NewSQL

Immanuel Trummer
itrummer@cornell.edu
www.itrummer.org

Reading List

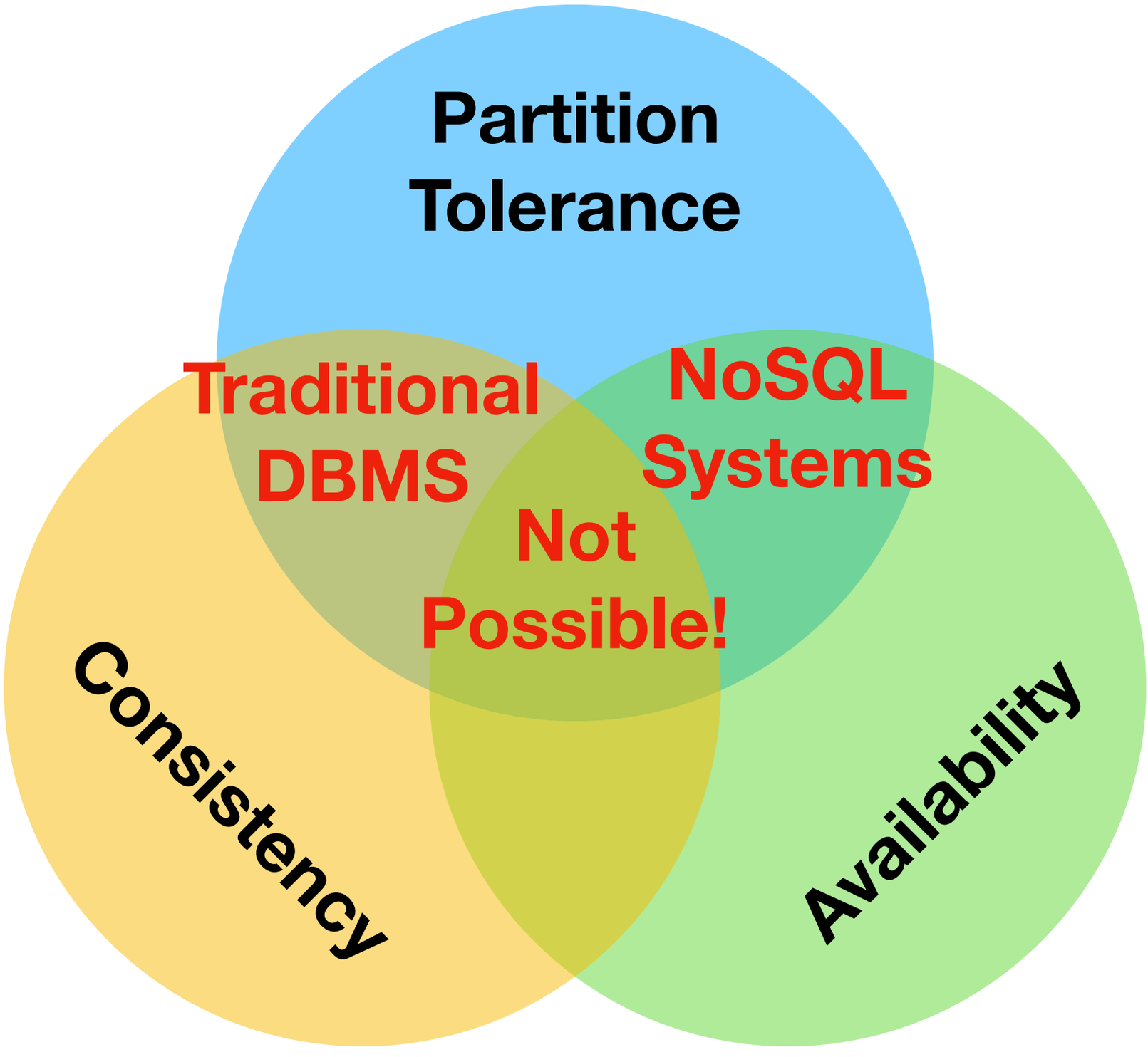
- https://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed/, E. Brewer.

- **H-Store: A High-Performance, Distributed Main Memory Transaction Processing System**,
  R. Kallman et al., 2008.

- **Spanner: Google's Globally Distributed Database**,
  J. Corbett et al., 2013.

