CS5223
Distributed Systems

Lecture 5: Synchronization (Time and Clock)

Instructor: YU Haifeng

Today's Roadmap

- Chapter 5 of textbook
- Physical clock synchronization
- Software clocks and applications
- Mutual exclusion
- Leader election
- Some contents (e.g., logical clocks and happened-before relation) overlap with CS4231
  - These concepts are critical in distributed system
  - So we need to cover these

Motivation for Physical Clocks

- Example: Timestamp files
  - Creation time, modification time, etc.
- Example: cron jobs
  - Virus scan
  - Software update
- Advantage of physical clock reading: Portable and context-free
  - Reading in one system will make sense even in other context

Motivation for Physical Clock Synchronization

- Electronic devices (including computers) usually advances clocks based on oscillations from some crystal

- Clock rate drift: Caused by slight differences in oscillation frequency
  - E.g., 1 minute my time = 2 minute your time

- Clock drift: Caused by clock rate drift and also lack of synchronization
  - E.g., your clock is 10 minute ahead of the correct time

- Physical clock synchronization usually only deal with clock drift

Clock Synchronization Can be Confusing!

- Human beings are usually used to talk about a single clock
  - Confusion usually arises when there are multiple clocks each may advance at its own pace
  - Such confusion already arises in some places in the textbook…

- Need a crystal-clear formal model
  - T: Global accurate time (starts from time 0)
  - For node A:

$$Ta = T * (1+Xa) + Ya$$

clock reading on A

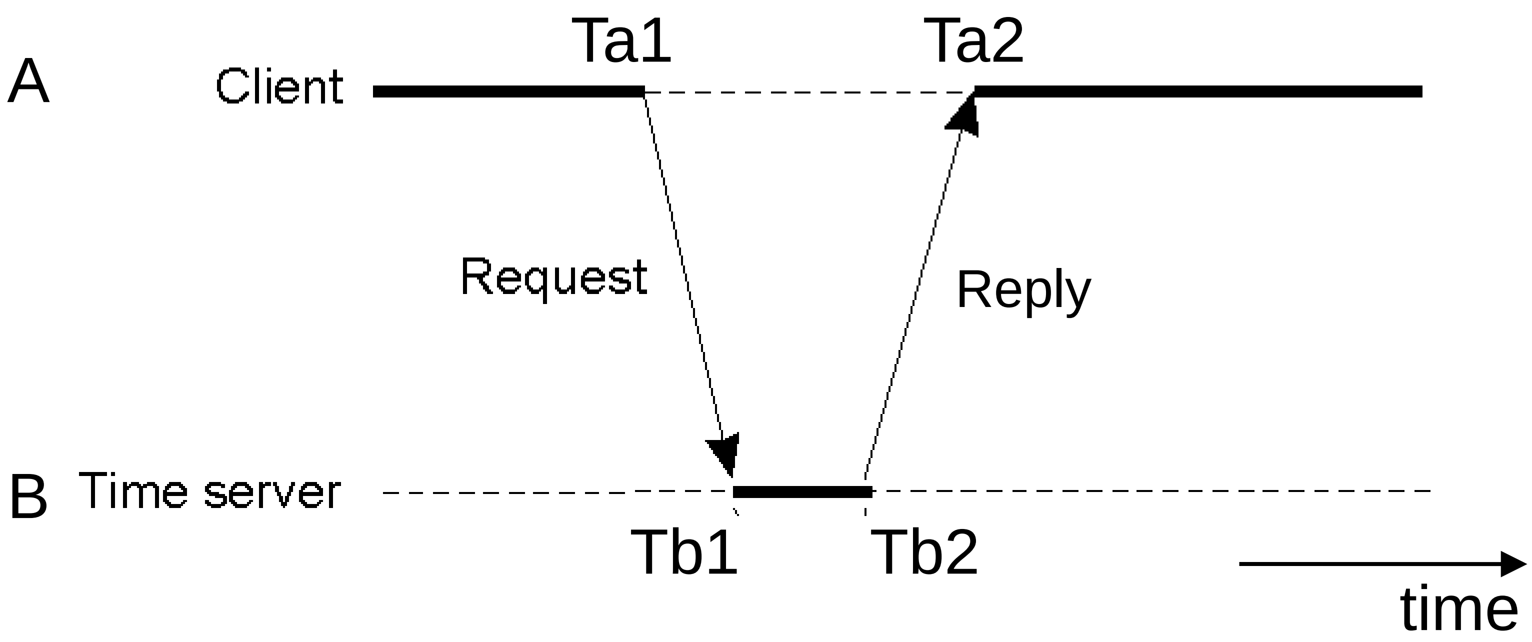
clock rate drift on A

initial clock drift on A

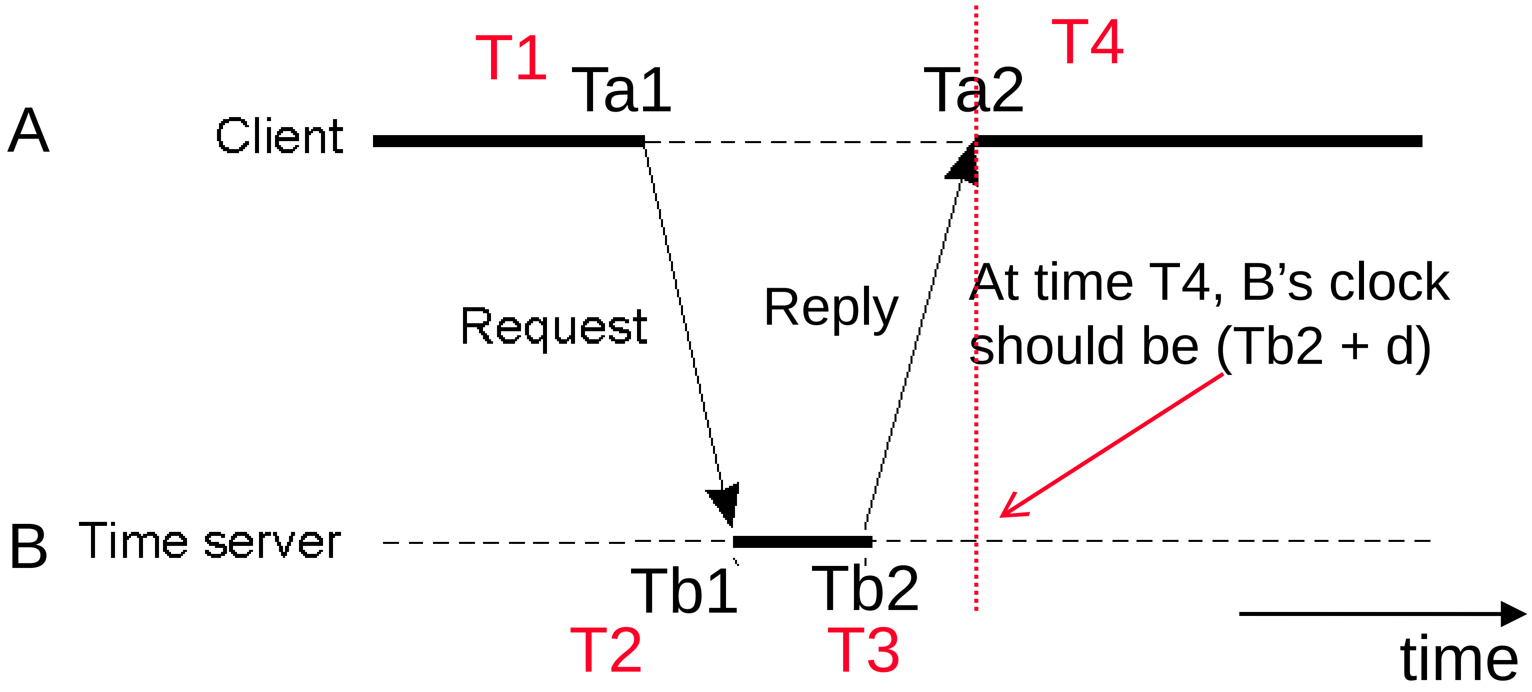
Xa may change from time to time

Cristian's Algorithm

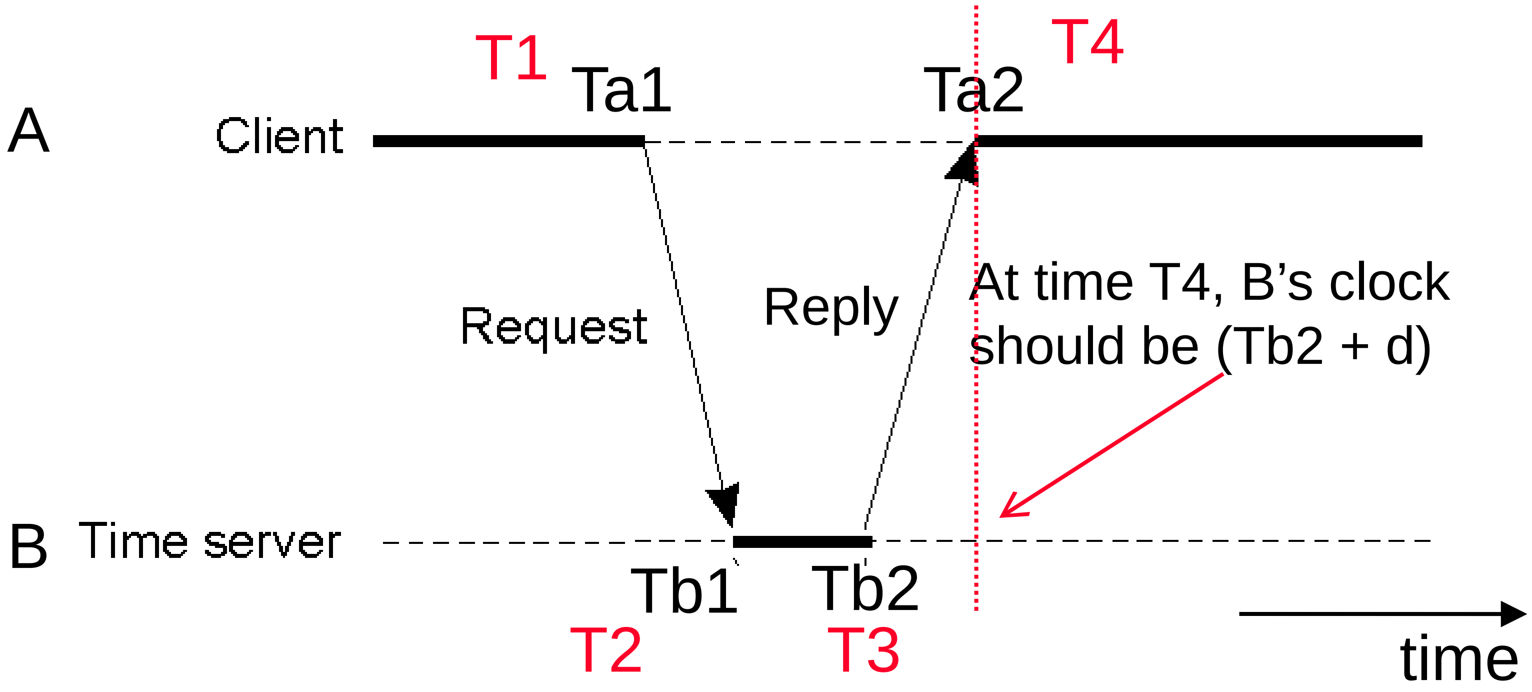
- Assumptions
  - Propagation delay for request and reply is the same
  - Clock rate drift is 0



- T1, T2, T3, T4 are accurate times
- Ta2 – Ta1 = T4 – T1 (since clock rate drift is 0)
- Tb2 – Tb1 = T3 – T2
- Propagation delay d = ((T4 - T1) – (T3 - T2)) / 2



- Key idea: (T4 – T1) is accurate even if A has clock drift
- What is A's clock rate drift is not 0?
  - Consider a scenario where d = 0, Tb2 - Tb1 = 1ms, Ta2 – Ta1 = 2 minutes
- Thus this implicit assumption (not mentioned in textbook) is critical !



