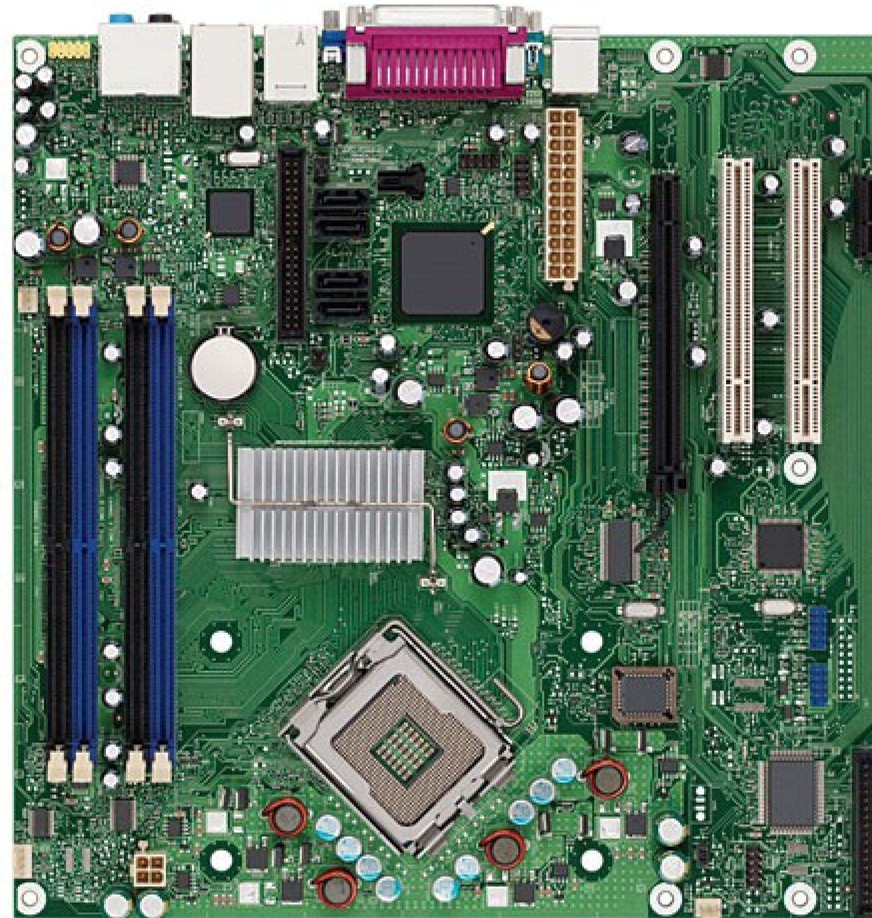
Lecturer: Toby Daniel

Synchronisation

- Clock Synchronisation
- Logical Clocks
- Global State
- Election Algorithms
- Mutual Exclusion
- Distributed Transactions

Sychronisation

- What happens in a computer with a single processor?



Sychronisation

- With a single processor, a timing request can be simply directed;
  - A system call to the kernel returns system time.
- Sequences are easy to identify;
  - Message m1 arrived before Message m2.

Synchronisation Problems

- So what happens in a distributed system?
  - For example a grid DOS.

Synchronisation Problems

- With distributed systems, there is no global time, and many messages overlapping each other.

