

# Efficient Black-box Checking of Snapshot Isolation in Databases

(Conference VLDB'2024)

Hengfeng Wei

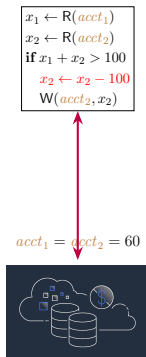
hfwei@nju.edu.cn

August 21, 2023



# Transaction and Isolation Level

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



# Transaction and Isolation Level

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

```
x1 ← R(acct1)
x2 ← R(acct2)
if x1 + x2 > 100
  x1 ← x1 - 100
  W(acct1, x1)
```

```
x1 ← R(acct1)
x2 ← R(acct2)
if x1 + x2 > 100
  x2 ← x2 - 100
  W(acct2, x2)
```

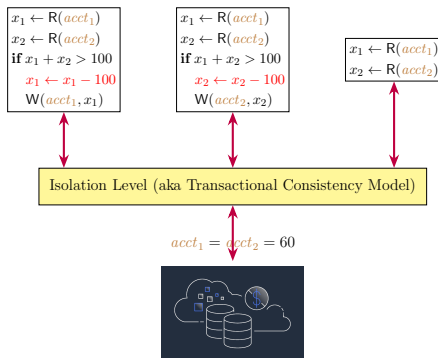
```
x1 ← R(acct1)
x2 ← R(acct2)
```

$acct_1 = acct_2 = 60$



# Transaction and Isolation Level

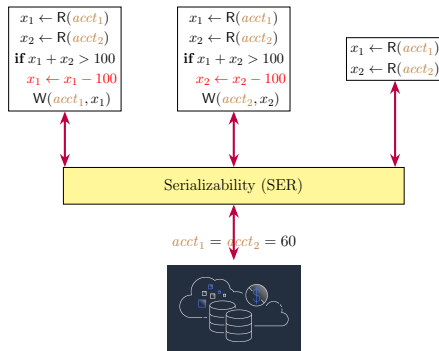
A transaction is a *group* of operations that is executed *atomically*.



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

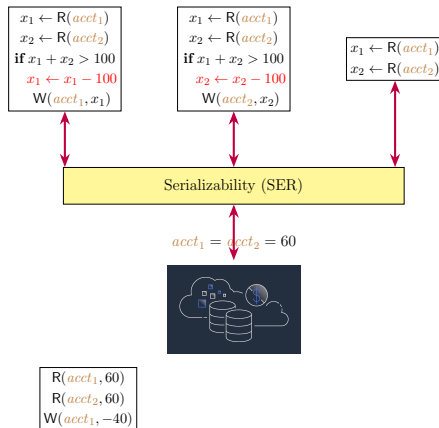
# Serializability (SER)

All transactions appear to execute in some total order.



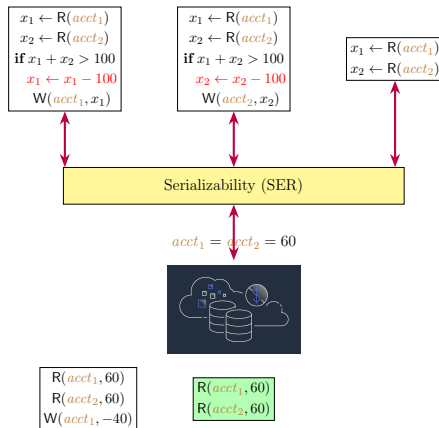
# Serializability (SER)

All transactions appear to execute in some total order.



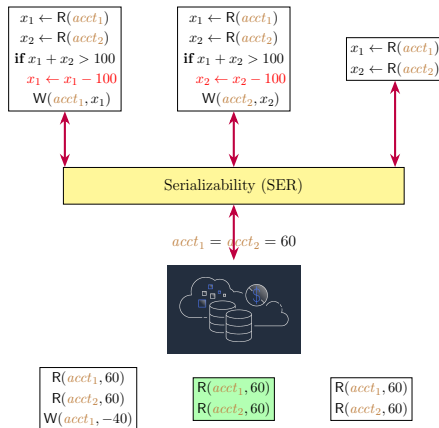
# Serializability (SER)

All transactions appear to execute in some total order.



# Serializability (SER)

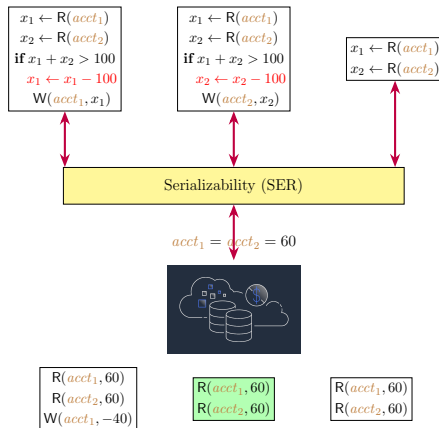
All transactions appear to execute in some total order.





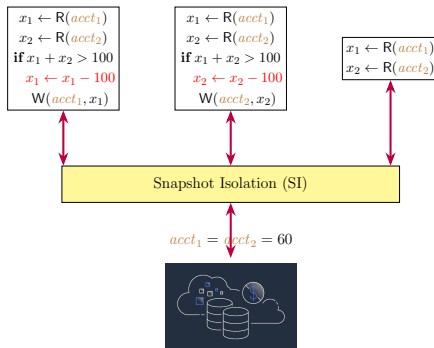
# Serializability (SER)

All transactions appear to execute in some total order.

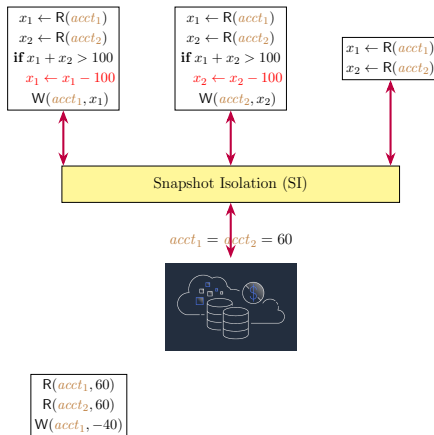


too expensive, especially for distributed transactions

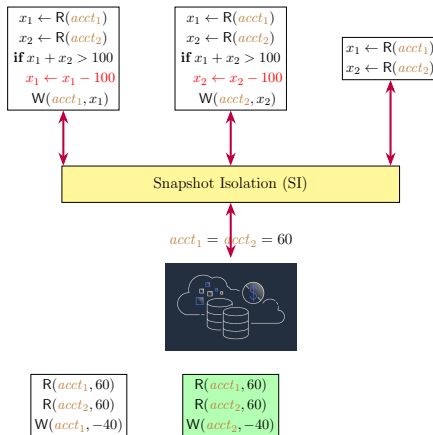
# Snapshot Isolation (SI)



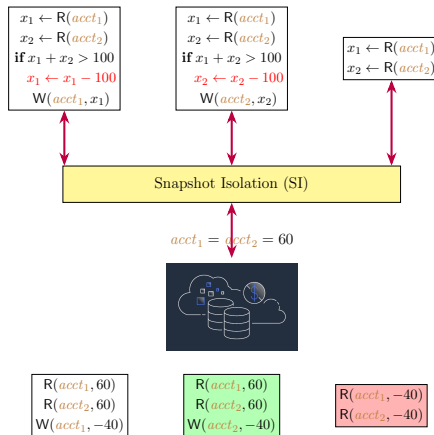
# Snapshot Isolation (SI)



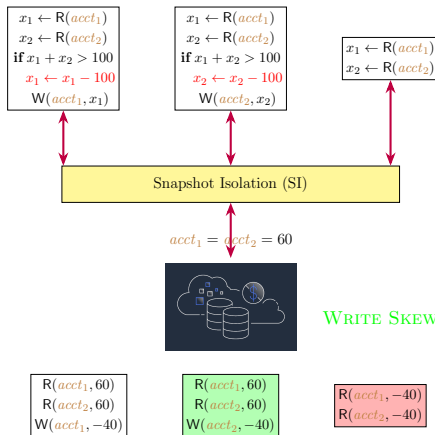
# Snapshot Isolation (SI)



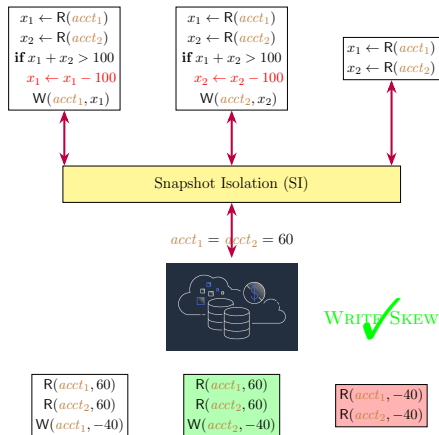
# Snapshot Isolation (SI)



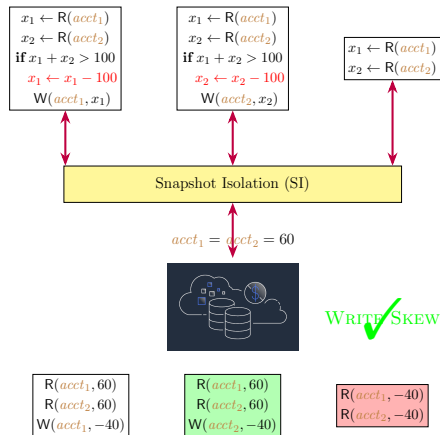
# Snapshot Isolation (SI)



# Snapshot Isolation (SI)



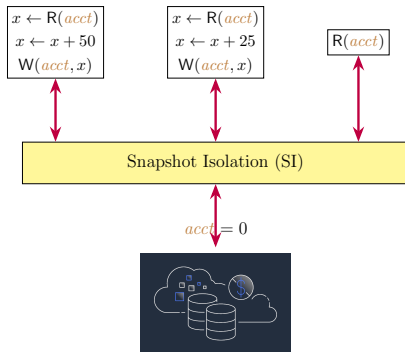
# Snapshot Isolation (SI)



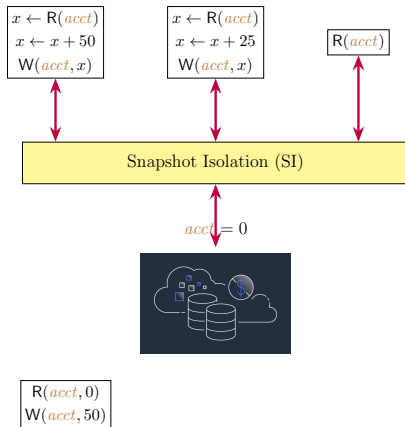
**Snapshot Read:** Each transaction reads data from a *snapshot* of committed data valid as of the (logical) time the transaction started.



