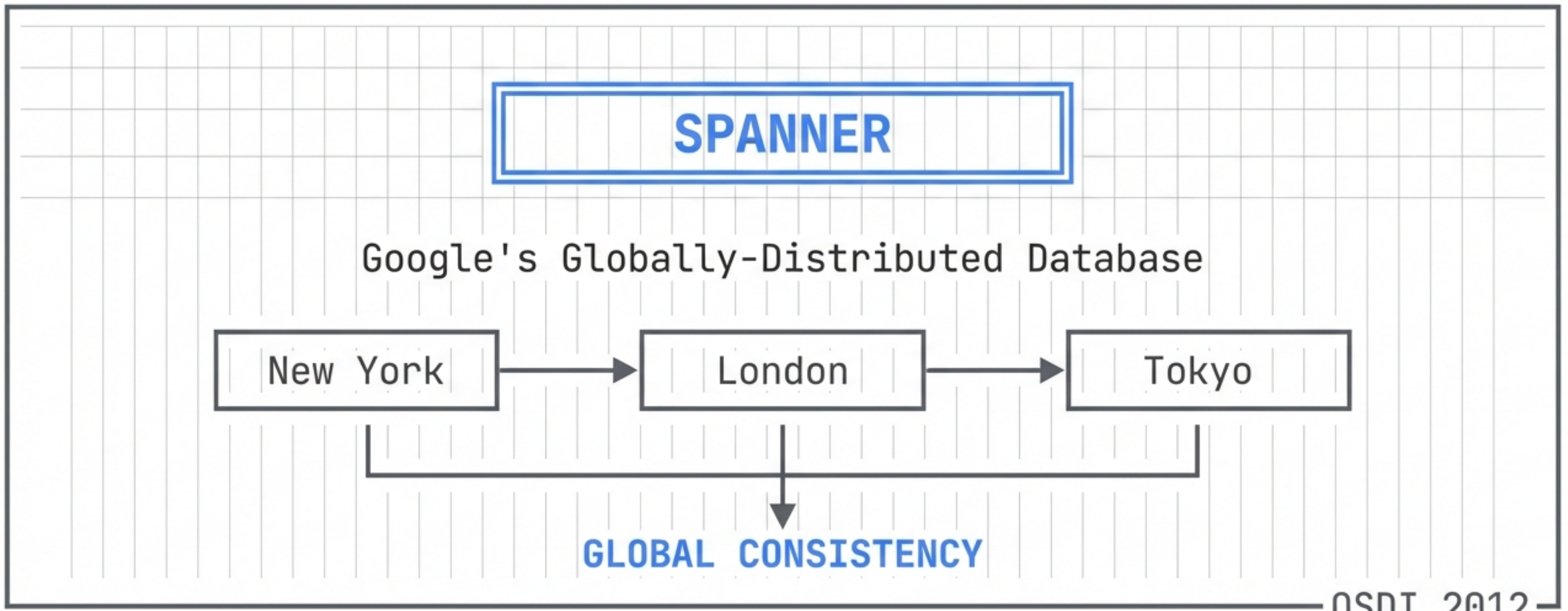
SPANNER: GLOBAL CONSISTENCY

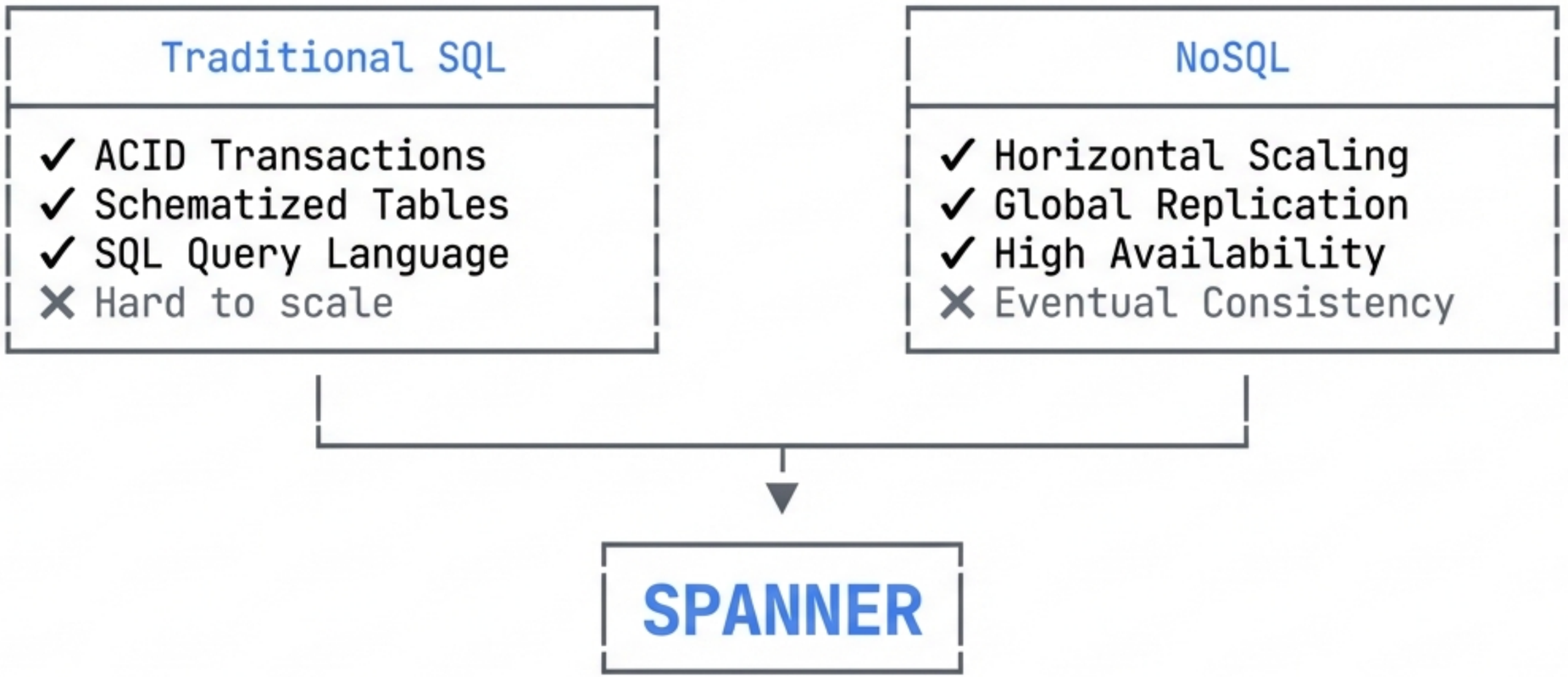
Based on Wilson Hsieh (OSDI '12)



> Spanner combines the scalability of NoSQL with the structure and guarantees of SQL. It is the backbone of Google's critical infrastructure.

