

POPL 2017
Paris

A Promising Semantics for Relaxed-Memory Concurrency

Seoul National University
(Korea)



Jeehoon Kang
Chung-Kil Hur

MPI-SWS
(Germany)



MAX-PLANCK-GESELLSCHAFT

Ori Lahav
Viktor Vafeiadis
Derek Dreyer

Relaxed-Memory Concurrency

- **Semantics of multi-threaded programs?**
 - Sequential consistency (SC): simple but **expensive**
- **Relaxed memory models (C/C++, Java)**
 - Many consistency modes (cost vs. consistency tradeoff)
 - **Open problem:** what is the “**right**” semantics?

“Right” Concurrency Semantics?

- **Conflicting goals of “masters”**
- **Compiler/hardware:** validating optimizations (e.g. reordering, merging)
- **Programmer:** supporting reasoning principles (e.g. DRF theorem, program logic)

“Right” Concurrency Semantics?

- **Conflicting goals of “masters”**
- **Compiler/hardware:** validating optimizations (e.g. reordering, merging) 🥰 **Java memory model**
- **Programmer:** supporting reasoning principles (e.g. DRF theorem, program logic)

“Right” Concurrency Semantics?

- **Conflicting goals of “masters”**
- **Compiler/hardware:** validating optimizations (e.g. reordering, merging) 🥰 **Java memory model**
- **Programmer:** supporting reasoning principles (e.g. DRF theorem, program logic) 🥰 **C/C++ memory model**

“Right” Concurrency Semantics?

- **Conflicting goals of “masters”**
- **Compiler/hardware:** validating optimizations (e.g. reordering, merging) 🥰 **Java memory model**
- **Programmer:** supporting reasoning principles (e.g. DRF theorem, program logic) 🥰 **C/C++ memory model**

Key problem: “out-of-thin-air”

“Out-of-thin-air” problem (1/3)

Load-Buffering (LB)

Thread 1

$a = X$

$Y = a$

($a=b=42?$)

Thread 2

$b = Y$

$X = 42$

“Out-of-thin-air” problem (1/3)

Load-Buffering (LB)

Registers

Thread 1

$a = X$

$Y = a$

Thread 2

$b = Y$

$X = 42$

($a=b=42?$)

“Out-of-thin-air” problem (1/3)

Load Interfering (LB)

Registers

Shared Locations

Thread 1

Thread 2

$a = X$

$b = Y$

$Y = a$

$X = 42$

($a=b=42?$)

“Out-of-thin-air” problem (1/3)

Load Ordering (LB)

Registers

Shared Locations

Thread 1

Thread 2

$a = X$

$b = Y$

$Y = a$

$X = 42$

CII
Relaxed

($a=b=42?$)

“Out-of-thin-air” problem (1/3)

Load-Buffering (LB)

Thread 1

$a = X$

$Y = a$

($a=b=42?$)

Thread 2

$b = Y$

$X = 42$

“Out-of-thin-air” problem (1/3)

Load-Buffering (LB)

Thread 1

$a = X$

$Y = a$

($a=b=42?$)

Thread 2

$b = Y$

$X = 42$

Allowed by reordering
(Power/ARM)

$X = 42$

$b = Y$

“Out-of-thin-air” problem (1/3)

Load-Buffering (LB)

Thread 1

a = X
Y = a

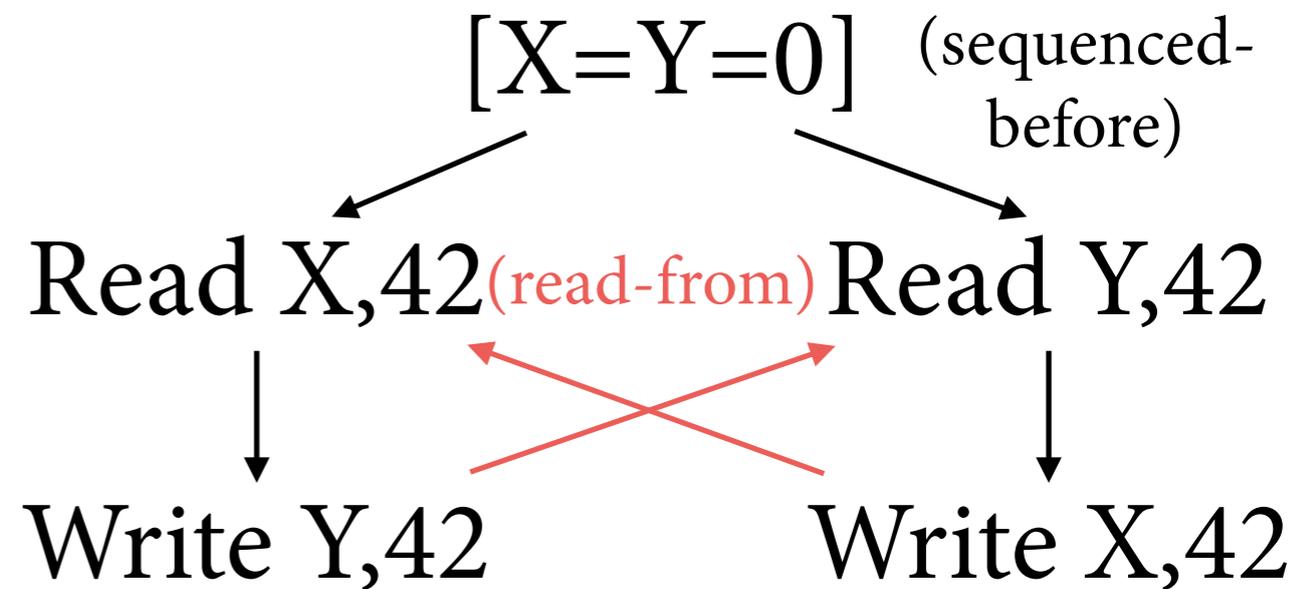
(a=b=42?)

Thread 2

b = Y
X = 42

Allowed by reordering
(Power/ARM)

X = 42
b = Y



Allowed by justification
(C/C++)

“Out-of-thin-air” problem (1/3)

Load-Buffering (LB)

Thread 1

a = X
Y = a

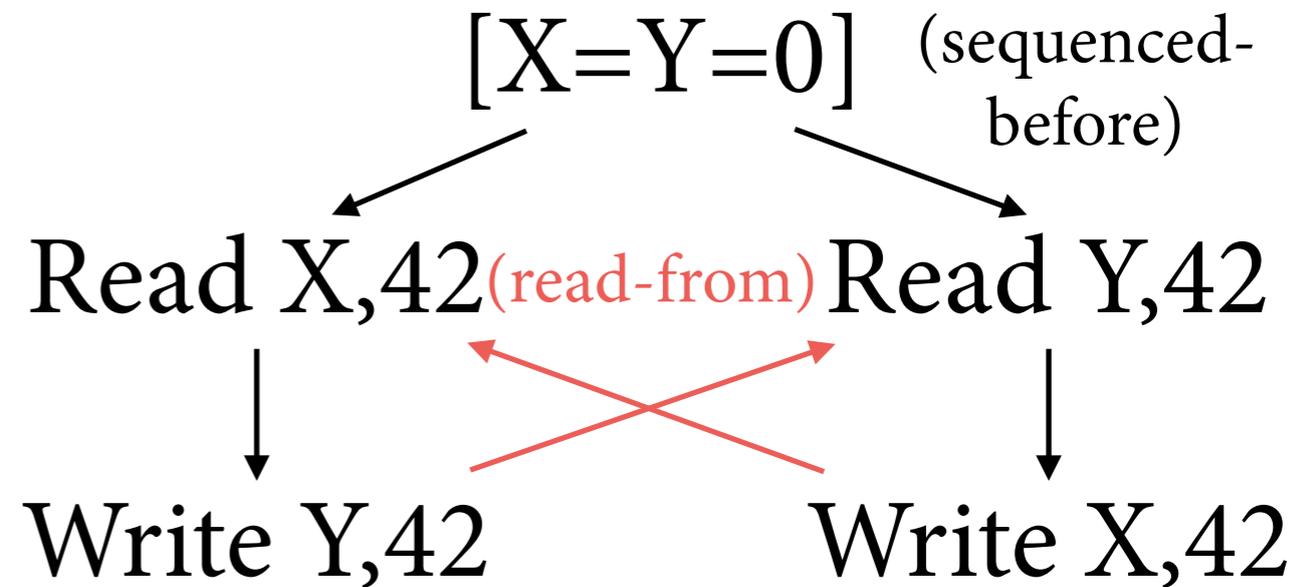
(a=b=42?)

Thread 2

b = Y
X = 42

Allowed by reordering
(Power/ARM)

X = 42
b = Y



Allowed by justification
(C/C++)

Justification is
too loose!

“Out-of-thin-air” problem (2/3)

Classic Out-of-thin-air (OOTA)

Thread 1

$a = X$

$Y = a$

($a=b=42?$)

Thread 2

$b = Y$

$X = b$

“Out-of-thin-air” problem (2/3)

Classic Out-of-thin-air (OOTA)

Thread 1

$a = X$

$Y = a$

Thread 2

$b = Y$

$X = b$

($a=b=42?$)

should be **forbidden**
42 is out-of-thin-air!

