# Safety of Transactions in Transactional Memory: TMS is Necessary and Sufficient

Hagit Attiya, Technion
Sandeep Hans, Technion
Alexey Gotsman, IMDEA
Noam Rinetzky, Tel-Aviv University

# TM Consistency Conditions

## Opacity

[Guerraoui & Kapalka 08]

- Validity of all transactions (included aborted ones) is checked together

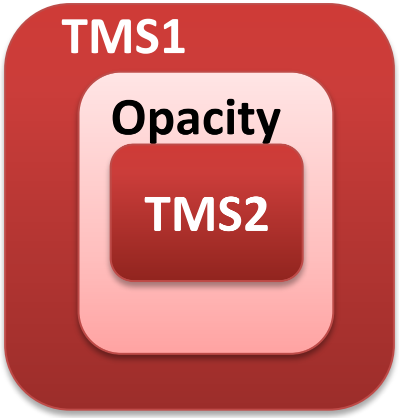
## Transactional Memory Specification (TMS1/2)

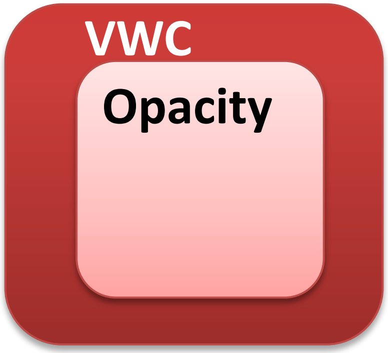
[Doherty, Groves, Luchangco, Moir 09]

- In TMS1, validity of each response is checked against a coherent subset of the transactions
  - May even include aborted transactions

## Virtual World Consistency

[Imbs & Raynal 09]





# Comparing TM Consistency Conditions

What is the **“right” consistency condition**?

Does the TM consistency condition allows to program with a simpler (i.e., atomic) TM in mind?

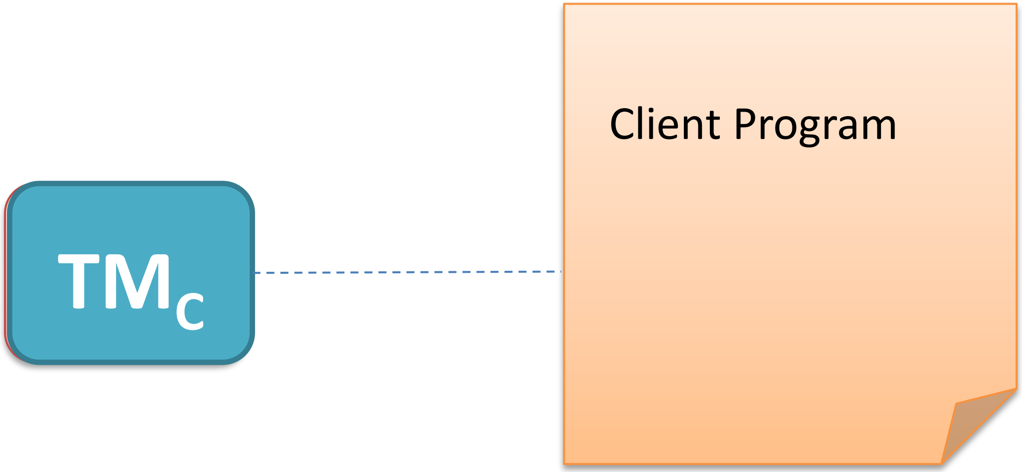
- If local variables are **rolled back** after a transaction aborts, TMS(1) is sufficient and necessary for programming with an atomic TM in mind
- If local variables are **not rolled back** on an abort (e.g., ScalaTM), the stronger Opacity condition is necessary

[Attiya, Gotsman, H, Rinetzky 13]

# Observational Refinement

[He, Hoare, Sanders 86]

- What is guaranteed for client programs, when an implementation is replaced with a simpler one?

