Reasoning under Release-Acquire Consistency

Viktor Vafeiadis

Max Planck Institute for Software Systems (MPI-SWS)

GandALF'15, Genova, 2015-09-22

The plan for the talk

Talk outline

- Weak memory models
- Release-acquire consistency
- Owicki-Gries is unsound for weak memory
- OGRA: Adapted Owicki-Gries for release-acquire
- Fences as a way of restoring SC

Sequential consistency

Sequential consistency (SC):

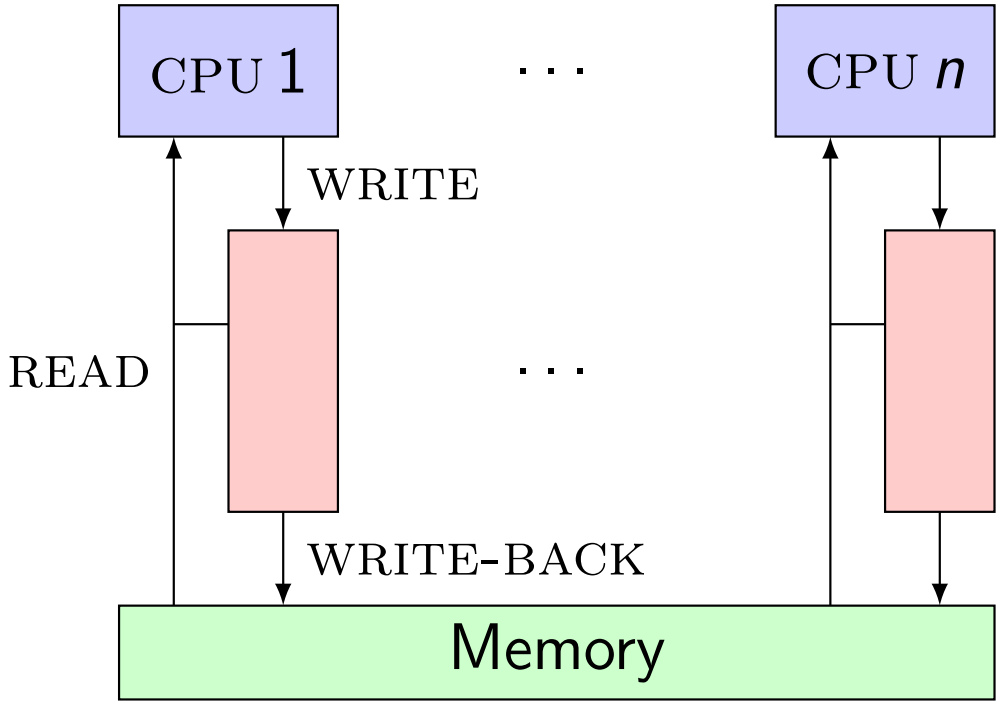
- The standard model for concurrency.
- Interleave each thread's atomic accesses.
- Almost all verification work assumes it.

Initially, $x = y = 0$.

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

In SC, this program cannot return $a = b = 0$.

Store buffering in x86-TSO



Initially, $x = y = 0$.

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

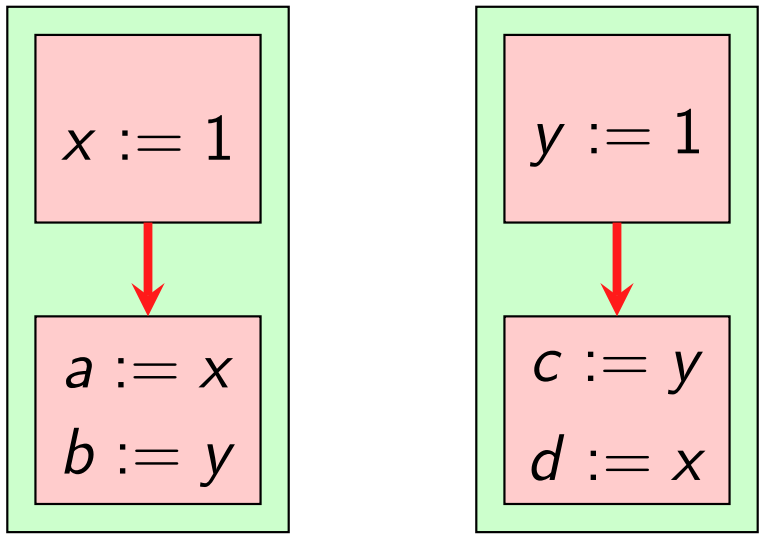
Allowed outcome: $a = b = 0$.

IRIW: Not just store buffering

Initially, $x = y = 0$.

$$x := 1 \;\Big\|\; y := 1 \;\Big\|\; \begin{array}{l} a := x; \\ b := y \end{array} \;\Big\|\; \begin{array}{l} c := y; \\ d := x \end{array}$$

Allowed outcome: $a = c = 1$ and $b = d = 0$.



A basic guarantee: Coherence

Coherence:

"SC for a single variable"

Initially, $x = 0$.

$$x := 1 \;\Big\|\; x := 2 \;\Big\|\; \begin{array}{l} a := x; \\ b := x \end{array} \;\Big\|\; \begin{array}{l} c := x; \\ d := x \end{array}$$

Forbidden outcome: $a = 1, b = 2, c = 2, d = 1$.

Release-acquire synchronization: Message passing

Initially, $x = y = 0$.

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

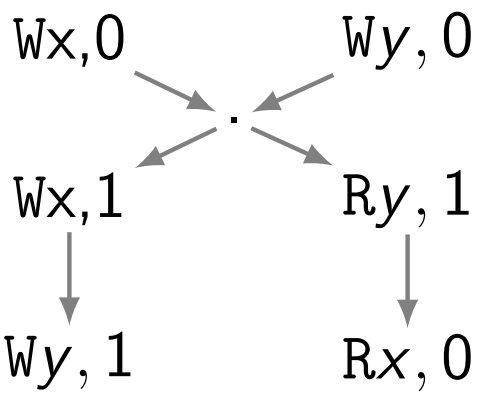
Disallowed outcome: $a = 1$ and $b = 0$.

Release-acquire synchronization: Message passing

Initially, $x = y = 0$.

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

Disallowed outcome: $a = 1$ and $b = 0$.



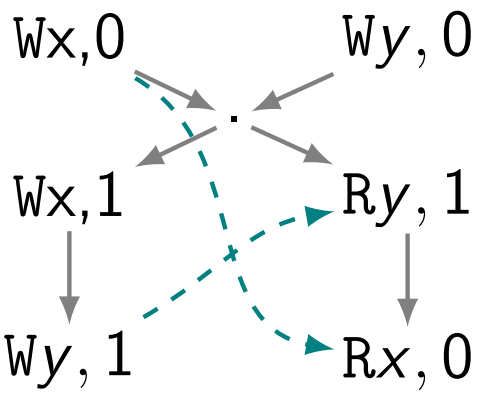
$\rightarrow$: program order, po

Release-acquire synchronization: Message passing

Initially, $x = y = 0$.

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

Disallowed outcome: $a = 1$ and $b = 0$.