Network Time Protocol (NTP)

- Inaccuracy in Cristian's protocol comes from
  - Non-identical propagation delay for request and reply (quite common in wide-area network)
  - Clock rate drift not being 0 -- impact is usually small (why?)

- Total propagation delay is an upper bound on the difference in propagation delays for request and response
  - $d1+d2 = d$ implies $|d1 – d2| <= d$

- Synchronize with multiple servers and pick the one with smallest d

NTP Continued

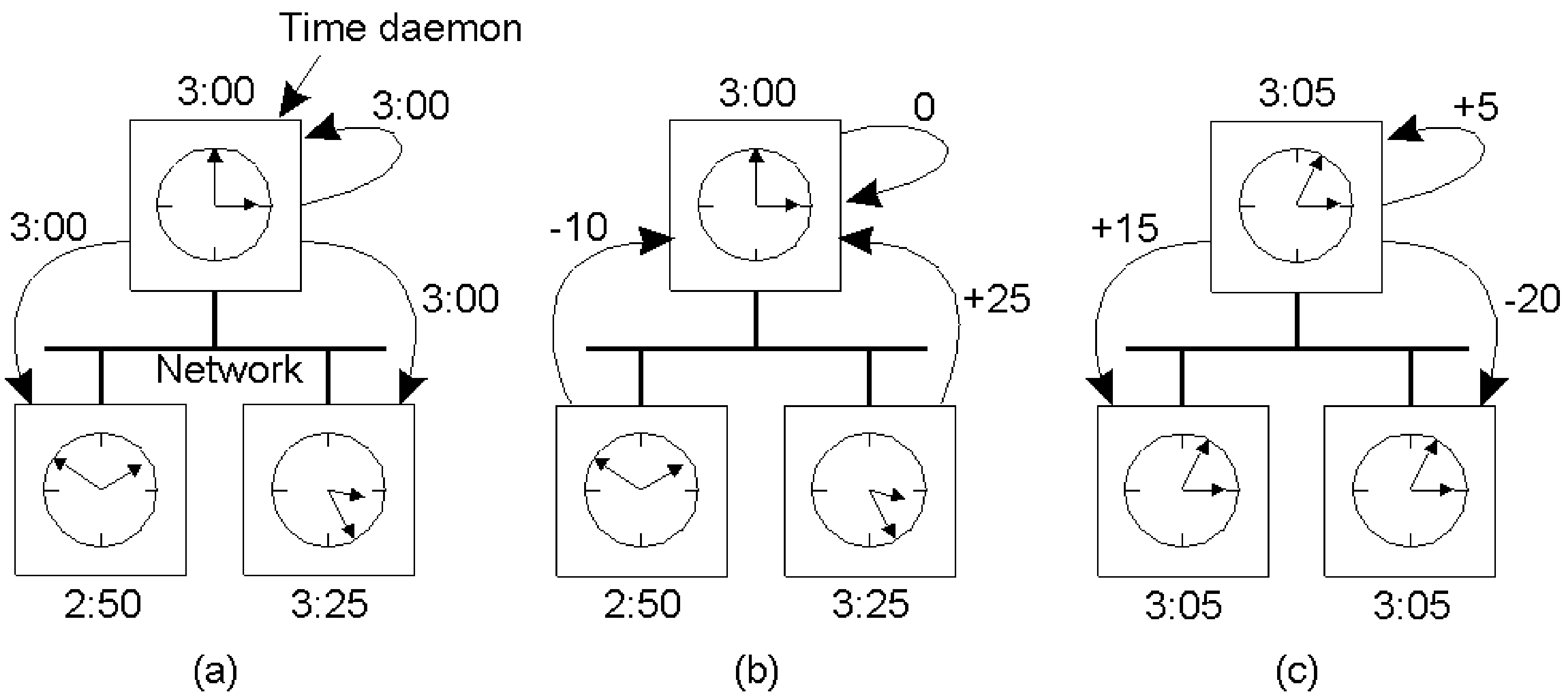
- Does not make sense to synchronize with a inaccurate clock
- Stratum-1 servers:
  - Servers with UTC receiver or atimic clocks
- Machines synchronize with Startum-1 servers become stratum-2
  - And so on…
- Current worldwide NTP accuracy
  - 1 to 50 milliseconds

The Berkeley Algorithm

- Assumptions
  - Network delay = 0
  - Usually for cases where no machine has the “accurate time”

- Idea: Hope that the clock drifts on different machines will cancel out

The Berkeley Algorithm: Example



The Berkeley Algorithm

- Quite straight-forward

- If you have a machine with “accurate time”
  - Should not take average
  - Just broadcast the time periodically

Problems with Physical Clocks

- Many protocols need to impose an ordering among events
  - If two players open the same treasure chest “almost” at the same time, who should get the treasure?
- Physical clocks:
  - Seems to completely solve the problem
  - But what about theory of relativity?
  - Even without theory of relativity – efficiency problems
- How accurate is sufficient?
  - Without out-of-band communication: Minimum message propagation delay
  - With out-of-band communication: distance/speed of light
  - In other words, some time it has to be “quite” accurate

Software “Clocks”

- Software “clocks” can incur much lower overhead than maintaining (sufficiently accurate) physical clocks
  - But does not allow comparison with external clocks
- Allows a protocol to infer ordering among events
- Goal of software “clocks”: Capture event ordering that are visible to users
  - Users do not have accurate physical clocks either
  - But what orderings are visible to users without physical clocks?