WHAT IS SPANNER?

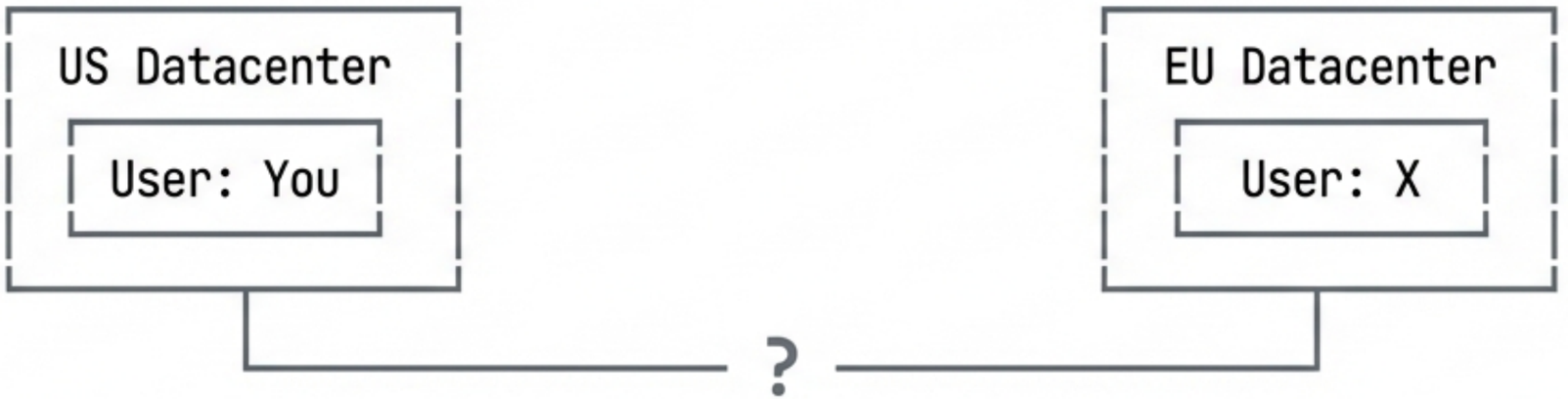
It bridges the gap between two traditional database worlds:



> Spanner was designed to replace sharded MySQL instances. It powers Google Ads, Gmail, and other mission-critical services that require both scale and strict correctness.

