Time and synchronization

("There's never enough time…")

Why Global Timing?

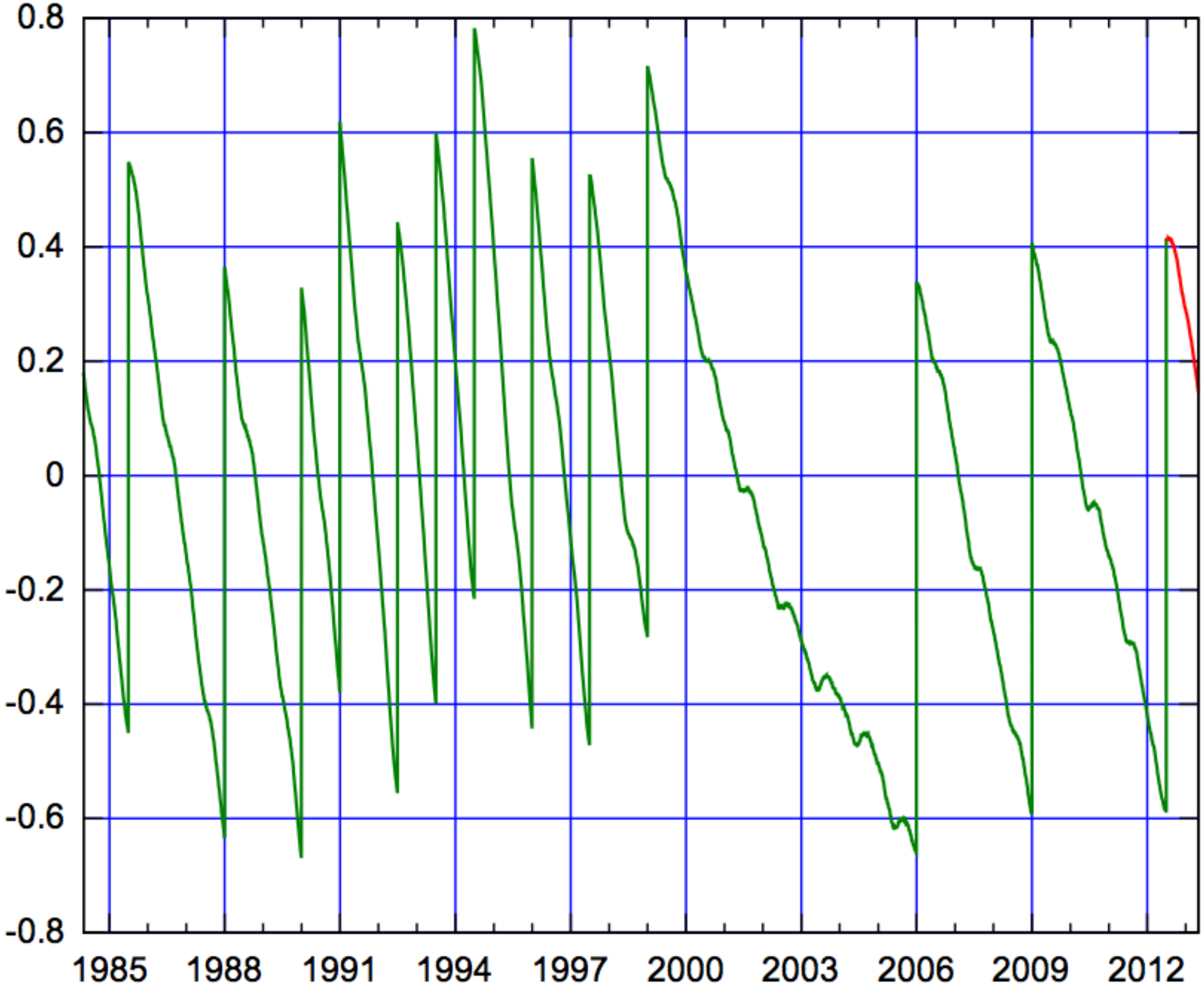
- Suppose there were a globally consistent time standard
- Would be handy
  - Who got last seat on airplane?
  - Who submitted final auction bid before deadline?
  - Did defense move before snap?