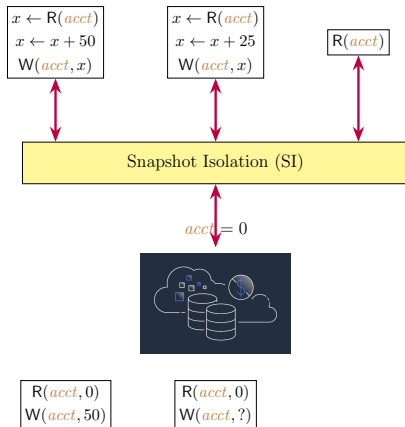
# Snapshot Isolation (SI)



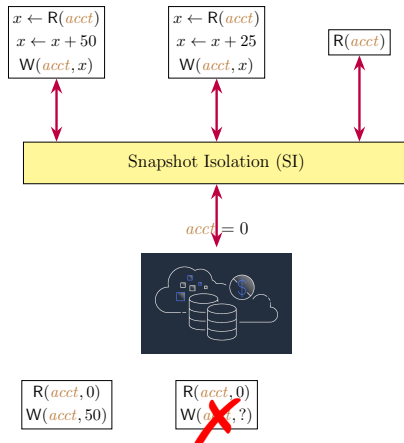
# Snapshot Isolation (SI)



# Snapshot Isolation (SI)

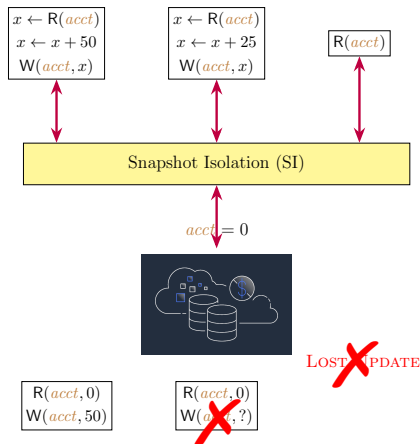


# Snapshot Isolation (SI)



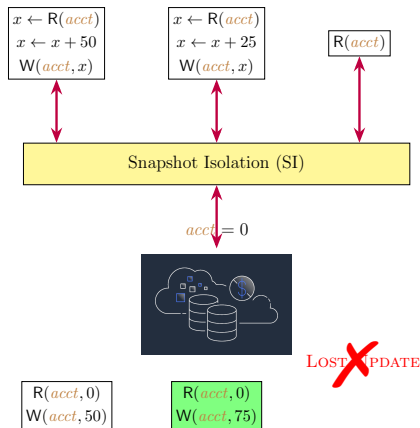
**Snapshot Write:** Concurrent transactions cannot write to the same key. One of them must be aborted.

# Snapshot Isolation (SI)



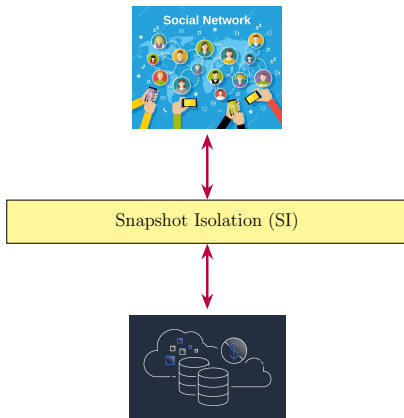
**Snapshot Write:** Concurrent transactions cannot write to the same key. One of them must be aborted.

# Snapshot Isolation (SI)

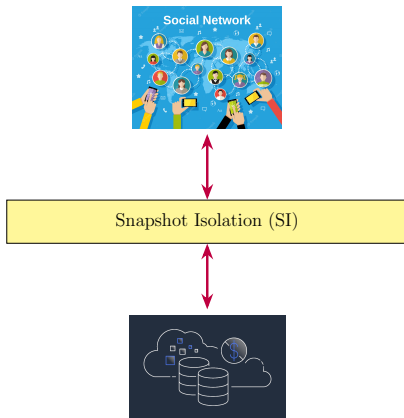


**Snapshot Write:** Concurrent transactions cannot write to the same key. One of them must be aborted.

# Snapshot Isolation (SI)



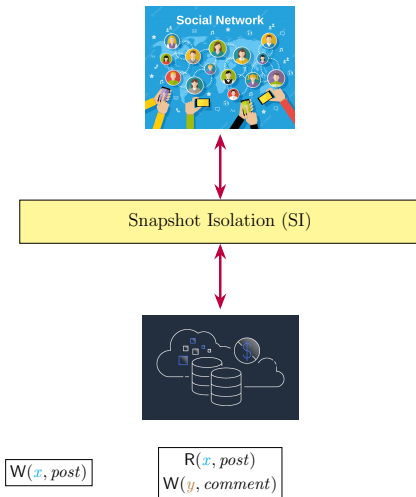
# Snapshot Isolation (SI)



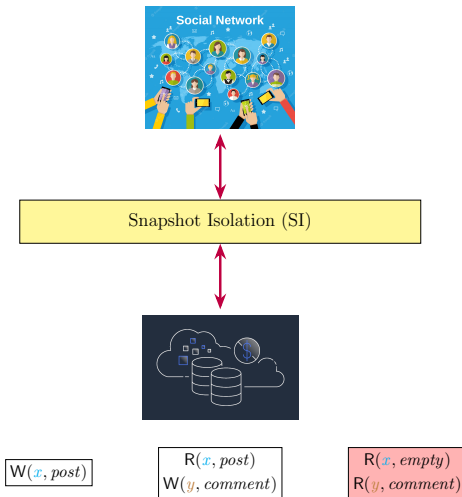
$W(x, post)$



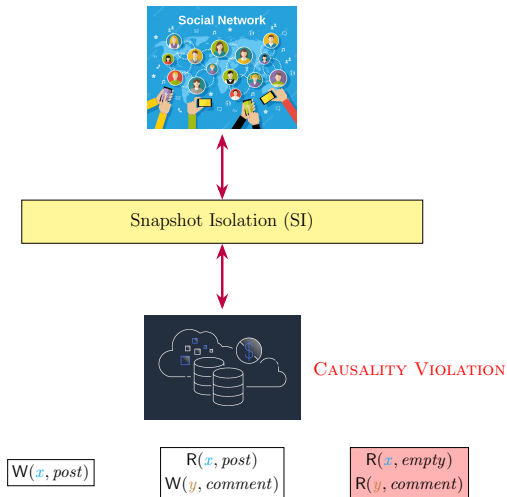
# Snapshot Isolation (SI)



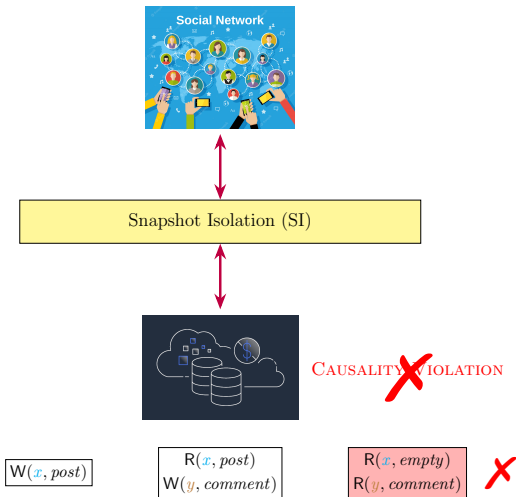
# Snapshot Isolation (SI)



# Snapshot Isolation (SI)



# Snapshot Isolation (SI)



# Databases and Snapshot Isolation

database logos

Many databases claim to support SI.

# Databases and Snapshot Isolation

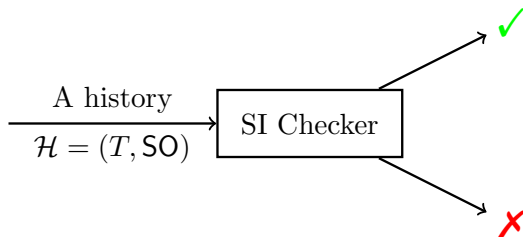
+papers

Databases may fail to provide SI as they claim.

# The SI Checking Problem

## Definition (The SI Checking Problem)

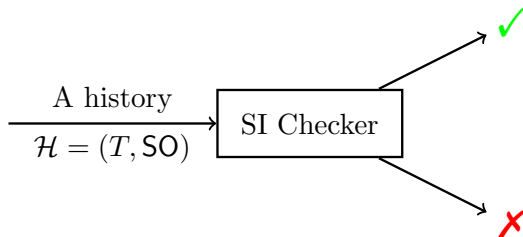
The SI checking problem is the **decision problem** of determining whether a given **history**  $\mathcal{H} = (T, SO)$  satisfies SI?



# The SI Checking Problem

## Definition (The SI Checking Problem)

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



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



# The SI Checking Problem

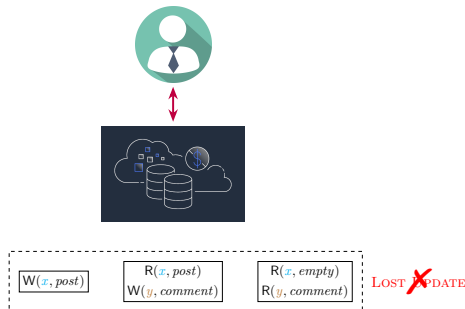
*Black-box checking:* do not rely on database internals



The histories are collected from database logs.

# The SI Checking Problem

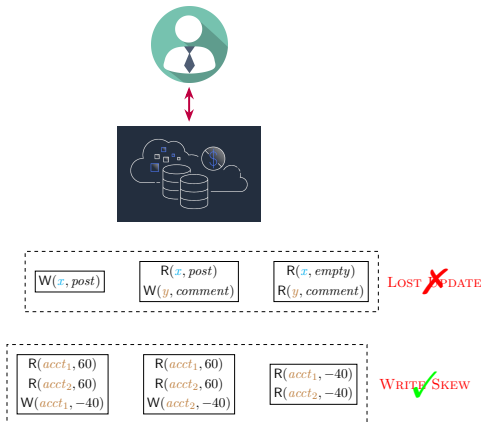
*Black-box checking:* do not rely on database internals



The histories are collected from database logs.

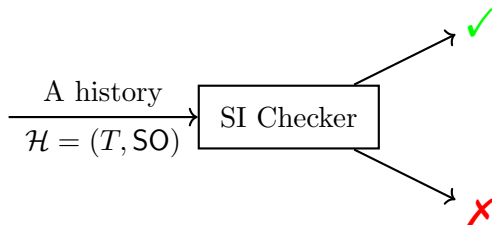
# The SI Checking Problem

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

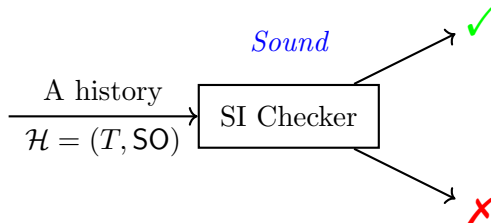


The histories are collected from database logs.

# The SI Checking Problem

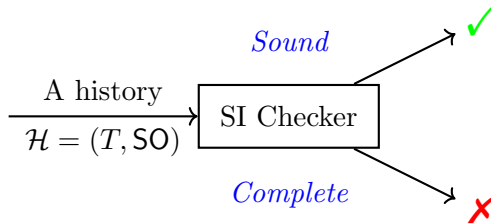


# The SI Checking Problem



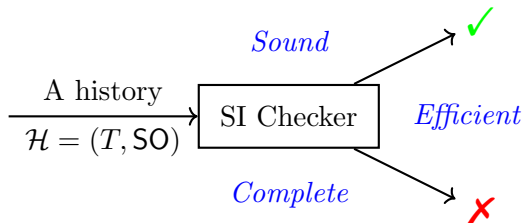
*Sound:* If the checker says **X**, then the history does *not* satisfy SI.

# The SI Checking Problem



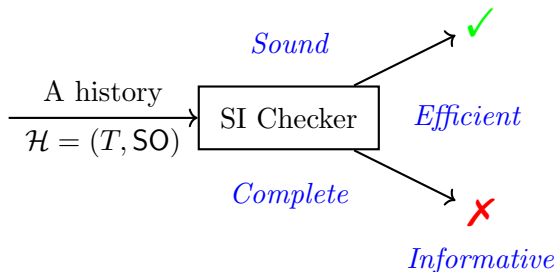
*Complete:* If the checker says ✓, then the history *satisfies* SI.

# The SI Checking Problem



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

# The SI Checking Problem



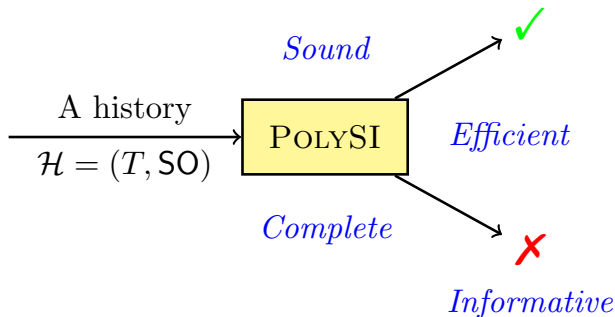
*Informative:* The checker should provide understandable *counterexamples* if it says **X**.



