PTP-based fbclock vs. HLC

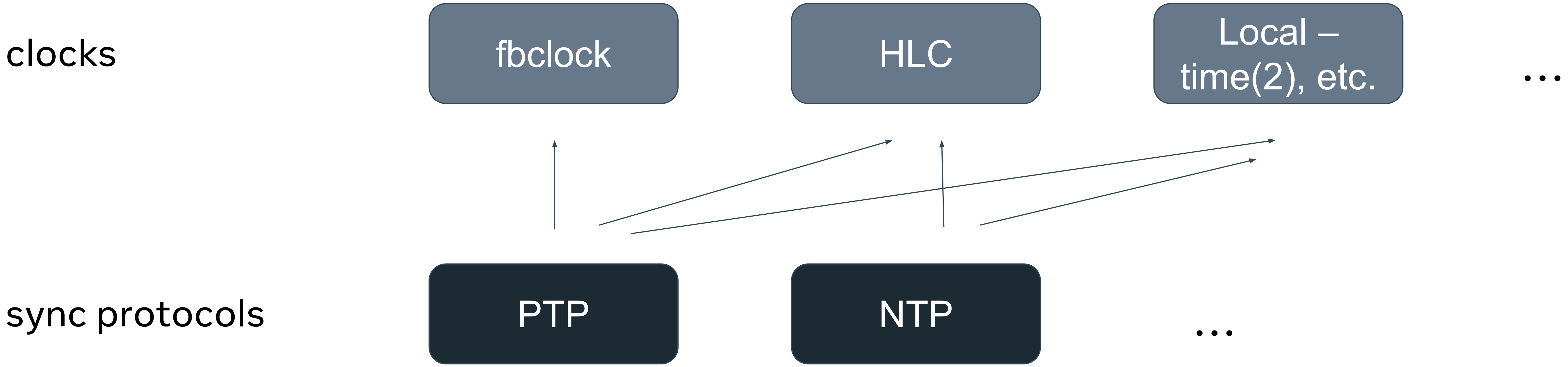
from a distributed database’s perspective

