

# Efficient Black-box Checking of Snapshot Isolation in Databases

Kaile Huang, Si Liu, Zhenge Chen,  
*Hengfeng Wei*, David Basin, Haixiang Li, Anqun Pan

hfwei@nju.edu.cn

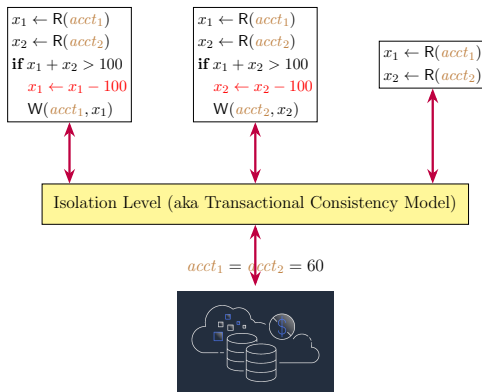
August 24, 2023



**ETH** zürich Tencent 腾讯

# Transaction and Isolation Level

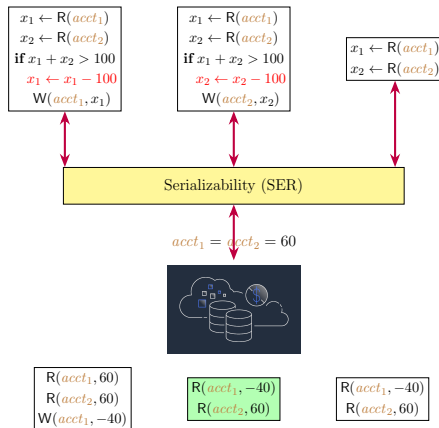
A transaction is a *group* of operations that are executed **atomically**.



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

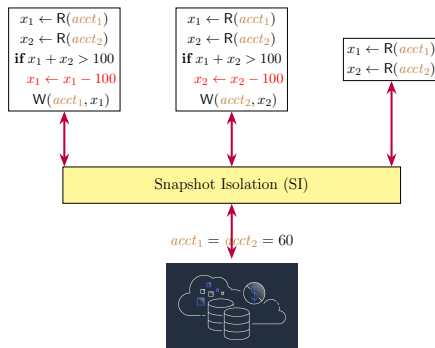
# Serializability (SER)

All transactions appear to be executed in some total order.



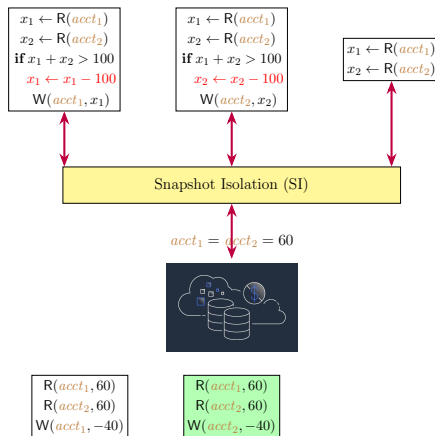
Implementing serializability is too expensive.

# Snapshot Isolation (SI)



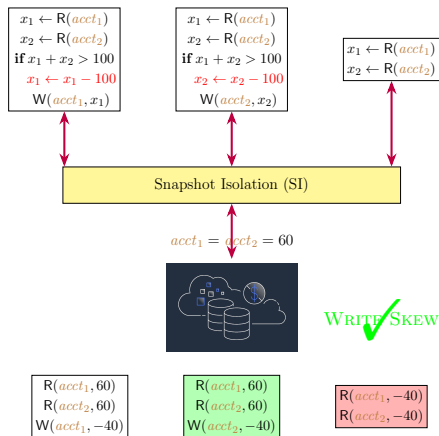
**Snapshot Read:** Each transaction reads data from a *snapshot* as of the time the transaction started.

# Snapshot Isolation (SI)



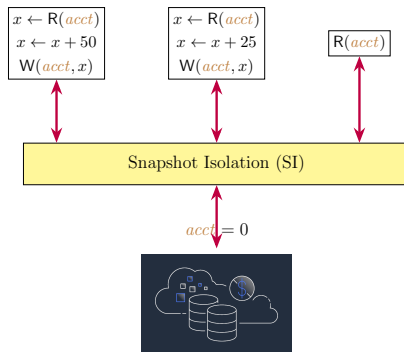
**Snapshot Read:** Each transaction reads data from a *snapshot* as of the time the transaction started.

# Snapshot Isolation (SI)



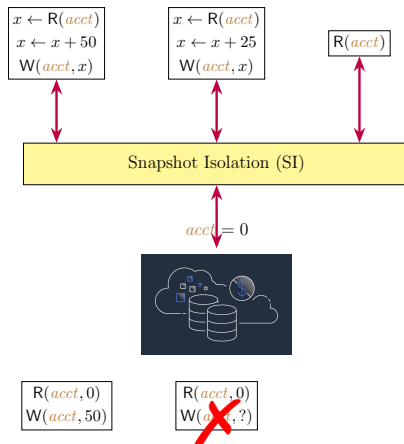
**Snapshot Read:** Each transaction reads data from a *snapshot* as of the time the transaction started.

# Snapshot Isolation (SI)



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

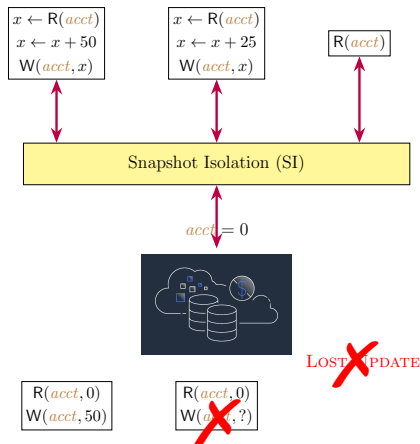
# Snapshot Isolation (SI)



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



# Snapshot Isolation (SI)



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

# Database systems and Snapshot Isolation

Many database systems implement snapshot isolation.



# Database Systems and Snapshot Isolation

Database systems may **fail** to provide snapshot isolation correctly.



## Elle: Inferring Isolation Anomalies from Experimental Observations

Kyle Kingsbury  
Jepsen  
aphyr@jepsen.io

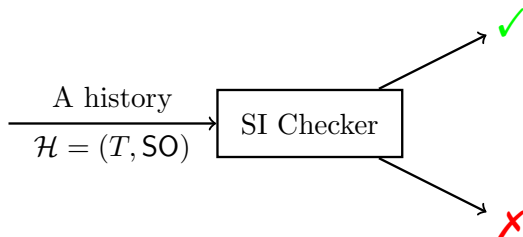
Peter Alvaro  
UC Santa Cruz  
palvaro@ucsc.edu



# The SI Checking Problem

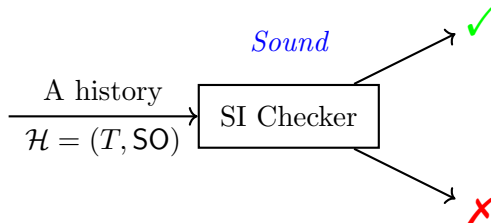
## Definition (The SI Checking Problem)

The SI checking problem is the **decision problem** of determining whether a *history*  $\mathcal{H} = (T, \text{SO})$  of a database system satisfies SI?



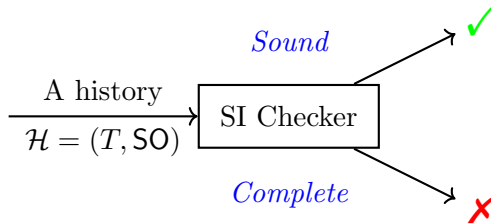
SO : *session order* among the set  $T$  of transactions

# The SI Checking Problem



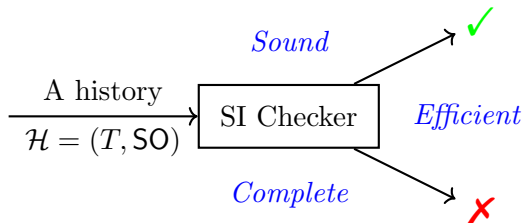
*Sound:* If the checker says **X**, then the history does *not* satisfy SI.

# The SI Checking Problem



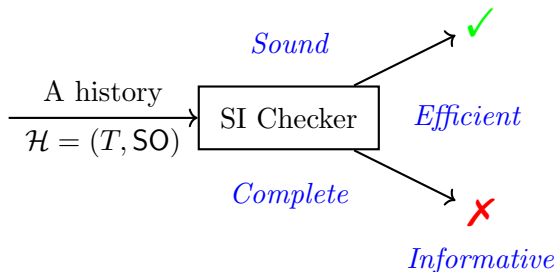
*Complete:* If the checker says ✓, then the history *satisfies* SI.

# The SI Checking Problem



*Efficient:* The checker should *scale* up to large workloads.