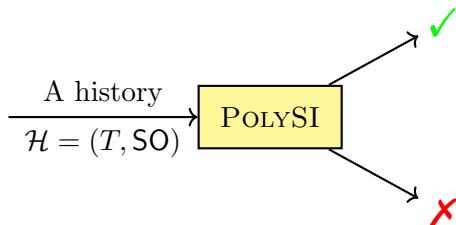
# The SI Checking Problem

related-work

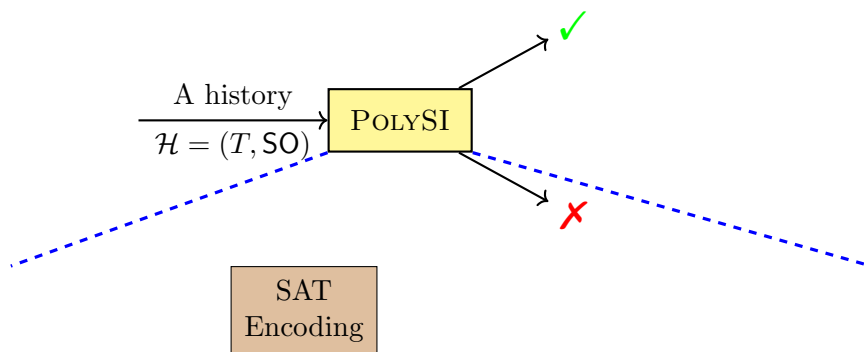
# Contribution: the POLYSI Checker



# Contribution: the POLYSI Checker

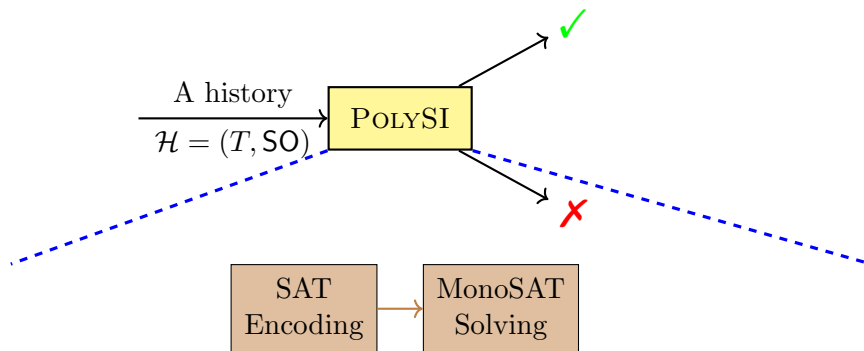


# Contribution: the POLYSI Checker



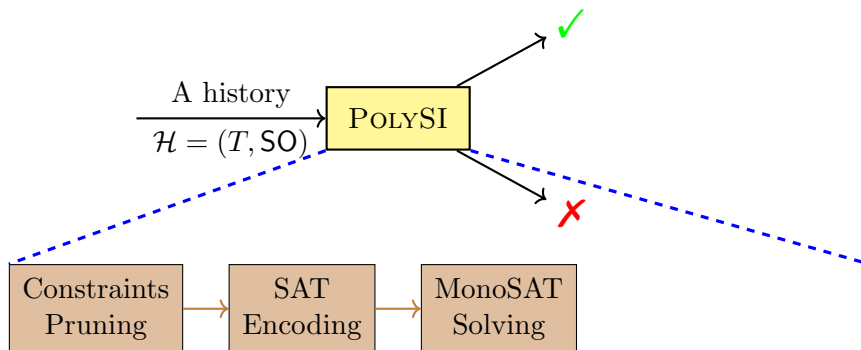
*Sound & Complete:* polygraph-based characterization of SI

# Contribution: the POLYSI Checker



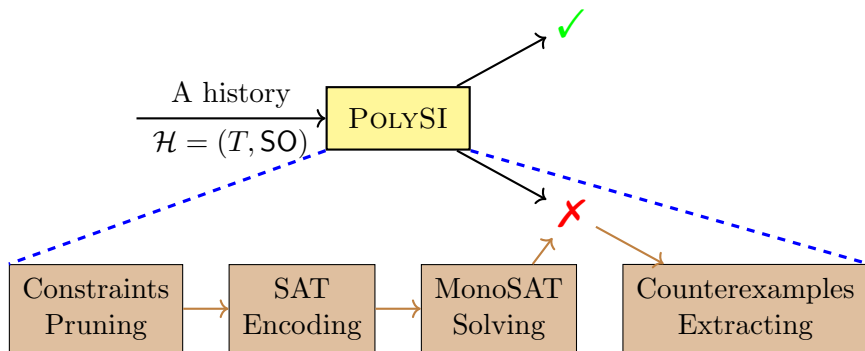
*Efficient:* utilizing MonoSAT solver optimized for graph problems

# Contribution: the POLYSI Checker



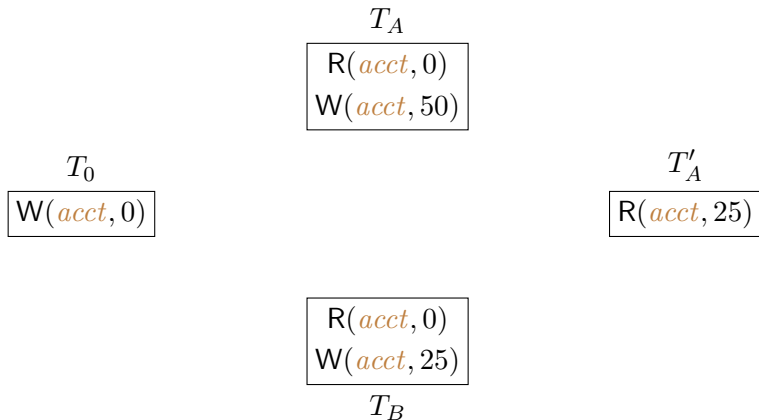
*Efficient:* domain-specific pruning before encoding

# Contribution: the POLYSI Checker



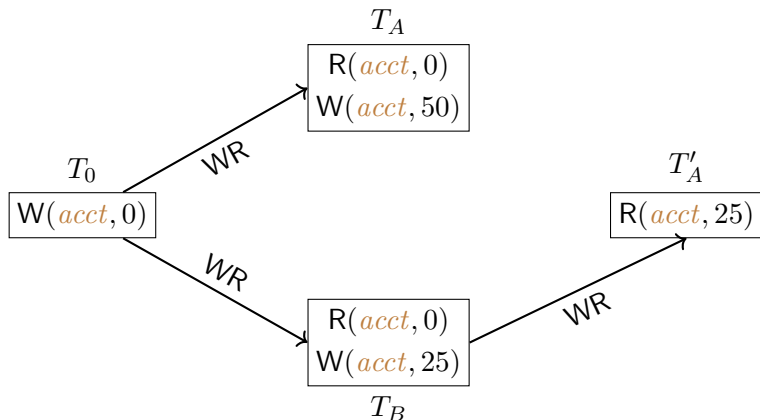
*Informative:* extract counterexamples from the unsatisfiable core

# Dependency Graph-based Characterization of SI



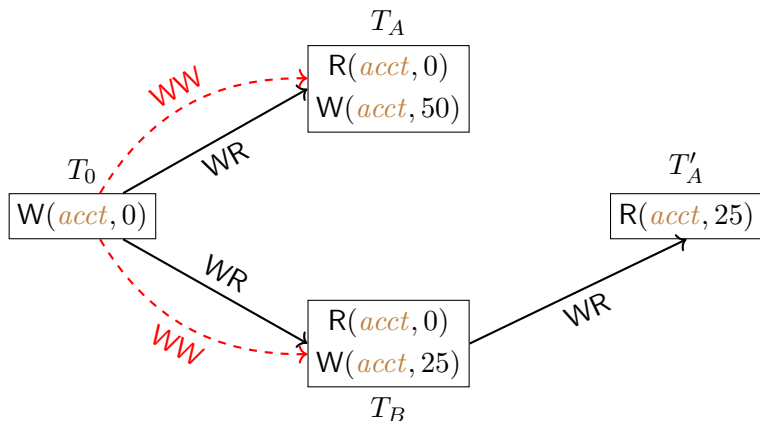


# Dependency Graph-based Characterization of SI



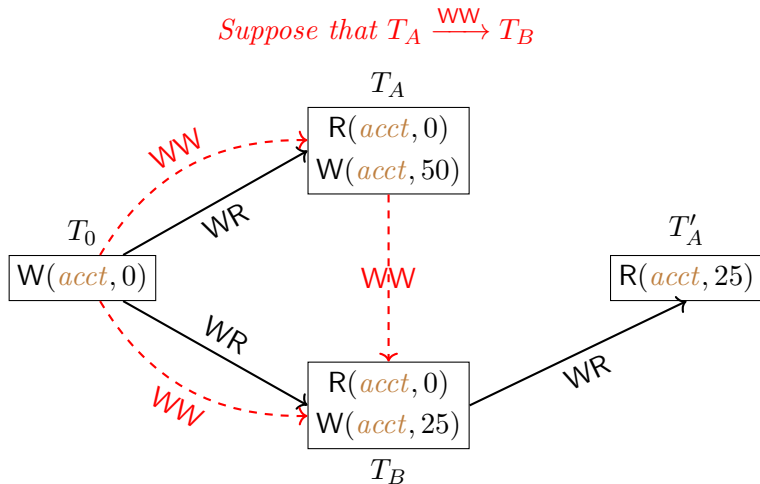
WR: “write-read” dependency capturing the “read-from” relation

# Dependency Graph-based Characterization of SI



WW: “write-write” dependency capturing the version order

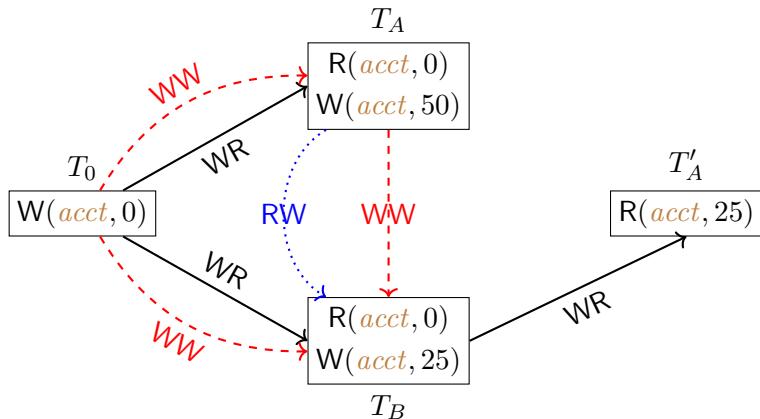
# Dependency Graph-based Characterization of SI



WW: “write-write” dependency capturing the version order

# Dependency Graph-based Characterization of SI

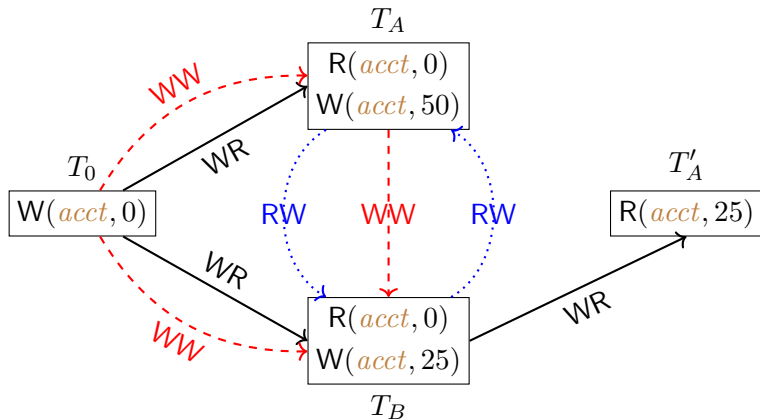
$$T_0 \xrightarrow{WR} T_A \wedge T_0 \xrightarrow{WW} T_B \implies T_A \xrightarrow{RW} T_B$$