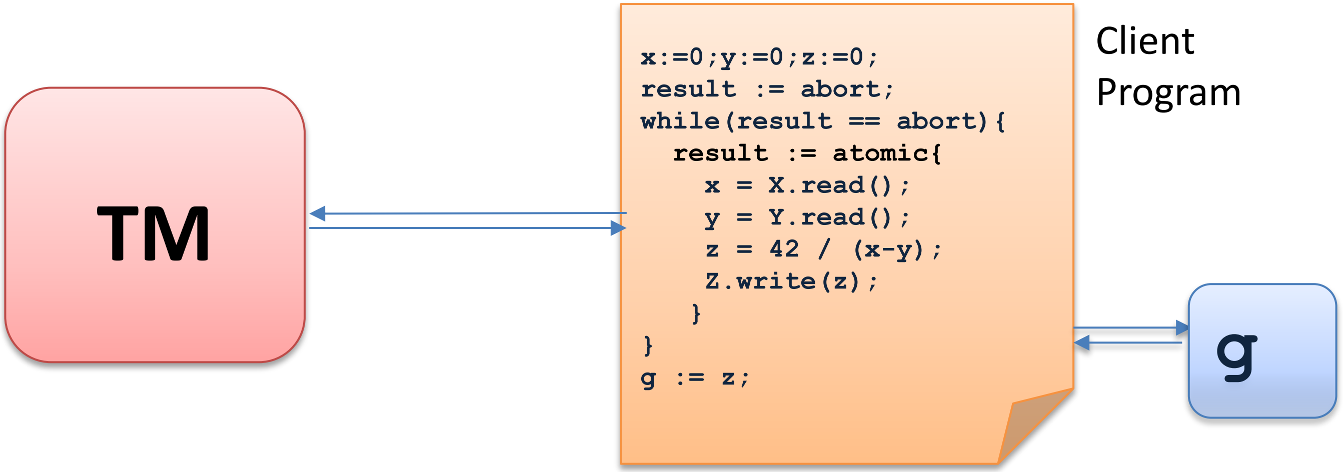


$TM_C$

Client Program

# Interactions of a Program using TM

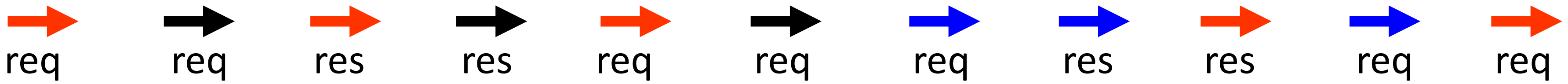
- **Local actions:** access only the local variables
- **Global actions:** interact with other client programs
- **Interface actions:** interact with TM



```
x:=0;y:=0;z:=0;
result := abort;
while(result == abort){
  result := atomic{
    x = X.read();
    y = Y.read();
    z = 42 / (x-y);
    Z.write(z);
   }
}
g := z;
```