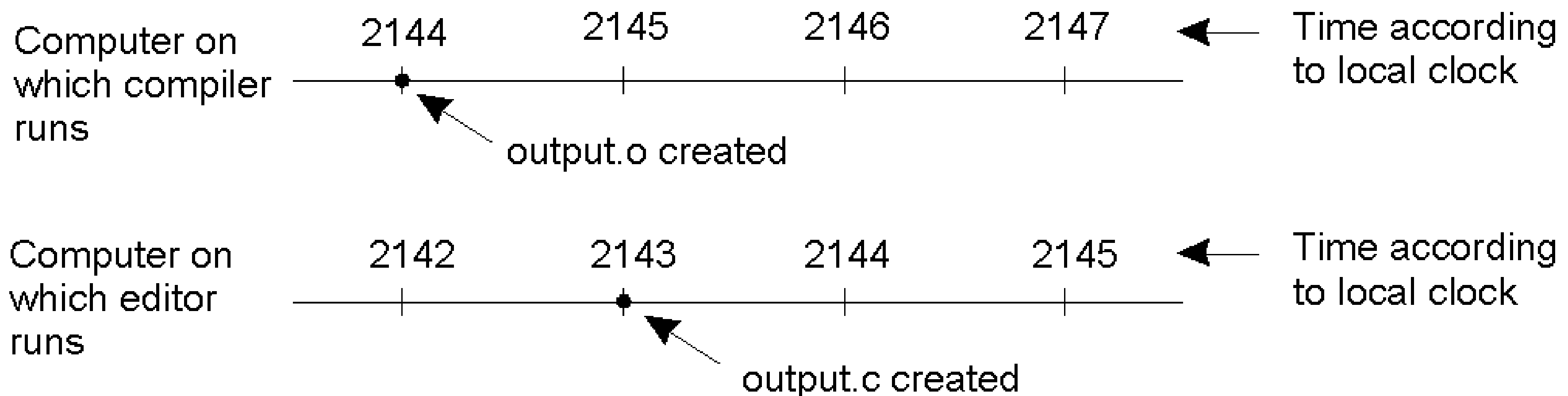
But why do we need sychronisation anyway?

The Need to Synchronise

- Often important to *control access* to a single, shared resource.
- Also often important to agree on the *ordering of events*.
  - Synchronisation in Distributed Systems is much more difficult than in uniprocessor systems.

Clock Synchronisation

- When each machine has its own clock, an event that occurred after another event may nevertheless be assigned an earlier time on another remote device.
  - Example: MAKE will not call the compiler for the newer version of the "output.c" program, even though it is "newer".

