# Efficient Black-box Checking of Snapshot Isolation in Databases

(Conference VLDB'2024)

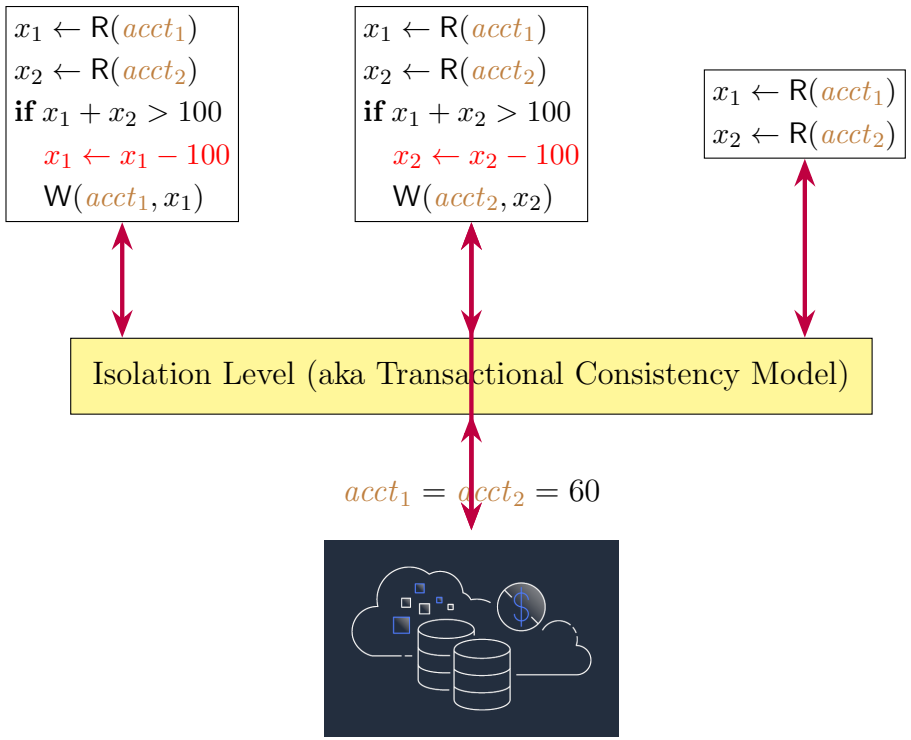
Hengfeng Wei

hfwei@nju.edu.cn

August 22, 2023


# Transaction and Isolation Level

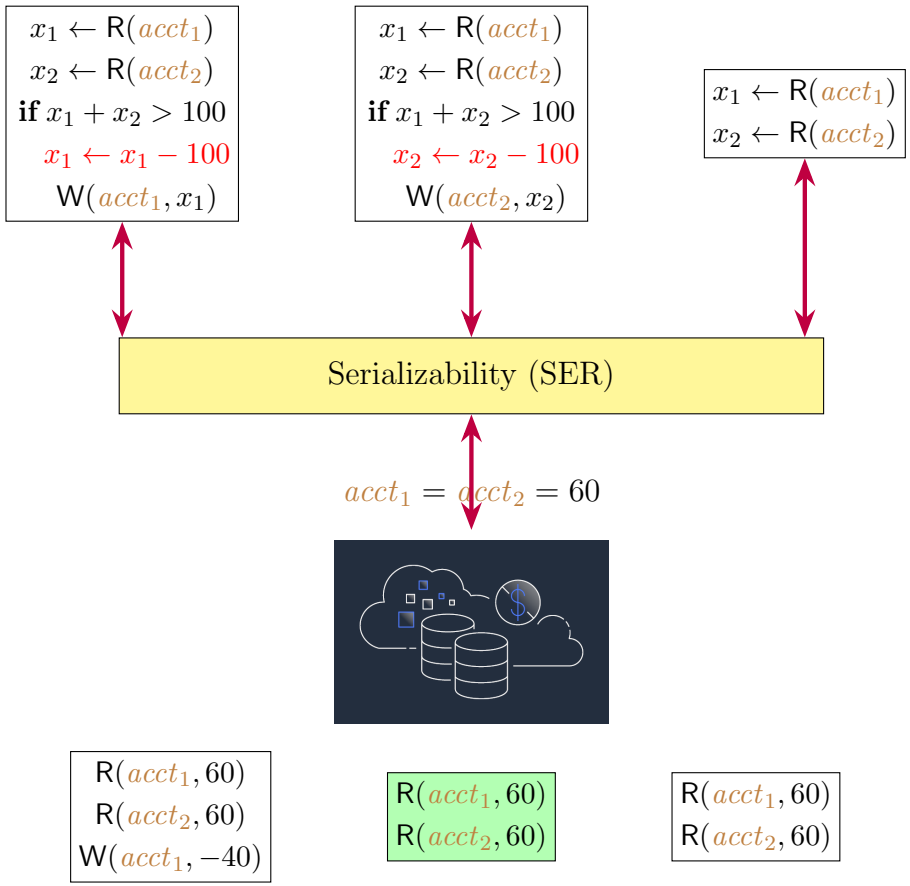
A transaction is a *group* of operations that is executed atomically.




The isolation levels specify how they are isolated from each other.

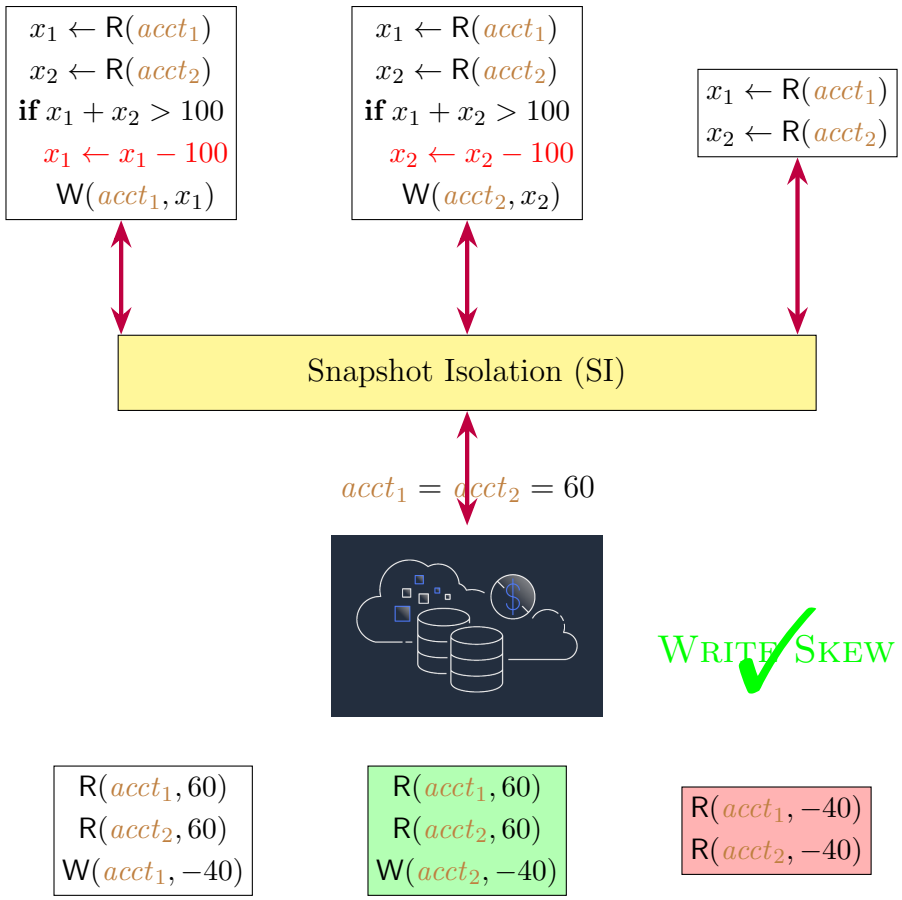
# Serializability (SER)

All transactions appear to execute in some total order.



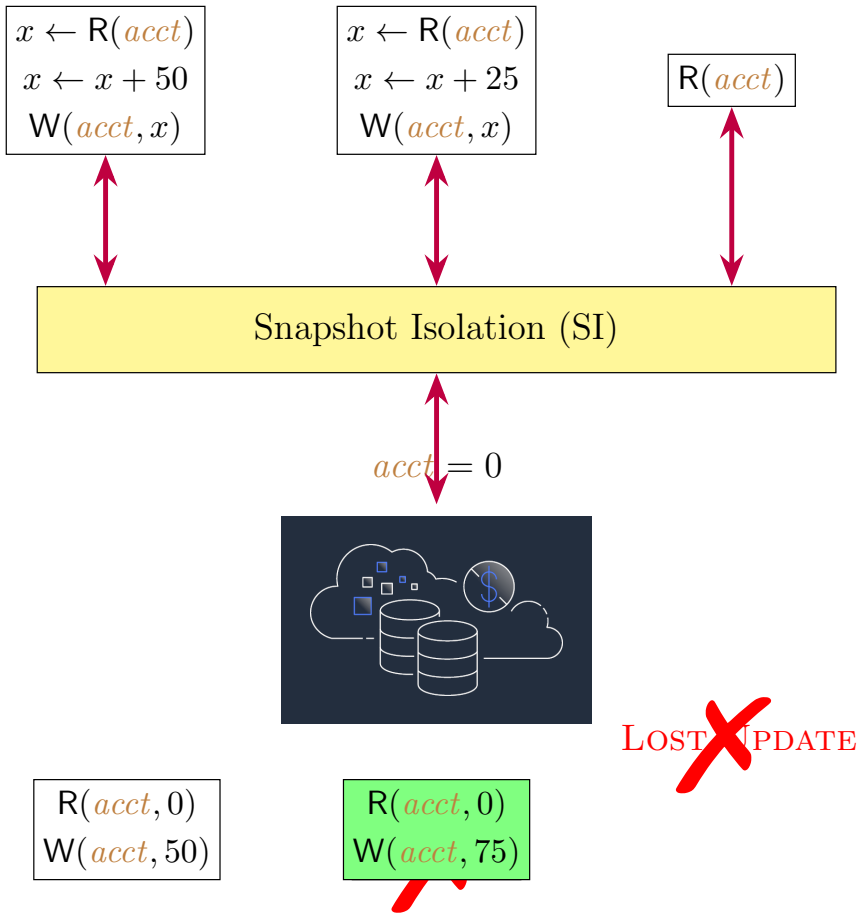

too expensive, especially for distributed transactions

# Snapshot Isolation (SI)




Snapshot Read: Each transaction reads data from a *snapshot* of committed data valid as of the (logical) time the transaction started.

# Snapshot Isolation (SI)




Snapshot Write: Concurrent transactions cannot write to the same key. One of them must be aborted.

# Snapshot Isolation (SI)


Snapshot Isolation (SI)

Causality Violation

W($x$, *post*)

R($x$, *post*)
W($y$, *comment*)

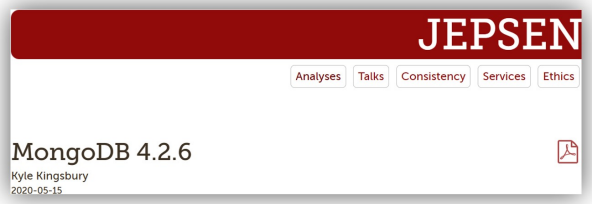
R($x$, *empty*)
R($y$, *comment*)

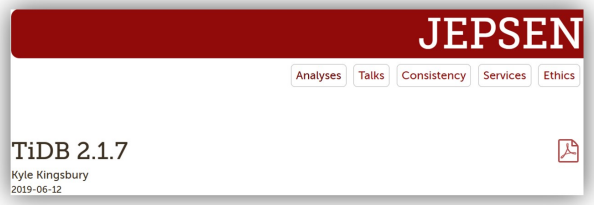
# Database systems and Snapshot Isolation

Many database systems choose to support SI.

# Database Systems and Snapshot Isolation

Database systems may fail to provide SI correctly as they claim.