Time Standards

- UT1
  - Based on astronomical observations
  - “Greenwich Mean Time”
- TAI
  - Started Jan 1, 1958
  - Each second is 9,192,631,770 cycles of radiation emitted by Cesium atom
  - Has diverged from UT1 due to slowing of earth’s rotation
- UTC
  - TAI + leap seconds to be within 800ms of UT1
  - Currently 35
  - Most recent: June 30, 2012

Comparing Time Standards

UT1 – UTC



Clocks

- Piezoelectric effect:
  - Squeeze a quartz crystal: generates electric field
  - Apply electric field: crystal bends
- Quartz crystal clock:
  - Resonation like a tuning fork
  - Accurate to parts per million
  - Gain/lose ½ second per day

Challenges

- Two clocks do not agree perfectly
- **Skew:** The time difference between two clocks
- Quartz oscillators vibrate at different rates
- **Drift:** The difference in rates of two clocks
- If we had two perfect clocks:
  - Skew = 0
  - Drift = 0

When we detect a clock has a skew

- Eg: it is 5 seconds behind
- Or 5 seconds ahead

- What can we do?

When we detect a clock has a skew

- e.g. it is 5 seconds behind
  - We can advance it 5 seconds to correct
  - Might skip over event scheduled in-between
- Or 5 seconds ahead
  - Pushing back 5 seconds is a bad idea
    - Message was received before it was sent
    - Document closed before it was saved etc..

  - We want **monotonicity**: time always increases
  - We want **continuity**: time doesn't make jumps

When we detect a clock has a skew

- e.g. it is behind
  - Run it faster until it catches up
- It is ahead
  - Run it slower until it catches up

- This does not guarantee correct clock in future
  - Need to check and adjust periodically

Distributed time

- Premise
  - The notion of time is well-defined (and measurable) at each single location
  - But the relationship between time at different locations is unclear
    - Can minimize discrepancies, but never eliminate them
- Reality
  - Stationary GPS receivers can get global time with < $1\mu s$ error
  - Few systems designed to use this

A football example

- Five locations: Goal Keeper K1, Player P1, Player P2, Player P3 and Goal Keeper K2
- Ten events:
  - e_1: keeper K1 kicks the ball
  - e_2: ball arrives to Player P1
  - e_3: P1 kicks the ball to Player P2
  - e_4: P2 runs with the ball
  - e_5: P2 kicks the ball towards Player P3
  - e_6: P3 kicks the ball towards other keeper K2
  - e_7: The ball arrives near the keeper
  - e_8: But another Player P4 touches the ball
  - e_9: Ball also touches the keeper K2
  - e_{10}: Ball went inside the goal

A football example

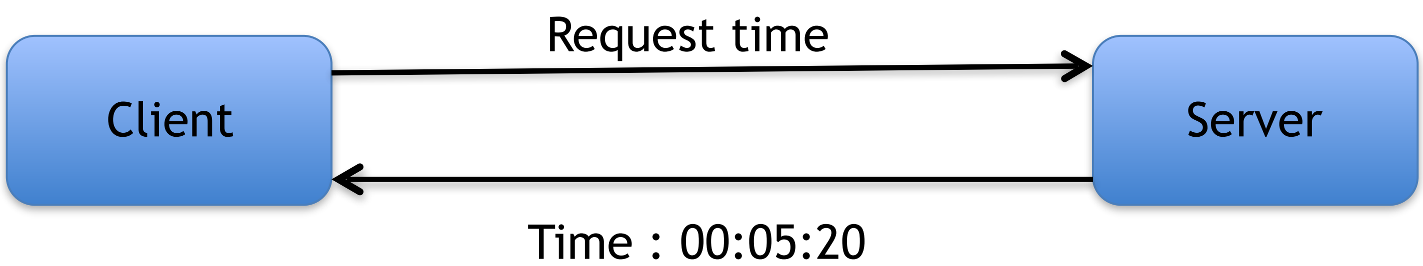
- Players and umpire knows e_1 happens before e_6, which happens before e_7
- Umpire knows e_2 is before e_3, which is before e_4, which is before e_8, …
- Relationship between e_8 and e_9 is unclear

Ways to synchronize

- Send message from a player to another?
  - Or to a central timekeeper
  - How long does this message take to arrive?
- Synchronize clocks before the game?
  - Clocks drift
    - million to one => 1 second in 11 days
- Synchronize continuously during the game?
  - GPS, pulsars, etc

