Repairing Sequential Consistency in C/C++11

Ori Lahav[1] Viktor Vafeiadis[1] Jeehoon Kang[2]
Chung-Kil Hur[2] Derek Dreyer[1]

[1]Max Planck Institute for Software Systems (MPI-SWS)

[2]Seoul National University

PLDI 2017

C11's spectrum of consistency

Access modes

`non-atomic` $\sqsubset$ `relaxed` $\sqsubset$ `release/acquire` $\sqsubset$ `sc`

C11's spectrum of consistency

Access modes

$$\texttt{non-atomic} \sqsubset \texttt{relaxed} \sqsubset \texttt{release/acquire} \sqsubset \texttt{sc}$$

Message passing

$$\begin{array}{l} x :=_{\texttt{sc}} 1; \\ y :=_{\texttt{sc}} 1; \end{array} \;\Big\|\; \begin{array}{ll} a := y_{\texttt{sc}}; & /\!/\, 1 \\ b := x_{\texttt{sc}}; & /\!/\, 0 \end{array} \quad \cong \quad \begin{array}{l} x :=_{\texttt{rlx}} 1; \\ y :=_{\texttt{rel}} 1; \end{array} \;\Big\|\; \begin{array}{ll} a := y_{\texttt{acq}}; & /\!/\, 1 \\ b := x_{\texttt{rlx}}; & /\!/\, 0 \end{array}$$

Store buffer

$$\begin{array}{ll} x :=_{\texttt{sc}} 1; & \\ a := y_{\texttt{sc}}; & /\!/\, 0 \end{array} \;\Big\|\; \begin{array}{ll} y :=_{\texttt{sc}} 1; & \\ b := x_{\texttt{sc}}; & /\!/\, 0 \end{array} \quad \not\cong \quad \begin{array}{ll} x :=_{\texttt{rel}} 1; & \\ a := y_{\texttt{sc}}; & /\!/\, 0 \end{array} \;\Big\|\; \begin{array}{ll} y :=_{\texttt{sc}} 1; & \\ b := x_{\texttt{sc}}; & /\!/\, 0 \end{array}$$

C11's spectrum of consistency

Access modes

$$\texttt{non-atomic} \sqsubset \texttt{relaxed} \sqsubset \texttt{release/acquire} \sqsubset \boxed{\texttt{sc}}$$

Message passing

$$\begin{array}{l} x :=_{\texttt{sc}} 1; \\ y :=_{\texttt{sc}} 1; \end{array} \Bigg\| \begin{array}{ll} a := y_{\texttt{sc}}; & /\!/\, 1 \\ b := x_{\texttt{sc}}; & /\!/\, 0 \end{array} \quad \cong \quad \begin{array}{l} x :=_{\texttt{rlx}} 1; \\ y :=_{\texttt{rel}} 1; \end{array} \Bigg\| \begin{array}{ll} a := y_{\texttt{acq}}; & /\!/\, 1 \\ b := x_{\texttt{rlx}}; & /\!/\, 0 \end{array}$$

Store buffer

$$\begin{array}{ll} x :=_{\texttt{sc}} 1; & \\ a := y_{\texttt{sc}}; & /\!/\, 0 \end{array} \Bigg\| \begin{array}{ll} y :=_{\texttt{sc}} 1; & \\ b := x_{\texttt{sc}}; & /\!/\, 0 \end{array} \quad \not\cong \quad \begin{array}{ll} x :=_{\texttt{rel}} 1; & \\ a := y_{\texttt{sc}}; & /\!/\, 0 \end{array} \Bigg\| \begin{array}{ll} y :=_{\texttt{sc}} 1; & \\ b := x_{\texttt{sc}}; & /\!/\, 0 \end{array}$$

C11's spectrum of consistency

Access modes

`non-atomic` ⊏ `relaxed` ⊏ `release/acquire` ⊏ `sc`

Mes

Sto

x

a

0

1. SC semantics is too strong (new correctness problem!)
2. SC semantics is too weak (SC-fences)
3. Out-of-thin-air reads (relaxed accesses)

C11's spectrum of consistency

Access modes

`non-atomic` ⊏ `relaxed` ⊏ `release/acquire` ⊏ `sc`

1. SC semantics is too strong (new correctness problem!)

   We show how to get SC semantics just right!

2. SC semantics is too weak (SC-fences)

3. Out-of-thin-air reads (relaxed accesses)

C11's spectrum of consistency

Access modes

non-atomic $\sqsubset$ `relaxed` $\sqsubset$ release/acquire $\sqsubset$ `sc`

1. SC semantics is too strong (new correctness problem!)

   We show how to get SC semantics just right!