# The SI Checking Problem



*Informative:* The checker should provide understandable *counterexamples* if it finds violations.



# Related Work

- dbcop [Biswas and Enea, 2019] checker for SI
  - not practically efficient;
  - not informative, returning only “False” upon violations
- Cobra [Tan et al., 2020] state-of-the-art checker for SER
  - The SI checking problem is *harder*.

# Related Work

Elle [Kingsbury and Alvaro, 2020] checker for various isolation levels

Work perfectly on traceable and recoverable histories;  
but may be incomplete on the key-value datatype

---

<sup>a</sup>Could Elle tell the difference between snapshot isolation and strong-session-snapshot-isolation? <https://github.com/jepsen-io/elle/issues/17>

<sup>b</sup>Elle may miss two types of transaction anomalies.  
<https://github.com/jepsen-io/elle/issues/21>

# Related Work

Elle [Kingsbury and Alvaro, 2020] checker for various isolation levels

Work perfectly on traceable and recoverable histories;  
but may be incomplete on the key-value datatype

SI checking based on the Adya-style notions [Adya, 1999]  
relies on the start/commit timestamps of transactions.

---

<sup>a</sup>Could Elle tell the difference between snapshot isolation and strong-session-snapshot-isolation? <https://github.com/jepsen-io/elle/issues/17>

<sup>b</sup>Elle may miss two types of transaction anomalies.  
<https://github.com/jepsen-io/elle/issues/21>

# Related Work

Elle [Kingsbury and Alvaro, 2020] checker for various isolation levels

Work perfectly on traceable and recoverable histories;  
but may be incomplete on the key-value datatype

SI checking based on the Adya-style notions [Adya, 1999]  
relies on the start/commit timestamps of transactions.

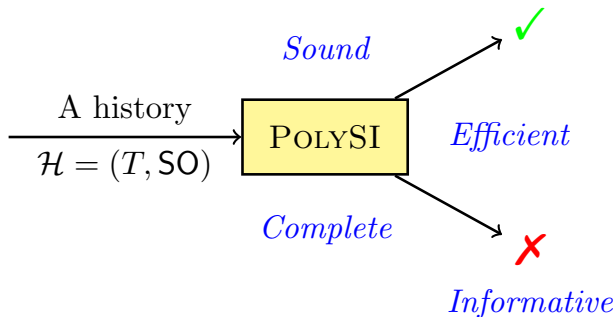
SI checking based on the [Cerone and Gotsman, 2018]  
notions may miss SI violations.<sup>a</sup> <sup>b</sup>

---

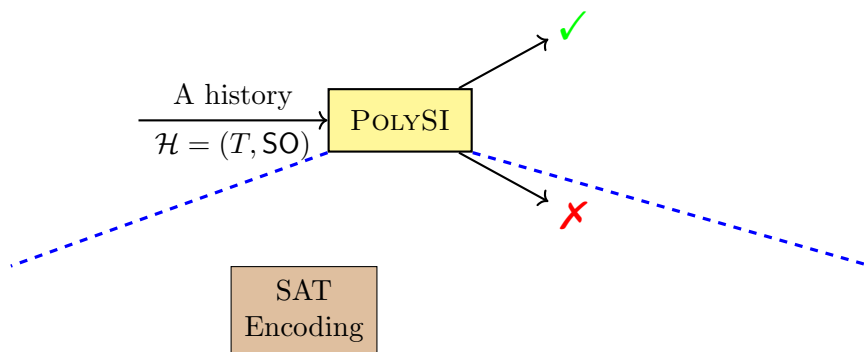
<sup>a</sup>Could Elle tell the difference between snapshot isolation and strong-session-snapshot-isolation? <https://github.com/jepsen-io/elle/issues/17>

<sup>b</sup>Elle may miss two types of transaction anomalies.  
<https://github.com/jepsen-io/elle/issues/21>

# Contribution: the POLYSI Checker

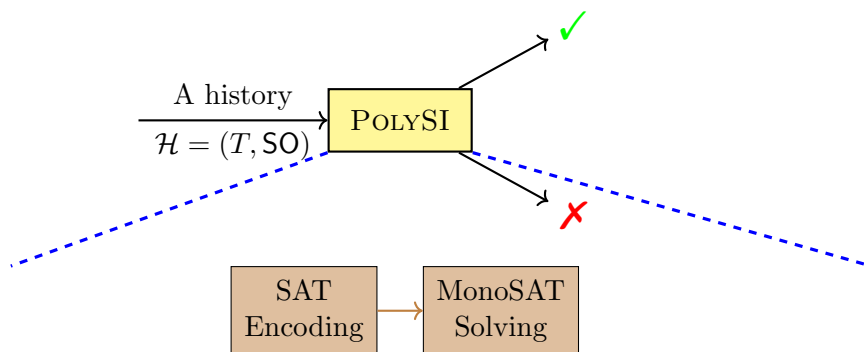


# Contribution: the POLYSI Checker



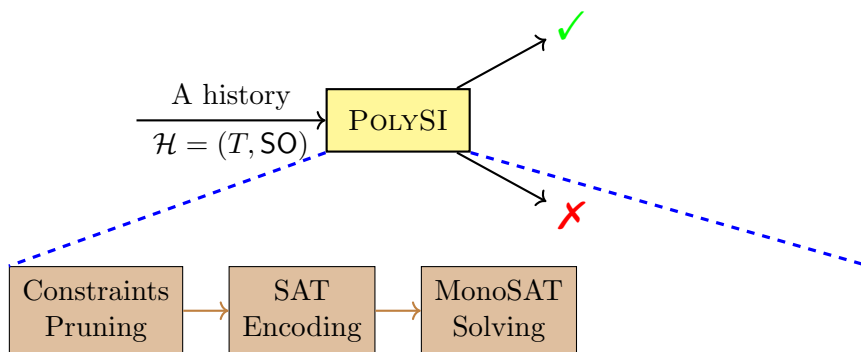
*Sound & Complete:* a novel polygraph based characterization of SI

# Contribution: the POLYSI Checker



*Efficient:* utilizing MonoSAT solver optimized for graph problems

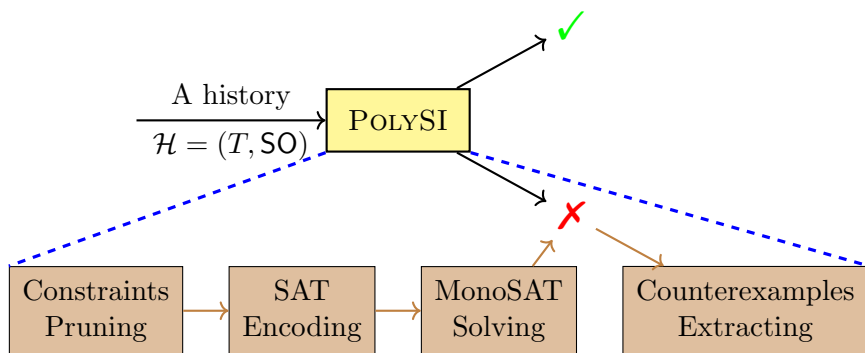
# Contribution: the POLYSI Checker



*Efficient:* domain-specific pruning before encoding



# Contribution: the POLYSI Checker



*Informative:* extract counterexamples from the UNSAT core

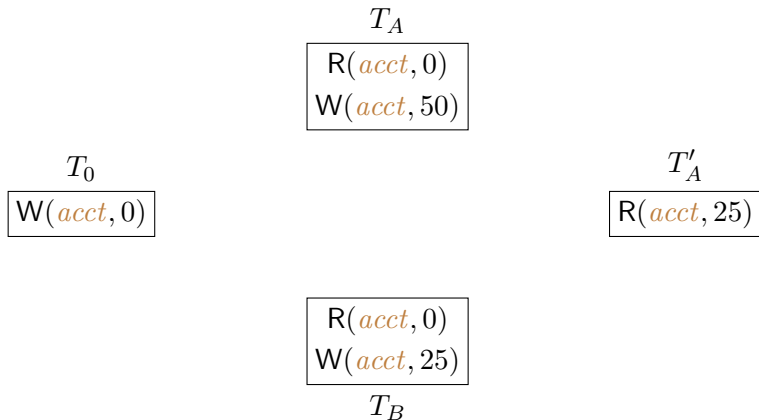
# POLYSI: Polygraph based Characterization of SI

Before this, we first review the *dependency graph* based characterization of SI [Cerone and Gotsman, 2018].