Reasoning principles
(e.g. invariant $a=b=X=Y=0$)

“Out-of-thin-air” problem (2/3)

Classic Out-of-thin-air (OOTA)

Thread 1

a = X
Y = a

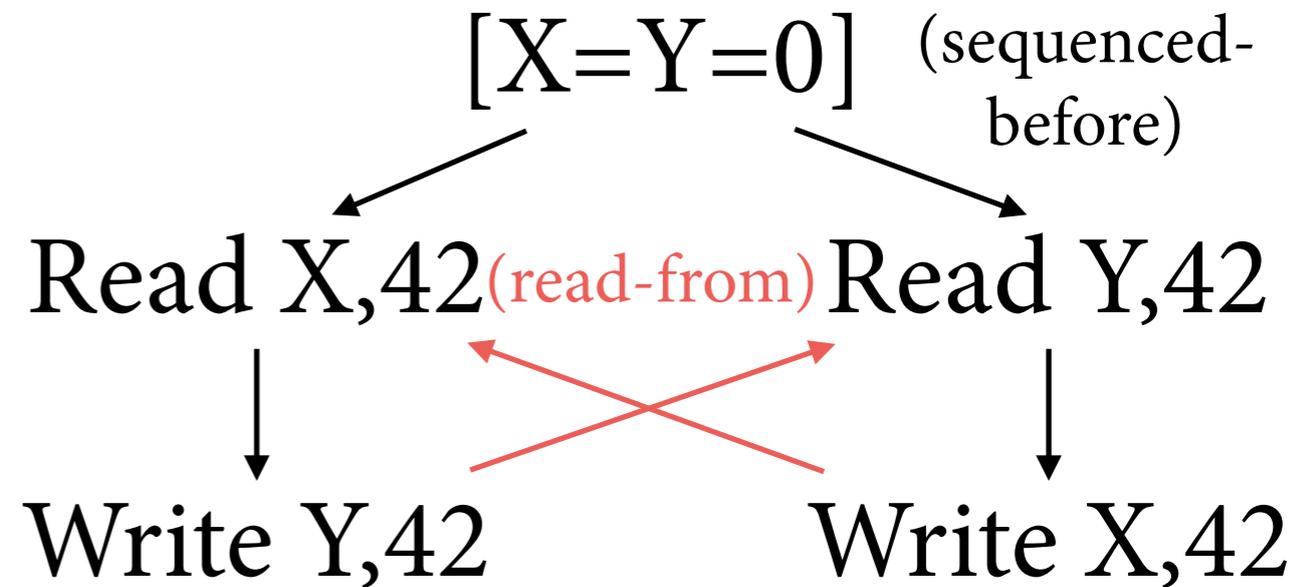
(a=b=42?)

Thread 2

b = Y
X = b

should be **forbidden**
42 is out-of-thin-air!

Reasoning principles
(e.g. invariant $a=b=X=Y=0$)



Allowed by justification
w/ same graph (C/C++)

“Out-of-thin-air” problem (2/3)

Classic Out-of-thin-air (OOTA)

Thread 1

a = X
Y = a

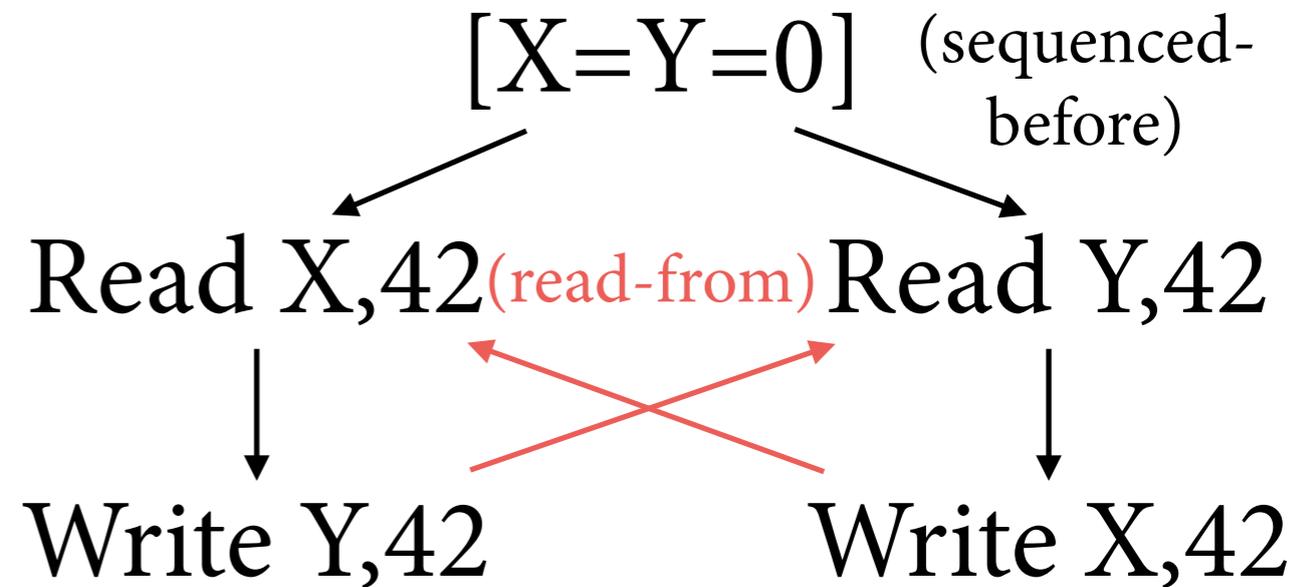
(a=b=42?)

Thread 2

b = Y
X = b

should be **forbidden**
42 is out-of-thin-air!

Reasoning principles
(e.g. invariant $a=b=X=Y=0$)



Allowed by justification
w/ same graph (C/C++)

What does
hardware do?

“Out-of-thin-air” problem (3/3)

Tracking Syntactic Dependency?

Thread 1

$a = X$

$Y = a$

($a=b=42?$)

Thread 2

$b = Y$

$X = 42$

Thread 1

$a = X$

$Y = a$

($a=b=42?$)

Thread 2

$b = Y$

$X = b$

“Out-of-thin-air” problem (3/3)

Tracking Syntactic Dependency?

Thread 1

$a = X$

$Y = a$

($a=b=42?$)

Thread 2

$b = Y$

$X = 42$

Thread 1

$a = X$

$Y = a$

($a=b=42?$)

Thread 2

$b = Y$

$X = b$ (dep.)

“Out-of-thin-air” problem (3/3)

Tracking Syntactic Dependency?

Thread 1

$a = X$

$Y = a$

($a=b=42?$)

Thread 2

$b = Y$

$X = 42$

allowed
in hardware

Thread 1

$a = X$

$Y = a$

($a=b=42?$)

Thread 2

~~$b = Y$~~

~~$X = b$~~ (dep.)

forbidden
in hardware

“Out-of-thin-air” problem (3/3)

Tracking Syntactic Dependency?

Thread 1

$a = X$

$Y = a$

($a=b=42?$)

Thread 2

$b = Y$

$X = b+42-b$

Thread 1

$a = X$

$Y = a$

($a=b=42?$)

Thread 2

~~$b = Y$~~
 ~~$X = b$~~ (dep.)

forbidden
in hardware

“Out-of-thin-air” problem (3/3)

Tracking Syntactic Dependency?

Thread 1

$a = X$

$Y = a$

($a=b=42?$)

Thread 2

~~$b = Y$~~

~~$X = b + 42 - b$~~

forbidden
in hardware

Thread 1

$a = X$

$Y = a$

($a=b=42?$)

Thread 2

~~$b = Y$~~

~~$X = b$~~ (dep.)

forbidden
in hardware

“Out-of-thin-air” problem (3/3)

Tracking Syntactic Dependency?

Thread 1

$a = X$
 $Y = a$

($a=b=42?$)

Thread 2

~~$b = Y$~~
 $X = b + 42 - b$

forbidden
in hardware

Thread 1

$a = X$
 $Y = a$

($a=b=42?$)

Thread 2

~~$b = Y$~~
 ~~$X = b$~~ (dep.)

forbidden
in hardware

could be optimized to “42”,
should be allowed in PL

“Out-of-thin-air” problem (3/3)

Tracking Syntactic Dependency?

Thread 1

$a = X$
 $Y = a$

($a=b=42?$)

Thread 2

~~$b = Y$~~
 $X = b + 42 - b$

forbidden
in hardware

Thread 1

$a = X$
 $Y = a$

($a=b=42?$)

Thread 2

~~$b = Y$~~
 $X = b$ (dep.)

forbidden
in hardware

could be optimized to “42”,
should be allowed in PL

Syntactic approach
doesn't work for PL!

“Out-of-thin-air” problem (3/3)

“A major open problem for PL semantics”
(Batty et al. ESOP 2015)

Thread 1

$a = X$
 $Y = a$

($a=b=42?$)

Thread 2

~~$b = Y$~~
 $X = b + 42 - b$

forbidden
in hardware

Thread 1

$a = X$
 $Y = a$

($a=b=42?$)

Thread 2

~~$b = Y$~~
 $X = b$ (dep.)

forbidden
in hardware

could be optimized to “42”,
should be allowed in PL

Syntactic approach
doesn't work for PL!

Promising Semantics

- Solving the **out-of-thin-air problem**
- Supporting **optimizations & reasoning principles**
- Covering most **C/C++ concurrency** features
- **Operational semantics w/o undefined behavior**

- Most results are **verified in Coq**
<http://sf.snu.ac.kr/promise-concurrency>



Key Idea: Promises

- A thread can **promise** to write $X=V$ in the future, after which **other threads can read** $X=V$.
- To avoid OOTA, the promising thread must **certify** that it can write $X=V$ **in isolation**.

Key Idea: Promises

Thread 1

$a = X$

$Y = a$

Thread 2

$b = Y$

$X = 42$

- A thread can **promise** to write $X=V$ in the future, after which **other threads can read** $X=V$.
- To avoid OOTA, the promising thread must **certify** that it can write $X=V$ **in isolation**.

Key Idea: Promises

Thread 1

$a = X$

$Y = a$

Thread 2

$b = Y$

$X = 42$

$X=42$ promised

- A thread can **promise** to write $X=V$ in the future, after which **other threads can read** $X=V$.
- To avoid OOTA, the promising thread must **certify** that it can write $X=V$ **in isolation**.

Key Idea: Promises

Thread 1
a = X
Y = a

Thread 2

b = Y

X = 42

X=42 promised

Certified:
T2 in isolation

- A thread can **promise** to write $X=V$ in the future, after which **other threads can read** $X=V$.
- To avoid OOTA, the promising thread must **certify** that it can write $X=V$ **in isolation**.

