Compositional reasoning about concurrent libraries on the axiomatic TSO memory model

Artem Khyzha
IMDEA Software Institute, Madrid, Spain

Joint work with Alexey Gotsman (IMDEA Software)

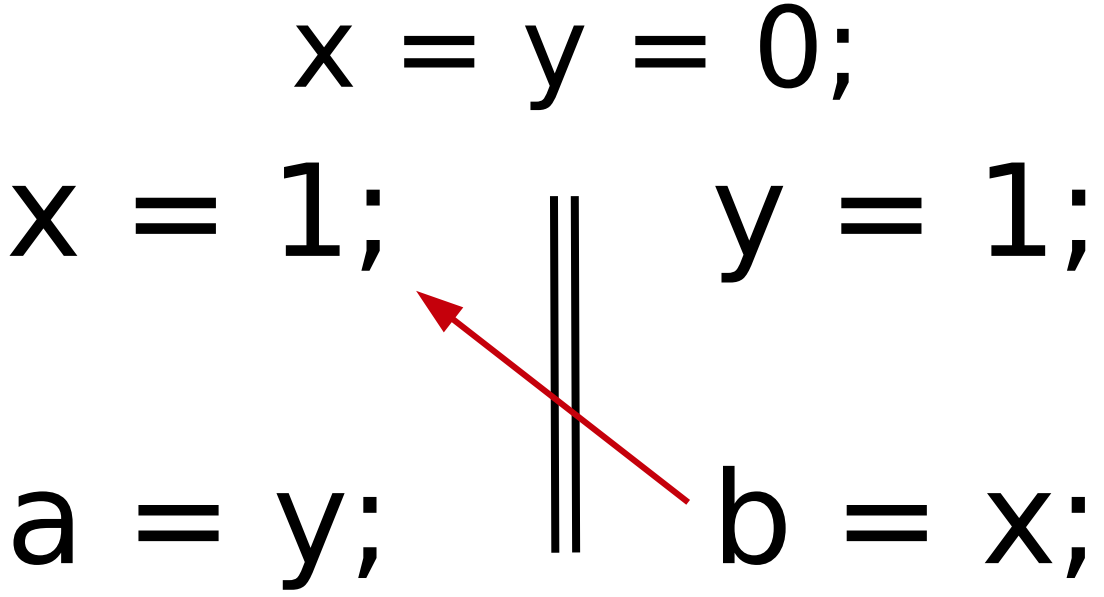
Weak memory

$$x = y = 0;$$

$$\begin{array}{l|l} x = 1; & y = 1; \\ \\ a = y; & b = x; \end{array}$$

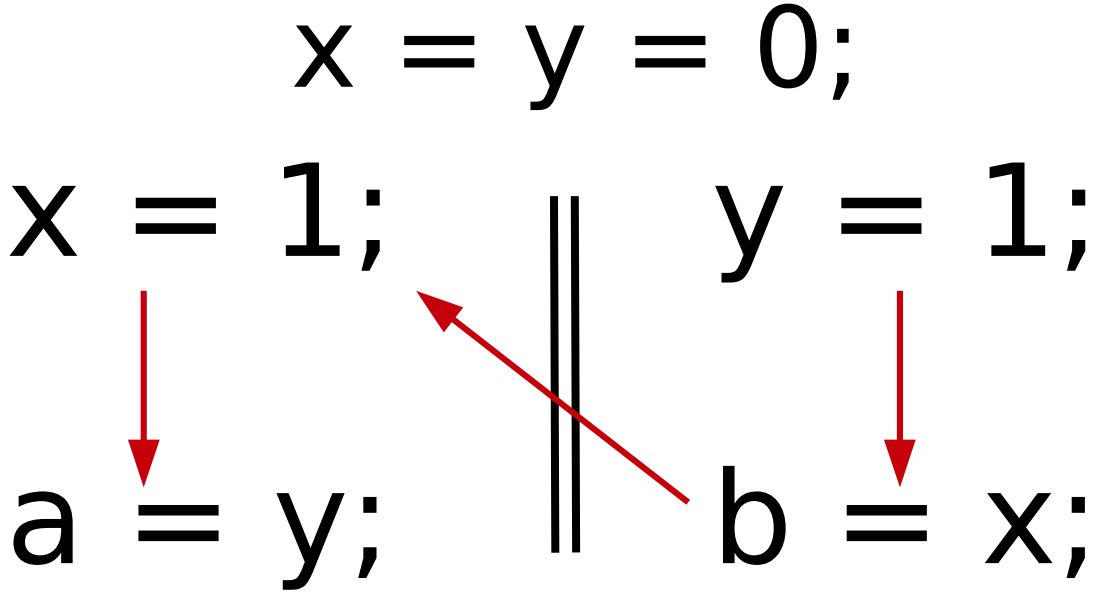
$$\{\ a = b = 0\ \}$$

Weak memory



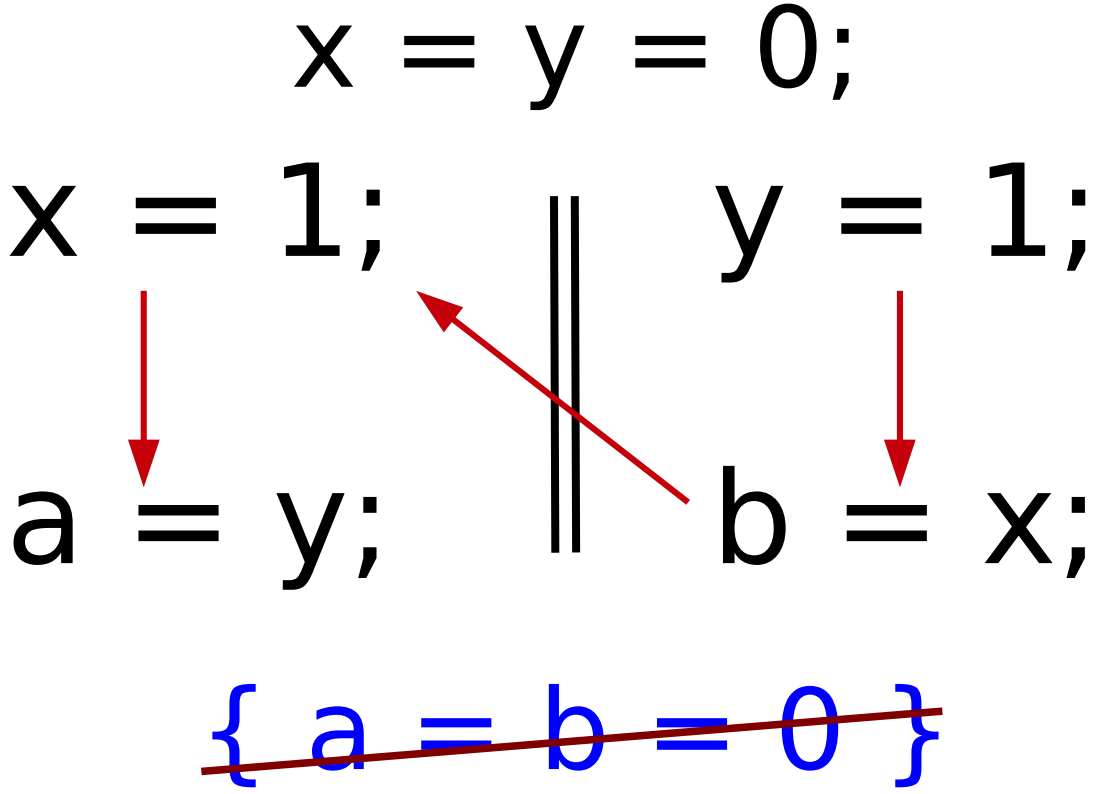
{ a = b = 0 }

Weak memory

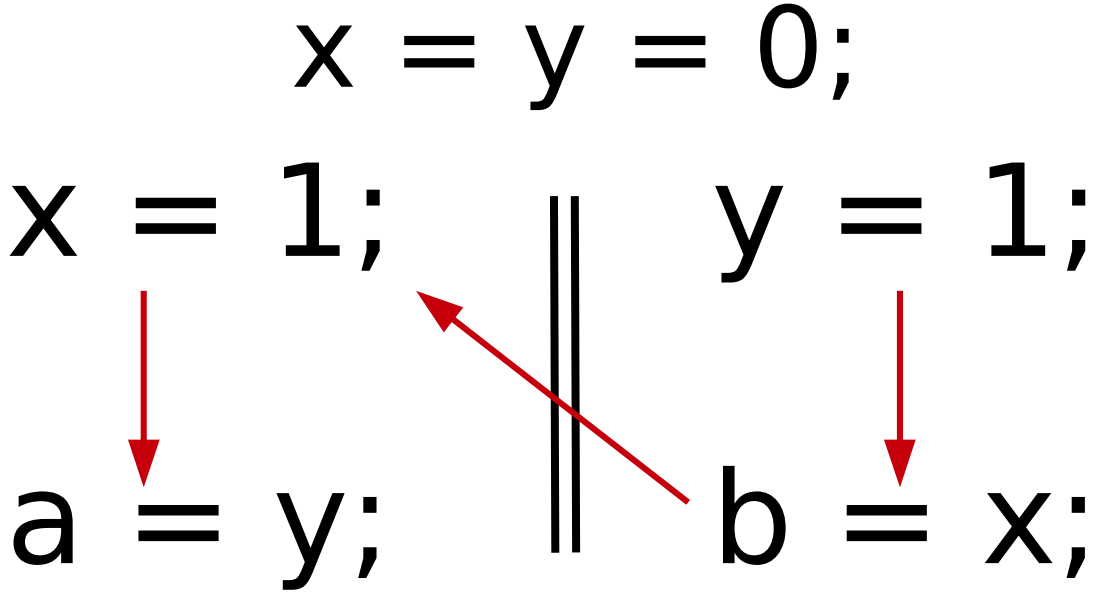


{ a = b = 0 }

Weak memory



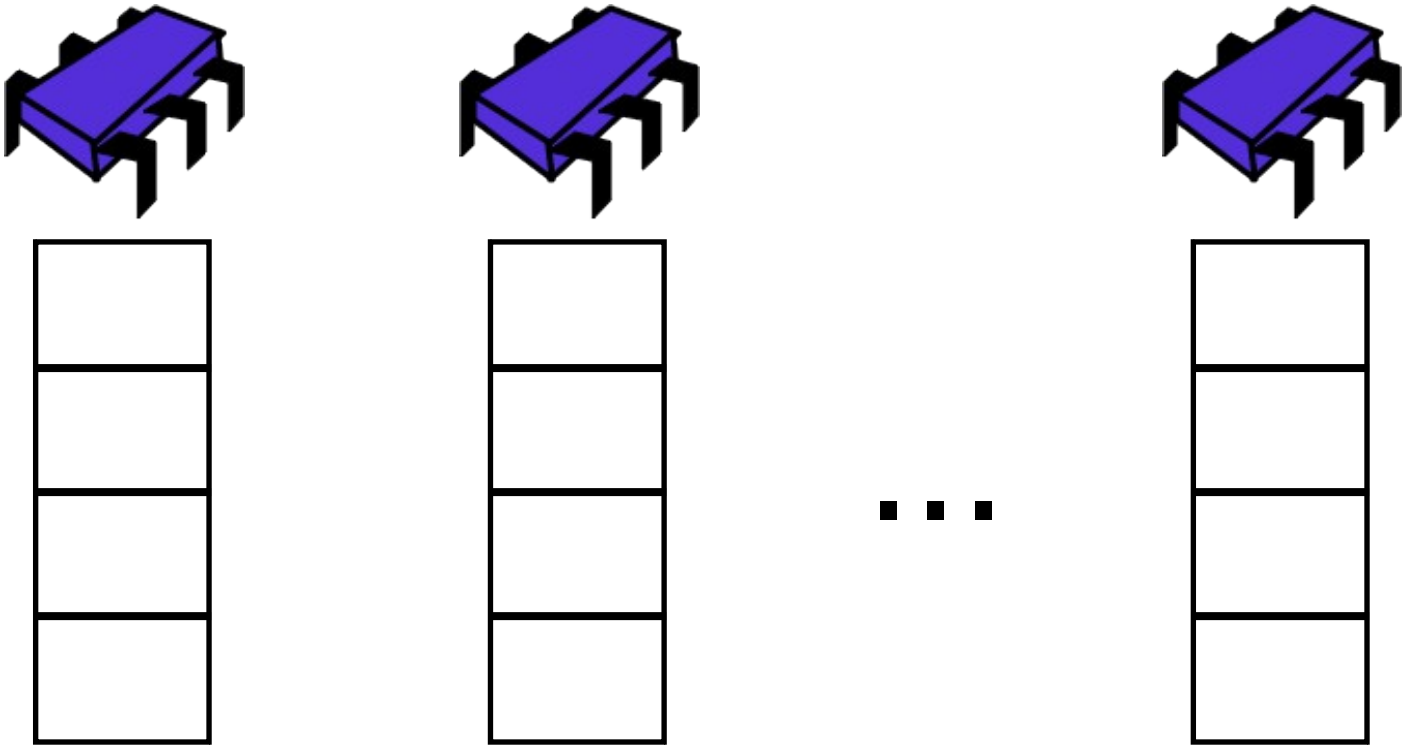
Weak memory



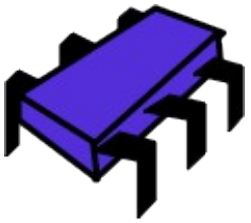
{ a = b = 0 }

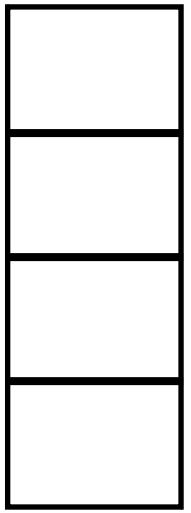
Possible on a weak memory model

The TSO memory model (x86)



The TSO memory model (x86)



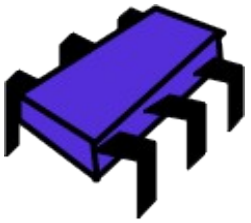


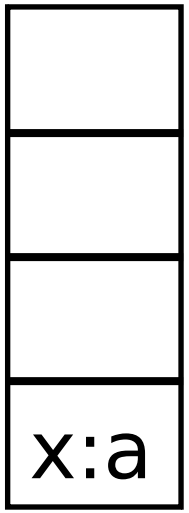
```
