ETH zürich



VerIso: Verifiable Isolation Guarantees for Database Transactions

Shabnam Ghasemirad, Si Liu, Christoph Sprenger, Luca Multazzu, David Basin

Information Security Group, ETH Zürich

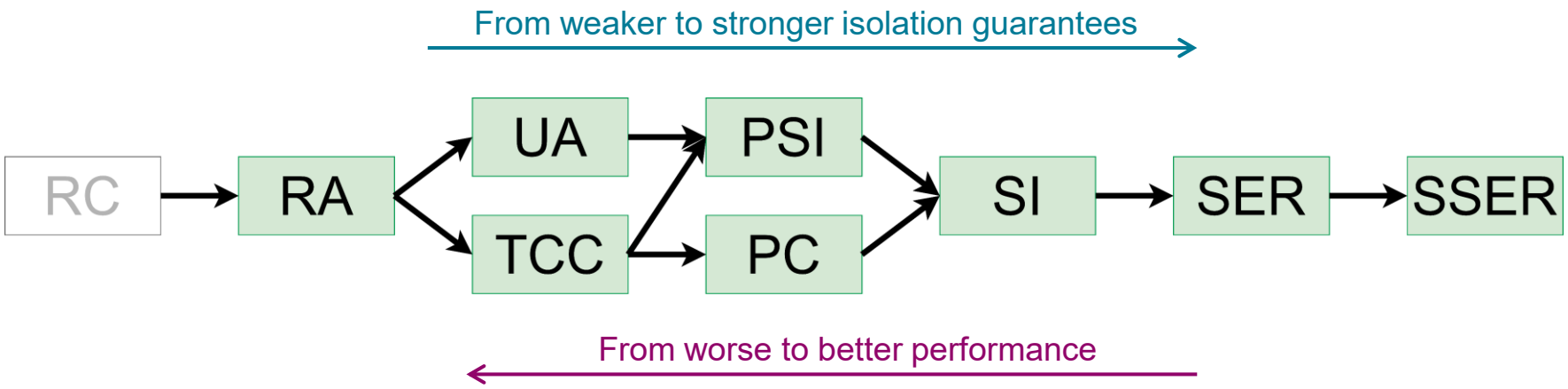
VLDB 2025, Transaction Management Session

Isolation Levels

Database = multiversioned key-value store

Concurrency control protocols → transaction **isolation**

Spectrum of isolation levels (IL):



Motivation

Isolation bugs in carefully designed and heavily tested databases

- Found in production databases
  - YugabyteDB: violates *SER*
  - MariaDB Galera Cluster: violates *SI*
  - TAPIR & DrTM: violate *SSER*
- Often due to **design-level** flaws, not implementation errors
- Costly to fix, hard to detect early



We need strong correctness guarantees at the design stage

Existing Approaches for Detecting Isolation Bugs

- Manual (pen-and-paper) proofs → error-prone
- Testing → limited coverage
- Model checking → small-scale only (limited #txns, #clients/servers)

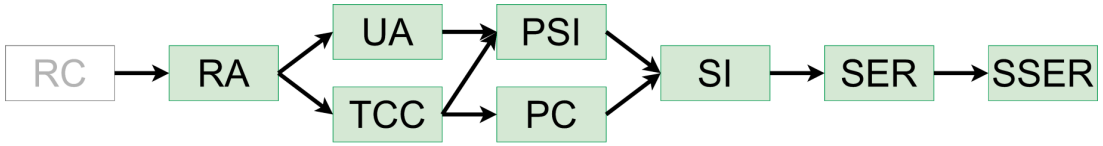


No *systematic*, *rigorous* framework to verify isolation guarantees for *all* system behaviours.

Contributions

We introduce **VerIso**

- The first mechanized framework for **formal (deductive) verification** of isolation guarantees in concurrency control protocols
- Supports a wide range of isolation levels
- **Verification** & **falsification** of protocol designs
  - Discovered an atomic visibility bug in TAPIR!