Key Idea: Promises

Thread 1

a = X
Y = a

Thread 2

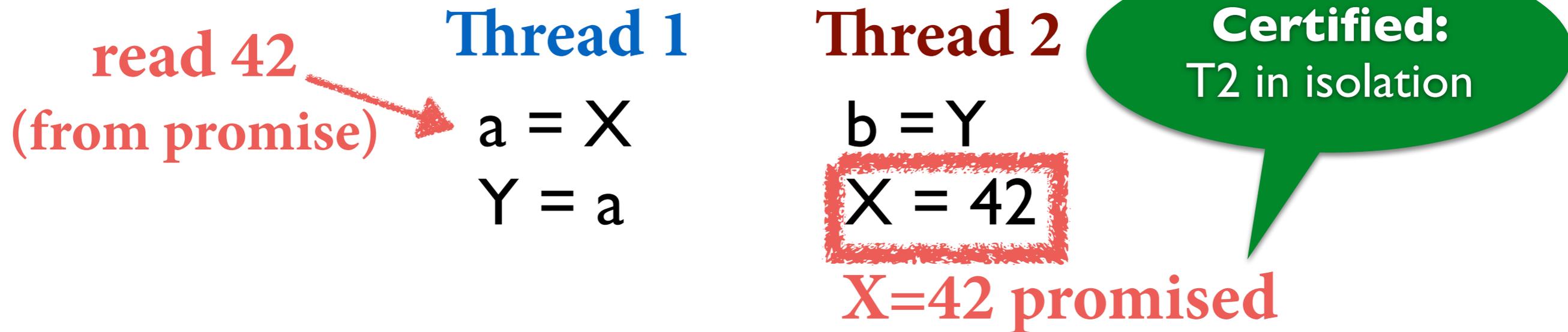
b = Y
X = 42

X=42 promised

Certified:
T2 in isolation

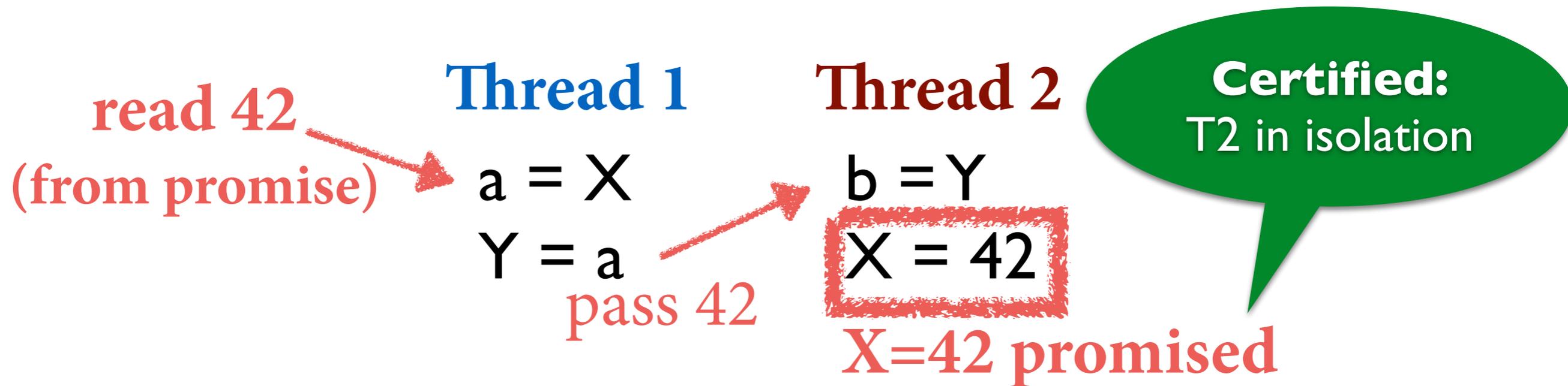
- A thread can **promise** to write $X=V$ in the future, after which **other threads can read** $X=V$.
- To avoid OOTA, the promising thread must **certify** that it can write $X=V$ **in isolation**.

Key Idea: Promises



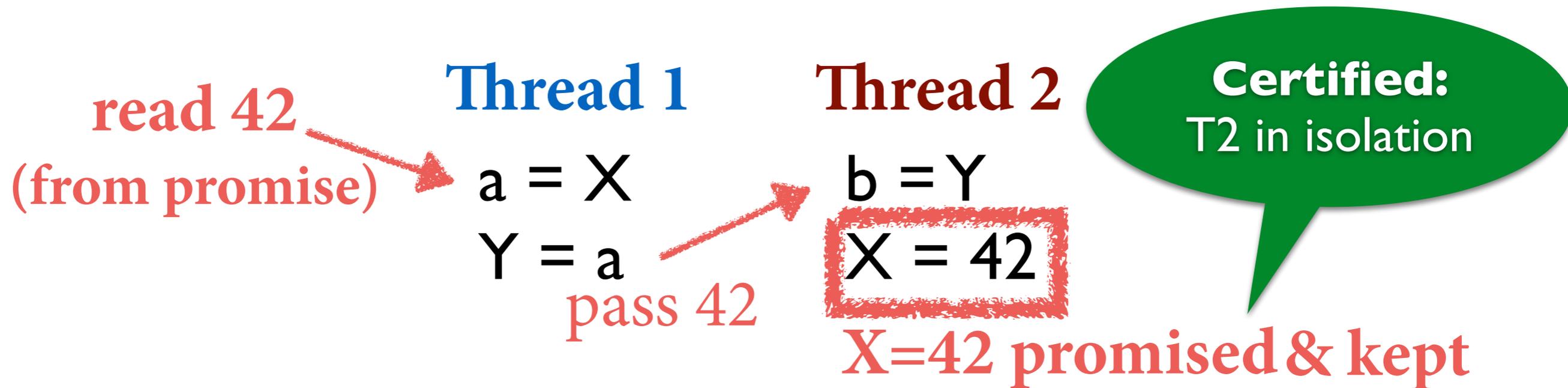
- A thread can **promise** to write $X=V$ in the future, after which **other threads can read** $X=V$.
- To avoid OOTA, the promising thread must **certify** that it can write $X=V$ **in isolation**.

Key Idea: Promises



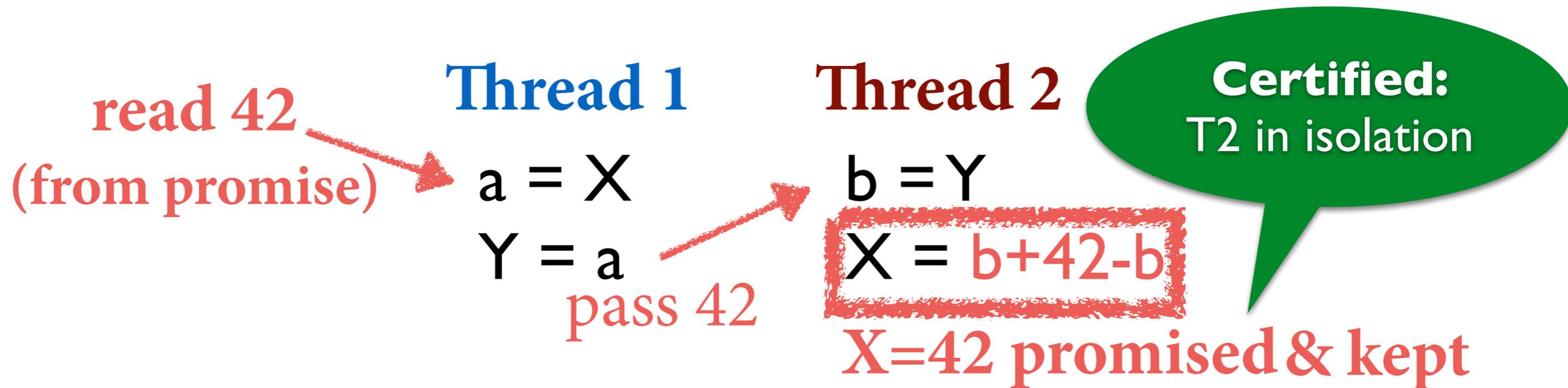
- A thread can **promise** to write $X=V$ in the future, after which **other threads can read** $X=V$.
- To avoid OOTA, the promising thread must **certify** that it can write $X=V$ **in isolation**.

Key Idea: Promises



- A thread can **promise** to write $X=V$ in the future, after which **other threads can read** $X=V$.
- To avoid OOTA, the promising thread must **certify** that it can write $X=V$ **in isolation**.

Key Idea: Promises



- A thread can **promise** to write $X=V$ in the future, after which **other threads can read** $X=V$.
- To avoid OOTA, the promising thread must **certify** that it can write $X=V$ **in isolation**.

Key Idea: Promises

Thread 1

$a = X$

$Y = a$

Thread 2

$b = Y$

$X = b$

- A thread can **promise** to write $X=V$ in the future, after which **other threads can read** $X=V$.
- To avoid OOTA, the promising thread must **certify** that it can write $X=V$ **in isolation**.

Key Idea: Promises

Thread 1
a = X
Y = a

Thread 2

b = Y

X = b

cannot promise X=42

- A thread can **promise** to write $X=V$ in the future, after which **other threads can read** $X=V$.
- To avoid OOTA, the promising thread must **certify** that it can write $X=V$ **in isolation**.