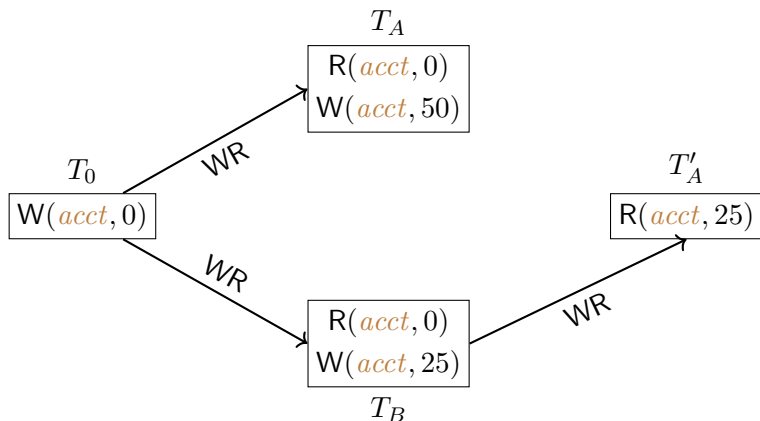
Theorem (Theorem 4.1 of [Cerone and Gotsman, 2018])

*Informally, a history satisfies SI if and 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

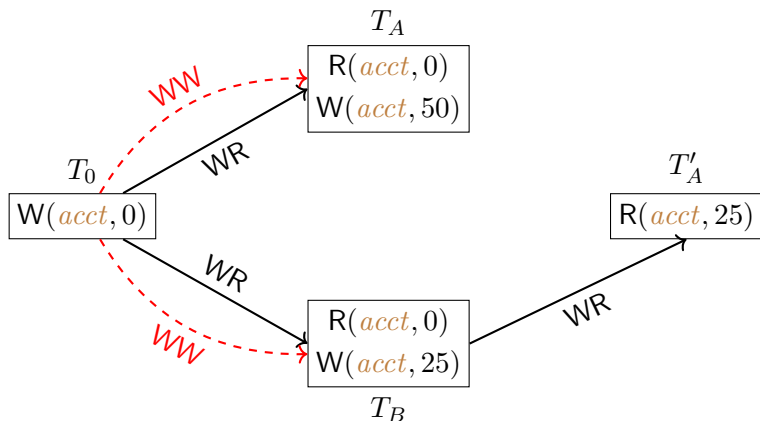


# Dependency Graph based Characterization of SI



WR: “write-read” dependency capturing the “read-from” relation

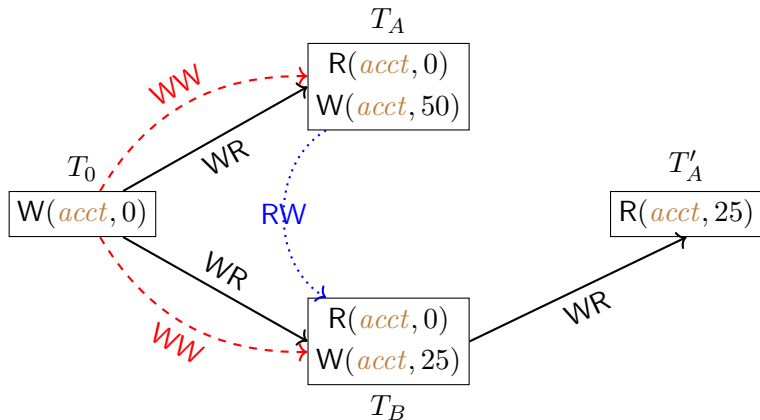
# Dependency Graph based Characterization of SI



**WW**: “write-write” dependency capturing the version order on *acct*

# Dependency Graph based Characterization of SI

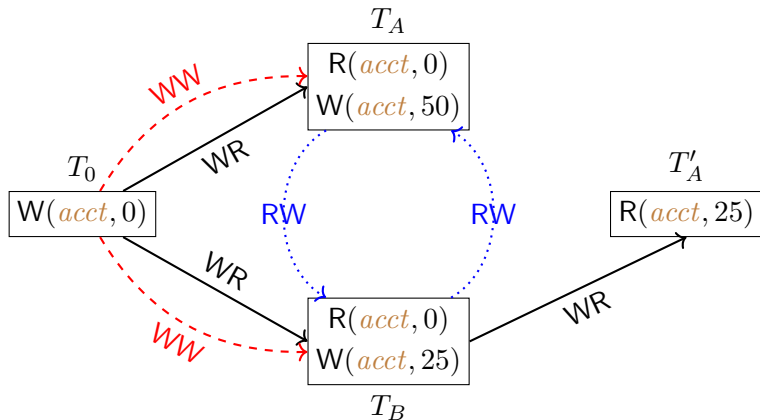
$T_A$  reads from  $T_0$  which is overwritten by  $T_B$



*RW*: “read-write” dependency

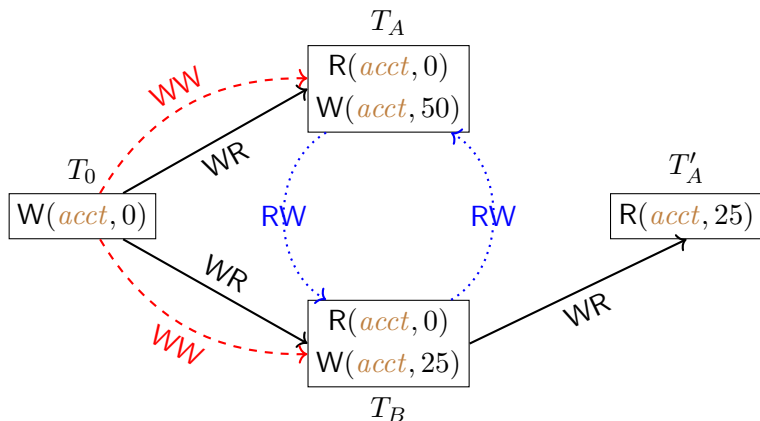
# Dependency Graph based Characterization of SI

$T_B$  reads from  $T_0$  which is overwritten by  $T_A$



*RW*: “read-write” dependency

# Dependency Graph based Characterization of SI

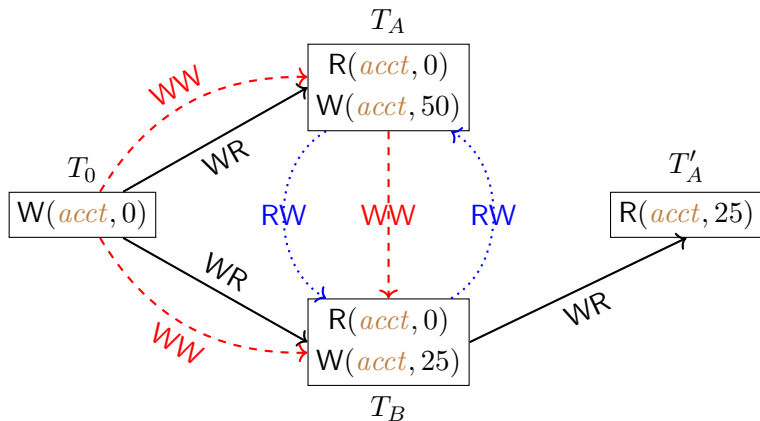


The cycle  $T_A \xrightarrow{RW} T_B \xrightarrow{RW} T_A$  is **allowed** by SI.



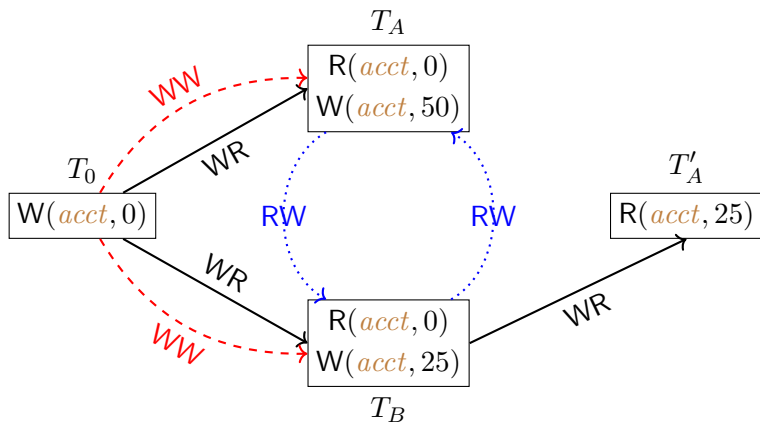
# Dependency Graph based Characterization of SI

Suppose that  $T_A \xrightarrow{WW} T_B$



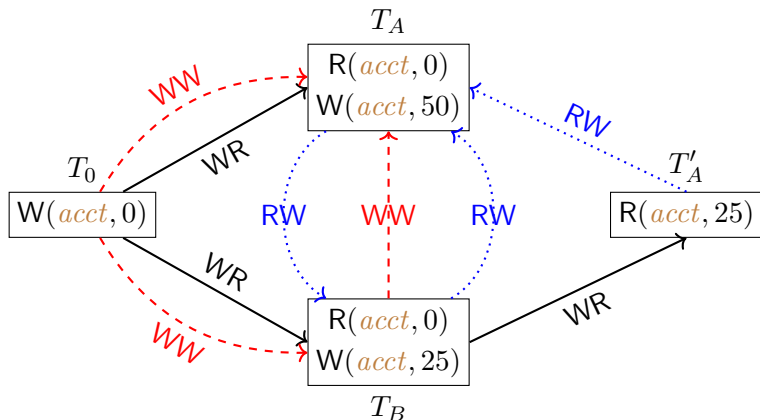
undesired cycle for SI:  $T_A \xrightarrow{WW} T_B \xrightarrow{RW} T_A$

