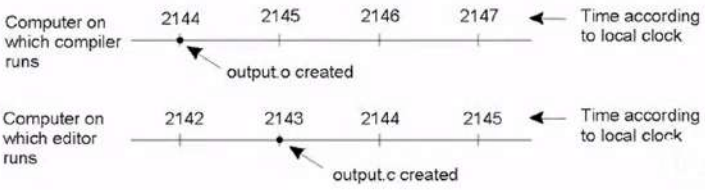
Clock Synchronization

- Time in unambiguous in centralized systems
  - System clock keeps time, all entities use this for time
- Distributed systems: each node has own system clock
  - Crystal-based clocks are less accurate (1 part in million)
  - *Problem:* An event that occurred after another may be assigned an earlier time



Powered by Zoom

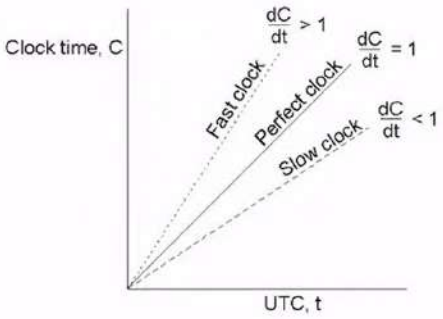
Physical Clocks: A Primer

- How do you tell time?
  - Use astronomical metrics (solar day)
- Accurate clocks are atomic oscillators (one part in 10^{13})
- Coordinated universal time *(UTC)* – international standard based on atomic time
  - Add leap seconds to be consistent with astronomical time
  - UTC broadcast on radio (satellite and earth)
  - Receivers accurate to 0.1 – 10 ms
- Most clocks are less accurate (e.g., mechanical watches)
  - Computers use crystal-based blocks (one part in million)
  - Results in *clock drift*
- Need to synchronize machines with a master or with one another

Powered by Zoom

Clock Synchronization

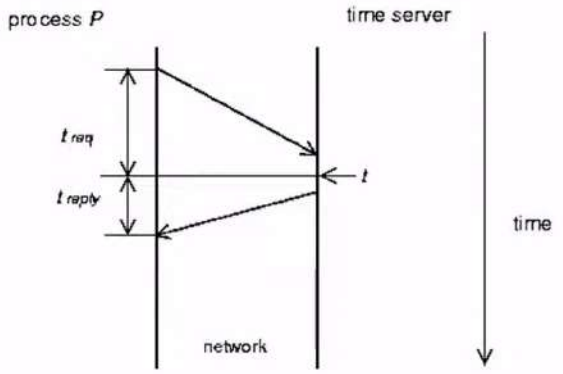
- Each clock has a maximum drift rate ρ
  - 1-ρ <= dC/dt <= 1+ρ
  - Two clocks may drift by 2ρ Δt in time Δt
  - To limit drift to δ => resynchronize every δ/2ρ seconds



Powered by Zoom

Cristian's Algorithm

- Synchronize machines to a *time server* with a UTC receiver
- Machine P requests time from server every $\delta/2\rho$ seconds
  - Receives time t from server, P sets clock to $t+t_{reply}$ where t_{reply} is the time to send reply to P
  - Use $(t_{req}+t_{reply})/2$ as an estimate of t_{reply}
  - Improve accuracy by making a series of measurements



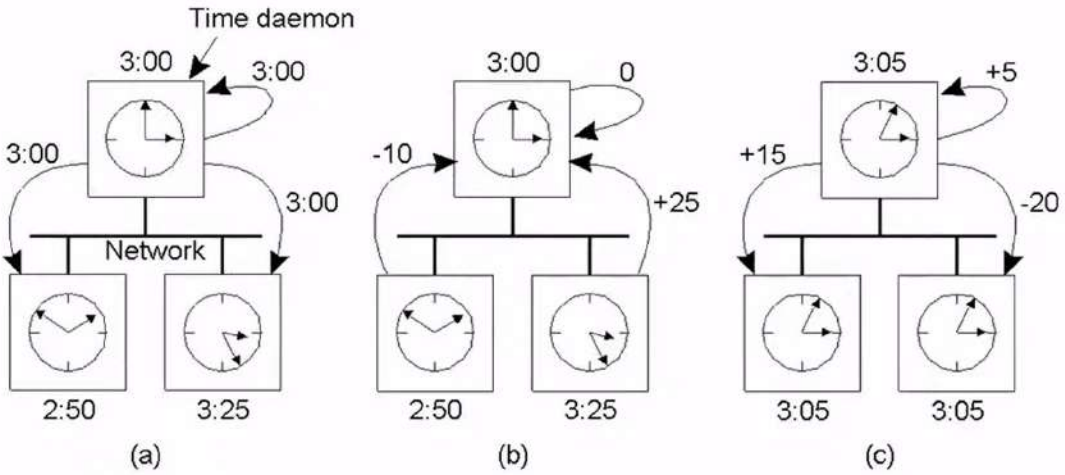
Powered by Zoom

Berkeley Algorithm

- Used in systems without UTC receiver
  - Keep clocks synchronized with one another
  - One computer is *coordinator*, other are *workers*
  - Master periodically polls slaves for their times
    - Average times and return differences to slaves
    - Communication delays compensated as in Cristian's algo
  - Failure of master => election of a new master