April 26, 2023

Lu Pan
Software Engineer

Meta

Clock sync protocols and Clocks



Agenda

01 PTP-based fbclock

02 Commit-wait

03 Linearizability

04 HLC

05 PTP-based HLC vs. NTP-based HLC

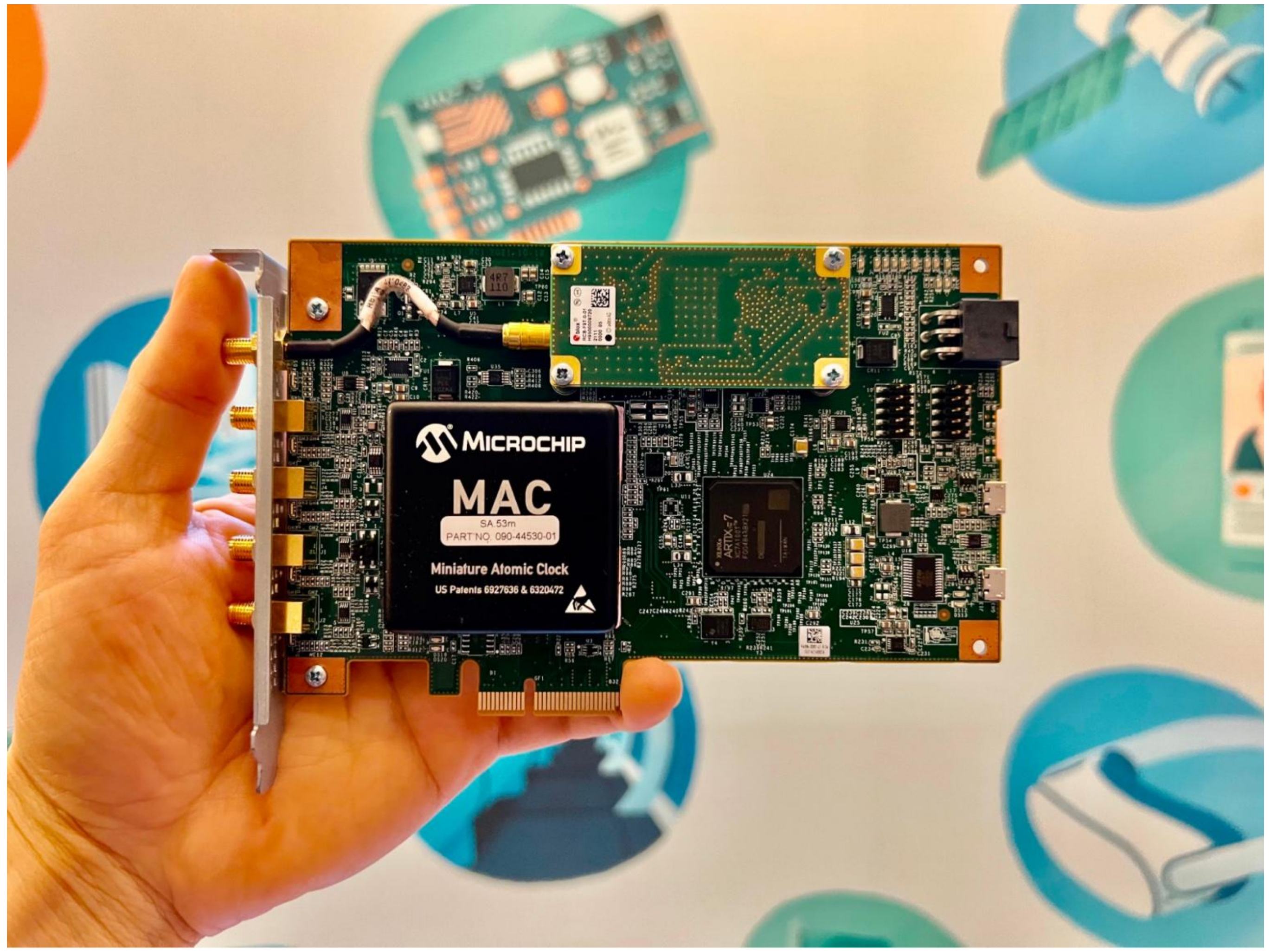
06 fbclock vs. HLC

PTP-based fblock

- PTP – Precision Time Protocol – is a hardware based clock synchronization protocol
- fbclock exposes a TrueTime-like API, first introduced by Spanner [Google OSDI'12]

```
typedef struct fbclock_truetime {
  uint64_t earliest_ns;
  uint64_t latest_ns;
} fbclock_truetime;
```

https://github.com/facebook/time/tree/main/fb



Commit-wait

Write Transaction Ordering

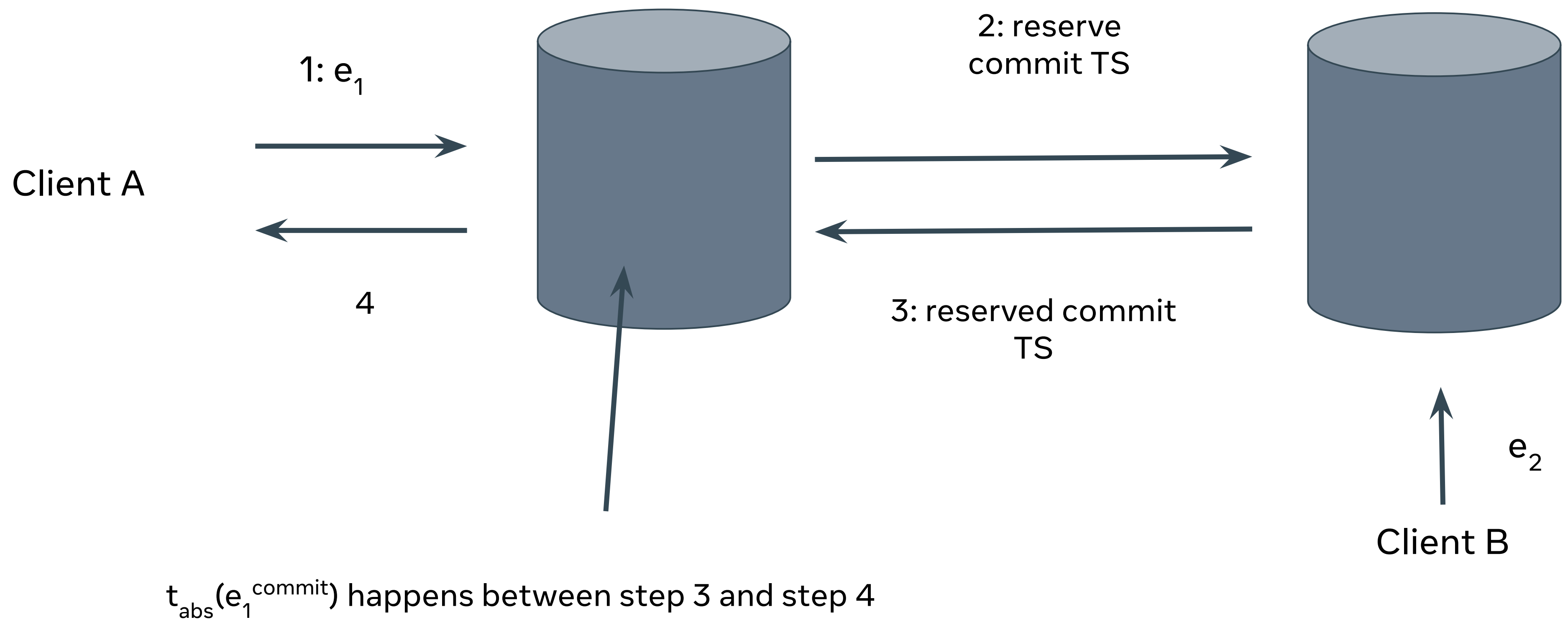
$$t_{abs}(e_1^{commit}) < t_{abs}(e_2^{start}) \Rightarrow s_1 < s_2$$

Define the start and commit events for a transaction T_i by e_i^{start} and e_i^{commit}; and the commit timestamp of a transaction T_i by s_i.
Denote the absolute time of an event e by $t_{abs}(e)$.