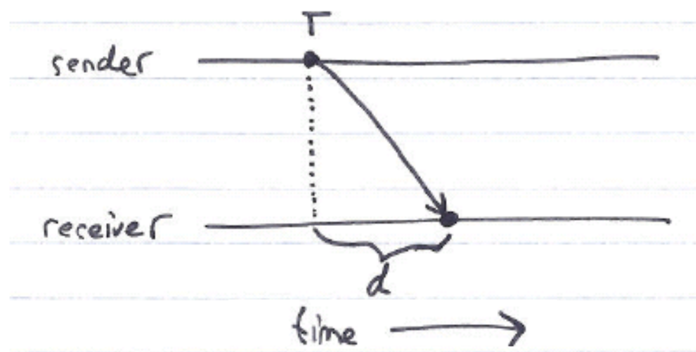
How clocks synchronise

- Obtain time from time server:



Perfect networks

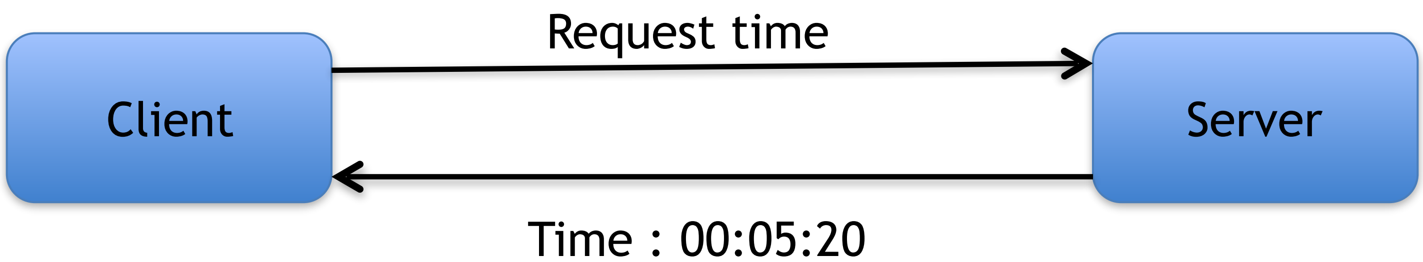
- Messages always arrive, with propagation delay exactly *d*



- Sender sends time *T* in a message
- Receiver sets clock to *T+d*
  - Synchronization is exact

How clocks synchronise

- Obtain time from time server:



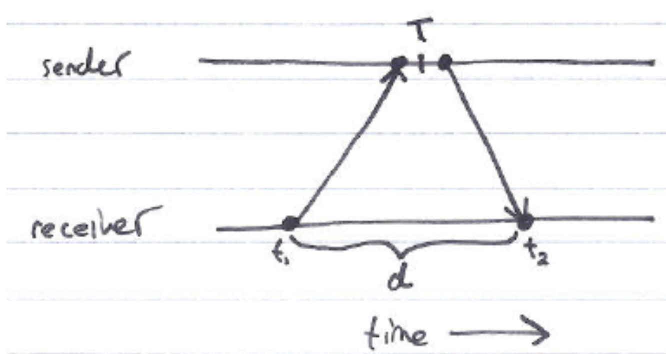
- Time is inaccurate
  – Delays in message transmission
  – Delays due to processing time
  – Server's time may be inaccurate

Synchronization in the real world

- Real networks are asynchronous
  - Propagation delays are arbitrary
- Real networks are unreliable
  - Messages don't always arrive

Cristian's algorithm

- Request time, get reply
  - Measure actual round-trip time d



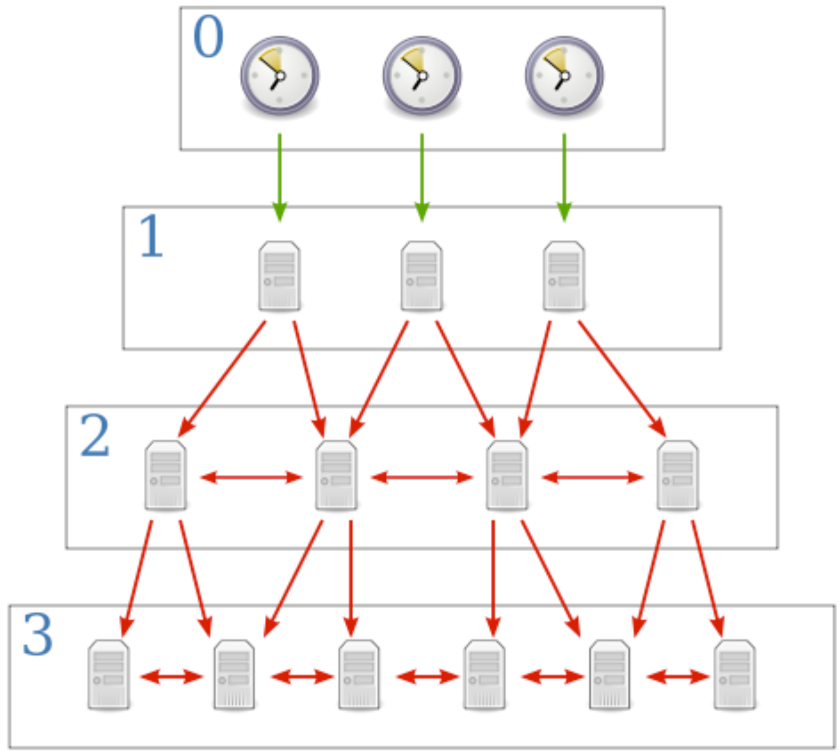
- Sender's time was T between t_1 and t_2
- Receiver sets time to $T + d/2$
  - Synchronization error is at most $d/2$
- Can retry until we get a relatively small d