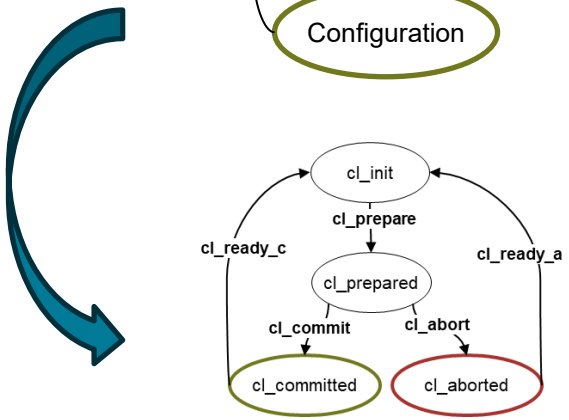




Protocol Verification

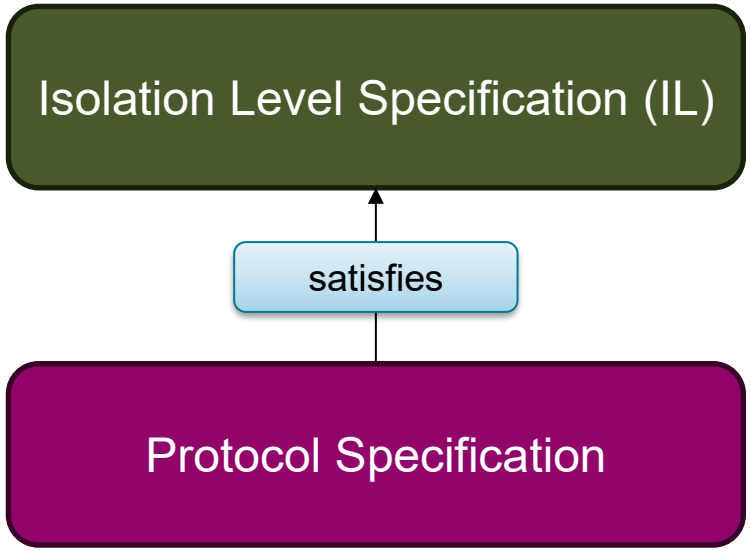
Protocol Verification Approach

Refinement



(Parameterized) Abstract Model

Concrete Model



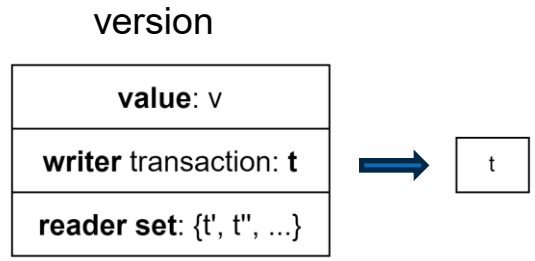
Refinement guarantees the protocol to only have allowed behaviours

Specification: labelled transition systems (LTS)

Abstract Model: Isolation Level

State (Configuration): (K, U)

- K: centralized multi-versioned key-value store
- U: client views, capturing the distributed aspect

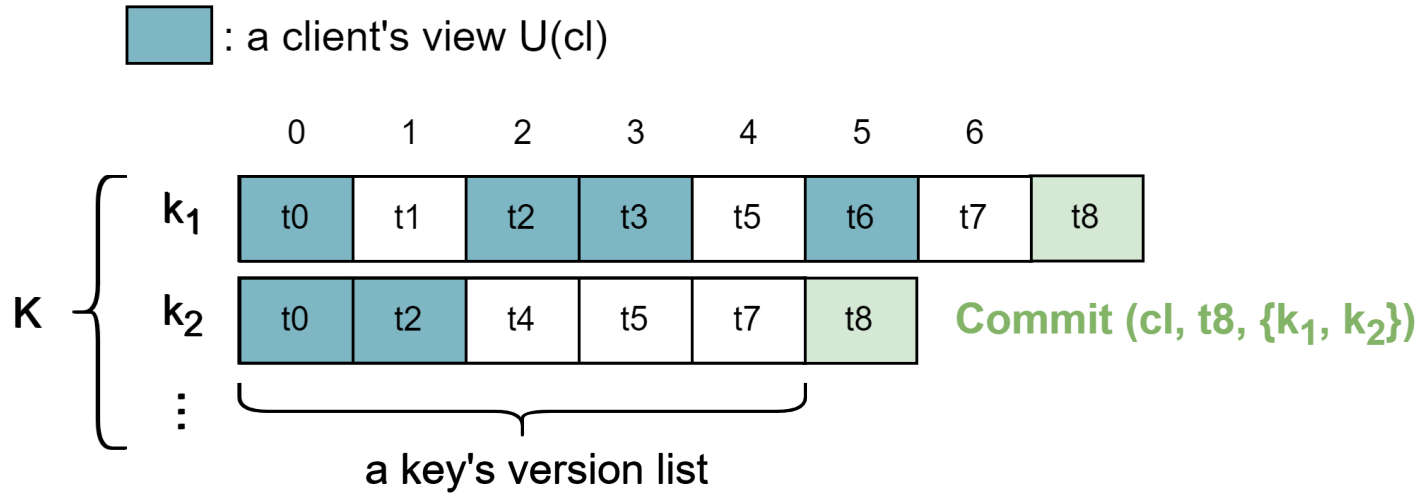


Transitions (Events):