THE MOTIVATING EXAMPLE: A SOCIAL NETWORK

Imagine a global social graph sharded across continents.



The Privacy Scenario:

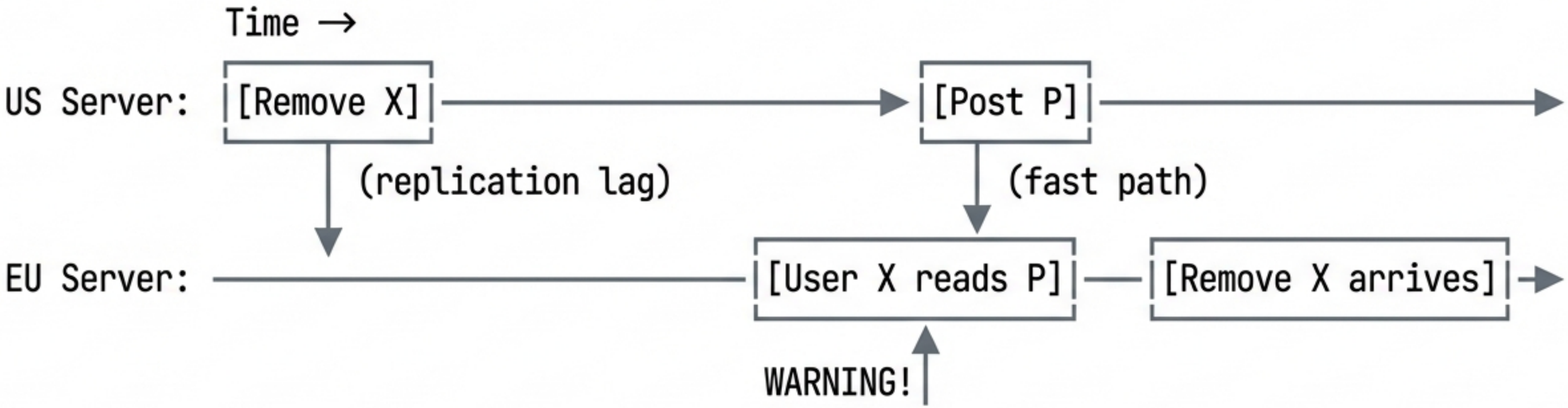
1. You decide to remove "X" from your friend list.
2. Immediately after, you post a sensitive message "P".

Requirement: X must NEVER see post P.

> This seems simple on one machine. But when "You" are in the US and "X" is in Europe, the speed of light and network latency make coordinating the order of these two events difficult.

THE PROBLEM: EVENTUAL CONSISTENCY

Without global synchronization, operations can appear out of order.

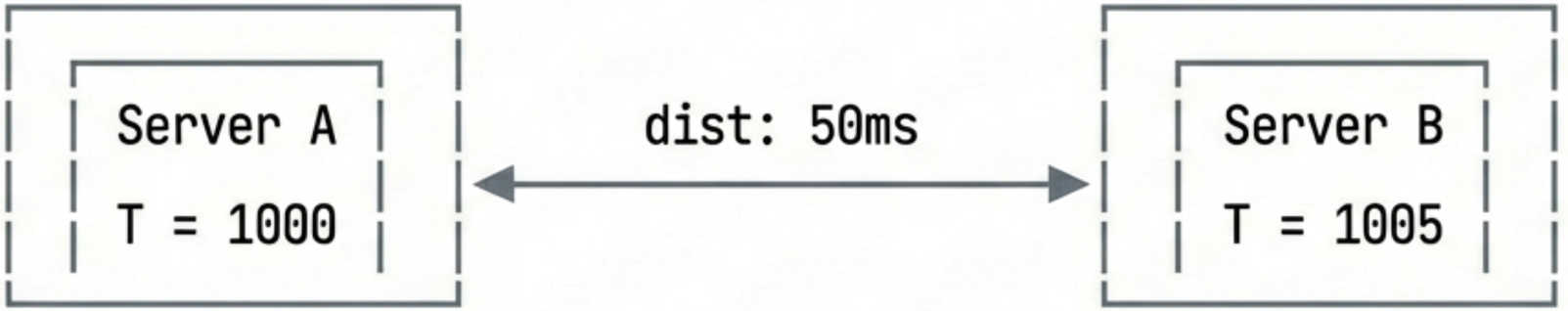


RESULT: **Broken Privacy.** User X saw the post before the system realized they were unfriended.