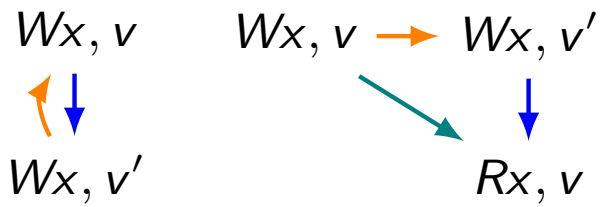
$\rightarrow$: program order, po
$\rightarrow$: reads-from, rf

Release-acquire memory model

- Each program $\rightsquigarrow$ a set of graphs (called: *executions*).
- An execution is *consistent* if it can be augmented with relations:
  - *reads-from*: associates each read with a corresponding write
  - *memory-order*: total order on all writes to the same location

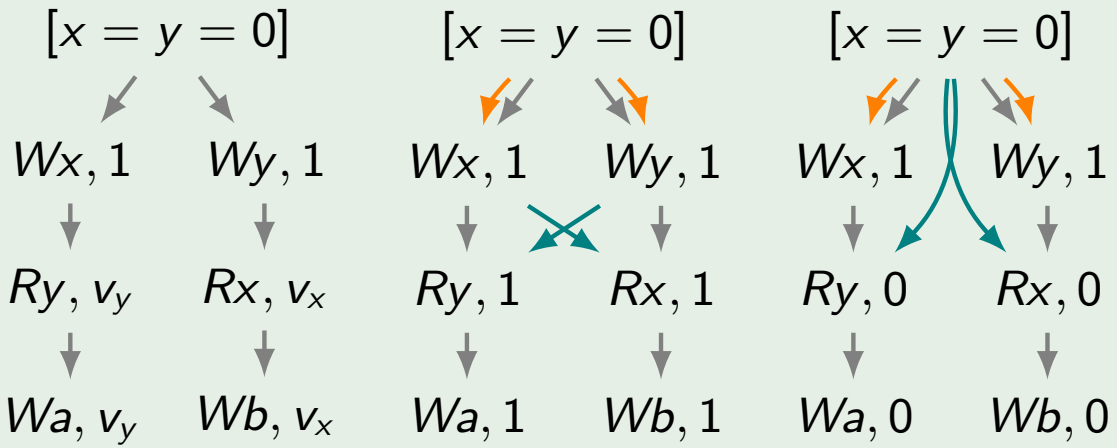
  s.t. $\mathit{happens\text{-}before} = (\mathit{prog\text{-}order} \cup \mathit{reads\text{-}from})^+$ is acyclic

  and none of the following occur:



Example (Store Buffering)

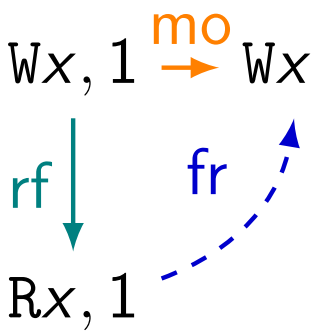
$$\begin{array}{c} x = y = 0 \\ x := 1 \;\|\; y := 1 \\ a := y \;\|\; b := x \end{array}$$



Alternative RA definition

Reads-before relation:

- a.k.a. "from-reads" or "conflict"



$$fr \triangleq rf^{-1}; mo$$

Definition (RA consistency)

An execution is RA consistent iff for all locations x, the relation $po \cup rf \cup mo_x \cup fr_x$ is acyclic.

Owicki-Gries and Weak Memory

- Standard OG is unsound.
- Restoring soundness under RA.

Owicki-Gries method (1976)

OG = Hoare logic + rule for parallel composition

$$\frac{\{P_1\}\, c_1\, \{Q_1\} \qquad \{P_2\}\, c_2\, \{Q_2\} \quad \text{the two proofs are } \textit{non-interfering}}{\{P_1 \wedge P_2\}\, c_1 \parallel c_2\, \{Q_1 \wedge Q_2\}}$$

Non-interference

$R \wedge P \vdash R\{u/x\}$ for every:

- assertion R in the proof outline of one thread
- assignment $x := u$ with precondition P in the proof outline of the other thread

Standard OG is unsound for WM

$$\{a \neq 0\}$$

$$x := 1; \quad \| \quad y := 1;$$

$$a := y \quad \| \quad b := x$$

$$\{a \neq 0 \vee b \neq 0\}$$

Standard OG is unsound for WM

$$
\begin{array}{c}
\{a \neq 0\} \\
\begin{array}{l|l}
\{a \neq 0\} & \{\top\} \\
x := 1; & y := 1; \\
\{x \neq 0\} & \{y \neq 0\} \\
a := y & b := x \\
\{x \neq 0\} & \{y \neq 0 \wedge (a \neq 0 \vee b = x)\}
\end{array} \\
\{a \neq 0 \vee b \neq 0\}
\end{array}
$$

Stronger non-interference condition

$$\frac{\{P_1\}\, c_1\, \{Q_1\} \qquad \{P_2\}\, c_2\, \{Q_2\} \quad \text{the two proofs are non-interfering}}{\{P_1 \wedge P_2\}\, c_1 \parallel c_2\, \{Q_1 \wedge Q_2\}}$$

Non-interference under RA

$R \wedge P \vdash R\{v/x\}$ for every:

- assertion R in the proof outline of one thread
- assignment $x := u$ with precondition P in the proof outline of the other thread
- value v such that $P \wedge R' \wedge u = v$ is satisfiable for some assertion R' above R

Example: message passing

$$\{x = 0\}$$

$$m := 42 \quad \| \quad \texttt{while } x = 0 \texttt{ do skip}$$

$$x := 1 \quad \| \quad a := m$$

$$\{a = 42\}$$

Example: message passing

$$
\begin{array}{c}
\{x = 0\} \\
\begin{array}{l|l}
\{\top\} & \{x \neq 0 \rightarrow m = 42\} \\
m := 42 & \texttt{while } x = 0 \texttt{ do skip} \\
\{m = 42\} & \{m = 42\} \\
x := 1 & a := m \\
\{\top\} & \{a = 42\}
\end{array} \\
\{a = 42\}
\end{array}
$$