Assumptions

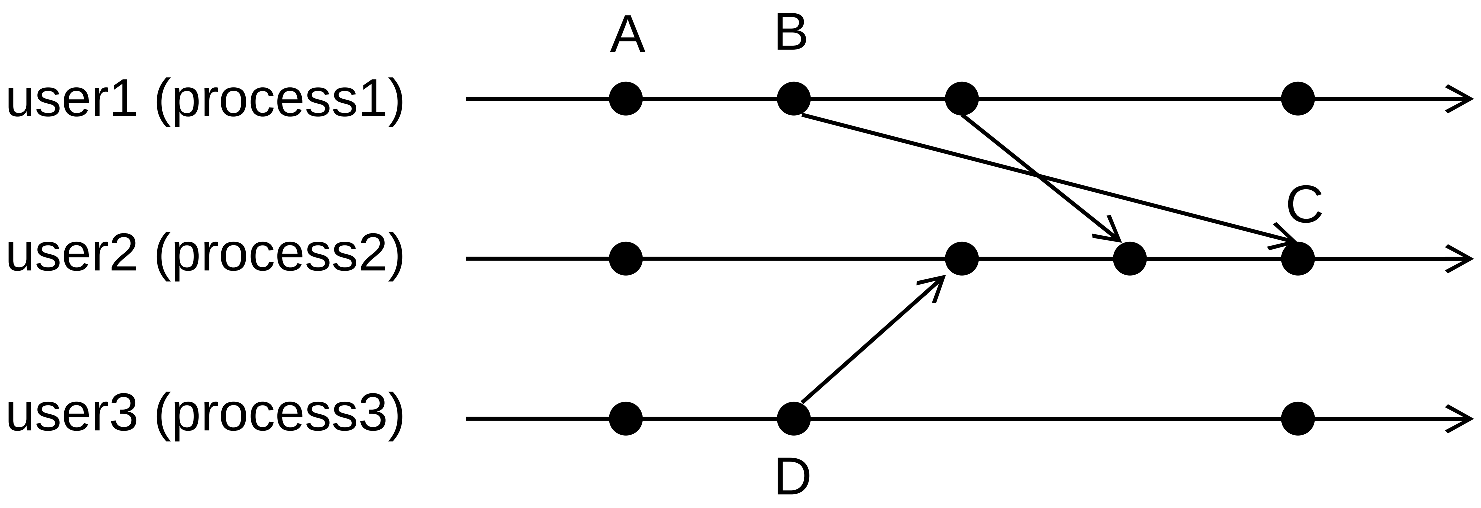
- Process can perform three kinds of atomic actions/events
  - Local computation
  - Send a single message to a single process
  - Receive a single message from a single process
  - No atomic broadcast
- Communication model
  - Point-to-point
  - Error-free, infinite buffer
  - Potentially out of order

Visible Ordering to Users

- A < B (process order)
- B < C (send-receive order)
- A < C (transitivity)

- A ? D
- B ? D

PollEv.com/haifengyu229

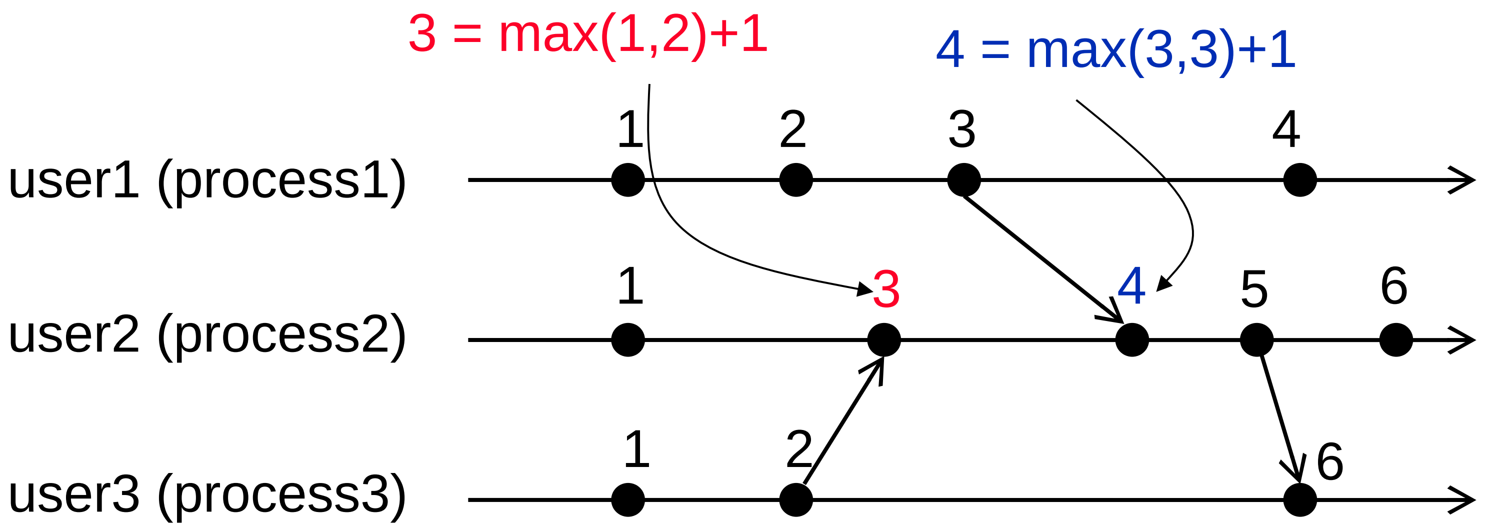


“Happened-Before” Relation

- “Happened-before” relation captures the ordering that is visible to users when there is no physical clock
  - A partial order among events
  - Process-order, send-receive order, transitivity
- First introduced by Lamport – Considered to be the first fundamental result in distributed computing
- Goal of software “clock” is to capture the above “happened-before” relation