RW: “read-write” dependency capturing the overwritten relation

# Dependency Graph-based Characterization of SI

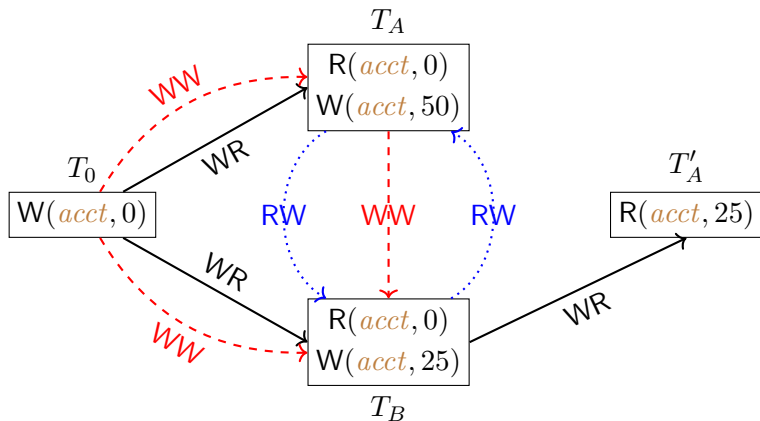
$$T_0 \xrightarrow{WR} T_B \wedge T_0 \xrightarrow{WW} T_A \implies T_A \xrightarrow{RW} T_A$$



RW: “read-write” dependency capturing the overwritten relation

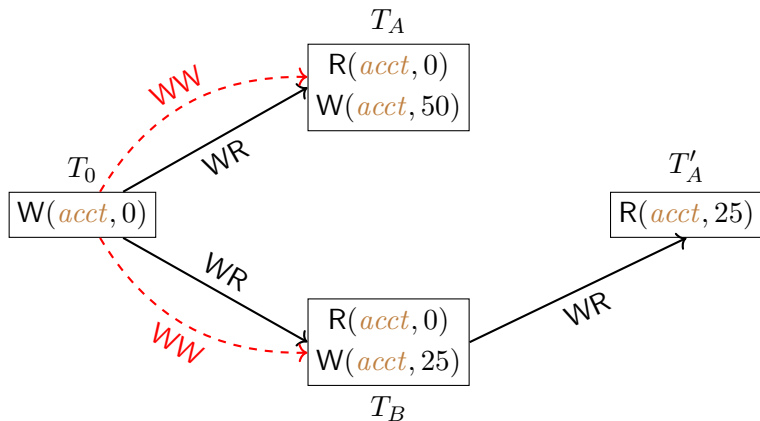
# Dependency Graph-based Characterization of SI

Suppose that  $T_A \xrightarrow{WW} T_B$



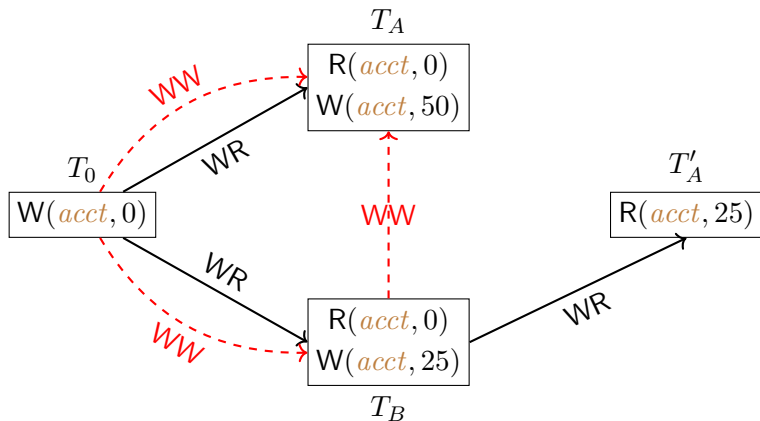
undesired cycle:  $T_A \xrightarrow{WW} T_B \xrightarrow{RW} T_A$

# Dependency Graph-based Characterization of SI



# Dependency Graph-based Characterization of SI

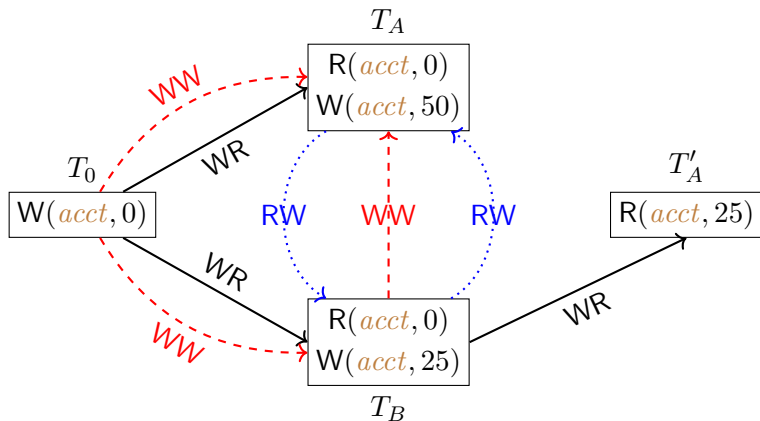
Suppose that  $T_B \xrightarrow{WW} T_A$





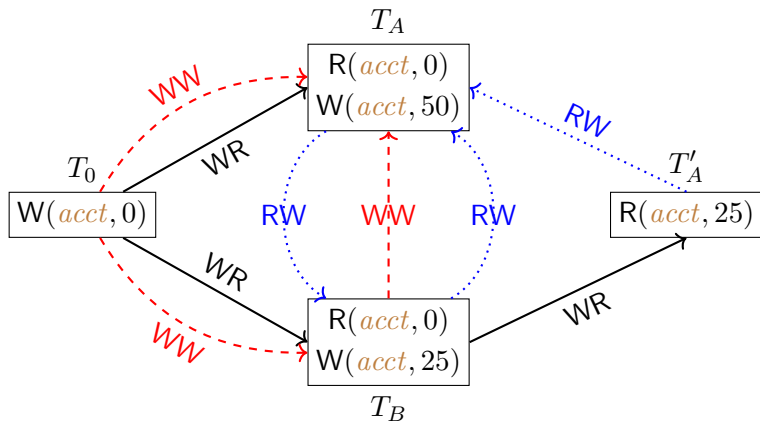
# Dependency Graph-based Characterization of SI

Suppose that  $T_B \xrightarrow{WW} T_A$



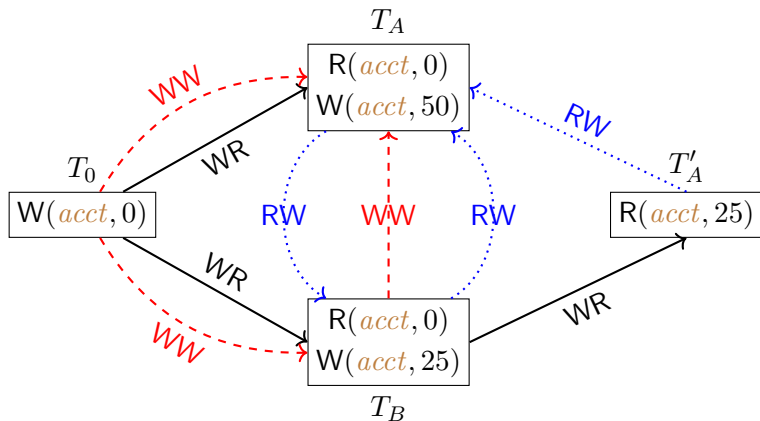
# Dependency Graph-based Characterization of SI

Suppose that  $T_B \xrightarrow{WW} T_A$



# Dependency Graph-based Characterization of SI

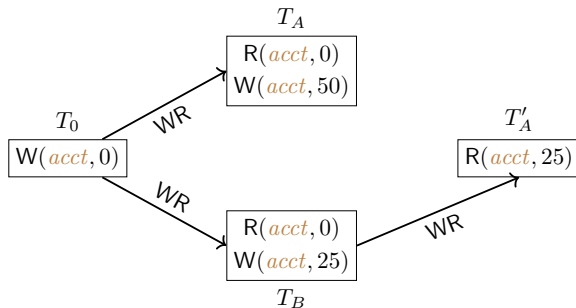
Suppose that  $T_B \xrightarrow{WW} T_A$



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

# Dependency Graph-based Characterization of SI

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



Either case leads to an undesired cycle.

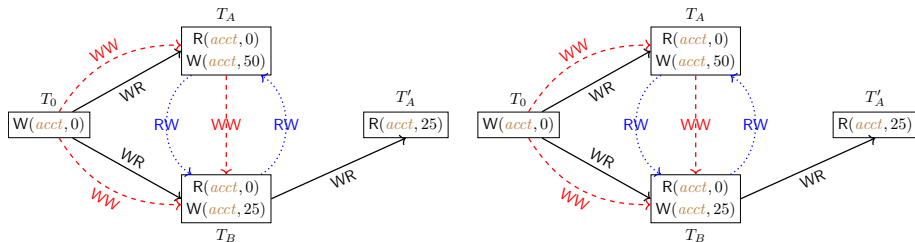
Therefore, it does not satisfy SI.


# 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.

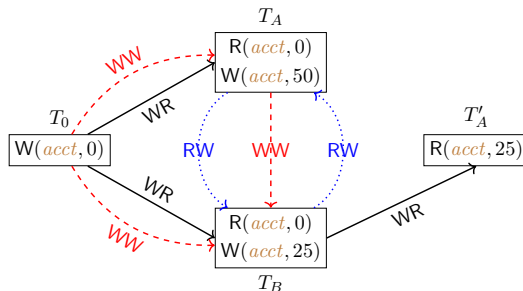
# Dependency Graph-based Characterization of SI

Theorem (Theorem 4.1 of [AnalysingSI:JACM2018])

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

$$\mathcal{H} \models \text{SI} \iff \mathcal{H} \models \text{INT} \wedge$$

$$\exists \text{WR, WW, RW. } \mathcal{G} = (\mathcal{H}, \text{WR, WW, RW}) \wedge \\ (((SO_{\mathcal{G}} \cup \text{WR}_{\mathcal{G}} \cup \text{WW}_{\mathcal{G}}) ; \text{RW}_{\mathcal{G}}?) \text{ is acyclic}).$$



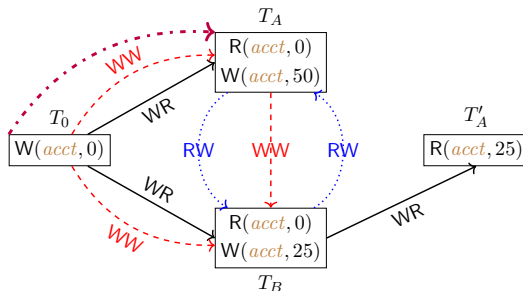
# Dependency Graph-based Characterization of SI

Theorem (Theorem 4.1 of [AnalysingSI:JACM2018])

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

$$\mathcal{H} \models \text{SI} \iff \mathcal{H} \models \text{INT} \wedge$$

$$\exists \text{WR, WW, RW. } \mathcal{G} = (\mathcal{H}, \text{WR, WW, RW}) \wedge \\ (((SO_{\mathcal{G}} \cup \text{WR}_{\mathcal{G}} \cup \text{WW}_{\mathcal{G}}) ; \text{RW}_{\mathcal{G}}?) \text{ is acyclic}).$$