Example: read-read coherence (CoRR2)

$$\{x = a = b = c = d = 0\}$$

$$x := 1 \quad \| \quad x := 2 \quad \| \quad \begin{array}{l} a := x \\ b := x \end{array} \quad \| \quad \begin{array}{l} c := x \\ d := x \end{array}$$

$$\{a \neq 1 \vee b \neq 2 \vee c \neq 2 \vee d \neq 1\}$$

Example: read-read coherence (CoRR2)

$$\left\{x = a = b = c = d = 0\right\}$$

$$\begin{array}{c|c|c|c}
\left\{\begin{array}{l} x \neq 1 \wedge \\ a \neq 1 \end{array}\right\} & \left\{\begin{array}{l} x \neq 2 \wedge \\ c \neq 2 \end{array}\right\} & \{\top\} & \{\top\} \\
x := 1 & x := 2 & a := x & c := x \\
\{\top\} & \{\top\} & \{\top\} & \{\top\} \\
 & & b := x & d := x \\
 & & \left\{\begin{array}{l} a \neq 1 \vee \\ b \neq 2 \vee \\ x = 2 \end{array}\right\} & \left\{\begin{array}{l} c \neq 2 \vee \\ d \neq 1 \vee \\ x = 1 \end{array}\right\}
\end{array}$$

$$\left\{a \neq 1 \vee b \neq 2 \vee c \neq 2 \vee d \neq 1\right\}$$

Example: read-read coherence (CoRR2)

$$\Big\{x = a = b = c = d = 0\Big\}$$

$$\begin{array}{c|c|c|c}
 & & \{\top\} & \{\top\} \\
\left\{\begin{array}{l} x \neq 1 \wedge \\ a \neq 1 \end{array}\right\} & \left\{\begin{array}{l} x \neq 2 \wedge \\ c \neq 2 \end{array}\right\} & a := x & c := x \\
x := 1 & x := 2 & \{\top\} & \{\top\} \\
\{\top\} & \{\top\} & b := x & d := x \\
 & & \left\{\begin{array}{l} a \neq 1 \vee \\ b \neq 2 \vee \\ x = 2 \end{array}\right\} & \left\{\begin{array}{l} c \neq 2 \vee \\ d \neq 1 \vee \\ x = 1 \end{array}\right\}
\end{array}$$

$$\Big\{a \neq 1 \vee b \neq 2 \vee c \neq 2 \vee d \neq 1\Big\}$$

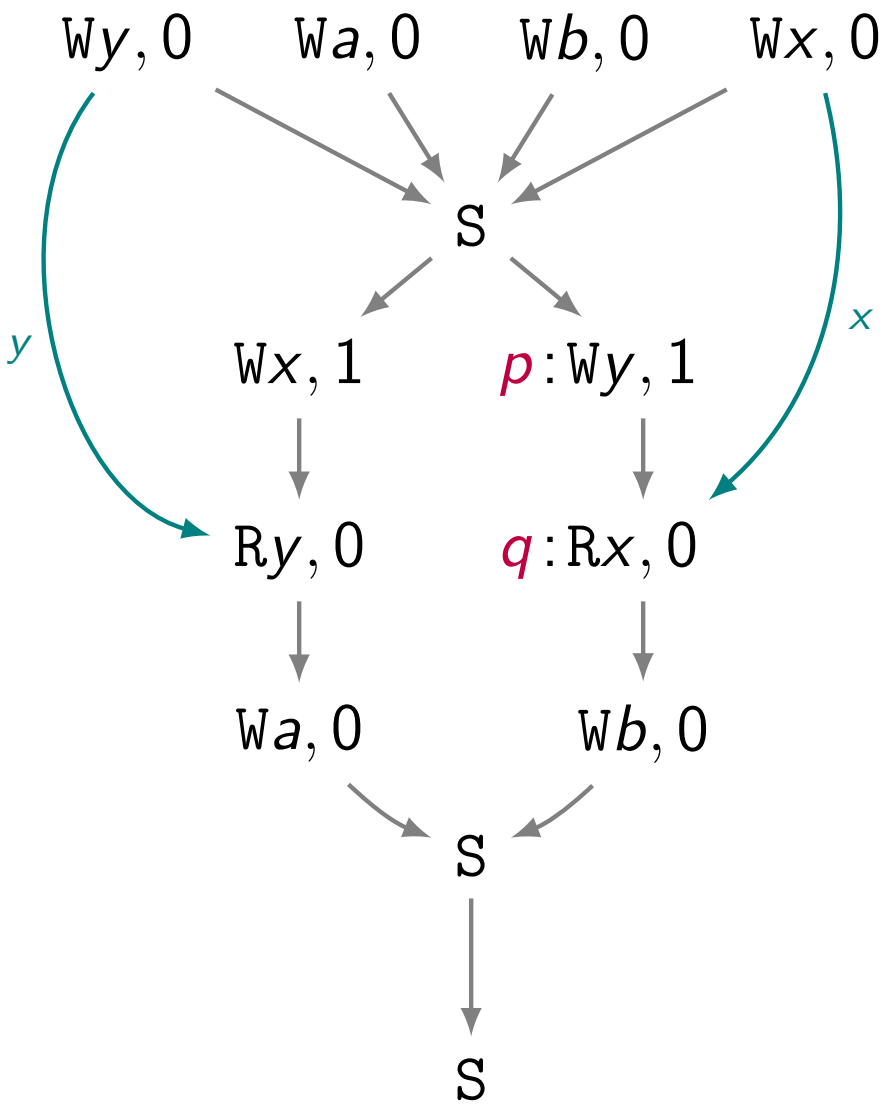
Soundness

Challenges in a weak memory setting:

- No intuitive operational semantics
- No notion of global state

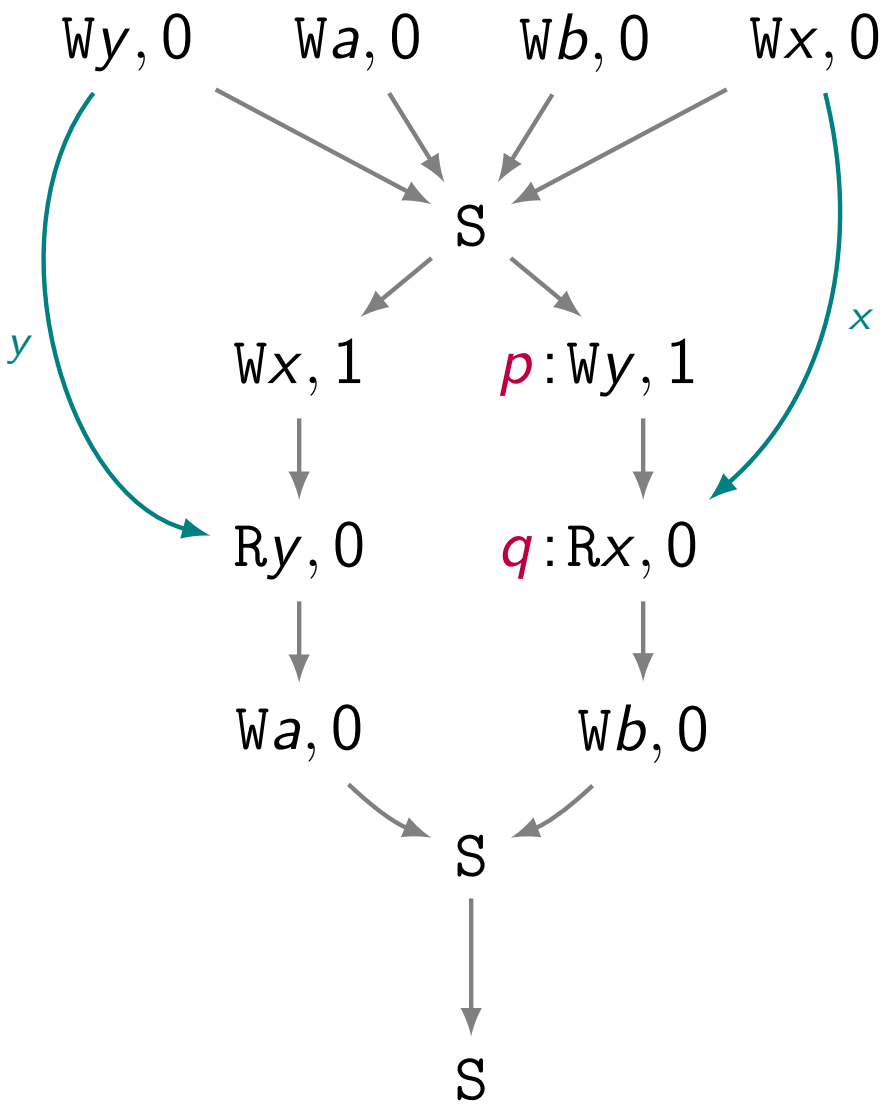
What does soundness exactly mean?

