Sequential Consistency in C11

October 24, 2022

Duc Than Nguyen

University of Illinois at Chicago

Outline

Outline

Introduction

- ❇ Programmer's goal—fully understand what they've written
- ❇ Compiler's responsibility—make code as optimized as possible
- ✘ Simple compiler optimizations can generate unexpected behaviors in concurrent setting
- ✔ Memory model—core of concurrent semantics of shared memory
  → alleviate tension between goals of programmer & compiler
- ☞ C11 memory model defines semantics in C progrogramming
- Batty et al. [2011] [1] formalized concurrency model in C++ standard
- Several issues discovered with semantics of SC C11 accesses
- Model has evolved with fixes and revisions (e.g., Vafeiadis et al. [2015], Batty et al. [2016], Lahav et al. [2017])

[1] Mathematizing C++ Concurrency. In POPL 2011.

Papers

Authors	Paper	Conference
Vafeiadis et al.	**Common compiler optimisations are invalid in the C11 memory model and what we can do about it**	POPL 2015
Batty et al.	**Overhauling SC atomics in C11 and OpenCL**	POPL 2016
Lahav et al.	**Repairing sequential consistency in C/C++11**	PLDI 2017

Outline

Sequential consistency

> … the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program. (Lamport [1979][2])

$$x := 0; \quad y := 0;$$

$$\begin{array}{l|l} x := 1 & y := 1 \\ a := {}^{*}y; \texttt{ //load 0} & b := {}^{*}x; \texttt{ //load 0} \end{array}$$

- Thread-local variables $a = b = 0$?
- 3 possible outcomes: (1) $a = b = 0$, (2) $a = 0, b = 1$, (3) $a = 1, b = 0$
- $a = b = 0$ cannot happen (no interleaving yields that result)
- Modern systems allow its behavior

[2] How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs. IEEE Trans. Computers 28, 9 (1979)

C11 memory model

$$\texttt{na} \sqsubset \texttt{rlx} \sqsubset \{\texttt{acq}, \texttt{rel}\} \sqsubset \texttt{sc}$$

The full C/C++11 is more general

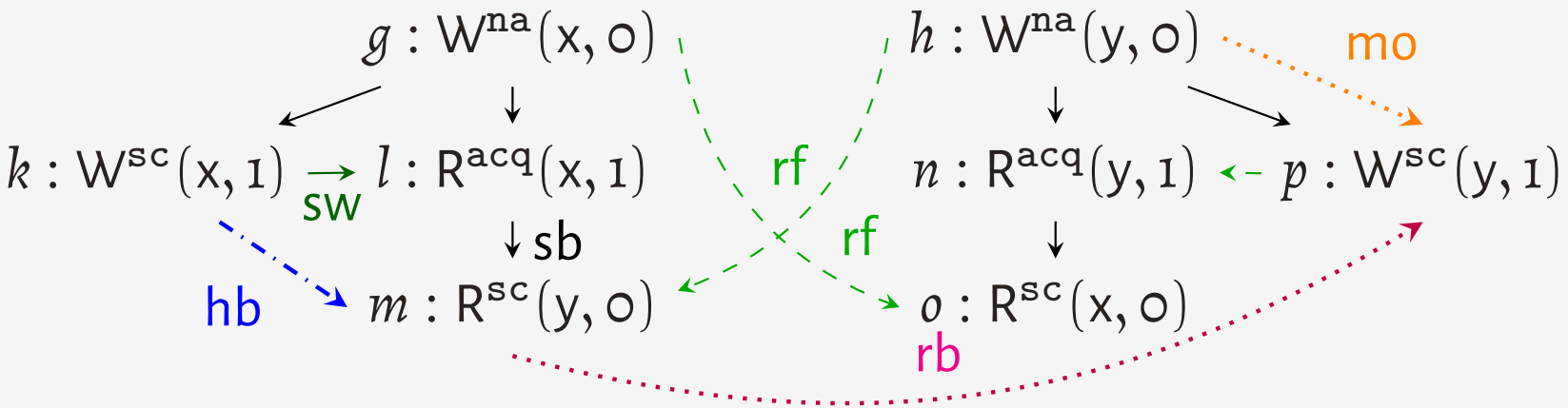
`sc` — Sequentially consistent accesses

`rel` — Release, and `acq` — Acquire accesses

`rlx` — Relaxed accesses

`na` — Non-atomic accesses (a.k.a. normal data accesses)