https://www.usenix.org/conference/osdi12/technical-sessions/presentation/corbett

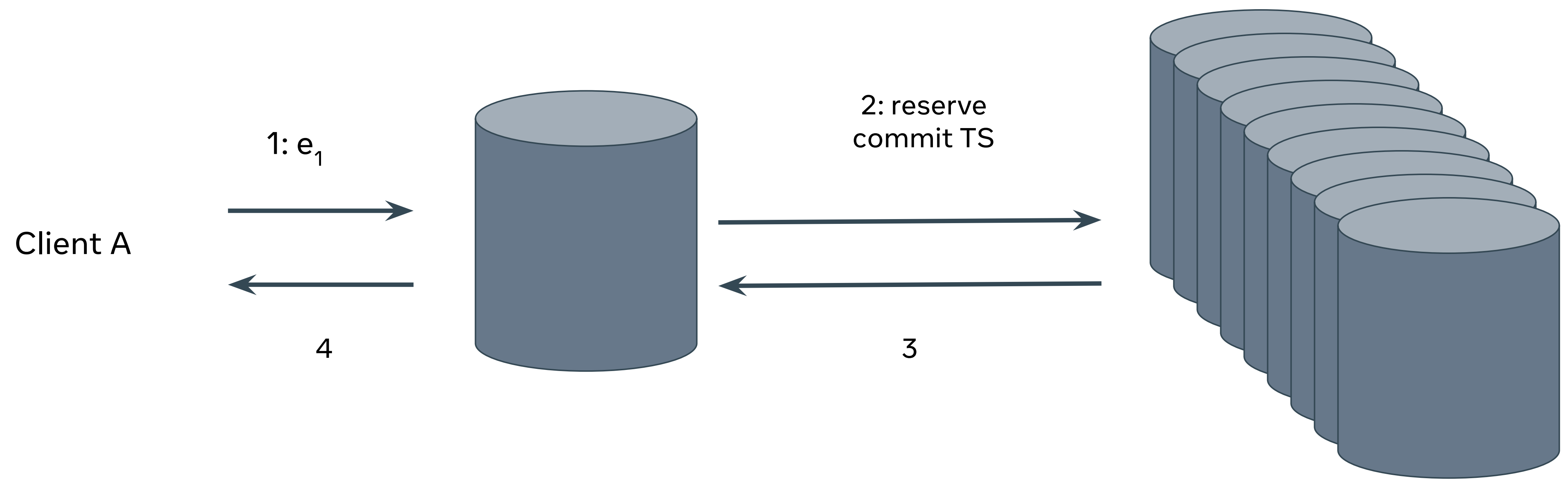
Naive Solution – coordination

$$t_{abs}(e_1^{commit}) < t_{abs}(e_2^{start}) \Rightarrow s_1 < s_2$$



Commit timestamps are monotonically increasing locally

Naive Solution – doesn’t scale



Commit-wait and order construction

Ordering two transactions using TrueTime and commit-wait

$$t_{abs}(e_1^{commit}) < t_{abs}(e_2^{start}) \Rightarrow s_1 < s_2$$

Define the start and commit events for a transaction T_i by e_i^{start} and e_i^{commit}; and the commit timestamp of a transaction T_i by s_i.
Denote the absolute time of an event e by $t_{abs}(e)$.

https://www.usenix.org/conference/osdi12/technical-sessions/presentation/corbett

Commit-wait and order construction

$$
\begin{aligned}
s_1 &< t_{abs}(e_1^{commit}) && \text{(commit wait)} \\
t_{abs}(e_1^{commit}) &< t_{abs}(e_2^{start}) && \text{(assumption)} \\
t_{abs}(e_2^{start}) &\leq t_{abs}(e_2^{server}) && \text{(causality)} \\
t_{abs}(e_2^{server}) &\leq s_2 && \text{(start)} \\
s_1 &< s_2 && \text{(transitivity)}
\end{aligned}
$$

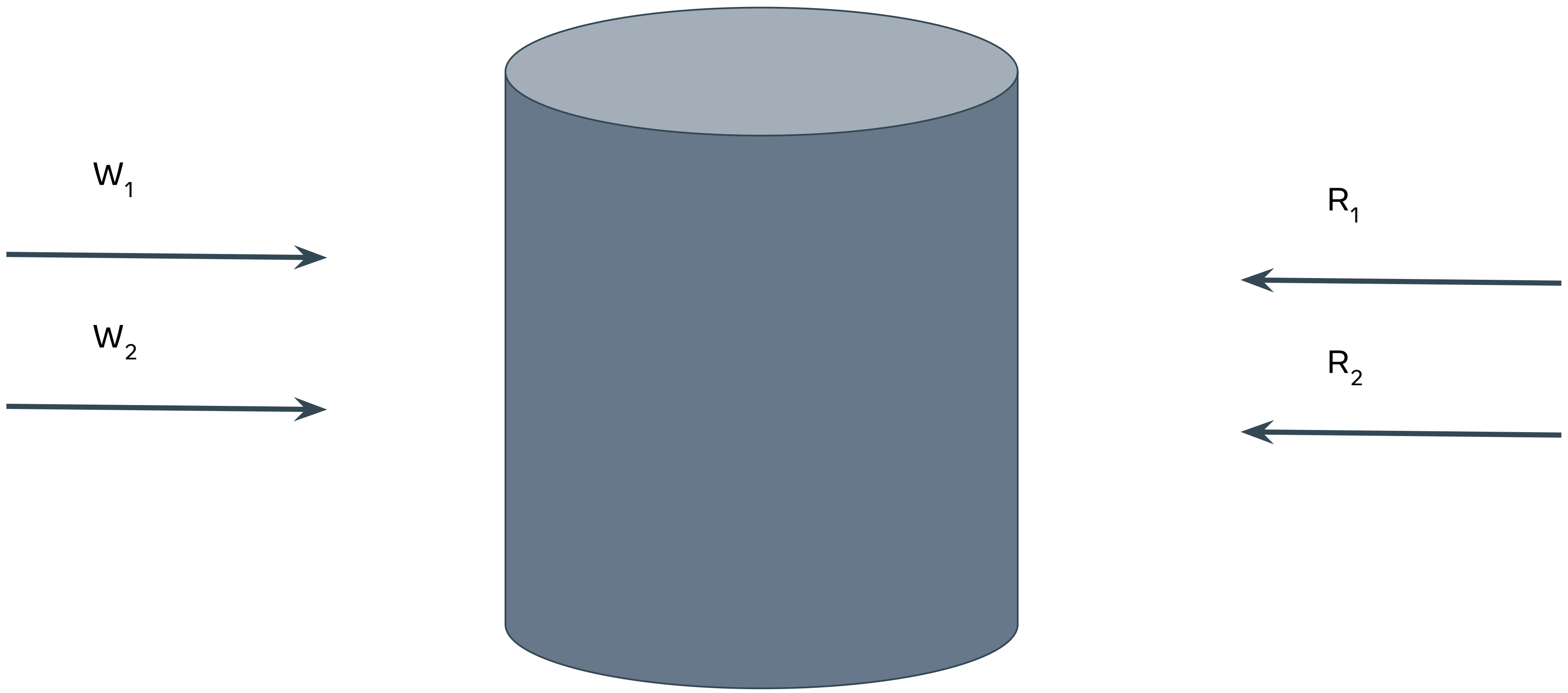
Define the start and commit events for a transaction T_i by e_i^{start} and e_i^{commit}; the arrival event of the commit request to be e_i^{server} , and the commit timestamp of a transaction T_i by s_i.
Denote the absolute time of an event e by $t_{abs}(e)$.