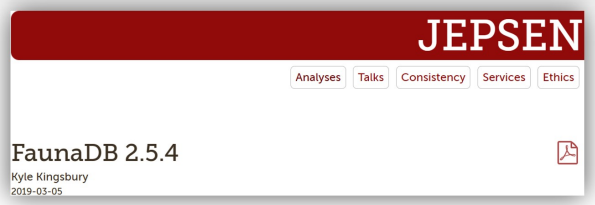


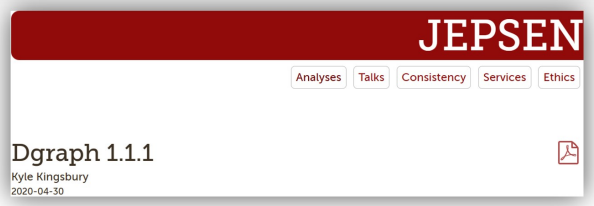




**Elle: Inferring Isolation Anomalies from Experimental Observations**

Kyle Kingsbury
Jepsen
aphyr@jepsen.io

Peter Alvaro
UC Santa Cruz
palvaro@ucsc.edu

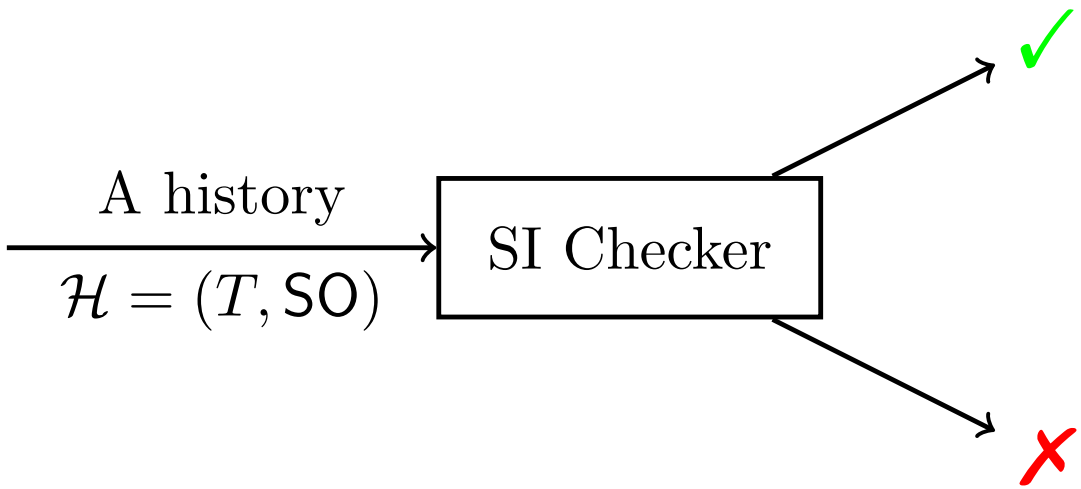


# The SI Checking Problem

**Definition (The SI Checking Problem)**

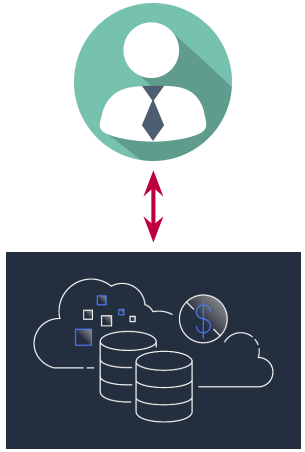
The SI checking problem is the decision problem of determing whether a given history $\mathcal{H} = (T, \mathsf{SO})$ satisfies SI?

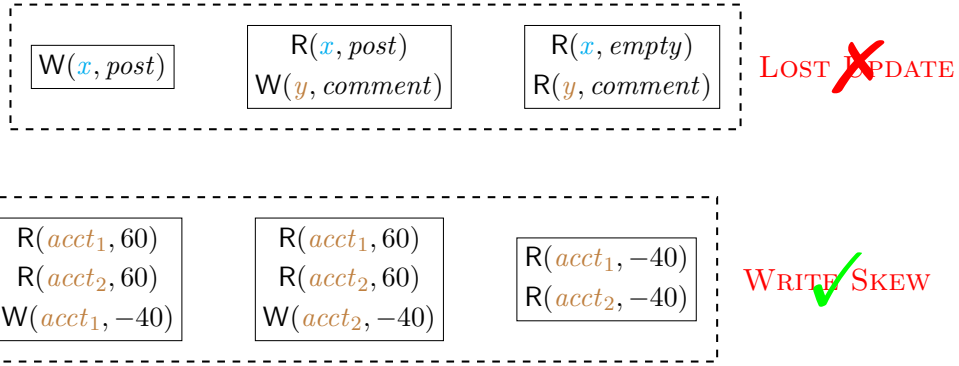



$\mathsf{SO}$ : *session order* among the set $T$ of transactions

# The SI Checking Problem

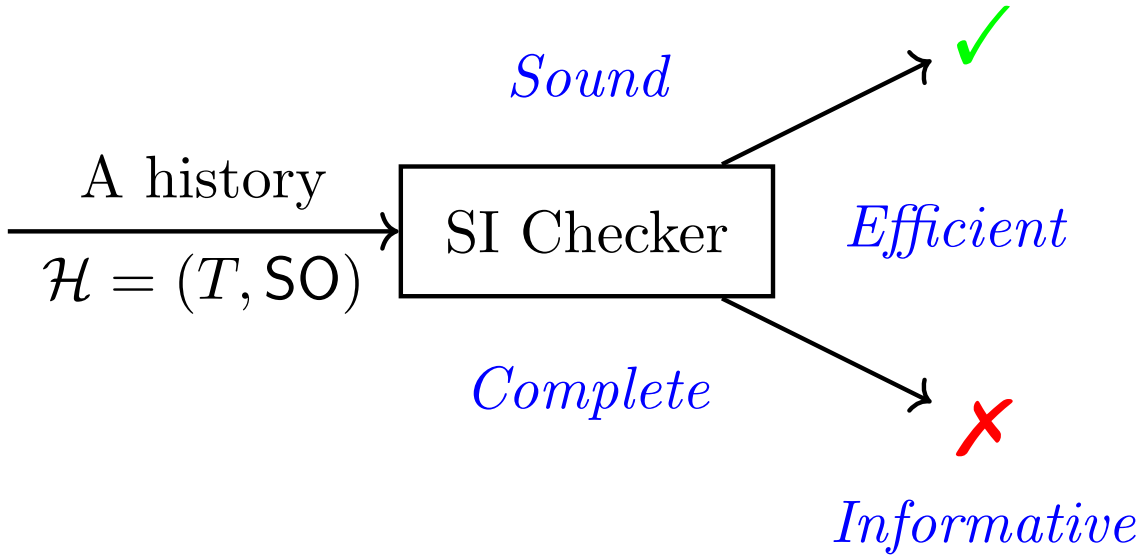
*Black-box checking:* do not rely on database internals






The histories are collected from database logs.

# The SI Checking Problem




*Sound:* If the checker says ✗, then the history does *not* satisfy SI.
*Complete:* If the checker says ✓, then the history *satisfies* SI.
*Efficient:* The checker should *scale* up to large workloads.
*Informative:* The checker should provide understandable *counterexamples* if it says ✗.

# Related Work

dbcop [**Complexity:OOPSLA2019**] checker for SI

not practically efficient;

not informative, only "False" upon violations

Elle [**Elle:VLDB2020**] checker for various isolation levels

SI checking based on [**Adya:PhDThesis1999**] relies on start/commit timestamps

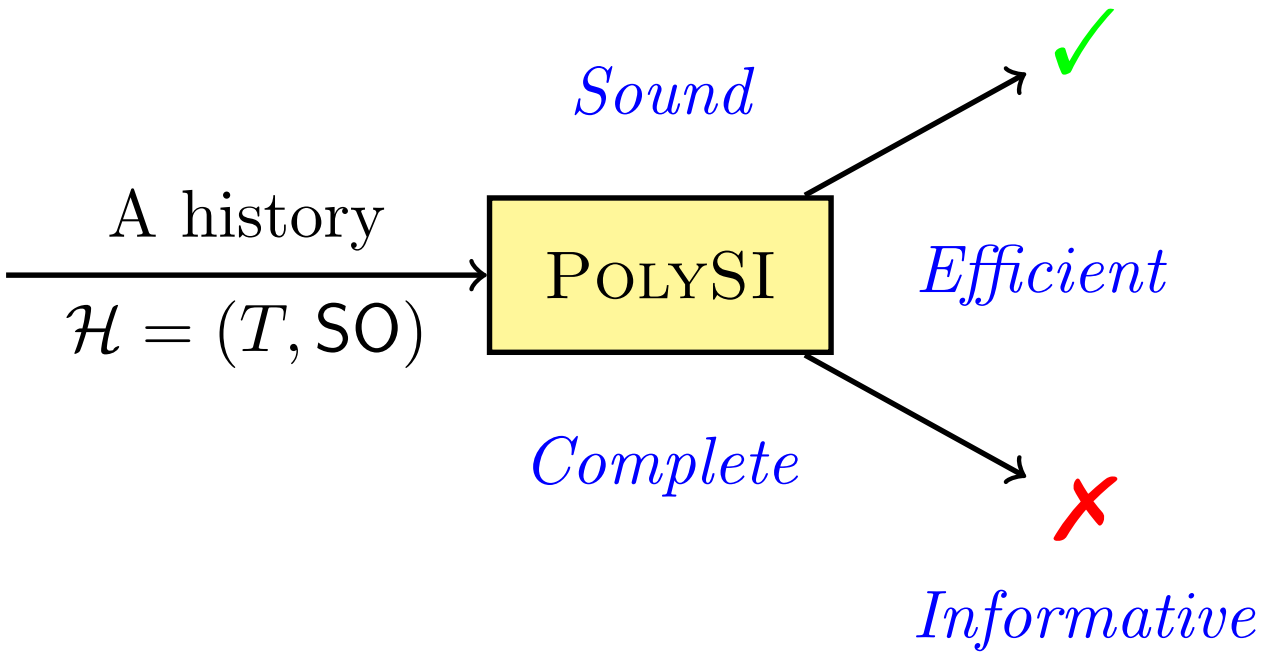
SI checking based on [**AnalysingSI:JACM2018**] is unsound for efficiency reasons [a]

Cobra [**Cobra:OSDI2020**] checker for SER

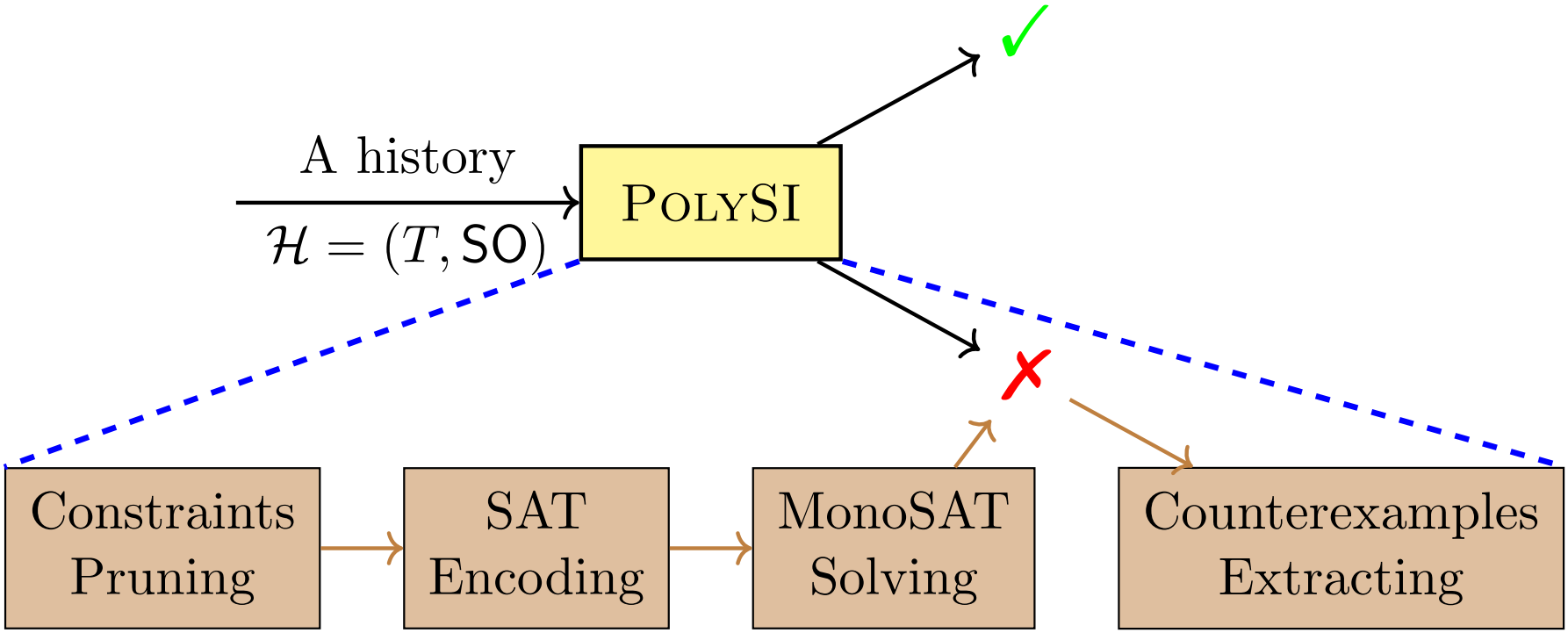
SI checking is *harder* than SER checking

[a] https://github.com/jepsen-io/elle/issues/17; Fixed now.