*x = a;
*y = b;
```

Writes stored into the write buffer in the order of issue

RAM

The TSO memory model (x86)



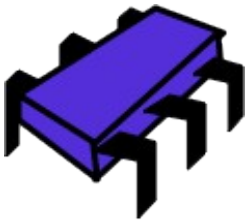


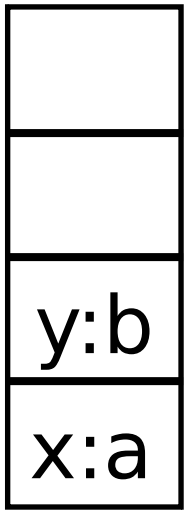
*x = a;
*y = b;

Writes stored into the write buffer in the order of issue

RAM

The TSO memory model (x86)



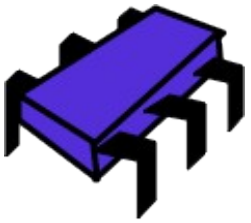


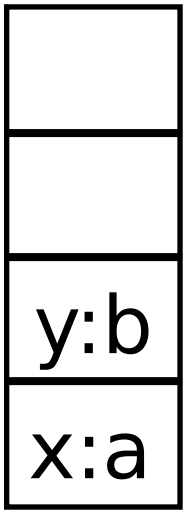

```
*x = a;
*y = b;
```

Writes stored into the write buffer in the order of issue

RAM

The TSO memory model (x86)

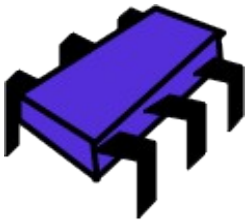




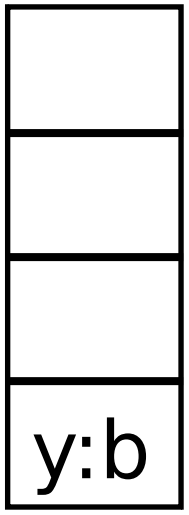

```
*x = a;
*y = b;
```

Writes flushed in FIFO order

The TSO memory model (x86)

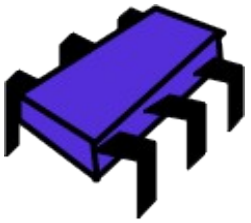


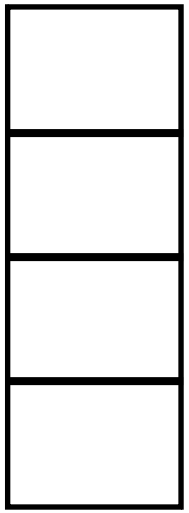
```
*x = a;
*y = b;
```



Writes flushed in FIFO order

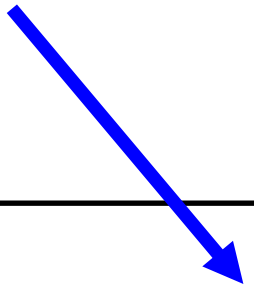
The TSO memory model (x86)