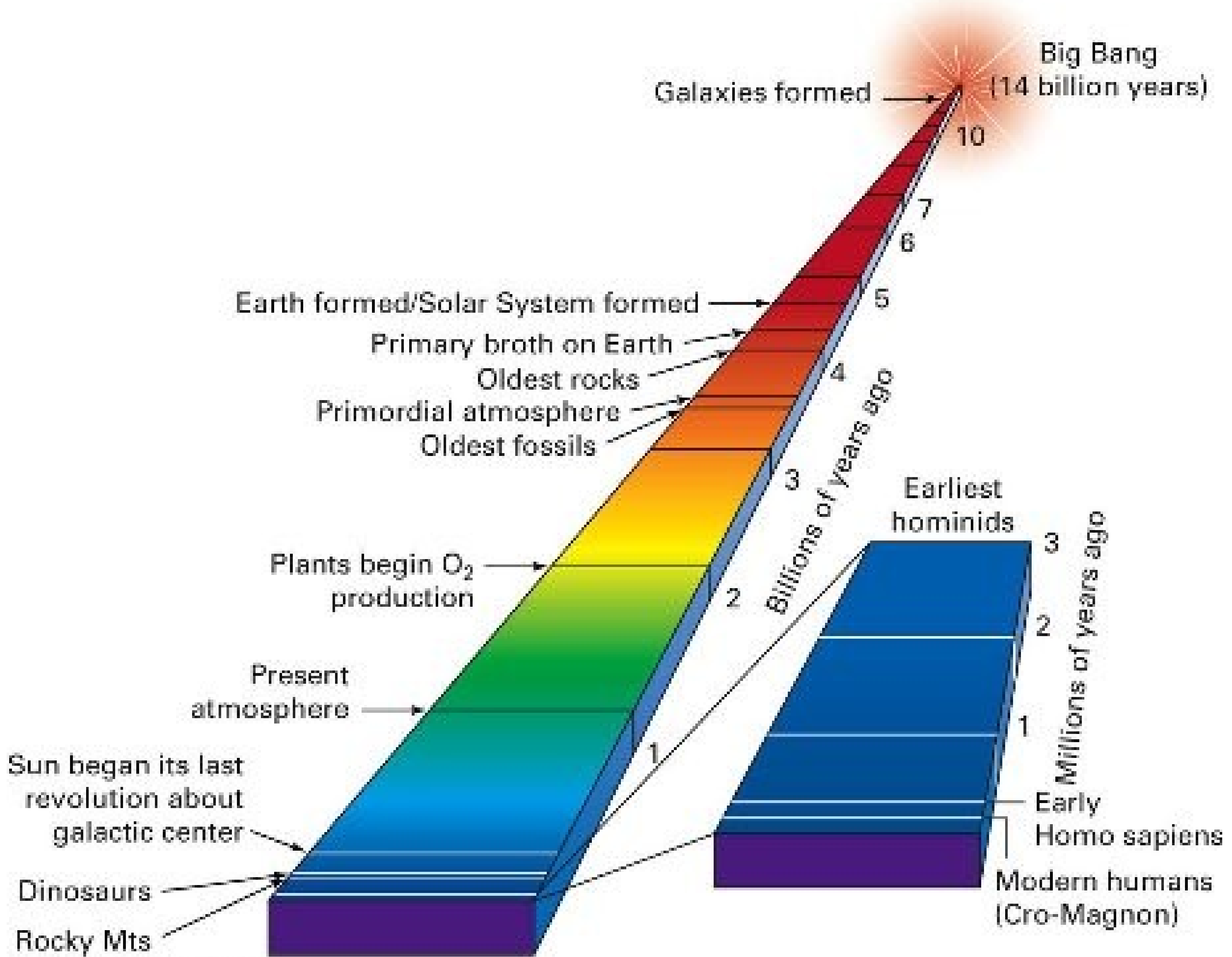


The problem

- Achieving agreement on time in a Distributed System is not trivial.
- Is it even possible to synchronise all the clocks in a Distributed System?
- With multiple computers, “clock skew” ensures that no two machines have the same value for the “current time”. But, how do we measure time?

What is Time?

- It has over 60 definitions in dictionary.com alone!
  - All of which could be described as human methods of control; a way of coordinating actions, or enabling a landlord to collect the rent.
- Therefore time for humans requires some consensus.
  - Think about the differences between the Thai calender and the western one...


Big Bang
(14 billion years)
Galaxies formed
10
7
6
Earth formed/Solar System formed
5
Primary broth on Earth
Oldest rocks
4
Primordial atmosphere
Oldest fossils
3
Billions of years ago
Plants begin O_2 production
2
Present atmosphere
1
Sun began its last revolution about galactic center
Dinosaurs
Rocky Mts
Earliest hominids
3
2
1
Millions of years ago
Early Homo sapiens
Modern humans (Cro-Magnon)
© 2007 Thomson Higher Education

Measuring Time

- The earths spin is slowing down, so while there was once 400 days per year, we're making less spins per year.
- Solution: the atomic clock.
  - Invented in 1948, the atomic clock counts the number of transitions of a cesium 133 atom. The cesium 133 atom is very stable, and independent of the earths motion, and it enabled physicists to define a second as 9,192,631,770 transitions of the atom. (This was the number of transitions in a measured second in 1948.

The beginning of Time

- To improve time measurements, 50 cesium clocks were placed in labs around the world, and their transitions are counted and then divided by 9,192,631,770 and periodically reported to the Bureau International de l'Heure.
- Since midnight January 1st 1958 the mean result of these cesium clocks has been considered the right time; well at least the International Atomic Time (TAI).

International Atomic Time

- There is still a problem. Currently 86,400 TAI seconds is about 3msec shorter than a solar day.
  - While we can sleep soundly at night knowing that we have a precisely accurate timing system, in a few years when we wake up in the morning we might find that noon has already passed!
- So, whenever the difference between TAI and solar time grows to 800msec, BIH introduces leap seconds

So what is the time?

- Several short wave radio stations issue signals every ‘second’ with call letter WWV.
- This radio receiver can be tracked and if precise time is needed it can be achieved.
- If one machine in the Distributed System has a WWV receiver then the goal is keeping all other machines in sync with it.
- If no machine has a WWV receiver the goal is keeping all machines together as best we can.