External Consistency (aka Linearizability)

… every operation appears to take place atomically, in some order, consistent with the real-time ordering of those operations

https://jepsen.io/consistency/models/linearizable

Linearizability is **more than** capturing causality everywhere



- Send two concurrent writes W1 and W2
- While W_1 and W_2 are still inflight, send two more concurrent read operations R_1 and R_2
- R_1 observes W_1 but not W_2
- R_2 observes W_2 but not W_1

Spanner’s solution – ordering reads with timestamps (surprise)

- Read from Paxos group leader for single shard read operation
- Use read timestamp TT.now().latest for multi-shard linearizable reads

HLC – Hybrid Logical Clock

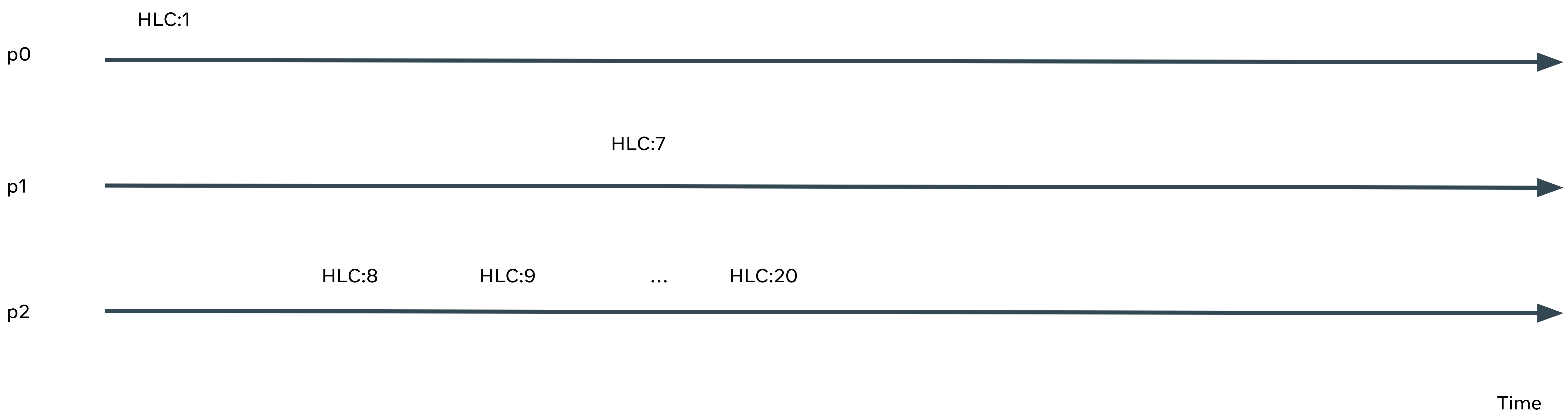
- Proposed by Kulkarni, et al. in 2014 as an alternative to TrueTime and logical clocks (Logical Clock, Vector Clock, etc.)
- Capable of providing consistent* snapshots without special hardware or commit-wait
- Naive HLC implementation

next_hlc = max(highest_observed_timestamp + 1, physical_timestamp)