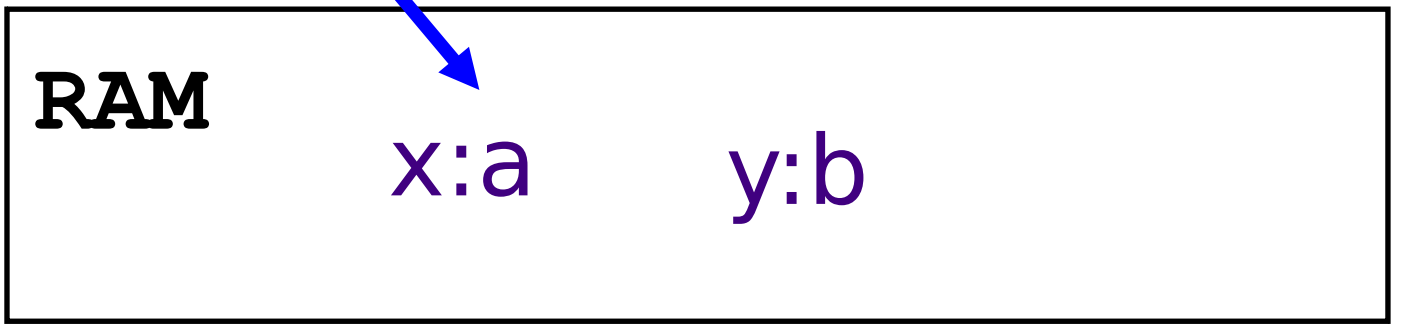




```
*x = a;
*y = b;
```

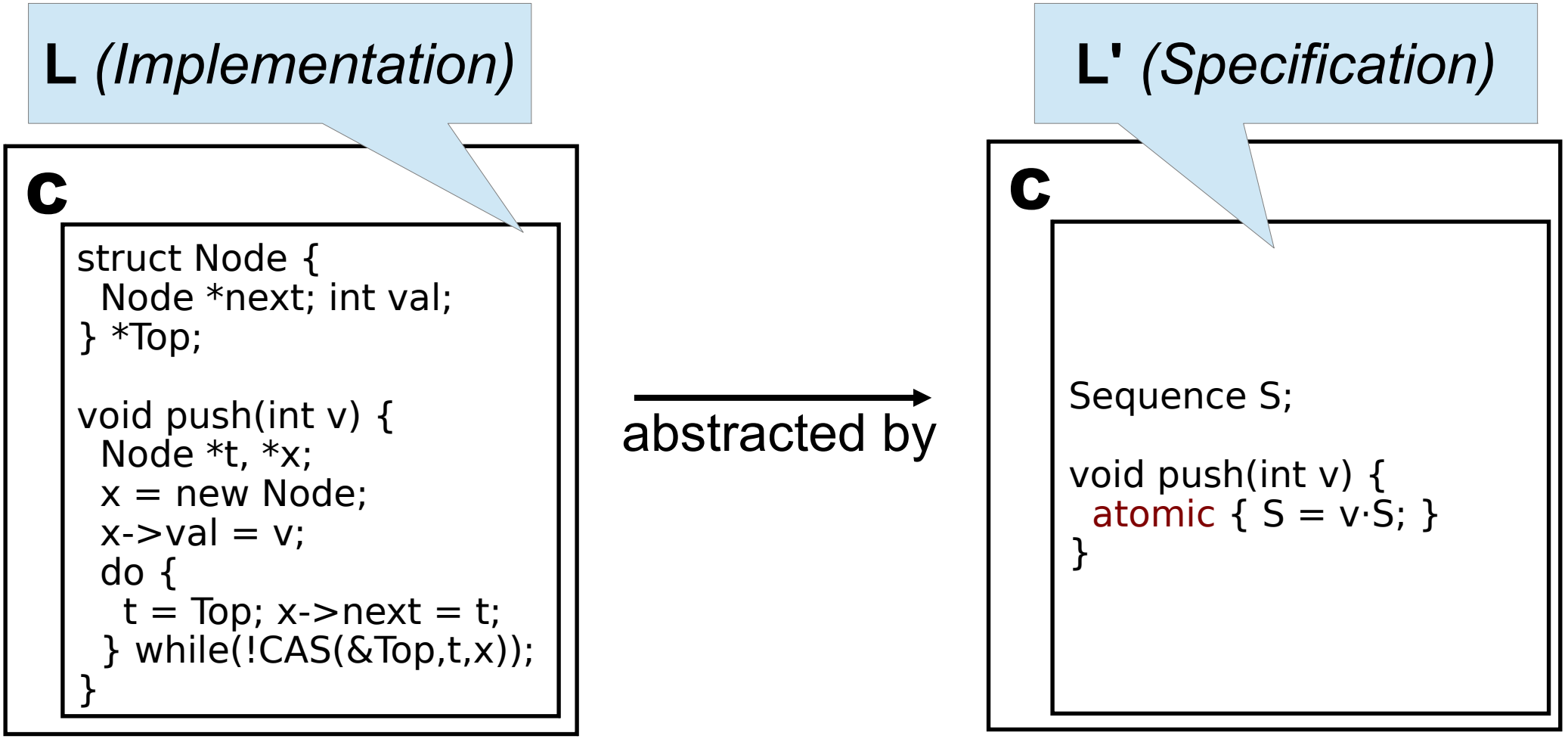
Writes flushed in FIFO order





Our goal

- Is compositional reasoning on weak memory possible?



Our goal

- Is compositional reasoning on weak memory possible?

Linearizability
[Herlihy, Wing]

L *(Implementation)*

c

```
struct Node {
  Node *next; int val;
} *Top;

void push(int v) {
  Node *t, *x;
  x = new Node;
  x->val = v;
  do {
    t = Top; x->next = t;
  } while(!CAS(&Top,t,x));
}
```

