Computer Science
UNIVERSITY OF TORONTO

Taming TSO Memory Consistency with Models

Michael N. Christoff

Presented by Michael N. **C**hristoff
December 19, 2012

Plan of Attack

1. High Level Motivation and Overview
2. (Some) Details of TSO Consistency
3. A look at TSO Helper via an Example
4. Topics not covered
5. Future Directions

1. High Level Motivation and Overview

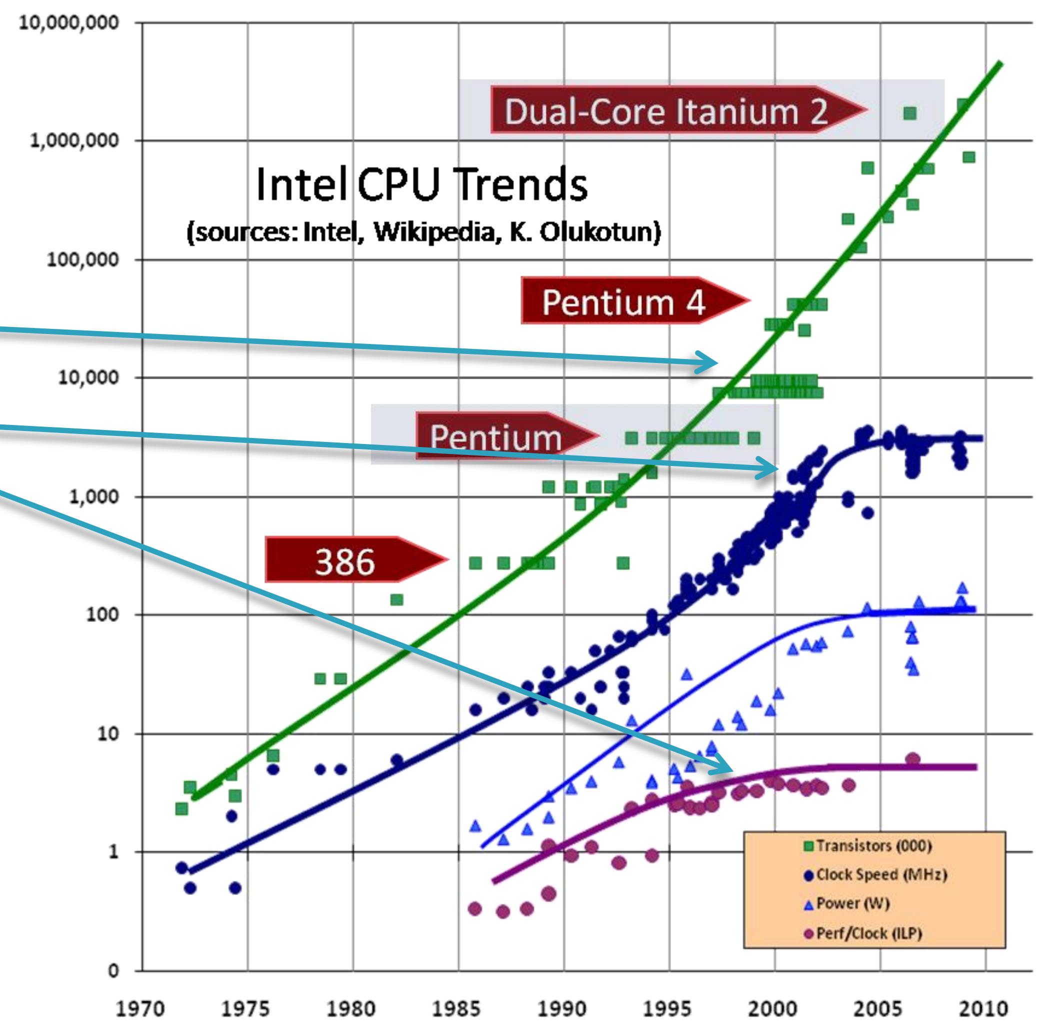
The End of the Age of Sequential Performance

Transistor Count

Clock Speed

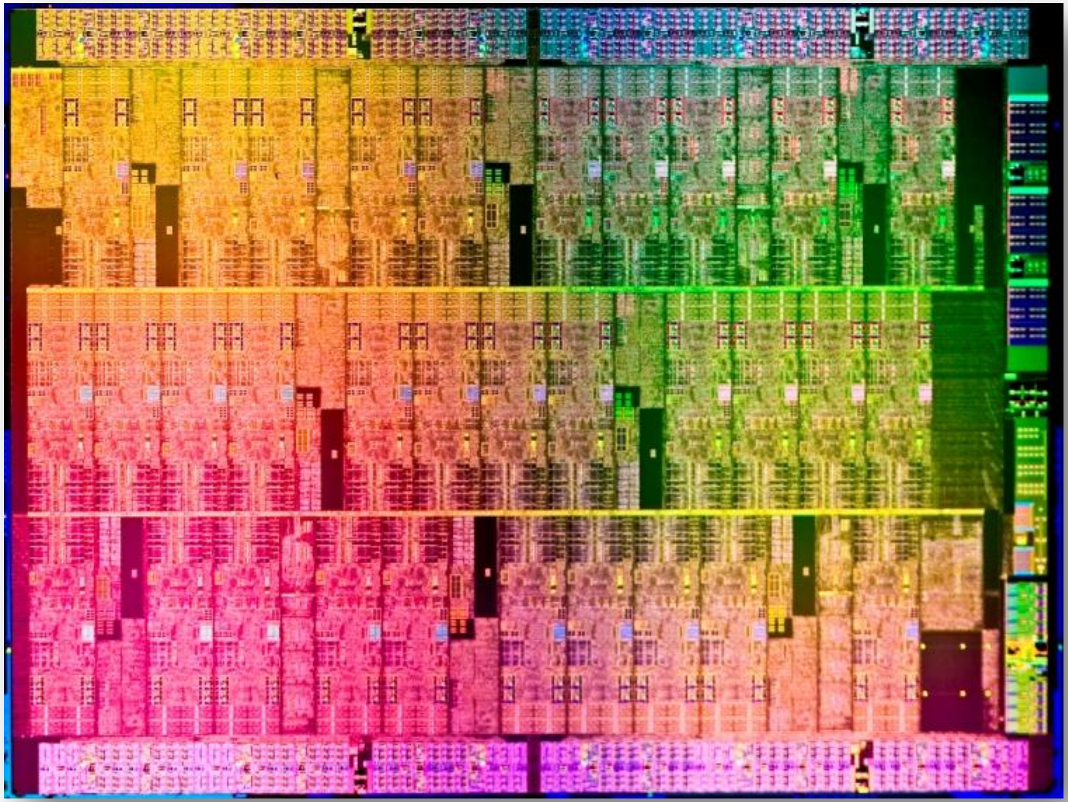
Single Core Performance

Sequential algorithms will no longer see the same speedups with new processor generations



http://www.gotw.ca/publications/concurrency-ddj.htm

The Beginning of the Age of Concurrency



Intel Xeon Phi 50 Core Microprocessor

The Beginning of the Age of Concurreny

- Need multi-threaded algorithms that take advantage of all available CPU cores
- In non-trivial multi-thread algorithms (versus ‘embarrassingly’ parallel algorithms), communication between threads is required
- Threads communicate by writing and reading messages to and from shared system memory

Sequential Memory Consistency

At the hardware level, the ordering of reads and writes in a program’s source code is not always the order that these reads and writes will be executed

Memory Consistency Model

- Defines ‘how much’ the original order of reads/writes in the source code can be re-ordered

- Sequential Consistency—This is the model we learned in school
  - Reads always return values from the shared memory
  - A write operation is always committed to main memory before the next instruction is executed

E.g.: When I write a value, I know everyone can see it!