Atomic Commit

$$(K, U) \xrightarrow{\text{commit}}_{IL} (K', U')$$

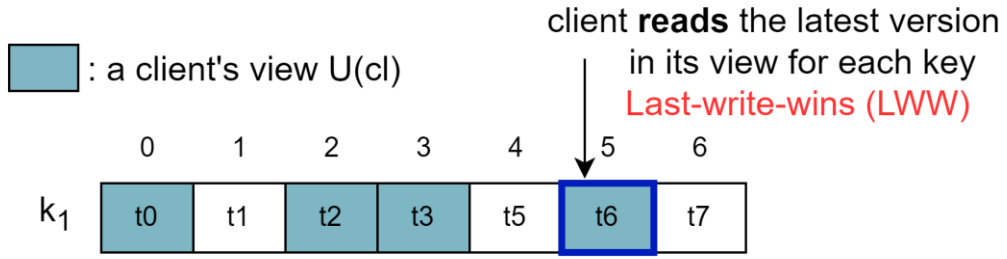
Parameterized!



Commit Conditions (parameterized)

$$(K, U) \xrightarrow{commit}_{IL} (K', U')$$

executes under these guards:



All ILs

1. View atomicity → baseline (RA)
2. Last-write-wins (LWW): client reads the newest version in its view

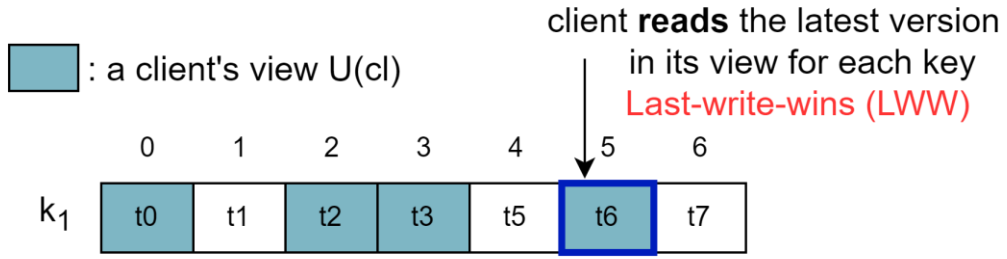
IL specific

3. canCommit$_{IL}$: commit condition parameterized on *IL*
4. Session guarantees$_{IL}$: client view guarantees parameterized on *IL*

Commit Conditions (example: TCC)

(K, U) $\xrightarrow{commit}_{TCC}$ (K', U')

executes under these guards:



All ILs

1. View atomicity → baseline (RA)
2. Last-write-wins (LWW): client reads the newest version in its view

IL specific

3. canCommit$_{TCC}$: view closed under **causal** dependencies (e.g. write-read)
4. Session guarantees$_{TCC}$: **monotonic-reads** and **read-your-writes**

Concrete Concurrency Control Protocol Verification

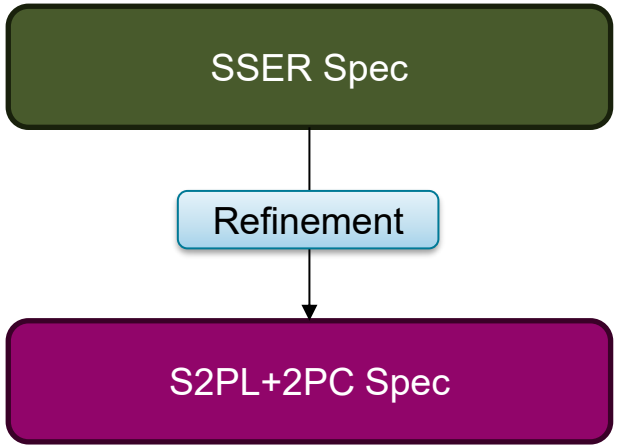
- We **instantiate** our parameterized framework to the **desired IL**

Case study:

Strict Two-Phase Locking (+ 2PC)

Desired IL: SSER

- $\text{canCommit}_{\textbf{SSER}}$: global view for all clients
- $\text{Session guarantees}_{\textbf{SSER}}$: none required

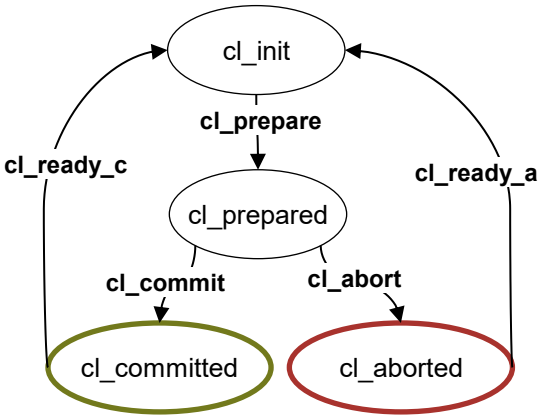


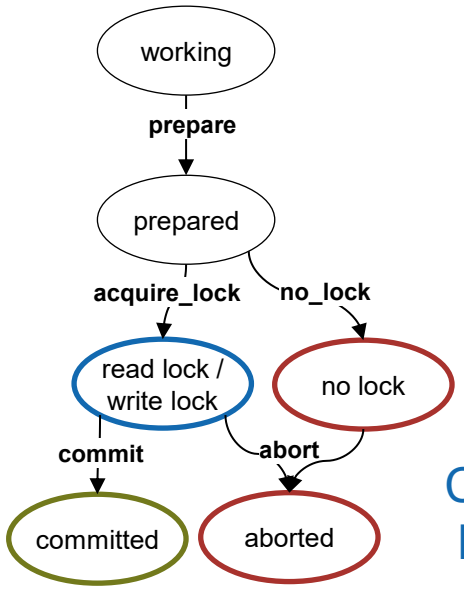
Case Study: S2PL+2PC – Protocol Model

Protocol Global State:

```
1  datatype txn_state =        -- transaction state
2    cl_init | cl_prepared | cl_committed | cl_aborted
3
4  record cl_conf =            -- client configuration
5    cl_state : txn_state
6    cl_sn : nat
7
8  datatype ver_state =        -- version state
9    working | prepared | read_lock | write_lock |
10   not_okay | committed | aborted
11
12 record svr_conf =           -- server configuration
13   svr_state : txid → ver_state
14   svr_vl : list(version)
15   svr_fp : txid × {R,W} → value
16
17 record global_conf =        -- global configuration
18   cls : cl_id → cl_conf
19   svrs : key → svr_conf
```

cl_commit refines the abstract atomic commit





✓ Refinement Successful

Clients' global view ensured by the locking mechanism

Other protocols verified in VerIso

Published in TACAS 2025 (part of ETAPS)

- **Eiger-PORT+ :** our novel cc protocol
  - ➢ *Stronger isolation* guarantee:

    refutes an open **conjecture**
  - ➢ *Superior performance:*

    2.5x throughput over state-of-the-art
- **Protocol verification in *VerIso***
  using Refinement and Reduction



Pushing the Limit: Verified Performance-Optimal Causally-Consistent Database Transactions

Shabnam Ghasemirad(✉), Christoph Sprenger, Si Liu, Luca Multazzu, and David Basin

ETH Zurich, Zurich, Switzerland
shabnam.ghasemirad@inf.ethz.ch

Abstract. Modern web services crucially rely on high-performance distributed databases, where concurrent transactions are isolated from each other using concurrency control protocols. Relaxed isolation levels, which permit more complex concurrent behaviors than strong levels like serializability, are used in practice for higher performance and availability. In this paper, we present Eiger-PORT+, a concurrency control protocol that achieves a strong form of causal consistency, called TCCv (Transactional Causal Consistency with convergence). We show that Eiger-PORT+ also provides performance-optimal read transactions in the presence of transactional writes, thus refuting an open conjecture that this is impossible for TCCv. We also deductively verify that Eiger-PORT+ satisfies this isolation level by refining an abstract model of transactions. This yields the first deductive verification of a complex concurrency control protocol. Furthermore, we conduct a performance evaluation showing Eiger-PORT+'s superior performance over the state-of-the-art.



