Distributed Computing

A-05. Google Cloud Spanner

Google's Problem

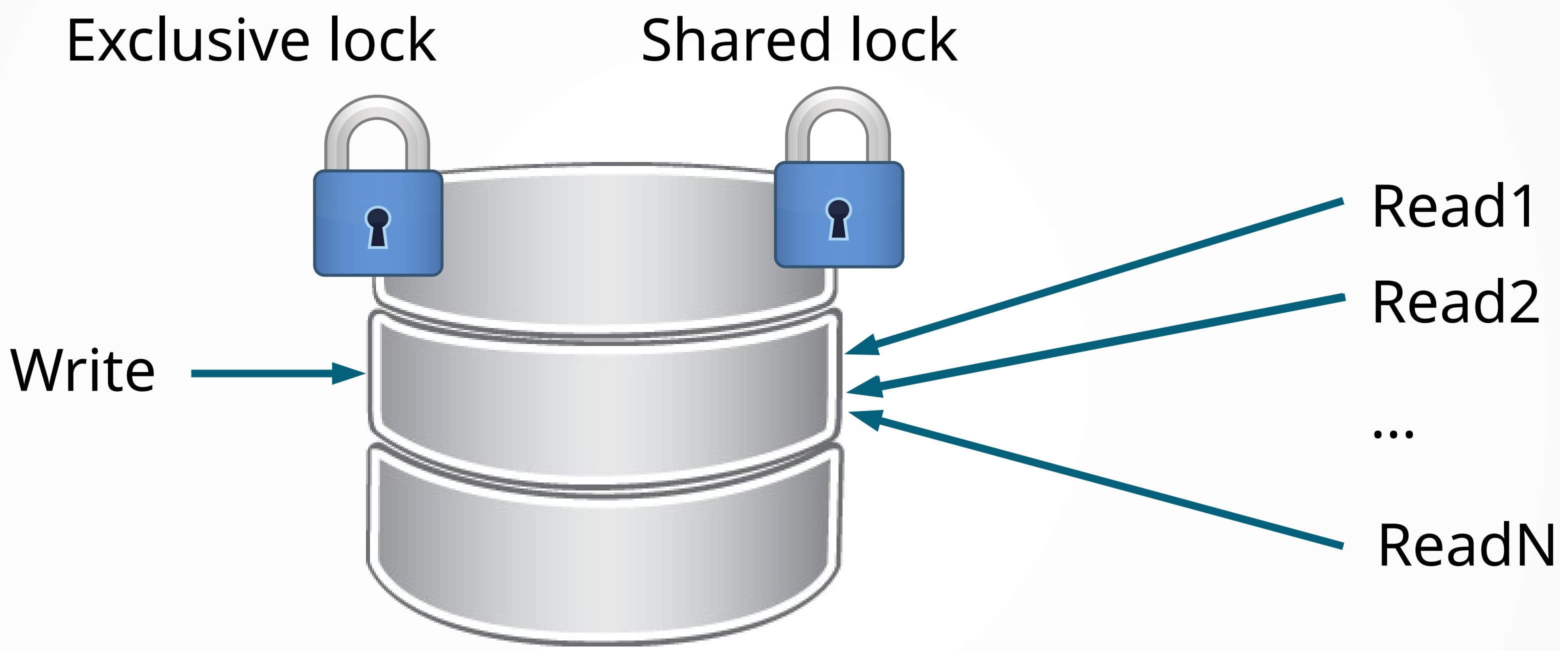
- Google engineers have **several datacenters** distributed across the world
  - **Sharded**, **replicated** database
- They wanted **linearization** (external consistency)—i.e., transactions are processed **in the same order they are seen in the real world**
- Example use case:
  - I remove an untrusted person X from my friend list
  - I post "my government is repressive"

Google's Problem (2)

- If we could build a whole log of all transactions...
- But we can't **all** go through Paxos/Raft
  - Remember: each update is sent to all servers in a Paxos instance
- Solution: use Paxos locally and then **resort to clocks**
- Reference: paper, slides & video

Philosophy

Locks?