linearized by

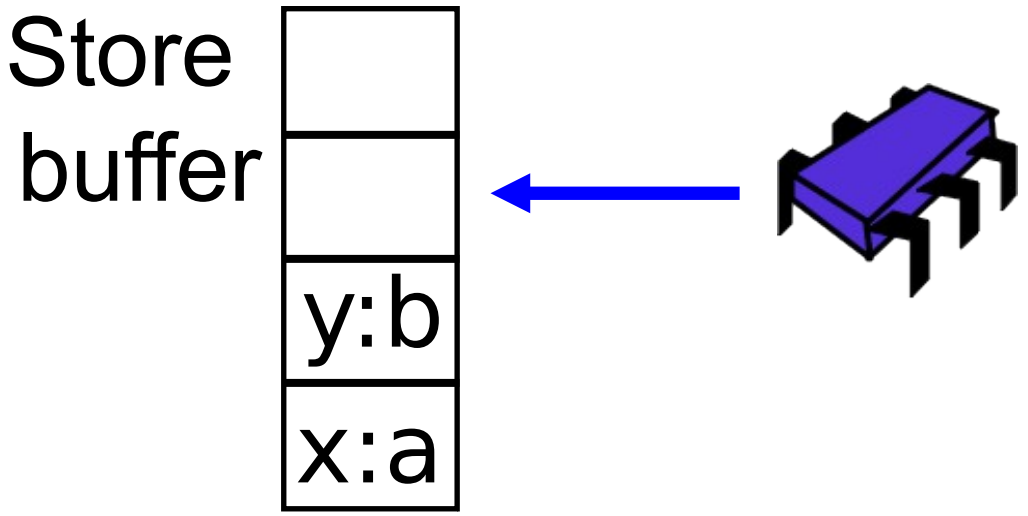
L' *(Specification)*

c

```
Sequence S;

void push(int v) {
  atomic { S = v·S; }
}
```

Challenge



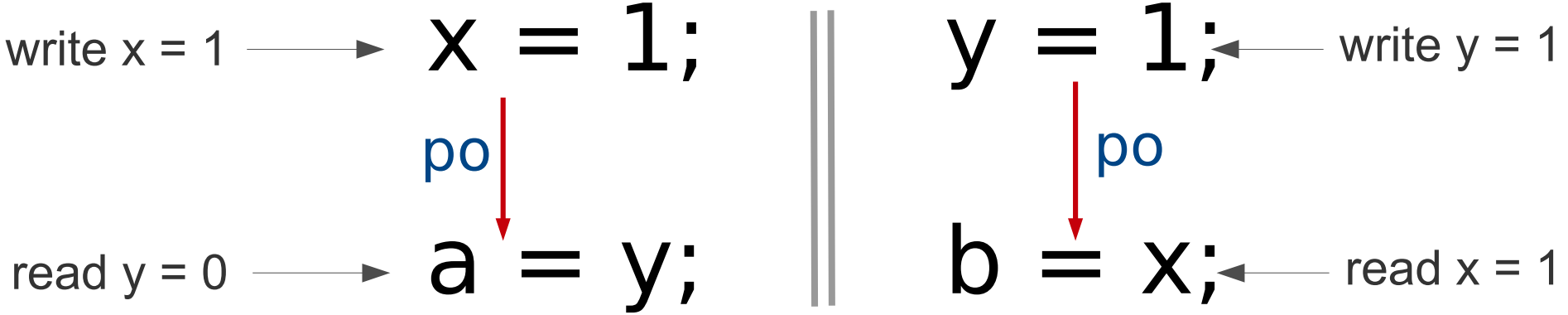

```
f() {
  *x = a;
  *y = b;
}
```

- Writes can be flushed after f returns
- Just parameters and return values not enough

Axiomatic model

An execution of ***C(L)*** is ***(A, po, rf, mo, ..., hb)***

$$x = y = a = b = 0$$

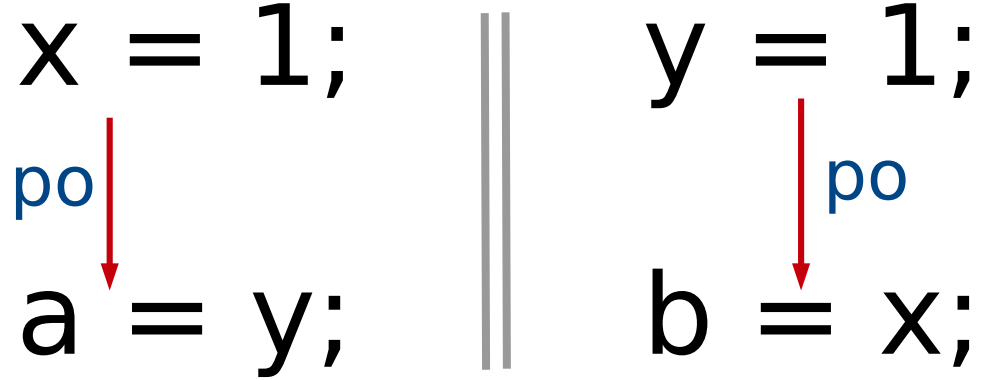


- Actions ***A***: reads, writes, calls, returns

Axiomatic model

An execution of ***C(L)*** is ***(A, po, rf, mo, ..., hb)***

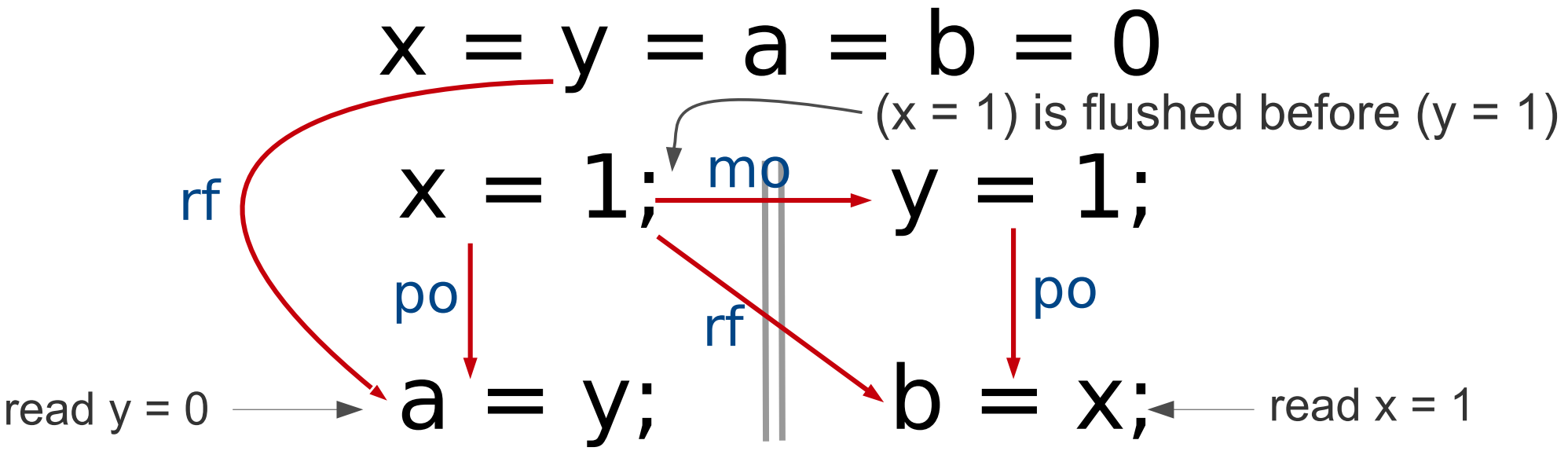
$$x = y = a = b = 0$$



- **po** – program order
  relates actions by the same thread

Axiomatic model

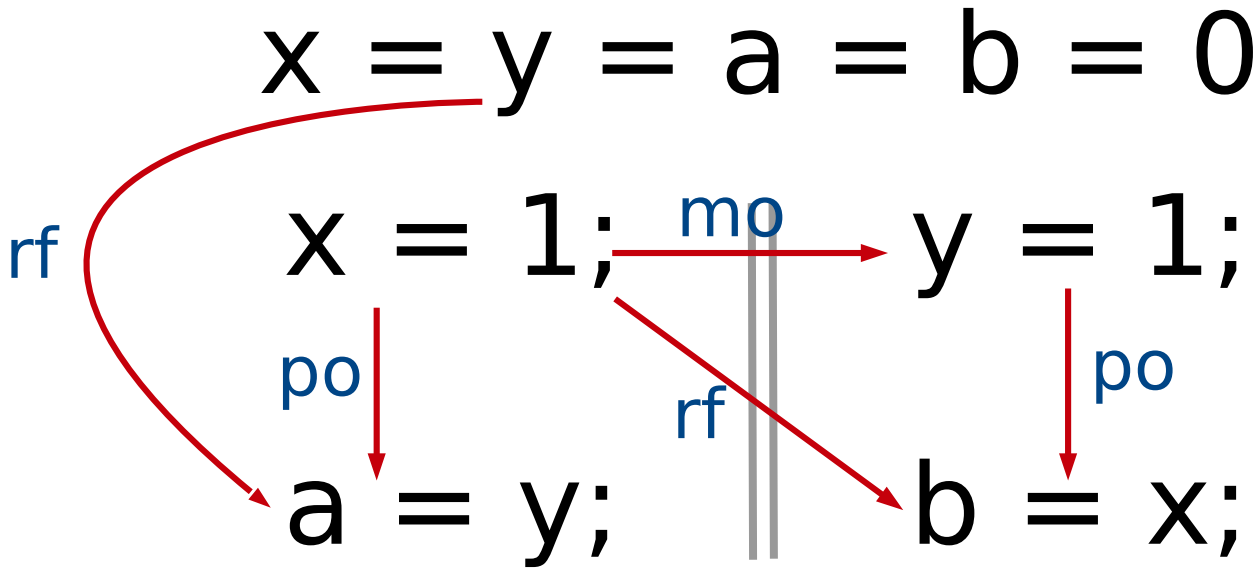
An execution of ***C(L)*** is ***(A, po, rf, mo, ..., hb)***



- **mo** – modification order
  order in which writes hit the memory
- **rf** – “reads from” relation
  relates reads to correspondent writes

Axiomatic model

An execution of ***C(L)*** is ***(A, po, rf, mo, ..., hb)***



- **hb** – “happens before” relation
  indicates precedence in the whole execution

$$\mathsf{hb} = (\mathsf{po} \cup \mathsf{rf})^{+}$$

Axioms

MOWF. mo is total, transitive, irreflexive and relates only store actions in the execution.

POWF. po is total, transitive, irreflexive and relates only actions by the same thread.

SCWF. sc is total, transitive, irreflexive and relates only fences in the execution.