Promises: “Semantic Solution” to OOTA

Thread 1

$a = X$

$Y = a$

Thread 2

$b = Y$

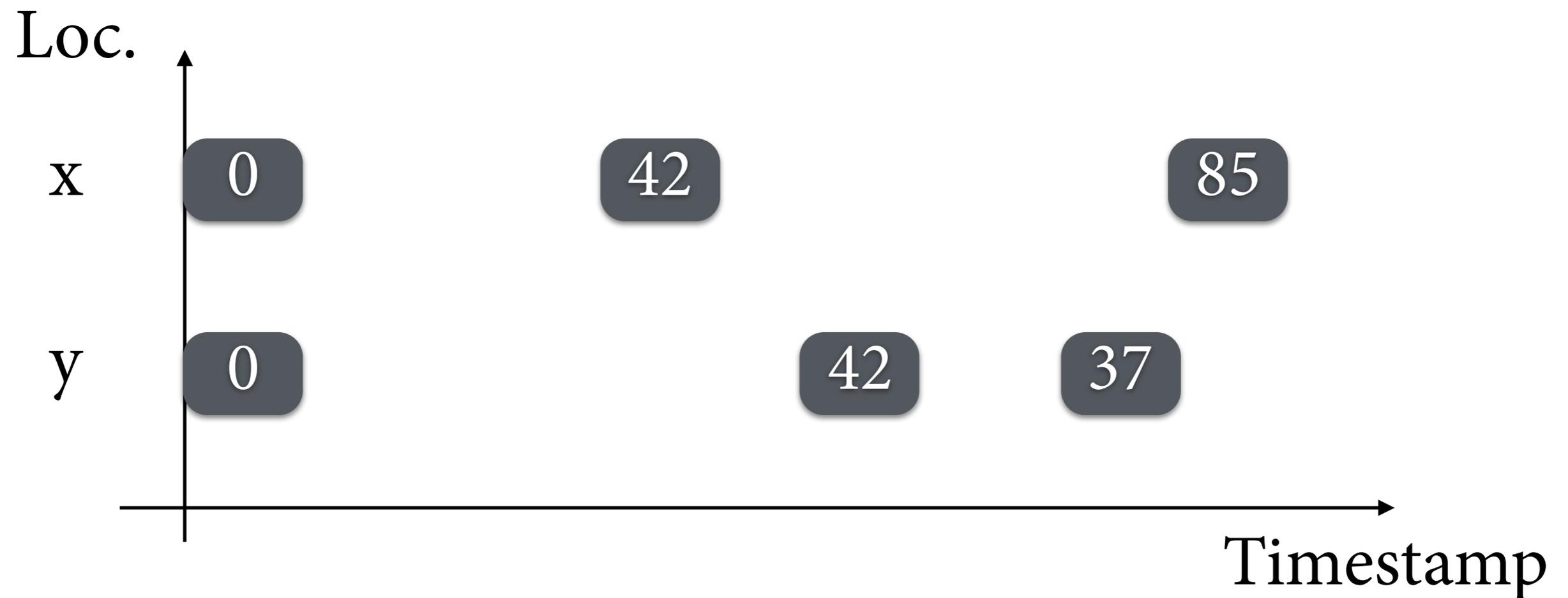
$X = b$

cannot promise $X=42$

- A thread can **promise** to write $X=V$ in the future, after which **other threads can read** $X=V$.
- To avoid OOTA, the promising thread must **certify** that it can write $X=V$ **in isolation**.

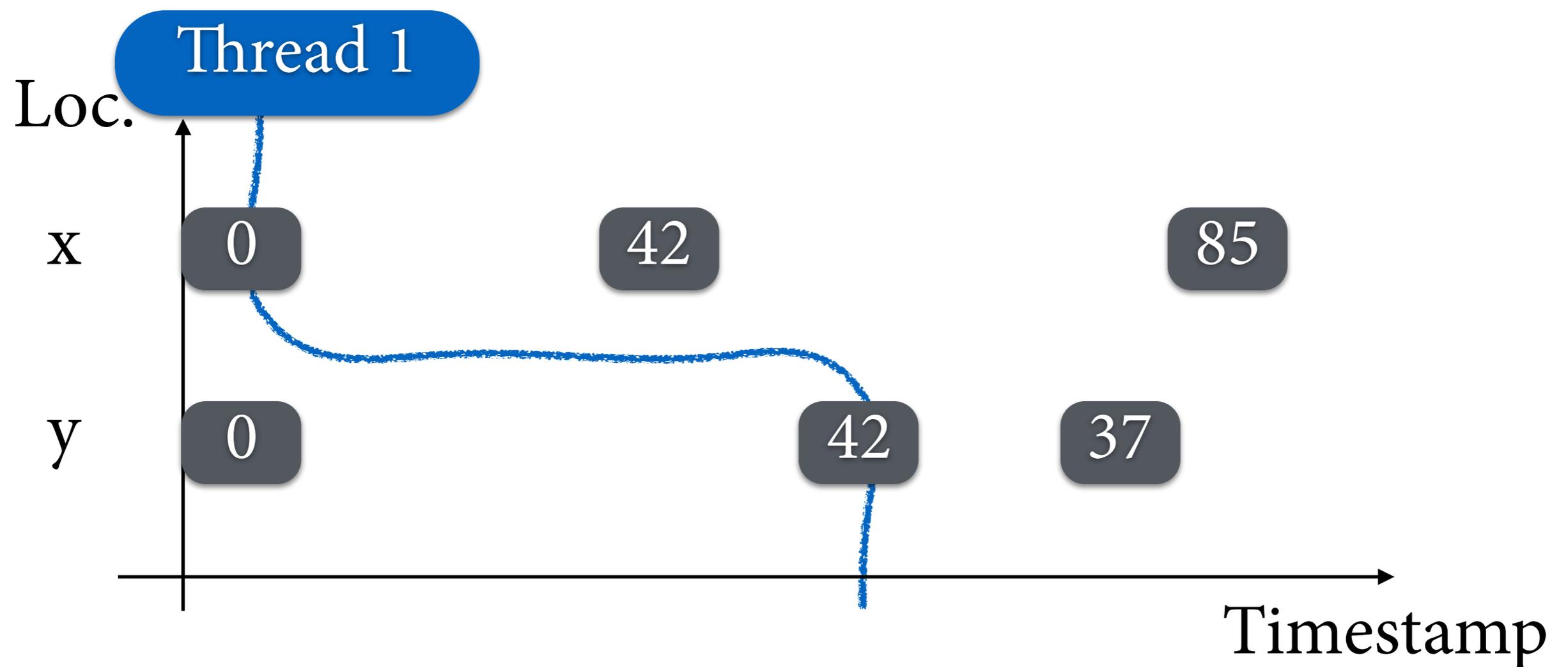
Basis: Operational Semantics

- Memory: pool of **messages** (loc, val, timestamp)
- Per-thread **view** on the memory



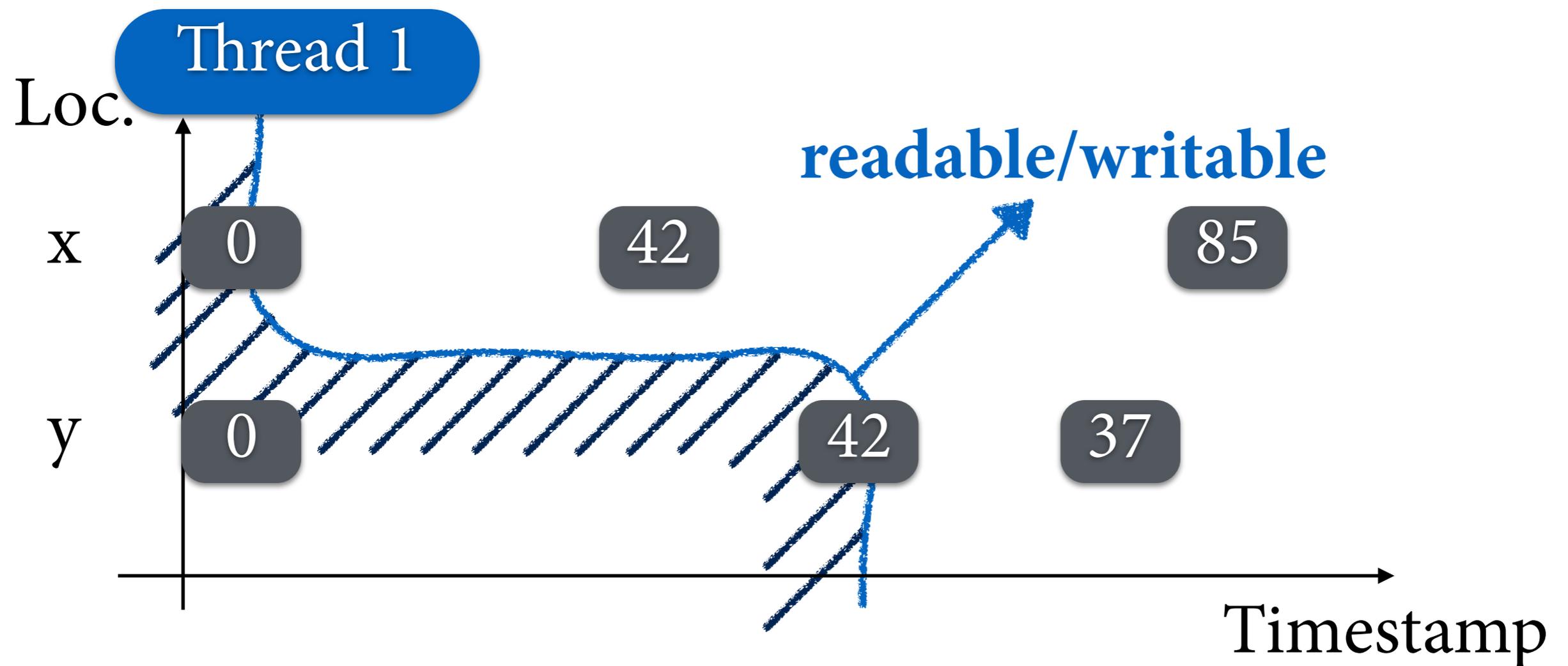
Basis: Operational Semantics

- Memory: pool of **messages** (loc, val, timestamp)
- Per-thread **view** on the memory