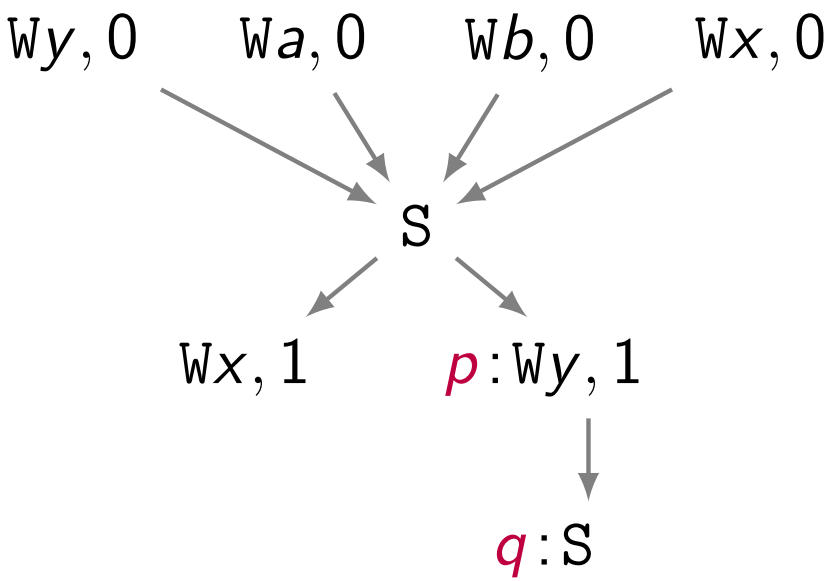
Visible states



$\sigma = \{x \mapsto 1, y \mapsto 1, a \mapsto 0, b \mapsto 0\}$ is visible at $\langle p, q \rangle$

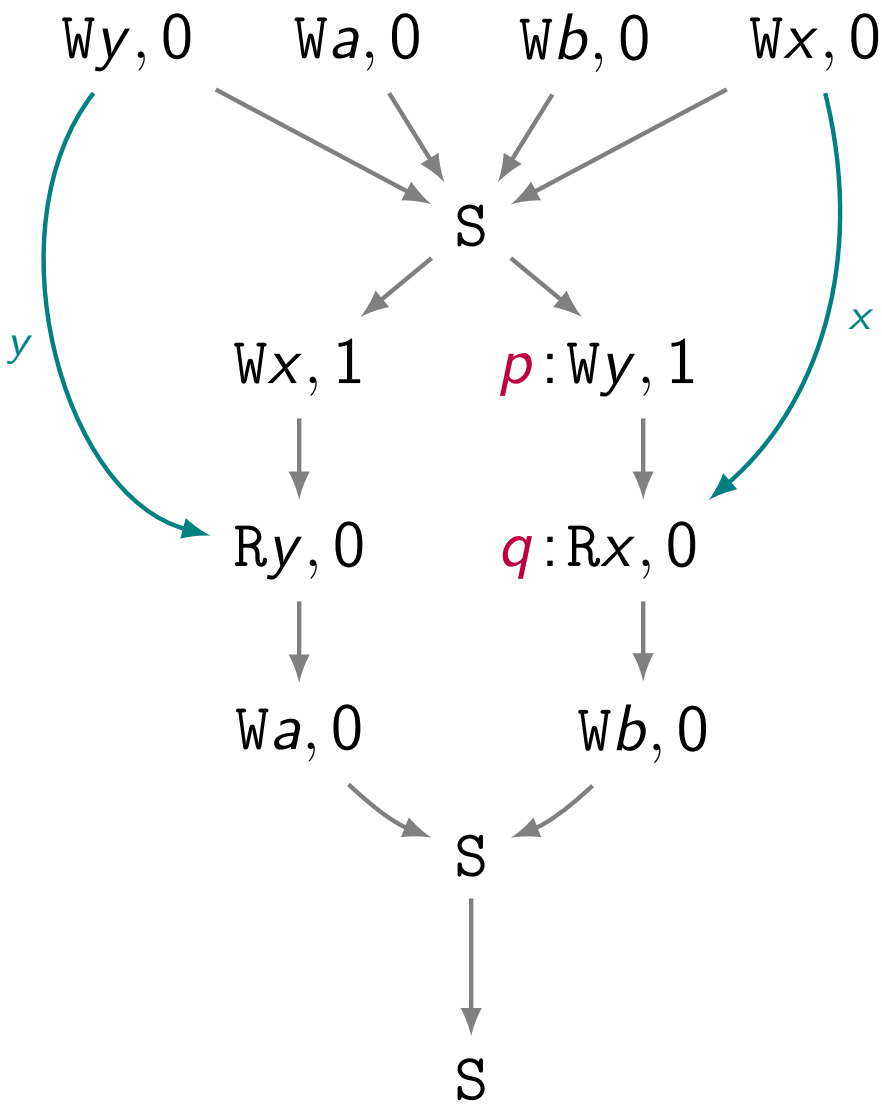
Visible states

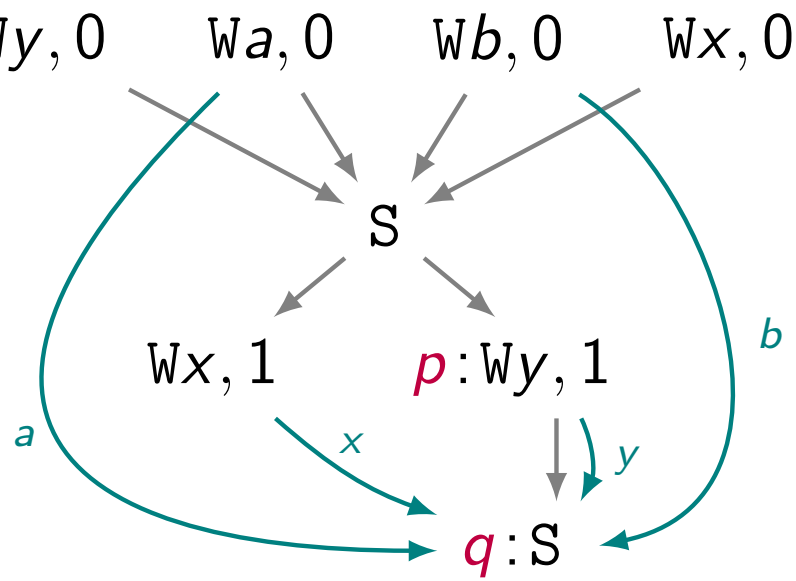




$\sigma = \{x \mapsto 1, y \mapsto 1, a \mapsto 0, b \mapsto 0\}$ is visible at $\langle p, q \rangle$

Visible states





$\sigma = \{x \mapsto 1, y \mapsto 1, a \mapsto 0, b \mapsto 0\}$ is visible at $\langle p, q \rangle$

Meaning of Hoare triples

Triple validity

$\{P\}\, c\, \{Q\}$ is *valid* if every state visible at the terminal edge of some coherent reads-from extension of some execution in $\mathcal{WG}(P); [\![c]\!]; \mathcal{SG}$ satisfies Q.