- **NewSQL vs. NoSQL for New OLTP**, M. Stonebraker, 2011.
  https://www.youtube.com/watch?v=uhDM4fcI2aI

The CAP Theorem



CAP Criticism

- Focuses on an **extreme case**: full partitions are rare
- **Simplifies tension** between conflicting design goals
  - E.g., can decide A vs. C for single transactions
  - E.g., consistency is not a binary property
  - ...

NewSQL

- "**Traditional SQL**": ACID at the expense of performance
- **NoSQL**: give up ACID for higher performance
- **NewSQL**: new ideas for ACID with high performance

Two NewSQL Systems

- **H-Store** (VoltDB)

- **Google Spanner**

Two NewSQL Systems

- **H-Store** (VoltDB)
- **Google Spanner**

H-Store: Observations

- Modern Transaction **Workloads**
  - Short running transactions
  - No user input needed
  - Transactions ~ templates
- Modern **Hardware**
  - Main memory often fits entire DB
  - Distributed systems common

H-Store: Observations

- Modern Transaction **Workloads**
  - Short running transactions
  - No user input needed
  - Transactions ~ templates
- Modern **Hardware**
  - Main memory often fits entire DB
  - Distributed systems common

Design Implications ?

H-Store: Observations

- Modern Transaction **Workloads**
  - Short running transactions
  - No user input needed
  - Transactions ~ templates
- Modern **Hardware**
  - Main memory often fits entire DB
  - Distributed systems common

H-Store: Observations

- Modern Transaction **Workloads**
  - Short running transactions
  - No user input needed
    - Transactions ~ templates
- Modern **Hardware**
  - Main memory often fits entire DB → ***Main memory DBMS***
  - Distributed systems common

H-Store: Observations

- Modern Transaction **Workloads**
  - Short running transactions
  - No user input needed
    - Transactions ~ templates
- Modern **Hardware**
  - Main memory often fits entire DB → ***Main memory DBMS***
  - Distributed systems common → ***Replication for fault tolerance***

H-Store: Observations

- Modern Transaction **Workloads**
  - Short running transactions
  - No user input needed
  - Transactions ~ templates

→ ***Serial Execution!***

- Modern **Hardware**
  - Main memory often fits entire DB → ***Main memory DBMS***
  - Distributed systems common → ***Replication for fault tolerance***

H-Store: Observations

- Modern Transaction **Workloads**
  - Short running transactions
  - No user input needed
  - Transactions ~ templates