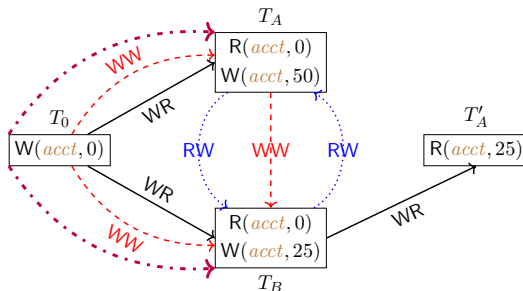
# Dependency Graph-based Characterization of SI

Theorem (Theorem 4.1 of [AnalysingSI:JACM2018])

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

$$\mathcal{H} \models \text{SI} \iff \mathcal{H} \models \text{INT} \wedge$$

$$\exists \text{WR, WW, RW. } \mathcal{G} = (\mathcal{H}, \text{WR, WW, RW}) \wedge \\ (((\text{SO}_{\mathcal{G}} \cup \text{WR}_{\mathcal{G}} \cup \text{WW}_{\mathcal{G}}) ; \text{RW}_{\mathcal{G}}?) \text{ is acyclic}).$$



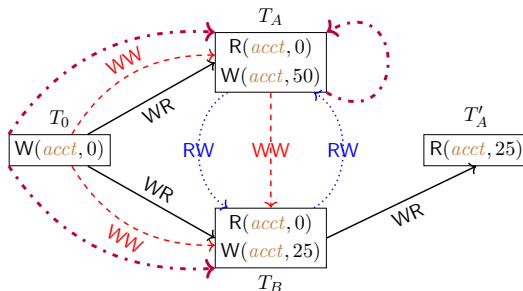
# Dependency Graph-based Characterization of SI

Theorem (Theorem 4.1 of [AnalysingSI:JACM2018])

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

$$\mathcal{H} \models \text{SI} \iff \mathcal{H} \models \text{INT} \wedge$$

$$\exists \text{WR, WW, RW. } \mathcal{G} = (\mathcal{H}, \text{WR, WW, RW}) \wedge \\ (((\text{SO}_{\mathcal{G}} \cup \text{WR}_{\mathcal{G}} \cup \text{WW}_{\mathcal{G}}) ; \text{RW}_{\mathcal{G}}?) \text{ is acyclic}).$$



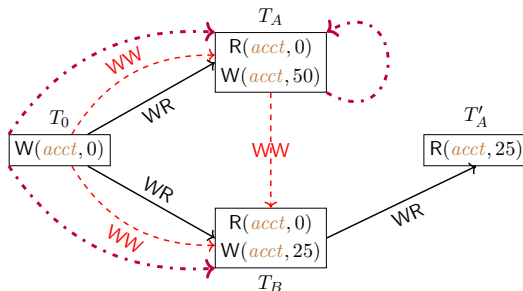
# Dependency Graph-based Characterization of SI

Theorem (Theorem 4.1 of [AnalysingSI:JACM2018])

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

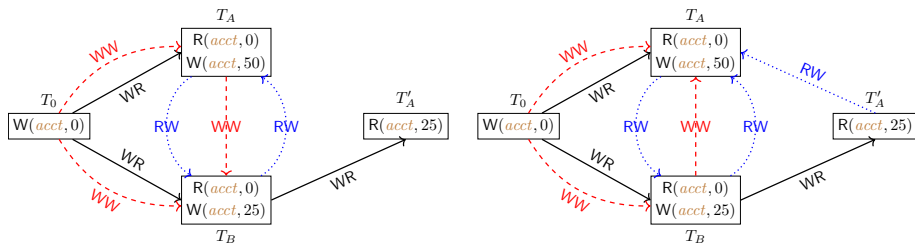
$$\mathcal{H} \models \text{SI} \iff \mathcal{H} \models \text{INT} \wedge$$

$$\exists \text{WR, WW, RW. } \mathcal{G} = (\mathcal{H}, \text{WR, WW, RW}) \wedge \\ (((\text{SO}_{\mathcal{G}} \cup \text{WR}_{\mathcal{G}} \cup \text{WW}_{\mathcal{G}}) ; \text{RW}_{\mathcal{G}}?) \text{ is acyclic}).$$



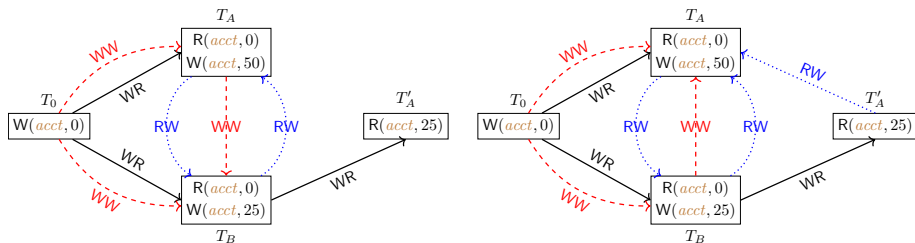
# Dependency Graph-based Characterization of SI

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



# Dependency Graph-based Characterization of SI

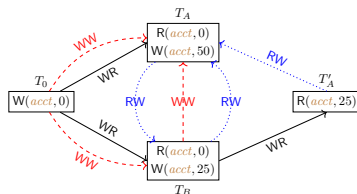
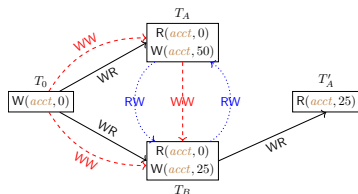
$\mathcal{Q}$  : How to capture and resolve all possible WW dependencies?



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

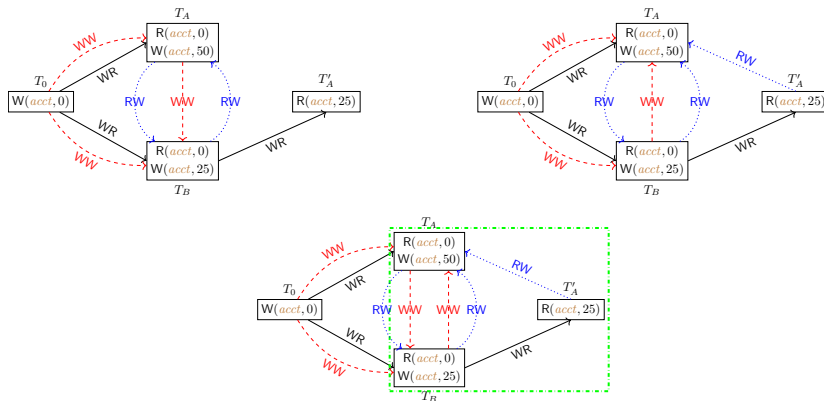
# Polygraphs: A Family of Dependency Graphs

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



# Polygraphs: A Family of Dependency Graphs

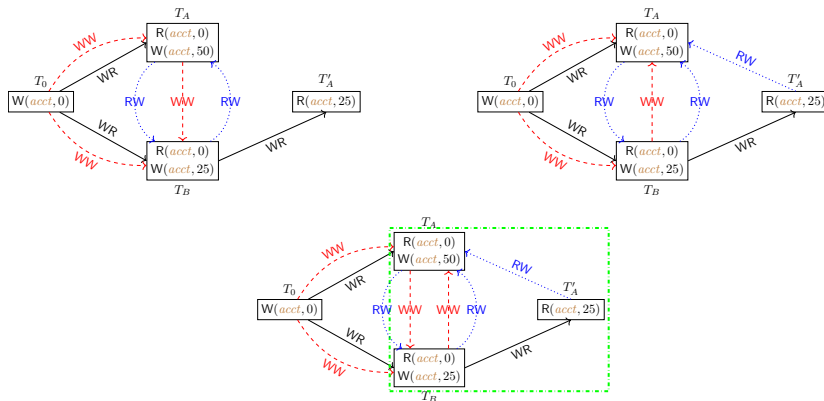
Consider the two cases of WW dependencies between  $T_A$  and  $T_B$ .



generalized polygraph:

# Polygraphs: A Family of Dependency Graphs

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

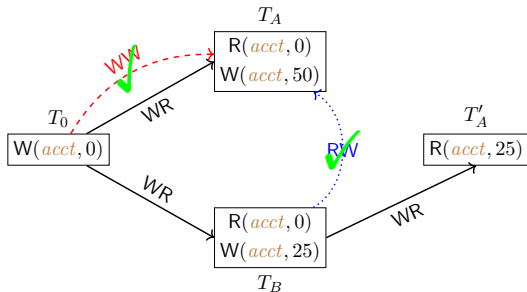
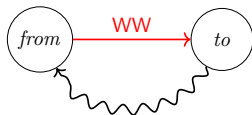


generalized polygraph:

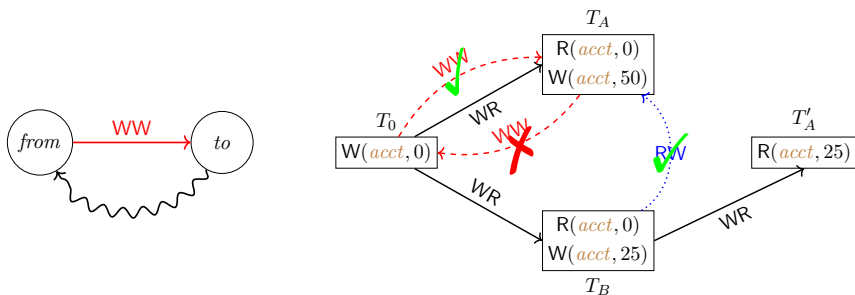
$$\langle \text{either} \triangleq \{T_A \xrightarrow{WW} T_B\}, \text{or} \triangleq \{T_B \xrightarrow{WW} T_A, T'_A \xrightarrow{RW} T_A\} \rangle \equiv$$



# POLYSI: Pruning before Encoding (the WW case)

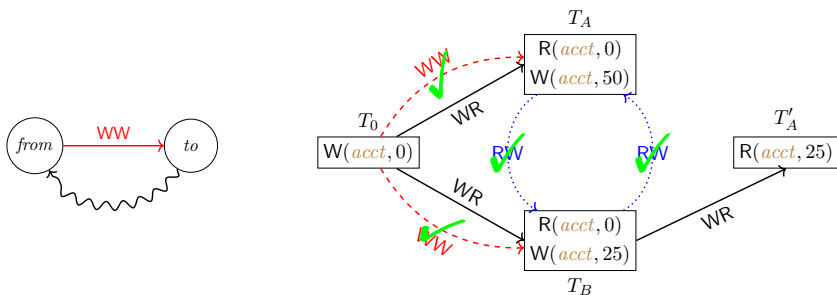


# POLYSI: Pruning before Encoding (the WW case)

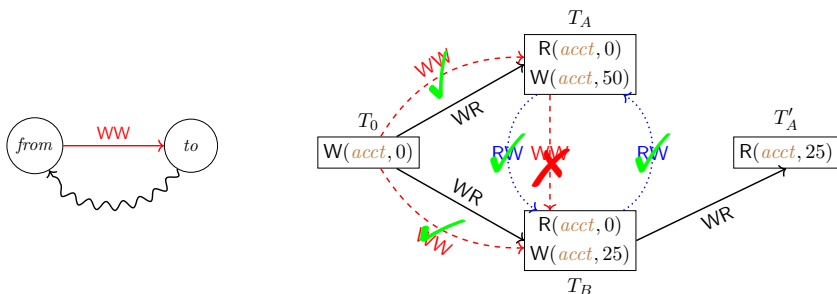


$T_A \xrightarrow{WW} T_0$  can be pruned due to the  $T_A \xrightarrow{WW} T_0 \xrightarrow{WR} T_A$  cycle.