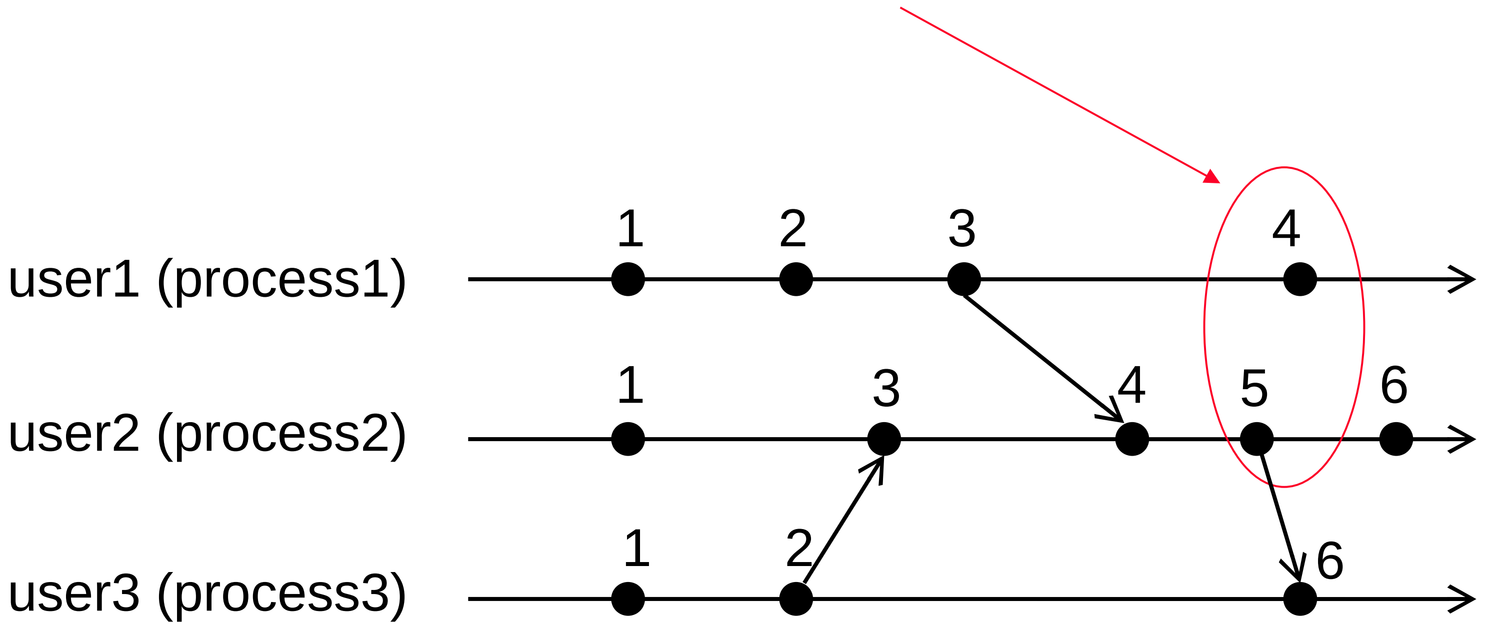
Software “Clock” 1: Logical Clocks

- Each event has a single integer as its logical clock value
  - Each process has a local counter C
  - Increment C at each “local computation” and “send” event
  - When sending a message, logical clock value V is attached to the message. At each “receive” event, C = max(C, V) + 1



Logical Clock Properties

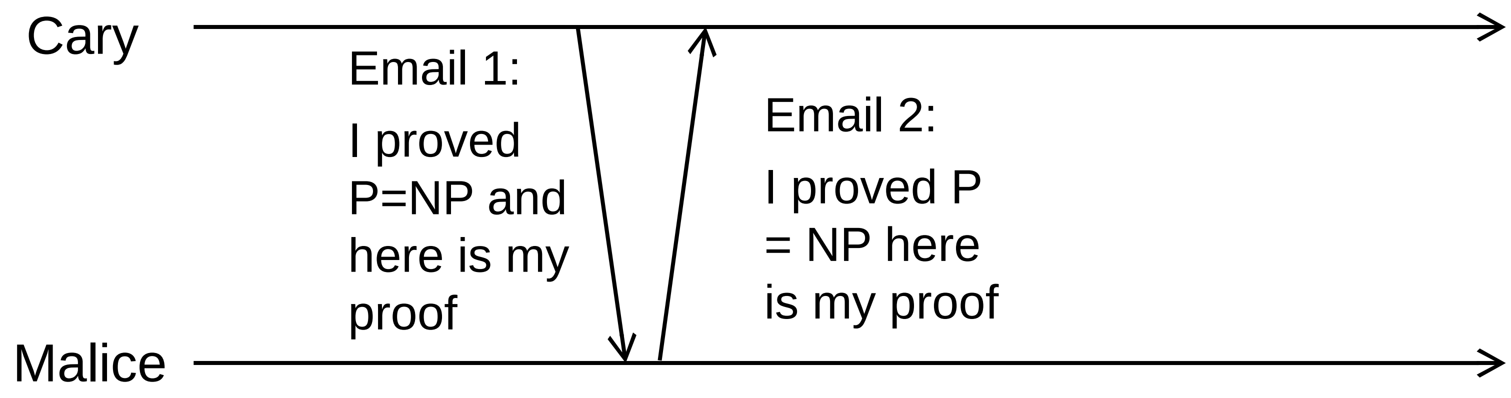
- Theorem:
  - Event s happens before t => the logical clock value of s is smaller than the logical clock value of t.
- The reverse may not be true



Example Application for Logical Clock

- Total-ordered broadcast protocol is the “standard” example application for logical clock (as in textbook)
  - But it is complex and will take half a lecture to go through
  - (The protocol is covered in CS4231…)

- We will use a much simpler application example



- We want to decide who to give Turing award to
  - Both Cary and Malice try to claim credit
  - Malice says Cary copies her proof
  - Cary says Malice copies her proof