# Contribution: the PolySI Checker



# Contribution: the PolySI Checker



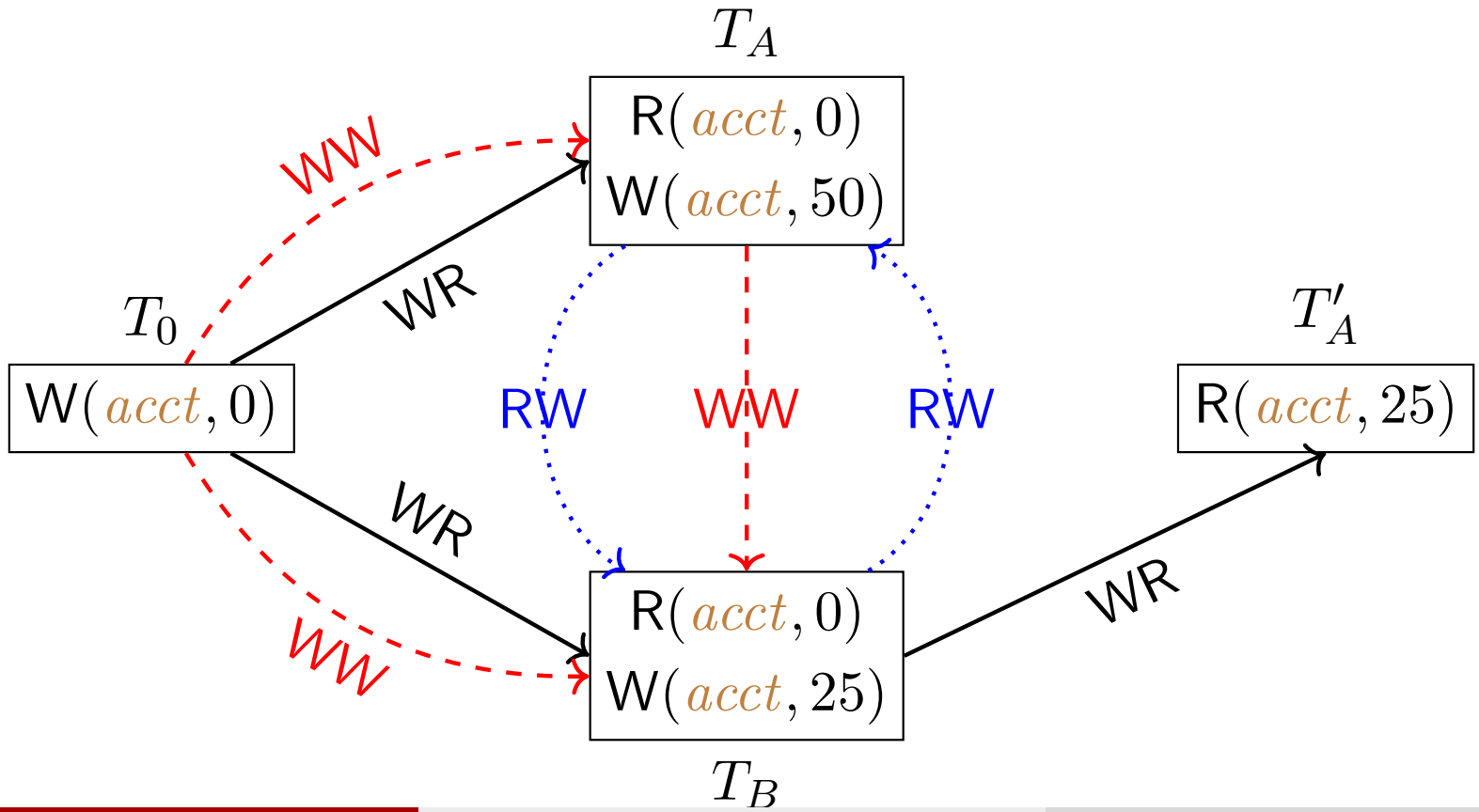

*Sound & Complete:* polygraph-based characterization of SI
*Efficient:* utilizing MonoSAT solver optimized for graph problems
*Efficient:* domain-specific pruning before encoding *Informative:* extract counterexamples from the unsatisifiable core

# Dependency Graph-based Characterization of SI

$$\textit{Suppose that } T_A \xrightarrow{\mathsf{WW}} T_B \quad T_0 \xrightarrow{\mathsf{WR}} T_A \wedge T_0 \xrightarrow{\mathsf{WW}} T_B \implies T_A \xrightarrow{\mathsf{RW}} T_B$$

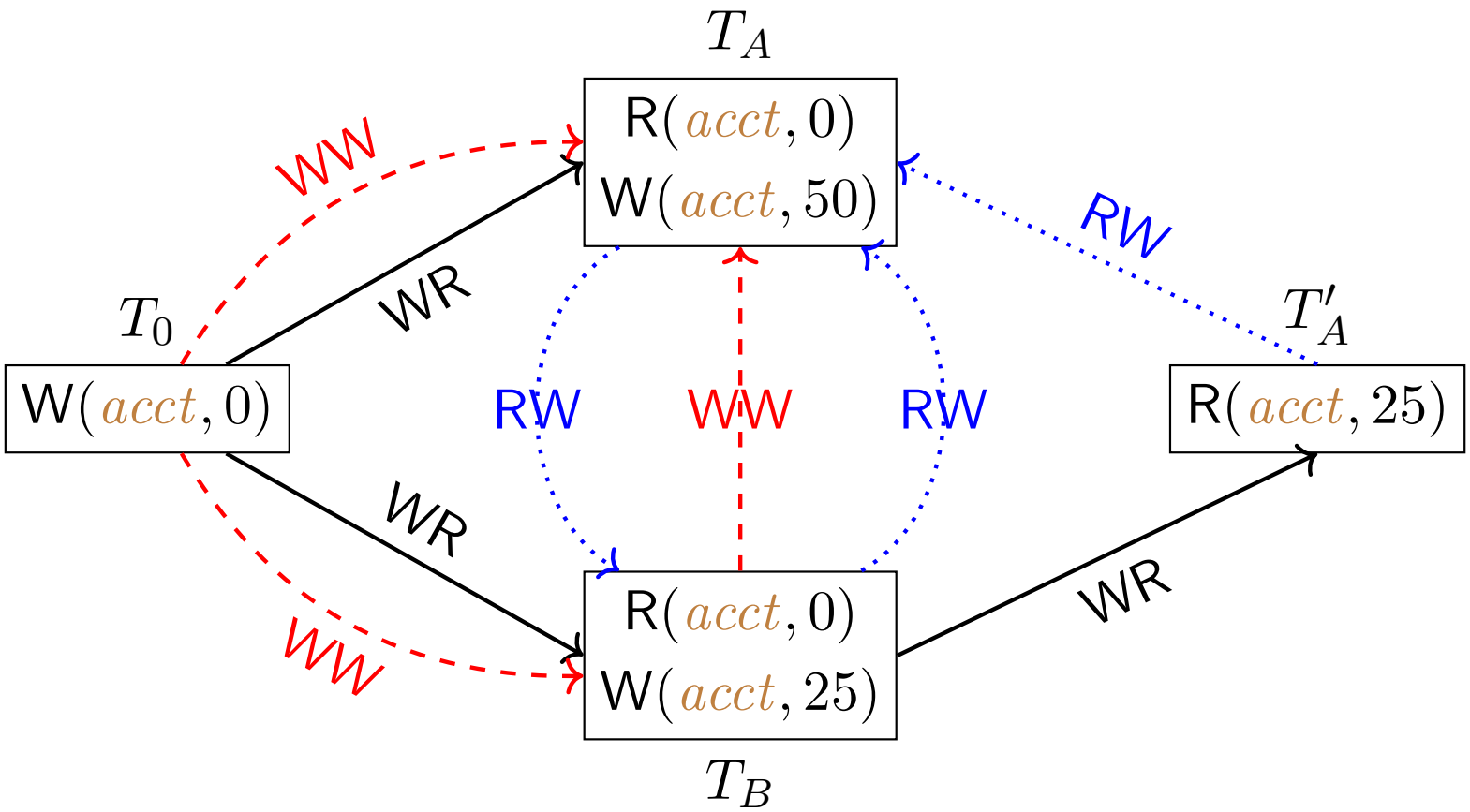
$$T_0 \xrightarrow{\mathsf{WR}} T_B \wedge T_0 \xrightarrow{\mathsf{WW}} T_A \implies T_A \xrightarrow{\mathsf{RW}} T_A$$

$$\textit{Suppose that } T_A \xrightarrow{\mathsf{WW}} T_B$$



# Dependency Graph-based Characterization of SI

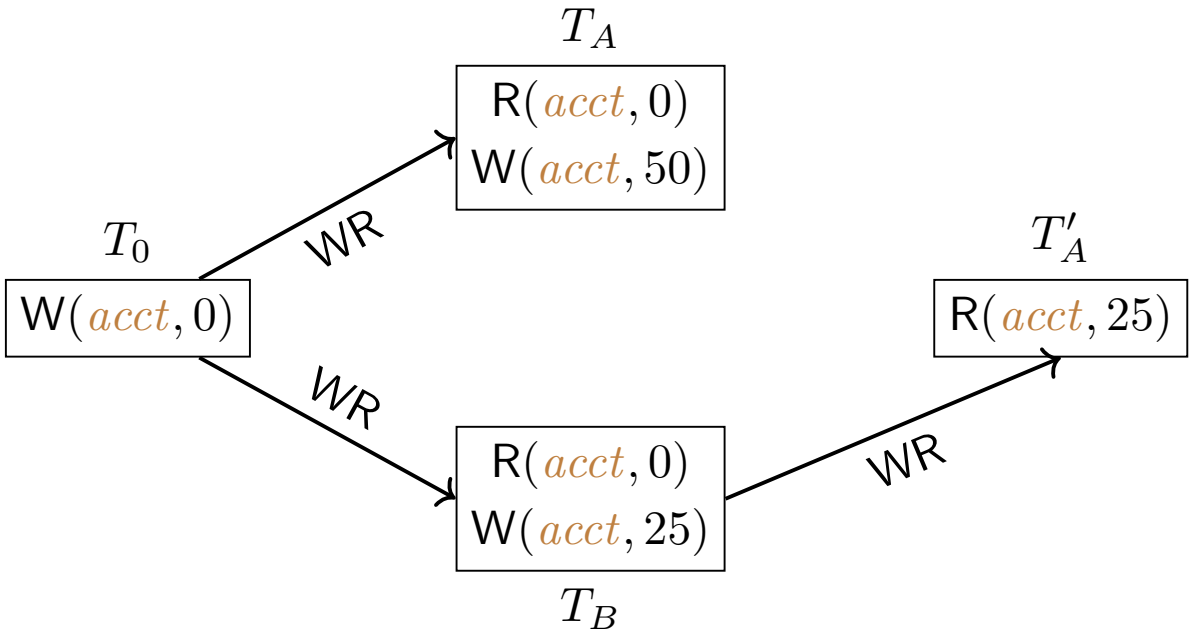
*Suppose that* $T_B \xrightarrow{\mathsf{WW}} T_A$




undesired cycle: $T_B \xrightarrow{\mathsf{WW}} T_A \xrightarrow{\mathsf{RW}} T_B$

# Dependency Graph-based Characterization of SI

We have considered both bases $T_A \xrightarrow{\mathsf{WW}} T_B$ and $T_B \xrightarrow{\mathsf{WW}} T_A$.