Network Time Protocol

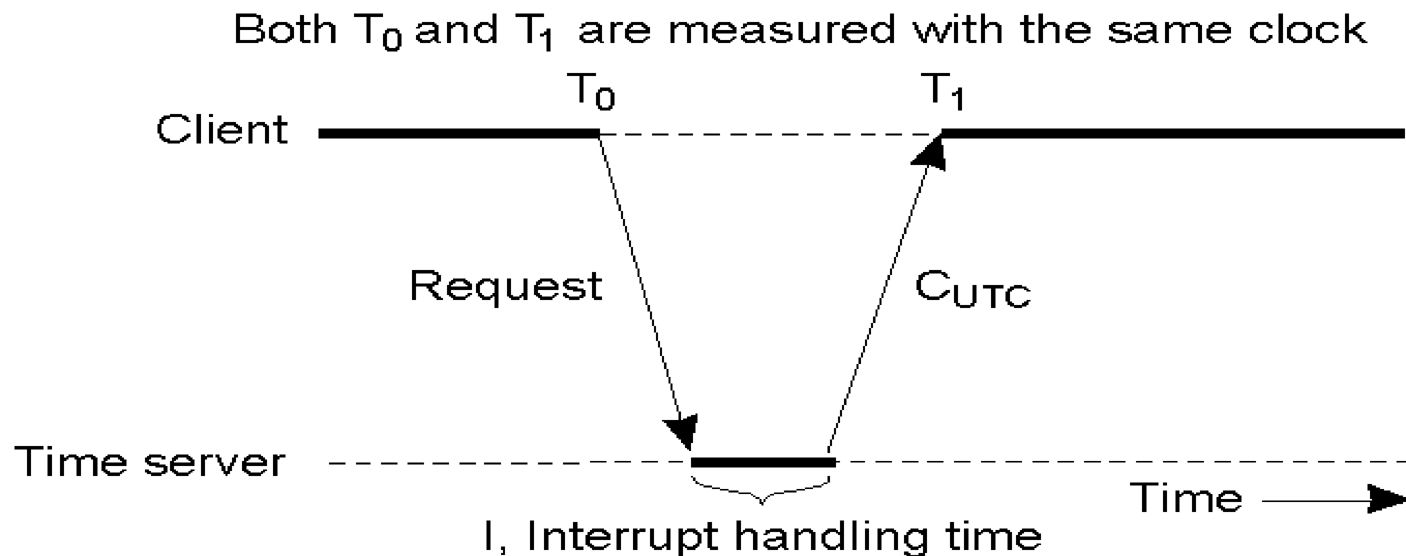
- NTP is a protocol designed to synchronize the clocks of computers over a network.
- http://www.ntp.org/
- Lists of Network Time Protocol (NTP) public time servers are available on the web.

Drift

- Suppose each machine has a timer, which creates an interrupt H times per second.
- An interrupt handler adds 1 to the software clock, C, each time the timer 'ticks'.
- If t is the correct time, then in an ideal world;
- $dC/dt = 1$.
- But in reality a fast clock has $dC/dt = 1+\rho$ and a slow clock has $dC/dt = 1-\rho$.
- So at time Δt after sync 2 clocks could be $2\rho\Delta t$ different.
- If we need the clocks not to differ by more than δ, the they need to be resynced at least every $\delta/2\rho$ seconds

Christian's Algorithm

Assuming we have a time server (perhaps with a WWV receiver) each machine sends a request to the time server at least every δ/2ρ seconds.

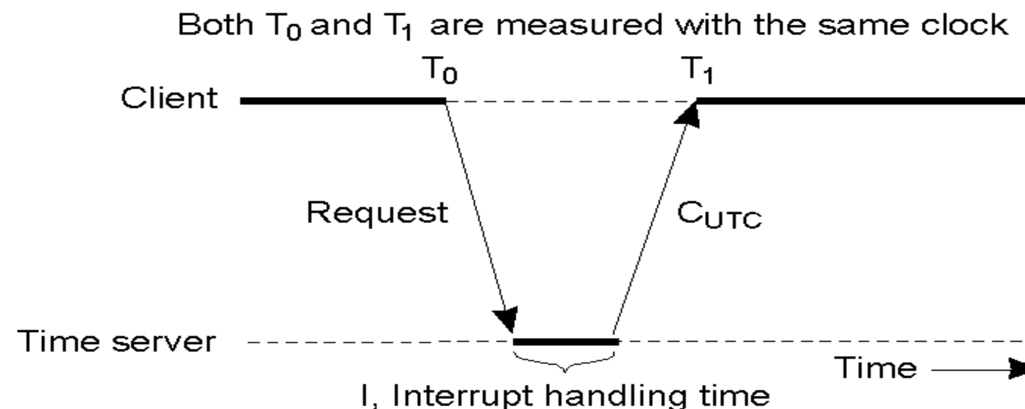


Problems encountered...