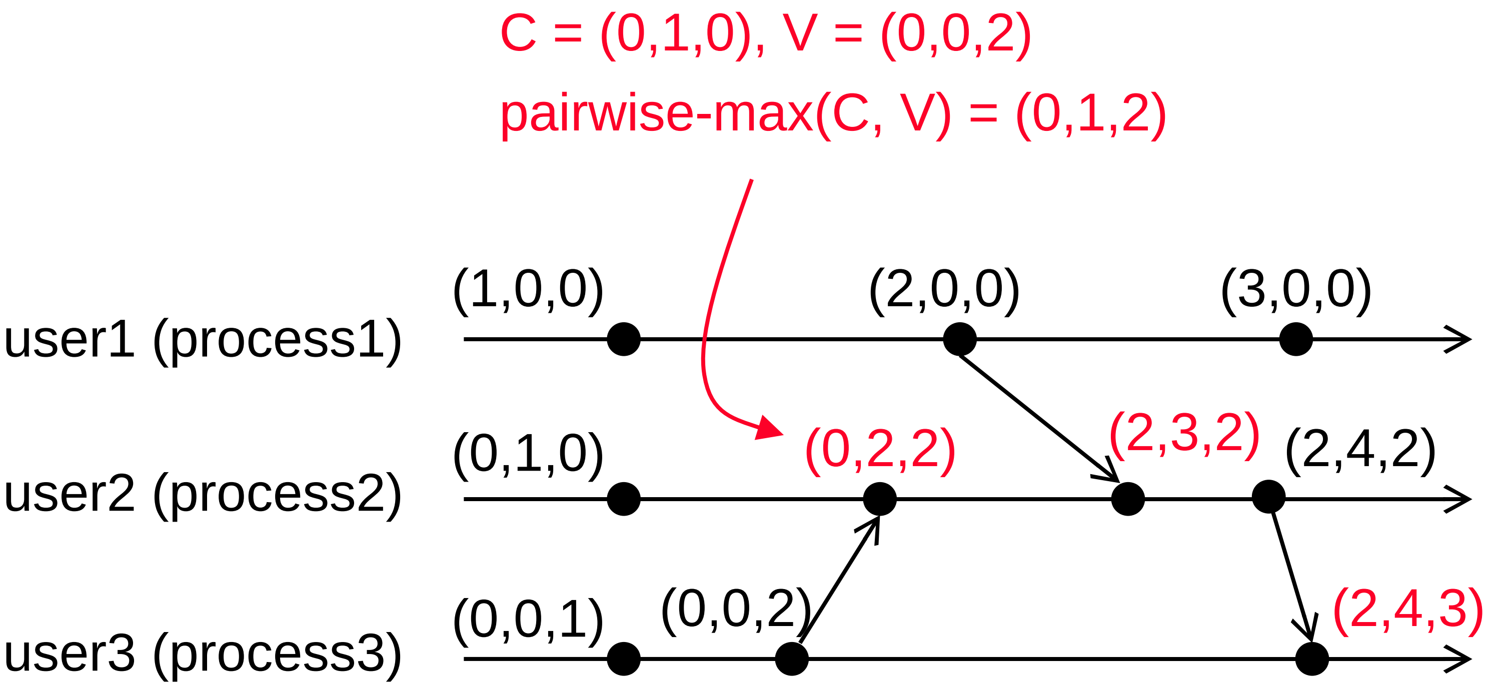
Software “Clock” 2: Vector Clocks

- Logical clock:
  - Event s happens before event t the logical clock value of s is smaller than the logical clock value of t.
- Vector clock:
  - Event s happens before event t the vector clock value of s is “smaller” than the vector clock value of t.
- Each event has a vector of *n* integers as its vector clock value
  - v1 = v2 if all n fields same
  - v1 <= v2 if every field in v1 is less than or equal to the corresponding field in v2
  - v1 < v2 if v1 <= v2 and v1 ≠ v2

Relation “<“ here is not a total order

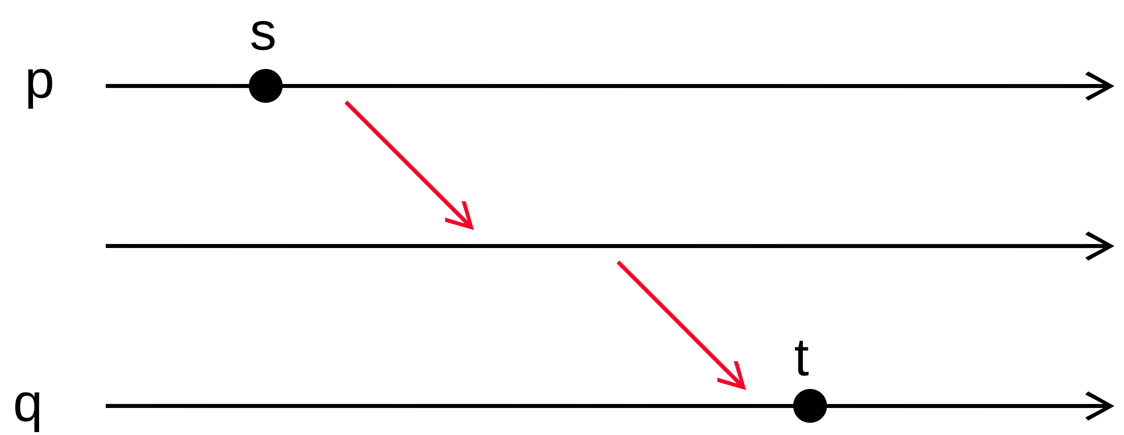
Vector Clock Protocol

- Each process i has a local vector C
- Increment C[i] at each “local computation” and “send” event
- When sending a message, vector clock value V is attached to the message. At each “receive” event, C = pairwise-max(C, V); C[i]++;



Vector Clock Properties

- Event s happens before t => vector clock value of s < vector clock value of t: There must be a chain from s to t
- Event s happens before t => vector clock value of s < vector clock value of t
  - If s and t on same process, done
  - If s is on p and t is on q, let VS be s's vector clock and VT be t's
  - VS < VT => VS[p] ≤ VT[p] => Must be a sequence of message from p to q after s and before t