- One write at a time **in the whole system**
- When you're writing somewhere, **you can't read**
- **Unreasonable** for a huge system

Historical View

- Pieces of information are annotated with time
  - We know **when they got in the system**
  - If they were canceled, **we know when**
- To do this, we need a **super-precise clock!**
  - Typical Internet latencies can be up to **hundreds of milliseconds**
- As we'll see, Spanner will **wait** to handle uncertainties in the clock
  - It's key to **drive uncertainties down**

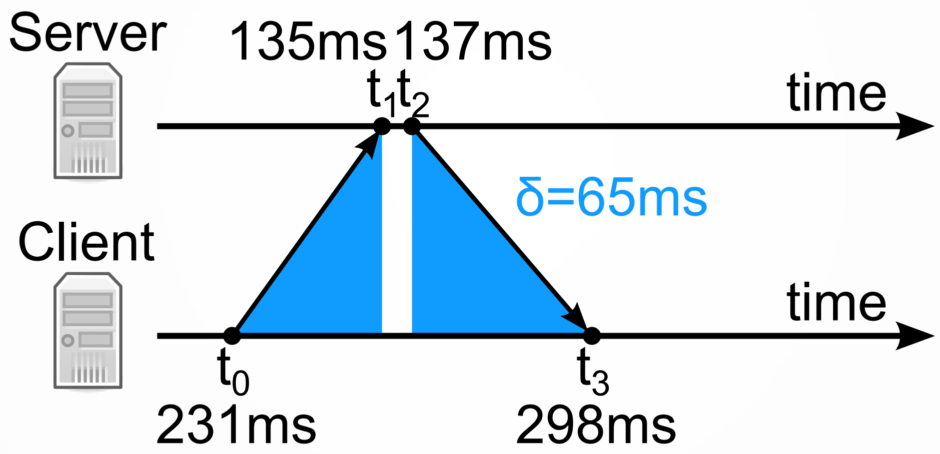
What’s the Time?

TrueTime

- A system to evaluate time with **high reliability**
- Idea here: **bounded uncertainty**
- `TrueTime.now()` returns an interval: *[earliest, latest]*
  - Meaning: time is **between earliest and latest**
- Each datacenter has a set of **time master** machines
  - Regular ones, equipped with GPS
  - **Armageddon masters**, with atomic clocks (!!!)

Talking to a Time Master (1)

- Not all details are given; assuming an approach similar to the **Network Time Protocol** (NTP):

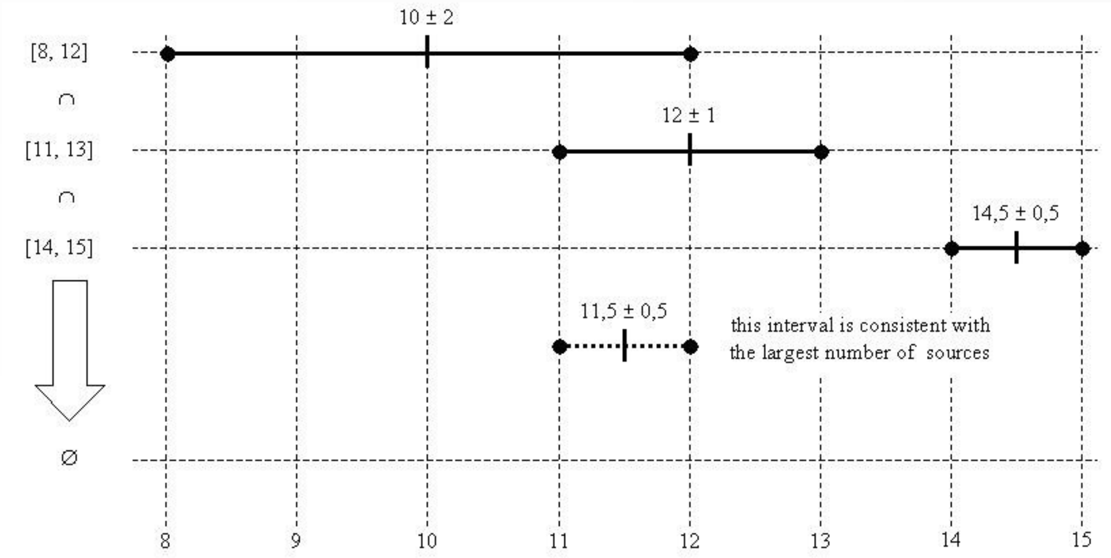


- Time offset $\theta = \dfrac{(t_1 - t_0) + (t_2 - t_3)}{2}$
- Round-trip delay $\delta = (t_3 - t_0) - (t_2 - t_1)$