TSO Memory Consistency

- TSO Memory Consistency is the native memory consistency model of all modern x86 and x64 CPUs (both Intel and AMD CPUs)

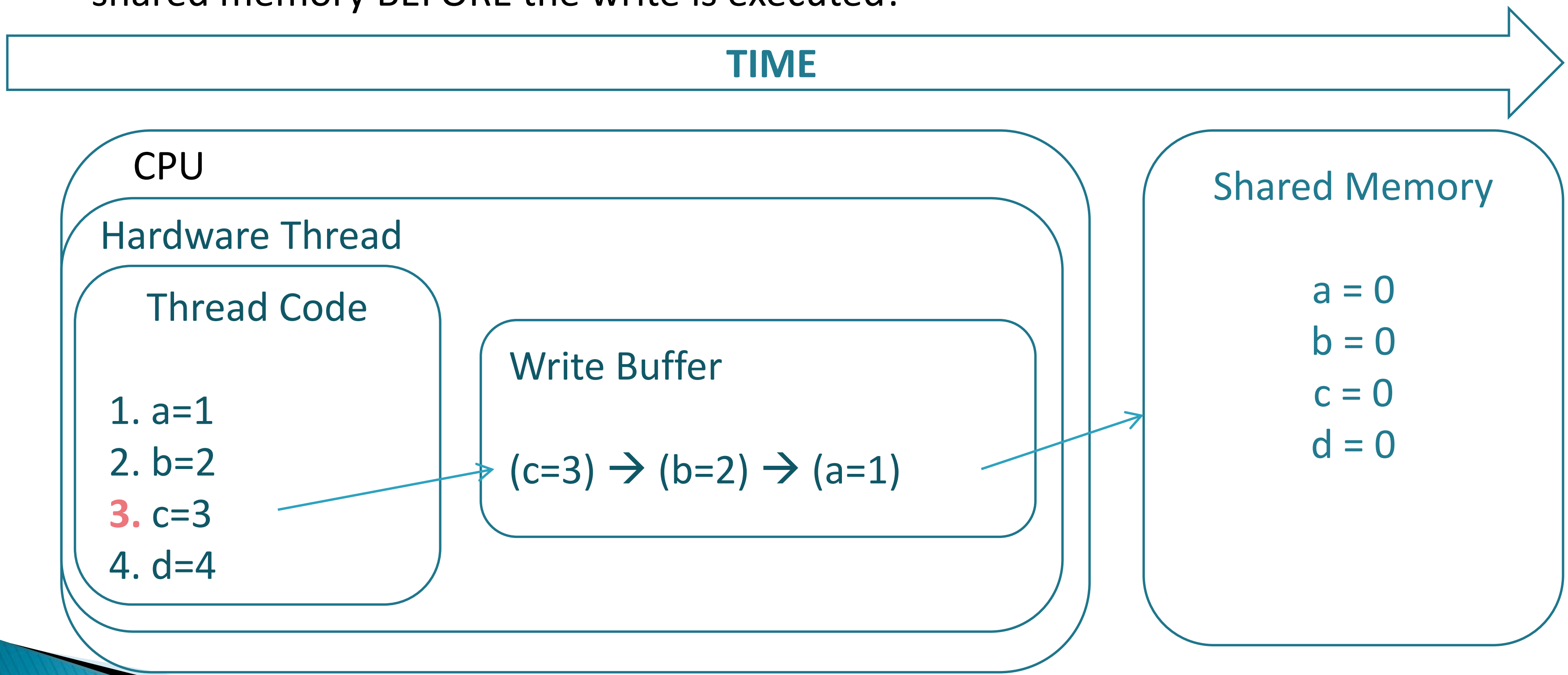
- In a system with TSO Memory Consistency, Read operations executed by a CPU core are not immediately applied to main memory

E.g.: *When I write a value, I'm not sure if everyone can see it!*

2. (Some) Details of TSO Consistency

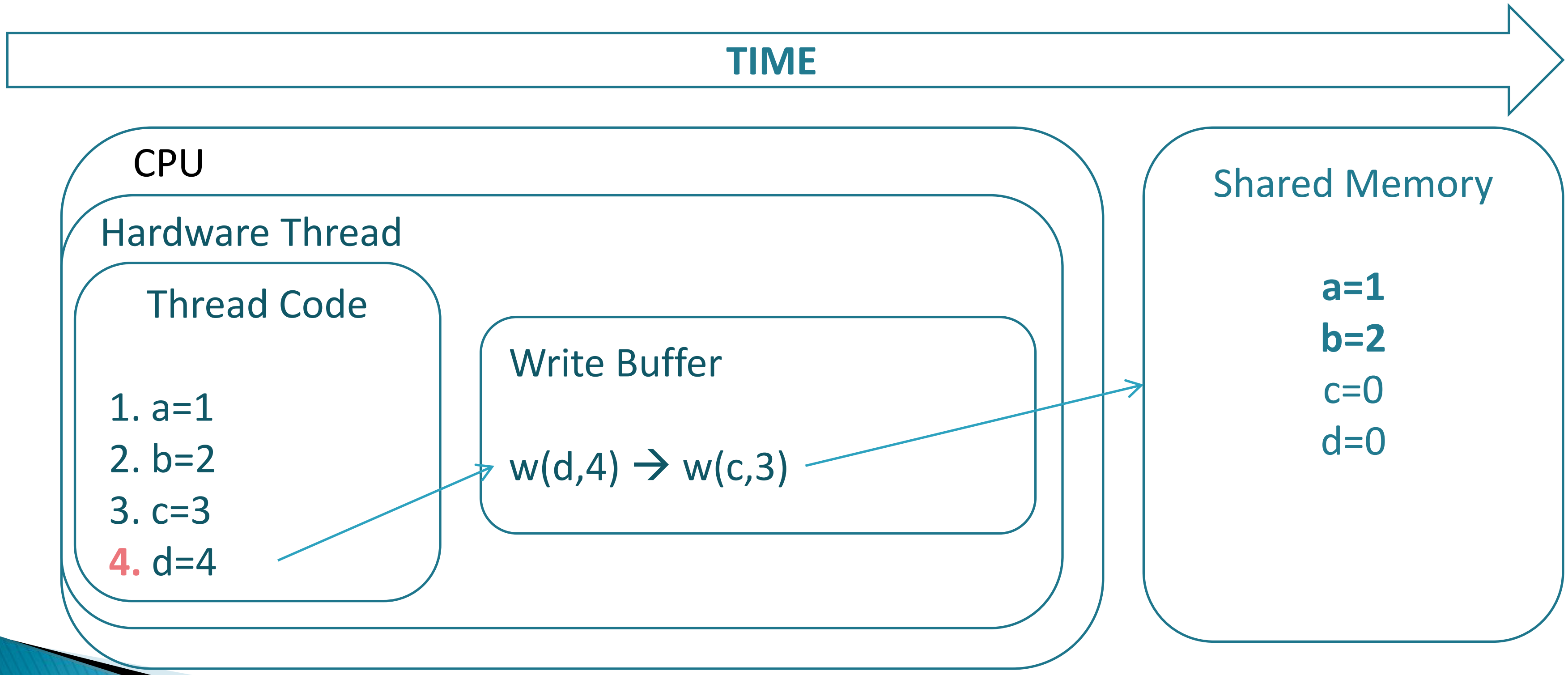
TSO Write Buffering

- Under TSO, write operations executed by threads ARE NOT immediately committed to memory
- When a thread executes x=v, the CPU takes the target variable x and value v, and puts it into a per-thread FIFO queue called a write buffer
- If you execute a WRITE, followed by a READ, the READ may grab a value from shared memory BEFORE the write is executed!