# POLYSI: Pruning before Encoding (the WW case)

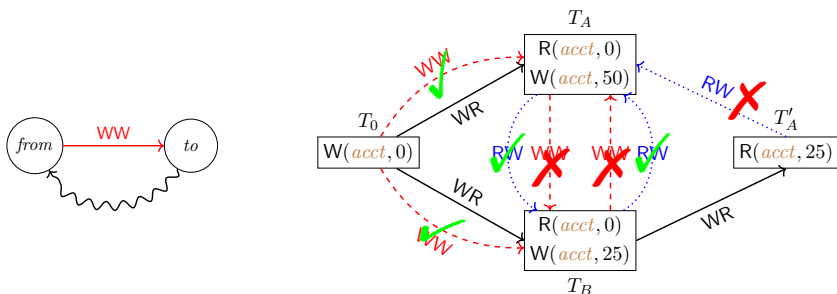


# POLYSI: Pruning before Encoding (the WW case)



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

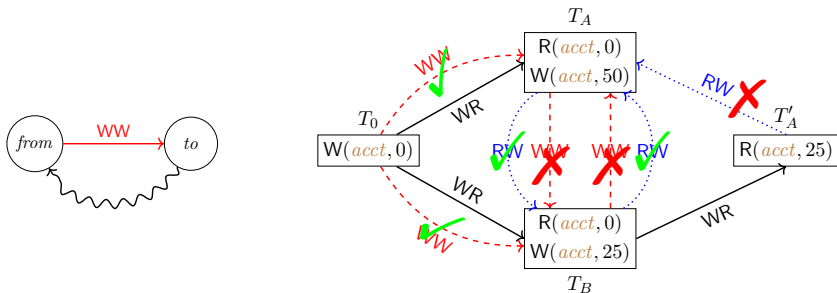
# POLYSI: Pruning before Encoding (the WW case)



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

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

### POLYSI: Pruning before Encoding (the WW case)



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

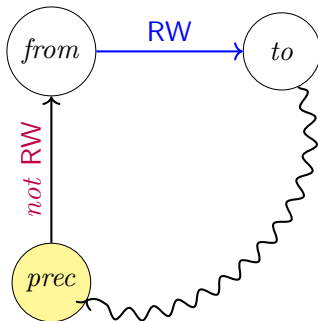
$T_B \xrightarrow{\text{WW}} T_A$  is pruned due to the  $T_B \xrightarrow{\text{WW}} T_A \xrightarrow{\text{RW}} T_B$  cycle.

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

# POLYSI: Pruning before Encoding (the RW case)

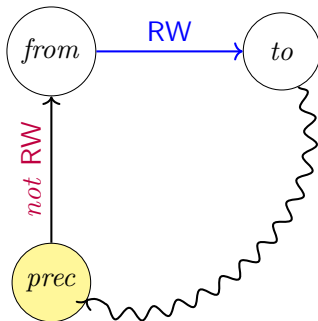


# POLYSI: Pruning before Encoding (the RW case)





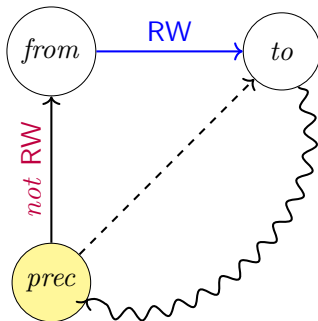
## POLYSI: Pruning before Encoding (the RW case)



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: 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”

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

# POLYSI: An Illustrating Example of “Long Fork”

$$T_1 \quad \boxed{W(\textcolor{teal}{x}, 1)}$$

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

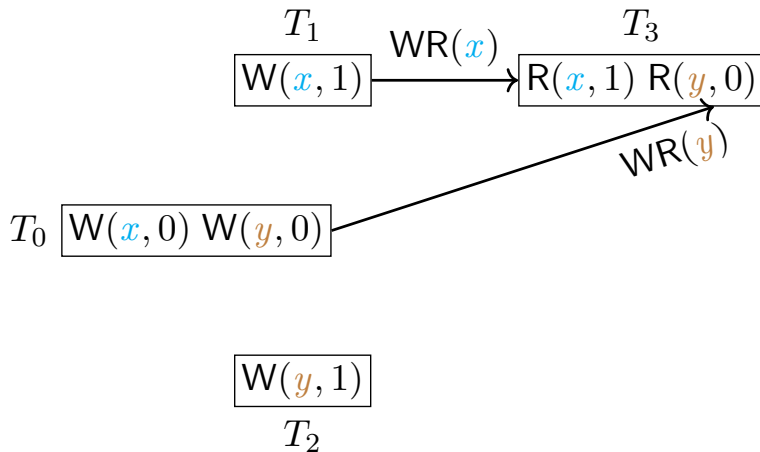
# POLYSI: An Illustrating Example of “Long Fork”

$$T_1$$
$$\boxed{W(x, 1)}$$

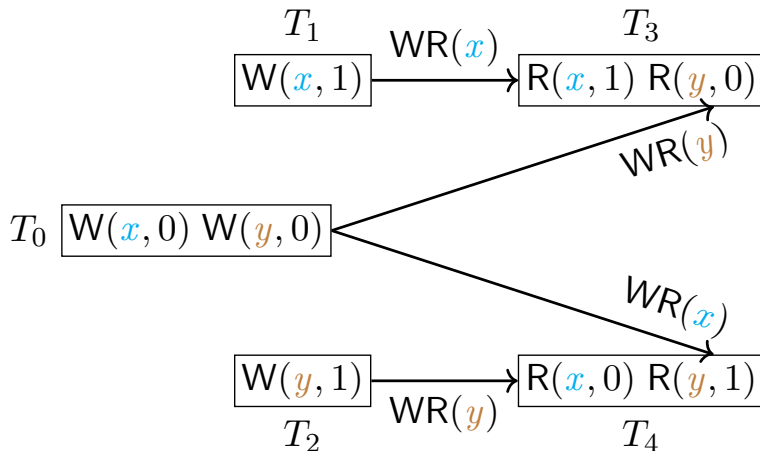
$$T_0 \quad \boxed{W(x, 0) \ W(y, 0)}$$

$$\boxed{W(y, 1)}$$
$$T_2$$

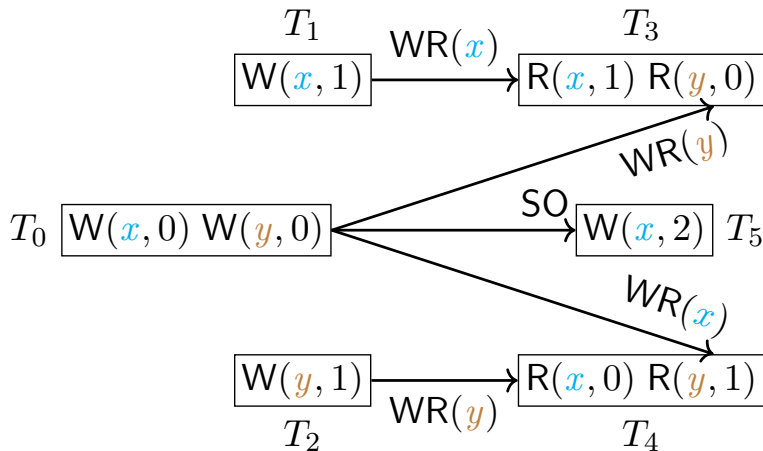
# POLYSI: An Illustrating Example of “Long Fork”



# POLYSI: An Illustrating Example of “Long Fork”



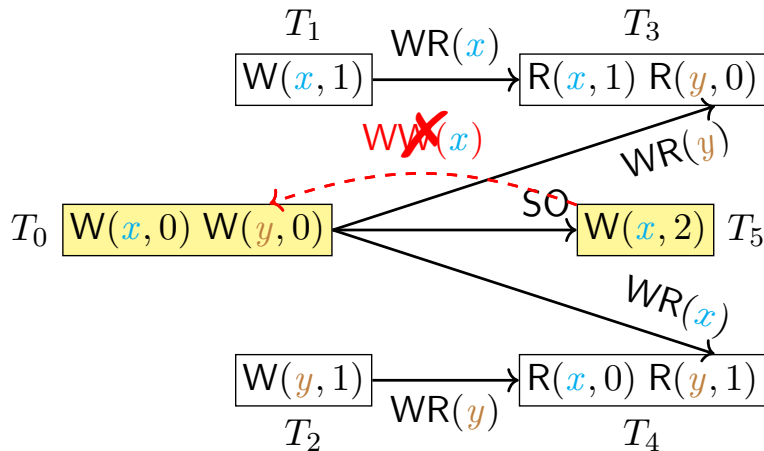
# POLYSI: An Illustrating Example of “Long Fork”



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

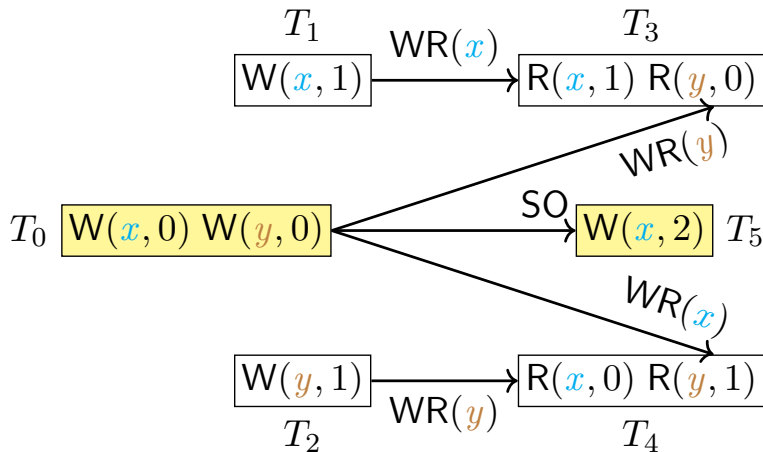


# POLYSI: An Illustrating Example of “Long Fork”

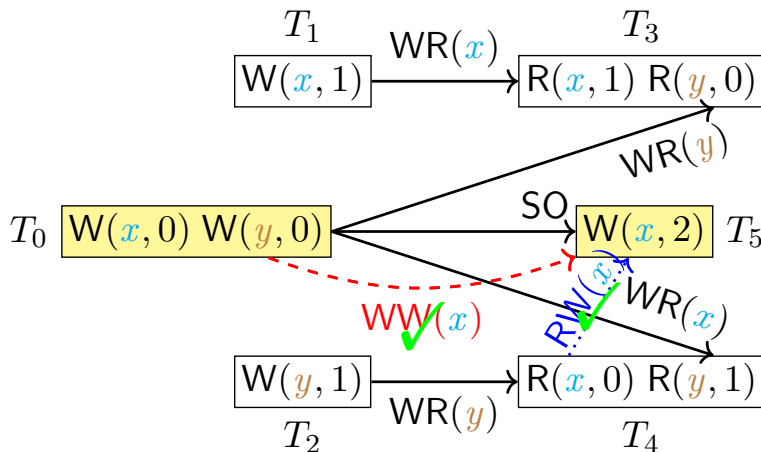


The  $T_5 \xrightarrow{WW(x)} T_0$  case is pruned due to  $T_0 \xrightarrow{SO} T_5 \xrightarrow{WW(x)} T_0$ .