Declarative/Axiomatic memory model & Semantics



$$x :=_{na} 0; \quad y :=_{na} 0;$$

$$x :=_{sc} 1; \;\Big\|\; \begin{array}{l} a :=^{*acq} x \quad //1 \\ c :=^{*sc} y; \quad //0 \end{array} \;\Big\|\; \begin{array}{l} b :=^{*acq} y; \quad //1 \\ d :=^{*acq} x; \quad //0 \end{array} \;\Big\|\; y :=_{sc} 1;$$

events — nodes & relations — edges

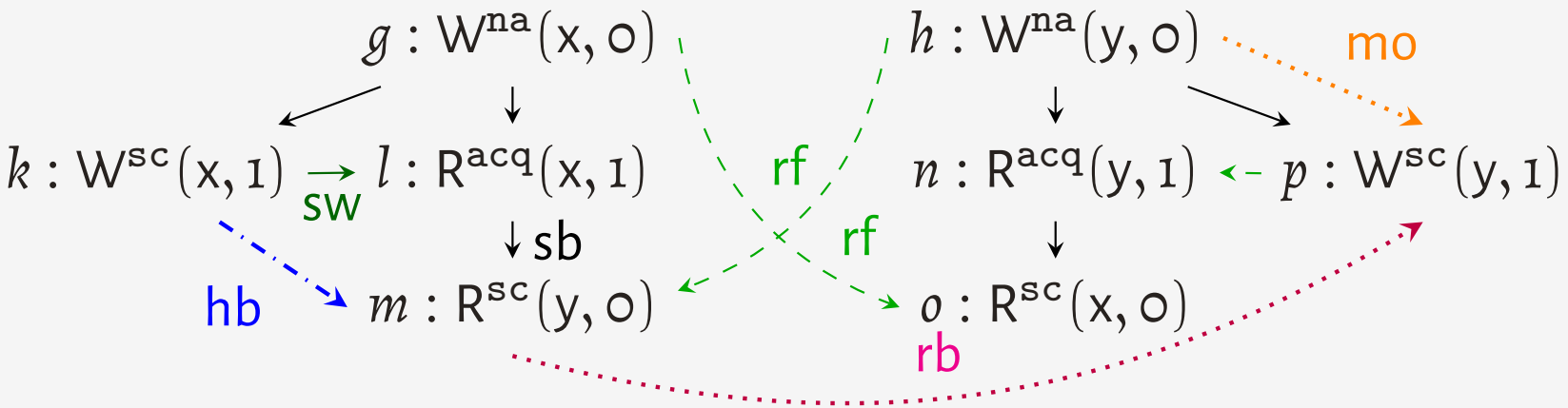
$W(x, 0)$ event $-x$ set to 0, and $R(x, 0)$ event $-x$ returns 0

sb— Sequenced-before : order of intra-thread memory accesses

rf— Reads-from: connect two write and read accesses

mo— Modification order: total order on all writes to same location

Declarative/Axiomatic memory model & Semantics



$$x :=_{na} 0; \quad y :=_{na} 0;$$

$$x :=_{sc} 1; \;\Big\|\; \begin{array}{l} a :=^{*acq} x \quad //1 \\ c :=^{*sc} y; \quad //0 \end{array} \;\Big\|\; \begin{array}{l} b :=^{*acq} y; \quad //1 \\ d :=^{*acq} x; \quad //0 \end{array} \;\Big\|\; y :=_{sc} 1;$$

Additional relational types

sw– Synchronized-with: an `acq` (`sc`) read reads from a `rel` (`sc`) write

hb– Happens-before: $hb = (sb \cup sw)^+$ (transitive closure)

rb– Reads-before: $rb = rf^{-1}; mo \setminus [E]$– read access reads from write that is mo– before, exclude update event from itself

Semantics

Rules of total order of sequentially consistent operations

1. Total order on SC operations: any two SC operations must be ordered w.r.t. each other
2. Be consistent with hb and mo restricted to SC atomics
3. SC reads (SCReads axiom) must either
   - read from most recent SC write before them in SC order, or
   - read from non-SC write that does **not** happen-before most recent SC write to that location

Outline

Optimization transformations (OT)

- Compiler converts program to architecture, preserving its semantics
- Compiler must be aware of memory consistency models of programming language & architecture
- C11 compilers (GCC, LLVM) rarely perform C code compilation in a single step
- Front-end translate C11 to intermediate representations (IR), compiler executes several OT on IR then generates target code
- Source & target code maintains same consistency model
- OT: reordering independent memory & removing redundant memory accesses
- $x :=_{\text{rel}} 1;\ y :=_{\text{na}} 1; \quad \rightsquigarrow \quad y :=_{\text{na}} 1;\ x :=_{\text{rel}} 1;$

Monotonicity

"Adding synchronisation should not introduce new behaviors"

- Adding a memory fence
- *Strengthening the access mode of an operation*
- Reducing parallelism $C_1 || C_2 \rightsquigarrow C_1; C_2$
- Roach motel reorderings

Strengthening access mode of an operation

(Recall) SCReads axiom — SC reads must either (1) read from most recent SC write before them in SC order, or (2) read from non-SC write that does **not** happen-before most recent SC write to that location

$$
\begin{array}{l||l||l||l}
x :=_{\text{rlx}} 1; & x :=_{\text{rlx}} 3; & y :=_{\text{sc}} 3; & s_1 := {}^{*\text{rlx}} x; \texttt{//1} \\
x :=_{\text{sc}} 2; & y :=_{\text{sc}} 2; & r := {}^{*\text{sc}} x; \texttt{//1} & s_2 := {}^{*\text{rlx}} x; \texttt{//2} \\
y :=_{\text{sc}} 1; & & & s_3 := {}^{*\text{rlx}} x; \texttt{//3} \\
 & & & t_1 := {}^{*\text{rlx}} y; \texttt{//1} \\
 & & & t_2 := {}^{*\text{rlx}} y; \texttt{//2} \\
 & & & t_3 := {}^{*\text{rlx}} y; \texttt{//3}
\end{array}
$$