The Berkeley algorithm

- Master uses Cristian's algorithm to get time from many clients
  - Computes average time
  - Can discard outliers
- Sends time adjustments back to all clients

The Network Time Protocol (NTP)

- Uses a hierarchy of time servers
  - Class 1 servers have highly-accurate clocks
    - connected directly to atomic clocks, etc.
  - Class 2 servers get time from only Class 1 and Class 2 servers
  - Class 3 servers get time from any server
- Synchronization similar to Cristian's alg.
  - Modified to use multiple one-way messages instead of immediate round-trip
- Accuracy: Local ~1ms, Global ~10ms



Source: http://upload.wikimedia.org/wikipedia/commons/c/c9/Network_Time_Protocol_servers_and_clients.svg

Real synchronization is imperfect

- Clocks never exactly synchronized
- Often inadequate for distributed systems
  - might need totally-ordered events
  - might need millionth-of-a-second precision

Logical clocks

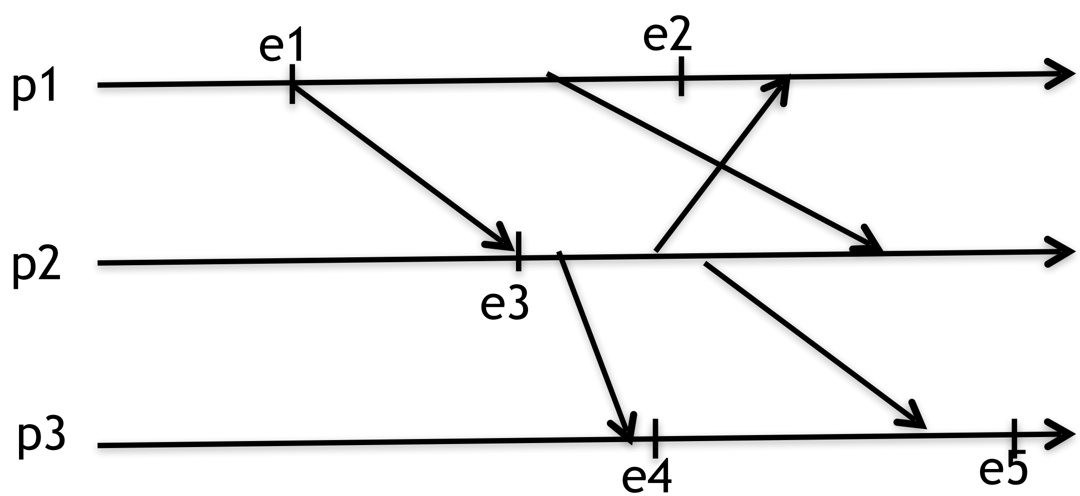
- Why do we need clocks?
  - To determine when one thing happened before another
- Can we determine that without using a "clock" at all?
  - Then we don't need to worry about synchronisation, millisecond errors etc..

Happened before

- $a \longrightarrow b$: a happened before b
  - If a and b are successive events in same process then $a \longrightarrow b$
  - Send before receive
    - If a : “send” event of message m
    - And b : “receive” event of message m
    - Then $a \longrightarrow b$
  - Transitive: $a \longrightarrow b$ and $b \longrightarrow c \Longrightarrow a \longrightarrow c$