# Dependency Graph based Characterization of SI



# Dependency Graph based Characterization of SI

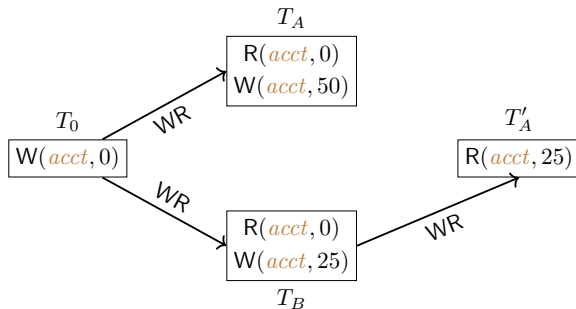
Suppose that  $T_B \xrightarrow{WW} T_A$



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

# Dependency Graph based Characterization of SI

We have considered both bases  $T_A \xrightarrow{WW} T_B$  and  $T_B \xrightarrow{WW} T_A$ ,  
and each case leads to an undesired cycle for SI.



Therefore, this history does not satisfy SI.

# Dependency Graph based Characterization of SI

Theorem (Equivalence of Theorem 4.1 of [Cerone and Gotsman, 2018])

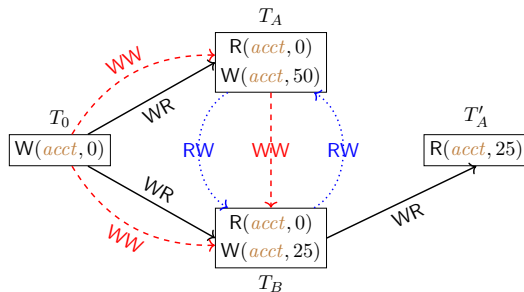
*Informally, a history satisfies SI if and only if*

*there exists a dependency graph  $\mathcal{G}$  for it such that*

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

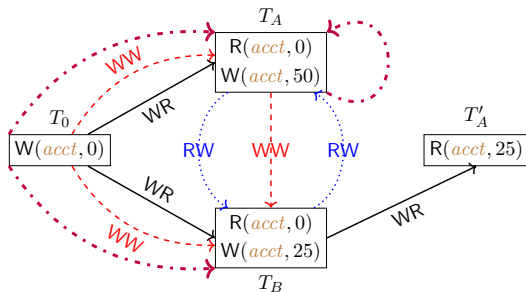
# Dependency Graph based Characterization of SI

induced graph  $\boxed{((SO_{\mathcal{G}} \cup WR_{\mathcal{G}} \cup \textcolor{red}{WW}_{\mathcal{G}}) ; \textcolor{blue}{RW}_{\mathcal{G}}?)}$  for  $\mathcal{G}$



# Dependency Graph based Characterization of SI

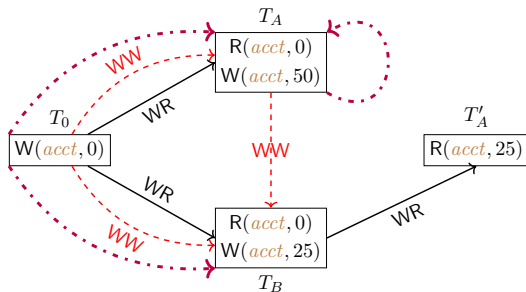
induced graph  $\boxed{((SO_{\mathcal{G}} \cup WR_{\mathcal{G}} \cup \textcolor{red}{WW}_{\mathcal{G}}) ; \textcolor{blue}{RW}_{\mathcal{G}}?)}$  for  $\mathcal{G}$



first composing ( ; )  $SO/WR/\textcolor{red}{WW}$  edges with  $\textcolor{blue}{RW}$  edges

# Dependency Graph based Characterization of SI

induced graph  $\boxed{((SO_{\mathcal{G}} \cup WR_{\mathcal{G}} \cup WW_{\mathcal{G}}) ; RW_{\mathcal{G}}?)}$  for  $\mathcal{G}$

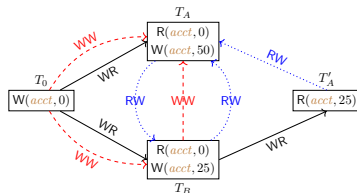
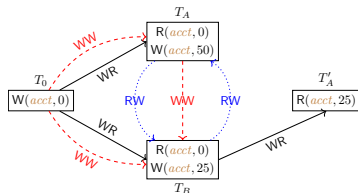


first composing ( ; )  $SO/WR/WW$  edges with  $RW$  edges  
 then deleting the  $RW$  edges



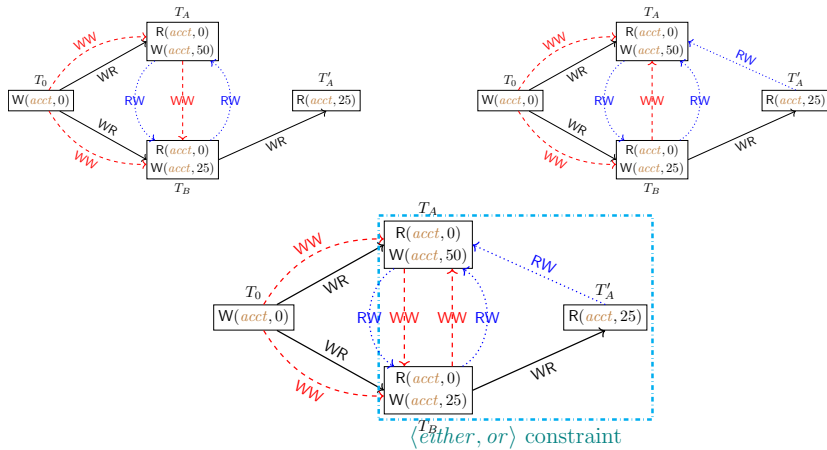
# Polygraph based Characterization of SI

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



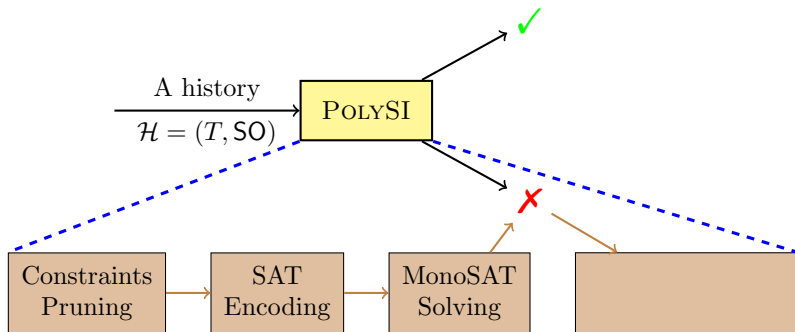
# Polygraph based Characterization of SI

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

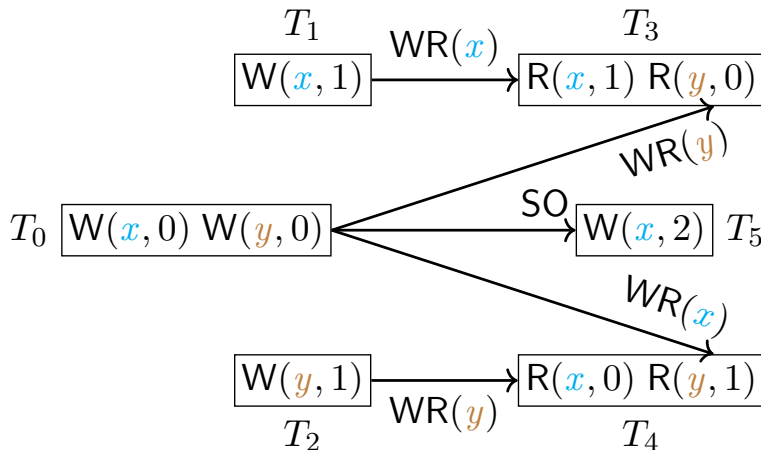


polygraph:  $\langle \text{either} \triangleq \{T_A \xrightarrow{\text{WW}} T_B\}, \text{or} \triangleq \{T_B \xrightarrow{\text{WW}} T_A, T'_A \xrightarrow{\text{RW}} T_A\} \rangle$