# POLYSI: An Illustrating Example of “Long Fork”

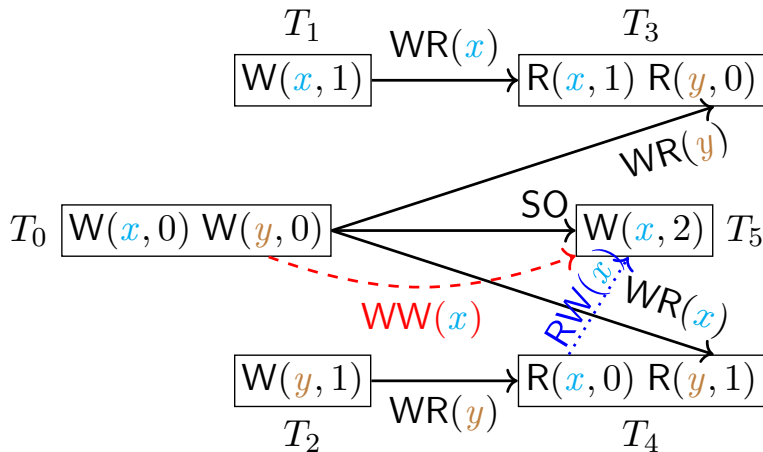


# POLYSI: An Illustrating Example of “Long Fork”

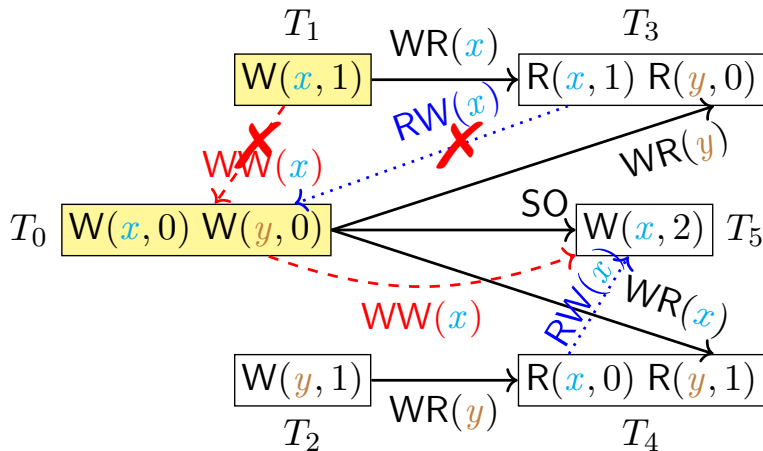


The  $T_0 \xrightarrow{WW(x)} T_5$  case becomes known.

# POLYSI: An Illustrating Example of “Long Fork”

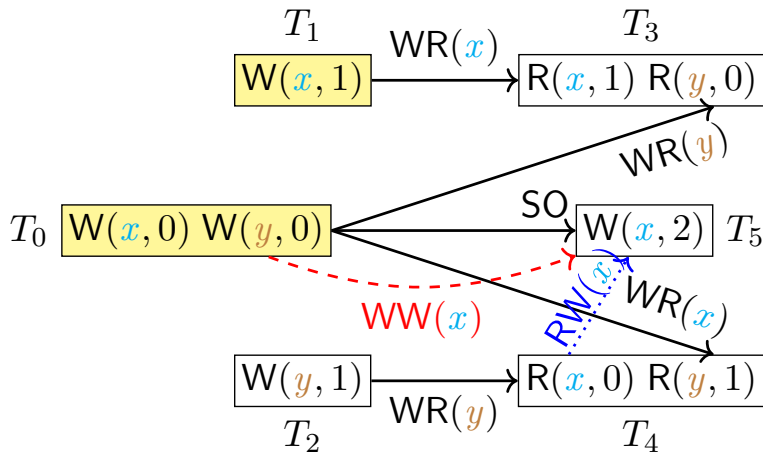


## POLYSI: An Illustrating Example of “Long Fork”

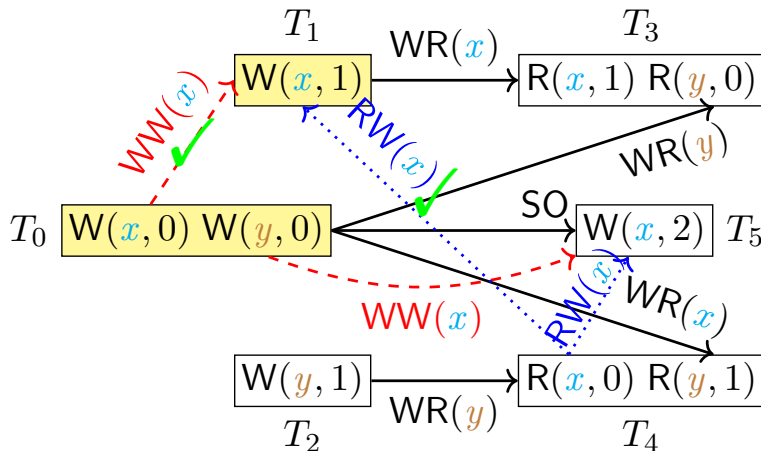


The  $T_1 \xrightarrow{\text{WW}(\textcolor{blue}{x})} T_0$  case is pruned due to  $T_3 \xrightarrow{\text{RW}(\textcolor{blue}{x})} T_0 \xrightarrow{\text{WR}(\textcolor{brown}{y})} T_3$ .

# POLYSI: An Illustrating Example of “Long Fork”

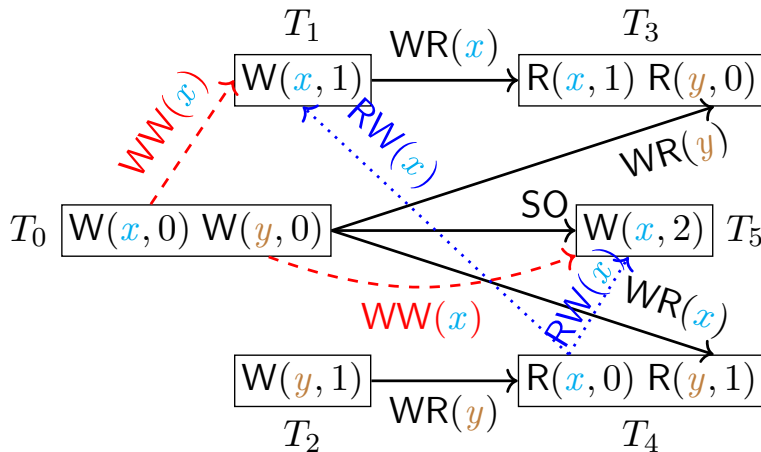


# POLYSI: An Illustrating Example of “Long Fork”



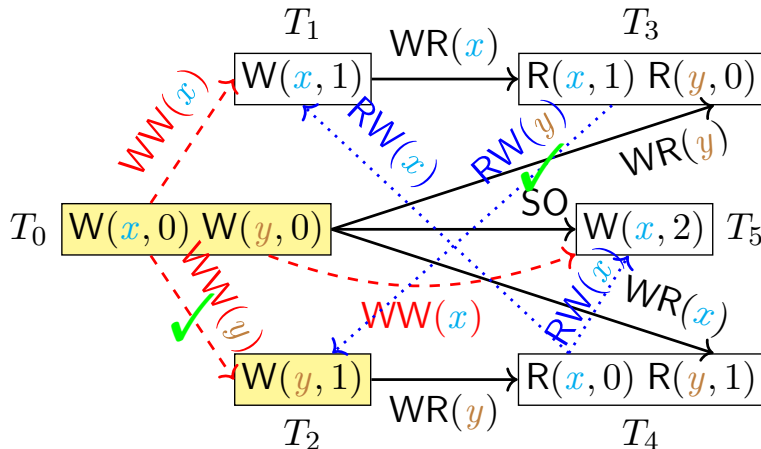
The  $T_0 \xrightarrow{WW(x)} T_1$  case becomes known.

# POLYSI: An Illustrating Example of “Long Fork”



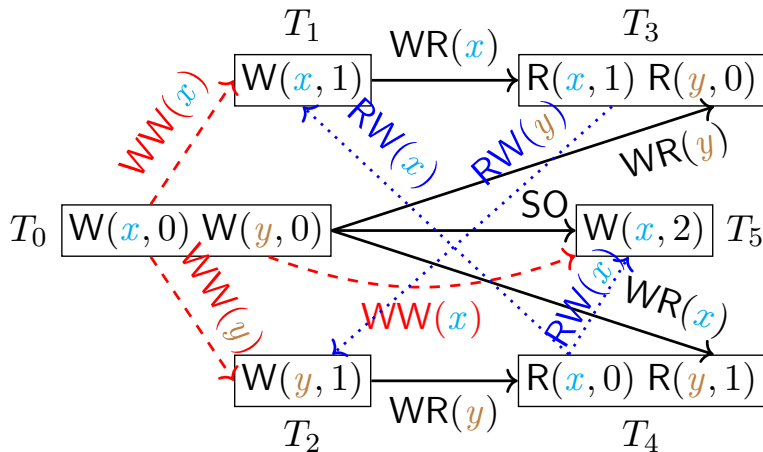


# POLYSI: An Illustrating Example of “Long Fork”

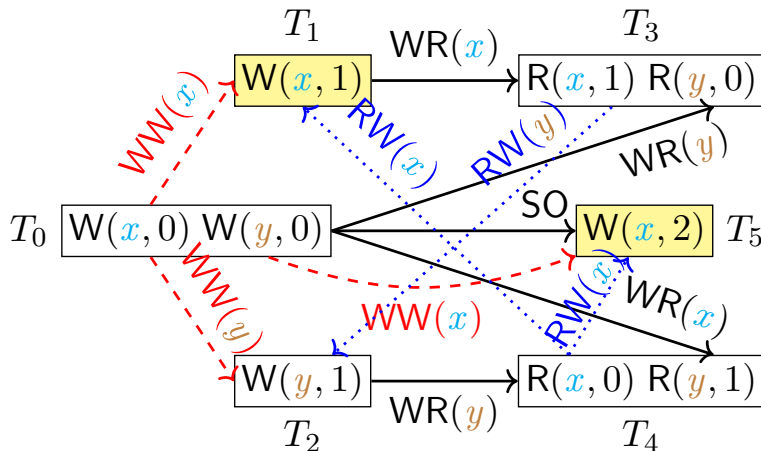


The  $T_2 \xrightarrow{WW(y)} T_0$  case is pruned,  
 while the  $T_0 \xrightarrow{WW(y)} T_2$  case becomes known.

# POLYSI: An Illustrating Example of “Long Fork”



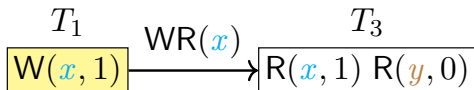
# POLYSI: An Illustrating Example of “Long Fork”



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

# POLYSI: An Illustrating Example of “Long Fork”

< , >



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

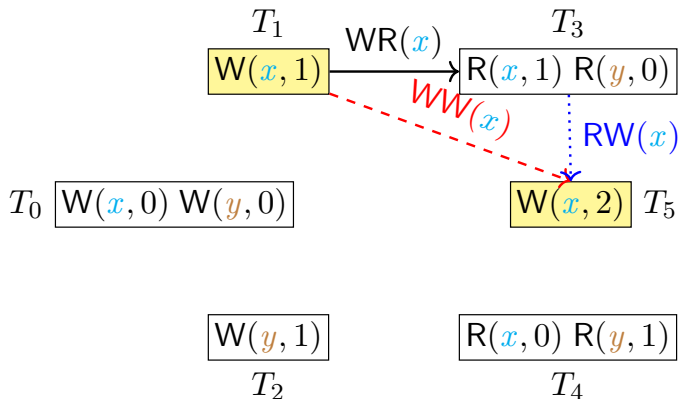
$\boxed{W(x, 2)}$   $T_5$

$\boxed{W(y, 1)}$   
 $T_2$

$\boxed{R(x, 0) \ R(y, 1)}$   
 $T_4$

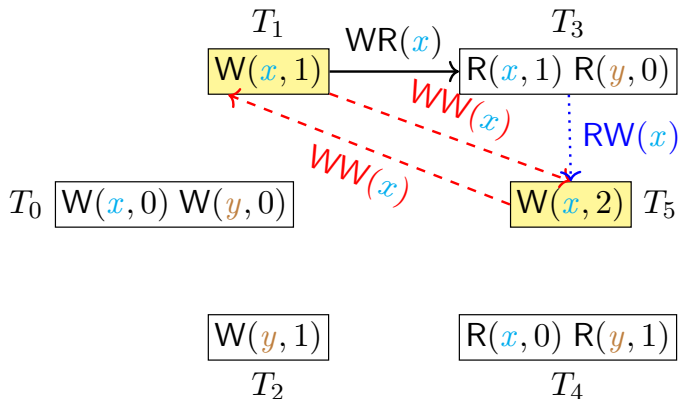
# POLYSI: An Illustrating Example of “Long Fork”

