Time and Clocks

- uses of time in distributed systems:
  - time-based algorithms (e.g. in security)
  - distributed make
  - gathering event traces for debugging
  - proving or disproving causality (insider trading)
- common element: need to know in which order events happened

Absolute Time

- this is what clocks tell us
- strategy: keep clocks synchronized; put time-stamp on each event
- problems
  - hard to synchronize distributed clocks
  - clock speeds vary unpredictably
  - handling time zones, daylight savings time, year 2000, etc.

Synchronizing Clocks

- many protocols exist
  - NTP (Network Time Protocol) standard
  - will discuss simpler protocol here
- use reference clock as baseline
  - "Coordinated Universal Time" (UCT)
  - from atomic clocks run by NIST
- other machines try to sync with UCT

Clock Synchronization Protocol

- A sends to B: "My clock says <value>"
  - only possible building block
- problem: message takes time to get to B
  - network delay is unknown and variable
- work-around: measure round-trip time between A and B, assume it doesn't vary much and that delay is equal in both directions
  - not completely accurate

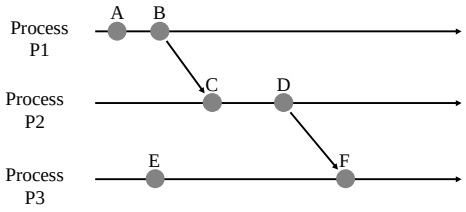
Clock Synchronization Problems

- synchronization is necessarily inaccurate
  - "happens before" judgements might be wrong
- can get out of sync badly if network is partitioned
- vulnerable to dishonest time-servers
- bottom line: OK for some applications
- alternative: logical time

Logical Time

- insight: often don't care about when something happened, only about which thing happened first
- logical time talks about "happened before" relationships, without reference to absolute time
- (analogies to Einstein's relativity are common but bogus)

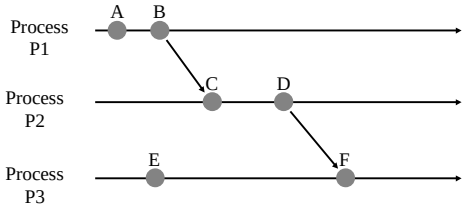
Example



The "Happened Before" Relation

- X --> Y means "X happened before Y
  - captures logical ordering, not temporal
- three rules:
  - if X and Y occur in the same process, and X occurs before Y, then X --> Y
  - if M is a message, then send(M) --> receive(M)
  - if X --> Y and Y --> Z, then X --> Z

Example



Logical Time Relationships

- Given two events X and Y, either
  - X --> Y, or
  - Y --> X, or
  - neither
    - » "X and Y are concurrent"
    - » X could not have caused Y, and vice versa
- --> relation defines a partial order
- How to determine --> in practice?

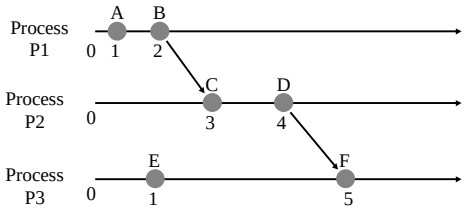
Logical Timestamp Algorithms

- simple algorithms to capture -->
- assign numerical timestamp to each event
  - no relation to absolute time
- simple timestamps
  - if X --> Y, then TS(X) < TS(Y)
- vector timestamps
  - X --> Y if and only if TS(X) --> TS(Y)

Simple Logical Timestamps

- timestamp is an integer
- each process has a "logical clock"
  - starts at zero
  - incremented on each local event
- each message has a timestamp
  - equal to sender's logical clock when sent
  - on receive, receiver's logical clock set to 1 + max(message timestamp, receiver's previous logical clock)

Logical Timestamp Example



Simple Logical Timestamps

- successfully capture all --> relationships
- also capture some false relationships
  - TS(X) < TS(Y) but not X --> Y
- good scheme to use if "extra" ordering isn't a problem
- otherwise, need something fancier

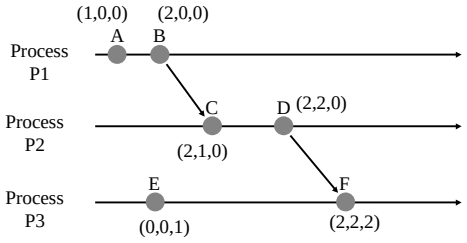
Vector Timestamps

- captures --> exactly
- more complicated than simple timestamps
  - uses more time and space
- represent a logical time as a vector with P entries (assuming P processes)
- each process has logical clock
- each message has a logical timestamp

Vector Timestamp Algorithm

- local event in process I
  - process I increments the I'th element of its logical clock
- message sent
  - message timestamp = logical clock of sender
- message received
  - for all J, receiver sets J'th element of logical clock to max of
    - » J'th component in message timestamp
    - » J'th component in receiver's logical clock

Vector Time Example



Vector Time and Ordering

- given two events X and Y,
  - X --> Y iff some X[i]<Y[i] and all X[i]<=Y[i]
  - Y --> X iff some Y[i]<X[i] and all Y[i]<=Y[i]
  - X || Y iff some X[i]<Y[i] and some X[j]>Y[j]
- captures happened-before relation exactly

Interpreting Vector Time

- each process numbers its events sequentially
  - represented by I'th element of process I's clock
- each process keeps track of which events on other processes have happened before the present time
  - if the I'th element of P's clock is N, that means that the first N events at I happened before the present on process P

Application: Event Logging Tool

- maintain vector logical clocks
- each process dumps events of interest to a local file
  - mark with logical timestamp
- postmortem analysis tool can interleave the local traces correctly
  - can answer questions of possible causality

Critique of Logical Time

- fine for some applications, but
- doesn't capture all of the real relationships
  - "messages" can flow outside the system
    - » insider trading example
- happened before doesn't capture causality
  - absolute time has this problem too