Talking to a Time Master (2)

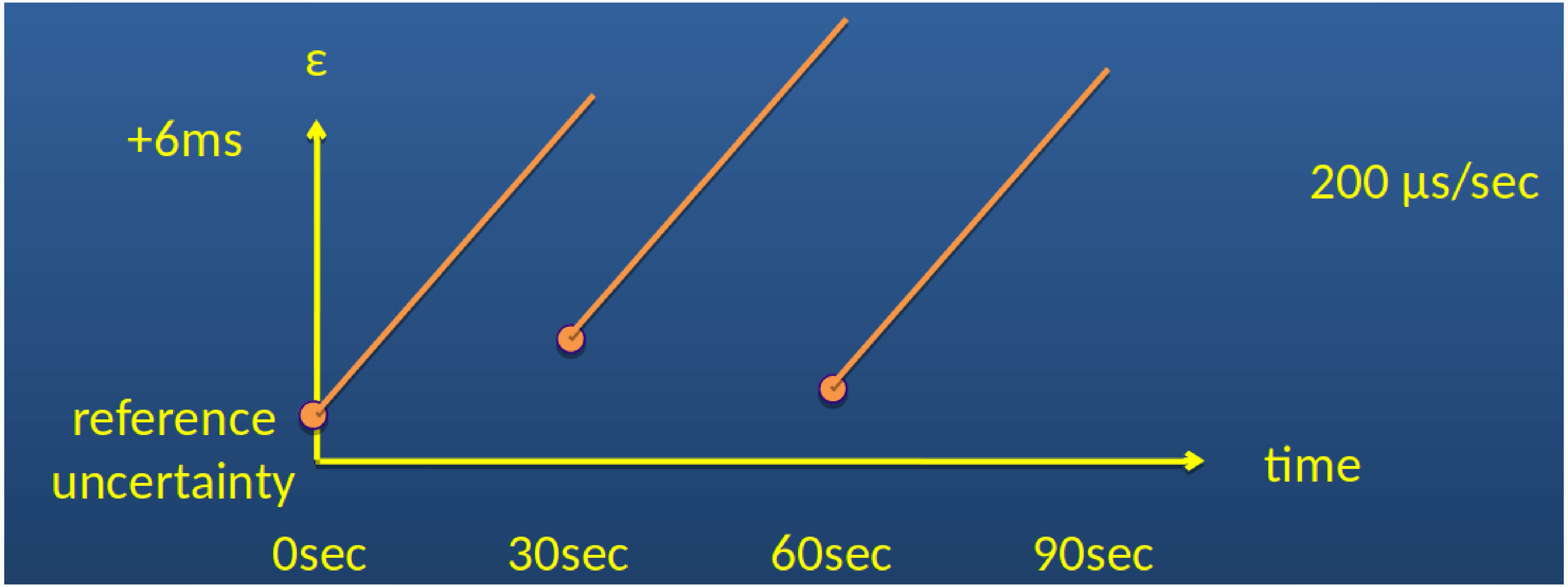
- Measurement as described before is repeated, and statistics are extracted
- The output is an interval $t \pm \varepsilon = [earliest, latest]$
- One of the time masters could be wrong: to weed out "liars", an algorithm is needed
- Problem: given n $[a_i, b_i]$ intervals, find the sub-interval that is consistent with most cases
  - (In case of a tie, return any of them)