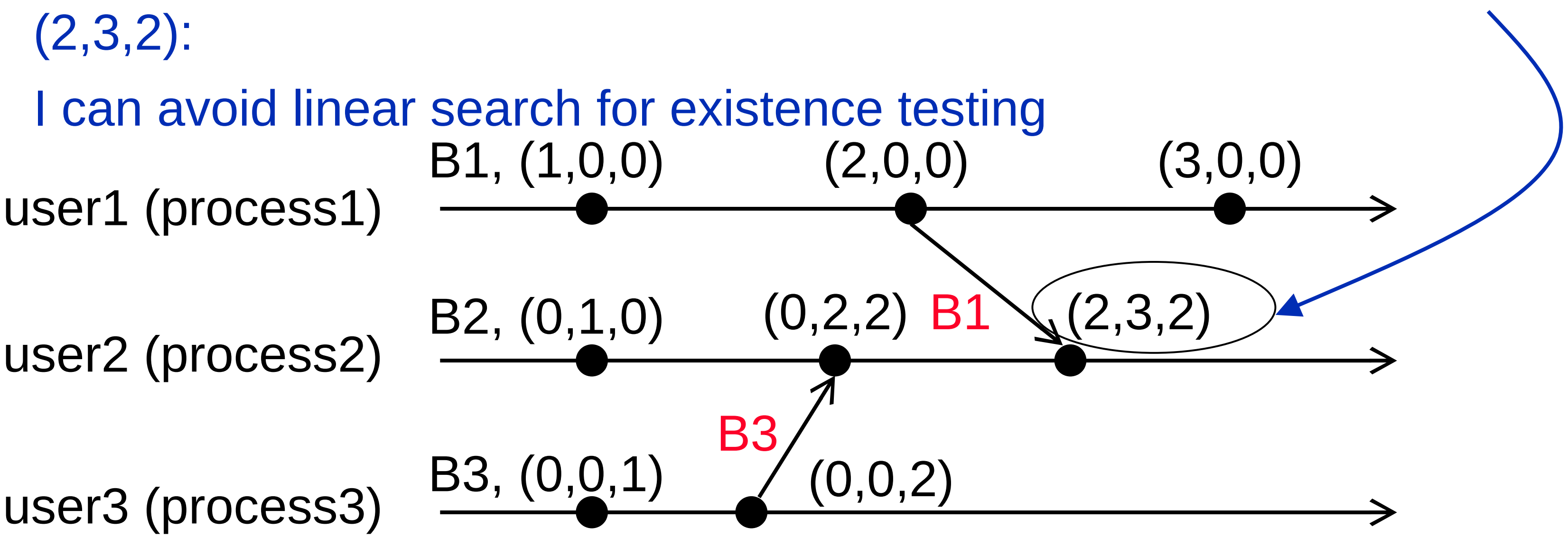


Example Application of Vector Clock

- Each Bitcoin node has some blocks (assume that Bitcoin runs on a fixed set of nodes)
  - Want all nodes to know all blocks
- Each block has a vector clock value

I have seen all blocks whose vector clock is smaller than (2,3,2):

I can avoid linear search for existence testing



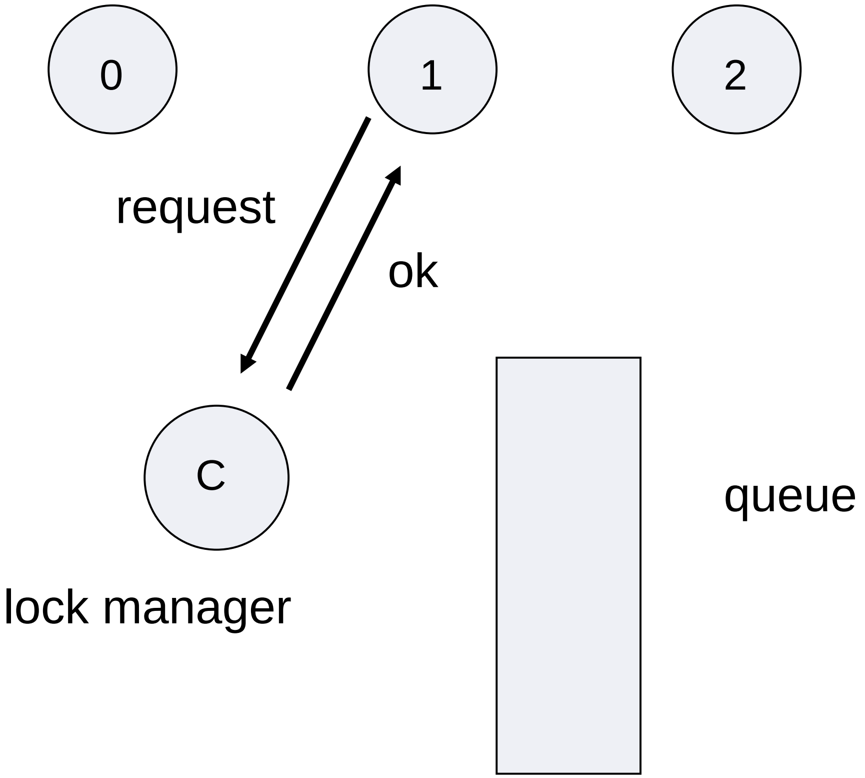
Mutual Exclusion: Token-based Approach

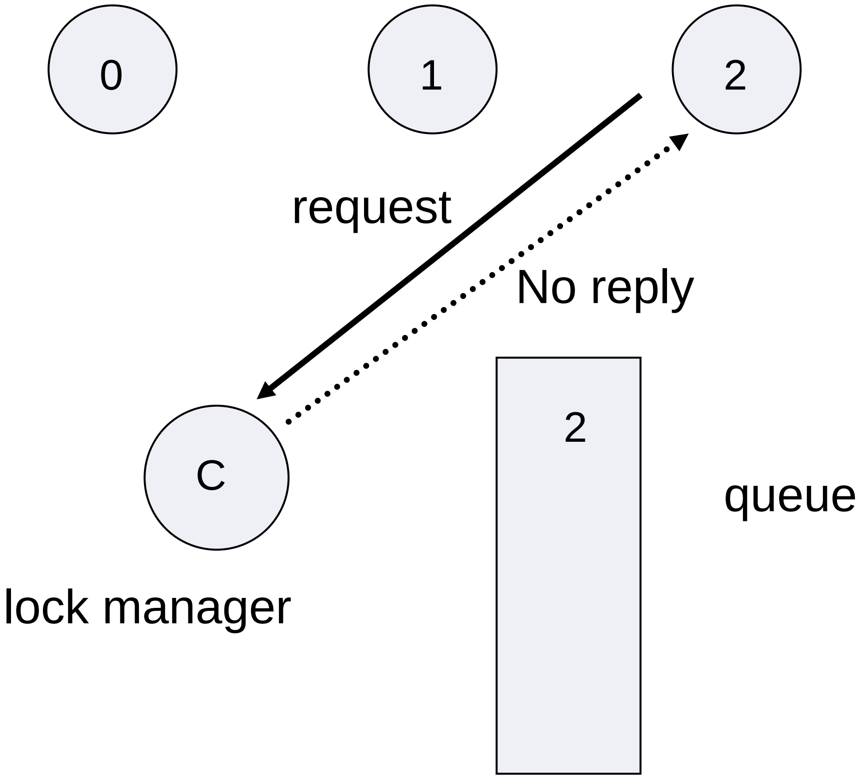
- Motivation is the same as in non-distributed systems