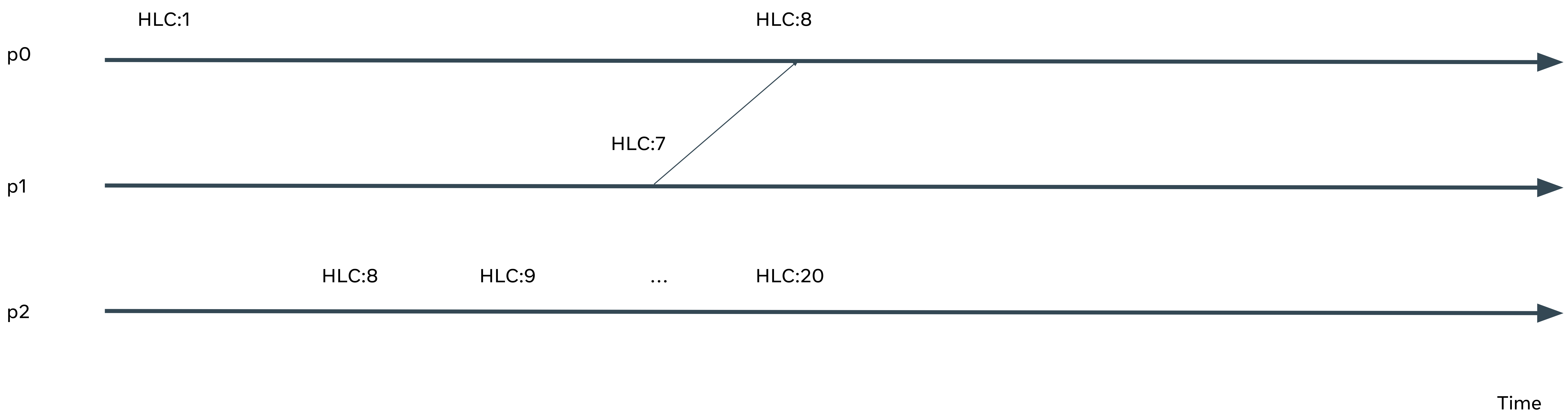
- Capture causality – if event *e* **happens-before** *f*, HLC(e) < HLC(f)
  - If HLC(e) == HLC(f), they must be concurrent events*