TSO Write Buffering

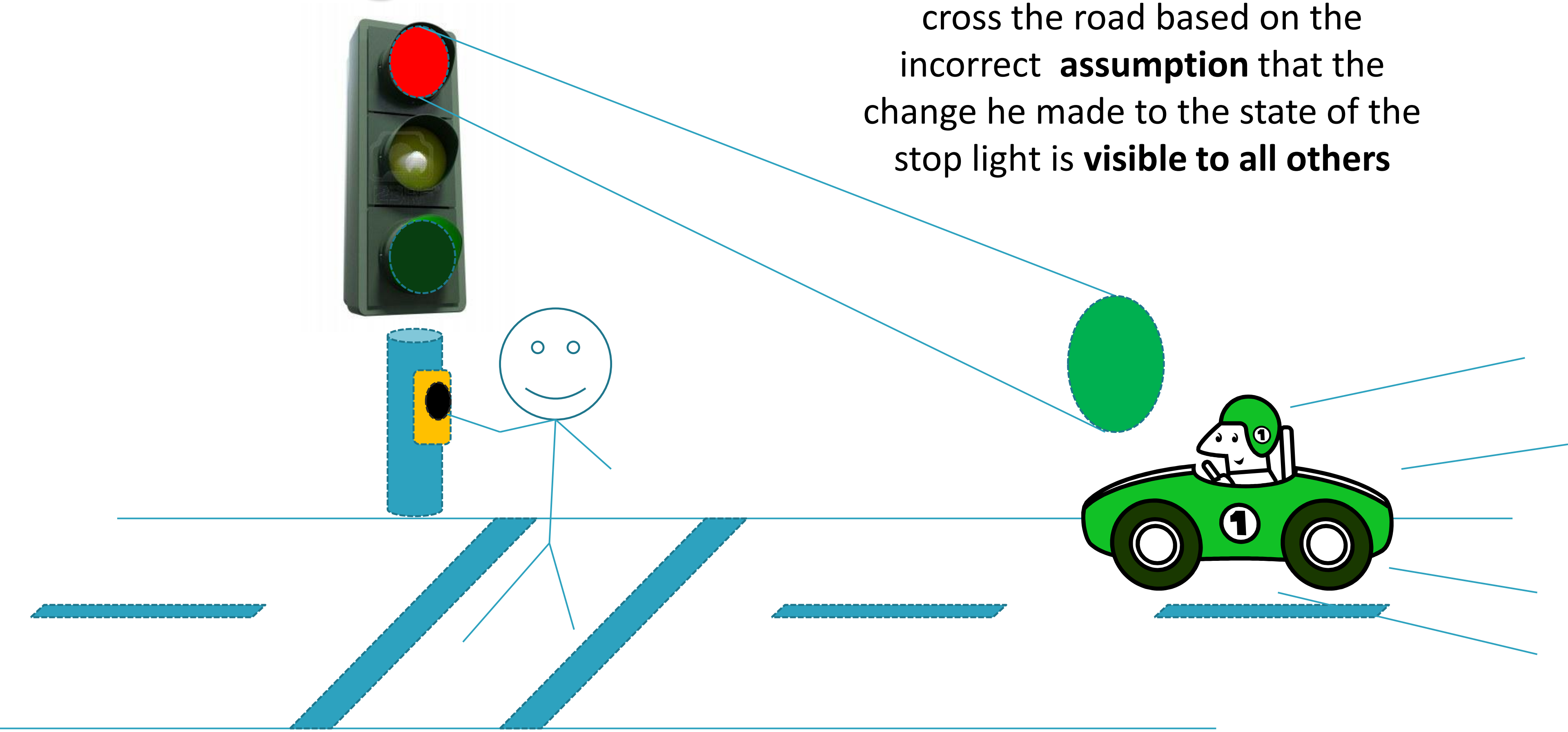
- At some later point in time, the buffered writes are committed to memory in the same order that they entered the write buffer
- It is **ONLY after a write is committed** that other processes can see the new value of the variable that was written to!



Delayed Writes Can be Dangerous

Crossing the Street

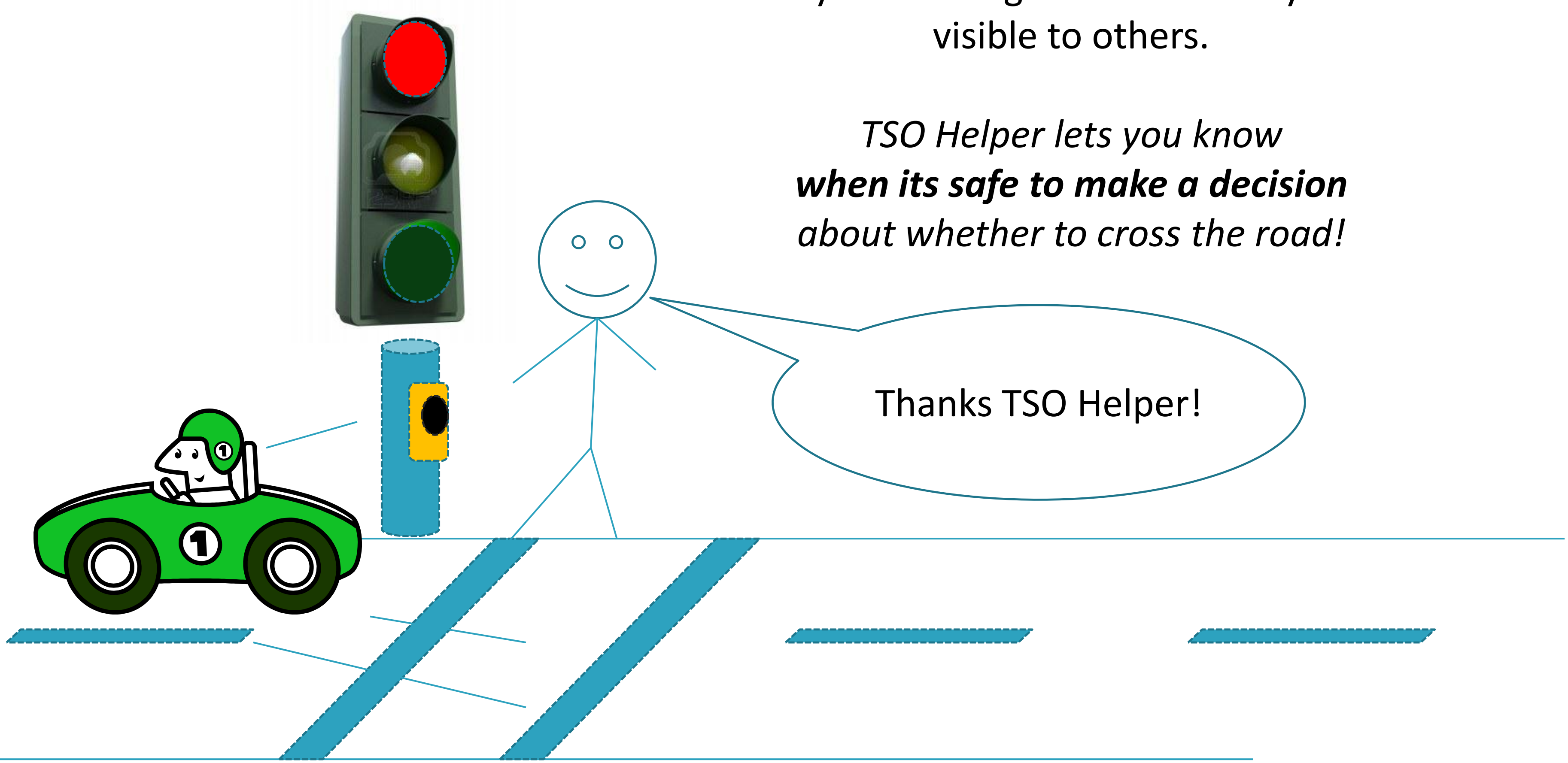
This person has made a **decision** to cross the road based on the incorrect **assumption** that the change he made to the state of the stop light is **visible to all others**



TSO Helper

TSO Helper shows you the places in the execution of your code where you can be guaranteed that your writes are visible to others.

TSO Helper lets you know
when its safe to make a decision
about whether to cross the road!



3. A look at TSO Helper via an Example

TSO Helper—An Example

A Banking Application

- X and Y are ATMs that perform deposit updates on a single bank account
- The current value of the account is stored in the shared variable ***acct***
- If it is a joint account, we must ensure only one account owner can deposit their money into the account at a time

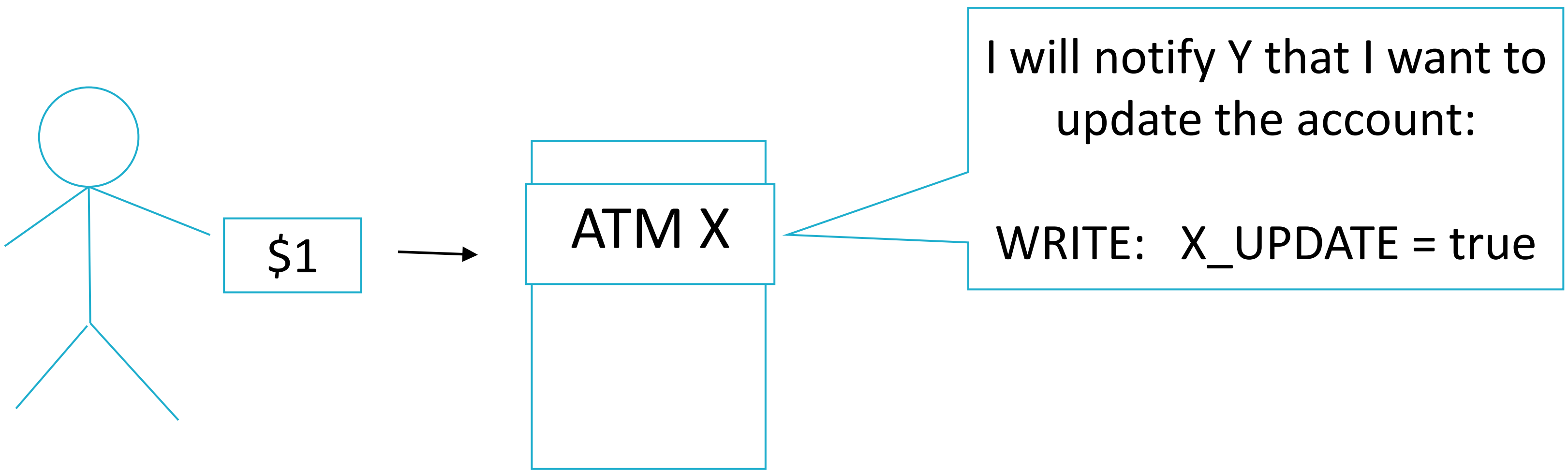
ATM X :