Main steps in soundness proof:

- Study properties of visibility under the RA model.
- Show that edges of consistent executions can be annotated with the assertions from the OG derivation such that every state visible at an edge satisfies its annotation.

Program logics for WM

Program logics for C11 release-acquire:

- Relaxed separation logic (V. & Narayan, OOPSLA'13)
- GPS (Turon et al., OOPSLA'14)
- OGRA (Lahav & V., ICALP'15)

Program logics for TSO:

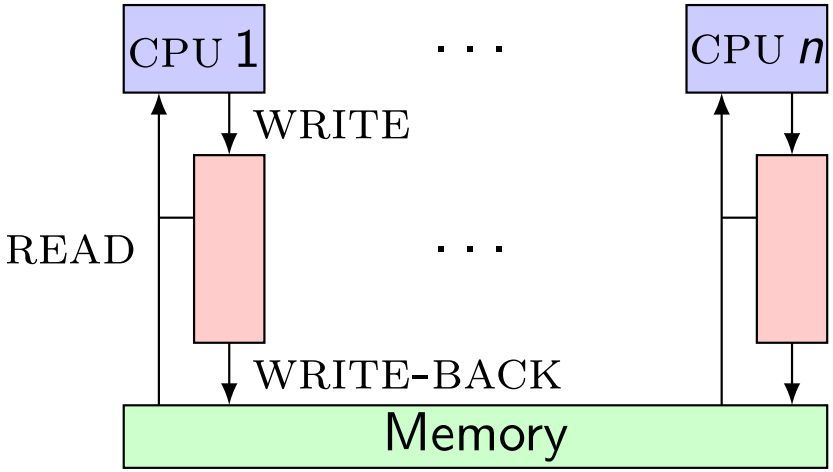
- Rely/guarantee for TSO (Ridge, VSTTE'10)
- Verifying TSO programs (Jacobs, 2014)
- iCAP-TSO (Sieczkowski et al., ESOP'15)

Reduction to SC

- A simple reduction theorem
- A more advanced reduction

Reduction to SC (robustness)

For TSO, it suffices to have a fence between every racy write & subsequent racy read.



For RA, we need more fences. Recall the IRIW example:

Initially, $X = Y = 0$.

$$X := 1 \;\Big\|\; Y := 1 \;\Big\|\; \begin{array}{l} a := X; \\ b := Y \end{array} \;\Big\|\; \begin{array}{l} c := Y; \\ d := X \end{array}$$

Allowed outcome: $a = c = 1$ and $b = d = 0$.

Simple reduction theorem

Theorem

Let $G = \langle A, po, rf \rangle$ *be a well-formed RA-coherent execution. If*

- *For every G-racy events* a, b*, if* $\langle a, b \rangle \in (po \cup rf)^+$*, then* $\langle a, c \rangle, \langle c, b \rangle \in (po \cup rf)^+$ *for some fence event* c*.*

Then, G is SC-coherent.

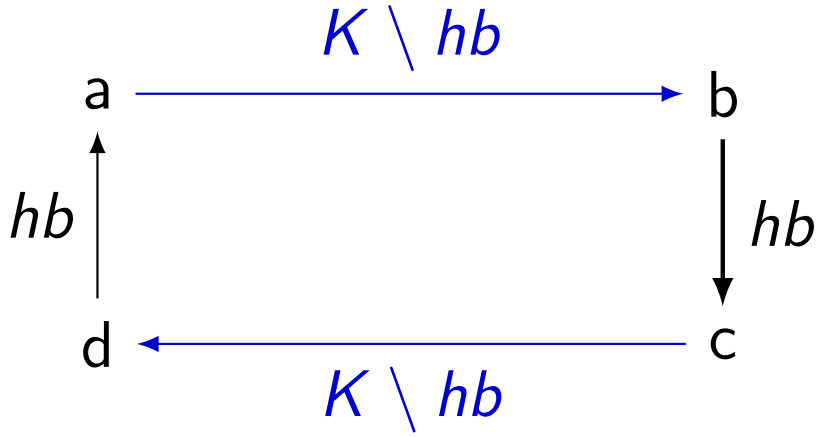
Proof of the simple reduction theorem (1/2)

Recall:

- RA coherent : $po \cup rf \cup mo_x \cup fr_x$ is acyclic for all x.
- SC coherent : $po \cup rf \cup \bigcup_x (mo_x \cup fr_x)$ is acyclic.
- Let $hb = (po \cup rf)^+$ and $K = \bigcup_x (mo_x \cup fr_x)$.

Consider minimal cycle in $(hb \cup K)$.

- Cycles with ≤ 1 K-edges disallowed by RA coherence.
- Cycle with two K-edges:



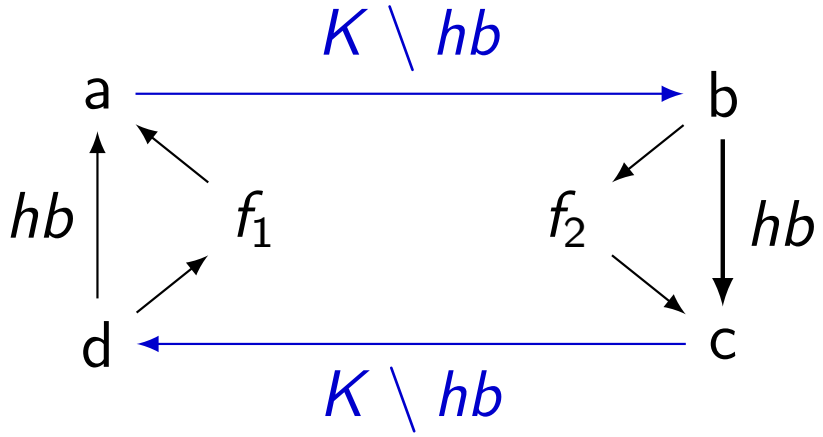
Proof of the simple reduction theorem (1/2)

Recall:

- RA coherent : $po \cup rf \cup mo_x \cup fr_x$ is acyclic for all x.
- SC coherent : $po \cup rf \cup \bigcup_x (mo_x \cup fr_x)$ is acyclic.
- Let $hb = (po \cup rf)^+$ and $K = \bigcup_x (mo_x \cup fr_x)$.