Either case leads to an undesired cycle.

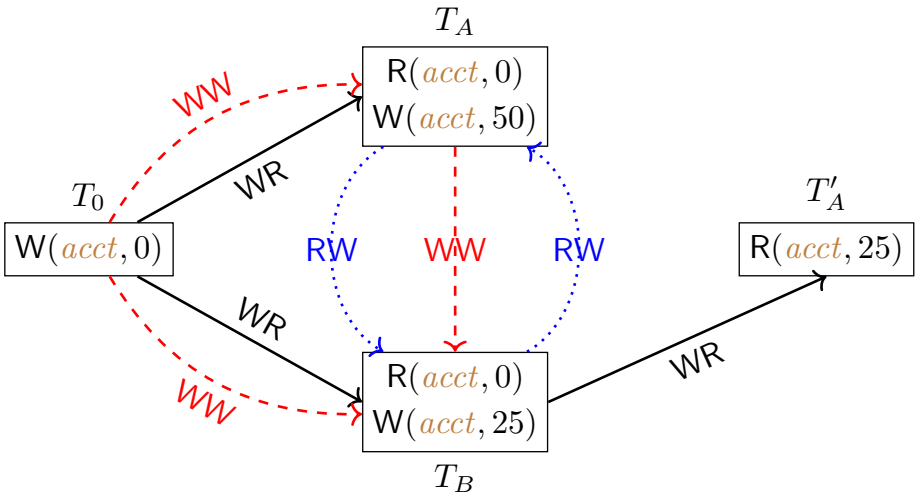
Therefore, it does not satisfy SI.

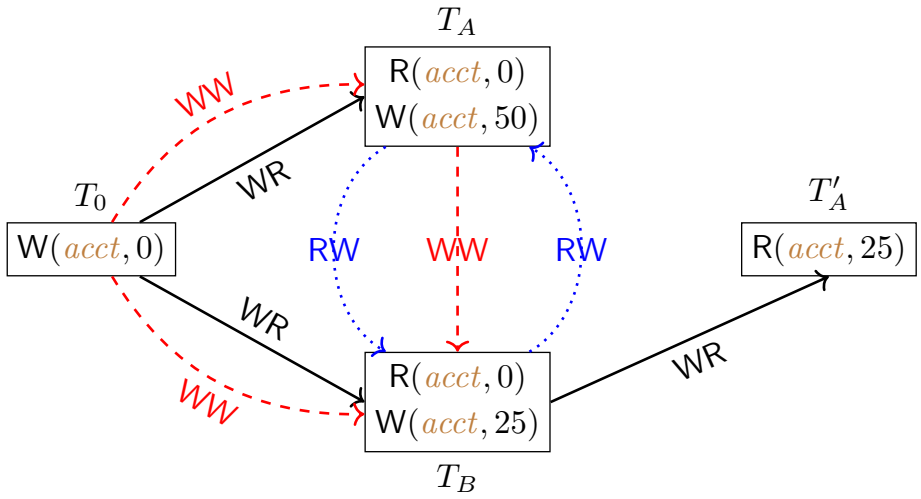
# Dependency Graph-based Characterization of SI

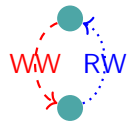
Theorem (Theorem 4.1 of [**AnalysingSI:JACM2018**])

*Informally, a history satisfies SI if only if*
*there exists a dependency graph for it that contains*
*only cycles (if any) with at least two adjacent* RW *edges.*

# Dependency Graph-based Characterization of SI




Every possible dependency graph contains an undesired 

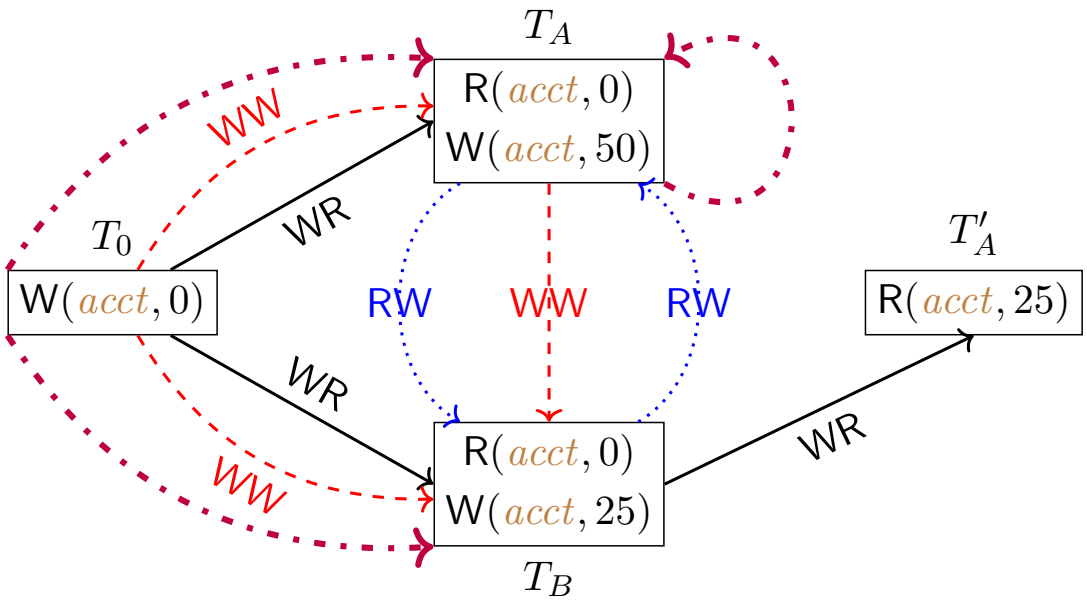
cycle.

# Dependency Graph-based Characterization of SI

**Theorem (Theorem 4.1 of [AnalysingSI:JACM2018])**

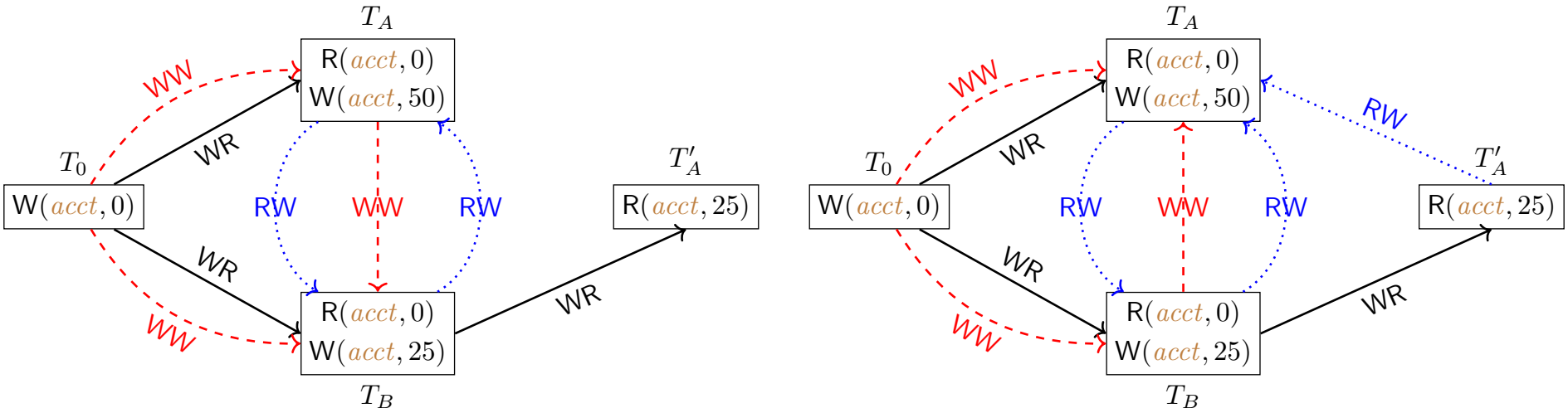
*For a history* $\mathcal{H} = (T, \mathsf{SO})$,

$$\mathcal{H} \models \text{SI} \iff \mathcal{H} \models \textsc{Int} \wedge$$
$$\exists\, \mathsf{WR}, \mathsf{WW}, \mathsf{RW}.\ \mathcal{G} = (\mathcal{H}, \mathsf{WR}, \mathsf{WW}, \mathsf{RW}) \wedge$$
$$((( \mathsf{SO}_{\mathcal{G}} \cup \mathsf{WR}_{\mathcal{G}} \cup \mathsf{WW}_{\mathcal{G}}) \,;\, \mathsf{RW}_{\mathcal{G}}?) \ \textit{is acyclic}).$$



# Dependency Graph-based Characterization of SI

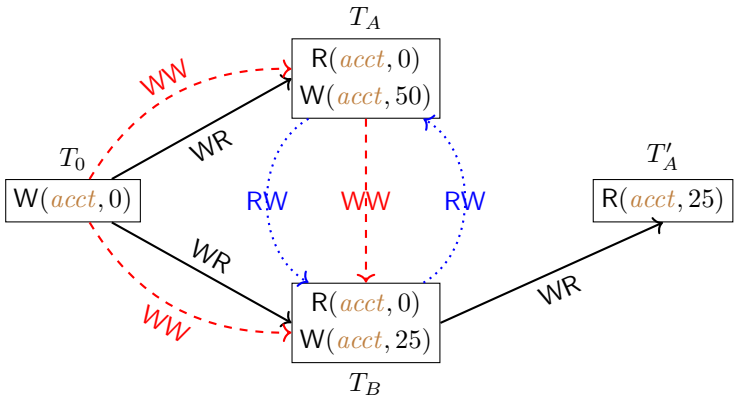
$\mathcal{Q}$ : How to capture and resolve all possible WW dependencies?

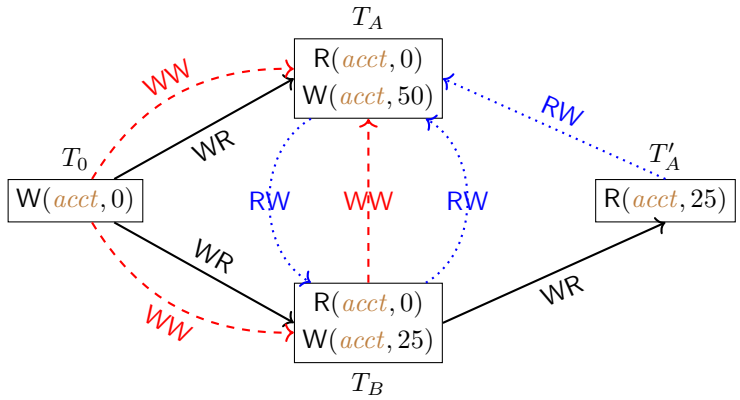


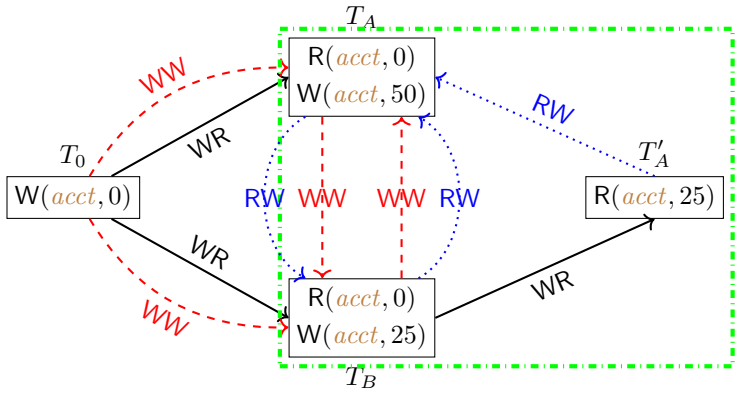

$\mathcal{A}$ : encode them into SAT formulas based on (generalized) polygraphs and solve them using SAT solvers.