# POLYSI: An Illustrating Example

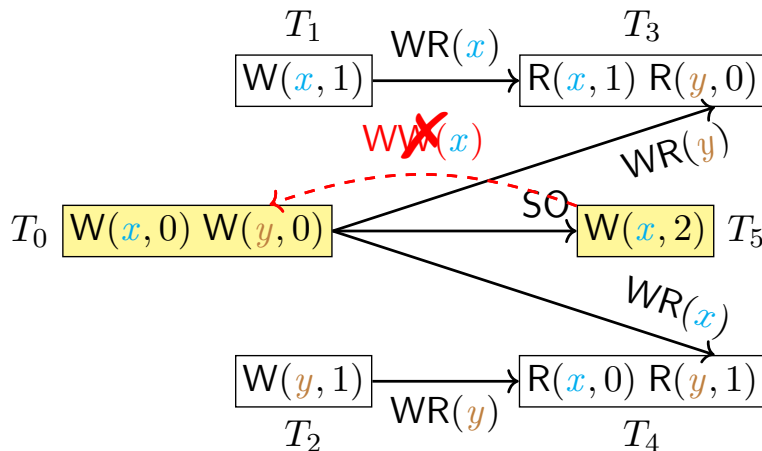


# POLYSI: An Illustrating Example



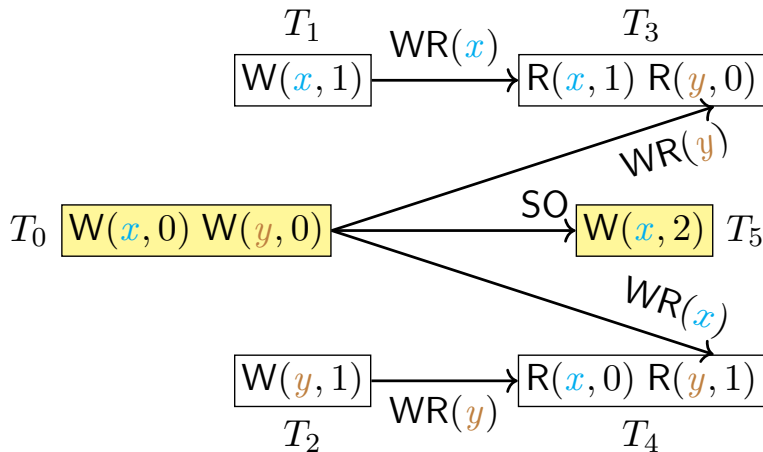
**WW** between  $T_0$ ,  $T_1$ , and  $T_5$  (on  $x$ ) and between  $T_0$  and  $T_2$  (on  $y$ )

# POLYSI: An Illustrating Example

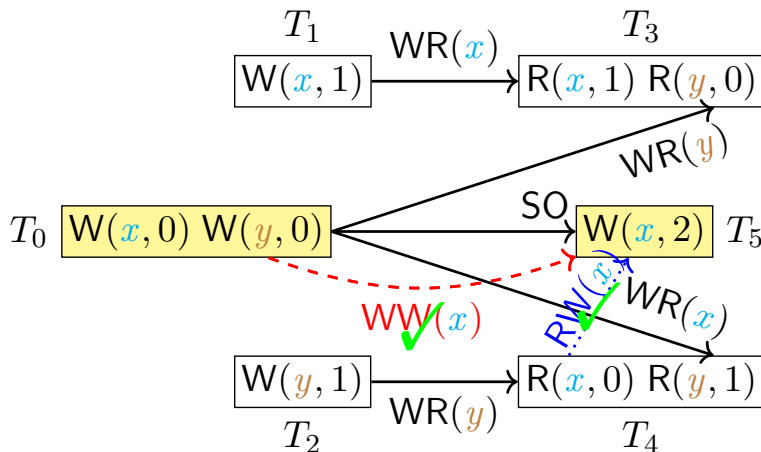


The  $T_5 \xrightarrow{WW(x)} T_0$  case is pruned due to  $T_0 \xrightarrow{SO} T_5 \xrightarrow{WW(x)} T_0$ .

# POLYSI: An Illustrating Example

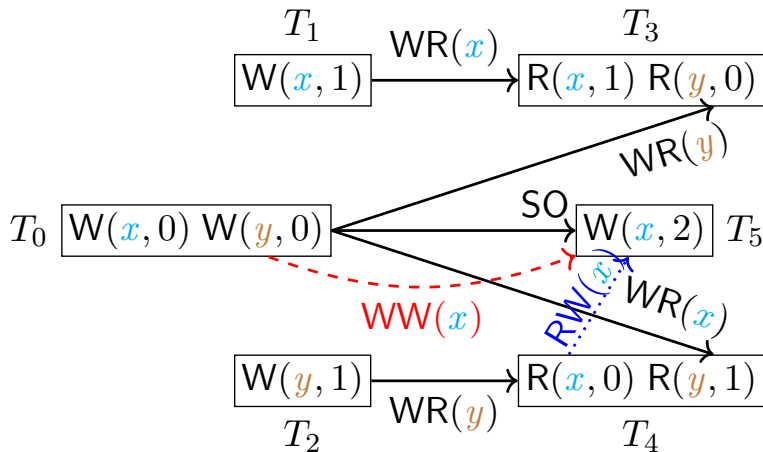


# POLYSI: An Illustrating Example



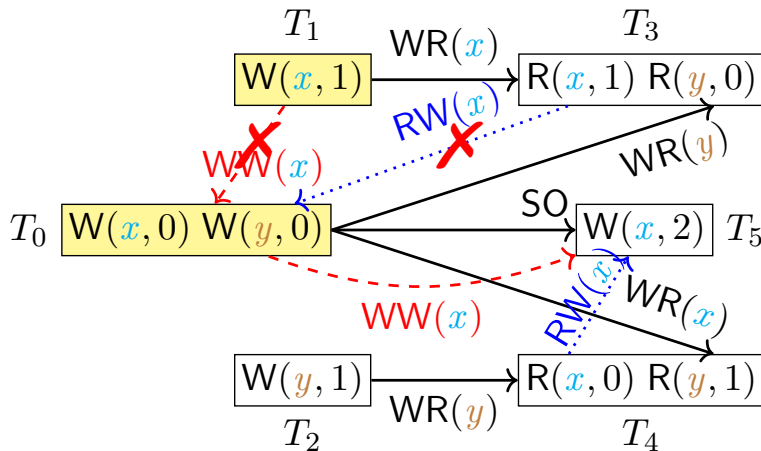
The  $T_0 \xrightarrow{WW(x)} T_5$  case becomes known.

# POLYSI: An Illustrating Example



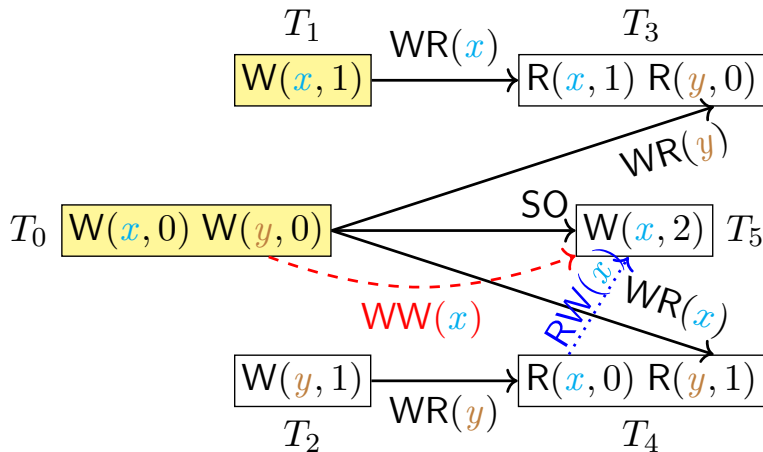


# POLYSI: An Illustrating Example

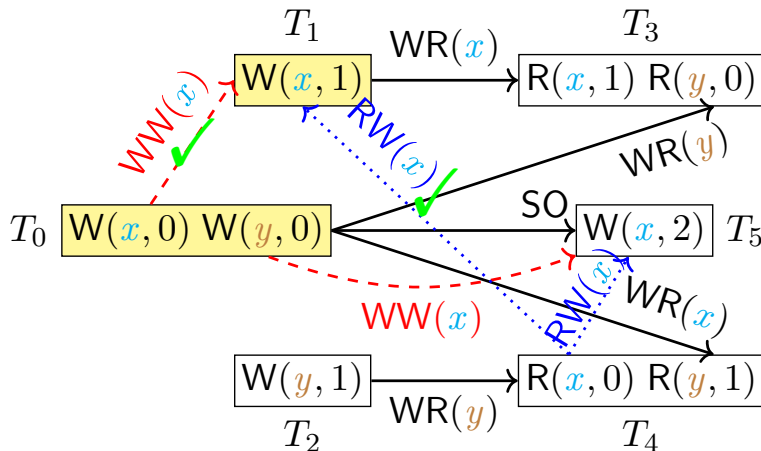


The  $T_1 \xrightarrow{WW(x)} T_0$  case is pruned due to  $T_3 \xrightarrow{RW(x)} T_0 \xrightarrow{WR(y)} T_3$ .