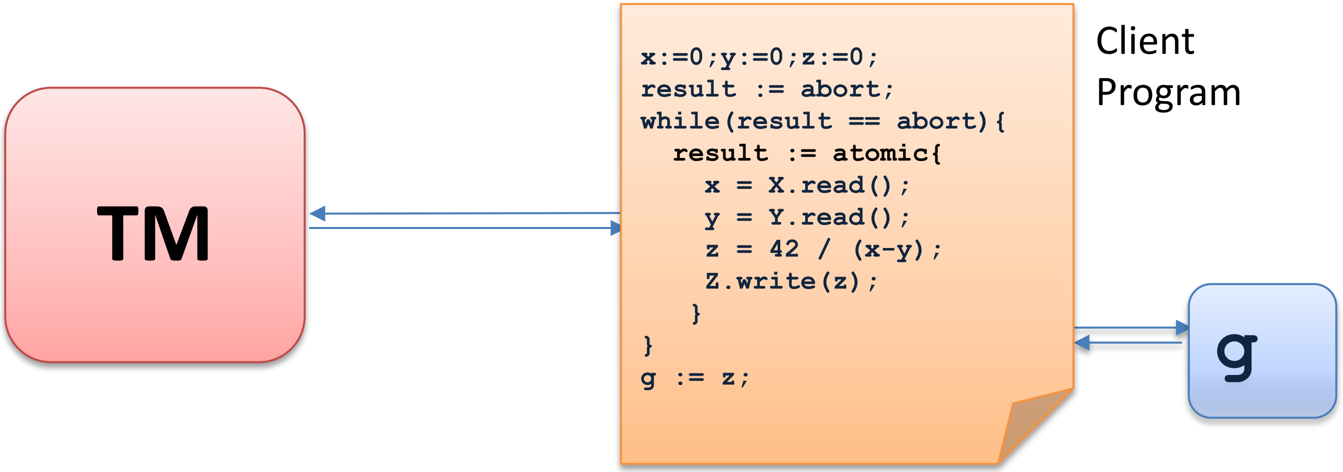
# Histories

**History**: Finite sequence of interface actions



req req res res req req req res res req req

**Well-formed**: Threads are sequential



TM

```
x:=0;y:=0;z:=0;
result := abort;
while(result == abort){
  result := atomic{
    x = X.read();
    y = Y.read();
    z = 42 / (x-y);
    Z.write(z);
   }
}
g := z;
```

Client
Program

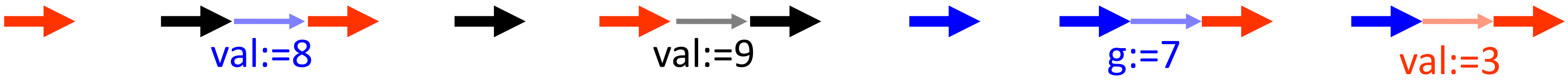
g

**Transactional Memory (TM)**: set of histories

– well-formed, prefix-closed

# Trace Equivalence

**Trace:** includes also local and global actions

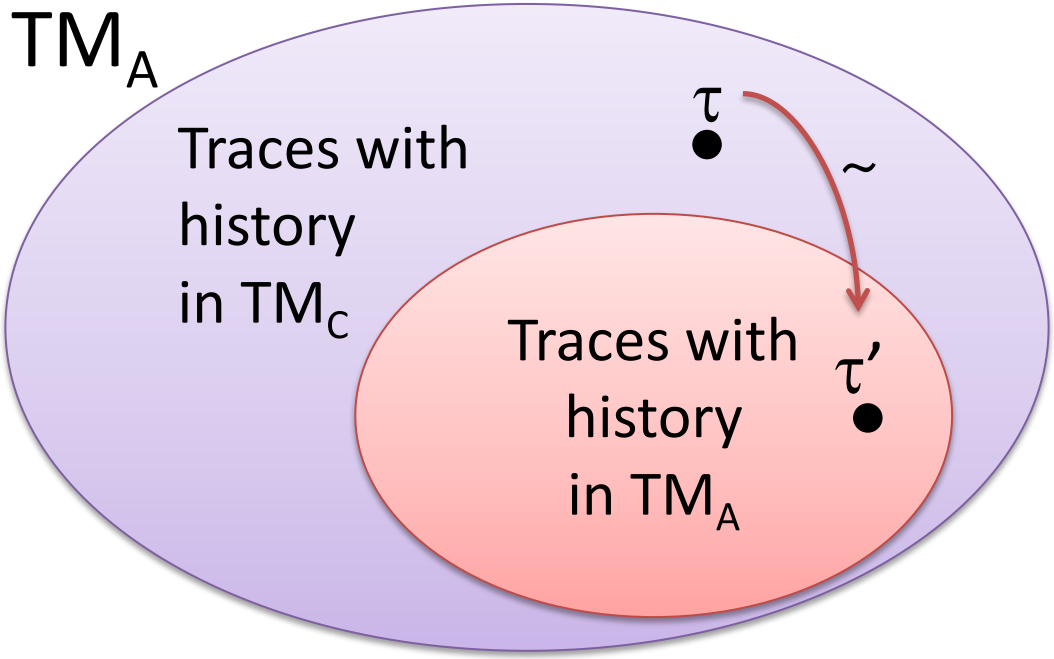


Two traces are **observationally equivalent** $\tau \sim \tau'$
if threads have the same sequence of local values,
except for local values inside aborted transactions