# Polygraphs: A Family of Dependency Graphs

Consider the two cases of WW dependencies between $T_A$ and $T_B$.



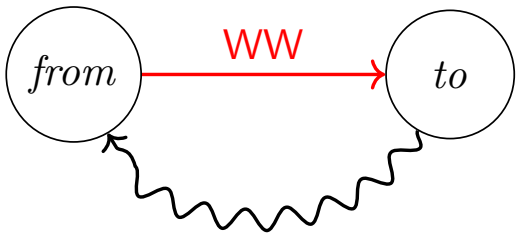



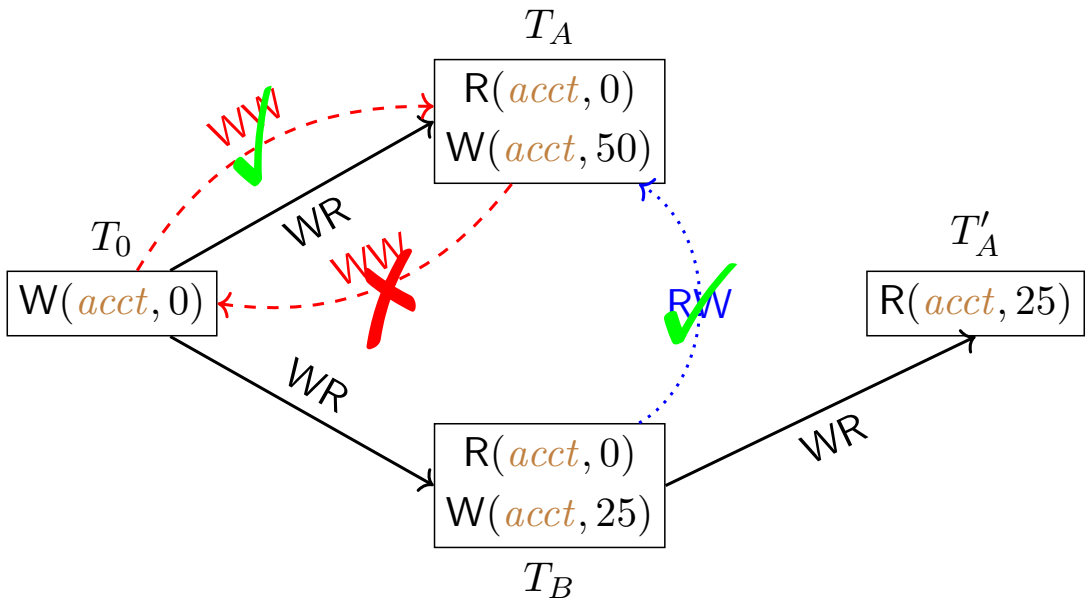




generalized polygraph:

$$\langle either \triangleq \{T_A \xrightarrow{\text{WW}} T_B\}, or \triangleq \{T_B \xrightarrow{\text{WW}} T_A, T'_A \xrightarrow{\text{RW}} T_A\}\rangle$$

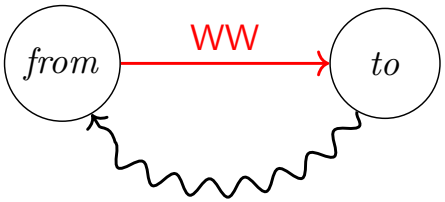
# PolySI: Pruning before Encoding (the WW case)

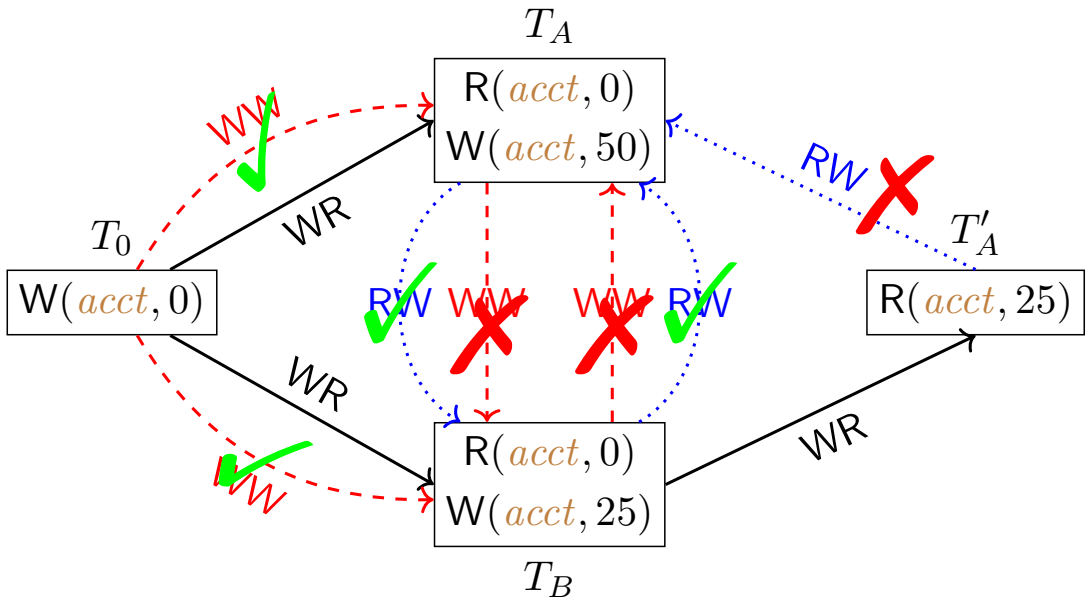



$T_A \xrightarrow{\text{WW}} T_0$ can be pruned due to the $T_A \xrightarrow{\text{WW}} T_0 \xrightarrow{\text{WR}} T_A$ cycle.

# PolySI: Pruning before Encoding (the WW case)



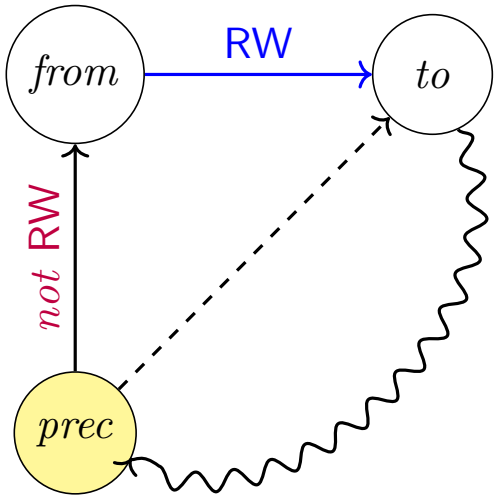




$T_A \xrightarrow{\text{WW}} T_B$ is pruned due to the $T_A \xrightarrow{\text{WW}} T_B \xrightarrow{\text{RW}} T_A$ cycle.

$T_B \xrightarrow{\text{WW}} T_A$ is pruned due to the $T_B \xrightarrow{\text{WW}} T_A \xrightarrow{\text{RW}} T_B$ cycle.

Therefore, we are sure that the history does *not* satisfy SI.

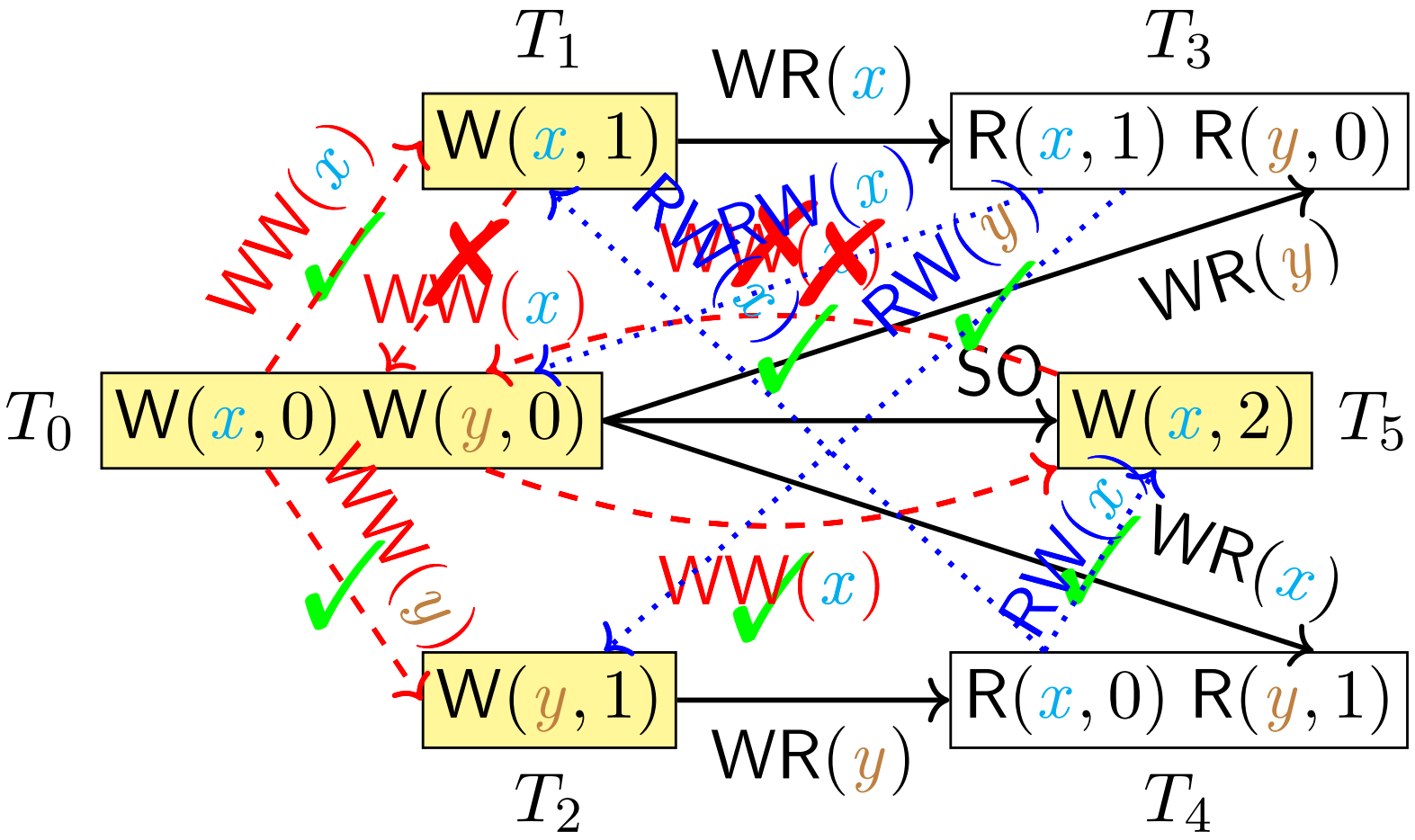
# PolySI: Pruning before Encoding (the RW case)




Theorem (Theorem 4.1 of [**AnalysingSI:JACM2018**])

*Informally, a history satisfies SI if only if there exists a dependency graph for it that contains only cycles (if any) with at least two adjacent* RW *edges.*