→ ***Serial Execution!***

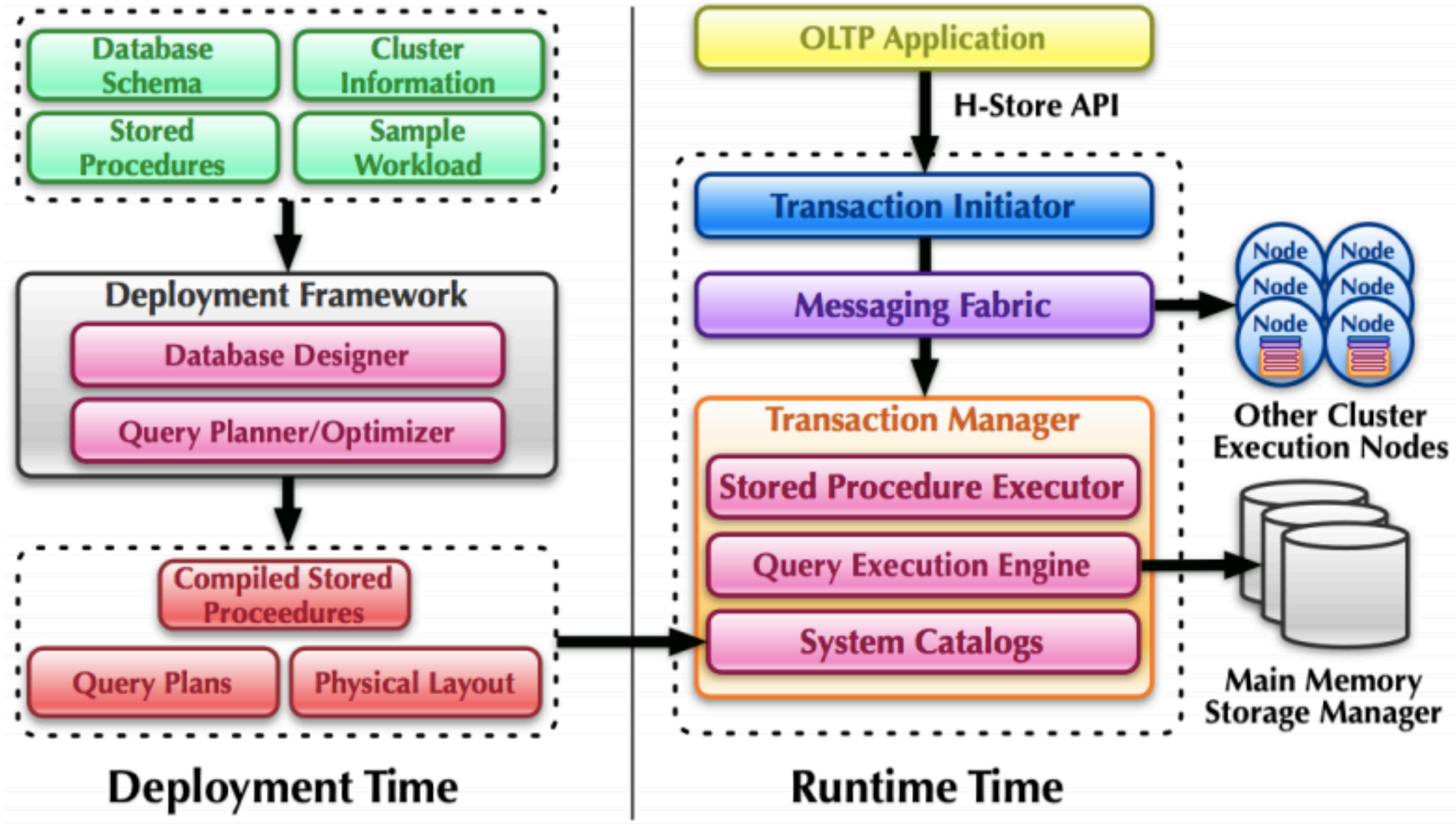
→ ***Exploit Pre-processing***

- Modern **Hardware**
  - Main memory often fits entire DB
  - Distributed systems common

→ ***Main memory DBMS***

→ ***Replication for fault tolerance***

H-Store Overview



H-Store: A High-Performance, Distributed Main Memory Transaction Processing System, R. Kallman et al.

Pre-Processing

- Transaction **templates** and example **workload** given
- Transactions are compiled as **stored procedures**
- Optimize physical **data layout**
  - **Replicate** frequently accessed tables at each site
  - Store joinable **partitions** of different tables together
  - ...

Concurrency Control

- Previously: motivation for **interleaving** transactions
  - **Long** transactions may block shorter ones
  - Overlap transactions waiting for **disk** access
- Now: **workloads** and **hardware** has changed
  - No **long running** transactions anymore
  - No wait for **disk** access since data in memory
- Hence, executing transactions **serially** works

Fault Tolerance

- Data is stored in memory and is **lost at system crash**
- **Replicate** data at different sites for failure tolerance
- **Cannot store copies** of entire tables on single site
- **Partition** table rows e.g. by primary key value
- **K-safety**: no k sites hold unique copies of any partition

Logging and Recovery

- We load data from **other site** after crash
- Hence, no need for any "**Redo**" logging
- May still need to **undo** transactions after abort
- Keep "Undo" log in **memory** until transaction finished
  - ***Why no need to make it persistent?***

One Site/One-Shot Transactions

- **One site** transaction: executes on one single site
  - Very efficient to execute
  - Can enable by good data partitioning
- **One shot** transaction: simple distributed transaction
  - Transaction initiator assigns transaction timestamp
  - Assume closely synchronized local clocks
  - Each node executes transactions in timestamp order