- First, time can’t go backwards so a fast clock can’t simply be changed back to an earlier time – imagine the complications of 2 files being compiled sequentially, but the clock change meaning the 2nd one is time stamped before the first one.
  - Normally changes to the time are introduced gradually by adding or taking from the interrupt interval.

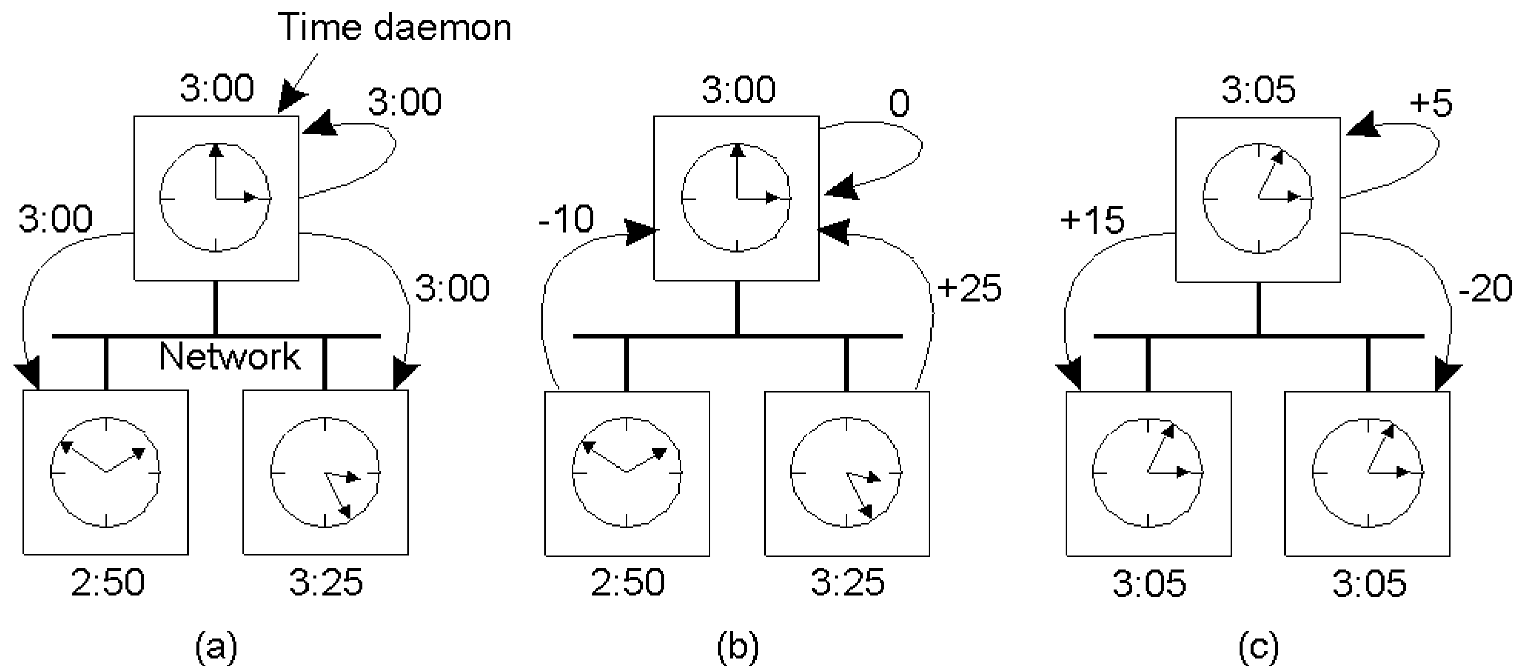
Problems encountered...2

- The time server request will take a nonzero amount of time which can vary according to the network load.
- Christian's Algorithm attempts to measure this problem, by $(T_1 - T_0 - I)/2$ where T_1 and T_0 can be measured on the same clock, and I estimated by the server.