? Allowed – $r = s_1 = t_1 = 1 \wedge s_2 = t_2 = 2 \wedge s_3 = t_3 = 3$

☞ If it holds, $x :=_{\text{sc}} 2$ — immediate SC prior write w.r.t. $r := {}^{*\text{sc}} x$

☞ Reading 1 of $r := {}^{*\text{sc}} x$ from $x :=_{\text{rlx}} 1$ is **not** valid
($x :=_{\text{rlx}} 1$ happens before $x :=_{\text{sc}} 2$ via sb)

Strengthening access mode of an operation

(Recall) SCReads axiom — SC reads must either (1) read from most recent SC write before them in SC order, or (2) read from non-SC write that does **not** happen-before most recent SC write to that location

$$
\begin{array}{l||l||l||l}
x :=_{\text{rlx}} 1; & x :=_{\text{sc}} 3; & y :=_{\text{sc}} 3; & s_1 := {}^{*\text{rlx}} x; \texttt{//1} \\
x :=_{\text{sc}} 2; & y :=_{\text{sc}} 2; & r := {}^{*\text{sc}} x; \texttt{//1} & s_2 := {}^{*\text{rlx}} x; \texttt{//2} \\
y :=_{\text{sc}} 1; & & & s_3 := {}^{*\text{rlx}} x; \texttt{//3} \\
 & & & t_1 := {}^{*\text{rlx}} y; \texttt{//1} \\
 & & & t_2 := {}^{*\text{rlx}} y; \texttt{//2} \\
 & & & t_3 := {}^{*\text{rlx}} y; \texttt{//3}
\end{array}
$$

? Strengthening $x :=_{\text{rlx}} 3$ into $x :=_{\text{sc}} 3$

☞ Establish SC order from $y :=_{\text{sc}} 1$ to $x :=_{\text{sc}} 3$

☞ $x :=_{\text{sc}} 3$ is immediately prior in SC order of $r := {}^{*\text{sc}} x$

☞ Reading 1 of $r := {}^{*\text{sc}} x$ is valid (since $x :=_{\text{rlx}} 1$ does not hb $x :=_{\text{sc}} 3$)

✖ New behavior introduced

Correcting the SCReads Axiom

$$\forall a, b.\ \mathsf{rf}(b) = a \wedge \mathrm{isSC}(b) \Rightarrow \mathrm{imm}(\mathrm{scr}, a, b) \vee \neg\mathrm{isSC}(a) \\ \wedge \nexists x.\ \mathsf{hb}(a, x) \wedge \mathrm{imm}(\mathrm{scr}, x, b) \tag{1}$$

❶ $\mathrm{isSC}(a) = \mathrm{mode}(a) = \mathtt{sc}$
❷ $\mathrm{imm}(R, a, b) = R(a, b) \wedge \nexists c.\ R(a, c) \wedge R(c, b)$
❸ $\mathrm{scr}(a, b) = \mathrm{sc}(a, b) \wedge \mathrm{iswrite}_{\mathrm{loc}(b)}(a)$
❹ $\mathrm{iswrite}_l(a) = \exists v.(\exists X, v'.\ \mathrm{lab}(a) \in \{\mathtt{W}^X(l, v), \mathtt{U}^X(l, v', v)\})$

? Problem: disallows a non-SC write that hb another SC-write that is immediately prior in SC order to same location

✔ Fix: change $\mathrm{imm}(\mathrm{scr}, x, b)$ into $\mathrm{scr}(x, b)$
constrains all SC prior same-location writes instead of only immediate prior write on same location

✔ No hb between write $\mathsf{rf}(b)$ and any same location write sc-prior to read $\rightarrow r :=^{*\mathrm{sc}} x$ reading from $x :=_{\mathrm{rlx}} 1$ not valid

Outline

Derived sets and relations

They are formalized as C11 axioms by referencing the C11 standard

❶ $\mathsf{irr}(\mathsf{S}; r_1)$ where $r_1 = \mathsf{hb}$

❷ $\mathsf{irr}(\mathsf{S}; r_2)$ where $r_2 = ([\mathrm{F}^{\mathrm{sc}}]; \mathsf{sb})^?; \mathsf{mo}; (\mathsf{sb}; [\mathrm{F}^{\mathrm{sc}}])^?$

❸ $\mathsf{irr}(\mathsf{S}; r_3)$ where $r_3 = \mathsf{rf}^{-1}; [\mathrm{E}^{\mathrm{sc}}]; \mathsf{mo}$

❹ $\mathsf{irr}((\mathsf{S} \setminus (\mathsf{mo}; \mathsf{S})); r_4)$ where $r_4 = \mathsf{rf}^{-1}; \mathsf{hbl}; [\mathrm{E}^{\mathrm{W}}]$,
and $\mathsf{hbl} - \mathsf{hb}$ to events on same location

❺ $\mathsf{irr}(\mathsf{S}; r_5)$ where $r_5 = ([\mathrm{F}^{\mathrm{sc}}]; \mathsf{sb}); \mathsf{rb}$