# PolySI: An Illustrating Example of "Long Fork"

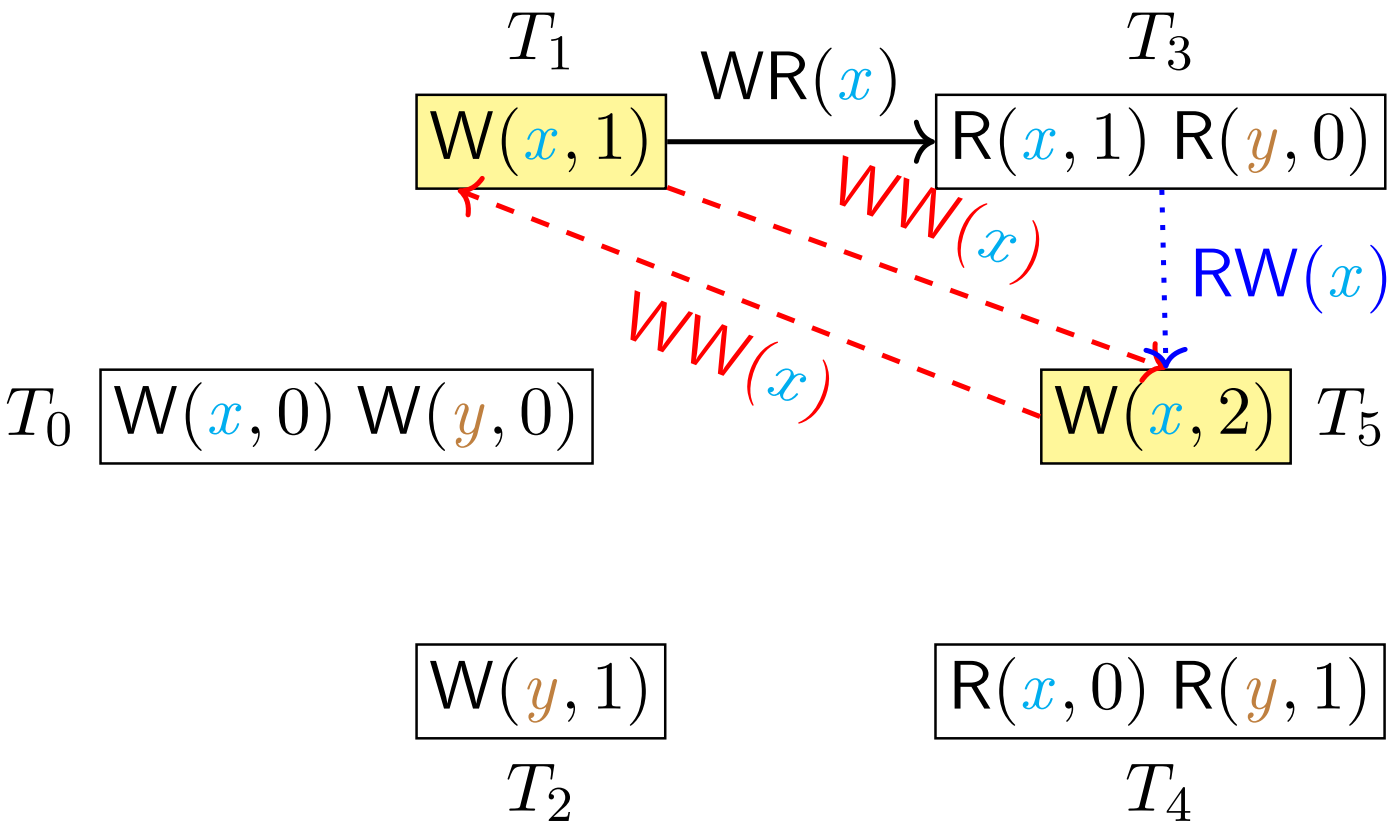



order between $T_0$, $T_1$, and $T_5$ (on $x$) and between $T_0$ and $T_2$ (on $y$)

The $T_5 \xrightarrow{\mathsf{WW}(x)} T_0$ case is pruned due to $T_0 \xrightarrow{\mathsf{SO}} T_5 \xrightarrow{\mathsf{WW}(x)} T_0$.

# PolySI: An Illustrating Example of "Long Fork"

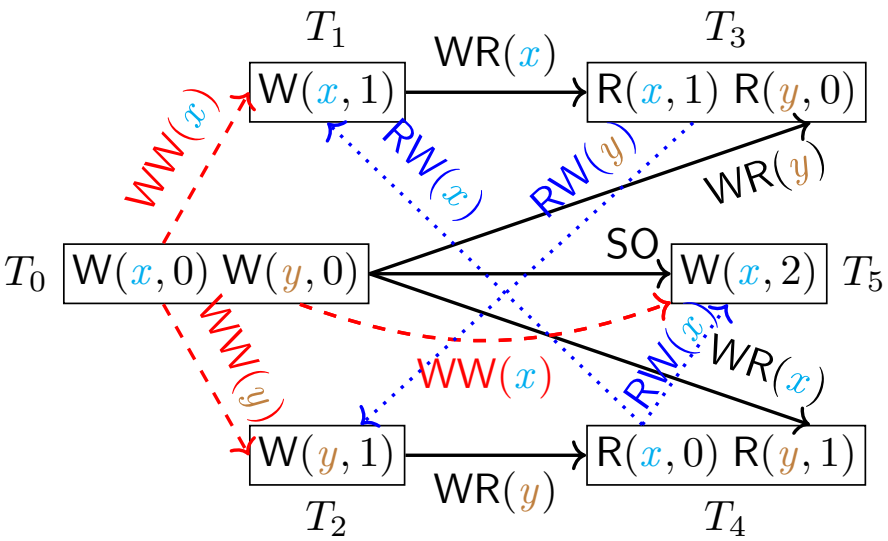
$$\langle either = \{T_1 \xrightarrow{\mathsf{WW}(x)} T_5, T_3 \xrightarrow{\mathsf{RW}(x)} T_5\}, or = \{T_5 \xrightarrow{\mathsf{WW}(x)} T_1\}\rangle$$

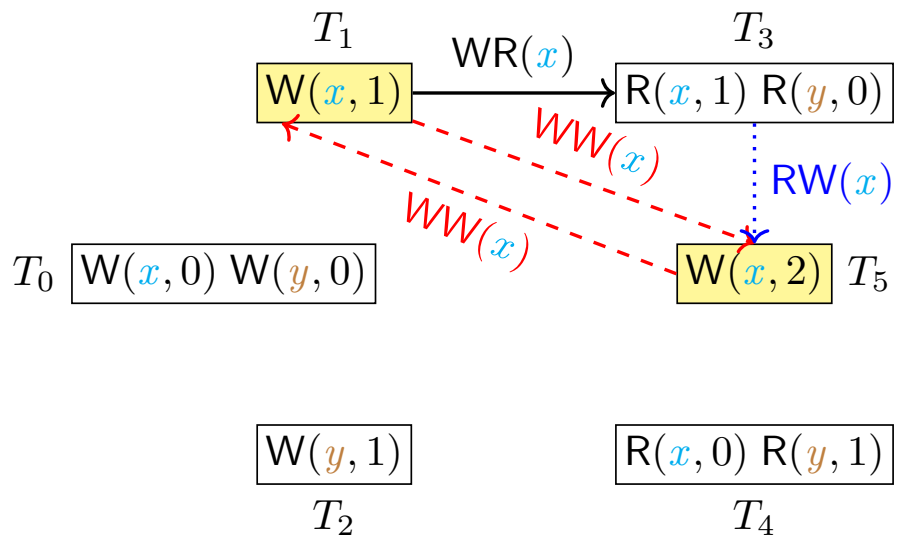



$$(\mathsf{BV}_{1,5} \wedge \mathsf{BV}_{3,5} \wedge \neg\mathsf{BV}_{5,1}) \vee (\mathsf{BV}_{5,1} \wedge \neg\mathsf{BV}_{1,5} \wedge \neg\mathsf{BV}_{3,5})$$

# PolySI: An Illustrating Example of "Long Fork"

$((\mathsf{SO}_{\mathcal{G}} \cup \mathsf{WR}_{\mathcal{G}} \cup \mathsf{WW}_{\mathcal{G}}) \; ; \; \mathsf{RW}_{\mathcal{G}}?)$ *is acyclic.*




We need to encode the "composition ( ; )" of dependency edges.

$$T_1 \xrightarrow{\mathsf{WR}} T_3 \xrightarrow{\mathsf{RW}} T_2 : \; \mathsf{BV}^{I}_{1,2} = \mathsf{BV}_{1,3} \wedge \mathsf{BV}_{3,2} \quad (I \text{ for the induced graph})$$

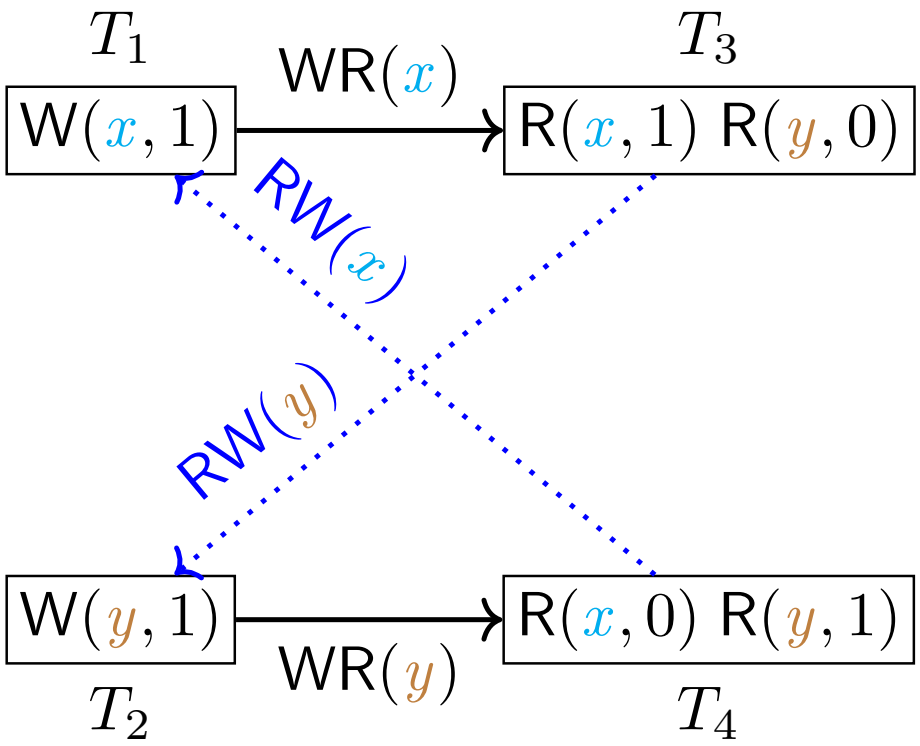
$$T_1 \xrightarrow{\mathsf{WR}} T_3 \xrightarrow{\mathsf{RW}} T_5 : \; \mathsf{BV}^{I}_{1,5} = \mathsf{BV}_{1,3} \wedge \mathsf{BV}_{3,5} \quad (I \text{ for the induced graph})$$

# PolySI: An Illustrating Example of "Long Fork"

Feed the SAT formula into the MonoSAT solver [**MonoSAT:AAAI2015**] optimized for *cycle detection*


Assert that the induced graph $I$ is acyclic.

# PolySI: An Illustrating Example of "Long Fork"




The undesired cycle for "long fork" found by MonoSAT.

# Experimental Evaluation

(1) *Effective:* Can PolySI find SI violations in production databases?

(2) *Informative:* Can PolySI provide understandable counterexamples for SI violations?

(3) *Efficient:* How efficient is PolySI? Is it scalable?

`https://github.com/hengxin/PolySI-PVLDB2023-Artifacts`

# Workloads

Table: Workload parameters and their default values.

| Parameter | Default Value |
|---|---|
| #sess | 20 |
| #txns/sess | 100 |
| #ops/txn | 15 |
| #keys | 10, 000 |
| %reads | 50% |
| distribution | zipfian |

# Benchmarks

RuBis: an eBay-like bidding system

TPC-C: an open standard for OLTP benchmarking

C-Twitter: a Twitter clone

GeneralRH: read-heavy workloads with 95% reads

GeneralRW: medium workloads with 50% reads

GeneralWH: write-heavy workloads with 30% reads

Use a simple database schema of a *two-column table* storing keys and values.

# Finding SI Violations

Table: Reproducing known SI violations.

| Database | GitHub Stars | Kind | Release |
|---|---|---|---|
| CockroachDB | 25.1k | Relational | v2.1.0, v2.1.6 |
| MySQL-Galera | 381 | Relational | v25.3.26 |
| YugabyteDB | 6.7k | Multi-model | v1.1.10.0 |

An extensive collection of 2477 anomalous histories

**[Complexity:OOPSLA2019; CockroachDB-bug; YugabyteDB-bug]**

# Finding SI Violations

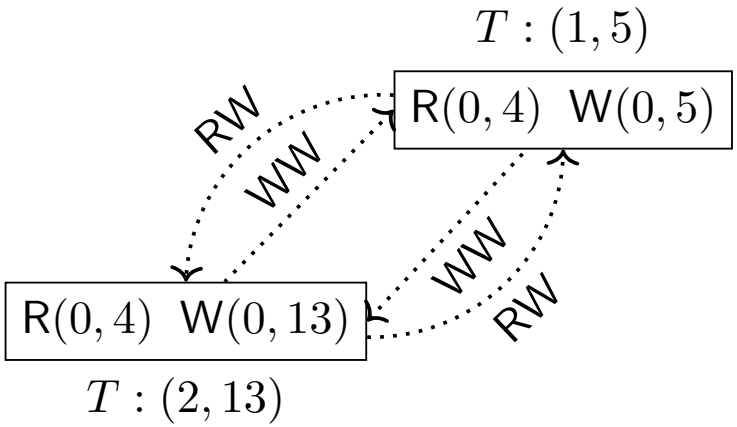
Dgraph: helped the Dgraph team confirm some of their suspicions about their latest release

Table: Detecting new violations.