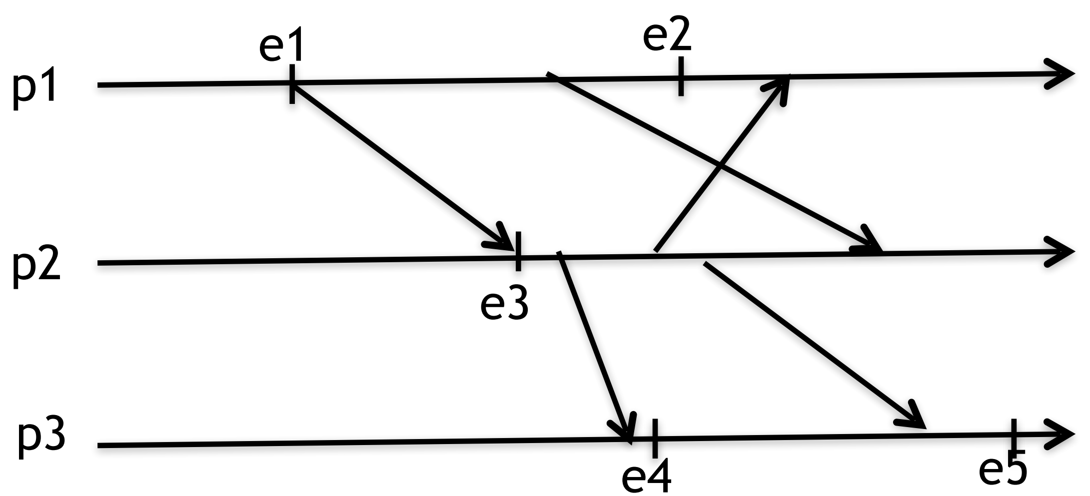
Example

- Events without a happened before relation are "concurrent"
- e1$\longrightarrow$e2, e3$\longrightarrow$e4,e1$\longrightarrow$e5, e5||e2



Example

- Events without a happened before relation are "concurrent"
- Happened before is a partial ordering



Happened before & causal order

- Happened before ==
  could have caused/influenced
- Preserves causal relations
- Implies a partial order
  - Implies time ordering between certain pairs of events
  - Does not imply anything about ordering between concurrent events