1. X_OTHER_BANKING_WORK();
2. X_UPDATE **= true**
3. **await(**¬Y__UPDATE**)**
4. *x_temp = acct + 1*
5. *acct = x_temp*
6. X_UPDATE **= false**
7. **goto 1**

ATM Y :

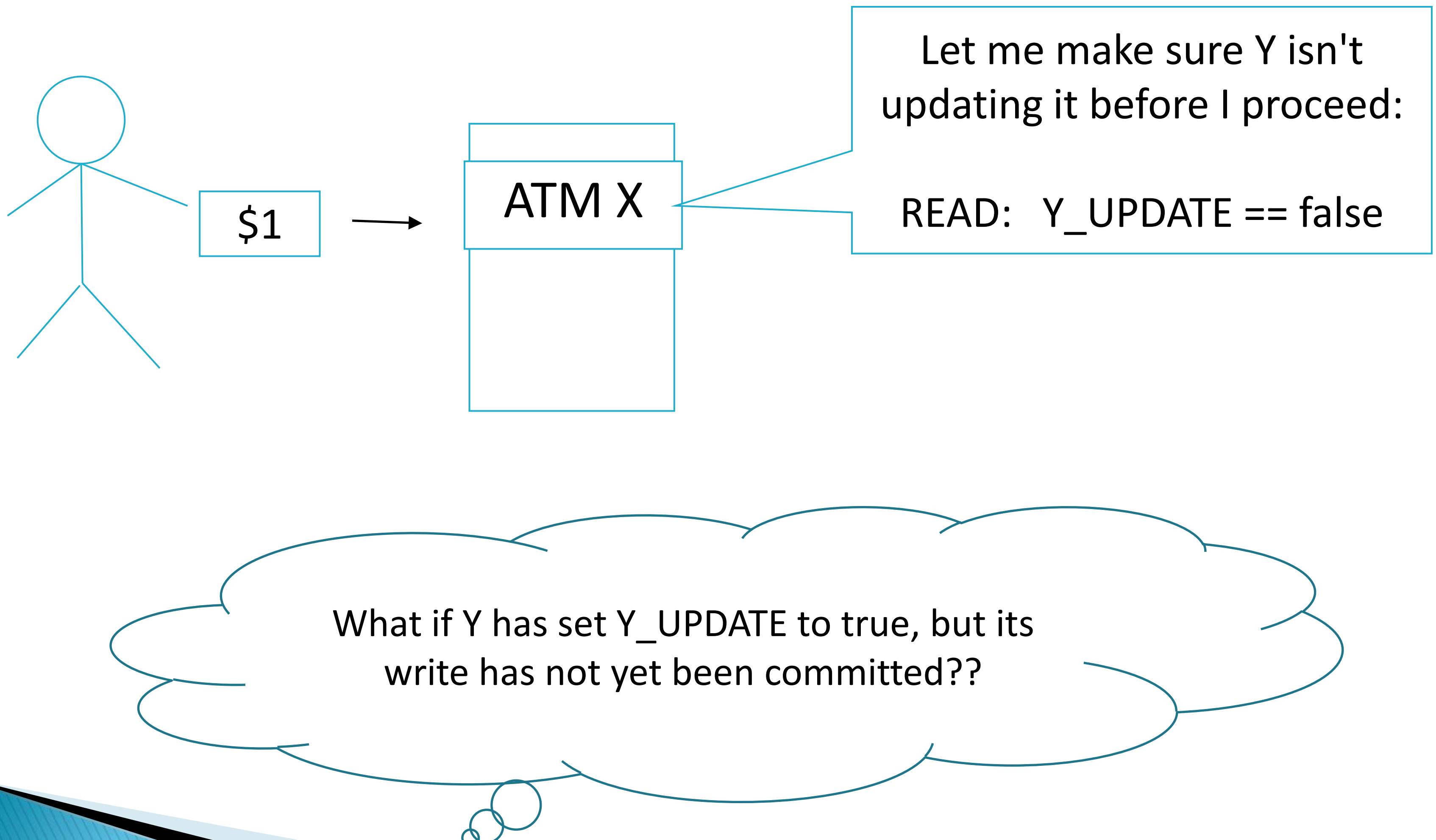
8. Y_OTHER_BANKING_WORK();
9. Y_UPDATE **= true**
10. **await(**¬X_UPDATE**)**
11. y_*temp = acct + 1*
12. *acct = y_temp*
13. Y_UPDATE **= false**
14. **goto 8**

1. TSO Helper—An Example

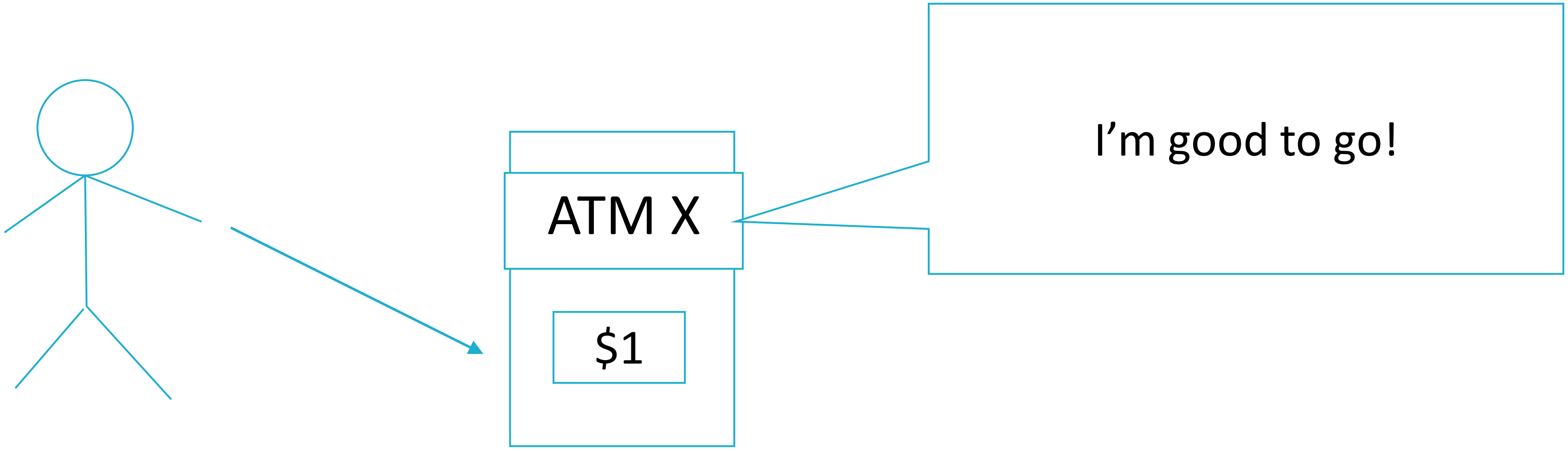


What if the write w(X_UPDATE,true) is buffered and not committed??