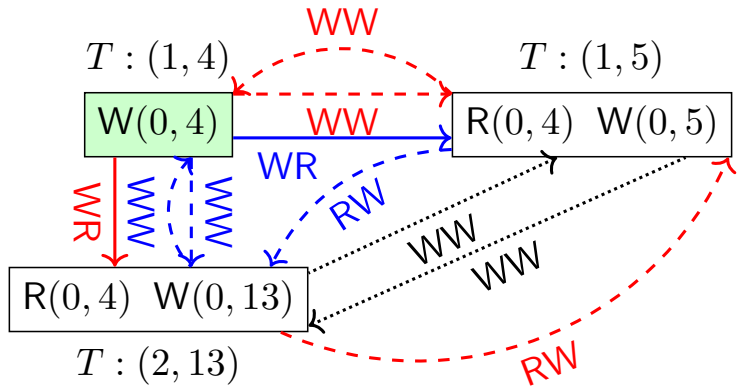
| Database | GitHub Stars | Kind | Release |
|---|---|---|---|
| Dgraph | 18.2k | Graph | v21.12.0 |
| MariaDB-Galera | 4.4k | Relational | v10.7.3 |
| YugabyteDB | 6.7k | Multi-model | v2.11.1.0 |

Galera: confirmed the incorrect claim on preventing "lost updates" for transactions issued on different cluster nodes

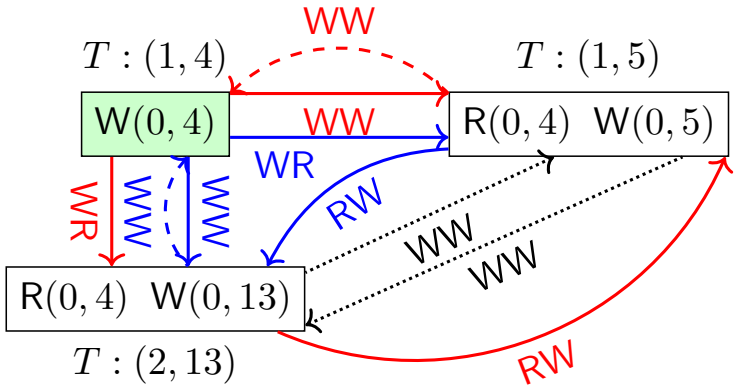
# Understanding Violations (Lost Update)



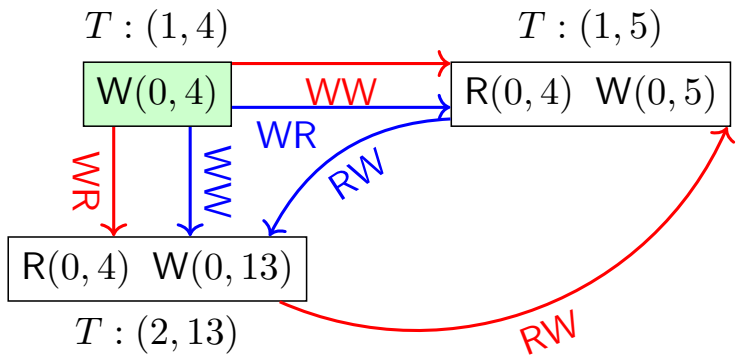

(a) Original output




(b) Missing participants




(c) Recovered scenario




(d) Finalized scenario

# Performance Evaluation

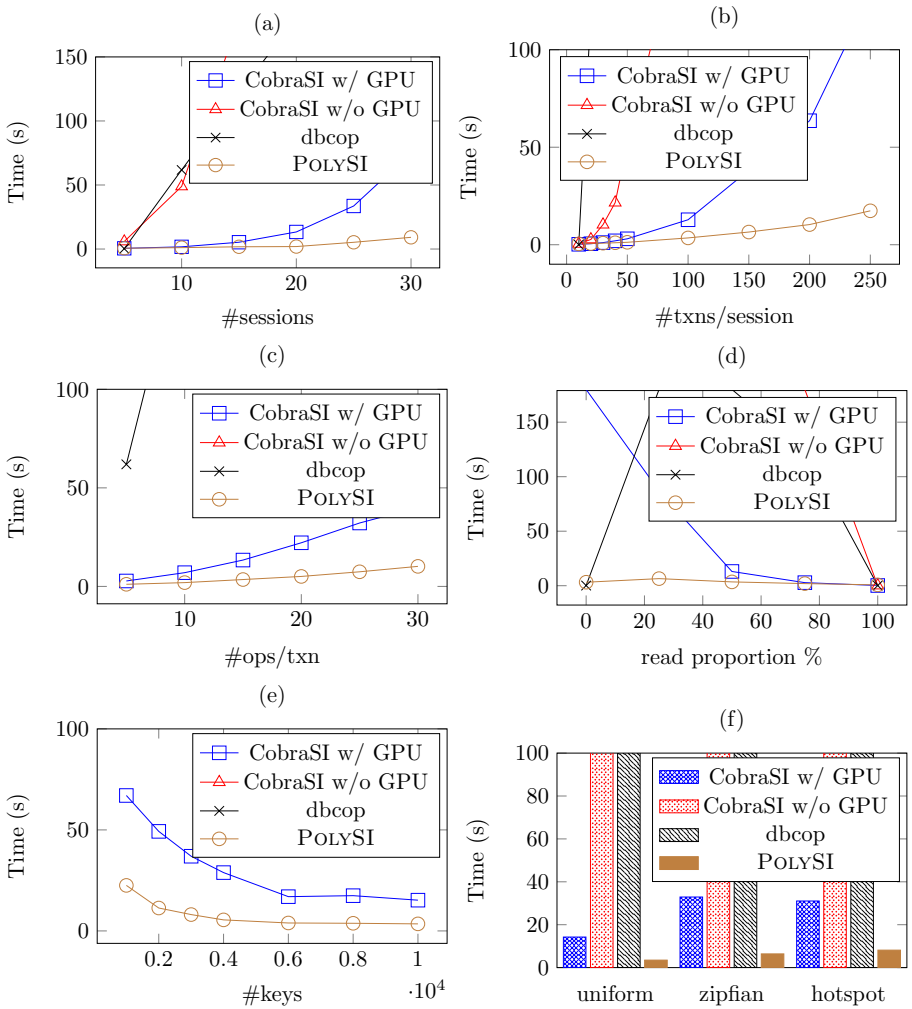
dbcop [**Complexity:OOPSLA2019**]: the state-of-the-art SI checker without using SAT solvers

Cobra [**Cobra:OSDI2020**]: the state-of-the-art SER checker using both MonoSAT and GPU; as a baseline

CobraSI: reducing SI checking to SER checking [**Complexity:OOPSLA2019**] to leverage Cobra with/without GPU

# Performance Evaluation: Runtime

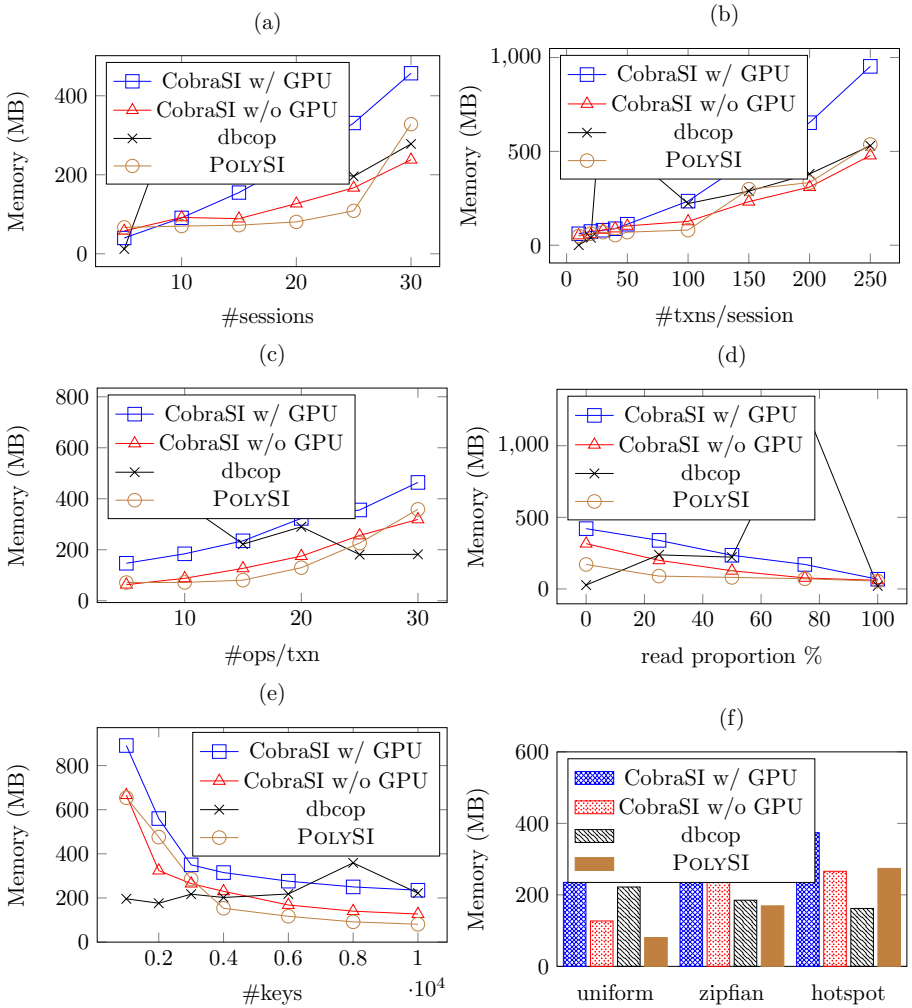
PolySI significantly outperforms the competitors.




All the input histories extracted from PostgreSQL satisfy SI.

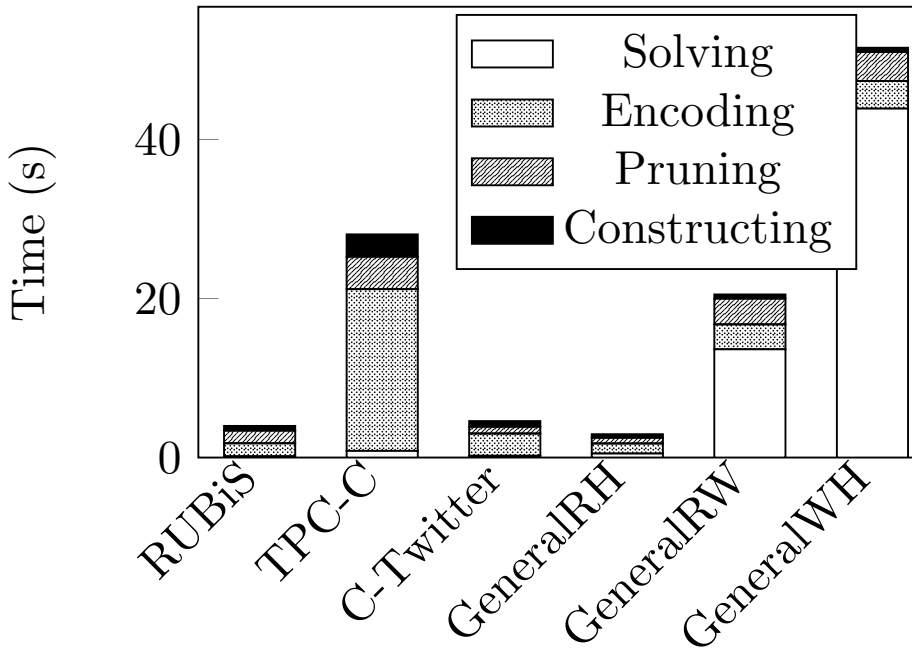
# Performance Evaluation: Memory

PolySI consumes less memory.