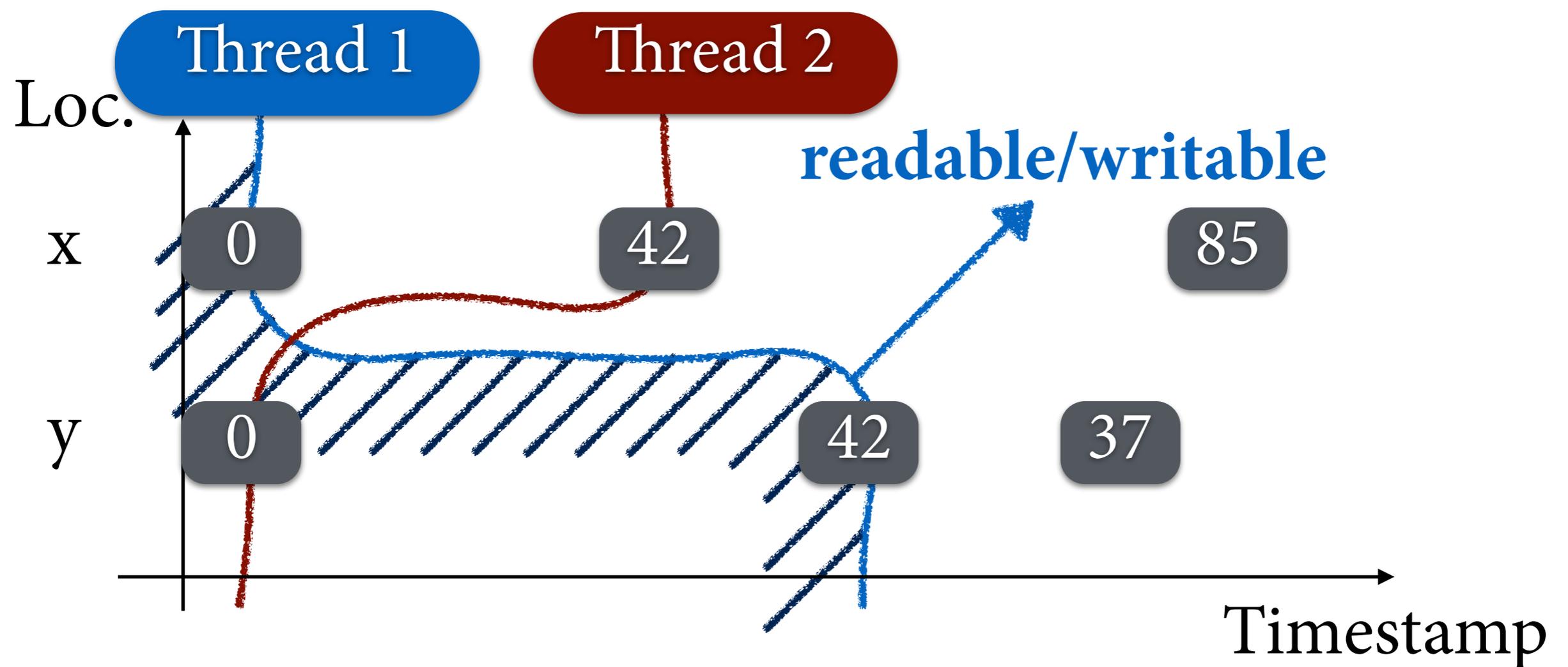
Basis: Operational Semantics

- Memory: pool of **messages** (loc, val, timestamp)
- Per-thread **view** on the memory



Basis: Operational Semantics

- Memory: pool of **messages** (loc, val, timestamp)
- Per-thread **view** on the memory



Example (1/3)

Store Buffering

Thread 1

→ $Y = 42$
 $a = X$

Thread 2

→ $X = 42$
 $b = Y$

(allowed: $a=b=0$)



Example (1/3)

Store Buffering

Thread 1
→ $Y = 42$
 $a = X$

Thread 2
→ $X = 42$
 $b = Y$

(allowed: $a=b=0$)

→
reorderable

(x86/Power/ARM)

$b = Y$
 $X = 42$



Example (1/3)

Store Buffering

Thread 1

→ $Y = 42$
 $a = X$

Thread 2

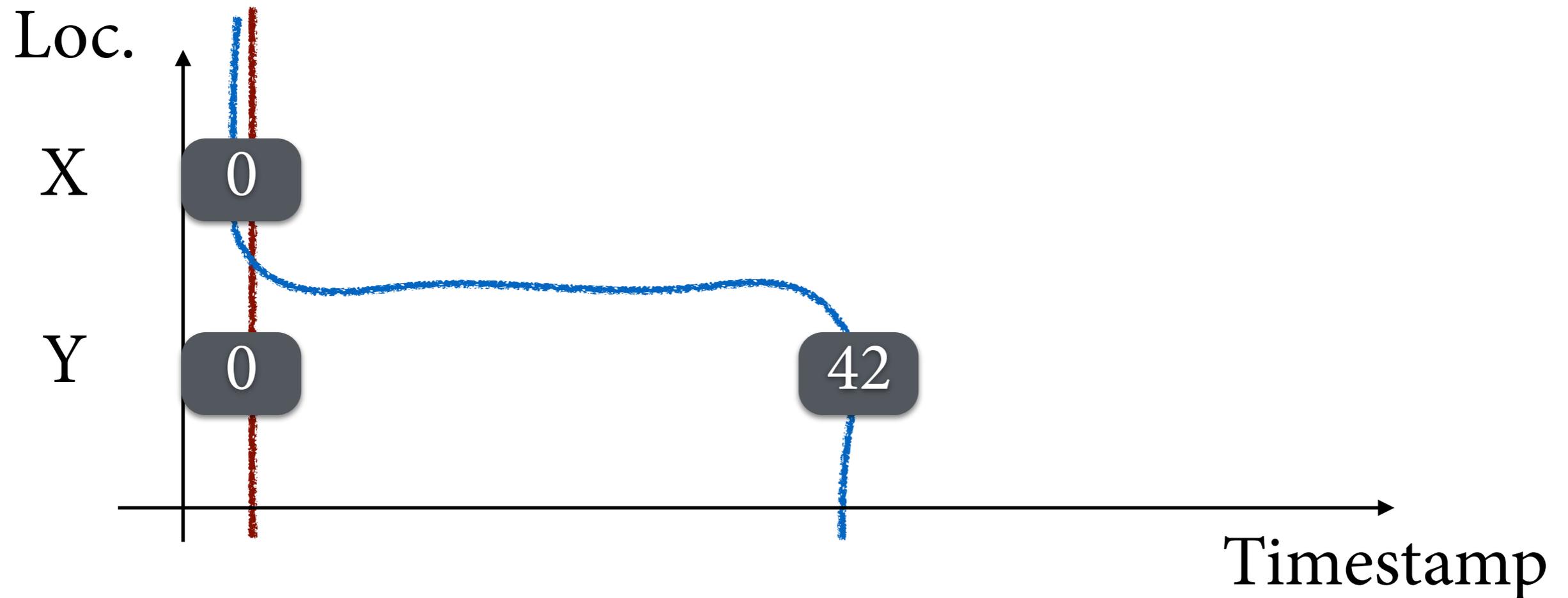
→ $X = 42$
 $b = Y$

(allowed: $a=b=0$)

→
reorderable

(x86/Power/ARM)

$b = Y$
 $X = 42$



Example (1/3)

Store Buffering

Thread 1

→ $Y = 42$
 $a = X$

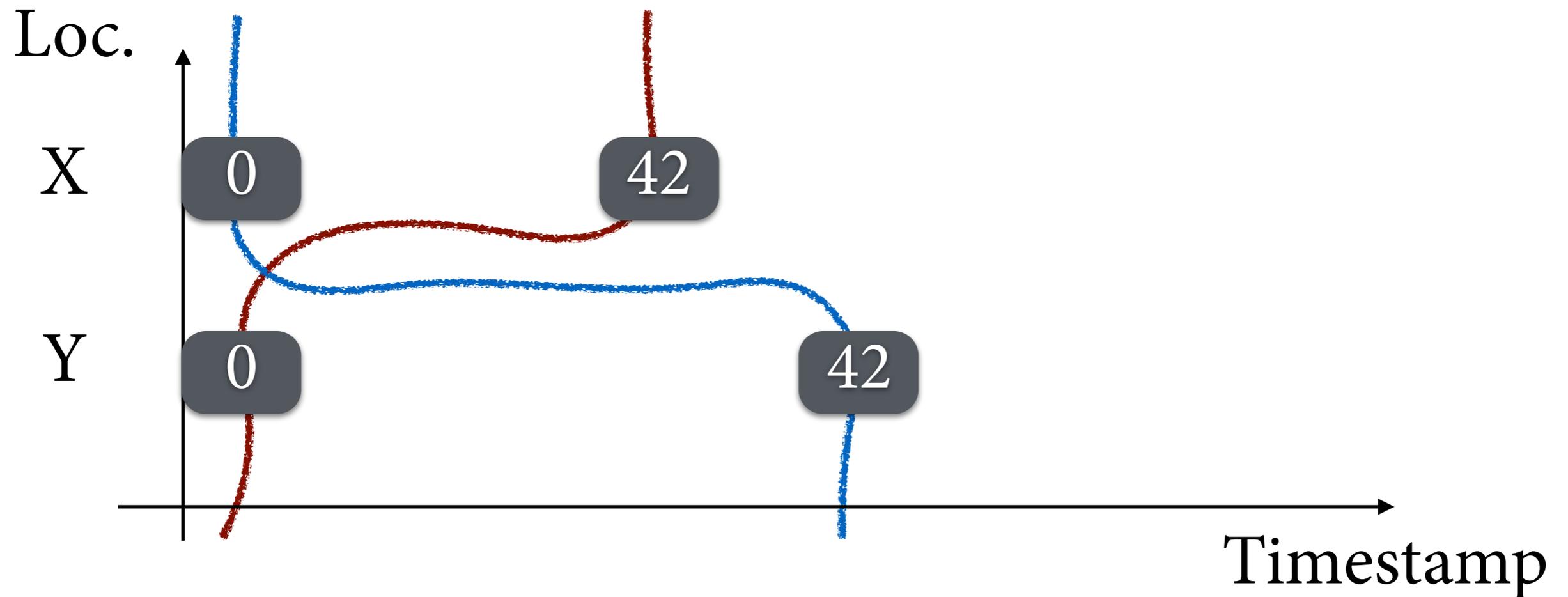
Thread 2

→ $X = 42$
 $b = Y$

(allowed: $a=b=0$)

→
reorderable
(x86/Power/ARM)