Two NewSQL Systems

- **H-Store** (VoltDB)
- **Google Spanner**

Spanner Overview

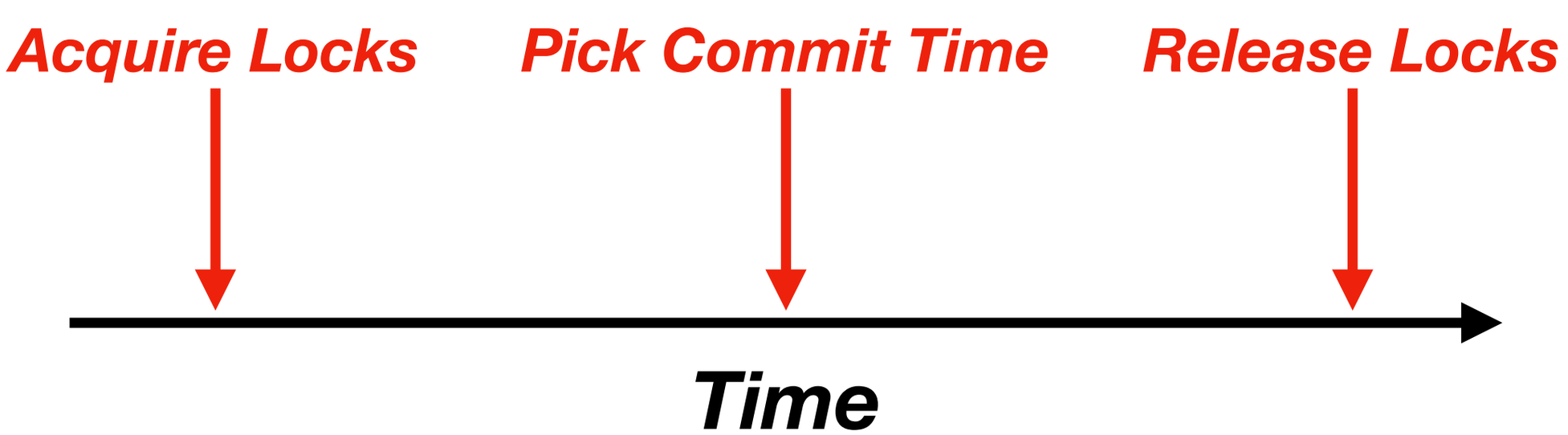
- A **NewSQL** database developed by Google
- Globally **distributed** database system
- Supports **SQL** queries and ACID transactions
- First to offer **external consistency** at global scale
  - T1 commits before T2 → T1 is serialized before T2
  - ***What's different from standard ACID transactions?***

External Consistency Motivation

- Assume database stores **bank accounts**
- Client **wires** money on an account (transaction 1)
- Then, client **checks** if money arrived (transaction 2)
- ***Why does external consistency matter here?***