PTP-based HLC vs. NTP-based HLC

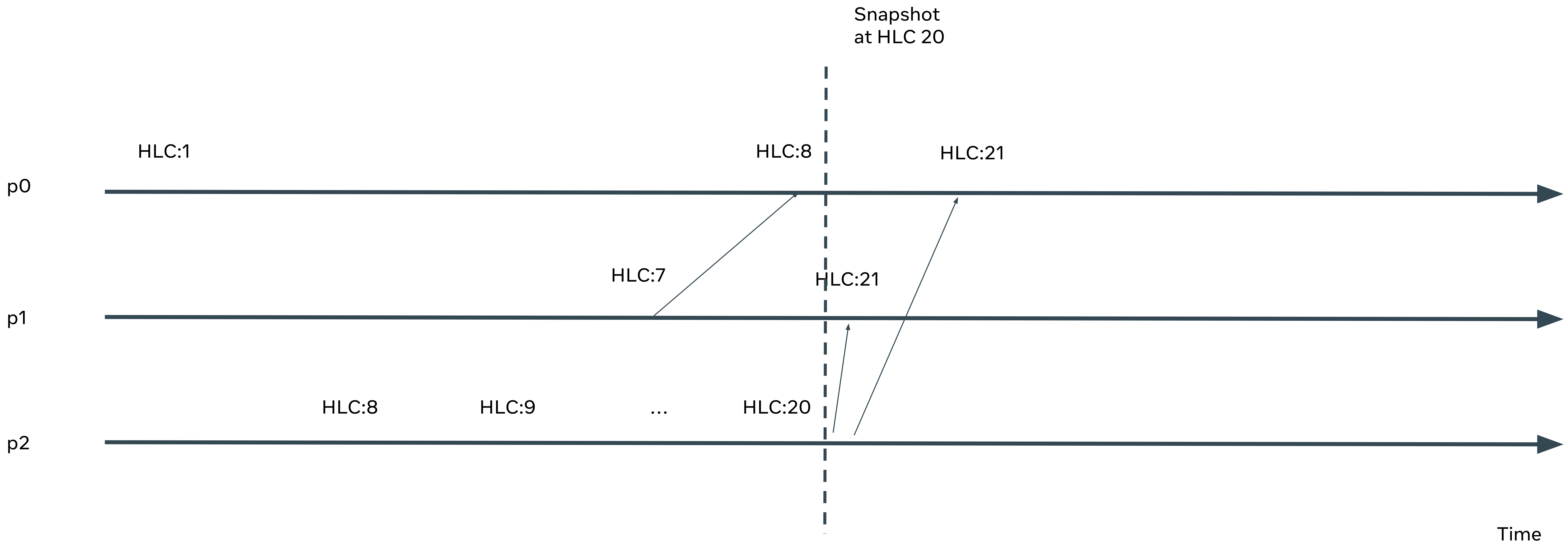
HLC stragglers and HLC rushers



HLC stragglers



HLC rushers

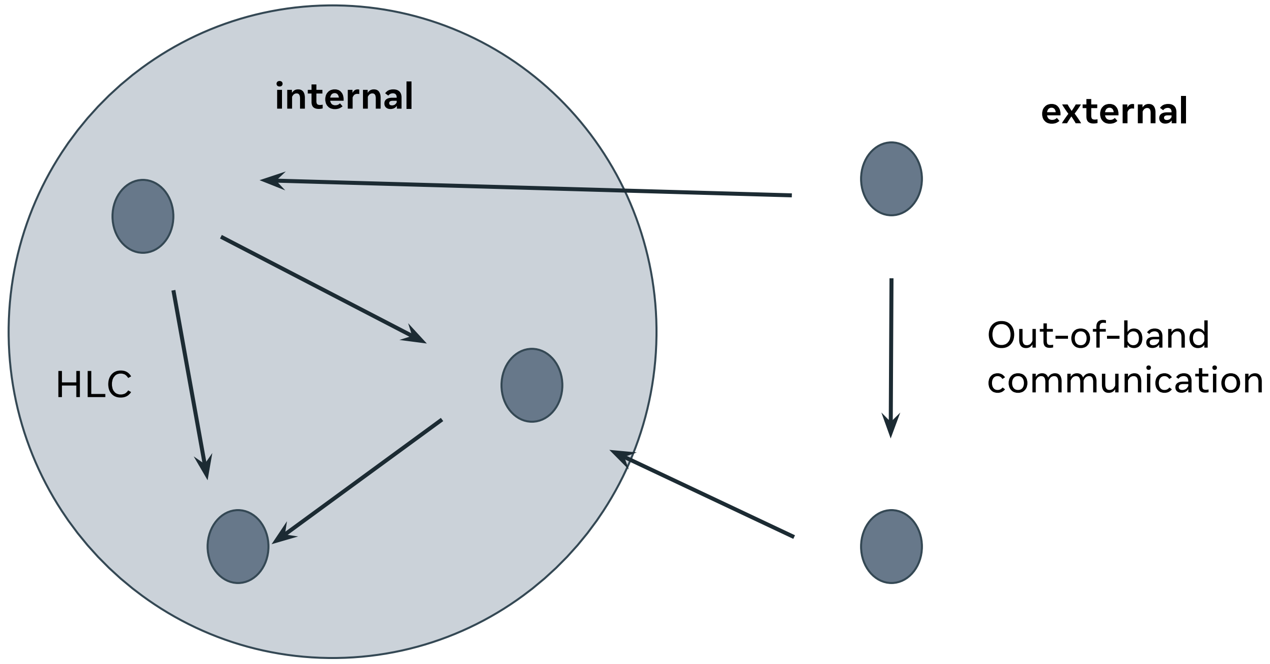


Snapshot
at HLC 20
HLC:1
HLC:8
HLC:21
p0
HLC:7
HLC:21
p1
HLC:8
HLC:9
...
HLC:20
p2
Time

fbclock vs. HLC

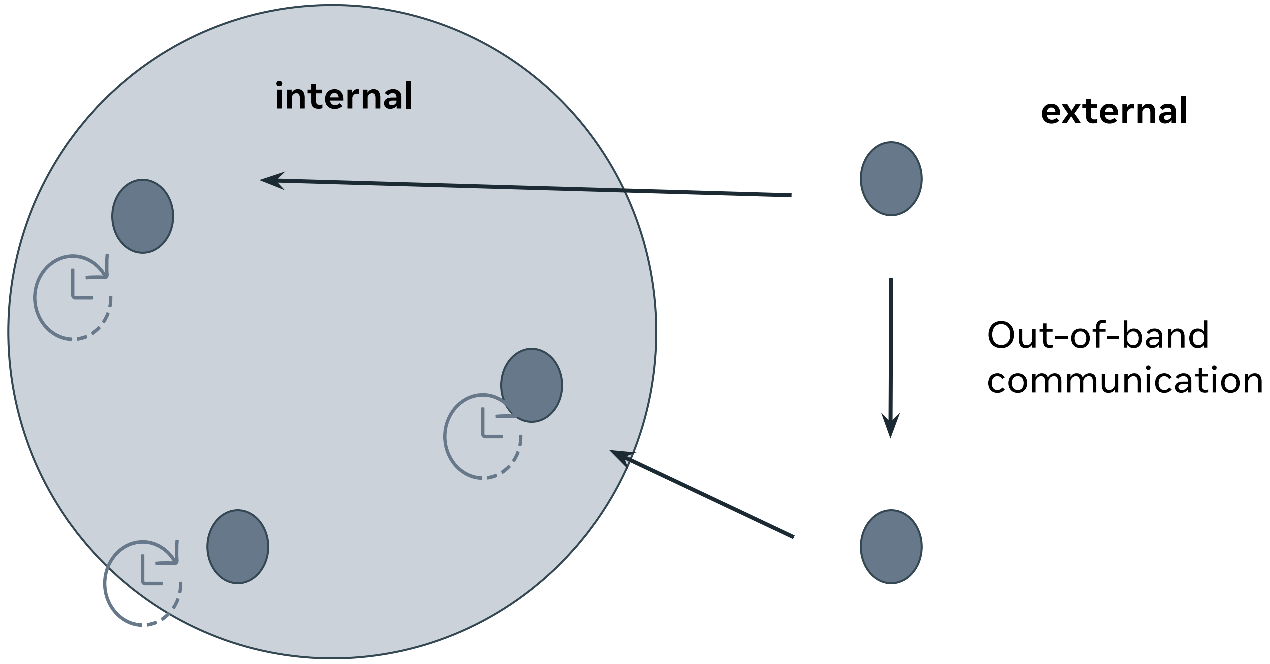
fbclock vs. HLC: Consistent Snapshots

- HLC-based snapshots
  - Capture causality – if event *e* **happens-before** *f*, HLC(e) < HLC(f)
    - If HLC(e) == HLC(f), they must be concurrent events*
  - “The causality relationship **captured**, called happened-before (hb), is defined based on passing of information, rather than passing of time.”
  - Users can observe anomalies when a system misses out-of-band communication and the causality it implies, which is simply considered out of the HLC scope.
- **Internally Consistent Snapshots**