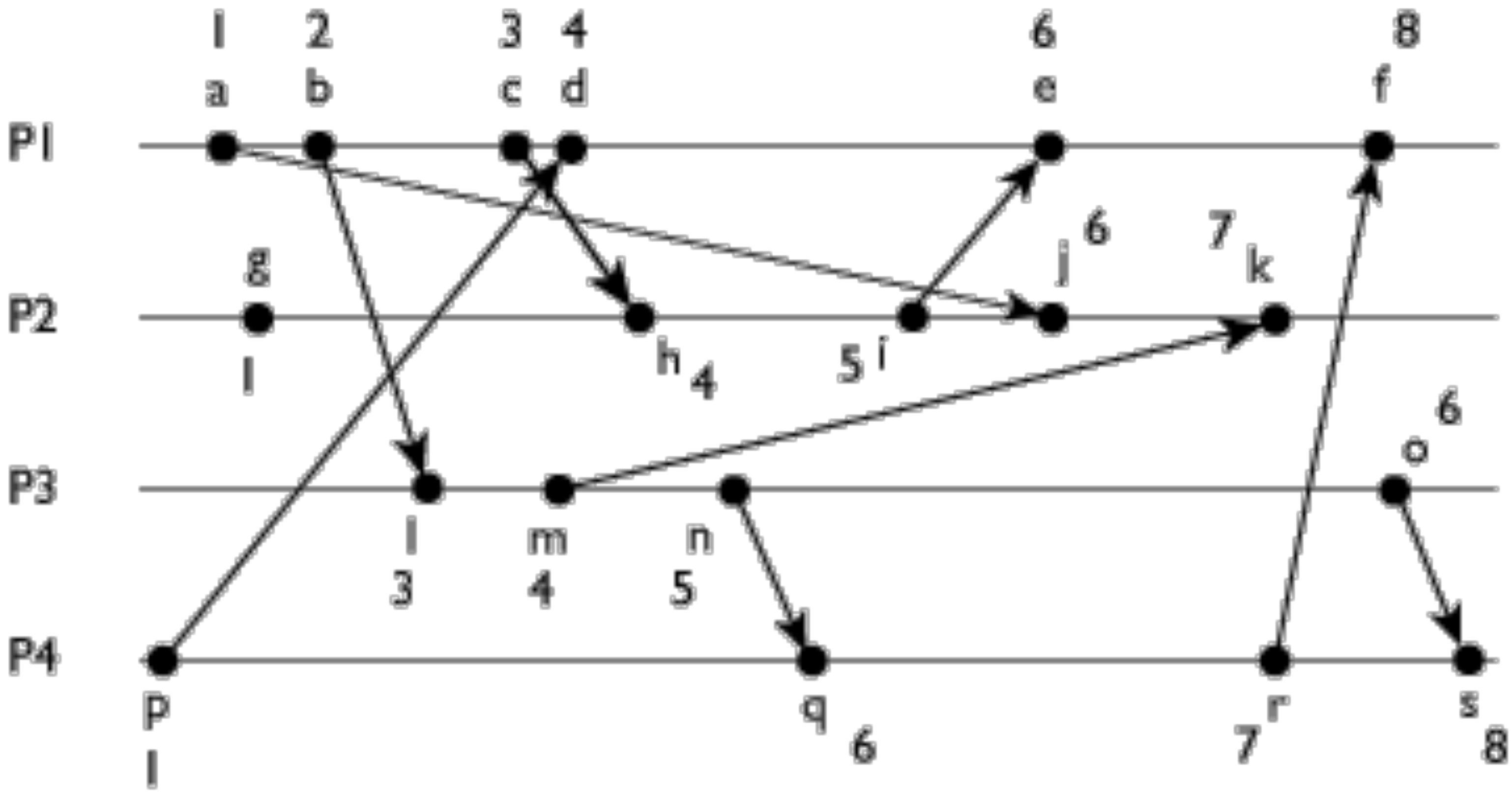
Logical clocks

- Idea: Use a counter at each process
- Increment after each event
- Can also increment when there are no events
  - Eg. A clock
- An actual clock can be thought of as such an event counter
- It counts the states of the process
- Each event has an associated time: The count of the state when the event happened

Lamport clocks

- Keep a logical clock (counter)
- Send it with every message
- On receiving a message, set own clock to max({own counter, message counter}) + 1
- For any event e, write c(e) for the logical time
- Property:
  - If $a \longrightarrow b$, then $c(a) < c(b)$
  - If $a \,||\, b$, then no guarantees

Lamport clocks: Example



Concurrency and Lamport clocks

- If e1$\longrightarrow$e2
  - Then no Lamport clock C exists with C(e1)== C(e2)

Concurrency and Lamport clocks

- If e1⟶e2
  - Then no Lamport clock C exists with C(e1)== C(e2)

- If e1||e2, then there exists a Lamport clock C such that C(e1)== C(e2)

The Purpose of Lamport Clocks

- If $a \longrightarrow b$, then $c(a) < c(b)$
- If we order all events by their Lamport clock times
  - We get a partial order, since some events have same time
  - The partial order satisfies “causal relations”

The purpose of Lamport clocks

- Suppose there are events in different machines
  - Transactions, money in/out, file read, write, copy
- An ordering of events that guarantees preserving causality