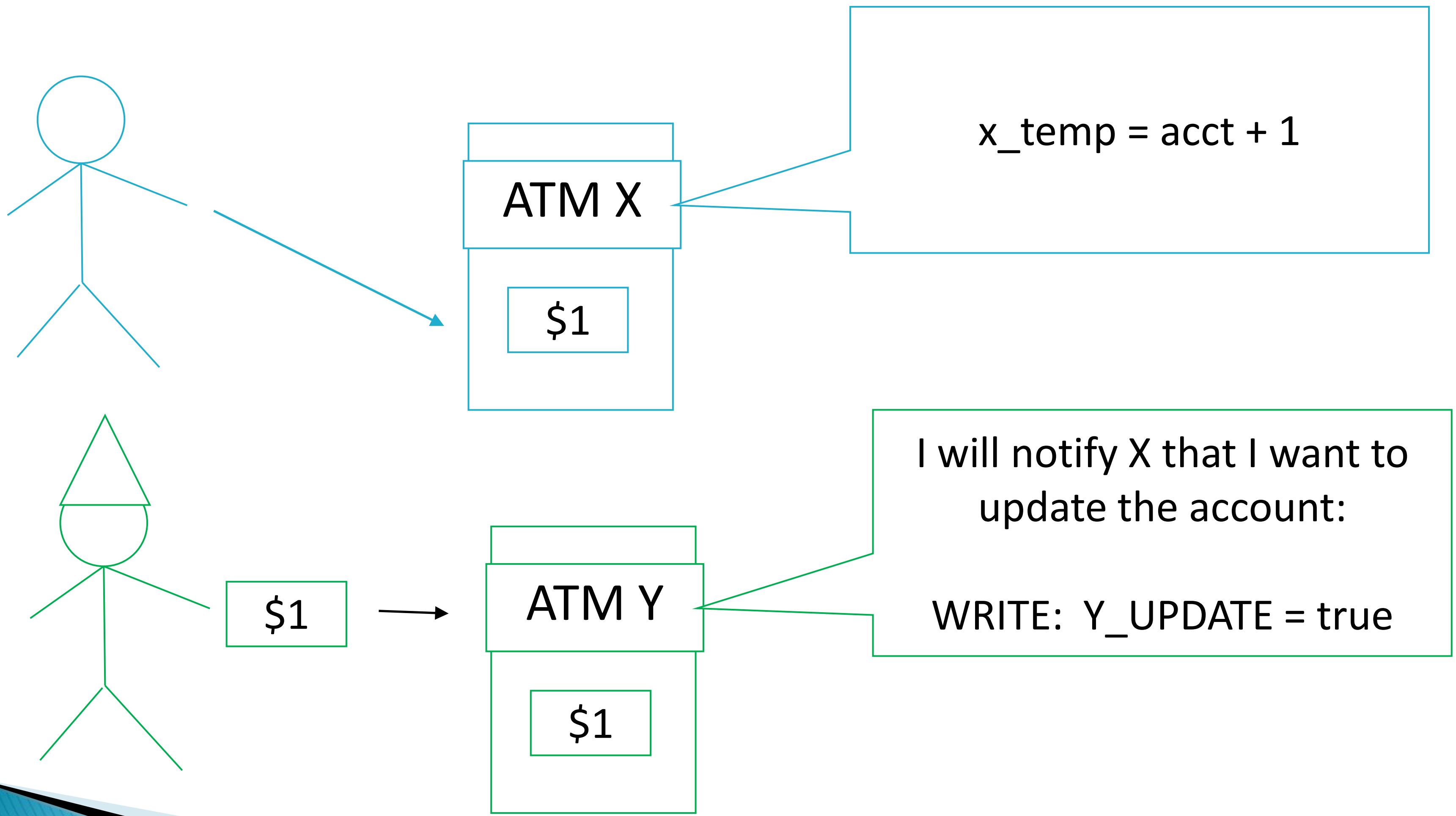
2. TSO Helper—An Example



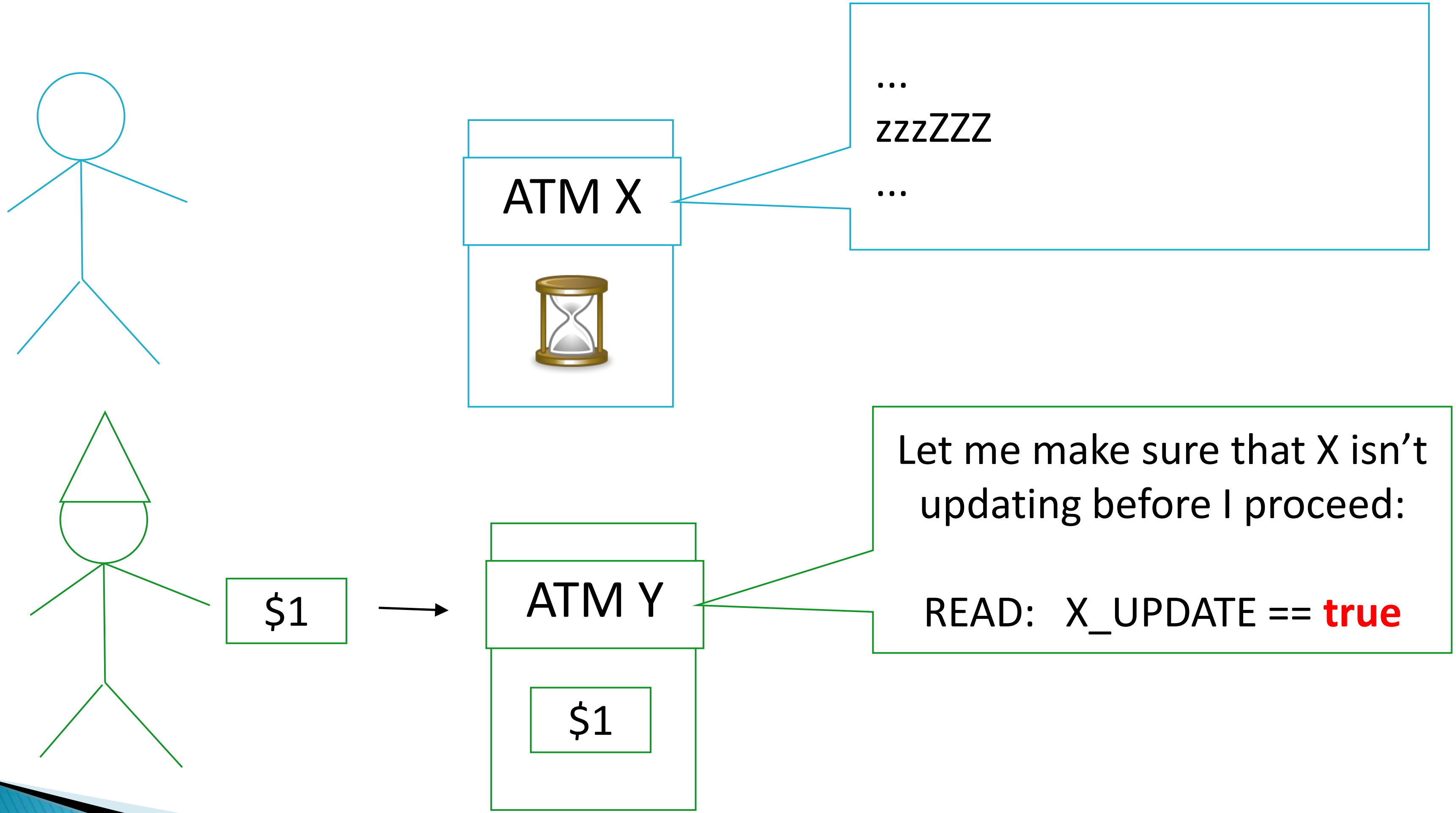
3. TSO Helper—An Example



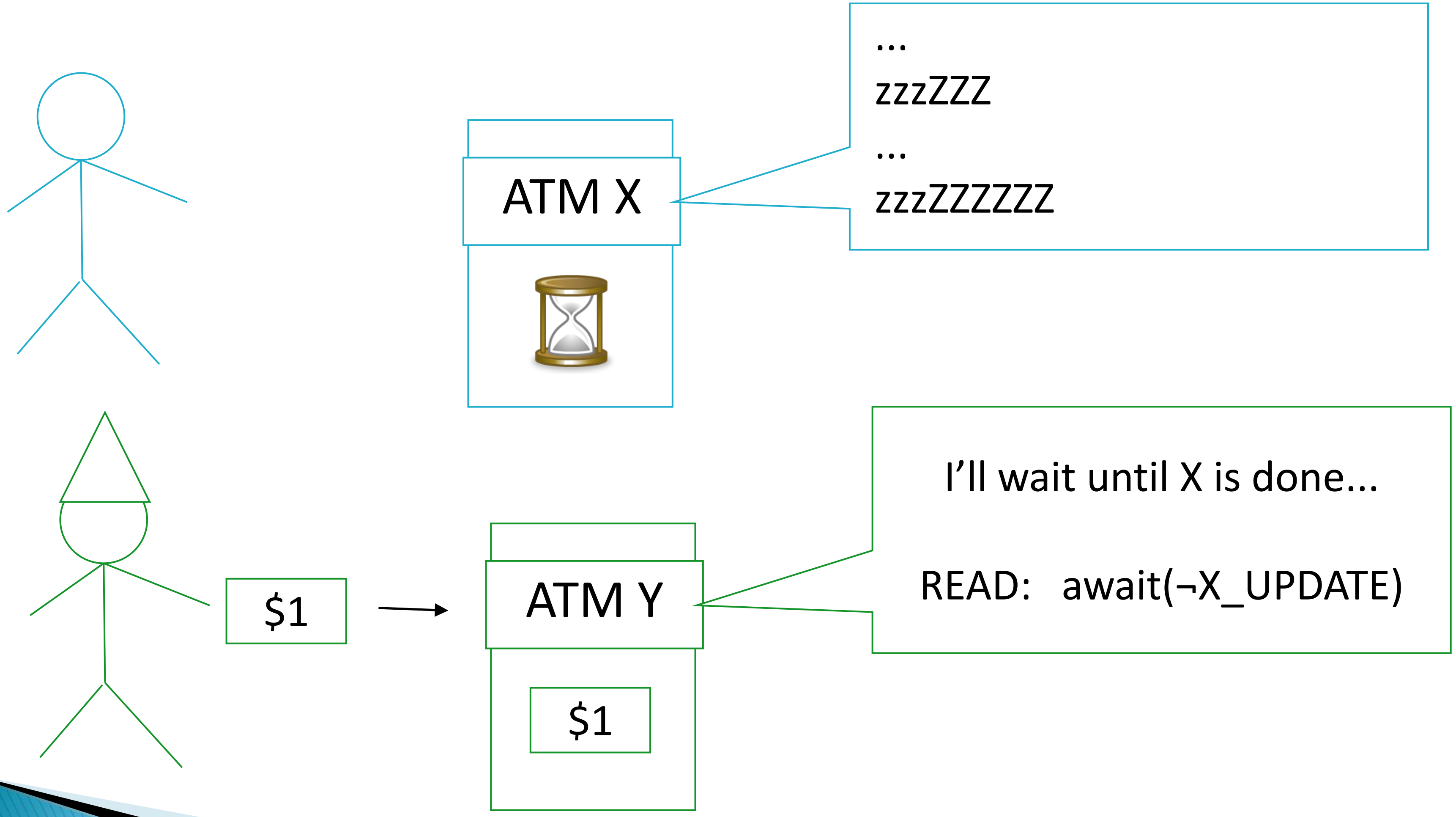
4. TSO Helper—An Example



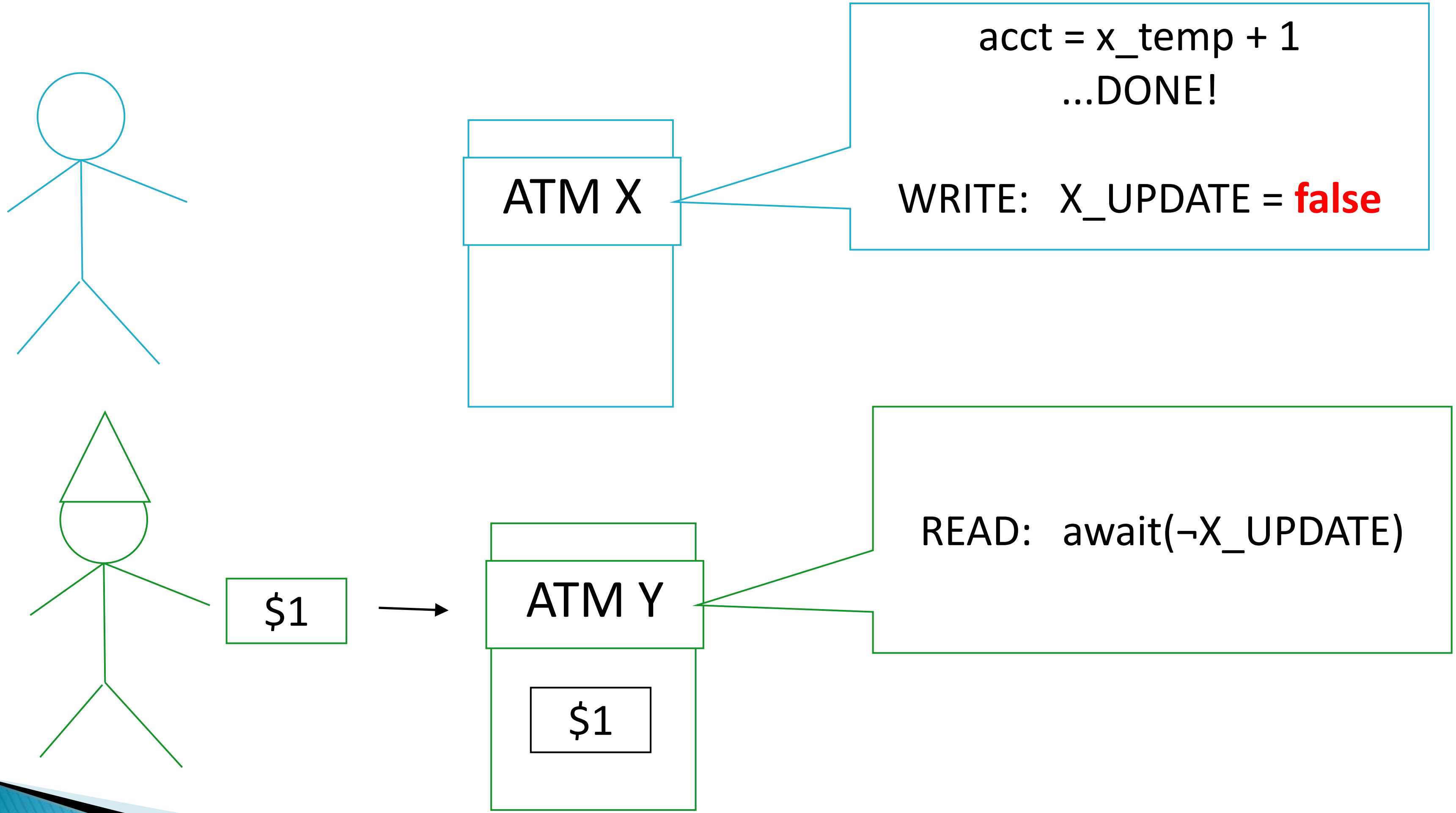
5. TSO Helper—An Example



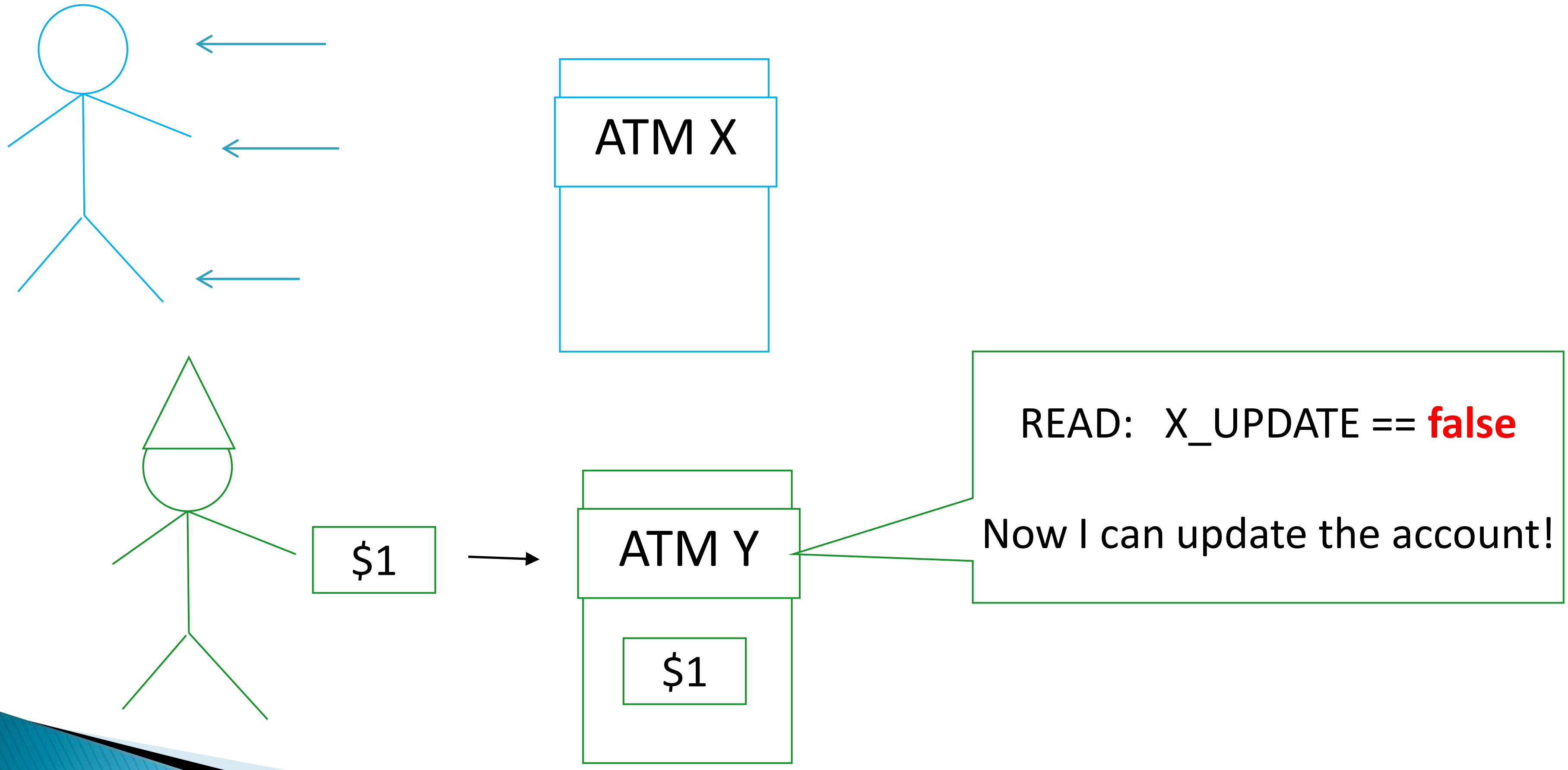