# Performance Evaluation: Decomposition

TPC-C incurs more overhead in *encoding* as the number of operations in total is 5x more than the others.




The solving time depends on the remaining constraints and unknown dependencies *after pruning*.

# Performance Evaluation: Pruning

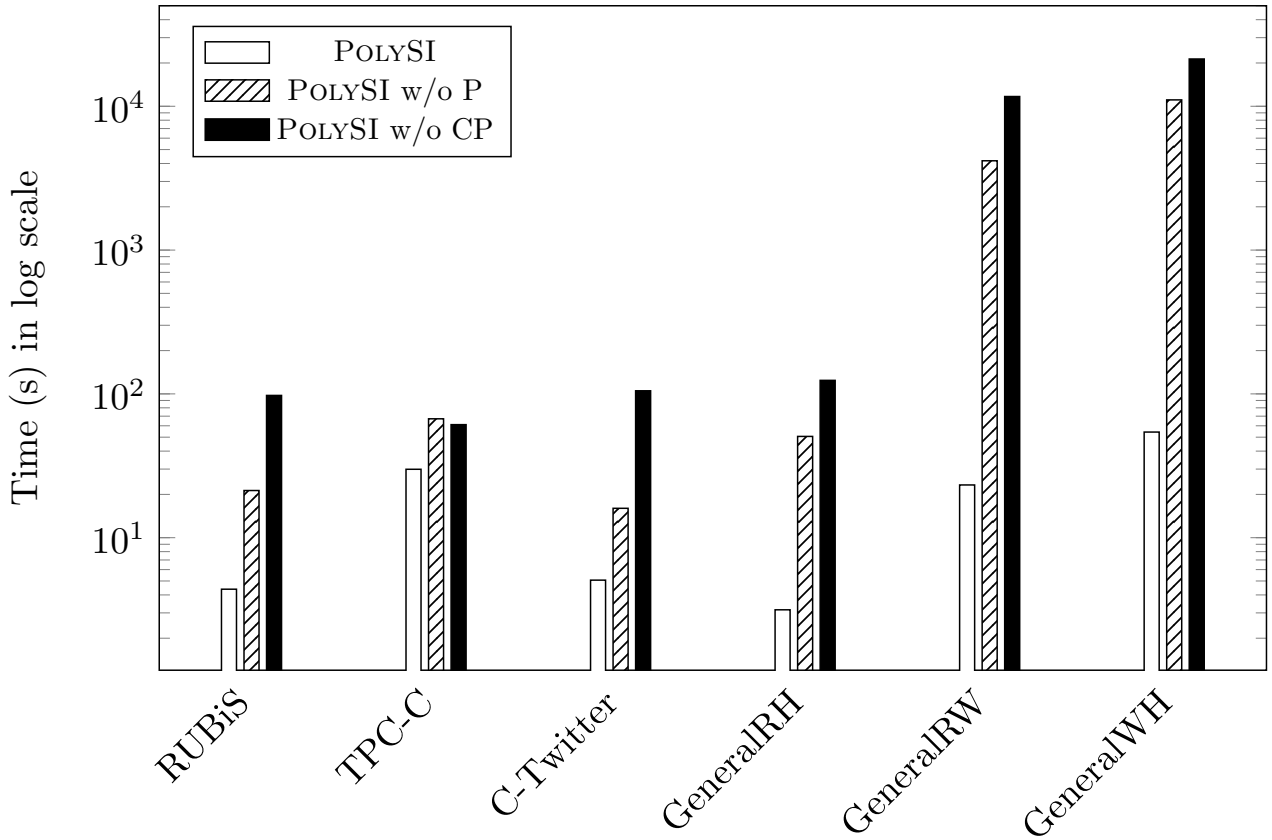
PolySI can effectively prune a huge number of constraints.

| Benchmark | #cons. before P | #cons. after P | #unk. dep. before P | #unk. dep. after P |
|---|---|---|---|---|
| TPC-C | 386k | 0 | 3628k | 0 |
| GeneralRH | 4k | 29 | 39k | 77 |
| RUBiS | 14k | 149 | 171k | 839 |
| C-Twitter | 59k | 277 | 307k | 776 |
| GeneralRW | 90k | 2565 | 401k | 5435 |
| GeneralWH | 167k | 6962 | 468k | 14376 |

TPC-C: read-only transactions + RMW transactions

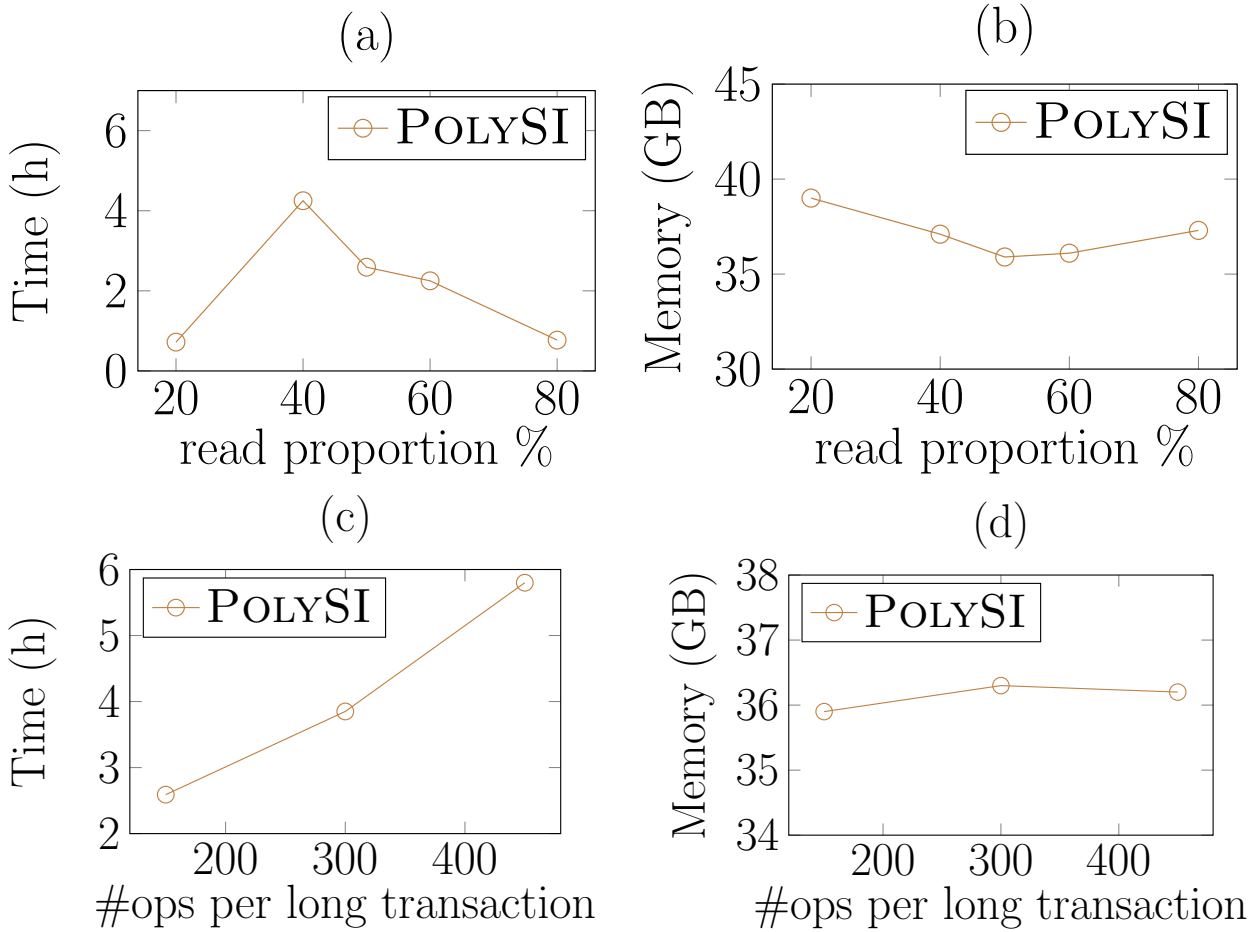
# Performance Evaluation: Differential Analysis

Pruning is crucial to the efficiency of PolySI.

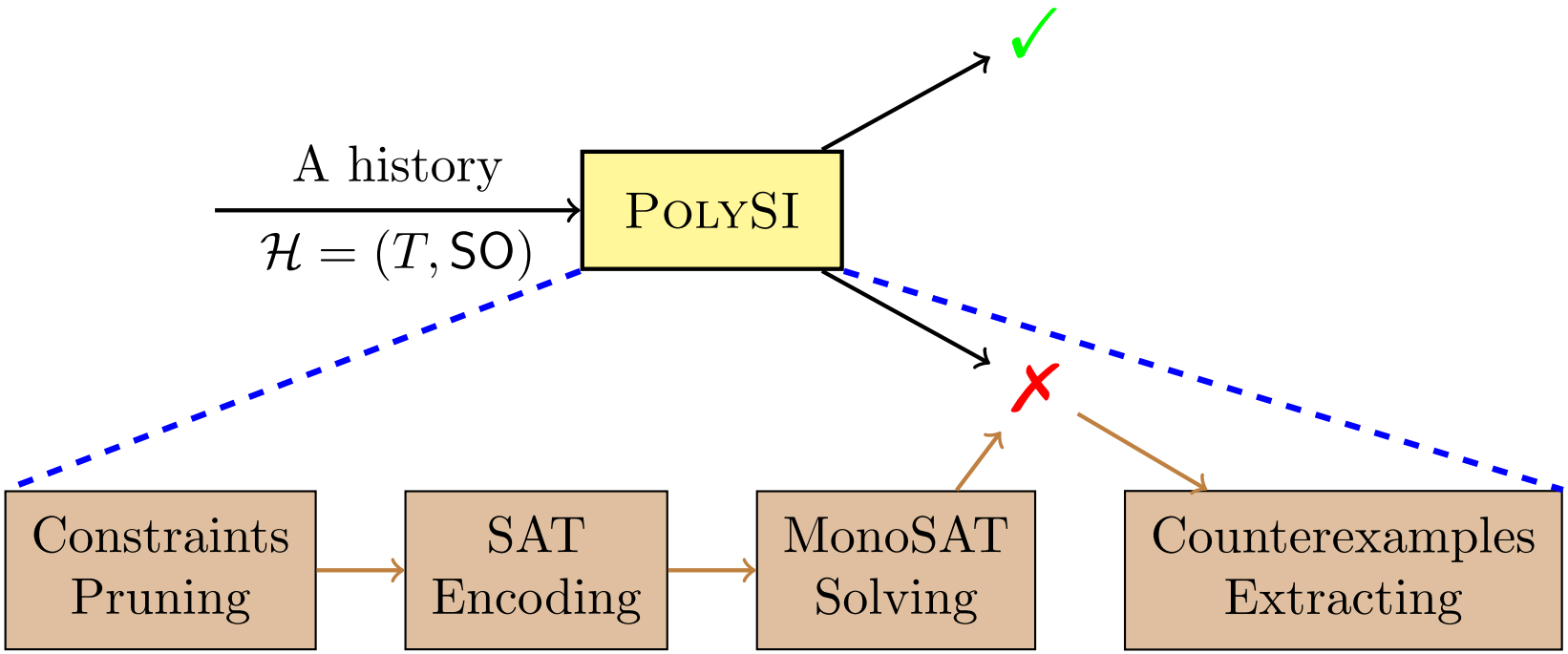


# Performance Evaluation: Scalability

several hours and 35 $\sim$ 40GB memory for checking 1M transactions



# Conclusion


A history
$\mathcal{H} = (T, \mathsf{SO})$
PolySI
Constraints Pruning
SAT Encoding
MonoSAT Solving
Counterexamples Extracting

# Future Work

PolySI uses MonoSAT as a black-box.

Working on a **theory solver** dedicated to isolation level checking, which is deeply integrated with SAT solvers [**Zord:PLDI2021**].


Hengfeng Wei (hfwei@nju.edu.cn)