Vector Clocks

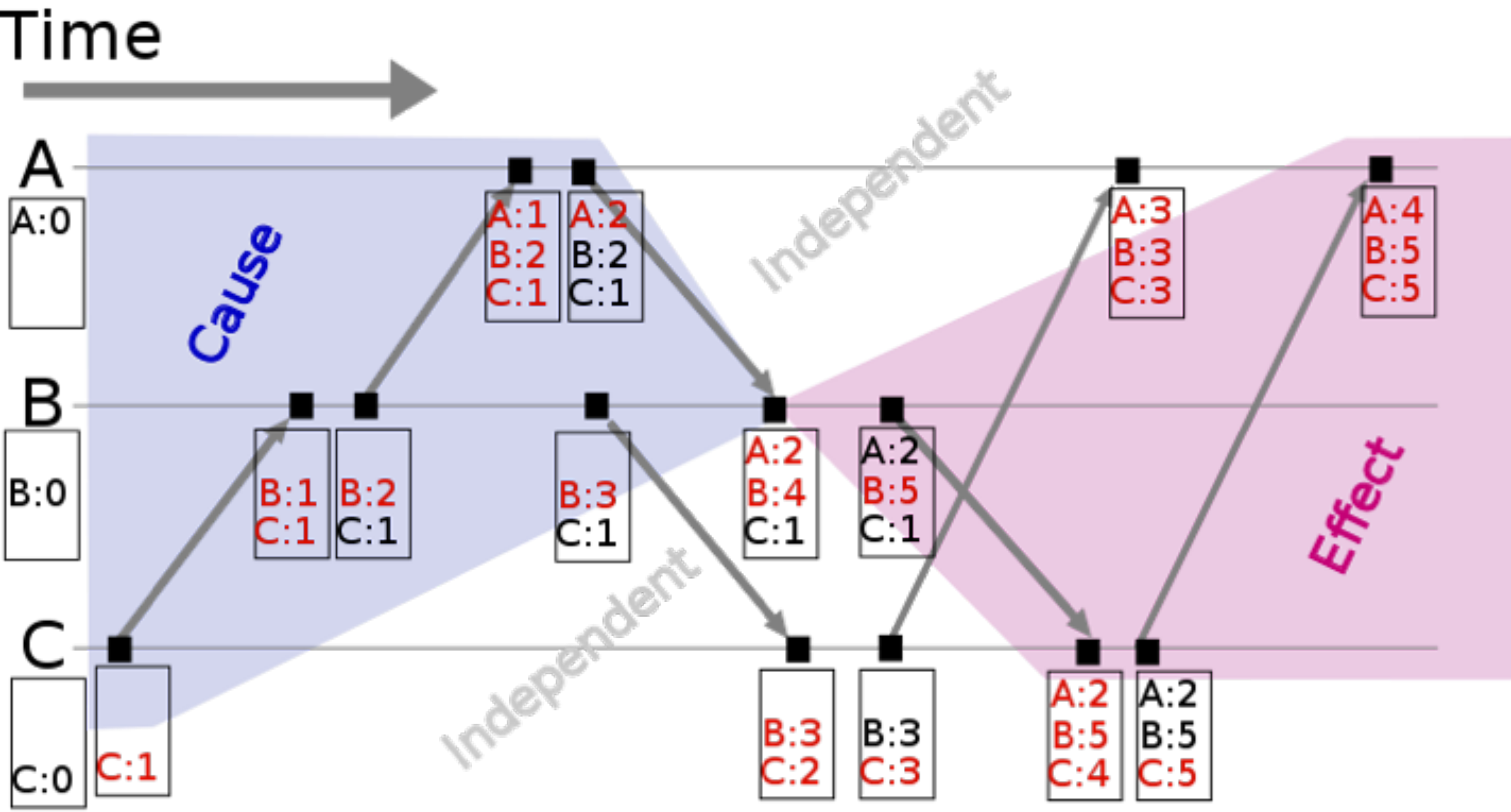
- We want a clock such that:
  - If $a \longrightarrow b$, then $c(a) < c(b)$
  - AND
  - If $c(a) < c(b)$, then $a \longrightarrow b$

  - Ref: Coulouris et al., V. Garg

Vector Clocks

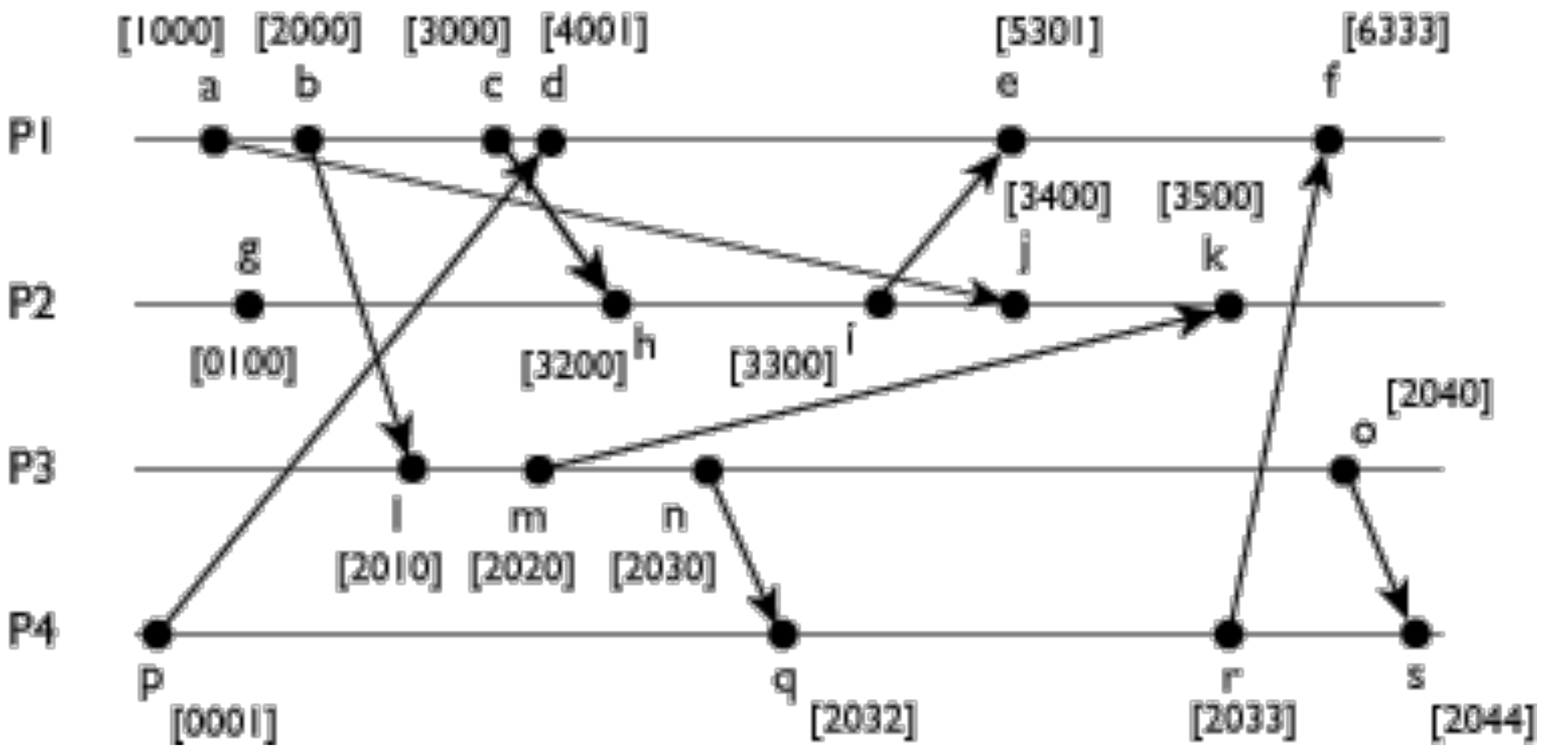
- Each process i maintains a vector V_i
- V_i has n elements
  - keeps clock $V_i[j]$ for every other process j
  - On every local event: $V_i[i] = V_i[i]+1$
  - On sending a message, i sends entire V_i
  - On receiving a message at process j:
    - Takes max element by element
    - $V_j[k] = \max(V_j[k], V_i[k])$, for $k = 1,2,\ldots,n$
    - And adds 1 to $V_j[j]$