❻ $\mathsf{irr}(\mathsf{S}; r_6)$ where $r_6 = \mathsf{rb}; (\mathsf{sb}; [\mathrm{F}^{\mathrm{sc}}])$

❼ $\mathsf{irr}(\mathsf{S}; r_7)$ where $r_7 = ([\mathrm{F}^{\mathrm{sc}}]; \mathsf{sb}); \mathsf{rb}; (\mathsf{sb}; [\mathrm{F}^{\mathrm{sc}}])$

S— Sequential consistency order

Axioms ❸ ❹ refer to as SCReads axiom

Axioms ❺ ❻ ❼ govern SC fences

❇ All axioms express as $\mathsf{irr}(\mathsf{S}; r)$ except ❹

Construction partial order on SC operations

❹ $\mathsf{irr}((S \setminus (\mathsf{mo}; S)); r_4)$, where $r_4 = \mathsf{rf}^{-1}; \mathsf{hbl}; [E^W]$, and hbl is hb to events on same location

✏ Replace $S \setminus (\mathsf{mo}; S)$ with S, obtain new ❹✓ $\mathsf{irr}(S; r_4)$

❹✓ coincides with revised model offered by Vafeiadis et al. [2015] in "Correcting the SCReads Axiom"

❹✓ disallows an SC read to see any write that happens before any SC write in S

$$\mathsf{acyclic}([E^{sc}]; (r_1 \cup r_2 \cup r_3 \cup r_4 \cup r_5 \cup r_6 \cup r_7); [E^{sc}]) \qquad (S_{\mathsf{partial}})$$

■ Proved $S_{\mathsf{partial}} = \exists S.\ \mathsf{WfS} \wedge ❶ \wedge ❷ \wedge ❸ \wedge ❹✓ \wedge ❺ \wedge ❻ \wedge ❼$

❀ S_{partial} does not require S anymore

✈ This axiom is faster to simulate!

✔ Existing compilation schemes (x86 and Power) remain valid

Stronger & simpler SC axiom

? Question: *Strengthen SC semantics without requiring changes to compilation schemes of C11 target architectures (x86 and Power)?*

- ❺ ❻ and ❼ contain rb at begin or finish at a fence
- ❹✓ contains rb since $r_4 = \mathsf{rf}^{-1}; \mathsf{hbl}; [\mathrm{E}^{\mathrm{W}}]$
  where hbl is hb to events on same location
  $\longrightarrow r_4 = \mathsf{rf}^{-1}; \mathsf{mo}$ and $\mathsf{rb} = \mathsf{rf}^{-1}; \mathsf{mo}$
- ❸ contains $r_3 = \mathsf{rf}^{-1}; [\mathrm{E}^{\mathrm{sc}}]; \mathsf{mo}$
- ✂ Remove $[\mathrm{E}^{\mathrm{sc}}]$, we have $r_3 = \mathsf{rb}$ and new ❸✓ $\mathsf{irr}(\mathrm{S}; \mathsf{rb})$
- ❀ Enhance $\mathrm{S}_{\mathsf{partial}}$ to have
  $\mathrm{S}_{\mathsf{simp}} = \mathsf{acyclic}([\mathrm{E}^{\mathrm{sc}}]; (r_1 \cup r_2 \cup \mathsf{rb} \cup r_4 \cup r_5 \cup r_6 \cup r_7); [\mathrm{E}^{\mathrm{sc}}])$

$$\mathsf{acyclic}([\mathrm{E}^{\mathrm{sc}}]; (([\mathrm{F}^{\mathrm{sc}}]; \mathsf{sb})^{?}; (\mathsf{hb} \cup \mathsf{rb} \cup \mathsf{mo}); (\mathsf{sb}; [\mathrm{F}^{\mathrm{sc}}])^{?}); [\mathrm{E}^{\mathrm{sc}}]) \quad (\mathrm{S}_{\mathsf{simp}})$$