- Token-based approached
  - A single token system-wide
  - Whoever holds the token has the privilege

- Typical software architecture
  - A special node acts as “lock manager” or “token manager”

Process 1 requests





Process 2 requests lock while process 1 is holding the lock

Mutual Exclusion: Token-based Approach

- Also possible to implement it as a token ring

- Pros:
  - Simplicity – lock manager approach is widely used
  - Fairness easy to ensure
  - No starvation

- Cons:
  - Cannot deal with failures nicely

Mutual Exclusion: Token-based Approach

- Dealing with client failures
  - Use leases – the lock is granted for only a finite amount of time
  - Automatically revoked after time out
  - Problem: A client is writing to a large file but cannot renew the lease due to lost network connection
- Dealing with server (lock manager) failures
  - Need to start a new server
  - Need to ensure the number of token is exactly 1

Mutual Exclusion: Voting

- We want to deal with server crashes
- Have n servers instead of 1
  - Each server has a vote
  - A server may crash and recover
- Assume clients do not crash
  - We can always use lease to deal with client crashes
- To acquire a lock,a client needs to get (n/2+1) votes
  - Servers get the votes back after the client releases the clock
  - Called majority voting

Mutual Exclusion: Voting

- Theorem:
  - No two clients can hold the lock at the same time with majority voting (even if some server crashes)
- Can tolerate at most n/2-1 server failures
  - Usually quantified as availability
- Disadvantage:
  - Need n servers instead of one
  - Need to contact n/2+1 servers to get lock

Mutual Exclusion: Voting

- Majority voting is extremely flexible
  - How about assigning different number of votes to different servers?
  - A (5 votes), B(2 vote), C(2 vote), D(2 vote)
  - What is the impact on availability and performance?

- What is the best vote assignment mechanism?
  - Machines fail iid with probability of p
  - Theorem: Democracy is the best and monarchy is the worst when $p < 0.5$
  - Theorem: Democracy is the worst and monarchy is the best when $p > 0.5$

Mutual Exclusion: Quorum Systems

- A (5 votes), B(2 vote), C(2 vote), D(2 vote) =

  { {A, B}, {A, C}, {A, D},

  {A, B, C}, {A, B, D}, {A, C, D}, {B, C, D},

  {A, B, C, D} }

- A voting system can always be expressed as a quorum system
  - The reverse is not true
- But it happens that the best quorum systems corresponds to majority voting (under the earlier failure model)

History Readings (Non-compulsory)

- “The reliability of voting mechanisms”, IEEE Transactions on Computers, pages 1197-1208, Oct 1987.

- “How to assign votes in a distributed system”, Journal of ACM, October 1985.

Leader Election: Motivation

- We often need a coordinator in distributed systems
  - Leader, distinguished node/process
- We will only discuss leader election algorithms for failure-free contexts
  - Fault-tolerant leader election algorithms are related to fault-tolerance and will be covered later in the course

Leader Election on General Graph (n known)

- Complete graph
  - Each node send its id to all other nodes
  - Wait until you receive n ids
  - Biggest id wins

- Any connected graph
  - Flood your id to all other nodes (how?)
  - Wait until you receive n ids
  - Biggest id wins

Leader Election on General Graph (n unknown)

- Complete graph
  - n must be known
- Any connected graph
  - An auxiliary protocol to calculate the number of nodes

- The protocol is initiated by any node who wants to know n
  - The protocol actually establishes a spanning tree starting from the initiator

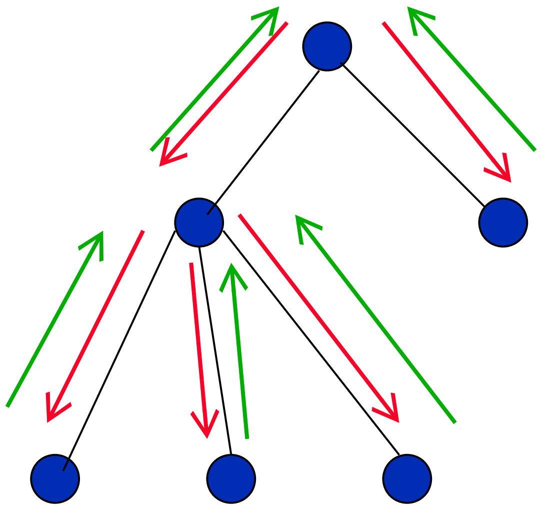
Spanning Tree Construction

- Remember: No centralized coordinator
  - Goal of the protocol: Each node knows its parent and children (i.e., a distributed tree)

- The root is initially marked
- Every marked node send “mark” msg to all its neighbors
- Upon receiving the first “mark” msg, an unmarked node becomes marked, and sends back “I am your child” msg
- For later “mark” msgs, send back “I am not your child” msg
- Tree construction completed if every node has received all replies from all its neighbours

Counting Nodes Using a Spanning Tree

- Disseminate “start count” msg down the tree
- Count will be aggregated up the tree



Summary

- Physical clock synchronization
- Software clocks and applications
- Mutual exclusion
- Leader election