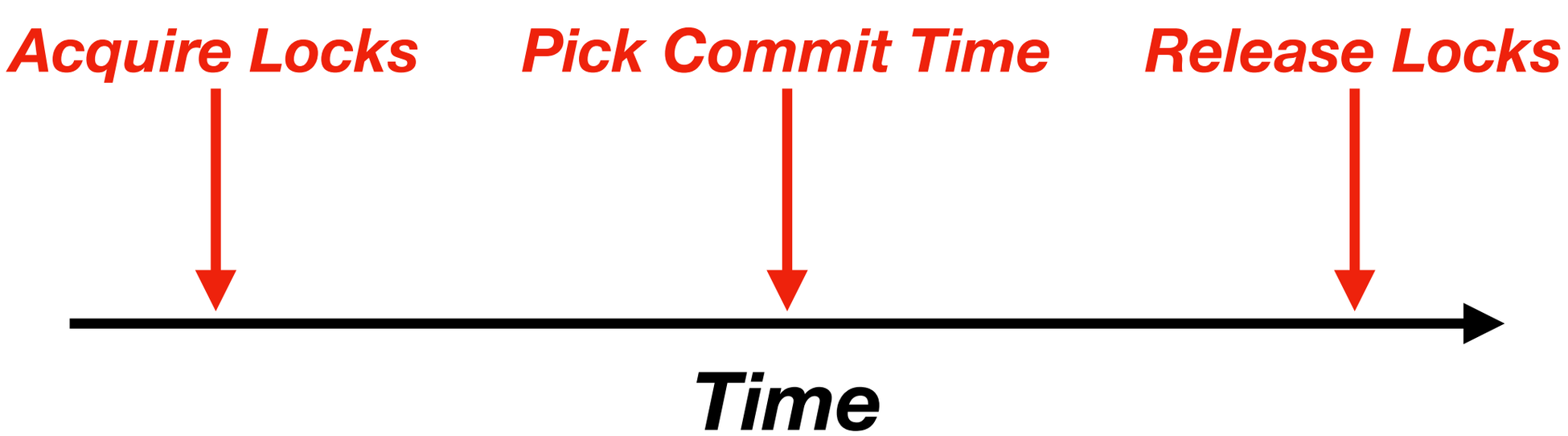
First Approach for External Consistency

Strict Two-Phase Locking Guarantees Exclusive Access



First Approach for External Consistency

Strict Two-Phase Locking Guarantees Exclusive Access

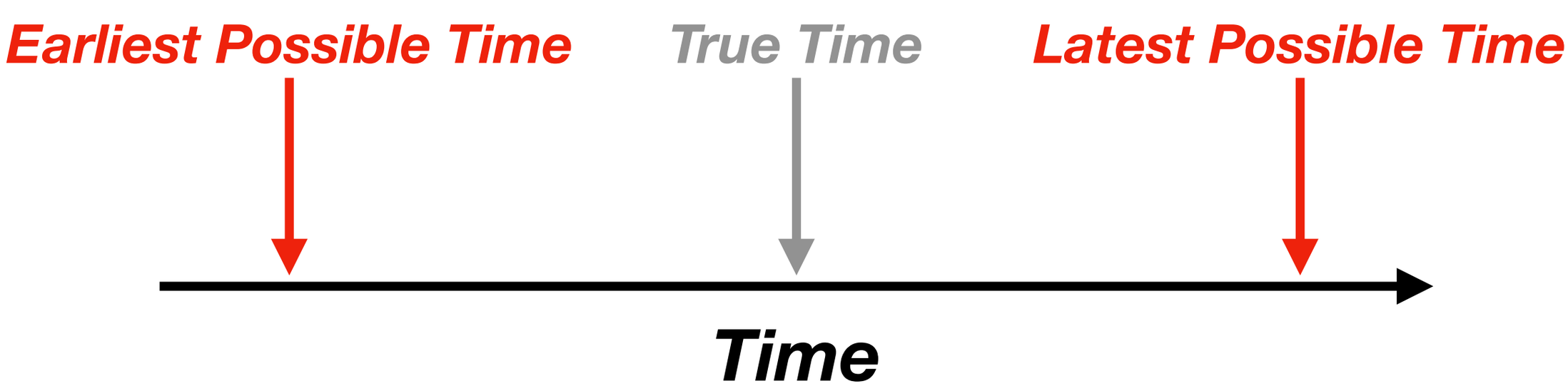


What is the Problem with Distributed Transactions ... ?

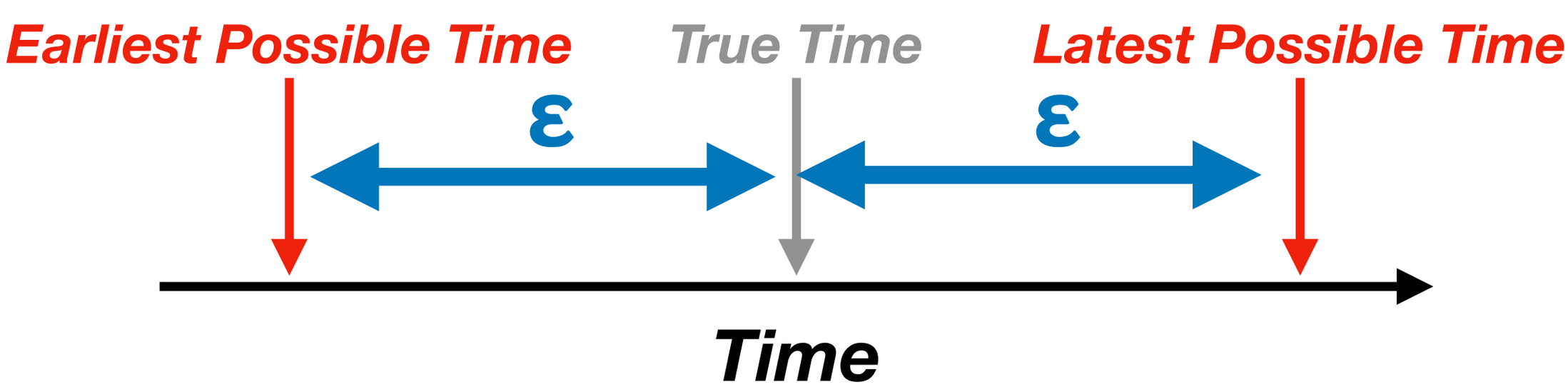
True Time API

- Clocks of different sites will not be perfectly in **sync**
- Google uses specialized **hardware** to increase precision
- Nevertheless, need to sync clocks **regularly**
- Uncertainty about time **grows** until next sync
- **True Time API** exposes uncertainty on time

True Time API



True Time API



True Time for External Consistency

- **Acquire locks** on objects
- Use **latest** possible time as commit timestamp
- Now **wait** until we are sure that this time has passed
  - Requires waiting for **2*ε** (interval of uncertainty)
- Now **release locks** on objects

External Consistency with TrueTime API