Berkeley Algorithm

- While the time server in Christian's algorithm is passive, Berkeley's algorithm takes a different approach where the time daemon polls every machine from time to time to ask what time it is. From this 'average time' can be calculated.



Logical Clocks

- All very interesting, but for many purposes the key is for a consensus or agreement about time to be reached.
  - It doesn’t matter if its really ‘10:00’ if we all agree that it is ‘10:02’ or even 0101101.
- What’s important is that the processes in the Distributed System *agree on the ordering in which certain events occur*.
- Such “clocks” are referred to as *Logical Clocks,* based on “relative time”.

Lamport’s Timestamps

- **First point**: if two processes do not interact, then their clocks do not need to be synchronized – they can operate *concurrently* without fear of interferring with each other.
- **Second (critical) point**: it does not matter that two processes share a common notion of what the “real” current time is. What does matter is that the processes have some agreement on the order in which certain events occur.
- Lamport used these two observations to define the “happens-before” relation (also often referred to within the context of *Lamport’s Timestamps*).

“Happens Before”

- If A and B are events in the same process, and A occurs before B, then we can state that:
  - *A “happens-before” B is true*.
  - $A \rightarrow B$
- Equally, if A is the event of a *message being sent by one process*, and B is the event of the same *message being received by another process*, then A “happens-before” B is also true.
- (Note that a message cannot be received before it is sent, since it takes a finite, nonzero amount of time to arrive … and, of course, time is not allowed to run backwards).

“Happens Before”(2)

- Obviously, if A “happens-before” B and B “happens-before” C, then it follows that A “happens-before” C.
  - $A \rightarrow B$ and $B \rightarrow C$ so $A \rightarrow C$.
- If the “happens-before” relation holds, deductions about the current clock “value” on each DS component can then be made.
  - It therefore follows that if C(A) is the time on A, then $C(A) < C(B)$, and so on.

“Happens Before”(3)

- Assume three processes are in a DS: A, B and C.
  - All have their own physical clocks (which are running at differing rates due to “clock skew”, etc.).
- A sends a message to B and includes a “timestamp”.
  - If this sending timestamp is less than the time of arrival at B, things are OK, as the “happens-before” relation still holds (i.e. A “happens-before” B is true).
  - However, if the timestamp is more than the time of arrival at B, things are NOT OK (as A “happens-before” B is not true, and this cannot be as the receipt of a message has to occur *after* it was sent).

“Happens Before”(4)