$b = Y$
 $X = 42$



Example (1/3)

Store Buffering

Thread 1

$Y = 42$
 $a = X$



Thread 2

$X = 42$
 $b = Y$



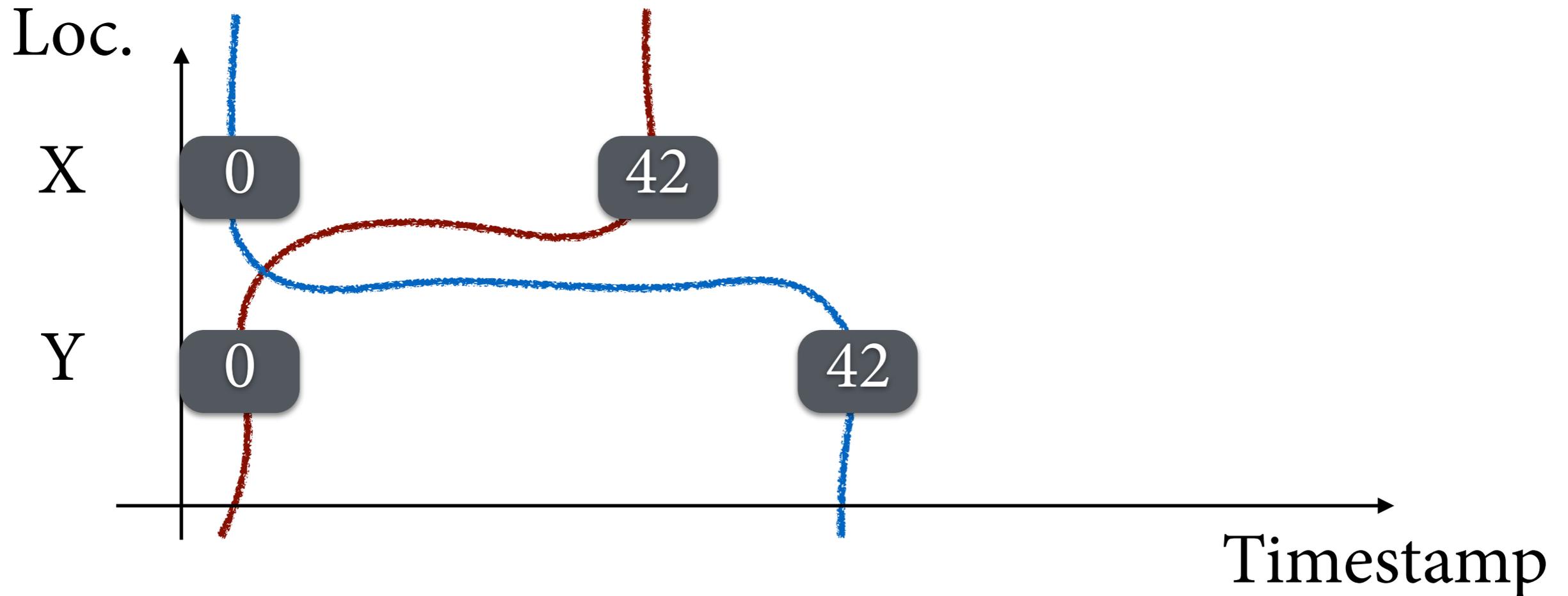
(allowed: $a=b=0$)



reorderable

(x86/Power/ARM)

$b = Y$
 $X = 42$



Example (1/3)

Store Buffering

Thread 1

$Y = 42$

$a = X$



Thread 2

$X = 42$

$b = Y$



(allowed: $a=b=0$)



reorderable

(x86/Power/ARM)

$b = Y$

$X = 42$

Loc.

X

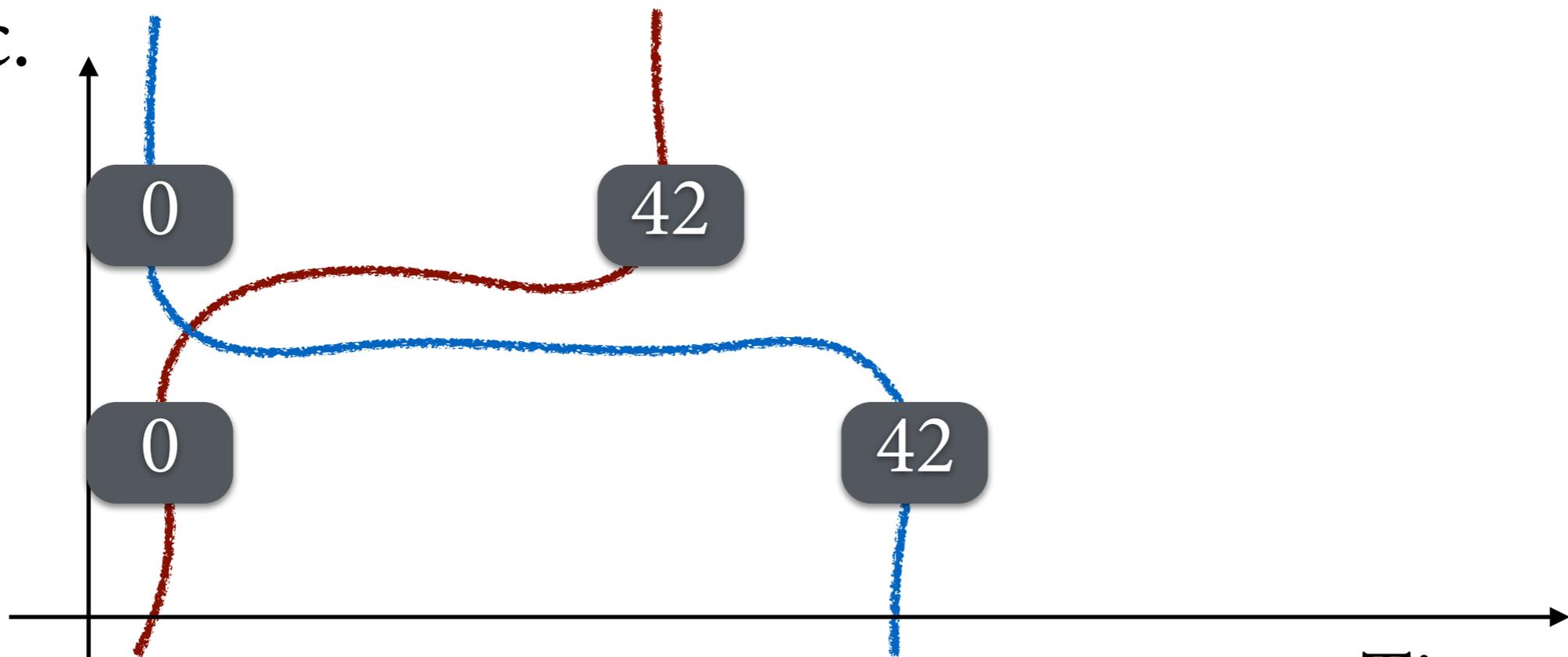
0

42

Y

0

42



Timestamp

Example (2/3)

Load Buffering (LB)

Thread 1



$a = X$

$Y = a$

Thread 2



$b = Y$

$X = 42$

(allowed: $a=b=42$)



Example (2/3)

Load Buffering (LB)

Thread 1



$a = X$

$Y = a$

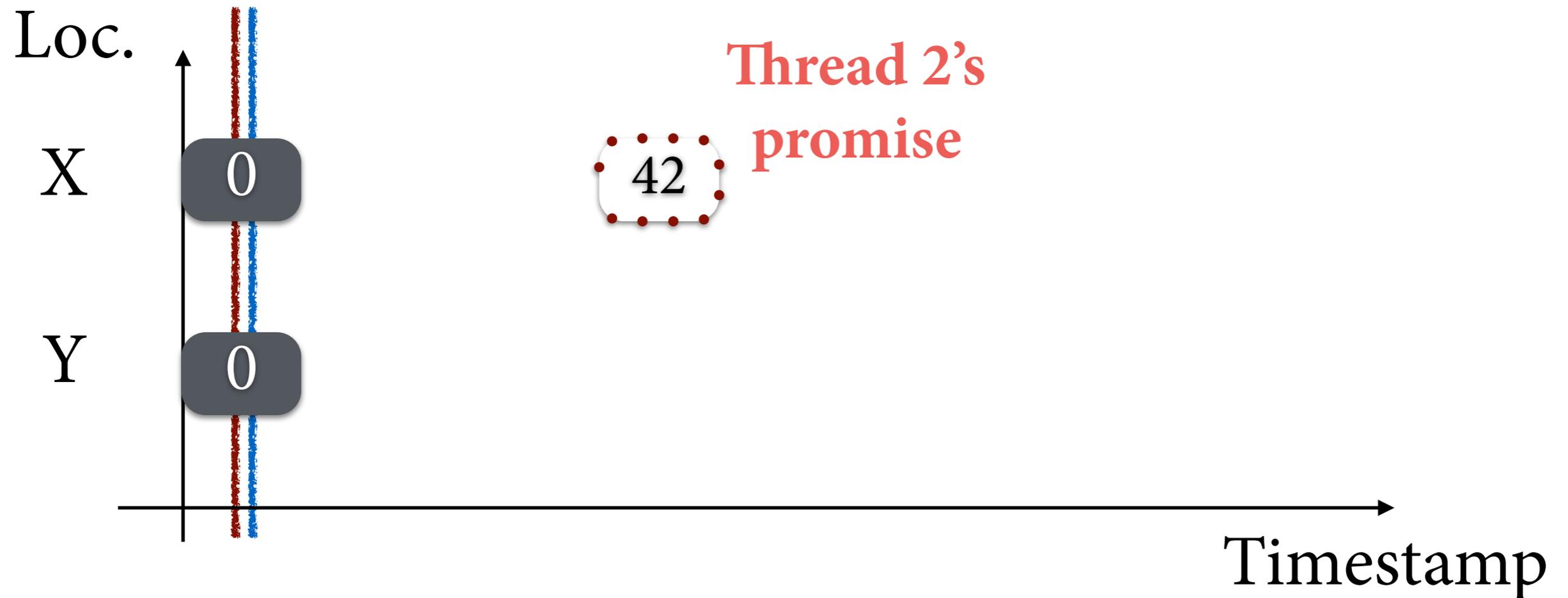
Thread 2



$b = Y$

$X = 42$

(allowed: $a=b=42$)



Example (2/3)

Load Buffering (LB)

Thread 1



$a = X$

$Y = a$

Thread 2



$b = Y$

$X = 42$

(allowed: $a=b=42$)

Loc.