$TM_C$ **observationally refines** $TM_A$ if
every trace $\tau$ with history in $TM_C$
has a trace $\tau' \sim \tau$ with history in $TM_A$

# Why Observational Refinement?

Prove properties for TM$_A$ and deduce the same for TM$_C$

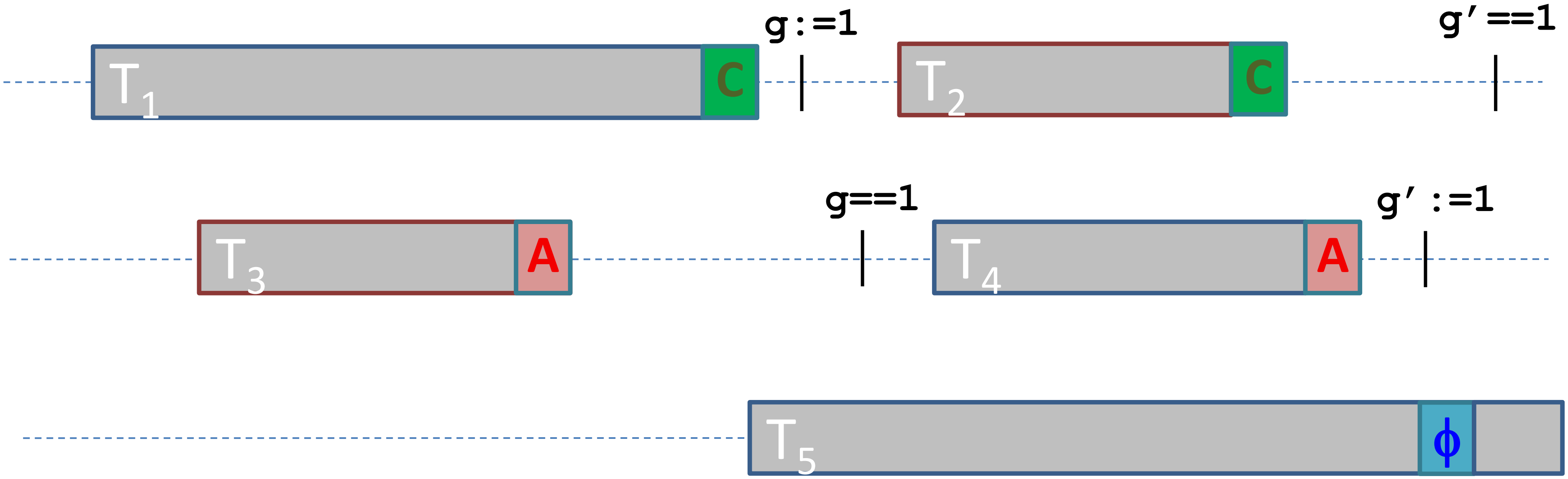


Traces with history in TM$_C$

Traces with history in TM$_A$

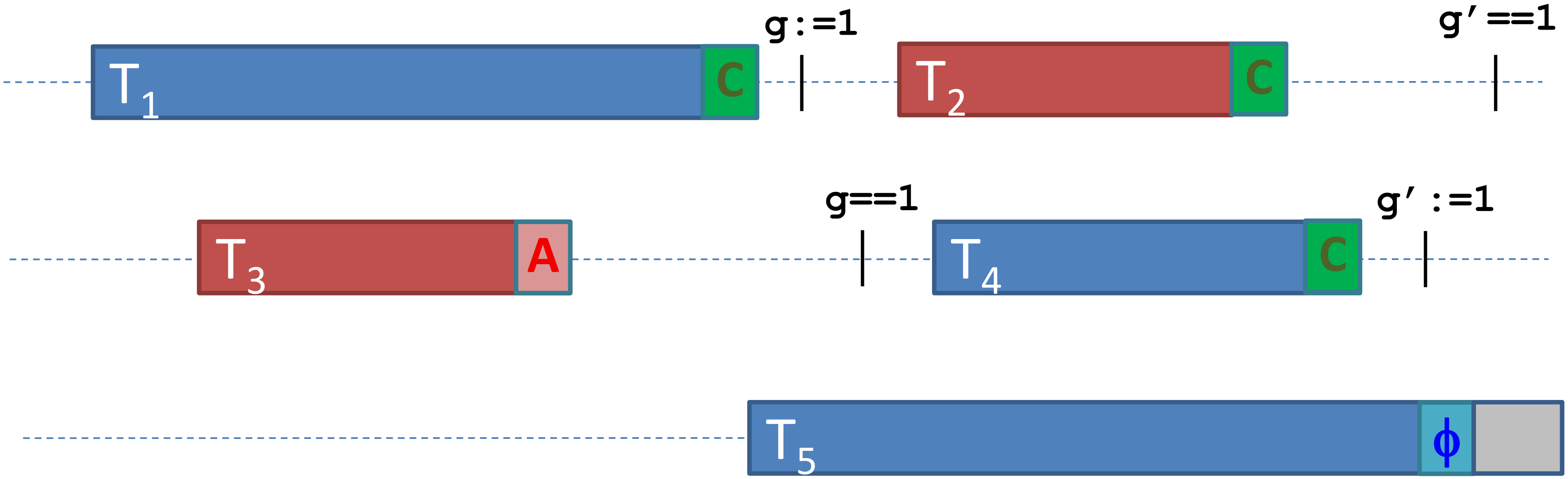
TM$_C$ **observationally refines** TM$_A$ if every trace $\tau$ with history in TM$_C$ has a trace $\tau' \sim \tau$ with history in TM$_A$

# TMS: By Example



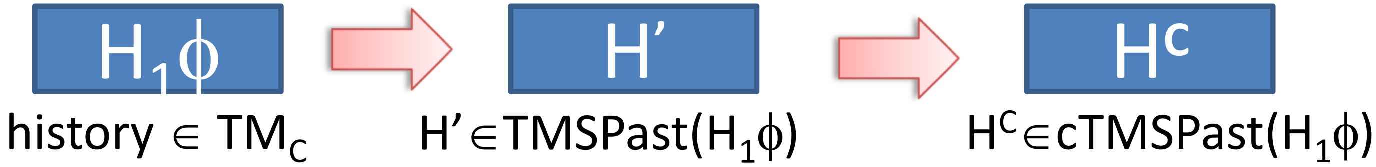
- transaction of ϕ is included
- **some visible** transactions are included