> In many NoSQL systems, “eventual consistency” means X will eventually be removed, but the interim state is dangerous. Spanner guarantees “External Consistency”—the system behaves as if all transactions happened sequentially in real time.

THE CHALLENGE: RELATIVITY

To solve this, we need to know exactly 'when' things happen. But there is no global clock in a distributed system.



Sources of Uncertainty

1. **Clock Drift:** Quartz crystals drift (200 µs/sec).
2. **Latency:** Communication takes time.

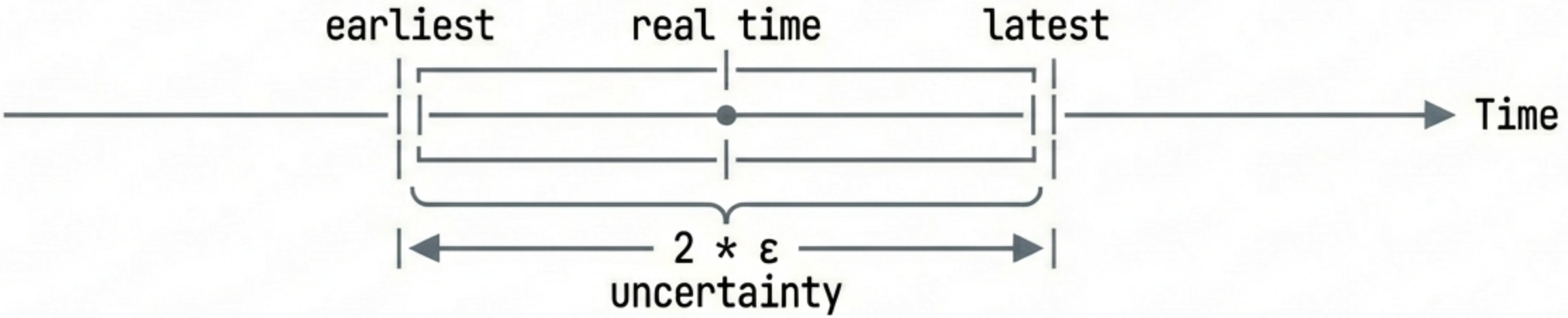
If we can't agree on the time, we can't agree on the order.

> Most systems give up here. Spanner solves this by not fighting the uncertainty, but by measuring it and integrating it into the API.

THE INNOVATION: TRUETIME API

Spanner does not treat time as a point. It treats it as an interval with bounded uncertainty.

Function: TT.now()
Returns: [earliest, latest]