RFWF. $(\forall w_1, w_2, r.\, w_1 \xrightarrow{\mathsf{rf}} r \wedge w_2 \xrightarrow{\mathsf{rf}} r \implies w_1 = w_2) \wedge$
$(\forall w, r.\, w \xrightarrow{\mathsf{rf}} r \implies \exists x, a.\, w = (\_, \mathsf{store}(x,a)) \wedge (r = (\_, \mathsf{load}(x,a))) \vee r = (\_, \mathsf{call}\ \_(a))) \vee r = (\_, \mathsf{ret}\ \_(a))))$

RFDET. $\forall x, w, r.(r = (\_, \mathsf{load}(x, \_)) \vee (\exists t.\, x = \mathsf{param}_t \wedge r = (t, \mathsf{call}\ \_)) \vee$
$(\exists t.\, x = \mathsf{retval}_t \wedge r = (t, \mathsf{ret}\ \_))) \wedge w = (\_, \mathsf{store}(x, \_)) \wedge (w \xrightarrow{\mathsf{hb}} r \vee w \xrightarrow{\mathsf{po}} r) \implies \exists w'.\, w' \xrightarrow{\mathsf{rf}} r$

HBDEF. $\mathsf{hb} = (\mathsf{po} \cup \mathsf{rf})^+$

HBVSMO. $\neg\exists w_1, w_2.$ $w_1 \underset{\mathsf{mo}}{\overset{\mathsf{hb}}{\rightleftarrows}} w_2$

RFMR. $\neg\exists w_1, w_2, r.$ $w_1 \xrightarrow{\mathsf{mo}} w_2 \xrightarrow{\mathsf{hb}} r$, with $w_1 \xrightarrow{\mathsf{rf}} r$

where w_1, w_2 and r access the same location.

RFMR'. $\neg\exists w, w_1, w_2, r.$

$w_1 \xrightarrow{\mathsf{mo}} w_2 \xrightarrow{\mathsf{mo}} w \xrightarrow{\mathsf{rf}} r' \xrightarrow{\mathsf{sb}} r$, with $w_1 \xrightarrow{\mathsf{rf}} r$

where w_1 and w_2 write to the same location,
and w and r' are by different threads.

HBWF. hb is acyclic.

HBVSSC. $\neg\exists f_1, f_2.$ $f_1 \underset{\mathsf{sc}}{\overset{\mathsf{hb}}{\rightleftarrows}} f_2$

MOVSSC. $\neg\exists w_1, w_2, f_1, f_2.$

$w_1 \xrightarrow{\mathsf{hb}} f_1 \xrightarrow{\mathsf{sc}} f_2 \xrightarrow{\mathsf{hb}} w_2$, with $w_2 \xrightarrow{\mathsf{mo}} w_1$

SCRF. $\neg\exists w, w', f_1, f_2, r.$

$w \xrightarrow{\mathsf{mo}} w' \xrightarrow{\mathsf{po}} f_1 \xrightarrow{\mathsf{sc}} f_2 \xrightarrow{\mathsf{po}} r$, with $w \xrightarrow{\mathsf{rf}} r$

Axioms

- Filter out impossible executions
- Capture the effect of store buffering

$$\text{HBvsMO}.\ \neg\exists w_1, w_2.\quad w_1 \underset{\text{mo}}{\overset{\text{hb}}{\rightleftarrows}} w_2$$

Disallows “writing to the future”

Abstraction Theorem

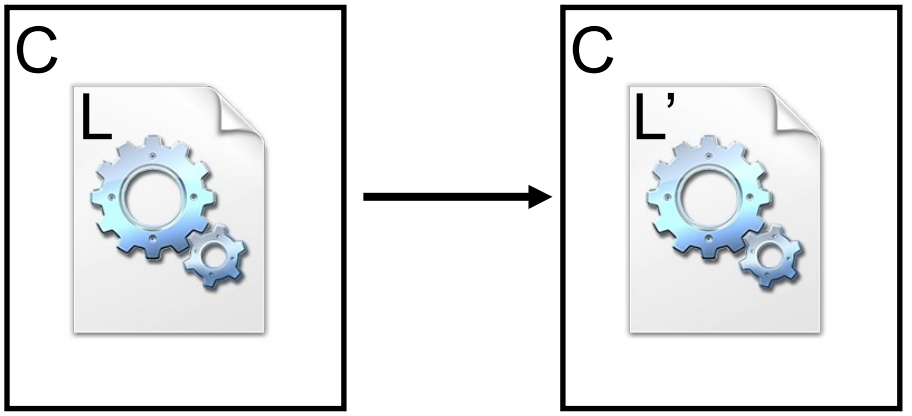
Assume that L $\sqsubseteq$ L'.

Then $\mathsf{client}([\![C(L)]\!]) \subseteq \mathsf{client}([\![C(L')]\!])$

L' specifies *L*:

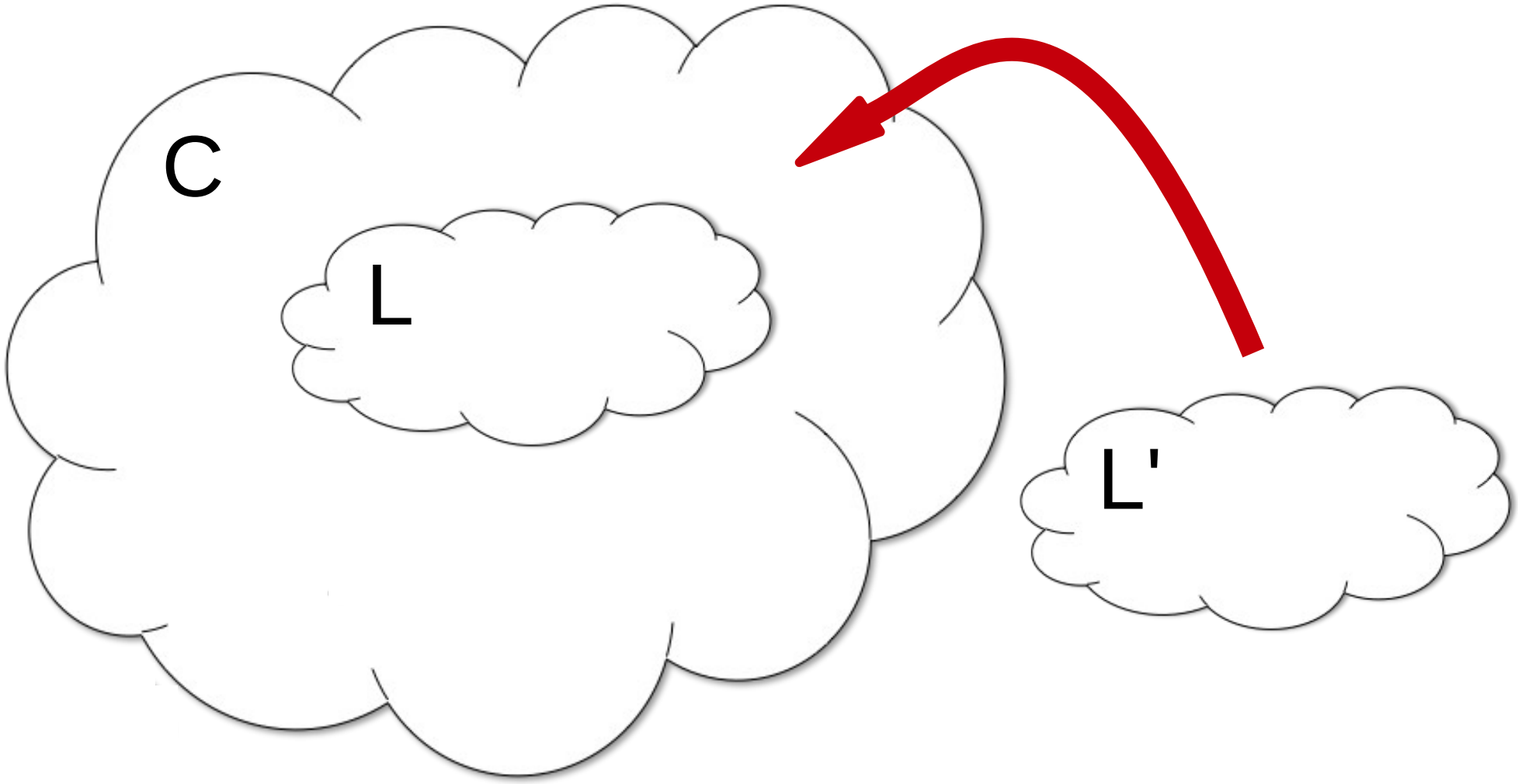
L $\sqsubseteq$ L'