- for every included transaction, **exactly** all past committed transactions are included

# TMS: By Example



- **Commit** included aborted transactions (by replacing abort with commit)
- **Commit** included commit-pending transactions
- **Remove** all other transactions

# TMS: The Past of an Action $\phi$



$H_1\phi$ → $H'$ → $H^C$

history $\in TM_C$

$H' \in TMSPast(H_1\phi)$

$H^C \in cTMSPast(H_1\phi)$

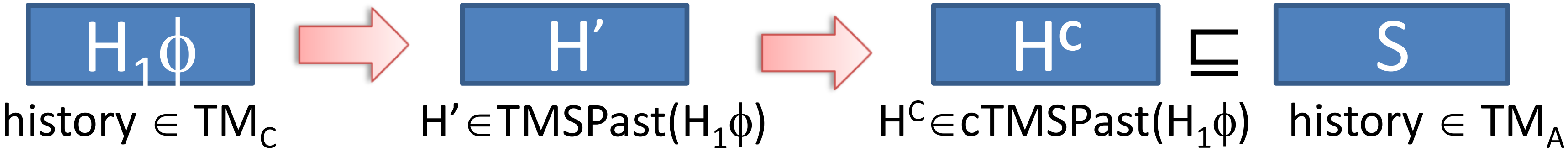
$H' \in TMSPast(H_1\phi)$

- H’ is a **subsequence** of H
- H’ contains **transaction of $\phi$** and some **visible** transactions in H
- for every included transaction T in H’, **exactly** all past committed transactions are included