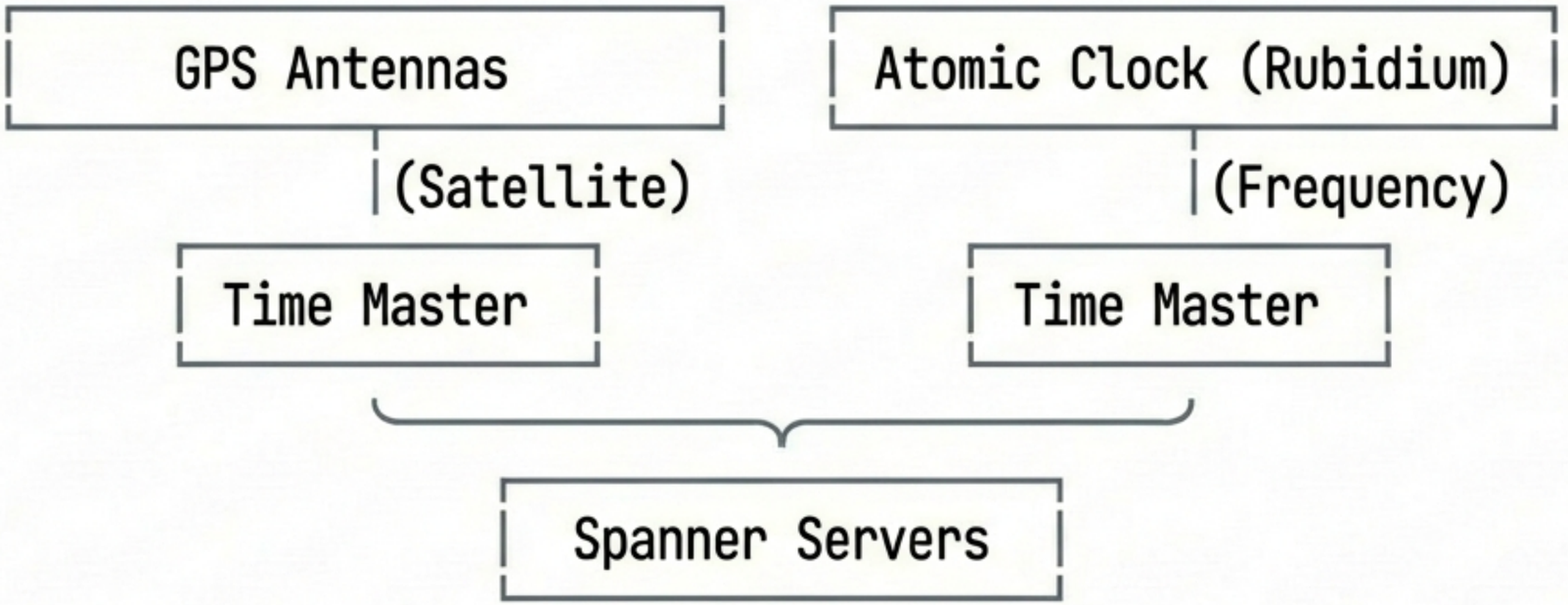


Guarantee: The actual time is somewhere inside the box.

> Epsilon (ε) represents the uncertainty. By explicitly exposing this interval to the database software, Spanner can make logical decisions based on worst-case scenarios.

TRUETIME INFRASTRUCTURE

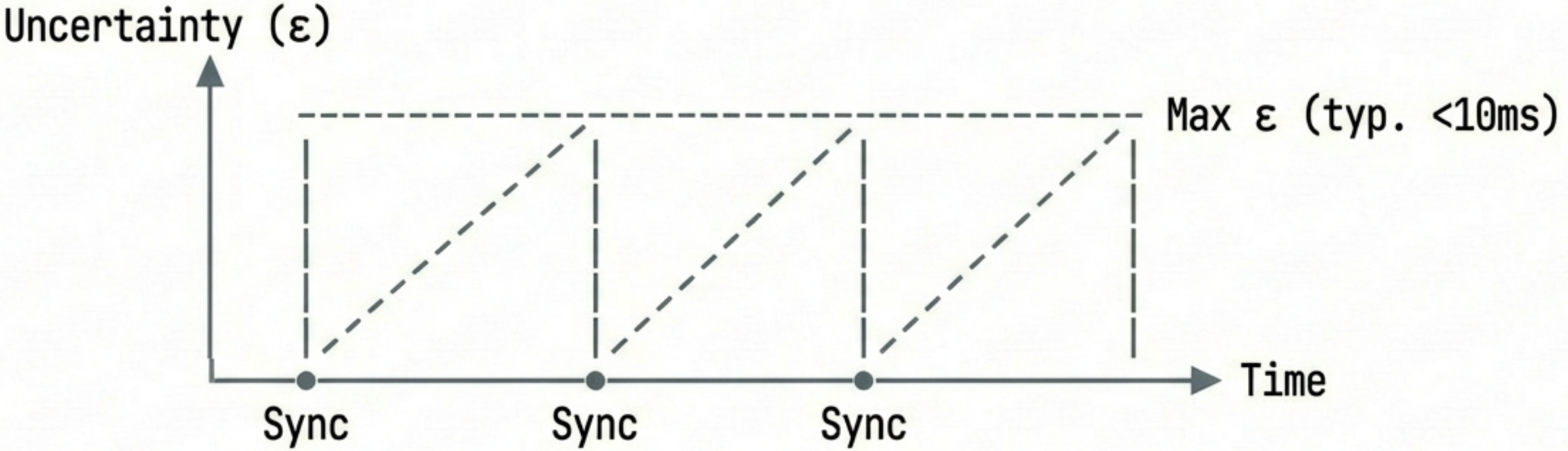
How does Google keep ε (uncertainty) small?
By deploying hardware "Time Masters" in every datacenter.



> GPS provides the primary time source. Atomic clocks act as a fallback ("Armageddon" resilience) if GPS fails or is jammed. The combination ensures high availability and accuracy.

MANAGING UNCERTAINTY (THE SAWTOOTH)

Between clock syncs, uncertainty grows due to drift.



Spanner daemons poll masters every 30 seconds.
If error > max bound, the server marks itself unhealthy.