2. SC semantics is too weak (SC-fences)

3. Out-of-thin-air reads (relaxed accesses)

Semantics of SC-atomics is too strong!

Example due to Yatin Manerkar et al. [CoRR abs/1611.01507]

$$\begin{array}{l} a := x_{\mathtt{acq}}; \quad /\!/\, 1 \\ b := y_{\mathtt{sc}}; \quad /\!/\, 0 \end{array} \;\Big\|\; x :=_{\mathtt{sc}} 1; \;\Big\|\; y :=_{\mathtt{sc}} 1; \;\Big\|\; \begin{array}{l} c := y_{\mathtt{acq}}; \quad /\!/\, 1 \\ d := x_{\mathtt{sc}}; \quad /\!/\, 0 \end{array}$$

C/C++11: behavior disallowed

Semantics of SC-atomics is too strong!

Example due to Yatin Manerkar et al. [CoRR abs/1611.01507]

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

C/C++11: behavior disallowed

Compilation of C/C++11 to Power

	$\mathtt{R}^{\mathtt{rlx}} \mapsto$ `ld`	$\mathtt{W}^{\mathtt{rlx}} \mapsto$ `st`
	$\mathtt{R}^{\mathtt{acq}} \mapsto$ `ld;lwsync`	$\mathtt{W}^{\mathtt{rel}} \mapsto$ `lwsync;st`
Leading sync:	$\mathtt{R}^{\mathtt{sc}} \mapsto$ `sync;ld;lwsync`	$\mathtt{W}^{\mathtt{sc}} \mapsto$ `sync;st`
Trailing sync:	$\mathtt{R}^{\mathtt{sc}} \mapsto$ `ld;sync`	$\mathtt{W}^{\mathtt{sc}} \mapsto$ `lwsync;st;sync`

Semantics of SC-atomics is too strong!

Example due to Yatin Manerkar et al. [CoRR abs/1611.01507]

$$
\begin{array}{l||l||l||l}
a := x_{\mathtt{acq}}; \quad /\!/ 1 & & & c := y_{\mathtt{acq}}; \quad /\!/ 1 \\
b := y_{\mathtt{sc}}; \quad /\!/ 0 & x :=_{\mathtt{sc}} 1; & y :=_{\mathtt{sc}} 1; & d := x_{\mathtt{sc}}; \quad /\!/ 0
\end{array}
$$

C/C++11: behavior disallowed

Compilation of C/C++11 to Power

	$\mathtt{R}^{\mathtt{rlx}} \mapsto$	`ld`	$\mathtt{W}^{\mathtt{rlx}} \mapsto$		`st`
	$\mathtt{R}^{\mathtt{acq}} \mapsto$	`ld;lwsync`	$\mathtt{W}^{\mathtt{rel}} \mapsto$	`lwsync;`	`st`
Leading sync:	$\mathtt{R}^{\mathtt{sc}} \mapsto$	`sync;ld;lwsync`	$\mathtt{W}^{\mathtt{sc}} \mapsto$	`sync;`	`st`
Trailing sync:	$\mathtt{R}^{\mathtt{sc}} \mapsto$	`ld; sync`	$\mathtt{W}^{\mathtt{sc}} \mapsto$	`lwsync;`	`st;sync`

Compilation result with "trailing sync" convention:

$$
\begin{array}{l||l||l||l}
a := x; \ /\!/ 1 & & & c := y; \ /\!/ 1 \\
\mathtt{lwsync}; & & & \mathtt{lwsync}; \\
b := y; \ /\!/ 0 & x := 1; & y := 1; & d := x; \ /\!/ 0 \\
\mathtt{sync}; & \mathtt{sync}; & \mathtt{sync}; & \mathtt{sync};
\end{array}
$$

Power: behavior allowed

Semantics of SC-atomics is too strong!

Other examples show unsoundness of:

- Leading `sync` compilation (implemented in GCC and LLVM)
- Placing `sync` both before and after SC-accesses

In order to recover the correctness of existing compilers, we suggest to weaken the standard.