Powered by Zoom

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

Powered by Zoom

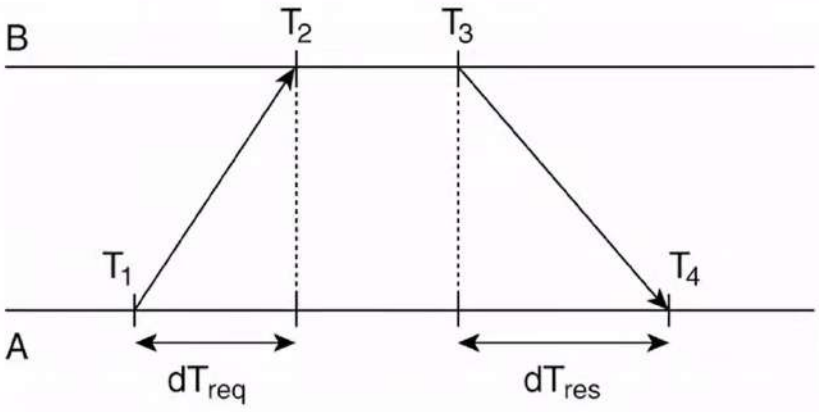
Distributed Approaches

- Both approaches studied thus far are centralized
- Decentralized algorithms: use resync intervals
  - Broadcast time at the start of the interval
  - Collect all other broadcast that arrive in a period *S*
  - Use average value of all reported times
  - Can throw away few highest and lowest values
- Approaches in use today
  - *rdate*: synchronizes a machine with a specified machine
  - Network Time Protocol (NTP) - discussed in next slide
    - Uses advanced techniques for accuracies of 1-50 ms

Powered by Zoom

Network Time Protocol

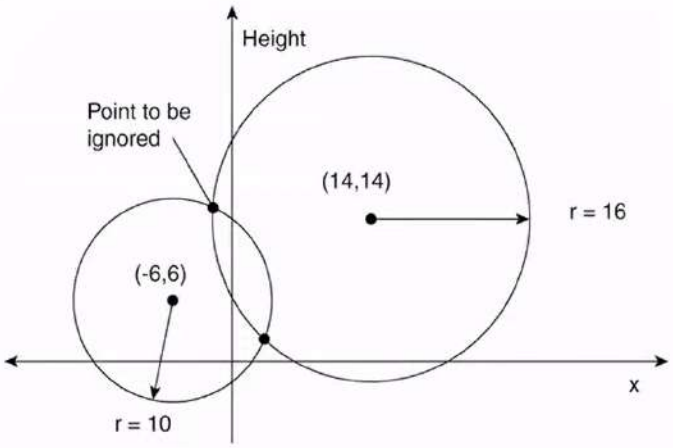
- Widely used standard - based on Cristian's algo
  - Uses eight pairs of delays from A to B and B to A.
- Hierarchical – uses notion of stratum
- **Clock can not go backward**



Powered by Zoom

Global Positioning System

- Computing a position in a two-dimensional space.