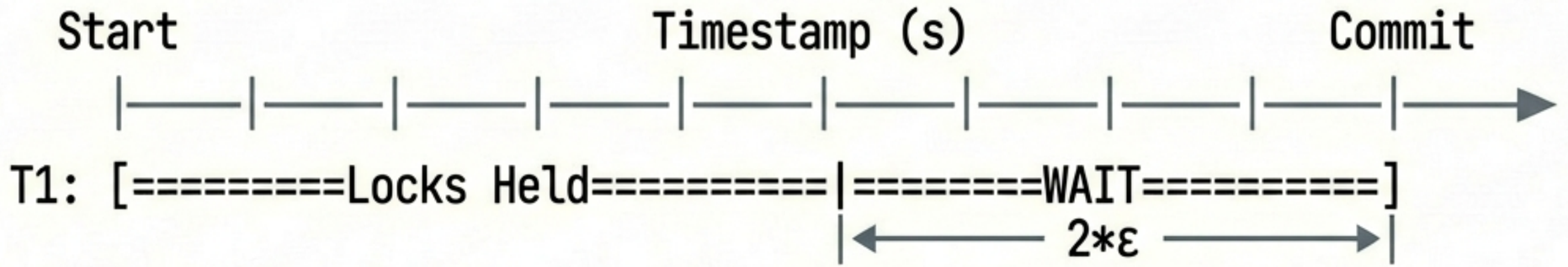
> The system constantly fights entropy. The uncertainty is empirically bounded.
During normal operation, the window (ε) is usually less than 10 milliseconds.

TRANSACTION STRATEGY: COMMIT WAIT

The ‘Secret Sauce.’ How do we ensure transaction T1 definitely happens before T2? We wait out the uncertainty window.

Protocol:

1. Acquire Locks
2. Pick Timestamp `s` = `TT.now().latest`
3. **WAIT** until `TT.now().earliest` > `s`
4. Commit & Release Locks

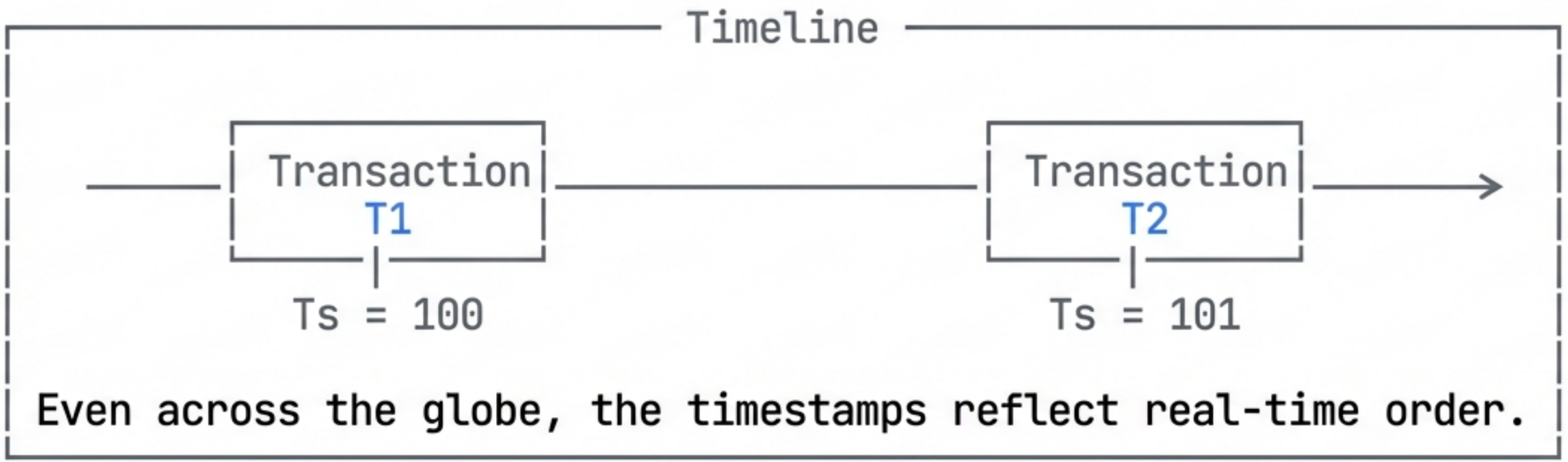


> By waiting until the ‘earliest’ possible current time is past the timestamp `s`, we guarantee that `s` is definitely in the past. No other machine can perceive the current time as being earlier than `s`.