Consider minimal cycle in $(hb \cup K)$.

- Cycles with ≤ 1 K-edges disallowed by RA coherence.
- Cycle with two K-edges:



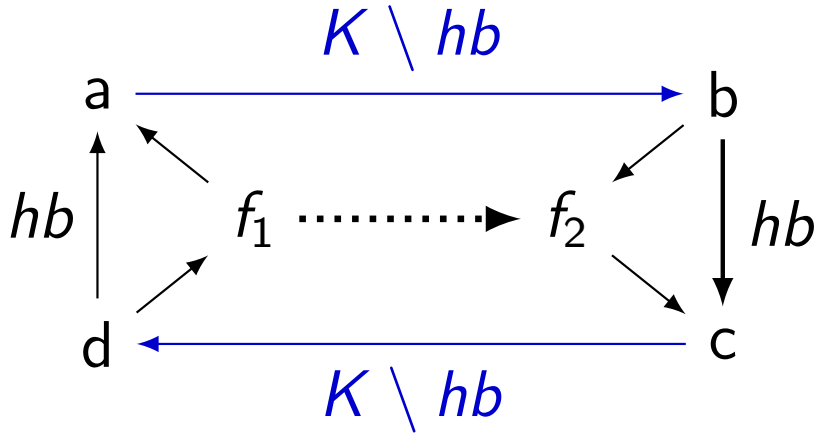
Proof of the simple reduction theorem (1/2)

Recall:

- RA coherent : $po \cup rf \cup mo_x \cup fr_x$ is acyclic for all x.
- SC coherent : $po \cup rf \cup \bigcup_x (mo_x \cup fr_x)$ is acyclic.
- Let $hb = (po \cup rf)^+$ and $K = \bigcup_x (mo_x \cup fr_x)$.

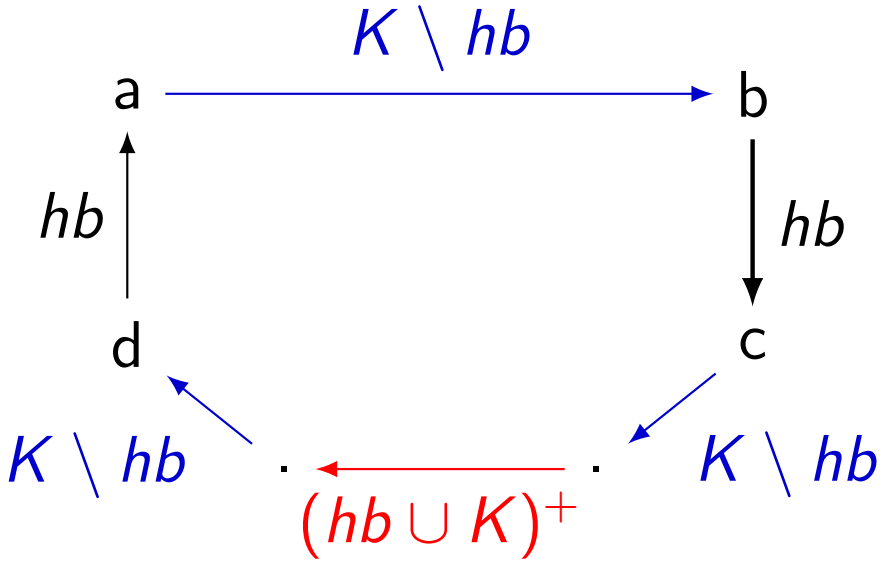
Consider minimal cycle in $(hb \cup K)$.

- Cycles with ≤ 1 K-edges disallowed by RA coherence.
- Cycle with two K-edges:



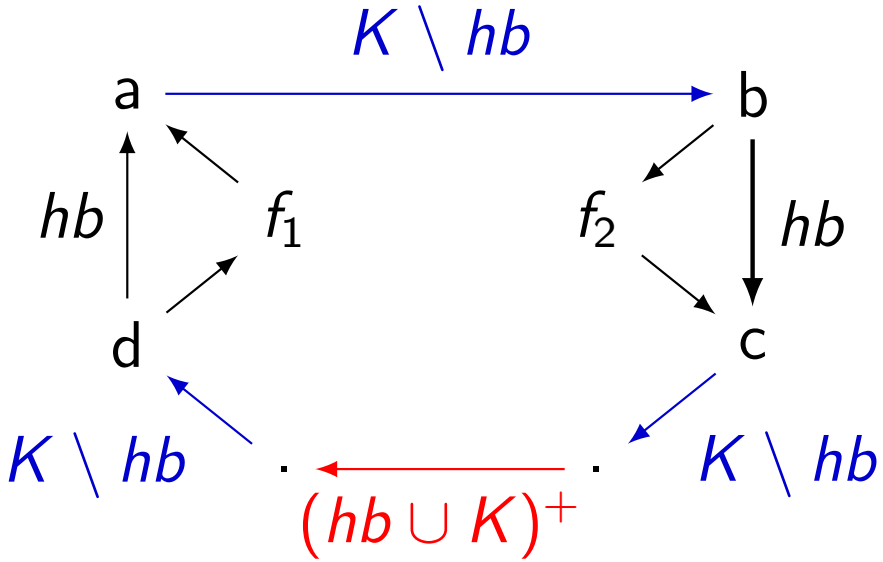
Proof of the simple reduction theorem (2/2)

Finally, consider a cycle with three or more K-edges.



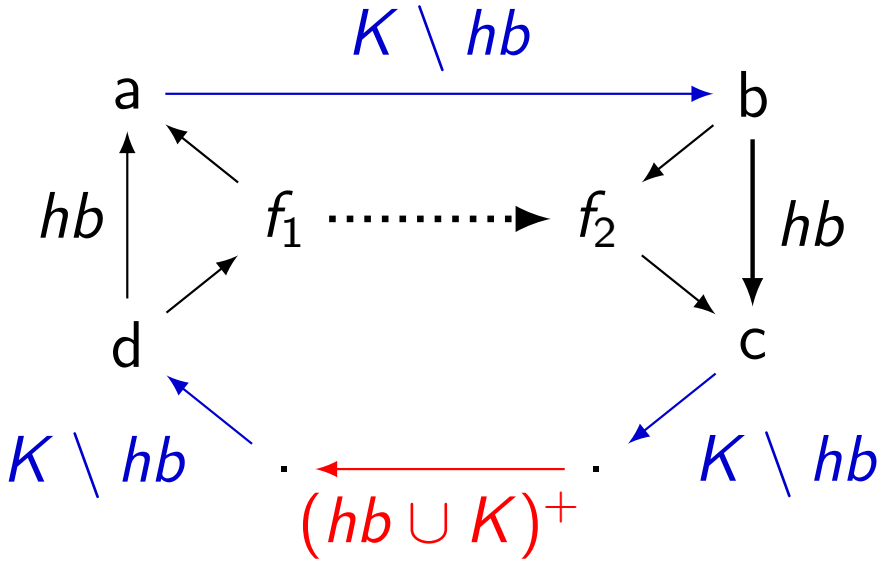
Proof of the simple reduction theorem (2/2)

Finally, consider a cycle with three or more K-edges.