Example



Time
A
A:0
A:1
B:2
C:1
A:2
B:2
C:1
A:3
B:3
C:3
A:4
B:5
C:5
Cause
Independent
B
B:0
B:1
C:1
B:2
C:1
B:3
C:1
A:2
B:4
C:1
A:2
B:5
C:1
Effect
C
C:0
C:1
Independent
B:3
C:2
B:3
C:3
A:2
B:5
C:4
A:2
B:5
C:5

Another Example



Comparing Timestamps

- V = V’ iff V[i] == V’[i] for i=1,2,…,n
- V < V’ iff V[i] < V’[i] for i=1,2,…,n

Comparing Timestamps

- V = V' iff V[i] == V'[i] for i=1,2,…,n
- V < V' iff V[i] < V'[i] for i=1,2,…,n

- For events a, b and vector clock V
  - a⟶b iff V(a) < V(b)

- Is this a total order?

Comparing Timestamps

- V = V’ iff V[i] == V’[i] for i=1,2,…,n
- V ≤ V’ iff V[i] ≤ V’[i] for i=1,2,…,n

- For events a, b and vector clock V
  - a⟶b iff V(a) ≤ V(b)

- Two events are concurrent if
  - Neither V(a) ≤ V(b) nor V(b) ≤ V(a)

Vector Clock Examples

- $(1,2,1) \leq (3,2,1)$ but $(1,2,1) \nleq (3,1,2)$

- Also $(3,1,2) \nleq (1,2,1)$
- No ordering exists

Vector Clocks

- What are the drawbacks?
- What is the communication complexity?

Vector Clocks

- What are the drawbacks?
  - Entire vector is sent with message
  - All vector elements (n) have to be checked on every message
- What is the communication complexity?
  - Ω(n) *per message*
  - Increases with time

Logical Clocks

- There is no way to have perfect knowledge on ordering of events
  - A “true” ordering may not exist..
  - Logical and vector clocks give us a way to have ordering consistent with causality

Logical time

- Capture just the "happens before" relationship between events
  - Discard the infinitesimal granularity of time
  - Corresponds roughly to causality
- Time at each process is well-defined
  - Definition ($\rightarrow_i$): We say $e \rightarrow_i e'$ if e happens before e' at process i

Important Points

- Physical Clocks
  - Can keep closely synchronized, but never perfect
- Logical Clocks
  - Encode causality relationship
  - Lamport clocks provide only one-way encoding
  - Vector clocks provide exact causality information