Outline

Compilation to Power is Broken

❶ $[E^{SC}]; hb; [E^{SC}] \subseteq S$

❷ $[E^{SC}]; mo; [E^{SC}] \subseteq S$

❸ $[E^{SC}]; rb; [E^{SC}] \subseteq S$

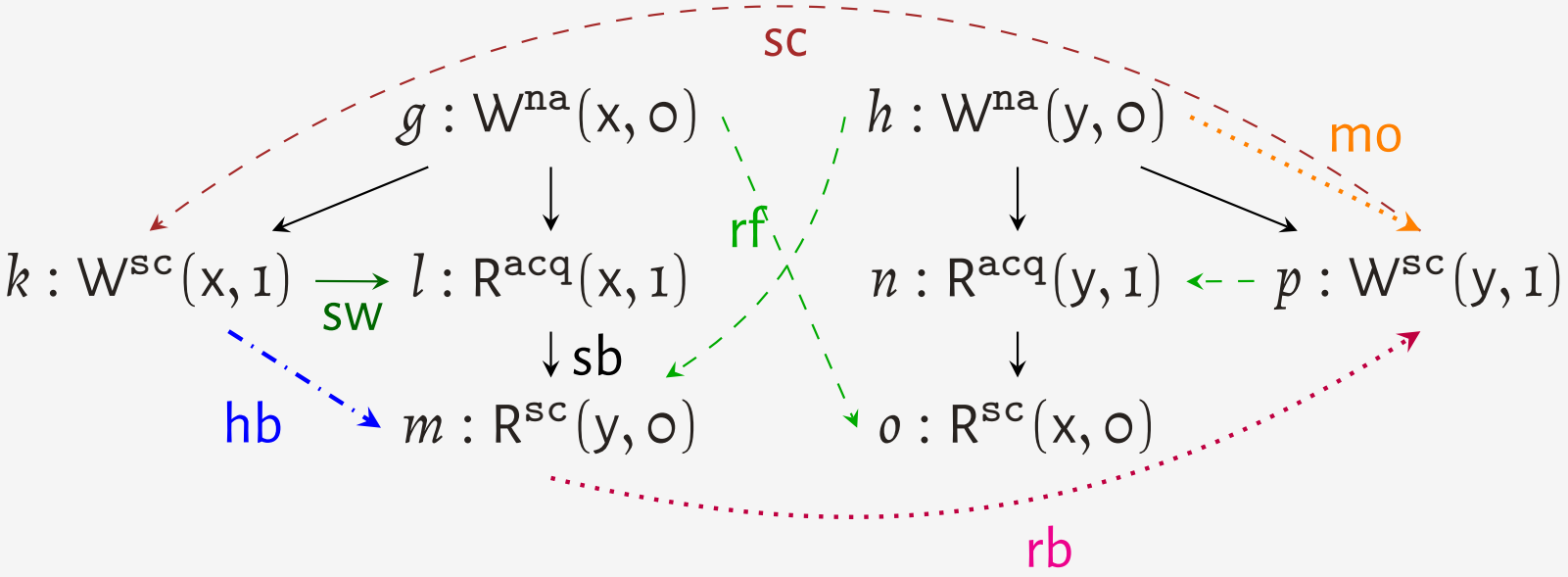
❹ ❺ ❻ ❼ indicate that S is required to comply with a few more conditions about SC fences.

Same location: Power & ARM ensure compilation preserves condition

❷ & ❸ force ordering between accesses to *same location*

❶ forces between accesses of *different locations*, requiring insertion of fence instructions

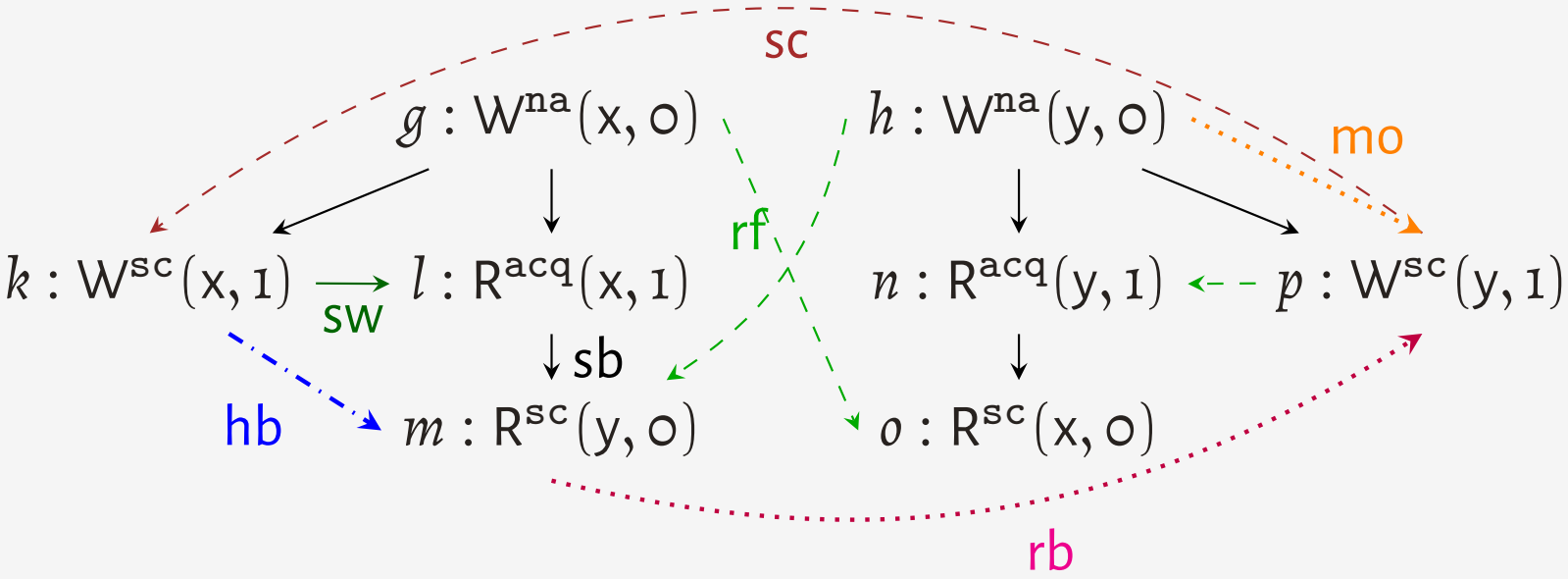
Compilation to Power is Broken



$$x :=_{sc} 1; \;\Big\|\; \begin{array}{l} a :=^{*acq} x \;\; //1 \\ c :=^{*sc} y; \;\; //0 \end{array} \;\Big\|\; \begin{array}{l} b :=^{*acq} y; \;\; //1 \\ d :=^{*acq} x; \;\; //0 \end{array} \;\Big\|\; y :=_{sc} 1;$$