EXTERNAL CONSISTENCY

Because of Commit Wait, we achieve this invariant:

If T2 starts after T1 finishes ⟶ Timestamp(T2) > Timestamp(T1)



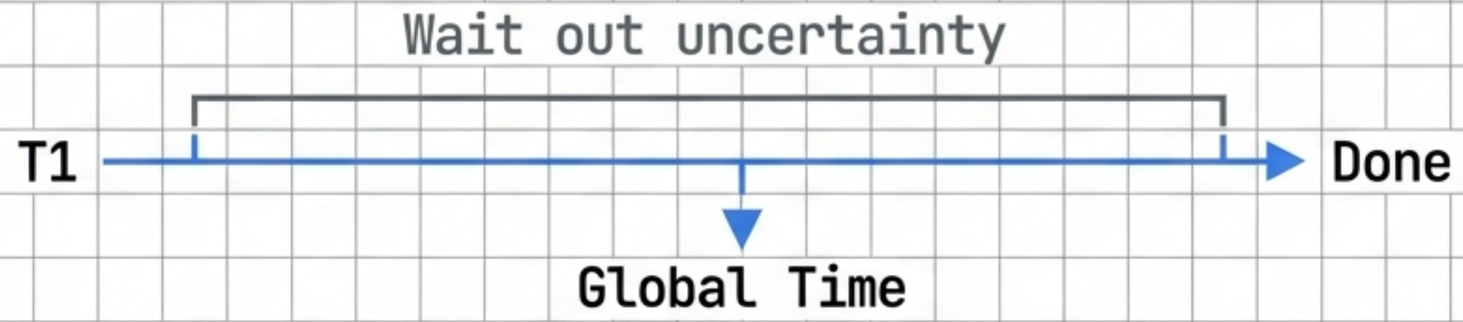
> This is "External Consistency" (or Linearizability). It is the strongest consistency model. It effectively makes the distributed system look like a single machine to the application developer.

THE SOCIAL NETWORK EXAMPLE: RESOLVED

Let’s revisit the Privacy Scenario with Spanner.

T1: Remove Friend (Timestamper picks s=8)

- System waits until time 8 is definitely past.
- Commit at real-time ~8.01



T2: Post Message (Starts after T1 finishes)

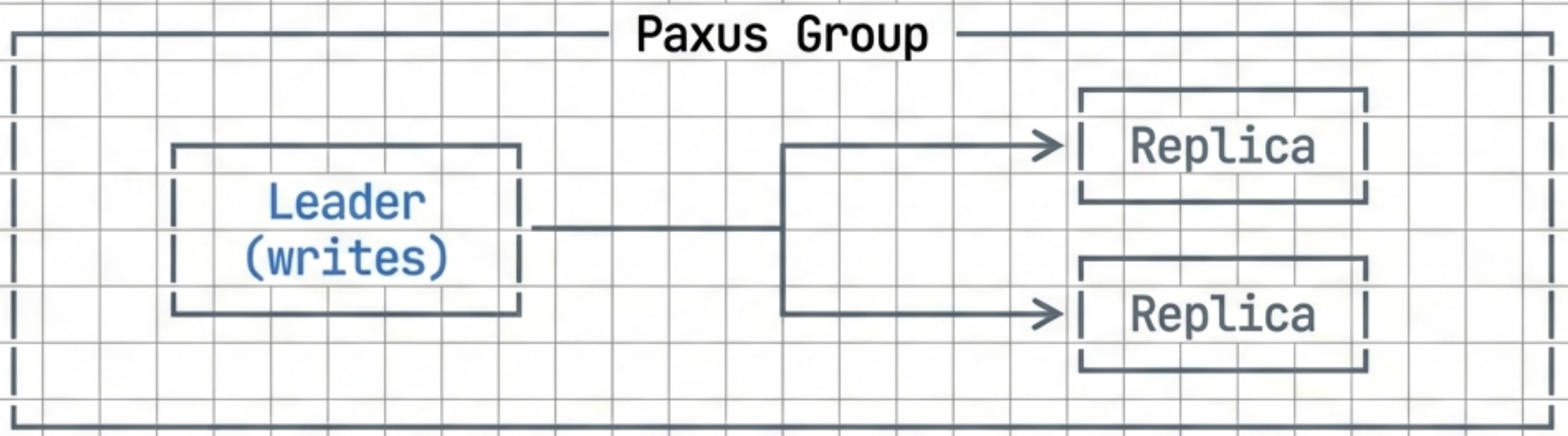
- Must pick timestamp > 8 (e.g., s=15)

Result: Snapshots at time 15 reflect the removal at time 8. User X does NOT see the post.