Intersection Algorithm



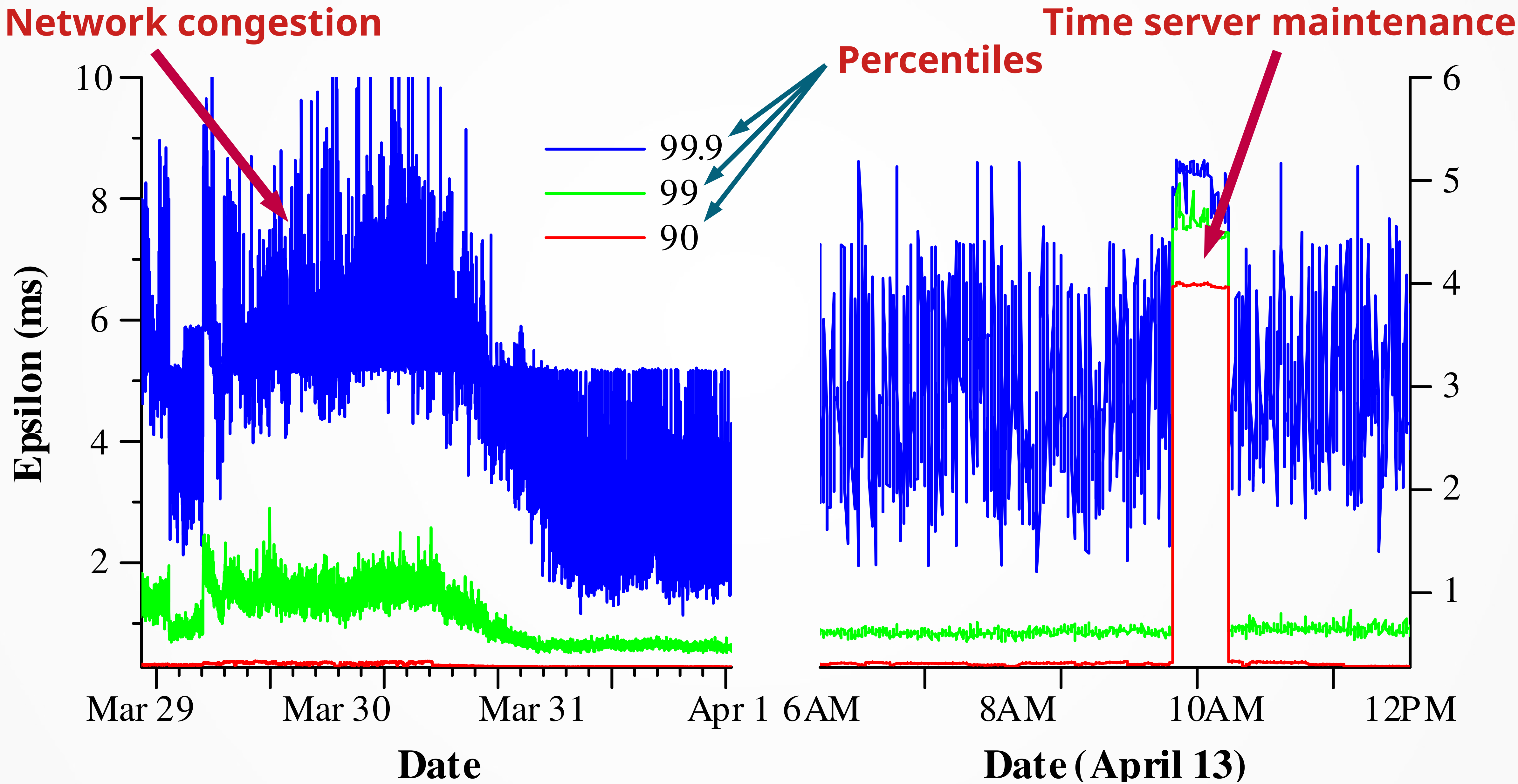
- Create a list containing *(a, +1)* and *(b,-1)* pairs for each *[a,b]* interval and sort them by the first element
- Let's call these values (v_i, d_i); $d_i \in \{-1, 1\}$
- Compute the cumulative sums s_i of all d_j values where $j \leq i$
  - It's the number of intervals overlapping in $[v_i, v_{i+1}]$
- Find the maximum value s_M; the result will be $[v_M, v_{M+1}]$

Catering for Local Clock Error



- After the clock is synchronized, the uncertainty grows according to worst-case assumptions on computers’ clock drift
- Clocks that “go crazy” are very rare (6 times less than CPUs that do)

TrueTime Precision



Making Transactions Linearizable

Version Management

Time	My friends	X's friends	My posts
4	[X]	[me]	
8	[]	[]	
15			["Government bad"]