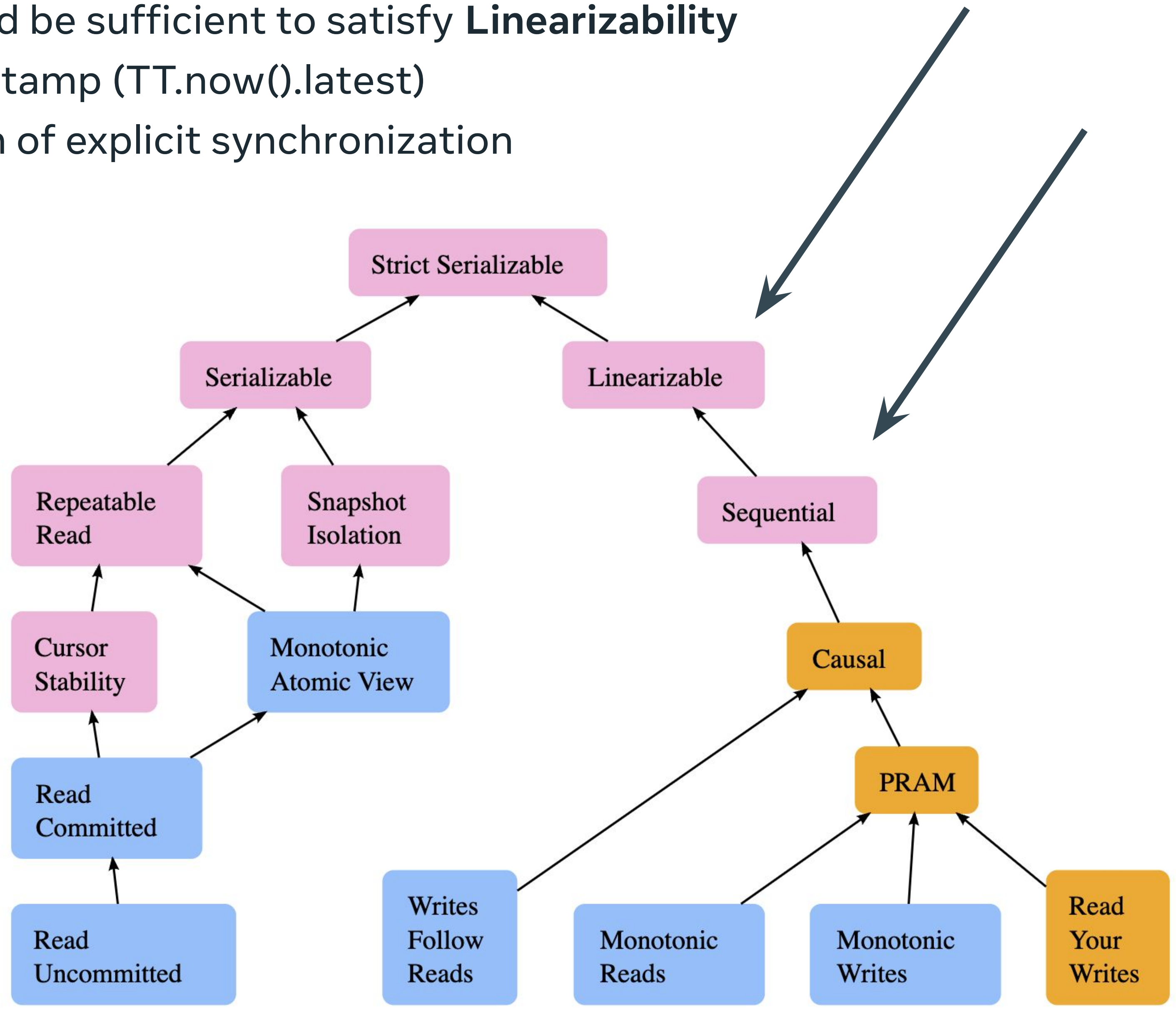
fbclock vs. HLC: Consistent Snapshots

- Fbclock-based snapshots
  - Can order events without explicit communication
  - GPS and atomic clocks only need to be inside the system to provide external consistency; while HLC guarantees are strictly limited by the scope where causality can be captured
- **Externally Consistent Snapshots**