> Because T1 waited out the uncertainty before telling the user ‘Success’, T2 is guaranteed to receive a higher timestamp. The causal link is preserved.

ARCHITECTURE: REPLICATION

Spanner sits on top of a massive replication system.

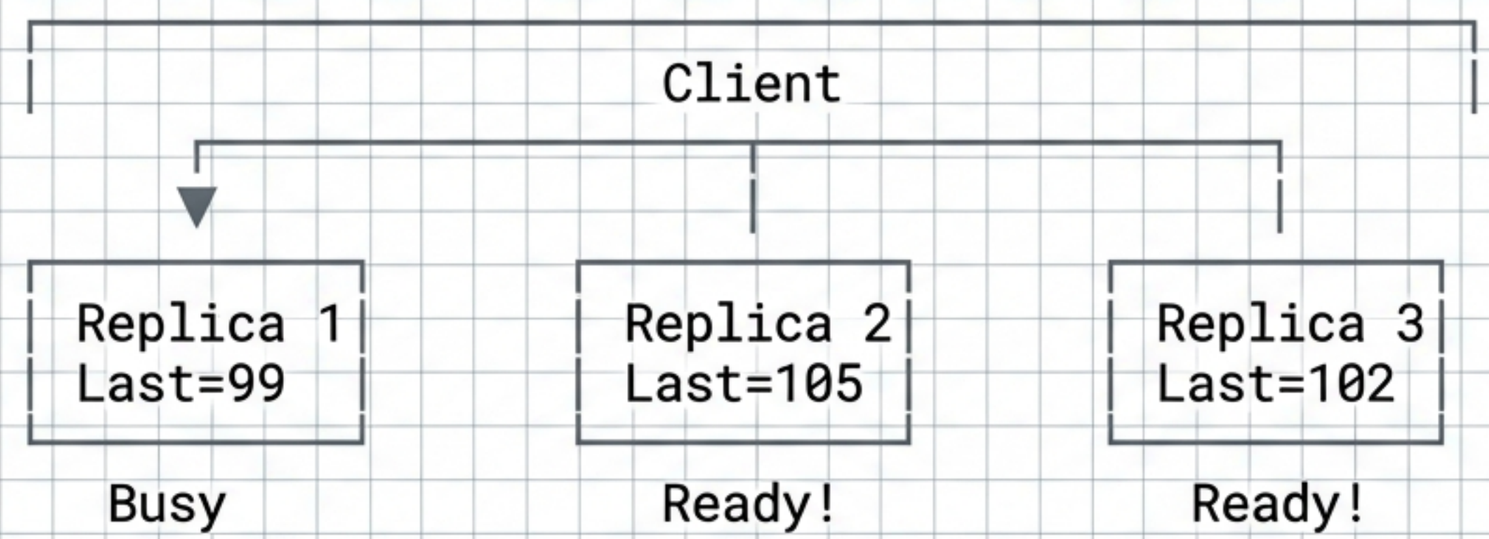


- Writes go through the Leader (Paxos).
- The "Commit Wait" happens at the Leader.
- Reads can happen at any Replica (if up to date).

> While TrueTime handles the global ordering, standard Paxos handles the local replication and durability. Spanner combines these into a cohesive whole.

SNAPSHOT READS

A superpower of Spanner is “Lock-Free Reads” in the past.
“Give me the database state at Timestamp = 100”



Replica 2 & 3 can serve the read without locking anything!

> Because data is immutable at specific timestamps (Multi-Version Concurrency Control), you can read the past without blocking incoming writes. This is massive for performance (e.g., MapReduce jobs, backups).

KEY TAKEAWAY: KNOWN UNKNOWNS

The philosophy of Spanner is better than its hardware.

Standard Systems	Spanner
We assume clocks are synced. Result: Unknown errors on drift.	We know clocks are uncertain. Result: We wait out the error.

Design Principle: Better to be slow (wait) than wrong.

> By reifying uncertainty into the API, Spanner forces the system to acknowledge the physical limitations of distributed computing, resulting in a more robust system.

CONCLUSIONS

CONCLUSIONS

Spanner proves that we don't have to choose between:

Global Scale AND Strong Consistency

**The Recipe*:
1. **GPS + Atomic Clocks*** (Hardware)
2. **TrueTime API*** (Software Abstraction)
3. **Commit Wait*** (Algorithm)

= The first system to provide external consistency at global scale.

> Spanner has become the standard for database design at Google, enabling developers to write code as if they were on a single machine, while operating on a global scale.