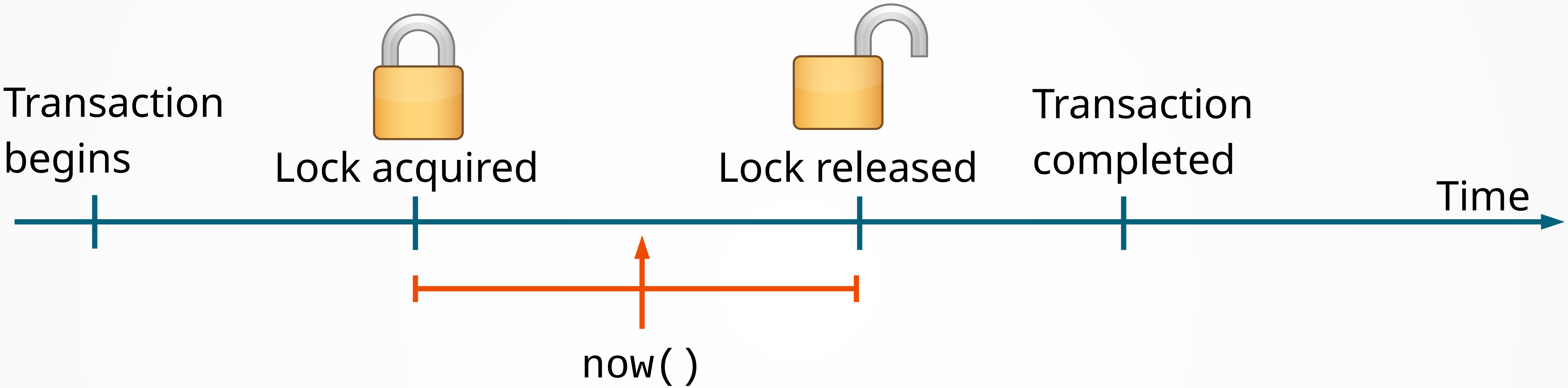
- X will never read my post:
  - The transactions that removes my friend happens before my complaint
  - A read timestamped before 8 won't see the post
  - With a read timestamped after 15, X won't see my profile
- Even if the write transactions happen in completely different clusters…

Data Model

- A key-value store
  - I.e., lookup the value for a given string, as if it was a huge hashtable
  - Holds data like

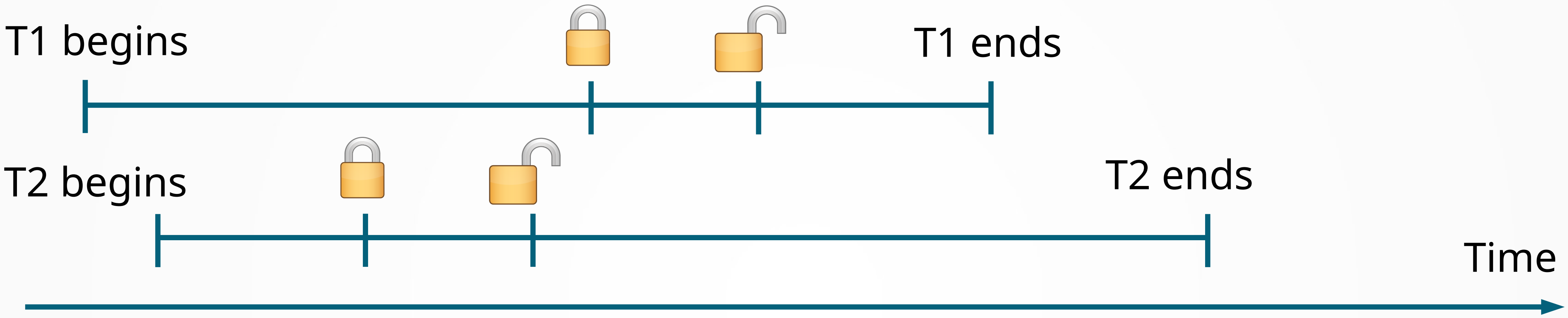
    `(key:string, timestamp:int64)` $\rightarrow$ `string`
- Nodes responsible of a key in **multiple continents**
  - Use Paxos to get consensus
- Allows asking the value **at a given moment in time**
- SQL-like semantics added afterwards