fbclock vs. HLC: linearizable read at scale

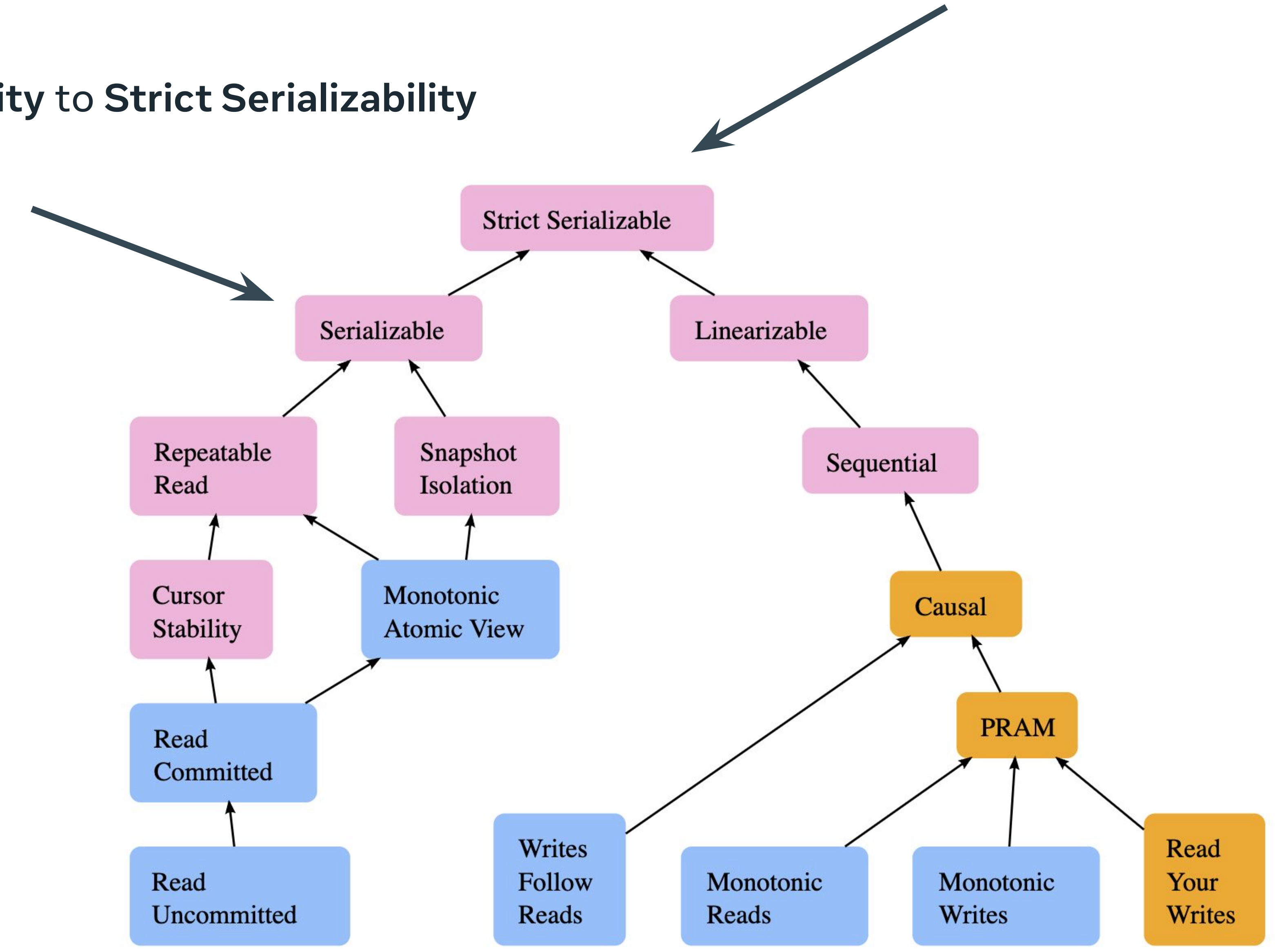
- Too much traffic to the primary database imposes a major reliability risk and scalability bottleneck
- A reply that is later than when client dispatches the read query (in absolute time) would be sufficient to satisfy **Linearizability**
  - The key challenge is to have server and client agree on a safe minimum reply timestamp (TT.now().latest)
- We can only achieve **Sequential Consistency** with HLC without resorting to some form of explicit synchronization



https://jepsen.io/consistency

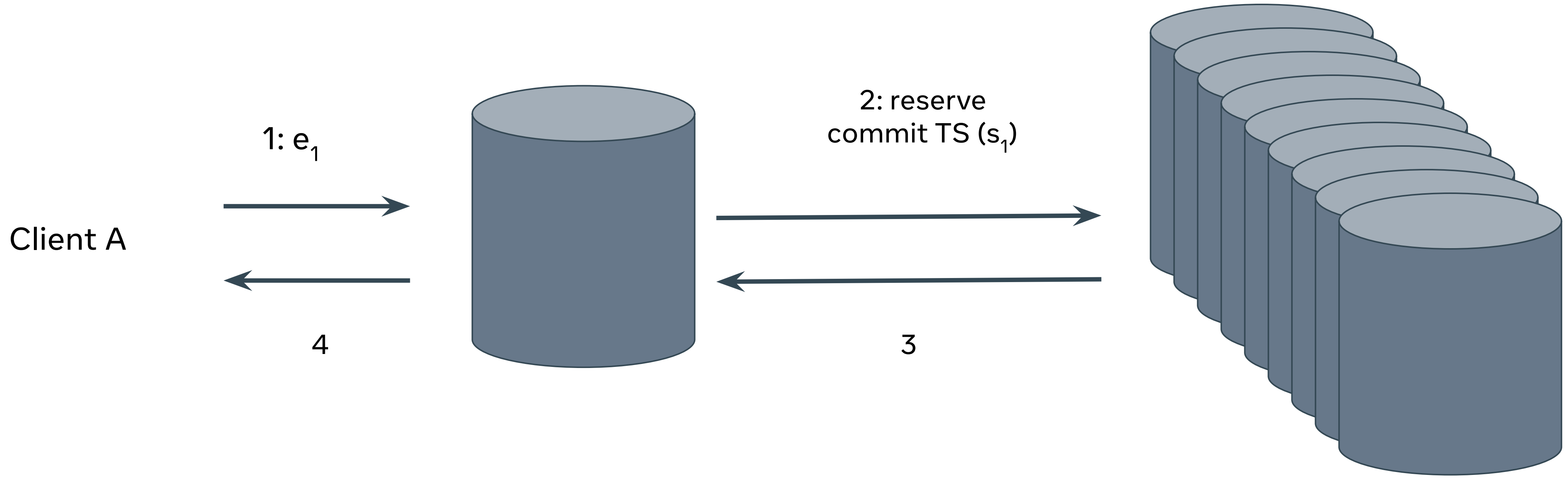
fbclock vs. HLC: Transaction Model

- If every (distributed) read-write transaction is committed at a single HLC, and every read-only transaction sees a consistent snapshot at a certain HLC, the system is **Serializable** by definition
  - We can pick an arbitrary tie-breaker for events with the same HLC
- fbclock affords linearizability, upgrading the transaction model from **Serializability** to **Strict Serializability**



https://jepsen.io/consistency

fbclock vs. HLC: Efficiency and Scalability



Meta