- ❇ Independent Reads Independent Writes (IRIW)
- ✖ Constraints by [Batty et al. 2016] — too strong to preserve compiler correctness from C11 to Power and ARMv7

Compilation to Power is Broken



- ❇ $S(p, k)$ via sc; $S(k, m)$ via hb (where hb $= (\mathsf{sw} \cup \mathsf{sb})^+$); then $\mathrm{sc}(p, m)$ via transitivity.
- ❇ $S(m, p)$ via rb (where rb $= \mathsf{rf}^{-1}; \mathsf{mo}$) $\longrightarrow$ S contains cycles
- ✖ banned by C11
- ✔ permitted by compilation into Power

Fixing the model

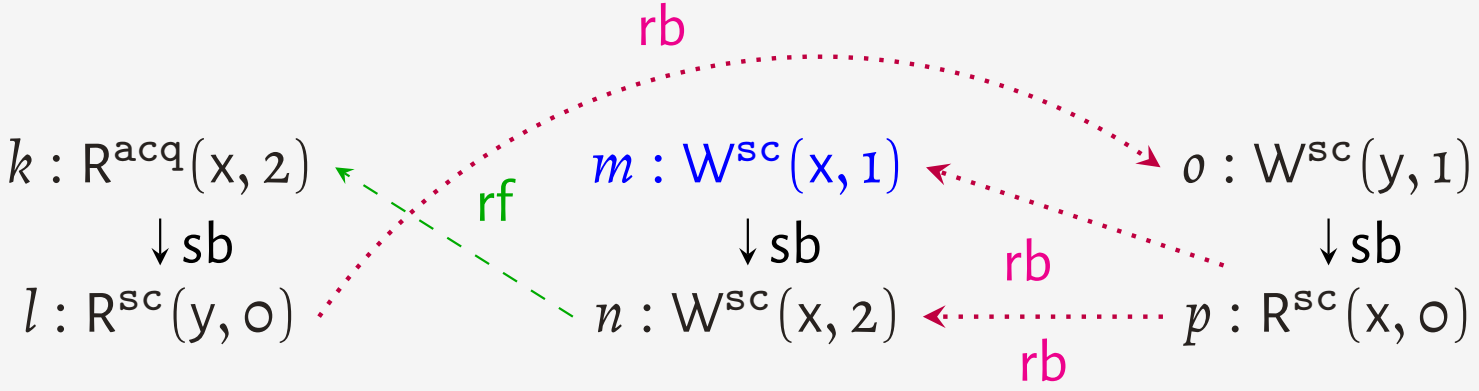
- ❶ $[E^{sc}]; hb; [E^{sc}] \subseteq S-$ same or different location
- ❇ same location: hardware maintain the order
- ❇ different location: a sync fence between SC accesses
  $\rightarrow$ start and ends hb with sb, i.e., $sb; hb; sb$
- ❀ Fix: replace hb with $(sb \cup sb; hb; sb \cup hb|_{=loc})$

New model ❶✓

$$\text{acyclic}([E^{sc}]; (sb \cup sb; hb; sb \cup hb|_{=loc} \cup mo \cup rb); [E^{sc}]) \quad (❶✓)$$

❶✓ forbids elimination of SC write immediately followed by SC write to same location, and SC read immediately followed by SC read from same location

Enabling Elimination of SC Accesses




```
a :=*acq x;  //2  ||  x :=sc 1;  ||  y :=sc 1;
b :=*sc y;   //2  ||  x :=sc 2;  ||  c :=*sc x;  //0
```

❶✓ forbids elimination of SC write immediately followed by SC write to same location, and SC read immediately followed by SC read from same location

- ❇ Eliminate $\mathrm{x} :=_{\mathrm{sc}} 1;$ (event $m : \mathrm{W}^{\mathrm{sc}}(\mathrm{x}, 1)$)
- ❇ Create cycle $n \to k \to l \to o \to p \to n$
- ✖ Violate condition ❶✓ must be acyclic

Fixing the model

$\mathsf{acyclic}([\mathrm{E}^{\mathrm{sc}}]; (\mathsf{sb} \cup \mathsf{sb}; \mathsf{hb}; \mathsf{sb} \cup \mathsf{hb}|_{=\mathsf{loc}} \cup \mathsf{mo} \cup \mathsf{rb}); [\mathrm{E}^{\mathrm{sc}}])$ (❶✓)

- ❀ Fix: weaken condition by replacing
  $\mathsf{sb}; \mathsf{hb}; \mathsf{sb}$ with $\mathsf{sb}|_{\neq\mathsf{loc}}; \mathsf{hb}; \mathsf{sb}|_{\neq\mathsf{loc}}$
  where $\mathsf{sb}|_{\neq\mathsf{loc}}$ – sb edges to different location