$\Rightarrow$

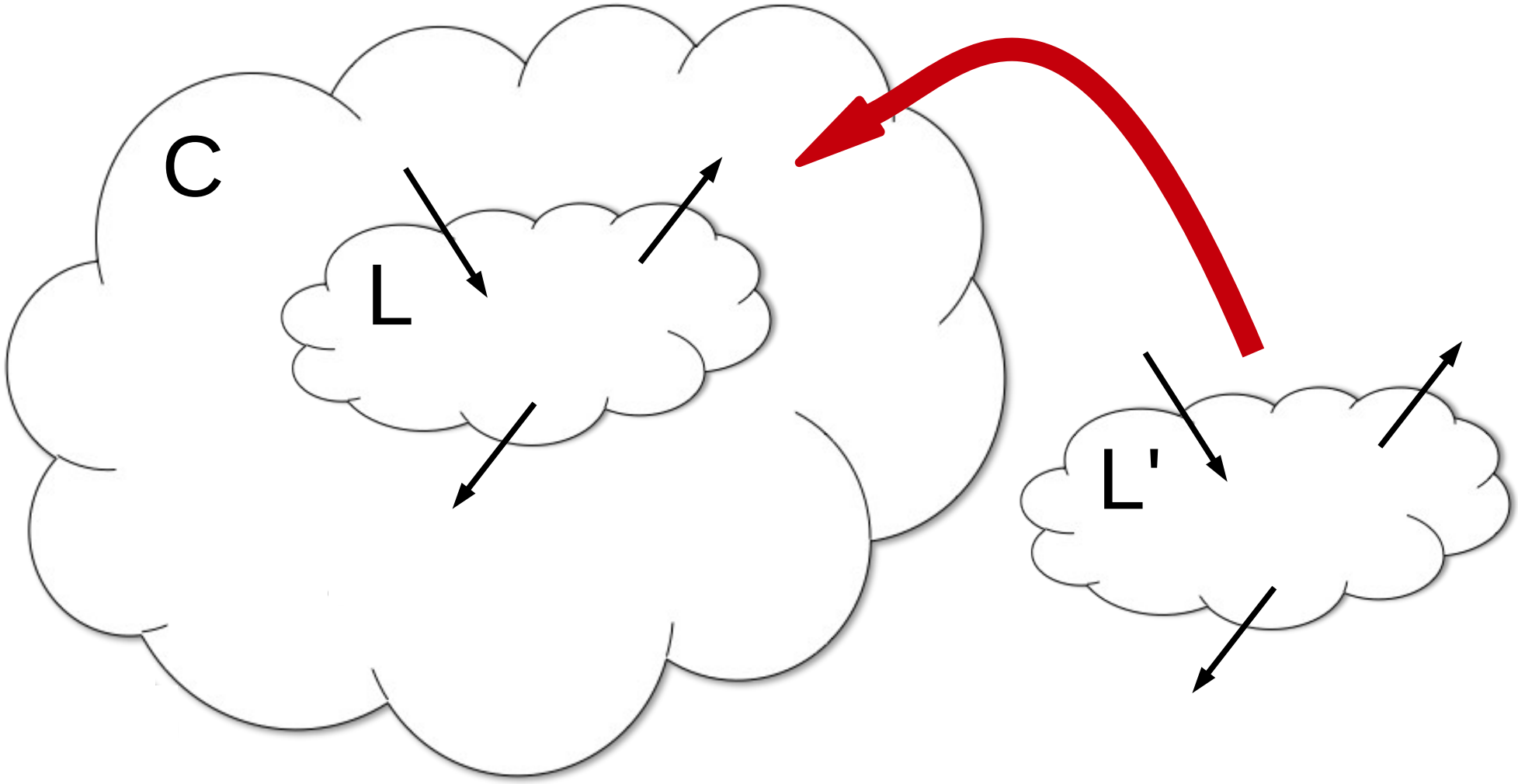


$$C(L') \models P \Rightarrow C(L) \models P$$

Subgraph replacement

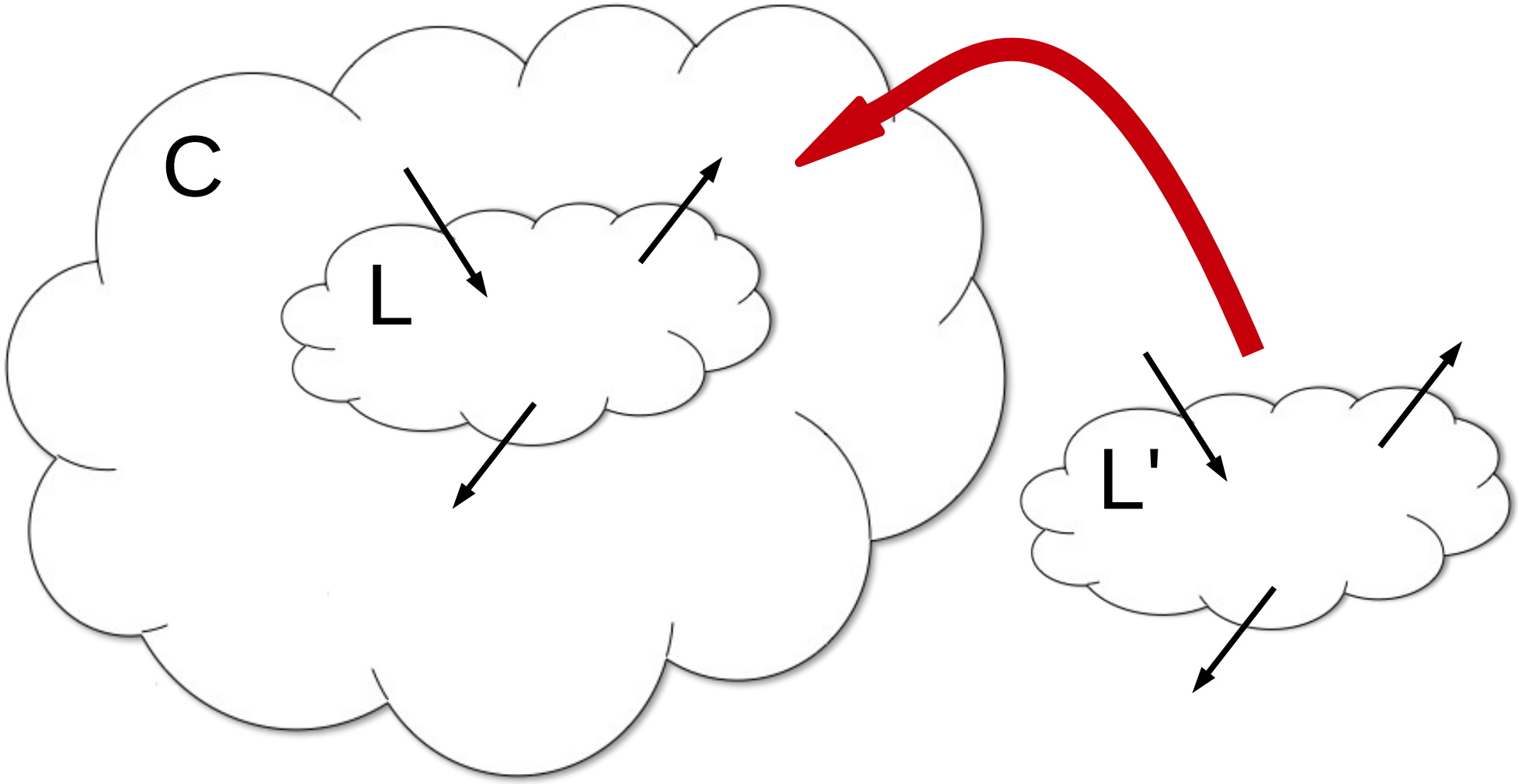


Subgraph replacement



History - several relations on boundary actions

Subgraph replacement



$$L \sqsubseteq L' \iff \forall H \in [\![L]\!] \,.\, \exists H' \in [\![L']\!] \,.\, H \sqsubseteq H'$$

Computing histories

- Take a library L on TSO or SC
- Take the most general client:

$$\Big\|_{k=1}^{n} \left(\begin{array}{ll} \texttt{while (true) \{} & \\ \quad \texttt{if (nondet())} & m_1\texttt{(nondet());} \\ \quad \texttt{else if (nondet())} & m_2\texttt{(nondet());} \\ \quad \texttt{...} & \\ \quad \texttt{else} & m_l\texttt{(nondet());} \\ \texttt{\}} & \end{array} \right)$$