Powered by Zoom

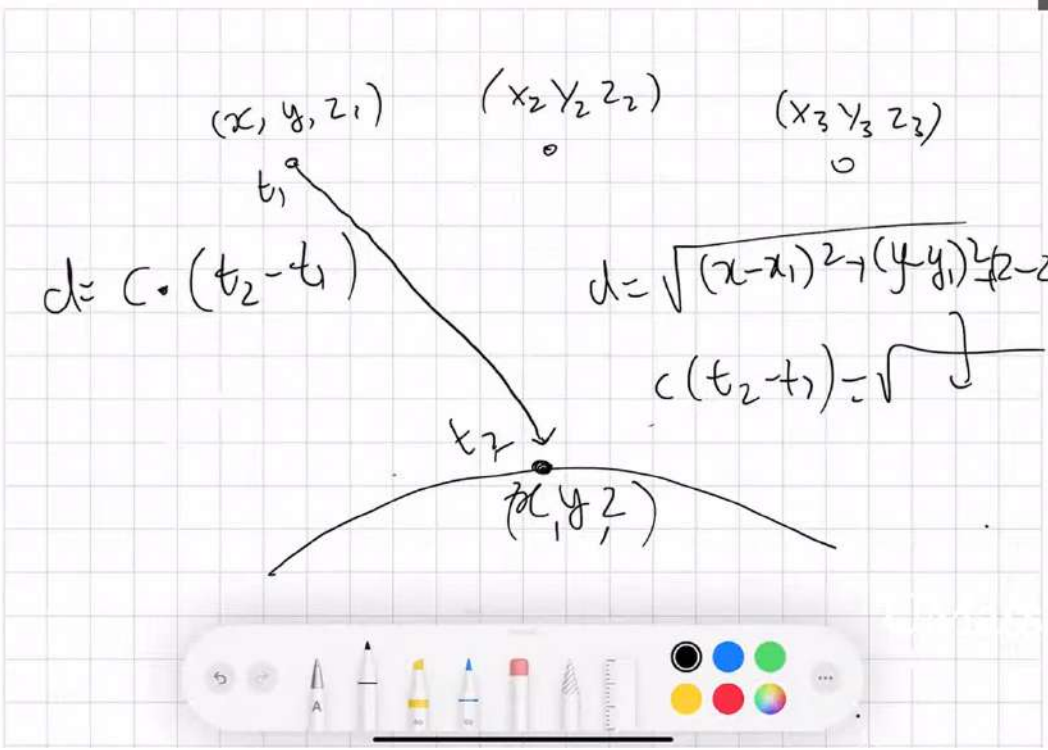
Global Positioning System

- Real world facts that complicate GPS
- It takes a while before data on a satellite's position reaches the receiver.
- The receiver's clock is generally not in synch with that of a satellite.

Powered by Zoom

3:15 PM Mon Mar 7

Lecture 12 Notes



nherst

Powered by Zoom

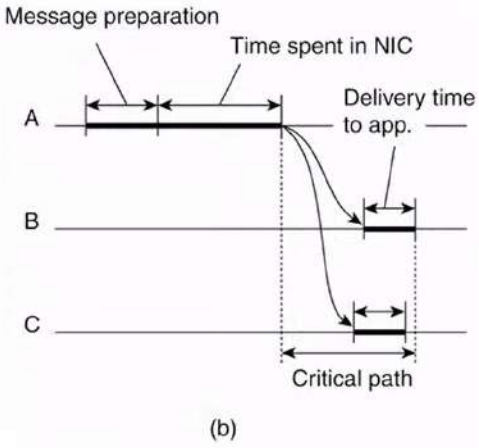
GPS Basics

- D_r – deviation of receiver from actual time
- Beacon with timestamp T_i received at T_{now}
  - Delay $D_i = (T_{now} - T_i) + D_r$
  - Distance $d_i = c\ (T_{now} - T_i)$
  - Also $d_i = \text{sqrt}[\ (x_i - x_r)^2 + (y_i - y_r)^2 + (z_i - z_r)^2\ ]$
- Four unknowns, need 4 satellites.
-

Powered by Zoom

Clock Synchronization in Wireless Networ

- Reference broadcast sync (RBS): receivers synchronize with one another using RB server
  - Mutual offset = $T_{i,s} - T_{j,s}$ (can average over multiple readings)



(b)

Powered by Zoom

Logical Clocks

- For many problems, internal consistency of clocks is important
  - Absolute time is less important
  - Use *logical* clocks
- Key idea:
  - Clock synchronization need not be absolute
  - If two machines do not interact, no need to synchronize them
  - More importantly, processes need to agree on the *order* in which events occur rather than the *time* at which they occurred

Powered by Zoom

Event Ordering

- *Problem:* define a total ordering of all events that occur in a system
- Events in a single processor machine are totally ordered
- In a distributed system:
  - No global clock, local clocks may be unsynchronized
  - Can not order events on different machines using local times
- Key idea [Lamport]
  - Processes exchange messages
  - Message must be sent before received
  - Send/receive used to order events (and synchronize clocks)