New condition (named SC-before) requires acyclicity of $[\mathrm{E}^{\mathrm{sc}}]; \mathsf{scb}; [\mathrm{E}^{\mathrm{sc}}]$

$$\mathsf{scb} = \mathsf{sb} \cup \mathsf{sb}|_{\neq\mathsf{loc}}; \mathsf{hb}; \mathsf{sb}|_{\neq\mathsf{loc}} \cup \mathsf{hb}|_{=\mathsf{loc}} \cup \mathsf{mo} \cup \mathsf{rb}$$

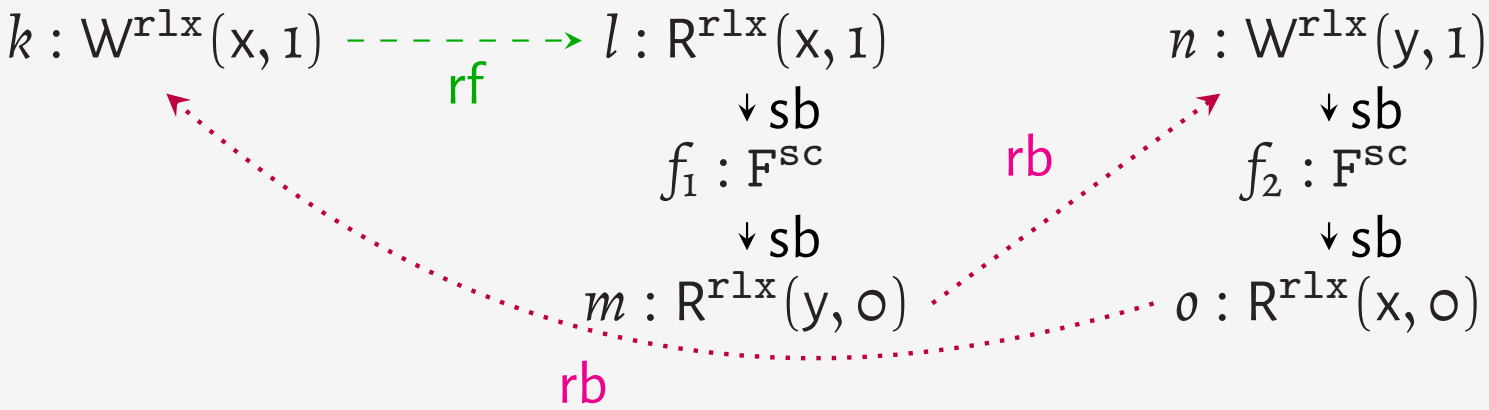
SC Fences are Too Weak

? Question: *Shall adding SC fences between every pair of shared memory restore interleaving behavior?*

- ✖ Original C11 & Batty et al. [2016] do **not** hold
- ✔ hold for Power & ARM
- ❇ Adapted from Batty el al. [2016]
  $\mathsf{acyclic}(([\mathsf{E}^{\mathsf{sc}}] \cup [\mathsf{F}^{\mathsf{sc}}]; \mathsf{sb}^{?}); (\mathsf{hb} \cup \mathsf{mo} \cup \mathsf{rb}); ([\mathsf{E}^{\mathsf{sc}}] \cup \mathsf{sb}^{?}; [\mathsf{F}^{\mathsf{sc}}]))$
- ❇ Replace $\mathsf{hb} \cup \mathsf{mo} \cup \mathsf{rb}$ with scb
  where $\mathsf{scb} = \mathsf{sb} \cup \mathsf{sb}|_{\neq \mathsf{loc}}; \mathsf{hb}; \mathsf{sb}|_{\neq \mathsf{loc}} \cup \mathsf{hb}|_{=\mathsf{loc}} \cup \mathsf{mo} \cup \mathsf{rb}$
- ❀ Require acyclicity of psc_1

$$\mathsf{psc}_1 = ([\mathsf{E}^{\mathsf{sc}}] \cup [\mathsf{F}^{\mathsf{sc}}]; \mathsf{sb}^{?}); \mathsf{scb}; ([\mathsf{E}^{\mathsf{sc}}] \cup \mathsf{sb}^{?}; [\mathsf{F}^{\mathsf{sc}}])$$

SC Fences are Too Weak



$$x :=_{rlx} 1; \quad \Big\| \quad \begin{array}{l} a :=^{*rlx} x; \quad //1 \\ \text{fence}_{sc} \\ b :=^{*rlx} y; \quad //0 \end{array} \quad \Big\| \quad \begin{array}{l} y :=_{rlx} 1; \\ \text{fence}_{sc} \\ c :=^{*rlx} x; \quad //0 \end{array}$$

- ☞ Path f_1 to f_2 via $\mathsf{sb} \longrightarrow \mathsf{rb} \longrightarrow \mathsf{sb}$
- ☞ Path f_2 to f_1 via $\mathsf{sb} \longrightarrow \mathsf{rb} \longrightarrow \mathsf{rf} \longrightarrow \mathsf{sb}$