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



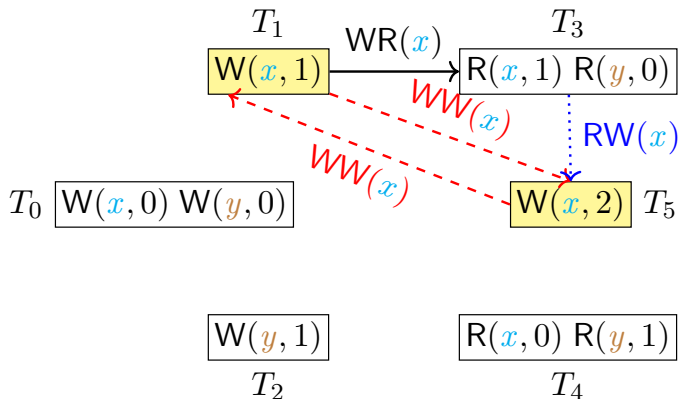
# POLYSI: An Illustrating Example of “Long Fork”

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



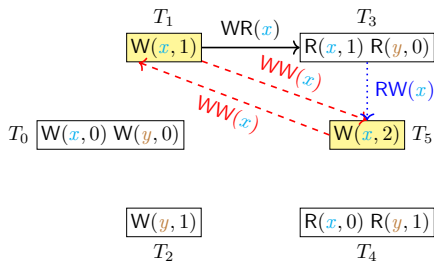
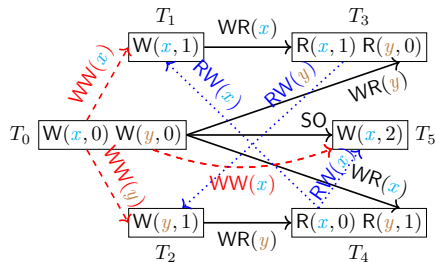
# POLYSI: An Illustrating Example of “Long Fork”

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



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

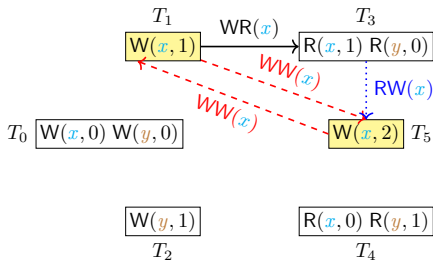
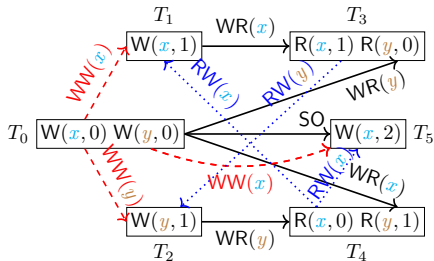
# POLYSI: An Illustrating Example of “Long Fork”





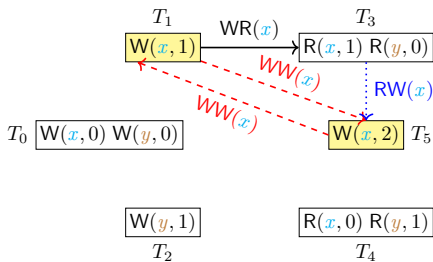
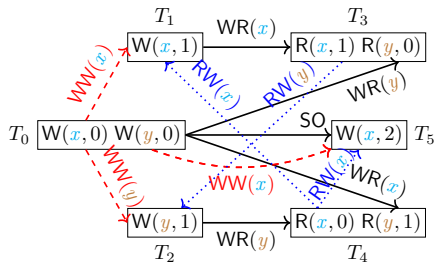
# POLYSI: An Illustrating Example of “Long Fork”

$((SO_G \cup WR_G \cup WW_G) ; RW_G?)$  is *acyclic*.



# POLYSI: An Illustrating Example of “Long Fork”

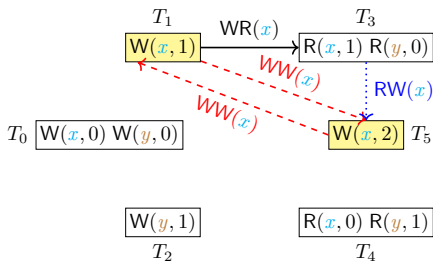
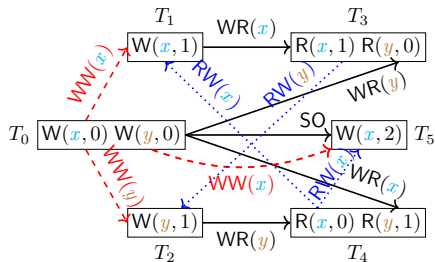
$((SO_G \cup WR_G \cup WW_G) ; RW_G?)$  is acyclic.



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

# POLYSI: An Illustrating Example of “Long Fork”

$((SO_G \cup WR_G \cup WW_G) ; RW_G?)$  is acyclic.

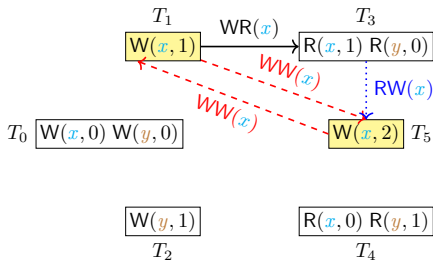
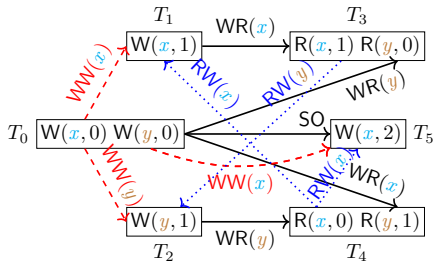


We need to encode the “composition ( ; )” of dependency edges.

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

# POLYSI: An Illustrating Example of “Long Fork”

$((SO_G \cup WR_G \cup WW_G) ; RW_G?)$  is acyclic.



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

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

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

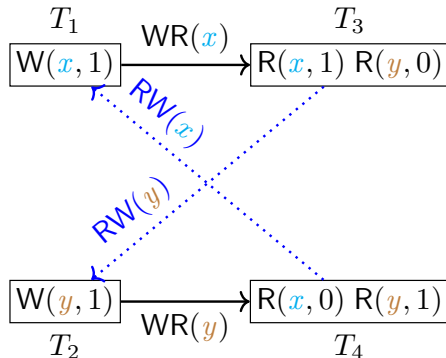
# POLYSI: An Illustrating Example of “Long Fork”

Feed the SAT formula into the **MonoSAT** solver [**MonoSAT:AAI2015**] 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

# Performance Evaluation

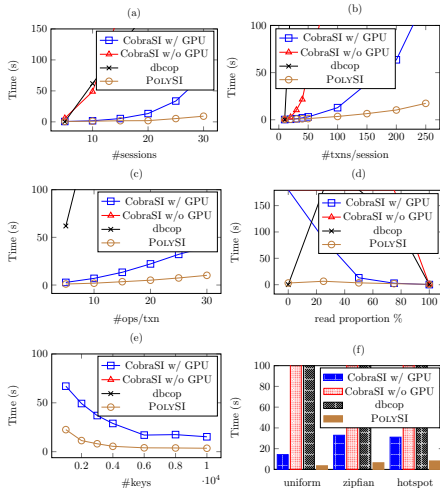
dbcop [Complexity:OOPSLA2019]: the state-of-the-art SI checker without using SAT solvers

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

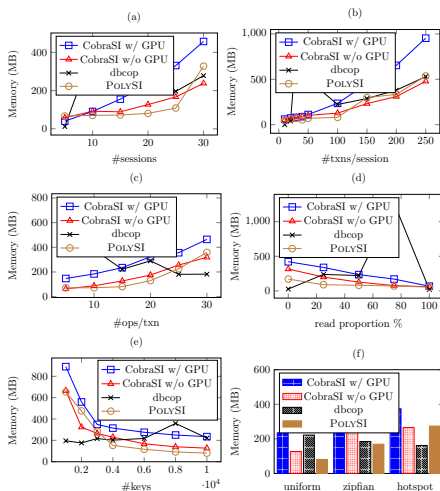
PolySI significantly outperforms the competitors.



All the input histories extracted from PostgreSQL satisfy SI

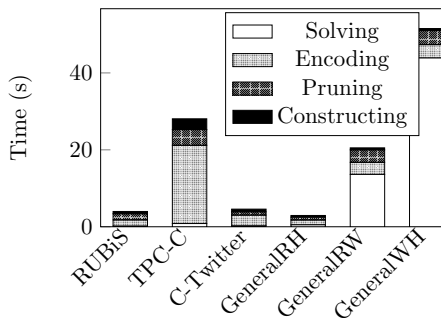
# Performance Evaluation: Memory

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

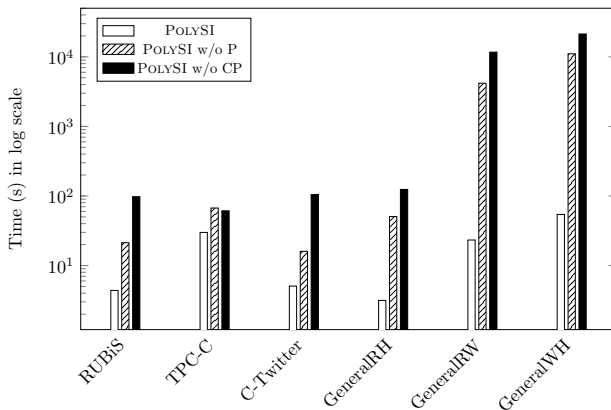
POLYSI can effectively **prune** a huge number of constraints.

Benchmark	#cons.	#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

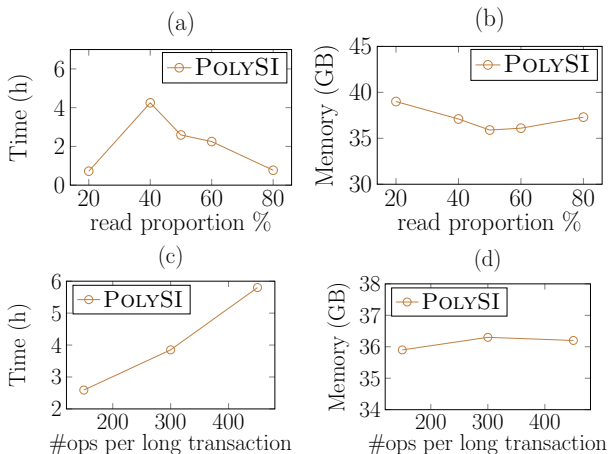
# Performance Evaluation: Differential Analysis

Pruning is crucial to the efficiency of POLYSI.



# Performance Evaluation: Scalability

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

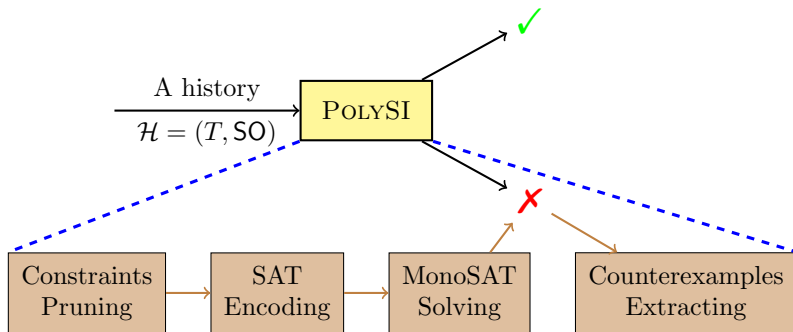








# Conclusion





Hengfeng Wei (hfwei@nju.edu.cn)