$H^c \in cTMSPast(H_1\phi)$

- **commit** all commit-pending transactions
- **replace** aborted actions by committed actions

# Definition of TMS



$H_1\phi$ → $H'$ → $H^C \sqsubseteq S$

history $\in TM_C$ $H' \in TMSPast(H_1\phi)$ $H^C \in cTMSPast(H_1\phi)$ history $\in TM_A$

$H^C \sqsubseteq S$

- S preserves the **per-thread** and **real-time** order of $H^C$

$H \sqsubseteq_{\mathbf{tms}} TM_{\mathbf{A}}$

- all committed transactions have a serialization
- for every response action $\phi$, there is a complete past $H^c$ and a history $S \in TM_A$ such that $H^c \sqsubseteq S$

$TM_C$ is TMS $\triangleq$ for every $H \in TM_C$, $H \sqsubseteq_{tms} TM_{ATOMIC}$

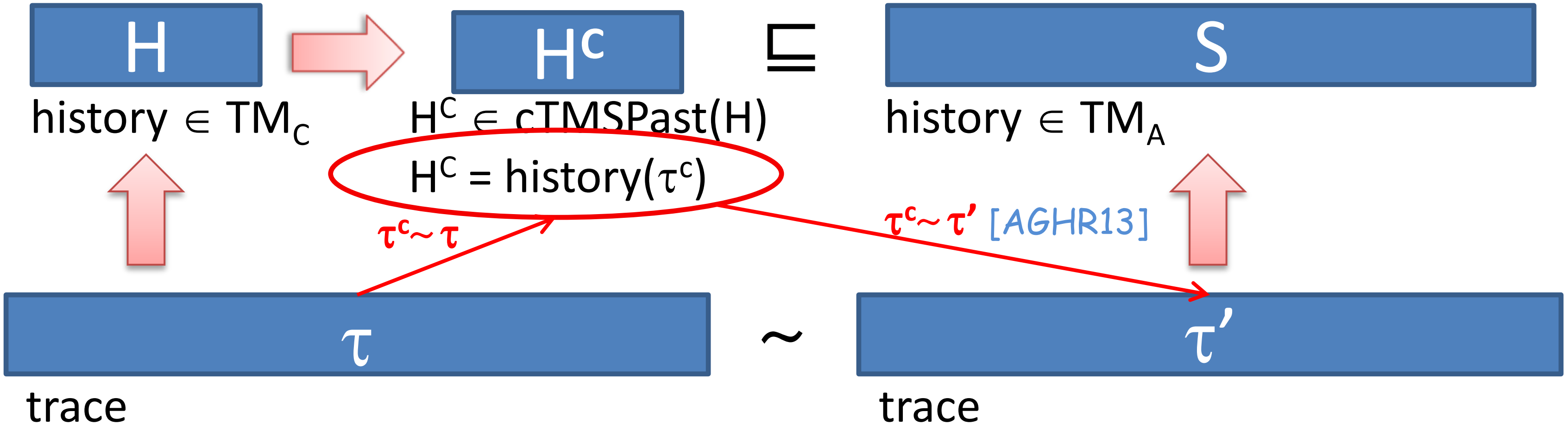
# Main Result

$TM_C \sqsubseteq_{tms} TM_A \quad \Leftrightarrow \quad TM_C$ observationally refines $TM_A$