- Get all possible library executions: $[\![L]\!]_{\mathsf{TSO}}$

$$L \sqsubseteq L' \iff \forall H \in [\![L]\!] \,.\, \exists H' \in [\![L']\!] \,.\, H \sqsubseteq H'$$

Computing histories

- Take a library L on TSO or SC
- Take the most general client:

Any methods, in any order, with any parameters

Any number of threads

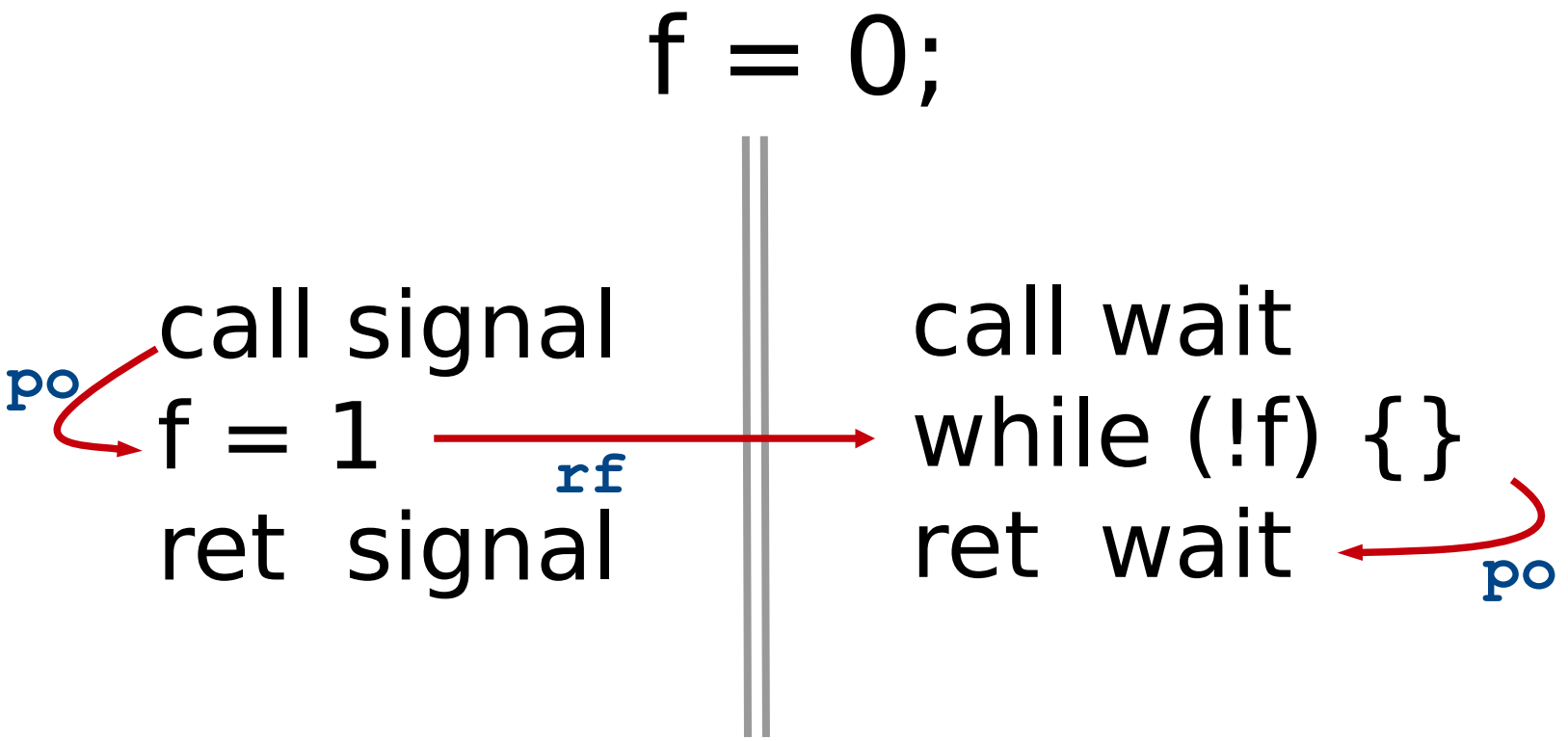
$$\Big\|_{k=1}^{n} \left(\begin{array}{l} \texttt{while (true) \{} \\ \quad \texttt{if (nondet())} \quad\quad\; m_1\texttt{(nondet());} \\ \quad \texttt{else if (nondet())} \; m_2\texttt{(nondet());} \\ \quad \texttt{...} \\ \quad \texttt{else} \quad\quad\quad\quad\; m_l\texttt{(nondet());} \\ \texttt{\}} \end{array} \right)$$

- Get all possible library executions: $[\![L]\!]_{\mathsf{TSO}}$

$$L \sqsubseteq L' \iff \forall H \in [\![L]\!] \,.\, \exists H' \in [\![L']\!] \,.\, H \sqsubseteq H'$$

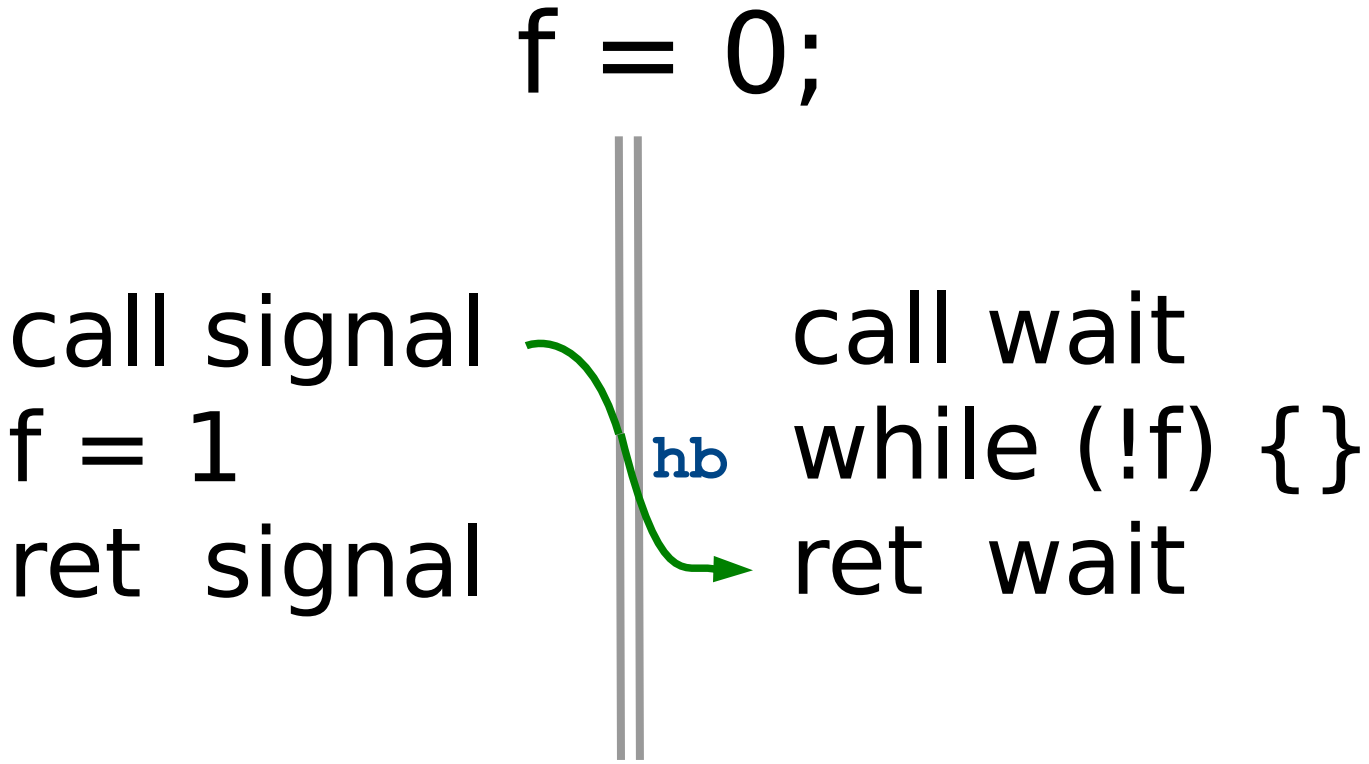
Guarantee relation

- Histories include the projection of happens-before to calls and returns:



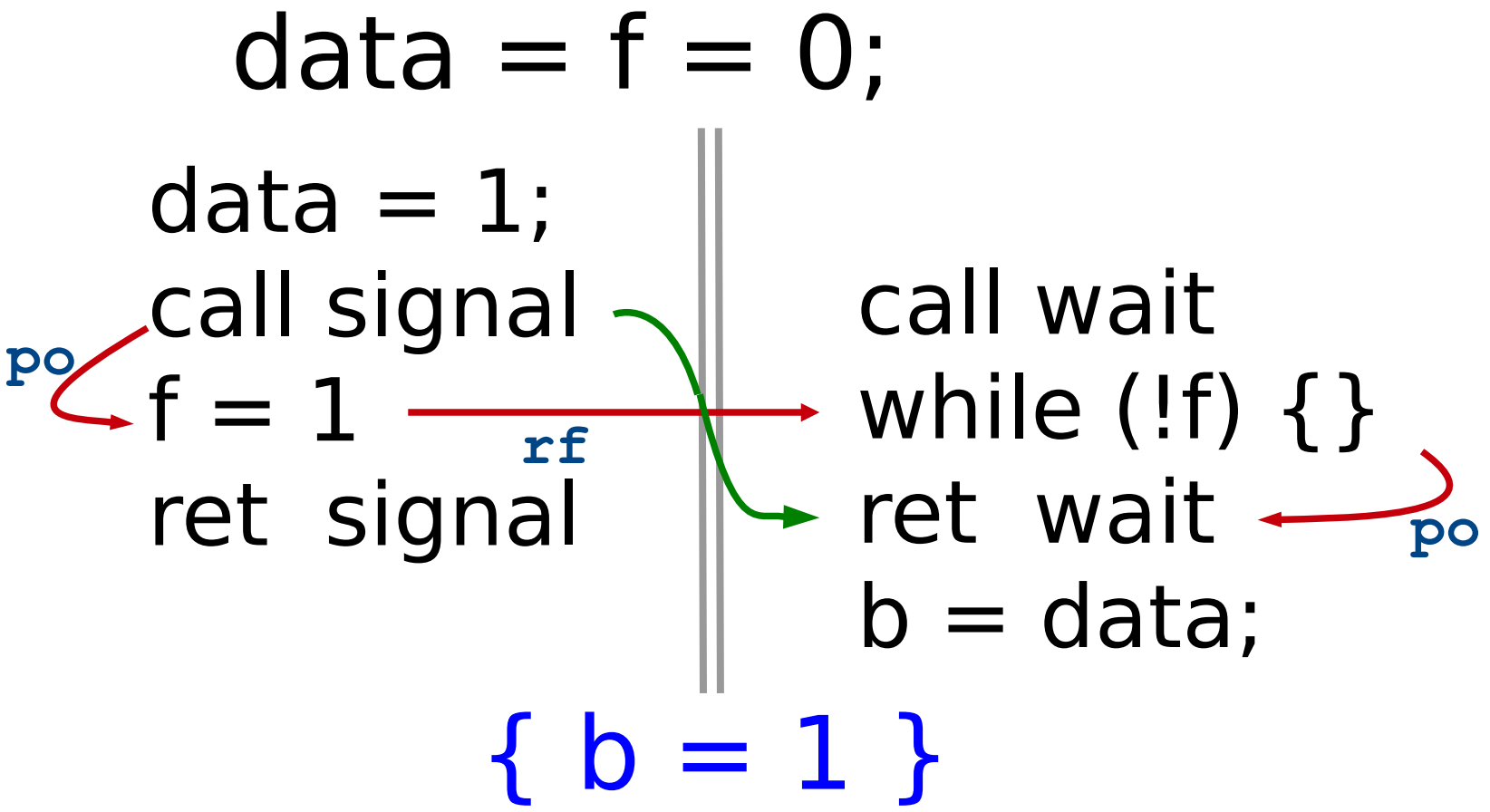
Guarantee relation

- Histories include the projection of happens-before to calls and returns:



Guarantee relation

- Histories include the projection of happens-before to calls and returns:



Conclusions

- Through thorns to the stars – compositional reasoning is possible on TSO.
- Axiomatic semantics can be straightforwardly implemented in SAT solvers
- Can be applied to more complicated models (IBM Power, ARM)

Related work

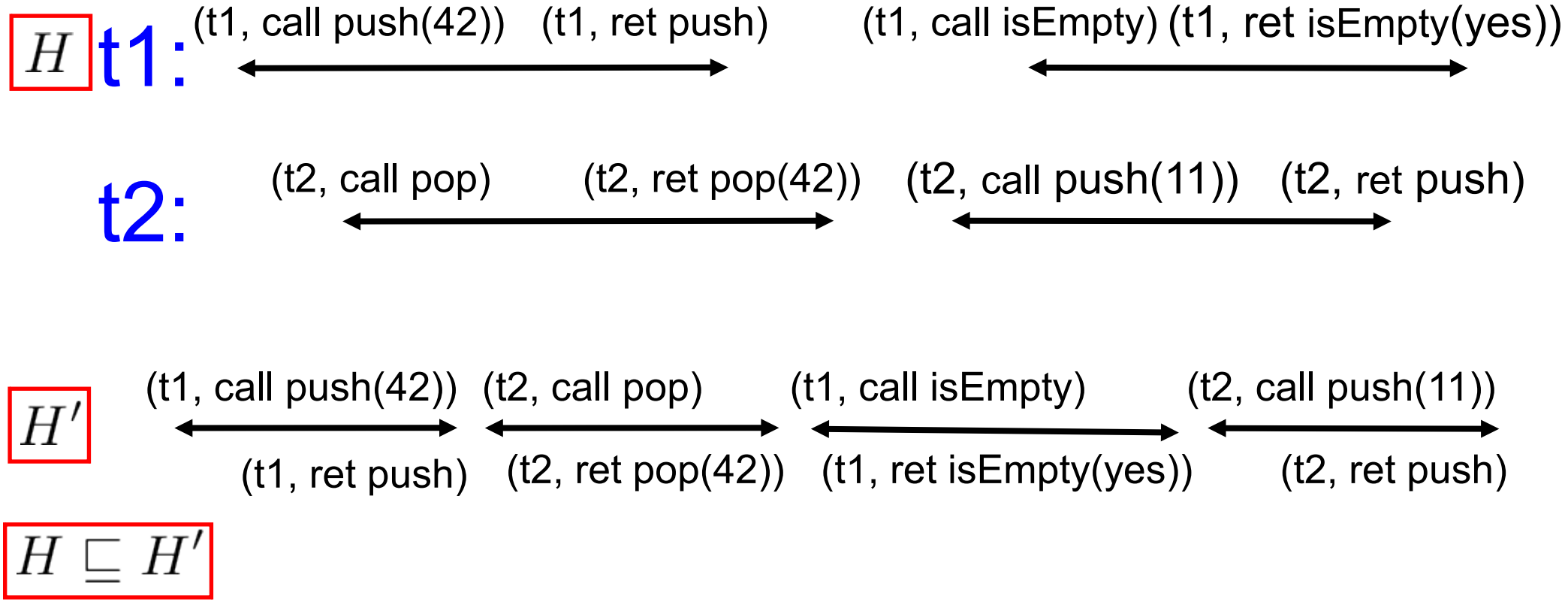
- **Concurrent library correctness on the TSO memory model**
- *S. Burckhardt et al.* (ESOP'12)

- **Library abstraction for C/C++ concurrency**
- *M. Batty et al. (POPL'13)*

Linearizability

- History: (Interface, Guarantee, Deny)
- $(I_1, G_1, D_1) \sqsubseteq (I_2, G_2, D_2) \iff$
  $(I_1 = I_2) \wedge (G_2 \subseteq G_1) \wedge (D_2 \subseteq D_1)$
- Specification can guarantee and deny less to the client
- So client has more behaviours when using specification instead of implementation
- $L_1 \sqsubseteq L_2 \iff \forall H \in [\![L]\!] \,.\, \exists H' \in [\![L']\!] \,.\, H \sqsubseteq H'$

Linearizability



- We can permute calls and returns by different threads
- But non-overlapping method invocations can't be rearranged

Linearizability

- How it fails:

$$x = y = 0;$$

$$\begin{array}{l|l} x = 1; & y = 1; \\ m(); & m(); \\ a = y; & b = x; \end{array}$$

$$\{\ a = b = 0\ \}$$

Linearizability

- How it fails:

$$x = y = 0;$$

$$\begin{array}{l|l} x = 1; & y = 1; \\ m(); & m(); \\ a = y; & b = x; \end{array}$$

$$\{\ a = b = 0\ \}$$

- Not possible, if ***m*** contains a fence.