6. TSO Helper—An Example



7. TSO Helper—An Example



7. TSO Helper—An Example

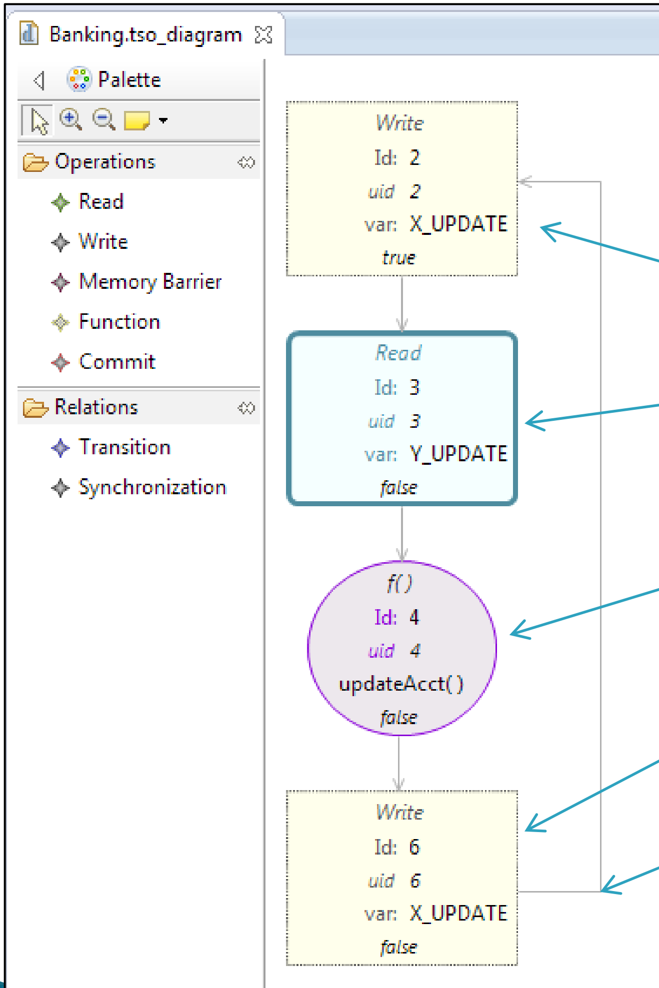


TSO Helper—An Example

- The Banking Algorithm works under SC where writes are immediately committed to shared memory
- It will not (always) work under TSO if writes are buffered

Can TSO Helper Help Us Detect this Issue?