X

0

Y

0

Thread 2's
promise

42

Thread 2 should be
able to write it
in isolation

Timestamp

Example (2/3)

Certification

Thread 2



$b = Y$

$X = 42$



Example (2/3)

Certification

Thread 2

→ $b = Y$
 $X = 42$



Example (2/3)

Certification

Thread 2

$b = Y$

$\rightarrow X = 42$



Example (2/3)

Certification

Thread 2

$b = Y$

$\rightarrow X = 42$



Example (2/3)

Load Buffering (LB)

Thread 1



$a = X$

$Y = a$

Thread 2



$b = Y$

$X = 42$

(allowed: $a=b=42$)

Loc.

X

0

Y

0

Thread 2's
promise

42

Promise is
certified!

Timestamp

Example (2/3)

Load Buffering (LB)

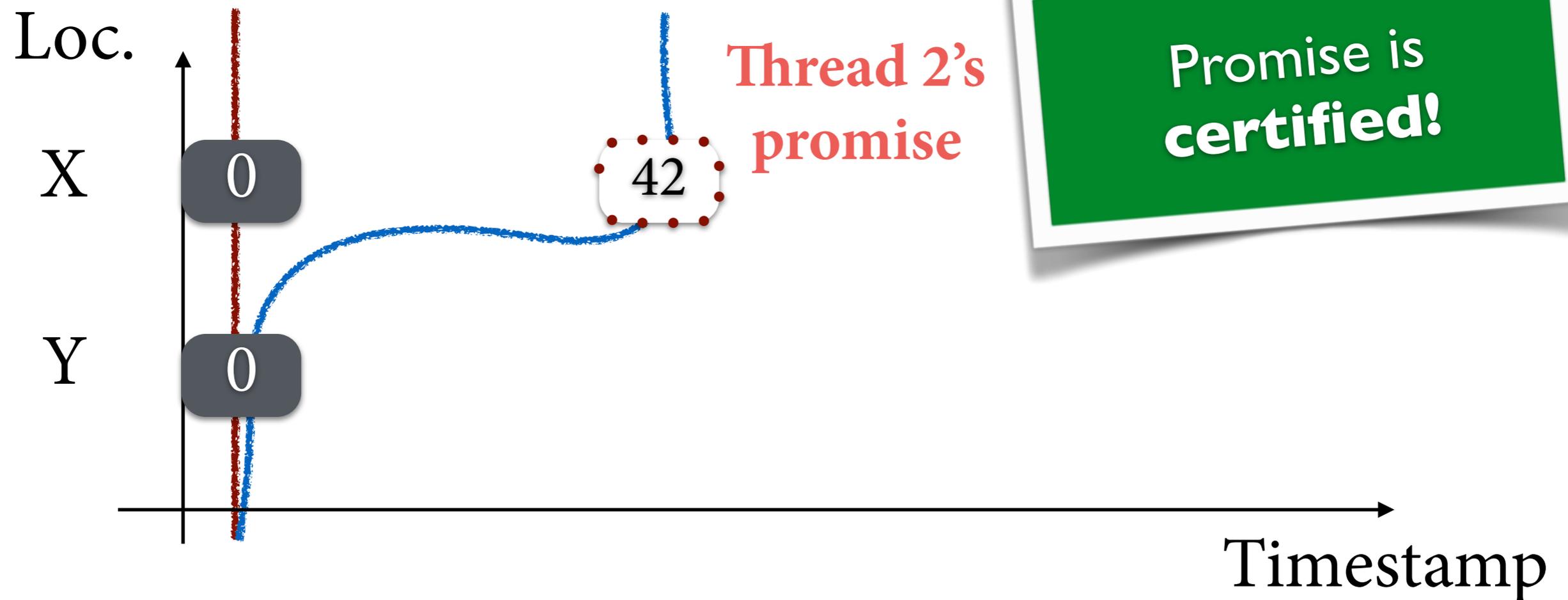
Thread 1

→ $a = X$
 $Y = a$

Thread 2

→ $b = Y$
 $X = 42$

(allowed: $a=b=42$)



Example (2/3)

Load Buffering (LB)

Thread 1

$a = X$

$Y = a$



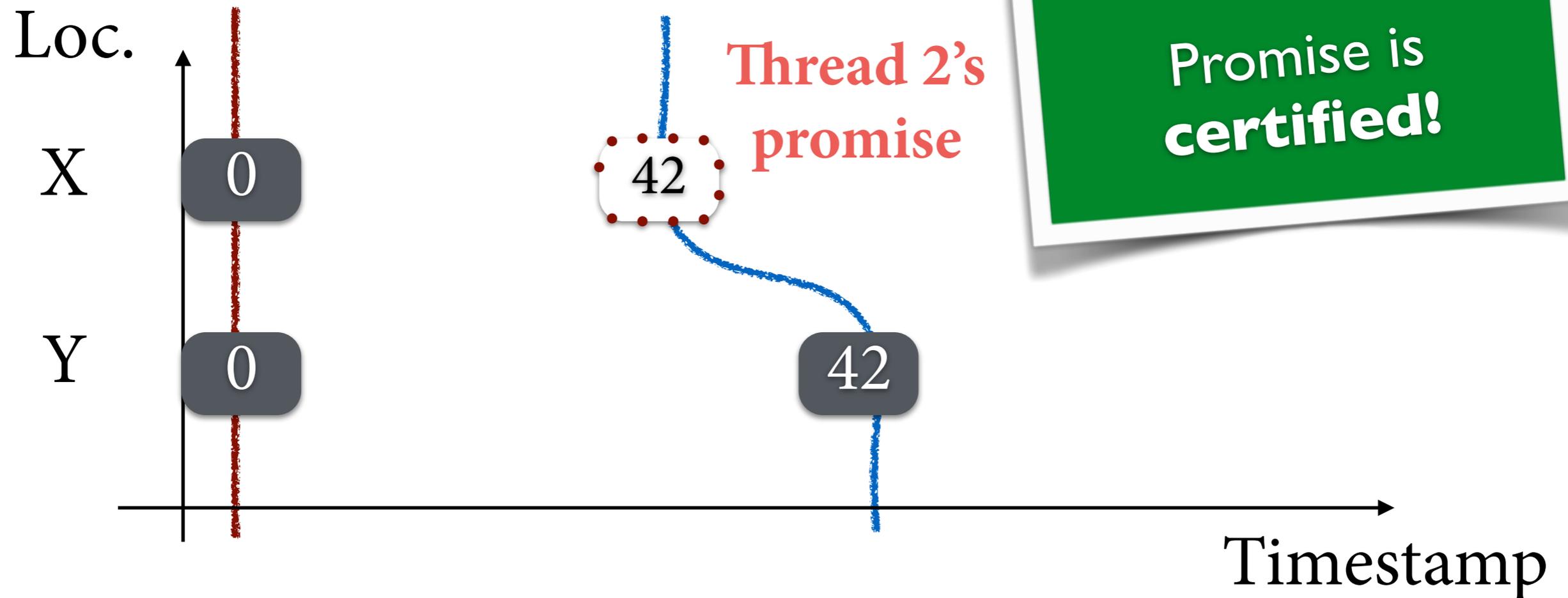
Thread 2



$b = Y$

$X = 42$

(allowed: $a=b=42$)



Example (2/3)

Load Buffering (LB)

Thread 1

$a = X$

$Y = a$



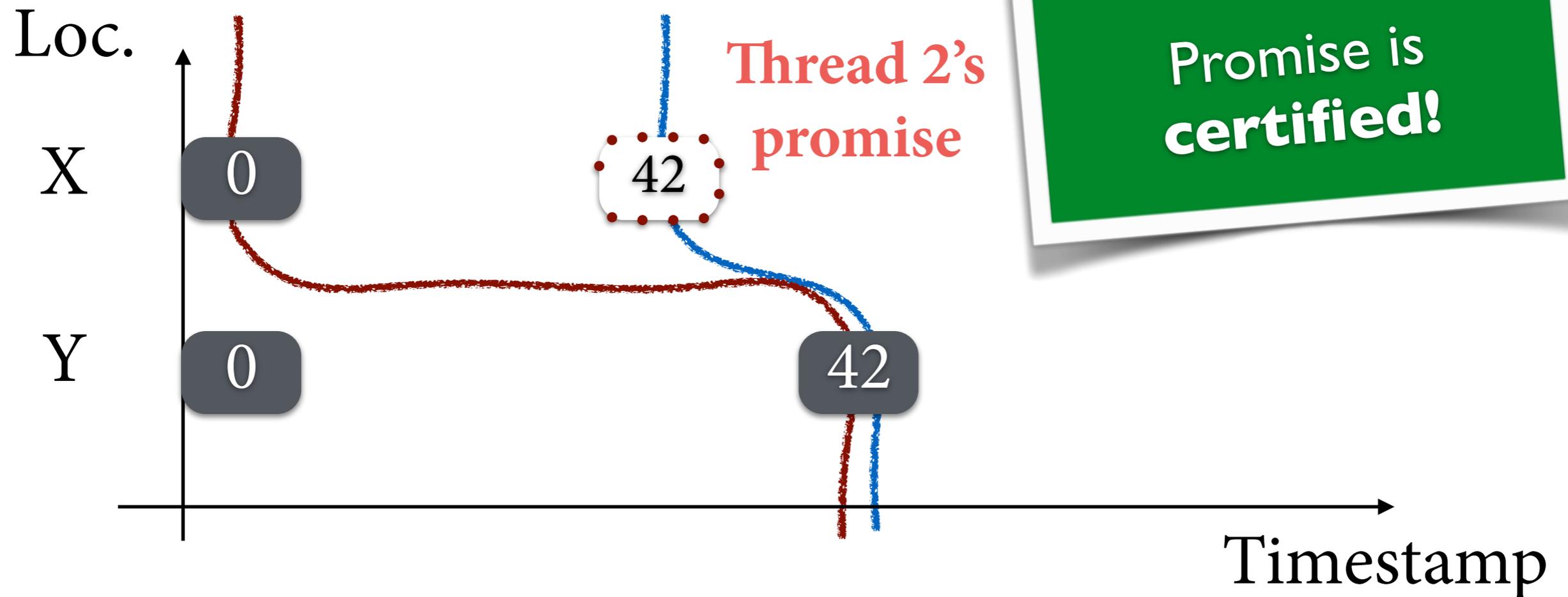
Thread 2

$b = Y$

$X = 42$



(allowed: $a=b=42$)



Example (2/3)

Load Buffering (LB)

Thread 1

$a = X$

$Y = a$



Thread 2

$b = Y$

$X = 42$



(allowed: $a=b=42$)

Loc.