  Applying $\mathsf{psc}_1 = ([E^{sc}] \cup [F^{sc}]; \mathsf{sb}^?); \mathsf{scb}; ([E^{sc}] \cup \mathsf{sb}^?; [F^{sc}])$ is acyclic
- ✖ Path f_2 to f_1 contributes neither Batty et al. [2016] nor psc_1

Fixing model

- Extended-coherence-order relation eco = $(\mathsf{rf} \cup \mathsf{mo} \cup \mathsf{rb})^+$
- Disallow the weak behavior, it is impossible for either psc_1 or $[\mathrm{F}^{\mathrm{sc}}]; \mathsf{sb}; \mathsf{eco}; \mathsf{sb}; [\mathrm{F}^{\mathrm{sc}}]$ containing cycles

$$\mathsf{acyclic}(\mathsf{psc}_1 \cup [\mathrm{F}^{\mathrm{sc}}]; \mathsf{sb}; \mathsf{eco}; \mathsf{sb}; [\mathrm{F}^{\mathrm{sc}}])$$

Outline

Selected critique

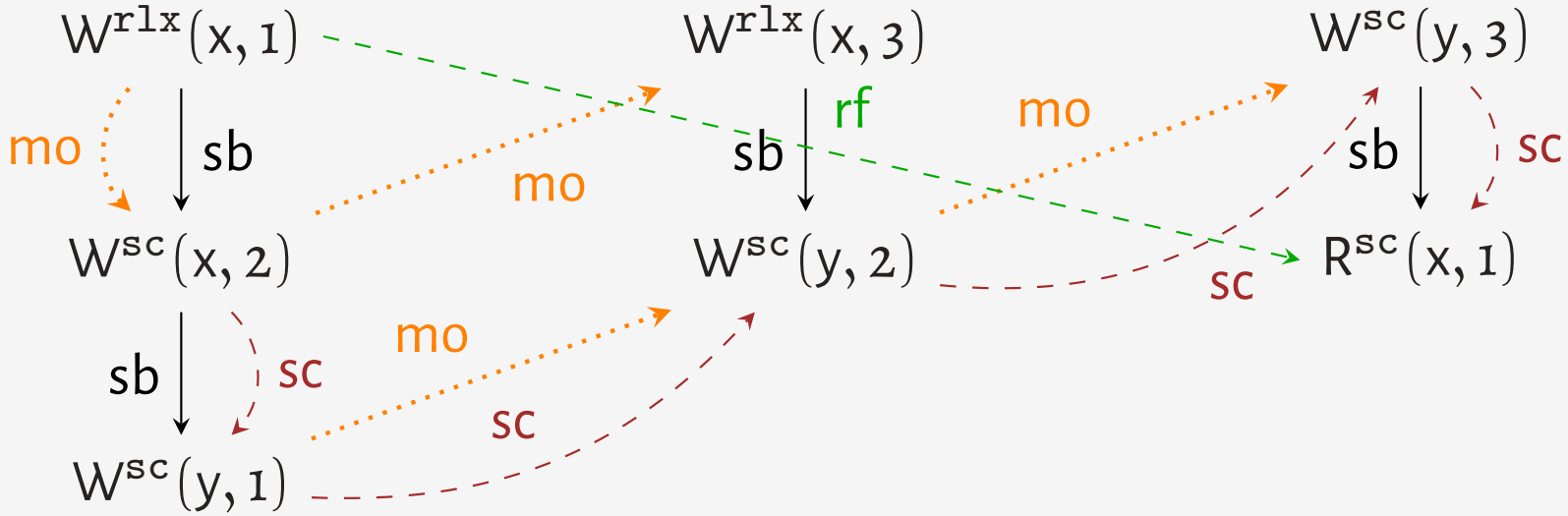
- (Vafeiadis et al. 2015) — disallows reordering of non-atomic load and store operations, when offered solution to out-of-thin-air reads that restricts $(\mathsf{hb} \cup \mathsf{rf})$ cycle (source or destination of rf edge is non-atomic)
- (Batty et al. 2016) — strong to preserve compiler correctness from C11 to Power and ARMv7
- (Lahav et al. 2017) — RC11 directly map high-level primitive to sequence of machine instructions without code optimizations, then challenging to apply methods to verify optimization passes

Thank you for listening!

Outline

backup slides

Strengthening access mode of an operation



$$
\begin{array}{l|l|l|l}
x :=_{\text{rlx}} 1; & x :=_{\text{rlx}} 3; & y :=_{\text{sc}} 3; & s_1 := {}^{*\text{rlx}} x; \ //1 \\
x :=_{\text{sc}} 2; & y :=_{\text{sc}} 2; & r := {}^{*\text{sc}} x; \ //1 & s_2 := {}^{*\text{rlx}} x; \ //2 \\
y :=_{\text{sc}} 1; & & & s_3 := {}^{*\text{rlx}} x; \ //3 \\
 & & & t_1 := {}^{*\text{rlx}} y; \ //1 \\
 & & & t_2 := {}^{*\text{rlx}} y; \ //2 \\
 & & & t_3 := {}^{*\text{rlx}} y; \ //3
\end{array}
$$