# POLYSI: An Illustrating Example

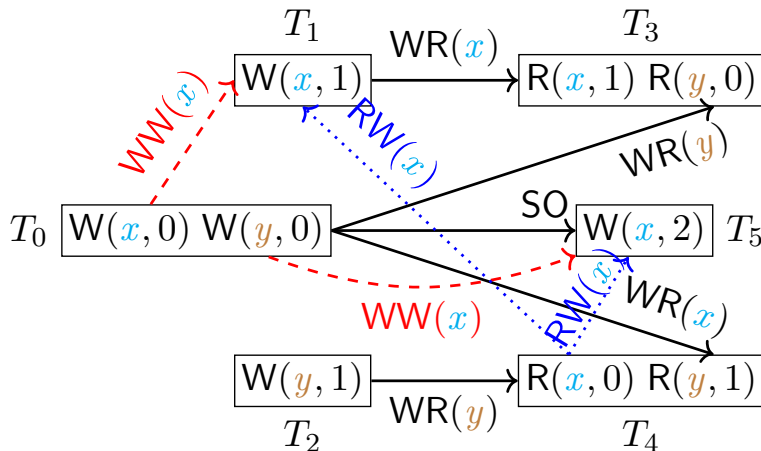


## POLYSI: An Illustrating Example

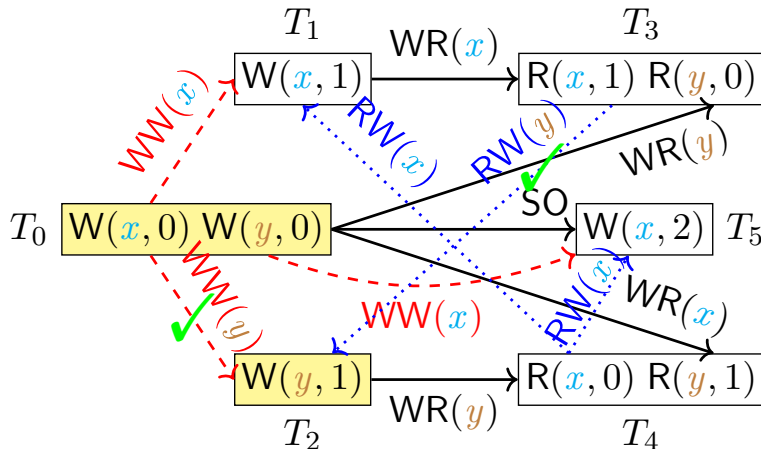


The  $T_0 \xrightarrow{\text{WW}(x)} T_1$  case becomes known.

# POLYSI: An Illustrating Example

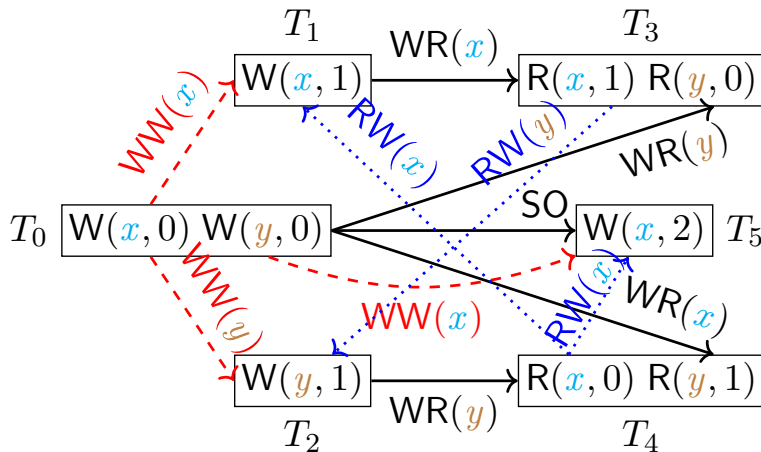


# POLYSI: An Illustrating Example

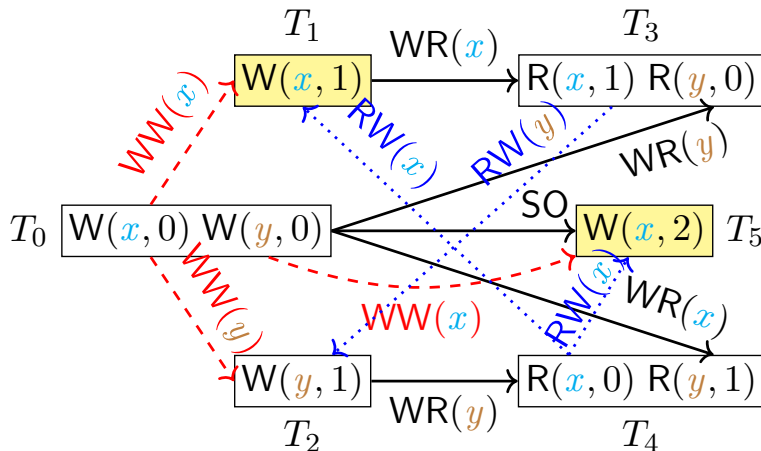


The  $T_2 \xrightarrow{WW(y)} T_0$  case is pruned,  
 while the  $T_0 \xrightarrow{WW(y)} T_2$  case becomes known.

# POLYSI: An Illustrating Example



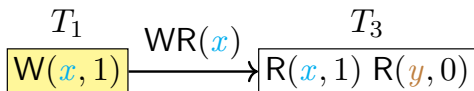
# POLYSI: An Illustrating Example



The **WW** order between  $T_1$  and  $T_5$  is still uncertain after pruning.

# POLYSI: An Illustrating Example

< , >



$$T_0 \quad \boxed{W(x, 0) \ W(y, 0)}$$

$$\boxed{W(x, 2)} \quad T_5$$

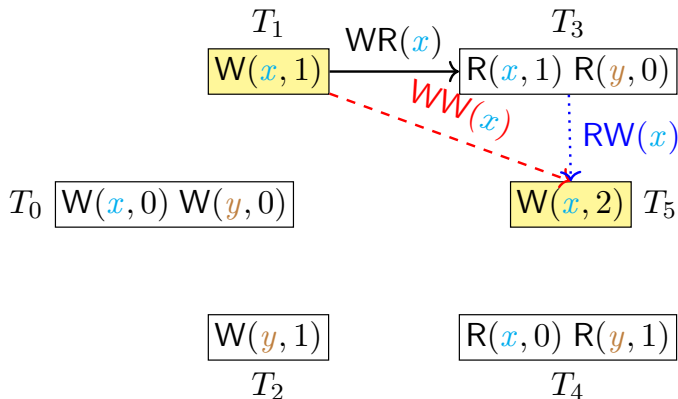
$$\begin{array}{c}
 \boxed{W(y, 1)} \\
 T_2
 \end{array}$$

$$\begin{array}{c}
 \boxed{R(x, 0) \ R(y, 1)} \\
 T_4
 \end{array}$$



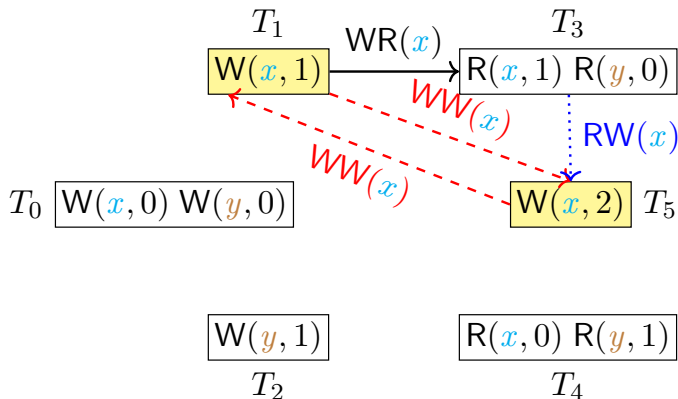
# POLYSI: An Illustrating Example

$$\langle \textit{either} = \{T_1 \xrightarrow{\textcolor{red}{WW}(x)} T_5, T_3 \xrightarrow{\textcolor{blue}{RW}(x)} T_5\}, \quad \rangle$$