X

0

Y

0

Thread 2's
promise

42

42

Promise is
certified!

Timestamp

Example (2/3)

Load Buffering (LB)

Thread 1

$a = X$

$Y = a$



Thread 2

$b = Y$

$X = b + 42 - b$



(allowed: $a=b=42$)

false dependency makes no difference!

Promise is certified!

Loc.

X

0

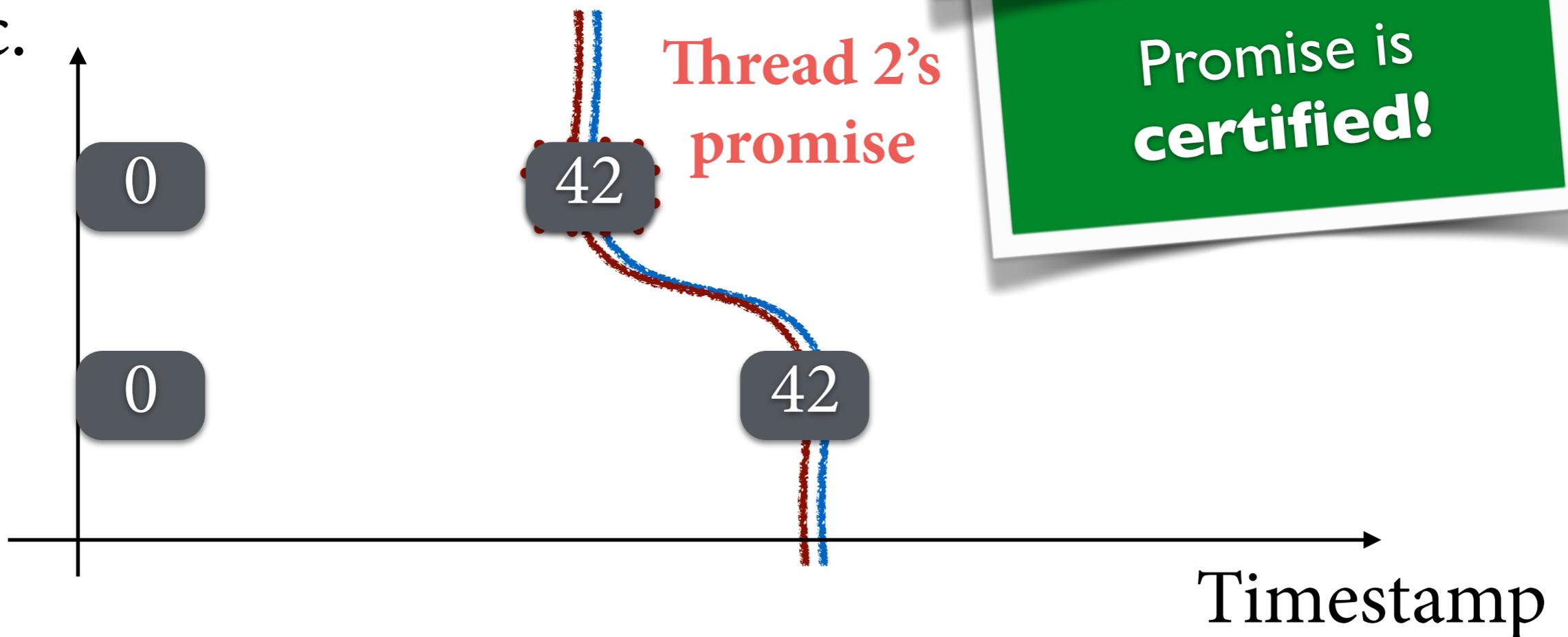
42

Y

0

42

Thread 2's promise



Example (3/3)

Classic Out-of-thin-air (OOTA)

Thread 1



$a = X$

$Y = a$

Thread 2



$b = Y$

$X = b$

(forbidden: $a=b=42$)



Example (3/3)

Classic Out-of-thin-air (OOTA)

Thread 1



$a = X$

$Y = a$

Thread 2



$b = Y$

$X = b$

(forbidden: $a=b=42$)

Loc.

X

0

Y

0

Thread 2's
promise?

42

Timestamp

Example (3/3)

Classic Out-of-thin-air (OOTA)

Thread 1

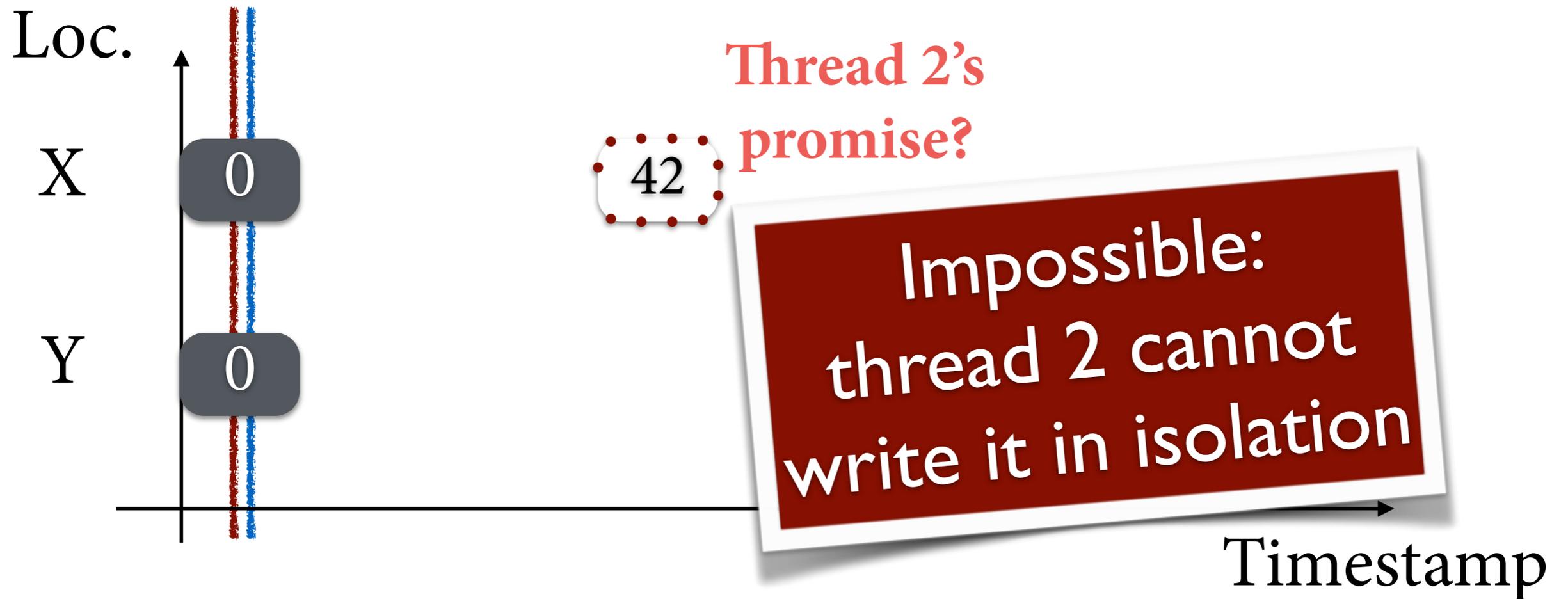
→

$a = X$
 $Y = a$

Thread 2 (forbidden: $a=b=42$)

→

$b = Y$
 $X = b$



Promises: "Semantic Solution" to OOTA

Thread 1



$a = X$

$Y = a$

Thread 2



$b = Y$

$X = b$

(forbidden: $a=b=42$)

Loc.

X

0

Y

0

Thread 2's
promise?

42

Impossible:
thread 2 cannot
write it in isolation

Timestamp

Compiler/HW Optimizations

- **Operational semantics** for C/C++ concurrency:
plain/relaxed/release/acquire r/w/u/fence, SC fence
- **Compiler optimizations** 
(reordering, merging, dead code elim., ...)
- **Compilation to x86**  & Power 

Results (2/2)

Reasoning Principles

- **DRF: Data Race Freedom \Rightarrow SC** 
 - DRF-PromiseFree: DRF \Rightarrow semantics w/o promises 
- **Invariant-based logic:** 
soundness of global invariant (e.g. $a=b=X=Y=0$)
- <http://sf.snu.ac.kr/promise-concurrency>



More comprehensive semantics for C/C++ concurrency

- **DRF: Data Race Freedom** \Rightarrow SC 
 - DRF-PromiseFree: DRF \Rightarrow semantics w/o promises 
- **Invariant-based logic:** 
soundness of global invariant (e.g. $a=b=X=Y=0$)
- <http://sf.snu.ac.kr/promise-concurrency>



Future Work

- Supporting **SC reads & writes**
(We found a **flaw in C/C++11 on SC**)
- Supporting **consume** reads
- Compilation to **ARMv8**
- Developing a rich **program logic & Verifying** fine-grained concurrent programs