Powered by Zoom

Happened Before Relation

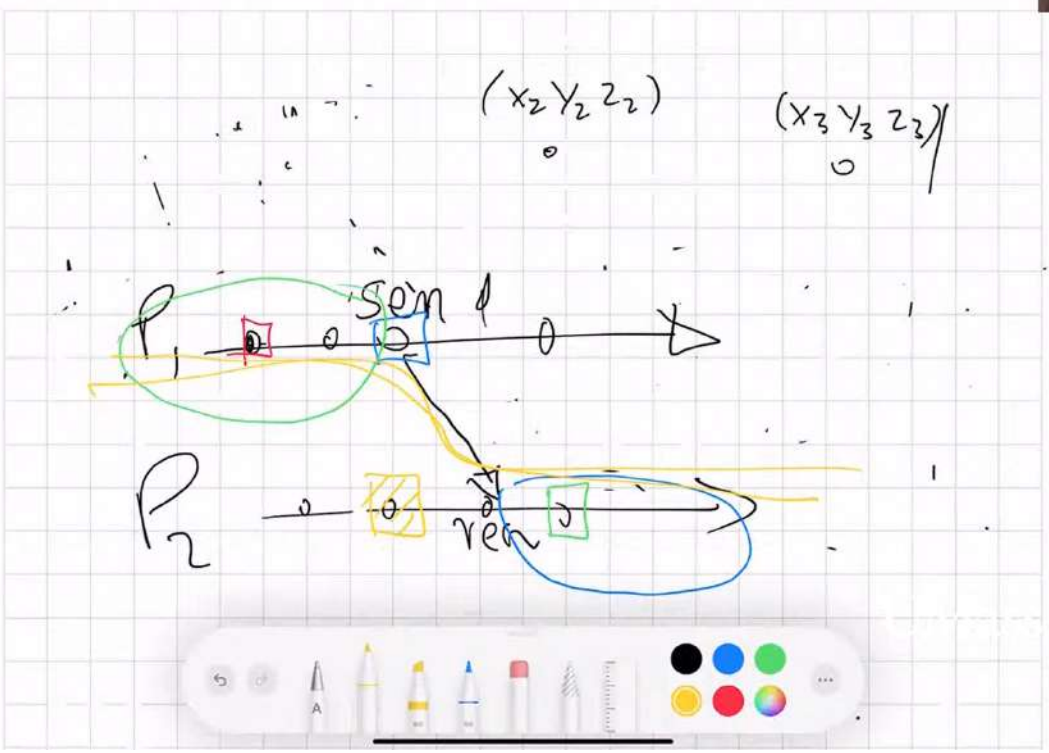
- If *A* and *B* are events in the same process and *A* executed before *B*, then *A* -> *B*

- If A represents sending of a message and B is the receipt of this message, then A -> B
- Relation is transitive:
  - A -> B and B -> C => A -> C
- Relation is undefined across processes that do not exchange messages
  - Partial ordering on events

Powered by Zoom

3:37 PM Mon Mar 7

Lecture 12 Notes



herst

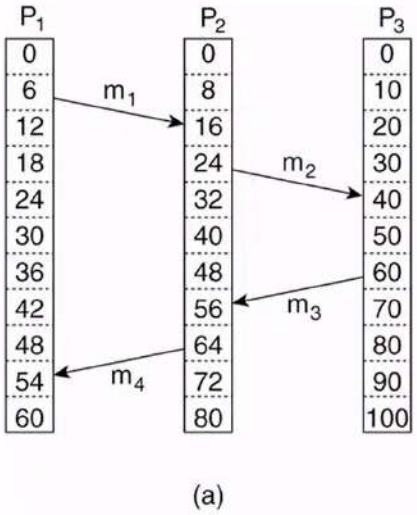
Powered by Zoom

Event Ordering Using *HB*

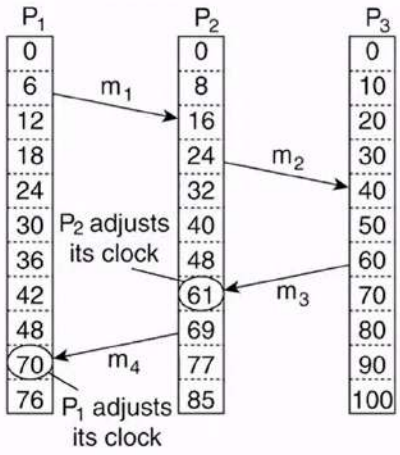
- Goal: define the notion of time of an event such that
  - If A-> B then C(A) < C(B)
  - If A and B are concurrent, then C(A) <, = or > C(B)
- Solution:
  - Each processor maintains a logical clock LC_i
  - Whenever an event occurs locally at I, $LC_i = LC_i + 1$
  - When *i* sends message to *j*, piggyback Lc_i
  - When *j* receives message from *i*
    - If $LC_j < LC_i$ then $LC_j = LC_i + 1$ else do nothing
  - Claim: this algorithm meets the above goals

Powered by Zoom

Lamport's Logical Clocks



(a)



(b)

Powered by Zoom