Proof of the simple reduction theorem (2/2)

Finally, consider a cycle with three or more K-edges.



More advanced reduction theorem

Let $G = \langle A, po, rf \rangle$ be a well-formed RA-coherent execution. Assume that G is *WW-race-free* and there exists a set $B \subseteq A$ of *protected events* such that:

1. $(po \cup rf)^+$ is total on B.
2. If a races with b in G, then either $a \in B$ or $b \in B$.
3. For every G-racy write/update event $a \in B$ and G-racy read event $b \in B$, if $\langle a, b \rangle \in (po \cup rf)^+$, then $\langle a, c \rangle, \langle c, b \rangle \in (po \cup rf)^+$ for some fence event c.
4. For every G-racy write/update event $a \notin B$ and G-racy read event $b \notin B$, if $\langle a, b \rangle \in (po \cup rf)^+$, then $\langle a, c \rangle, \langle c, b \rangle \in (po \cup rf)^+$ for some fence *or protected* event c.

Then, G is SC-coherent.

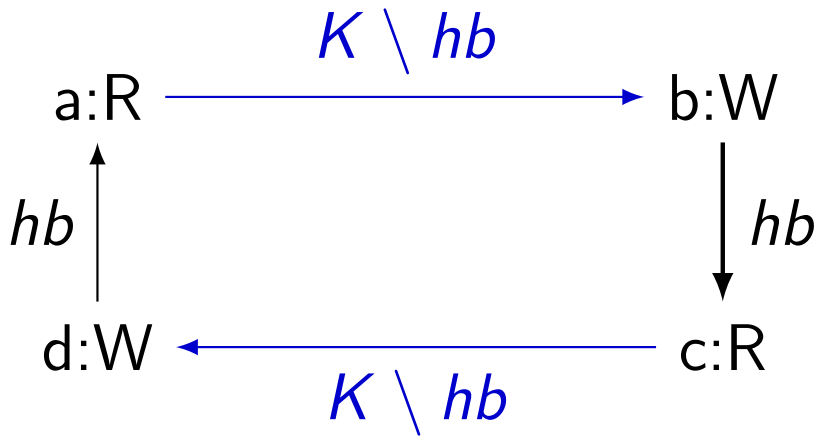
Proof sketch

Note:

- Because of WW-race-freedom, if $\langle a, b \rangle \in K \setminus hb$, then a is a read and b is a write (or update).

Consider minimal cycle in $(hb \cup K)$.

- Cycle with two K-edges:



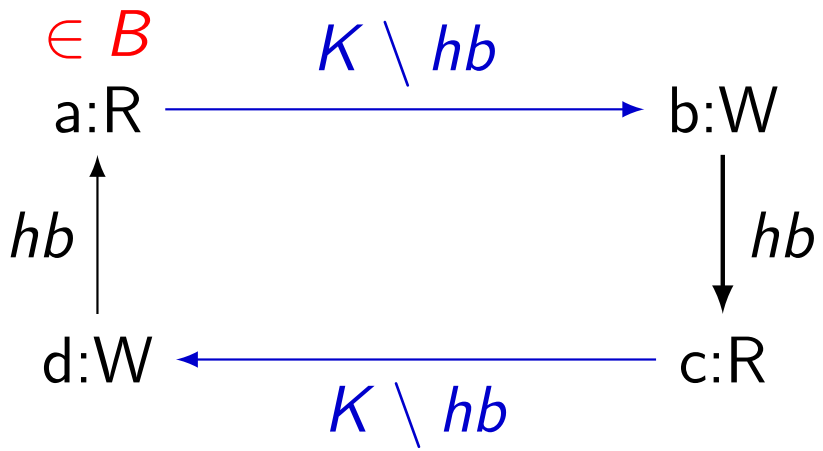
Proof sketch

Note:

- Because of WW-race-freedom, if $\langle a, b \rangle \in K \setminus hb$, then a is a read and b is a write (or update).

Consider minimal cycle in $(hb \cup K)$.

- Cycle with two K-edges:



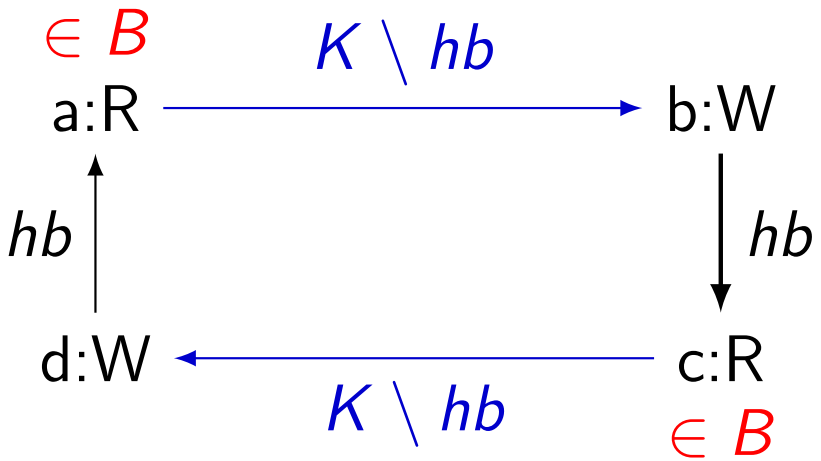
Proof sketch

Note:

- Because of WW-race-freedom, if $\langle a, b \rangle \in K \setminus hb$, then a is a read and b is a write (or update).

Consider minimal cycle in $(hb \cup K)$.

- Cycle with two K-edges:



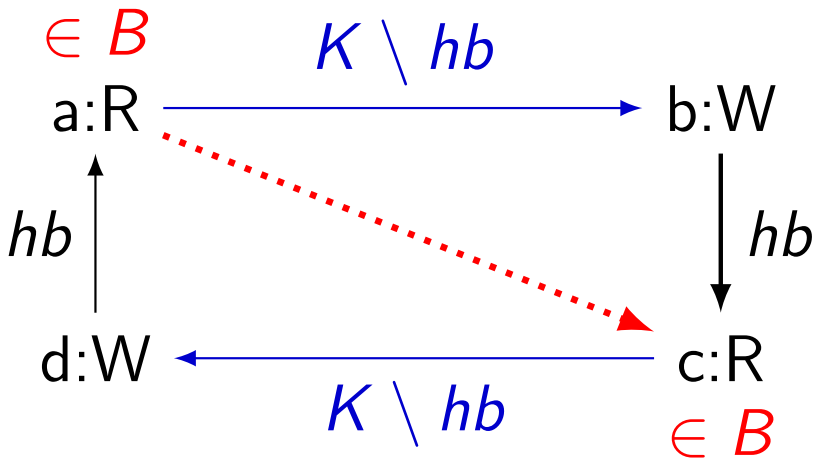
Proof sketch

Note:

- Because of WW-race-freedom, if $\langle a, b \rangle \in K \setminus hb$, then a is a read and b is a write (or update).