TSO Helper Designer – ‘Read/Write’ Model



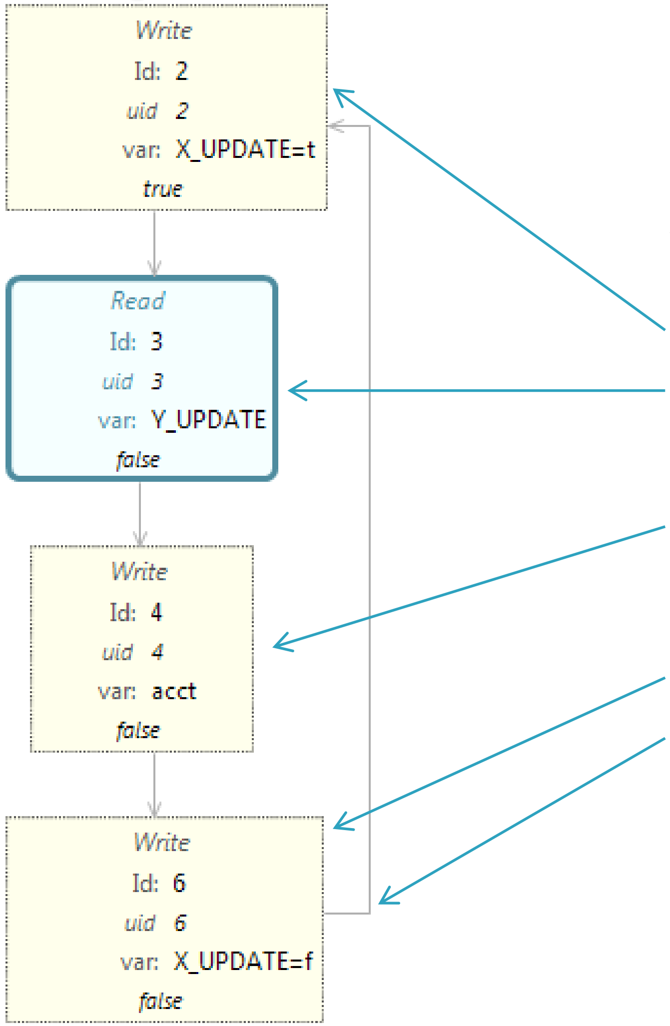
ATM X :

1. ~~X_OTHER_BANKING_WORK();~~
2. X_UPDATE **= true**
3. **await(¬Y__UPDATE)**

4. *x_temp = acct + 1*
5. *acct = x_temp*

6. X_UPDATE **= false**
7. **goto 1**

A Simpler Model



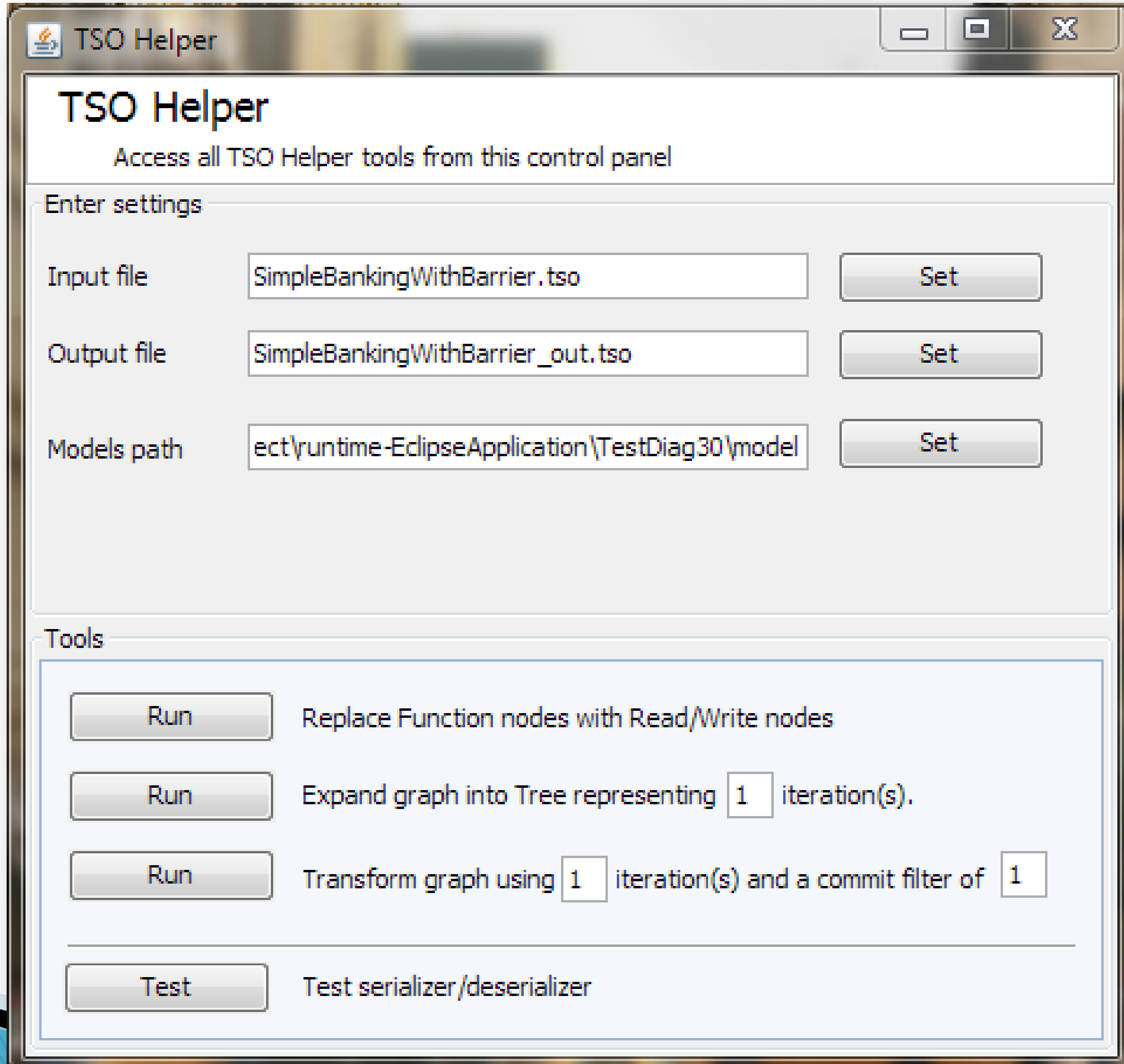
ATM X :