Protocol Falsification

Protocol Falsification Approach

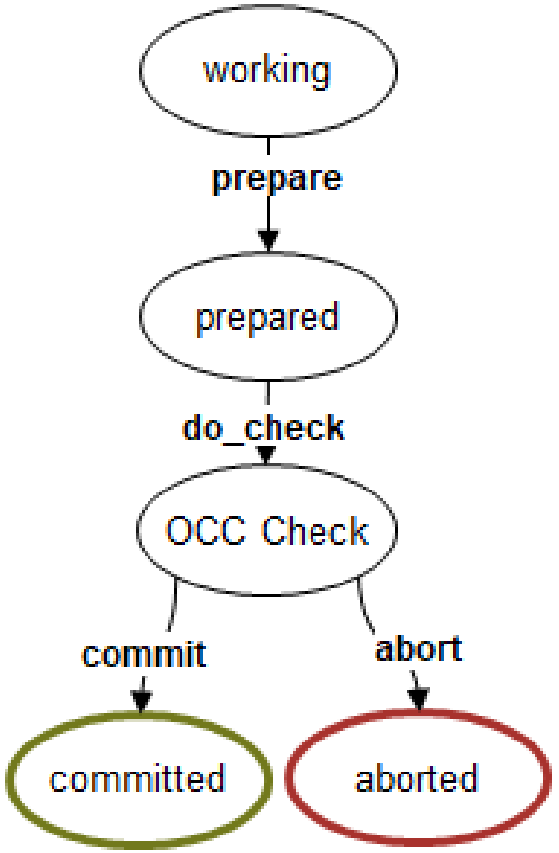
- Set up a **refinement proof in VerIso** for the **claimed IL**
- Encounter an **unprovable** proof obligation or required invariant

View Atomicity, Last-write-wins	**Atomic Visibility (baseline RA) violation**
Isolation Level (IL) specific conditions	**Violation of stronger isolation levels**

- Write down the scenario (**execution**) making the proof impossible
- Use the protocol model to validate the execution : **counterexample!**

Case Study: TAPIR

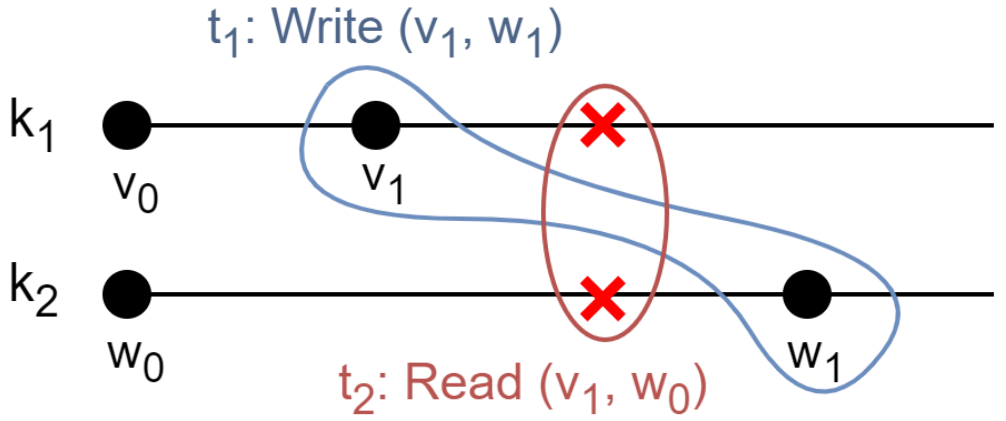
- TAPIR claims **SSER**
- 2PC + optimistic concurrency control (OCC)
  - Timestamp-based
  - OCC decision based on committed and pending reads and writes' timestamps



Case Study: TAPIR violates RA

- VerIso reveals an **atomic visibility** (RA) violation

➢ Client can read a transaction's update on one server but read an older version on another.



- Counterexample constructed from **failed Last-Write-Wins (LWW)** proof by assigning timestamps that would pass the OCC check
- The bug had escaped *manual proofs* and *model checking (TLA+)*

Conclusions

Conclusions

VerIso: The first mechanized framework for **formal verification** of **isolation guarantees** in concurrency control protocols.



Shabnam Ghasemirad
Information Security Group, ETH Zürich
shabnam.ghasemirad@inf.ethz.ch

Paper title:
VerIso: Verifiable Isolation Guarantees for Database Transactions

Authors:
Shabnam Ghasemirad, Si Liu, Christoph Sprenger, Luca Multazzu, David Basin