Consider minimal cycle in $(hb \cup K)$.

- Cycle with two K-edges:



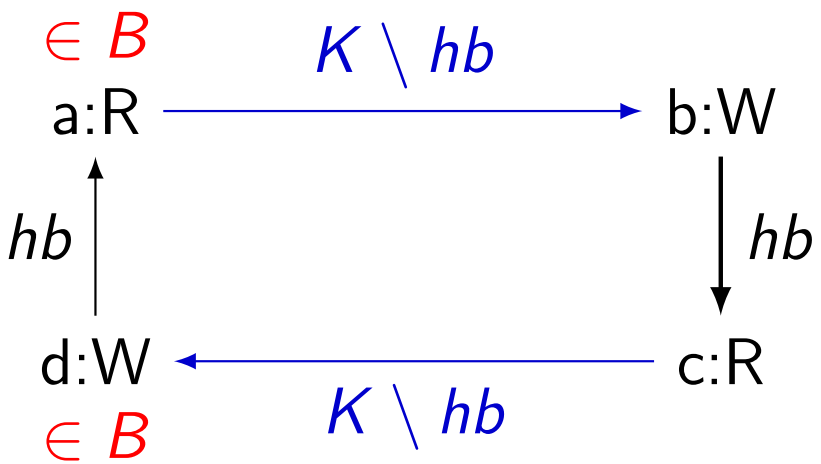
Proof sketch

Note:

- Because of WW-race-freedom, if $\langle a, b \rangle \in K \setminus hb$, then a is a read and b is a write (or update).

Consider minimal cycle in $(hb \cup K)$.

- Cycle with two K-edges:



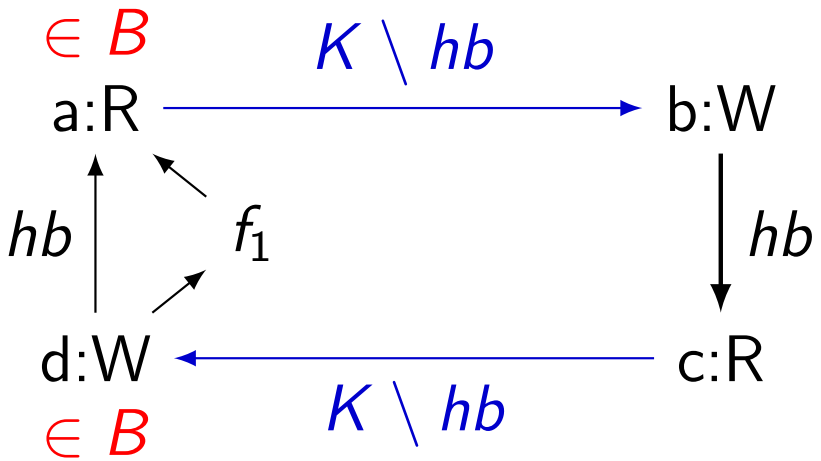
Proof sketch

Note:

- Because of WW-race-freedom, if $\langle a, b \rangle \in K \setminus hb$, then a is a read and b is a write (or update).

Consider minimal cycle in $(hb \cup K)$.

- Cycle with two K-edges:



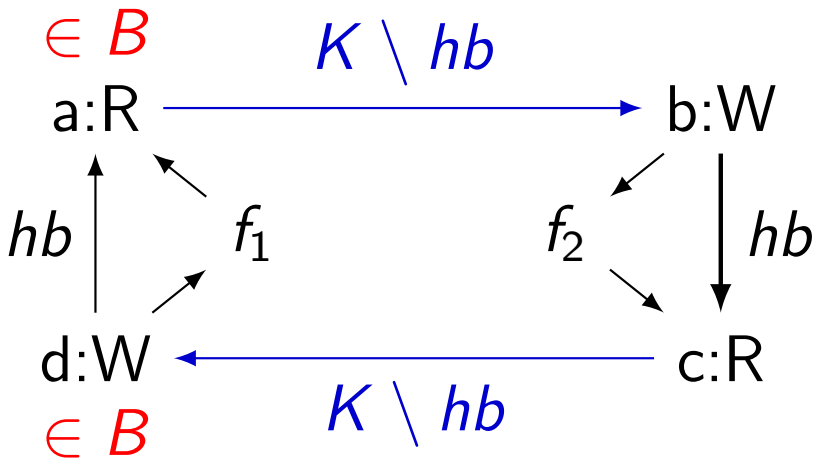
Proof sketch

Note:

- Because of WW-race-freedom, if $\langle a, b \rangle \in K \setminus hb$, then a is a read and b is a write (or update).

Consider minimal cycle in $(hb \cup K)$.

- Cycle with two K-edges:



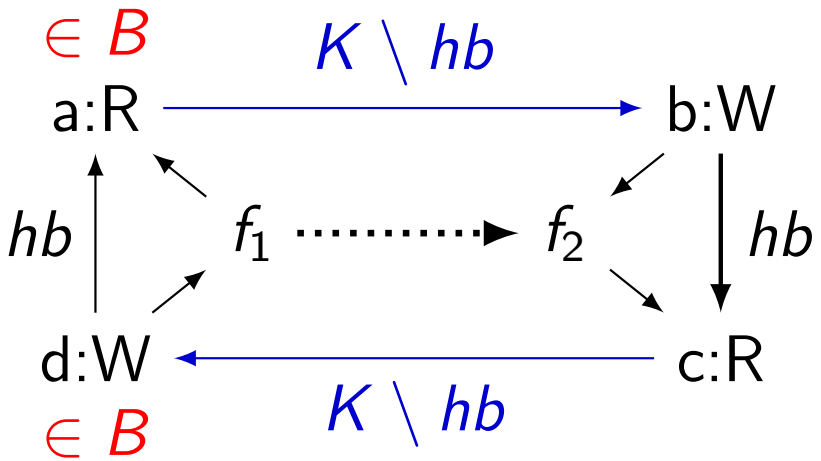
Proof sketch

Note:

- Because of WW-race-freedom, if $\langle a, b \rangle \in K \setminus hb$, then a is a read and b is a write (or update).

Consider minimal cycle in $(hb \cup K)$.

- Cycle with two K-edges:



Applying the theorem to RCU

```
rcu_quiescent_state():
   rc[get_my_tid()] := gc; fence();

rcu_thread_offline():
   rc[get_my_tid()] := 0; fence();

rcu_thread_online():
   rc[get_my_tid()] := gc; fence();

synchronize_rcu():
   local was_online := (rc[get_my_tid()] ≠ 0);
   if was_online then rc[get_my_tid()] := 0;
   lock();
     gc := gc + 1;
     fence();
     for i := 1 to N do wait (rc[i] ∈ {0,gc});
   unlock();
   if was_online then rc[get_my_tid()] := gc;
   fence();
```

Conclusion

Summary:

- Release-acquire consistency
- Adapting Owicki-Gries to RA consistency
- Restoring SC using fences

Other related work:

- Debugging the C/C++11 memory model
- Compilation and optimization results
- More advanced program logics for weak memory