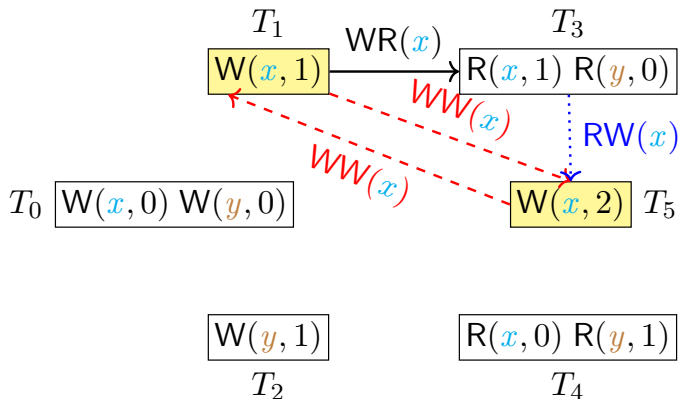
# POLYSI: An Illustrating Example

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



# POLYSI: An Illustrating Example

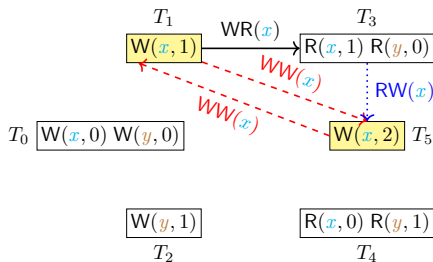
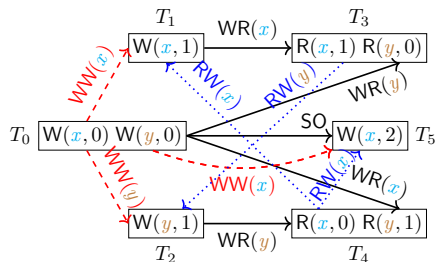
$$\langle \textit{either} = \{T_1 \xrightarrow{\text{WW}(x)} T_5, T_3 \xrightarrow{\text{RW}(x)} T_5\}, \textit{or} = \{T_5 \xrightarrow{\text{WW}(x)} T_1\} \rangle$$



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

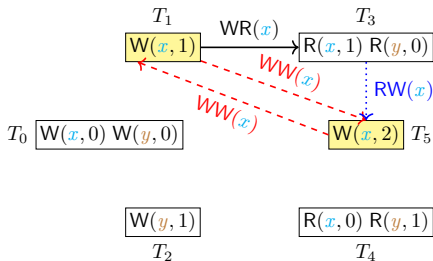
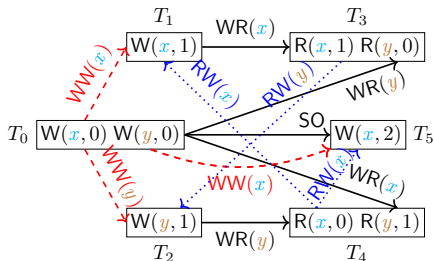
# POLYSI: An Illustrating Example

induced graph  $\mathcal{I} \left[ ((\text{SO}_{\mathcal{G}} \cup \text{WR}_{\mathcal{G}} \cup \text{WW}_{\mathcal{G}}) ; \text{RW}_{\mathcal{G}}?) \right]$



# POLYSI: An Illustrating Example

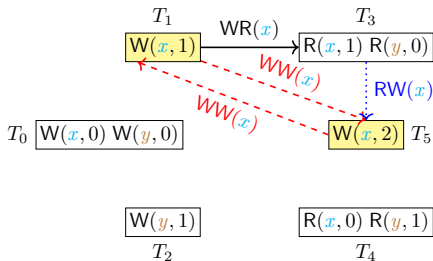
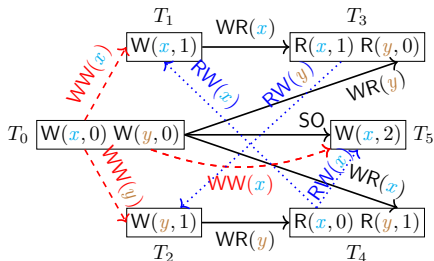
induced graph  $\mathcal{I} \left[ ((\text{SO}_{\mathcal{G}} \cup \text{WR}_{\mathcal{G}} \cup \text{WW}_{\mathcal{G}}) ; \text{RW}_{\mathcal{G}}?) \right]$



$$T_1 \xrightarrow{\text{WR}} T_3 \xrightarrow{\text{RW}} T_2 : \text{BV}_{1,2}^{\mathcal{I}} = \text{BV}_{1,3} \wedge \text{BV}_{3,2}$$

# POLYSI: An Illustrating Example

induced graph  $\mathcal{I} \left[ ((SO_{\mathcal{G}} \cup WR_{\mathcal{G}} \cup \textcolor{red}{WW}_{\mathcal{G}}) ; \textcolor{blue}{RW}_{\mathcal{G}}?) \right]$



$$T_1 \xrightarrow{WR} T_3 \xrightarrow{\textcolor{blue}{RW}} T_2 : BV_{1,2}^{\textcolor{teal}{\mathcal{I}}} = BV_{1,3} \wedge BV_{3,2}$$

$$T_1 \xrightarrow{WR} T_3 \xrightarrow{\textcolor{blue}{RW}} T_5 : BV_{1,5}^{\textcolor{teal}{\mathcal{I}}} = BV_{1,3} \wedge BV_{3,5}$$

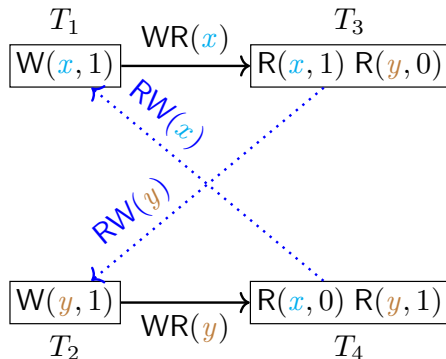
# POLYSI: An Illustrating Example

Feed the SAT formula into the **MonoSAT** solver [Bayless et al., 2015]  
which is optimized for *cycle detection*



Assert that the induced graph *I* is acyclic.

# POLYSI: An Illustrating Example



the undesired cycle 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?

# Workloads

Table: Workload parameters and their default values.<sup>c</sup>

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

<sup>c</sup>We use a database schema of a *two-column table* storing keys and values.

# 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

# Reproducing Known SI Violations

Database	GitHub Stars	Kind	Release
CockroachDB <sup>d</sup>	25.1k	Relational	v2.1.0 <sup>e</sup> , v2.1.6
MySQL-Galera	381	Relational	v25.3.26
YugabyteDB	6.7k	Multi-model	v1.1.10.0 <sup>f</sup>

An extensive collection of 2477 anomalous histories

[Biswas and Enea, 2019; Darnell, Accessed February 14, 2023; Jepsen, Accessed February 14, 2023]

---

<sup>d</sup>Remove SNAPSHOT isolation since (probably) v2.0.4.  
<https://github.com/cockroachdb/cockroach/pull/27040>

<sup>e</sup>Lessons learned from 2+ years of nightly Jepsen tests.  
<https://www.cockroachlabs.com/blog/jepsen-tests-lessons/>

<sup>f</sup>Acknowledged inserts can be present in reads for tens of seconds, then disappear. <https://github.com/YugaByte/yugabyte-db/issues/824>

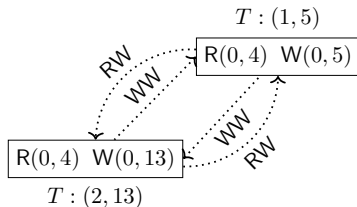
# Detecting New SI Violations

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

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

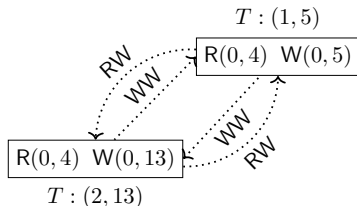
**Galera:** **confirmed** the incorrect claim on preventing “lost updates” for transactions issued on different cluster nodes

# Understanding Violations (Lost Update)

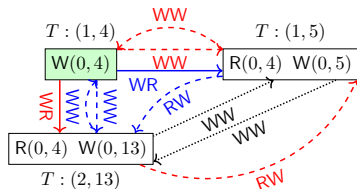


(a) Original output

# Understanding Violations (Lost Update)

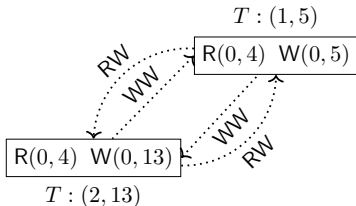


(a) Original output

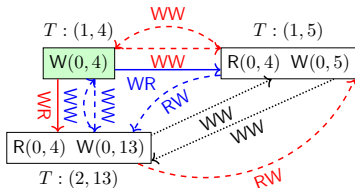


(b) Missing participants

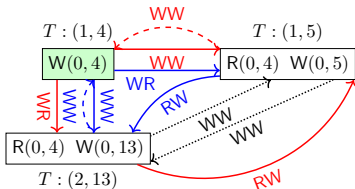
# Understanding Violations (Lost Update)



