CIS 505: Software Systems
Lecture Note on Physical Clocks

Insup Lee
Department of Computer and Information Science
University of Pennsylvania

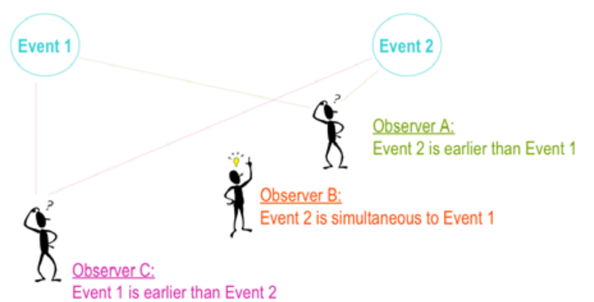
CIS 505, Spring 2007

Distributed Synchronization

- Communication between processes in a distributed system can have unpredictable delays, processes can fail, messages may be lost
- Synchronization in distributed systems is harder than in centralized systems because the need for distributed algorithms.
- Properties of distributed algorithms:
  1. The relevant information is scattered among multiple machines.
  2. Processes make decisions based only on locally available information.
  3. A single point of failure in the system should be avoided.
  4. No common clock or other precise global time source exists.
- Challenge: How to design schemes so that multiple systems can coordinate/synchronize to solve problems efficiently?

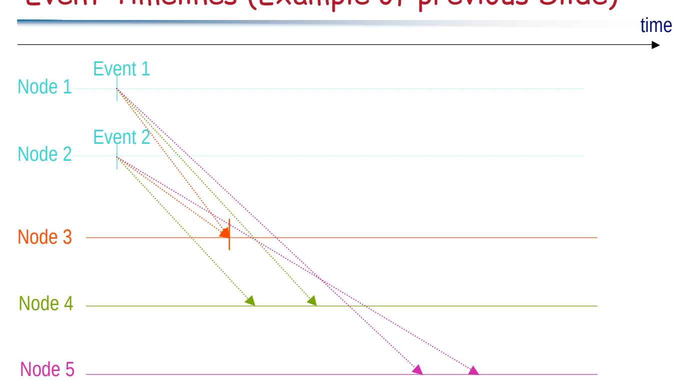
The Myth of Simultaneity

"Event 1 and event 2 at same time"



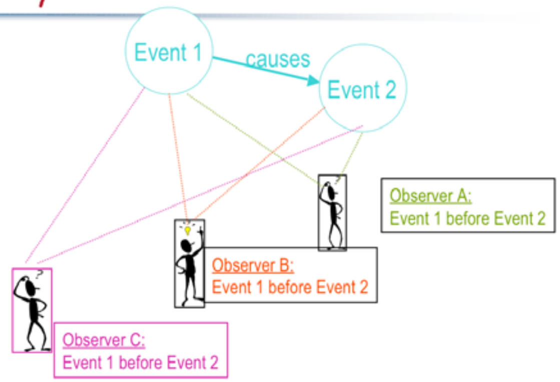
$e1 \parallel e2 = e1 \rightarrow e2 \mid e2 \rightarrow e1$

Event Timelines (Example of previous Slide)



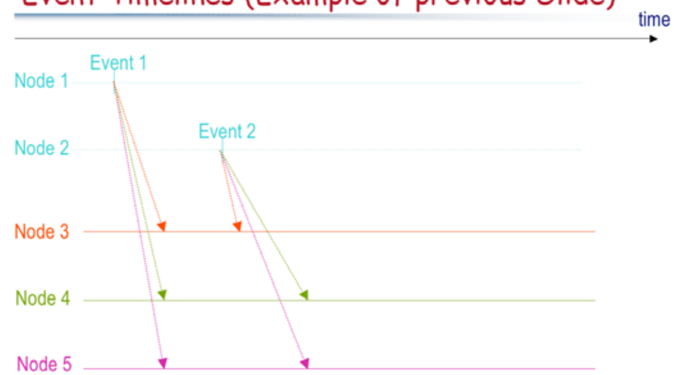
Note: The arrows start from an event and end at an observation.
The slope of the arrows depend of relative speed of propagation

Causality



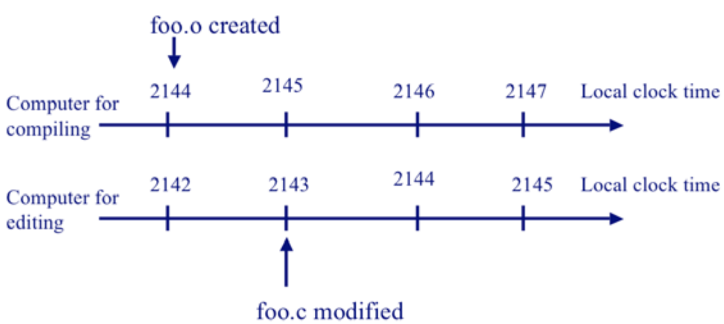
Requirement: We have to establish causality, i.e., each observer must see event 1 before event 2

Event Timelines (Example of previous Slide)



Note: In the timeline view, event 2 must be caused by some passage of information from event 1 if it is caused by event 1

Why need to synchronize clocks?