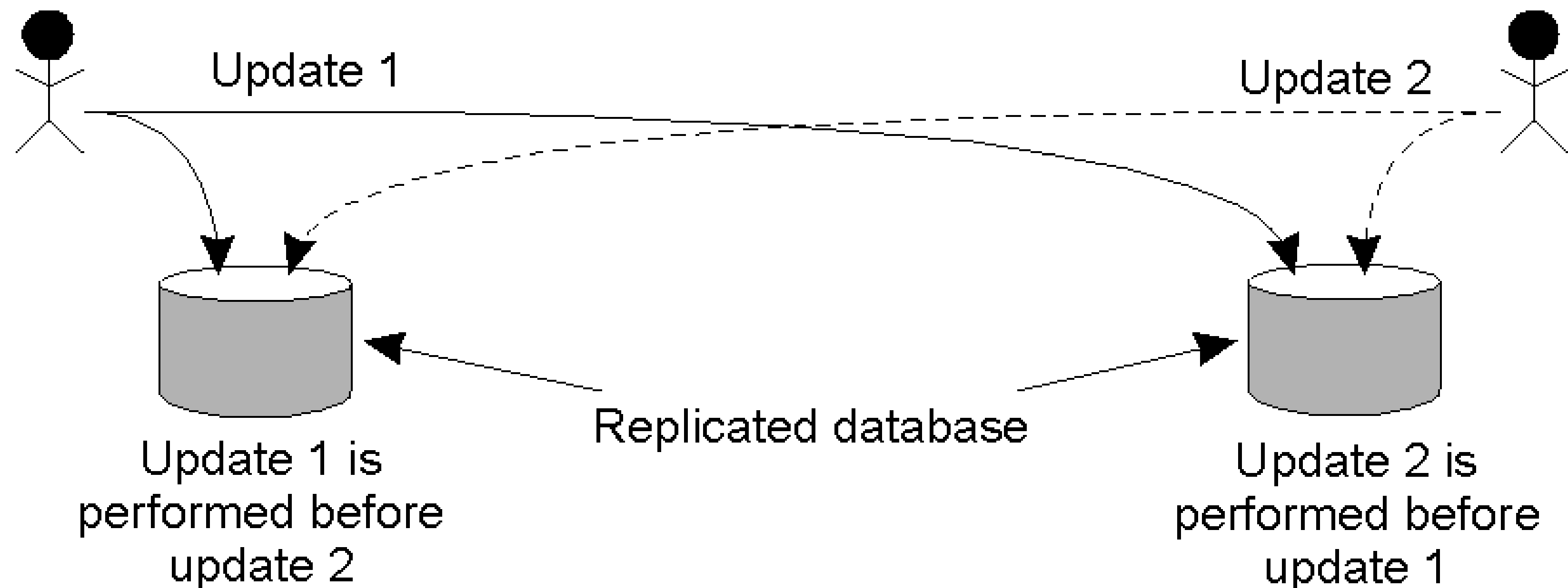
- **The question to ask is**:
  - How can some event that “happens-before” some other event possibly have occurred at a later time?

Answer...

- **The answer is**: it can't!
- So, Lamport's solution is to have *the receiving process adjust its clock forward to one more than the sending timestamp value*. This allows the "happens-before" relation to hold, and also keeps all the clocks running in a synchronised state. The clocks are all kept in sync *relative to each other*.

Problems with Synchronisation

- Updating a replicated database and leaving it in an inconsistent state: Update 1 adds 100 baht to an account, Update 2 calculates and adds 1% interest to the same account. Due to network delays, the updates may not happen in the correct order!

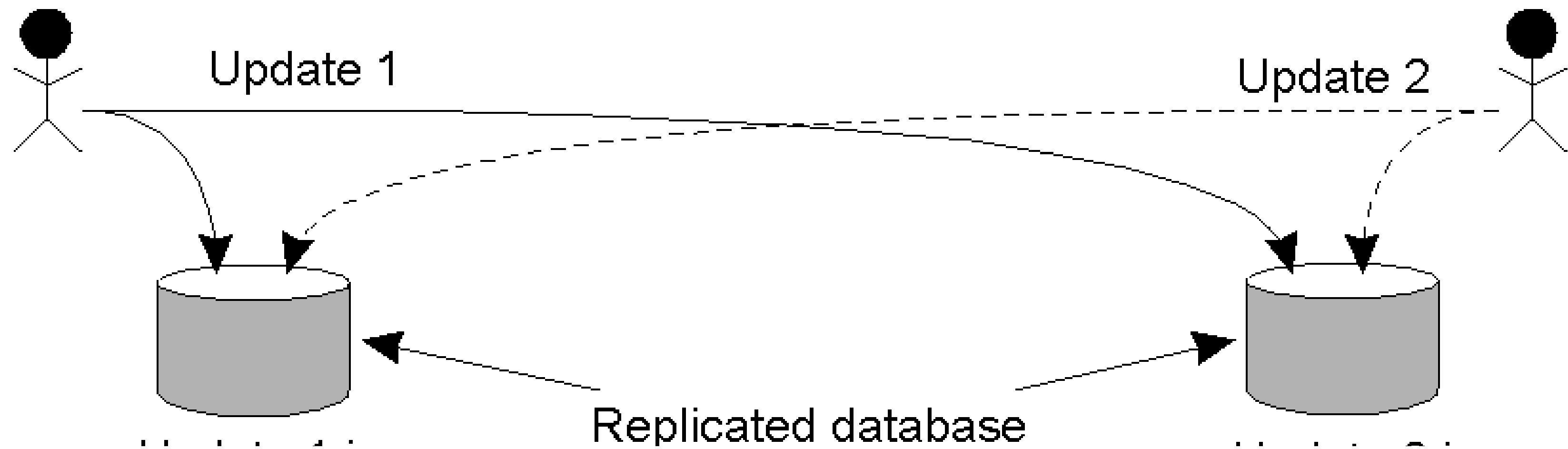


Solution (1)