(a) Original output



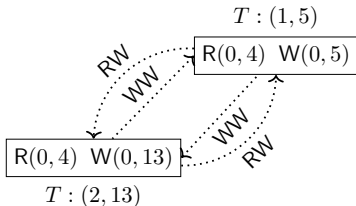
(b) Missing participants



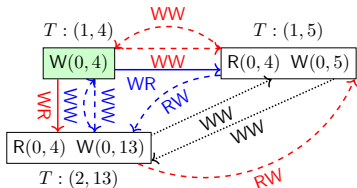
(c) Recovered scenario



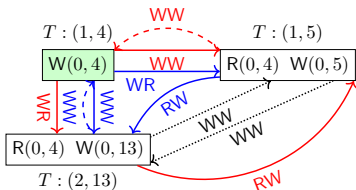
# Understanding Violations (Lost Update)



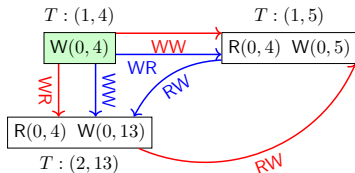
(a) Original output



(b) Missing participants



(c) Recovered scenario



(d) Finalized scenario

# Performance Evaluation

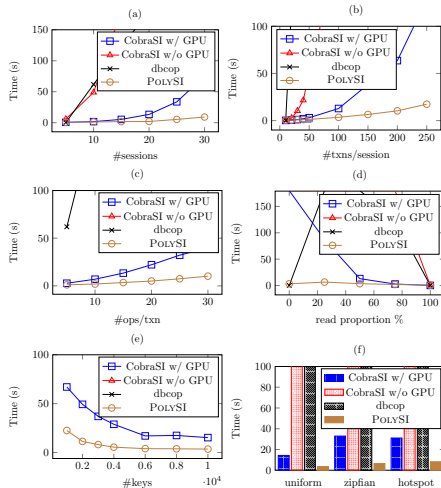
dbcop [Biswas and Enea, 2019]: the state-of-the-art SI checker

CobraSI: reducing SI checking to SER checking  
[Biswas and Enea, 2019] to leverage Cobra with/without GPU

Cobra [Tan et al., 2020]: the state-of-the-art SER checker  
using both MonoSAT and GPU

# Performance Evaluation: Runtime

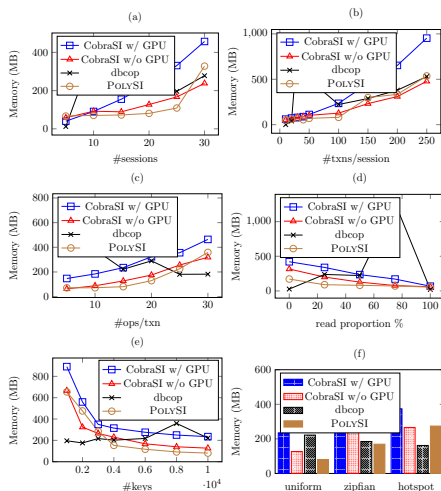
POLYSI significantly outperforms the competitors.<sup>g</sup>



<sup>g</sup>All the input histories extracted from PostgreSQL satisfy SI.

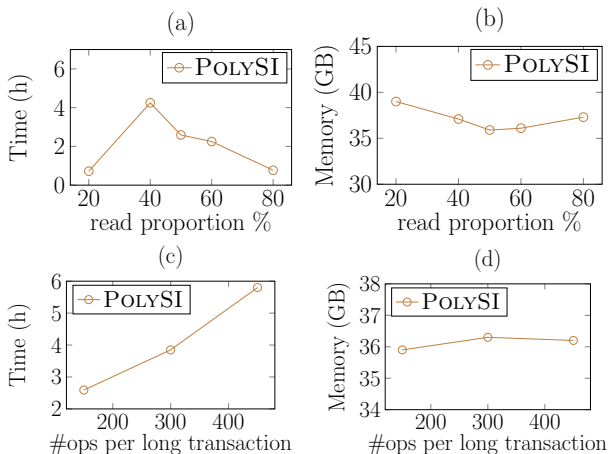
# Performance Evaluation: Memory

POLYSI consumes less memory.



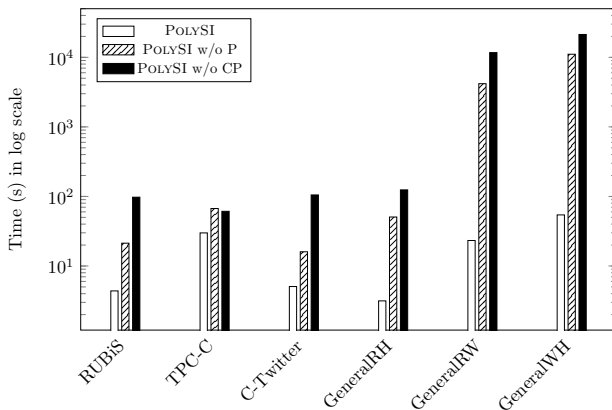
# Performance Evaluation: Scalability

several hours and 35 ~ 40GB memory for checking 1M transactions



# Performance Evaluation: Differential Analysis

Pruning (P) is crucial to the efficiency of POLYSI.<sup>h</sup>



<sup>h</sup>Compacting (C) encoding has been omitted in this presentation.

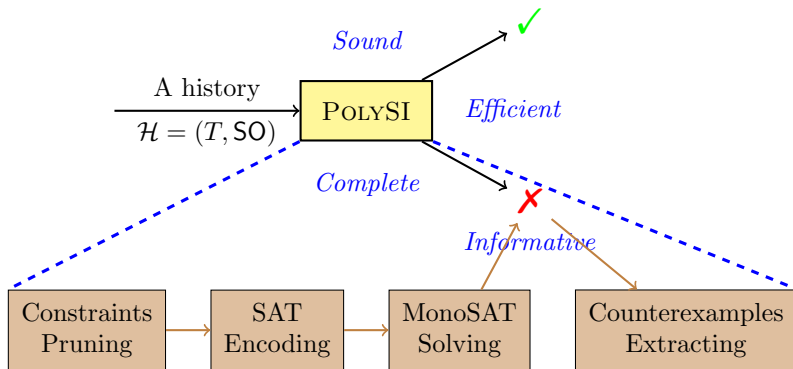
# Performance Evaluation: Pruning

POLYSI can effectively **prune** a huge number of constraints.

Benchmark	#cons.	#cons.	#unk. dep.	#unk. dep.
	before P	after P	before P	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

# Conclusion










# 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 [He, Sun, and Fan, 2021].



Hengfeng Wei (hfwei@nju.edu.cn)

-  Adya, Atul (1999). “Weak Consistency: A Generalized Theory and Optimistic Implementations for Distributed Transactions”. PhD thesis. USA.
-  Bayless, Sam, Noah Bayless, Holger H. Hoos, and Alan J. Hu (2015). “SAT modulo Monotonic Theories”. In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. AAAI’15. AAAI Press, pp. 3702–3709. ISBN: 0262511290.
-  Biswas, Ranadeep and Constantin Enea (Oct. 2019). “On the Complexity of Checking Transactional Consistency”. In: *Proc. ACM Program. Lang.* 3.OOPSLA. DOI: 10.1145/3360591. URL: <https://doi.org/10.1145/3360591>.
-  Cerone, Andrea and Alexey Gotsman (Jan. 2018). “Analysing Snapshot Isolation”. In: *J. ACM* 65.2. ISSN: 0004-5411. DOI: 10.1145/3152396. URL: <https://doi.org/10.1145/3152396>.
-  Darnell, Ben (Accessed February 14, 2023). *Lessons Learned from 2+ Years of Nightly Jepsen Tests*. <https://www.cockroachlabs.com/blog/jepsen-tests-lessons/>.

-  He, Fei, Zhihang Sun, and Hongyu Fan (2021). “Satisfiability modulo Ordering Consistency Theory for Multi-Threaded Program Verification”. In: *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. PLDI 2021. Virtual, Canada: Association for Computing Machinery, pp. 1264–1279. ISBN: 9781450383912. DOI: 10.1145/3453483.3454108. URL: <https://doi.org/10.1145/3453483.3454108>.
-  Jepsen (Accessed February 14, 2023). *Issue #824*. <https://github.com/YugaByte/yugabyte-db/issues/824>.
-  Kingsbury, Kyle and Peter Alvaro (Nov. 2020). “Elle: Inferring Isolation Anomalies from Experimental Observations”. In: *Proc. VLDB Endow.* 14.3, pp. 268–280. ISSN: 2150-8097.
-  Tan, Cheng, Changgeng Zhao, Shuai Mu, and Michael Walfish (2020). “COBRA: Making Transactional Key-Value Stores Verifiably Serializable”. In: *OSDI’20*. ISBN: 978-1-939133-19-9.