- no nesting of atomic blocks
- no access to global variables in atomic blocks

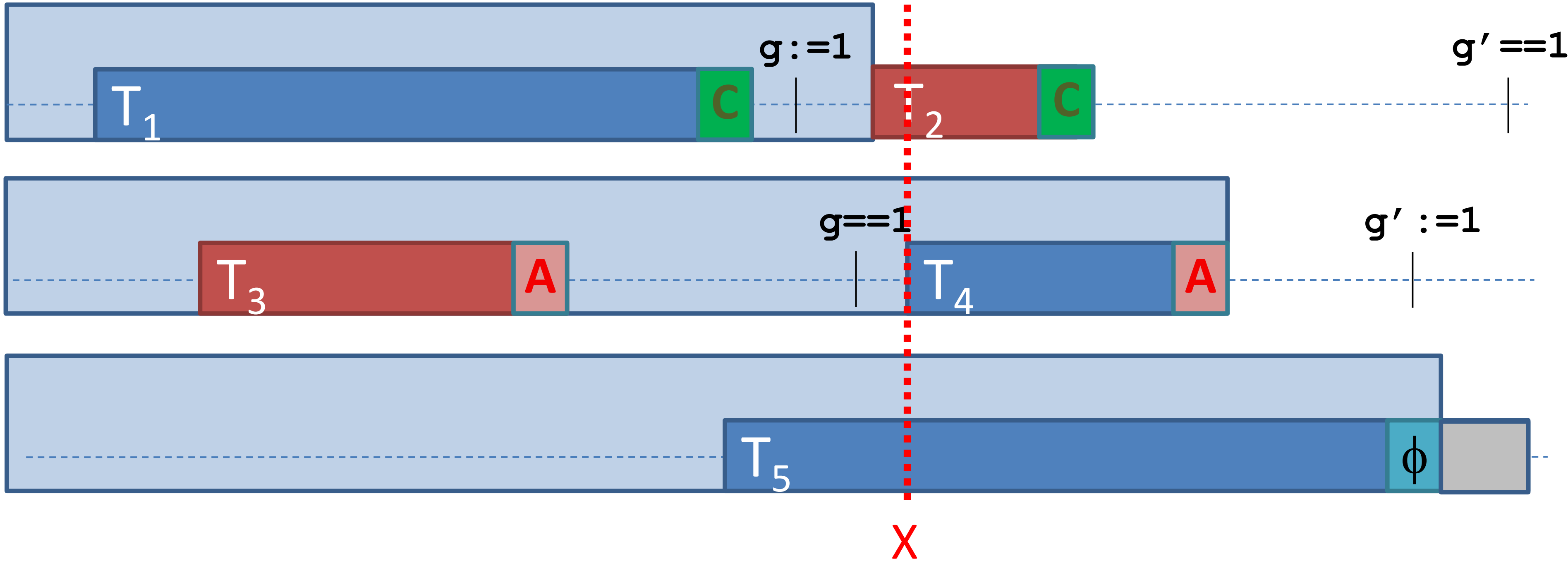
# $\sqsubseteq_{\text{tms}}$ is Sufficient

Every trace $\tau$ observed when running with $TM_C$ has an equivalent trace $\tau'$ observed when running with $TM_A$



- Consider a trace $\tau$ whose history H is in $TM_C$
- $TM_C \sqsubseteq_{\text{tms}} TM_A \Rightarrow$ $H^C \in$ cTMSPast(H) and $H^C \sqsubseteq S \in TM_A$
- From $\tau$ and S, get a trace $\tau' \sim \tau$ of $TM_A$ whose history is S

# Constructing $\tau^c \sim \tau$



- Let X be the beginning of the last included transaction
- For every thread t, take the trace until the latest of:
  - The last non-transactional action before X
  - The last transactional action of t in H'

# What's Next?

- Extend the results to handle nesting and access to global variables in atomic blocks
- Weaker observations are preserved by VWC?
- Stronger observations (e.g., global accesses in a transaction) are preserved by deferred update opacity or TMS2?