Physical Time

Some systems really need quite accurate absolute times.

How to achieve high accuracy?
Which physical entity may deliver precise timing?

1. The sun } Today: 1 sec ~ 1 day / 86400
but rotation of earth is slowing down

2. An Atom } State transitions in atoms (defined by BIH in Paris)
1 sec = time a cesium atom needs for 9 192631 770 state transitions*

BIH (Bureau International de l"Heure)
* TAI (International Atomic Time)

Problem with Physical Time

A TAI-day is about 3 msec shorter than a day

=>

BHI inserts 1 sec, if the difference between a day and a TAI-day is more than 800msec

=>

UTC (Universal Time Coordinated) is the base of any international time measure.

Physical Time

UTC-signals come from radio broadcasting stations or from satellites (GEOS, GPS) with an accuracy of:

- 1.0 msec (broadcasting station)
- 1.0 µsec (GPS)

GPS on all computers?

Clock Skew Problem

Clock skew(offset): the difference between the times on two clocks

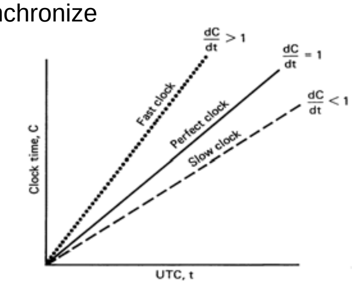
Clock drift : they count time at different rates

Ordinary quartz clocks drift by ~ 1sec in 11-12 days. (10^{-6} secs/sec).

High precision quartz clocks drift rate is ~ 10^{-7} or 10^{-8} secs/sec

Physical clock drift rate

- Maximum drift rate
  - One can determine how often they should be synchronize



Not all clock's tick precisely at the current rate.

Clock Synchronization

Adjusting physical clocks:
- local clock behind reference clock
- local clock ahead of reference clock

Observation:
Clocks in DS tend to drift apart and need to be resynchronized periodically

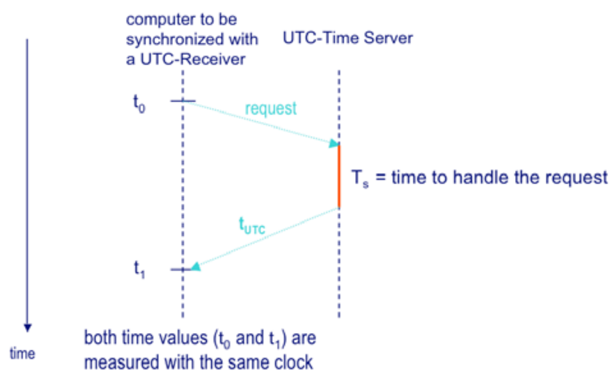
A. If local clock is behind a reference clock:
- could be adjusted in one jump or
- could be adjusted in a series of small jumps

B. What to do if local clock is ahead of reference clock?
Monotonicity.

Computer clocks

- How a computer timer works?
  - A counter register and a holding register.
  - The counter is decremented by a quartz crystals oscillator. When it reaches zero, an interrupted is generated and the counter is reloaded from the holding register.
  - E.g., interrupt 60 times per second.
    - Use 61 to run the wall-clock time faster by making the value of the holding register smaller.
- The clock skew problem
  - logical clocks -- to provide consistent event ordering
  - physical clocks -- clocks whose values must not deviate from the real time by more than a certain amount.

Absolute Clock Synchronization (Cristian's Algorithm)



Cristian's Algorithm

- Initialize local clock: $t := t_{UTC}$
  - (Problem: Message Transfer-Time)
- Estimate Message transfer-time, $(t_1 - t_0)/2 \Rightarrow t := t_{UTC} + (t_1 - t_0)/2$
  - (Problem: Time of the Request Message *tr*)
- Suppose: *tr* is known, $\Rightarrow t := t_{UTC} + (t_1 - t_0 - t_r)$
  - (Problem: Message transfer-times are load dependent)
- To improve accuracy: Multiple measurements $(t_1 - t_0)$:
  - Throw away measurements above a threshold value
  - Take all others to get an average
- Assume $(t_1 - t_0)$ ranges from *[min, max]*
  - What is the accuracy of *t* ?
- Centralized time server
  - What if the server crashes?
  - What if the server gives the wrong time?

Relative Clock Synchronization (Berkeley Algorithm)

If you need a uniform time (without a UTC-receiver per computer), but you cannot establish a central time-server:

- Peers elect a master
- Master polls all nodes to give him their times by the clock
- The master estimates the local times of all nodes regarding the involved message transfer times.
- Master uses the estimated local times for building the arithmetic mean
  - Add fault tolerance
- The deviations from the mean are sent to the nodes
  - Is this better than sending the actual time?

The Berkeley Algorithm

- Averaging algorithm
  - The time daemon asks all the other machines for their clock values.
  - The machines answer.
  - The Time daemon tells everyone how to adjust their clock.