- A multicast message is sent to all processes in the group, including the sender, together with the sender's timestamp.
- At each process, the received message is added to a local queue, ordered by timestamp.
- Upon receipt of a message, a multicast acknowledgement/timestamp is sent to the group.
- Due to the "happens-before" relationship holding, the timestamp of the acknowledgement is always greater than that of the original message.

Solution (2)

- Only when a message is marked as acknowledged by all the other processes will it be removed from the queue and delivered to a waiting application.
- Lamport's clocks ensure that each message has a unique timestamp, and consequently, the local queue at each process eventually contains the same contents.
- In this way, all messages are delivered/processed in the same order everywhere, and updates can occur in a consistent manner.



- Update 1 is time-stamped and multicast. Added to local queues.
  - Update 2 is time-stamped and multicast. Added to local queues.
  - Acknowledgements for Update 2 sent/received. Update 2 can now be processed.
  - Acknowledgements for Update 1 sent/received. Update 1 can now be processed.
- (Note: all queues are the same, as the timestamps have been used to ensure the “happens-before” relation holds.)

Global State

Global State

- Related to synchronisation is the concept of a global state;
  - The state of the entire distributed system
- It may be useful to know the entire global state of a system;
  - For instance when performing garbage collection as introduced last week.
  - Or to determine if a system has ‘hung’ or completed correctly.
- A way of determining global state is to take a “distributed snapshot” of the system.

Distributed Snapshots

- To take a snapshot, we need to know the state of each processor within the system.
  - So, any process can request a snapshot, by asking for the current state of the other processors.
- Clearly a message that has been sent but not yet received is acceptable, but a message received but not yet sent should not be possible.
  - Logical clocks can assist with this problem.

Election Algorithms

- Many Distributed Systems require a process to act as *coordinator* (for various reasons). The selection of this process can be performed automatically by an "election algorithm".
- For simplicity, we assume the following:
  - Processes each have a unique, positive identifier.
  - All processes know all other process identifiers.
  - The process with the highest valued identifier is duly elected coordinator.
- When an election "concludes", a coordinator has been chosen and is known to all processes.

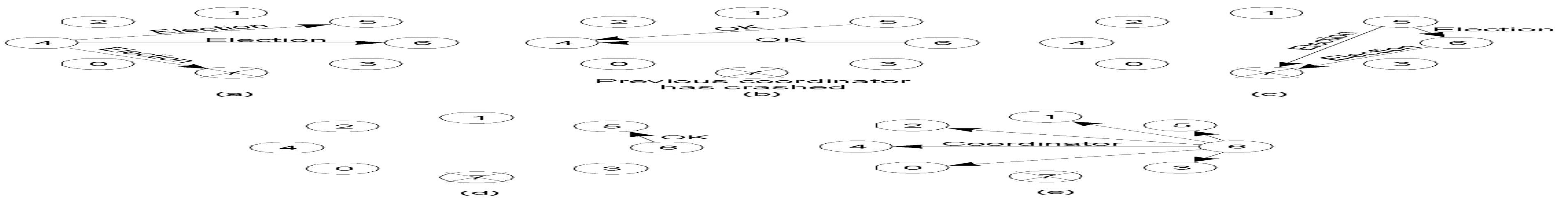
Why Elections?

- The overriding goal of all election algorithms is to have all the processes in a group *agree* on a coordinator.

- There are two types of algorithm:

1. **Bully**: “the biggest guy in town wins”.
2. **Ring**: a logical, cyclic grouping.

Bully Elections



Ring Elections

- The processes are ordered in a “logical ring”, with each process knowing the identifier of its successor (and the identifiers of all the other processes in the ring).
- When a process “notices” that a coordinator is down, it creates an ELECTION message (which contains its own number) and starts to circulate the message around the ring.

Next Lecture

- Protecting shared Resources
- Transaction Types