Assigning Timestamps to Writes



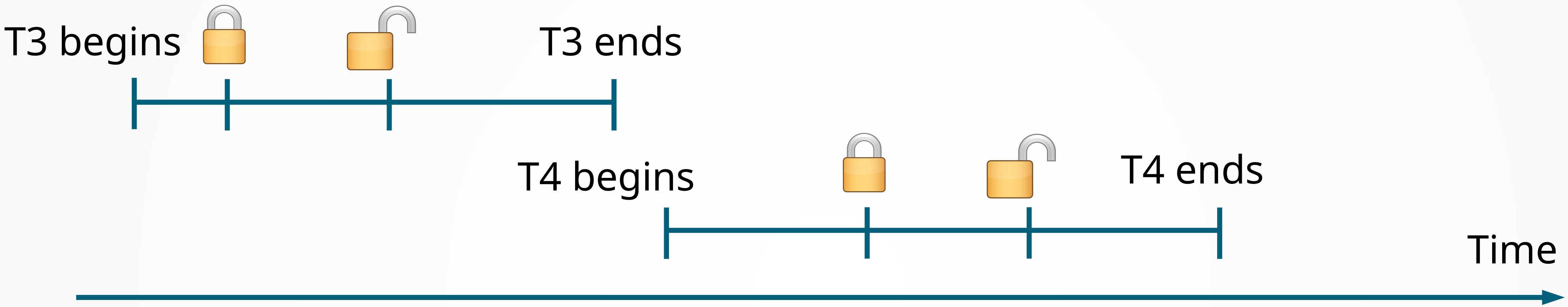
- The timestamp used is **any moment when the lock is acquired**

Conflicting Transactions



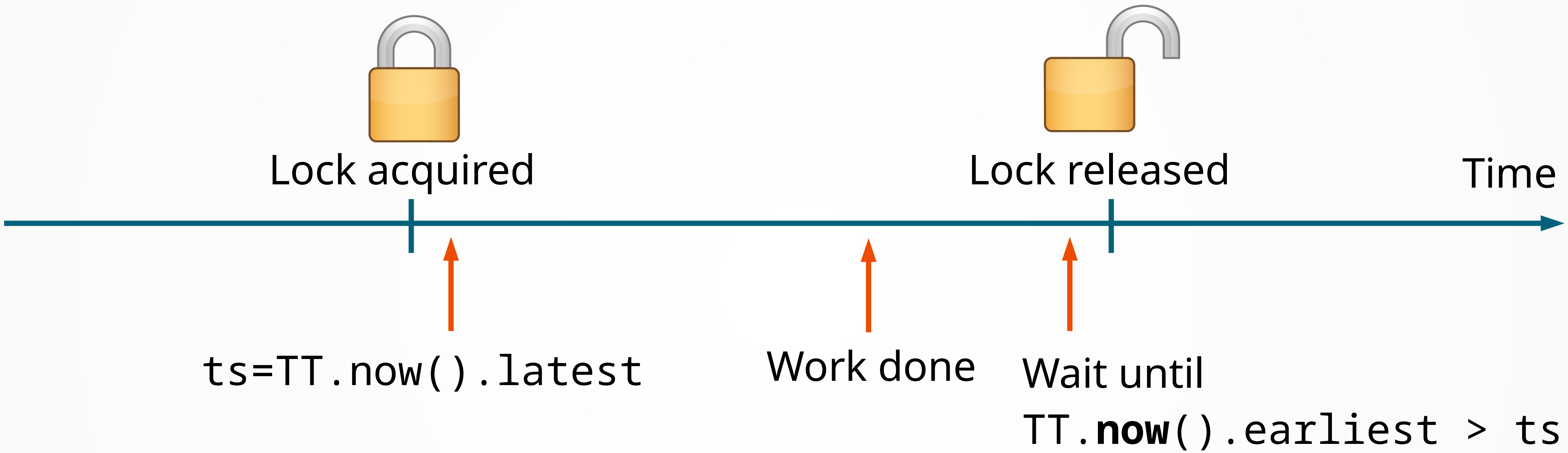
- Two transactions that touch conflicting data will have disjoint locking intervals (that’s how locks work!)
- Here, T2 will see data “before” T1, and will be timestamped before it!

Non-Conflicting Transactions



- If a transaction is executed before another, it will be timestamped before it

TrueTime to Assign Timestamps



- If the system is afraid of finishing before `ts`, it just **waits** until there's no doubt
- If uncertainty on time is too large, **the system gets slowed down**
- Clock uncertainty should be smaller than a transaction length
  - Paxos intercontinental consensus latency: 100s of ms
  - TrueTime latency: generally <10 ms