1. ~~X_OTHER_BANKING_WORK();~~
2. X_UPDATE **= true**
3. **await(**¬Y__UPDATE**)**

4. *x_temp = acct + 1*
5. *acct = x_temp*

6. X_UPDATE **= false**
7. **goto 1**

Run the Model Through TSO Helper



TSO Helper Does Not Like The Algo

- After exploring three iterations of the algorithm, TSO Helper concludes (at least within 3 iterations) that their is NO point in the algorithm at which any write can be guaranteed to have been committed.

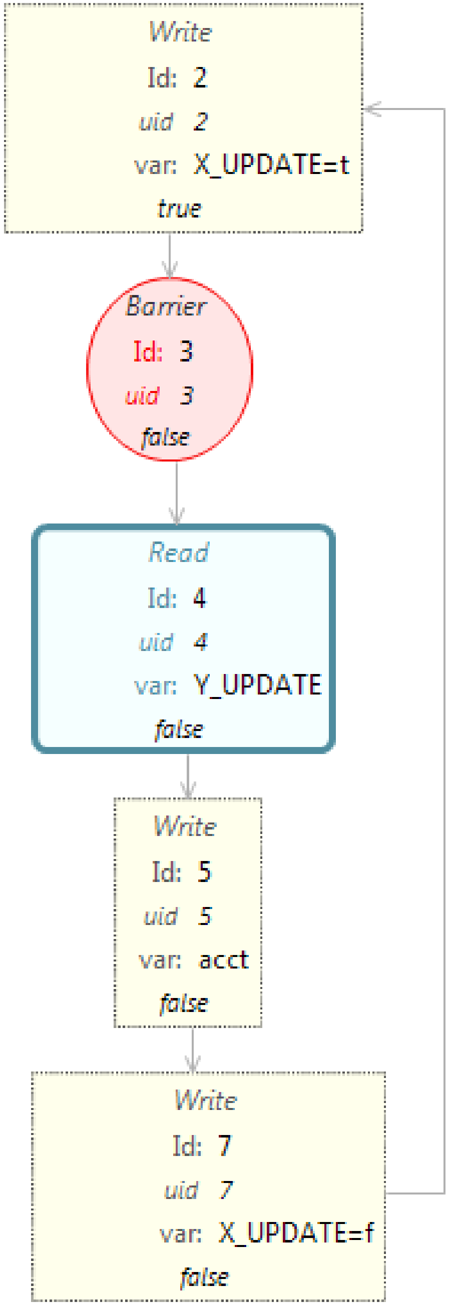
But this should be obvious!

- From inspection of the algorithm, it is clear that—given an infinite sized write buffer—that the write buffer could theoretically fill up forever.

What if we add a Memory Barrier?

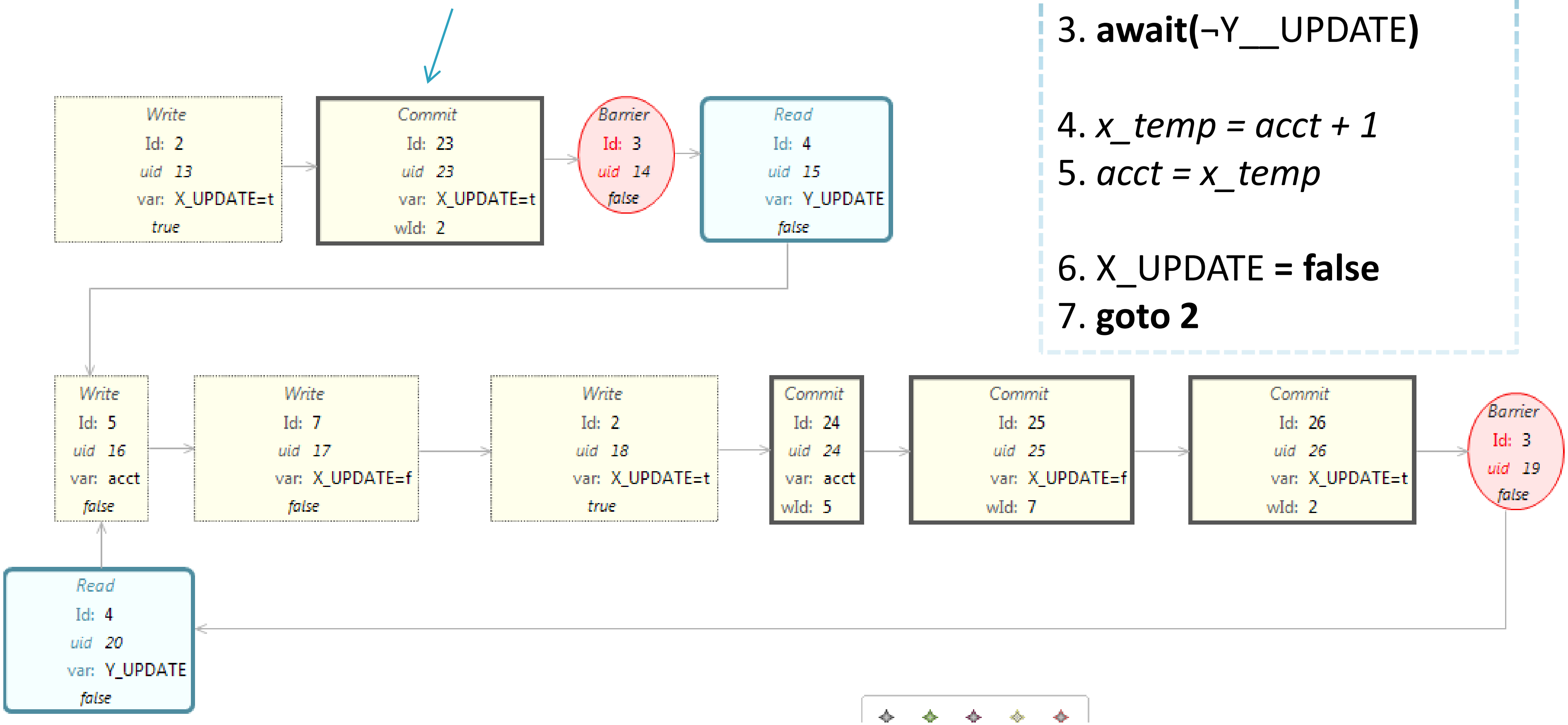
Banking With a Barrier

- A memory barrier instruction dequeues and commits all buffered writes
- It can be a very expensive instruction
- Now we are ensured that X's write, X_UPDATE = true, is visible to Y when X makes its next read.



This is the result...

Dark outlined boxes represent guaranteed commits



ATM X :

2. X_UPDATE **= true**
3. **BARRIER()**

3. **await(**¬Y__UPDATE**)**

4. *x_temp = acct + 1*
5. *acct = x_temp*

6. X_UPDATE **= false**
7. **goto 2**

4. Topics not covered

Topics not covered

- How TSO Helper works!
  - The key ‘TSO Helper’ lemma
- How TSO Helper can use knowledge about inter-process synchronization to better infer where writes must have been committed
- TSO Helper’s ability to filter displayed commits by the iteration they were generated in

5. Future Directions

Future Directions

- Algorithms to better layout transformed graphs
- Use of MMTF to connect 'Read/Write' graphs for different processes
- Case Study: Is TSO Helper practical and useful?

- Thanks!