C11's declarative semantics 101

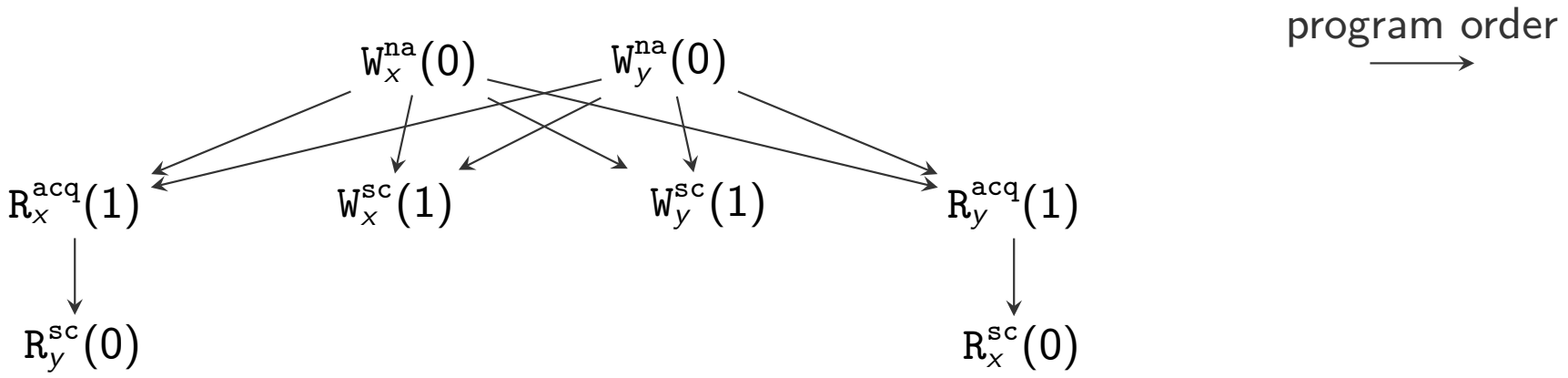
Batty et al. [POPL'16]

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

C11's declarative semantics 101

Batty et al. [POPL'16]

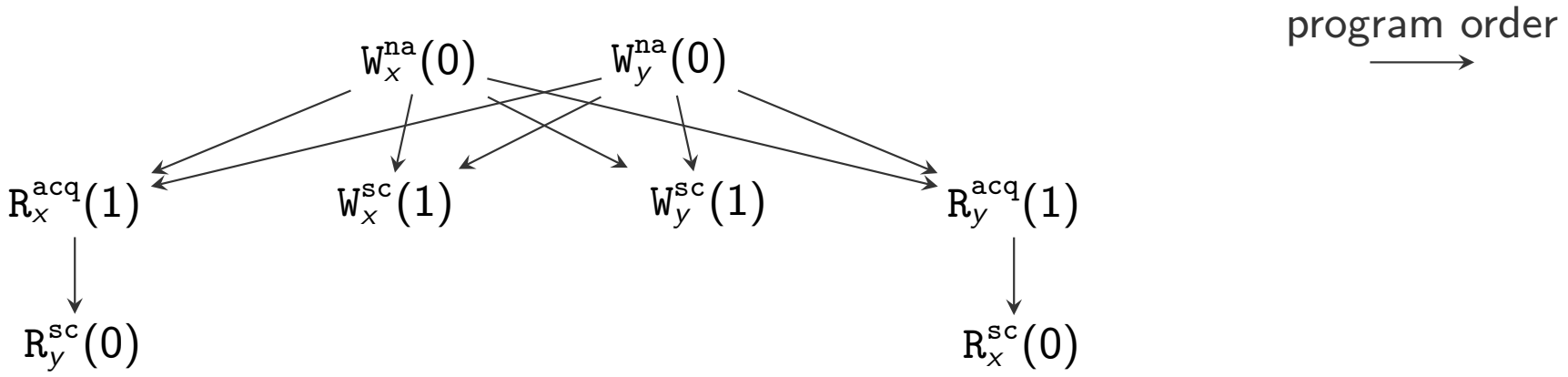
$$a := x_{\mathtt{acq}}; \quad /\!/ 1 \qquad b := y_{\mathtt{sc}}; \quad /\!/ 0 \;\Big\|\; x :=_{\mathtt{sc}} 1; \;\Big\|\; y :=_{\mathtt{sc}} 1; \;\Big\|\; c := y_{\mathtt{acq}}; \quad /\!/ 1 \qquad d := x_{\mathtt{sc}}; \quad /\!/ 0$$



C11's declarative semantics 101

Batty et al. [POPL'16]

$$a := x_{\mathtt{acq}}; \quad /\!/ 1 \qquad b := y_{\mathtt{sc}}; \quad /\!/ 0 \;\Big\|\; x :=_{\mathtt{sc}} 1; \;\Big\|\; y :=_{\mathtt{sc}} 1; \;\Big\|\; c := y_{\mathtt{acq}}; \quad /\!/ 1 \qquad d := x_{\mathtt{sc}}; \quad /\!/ 0$$



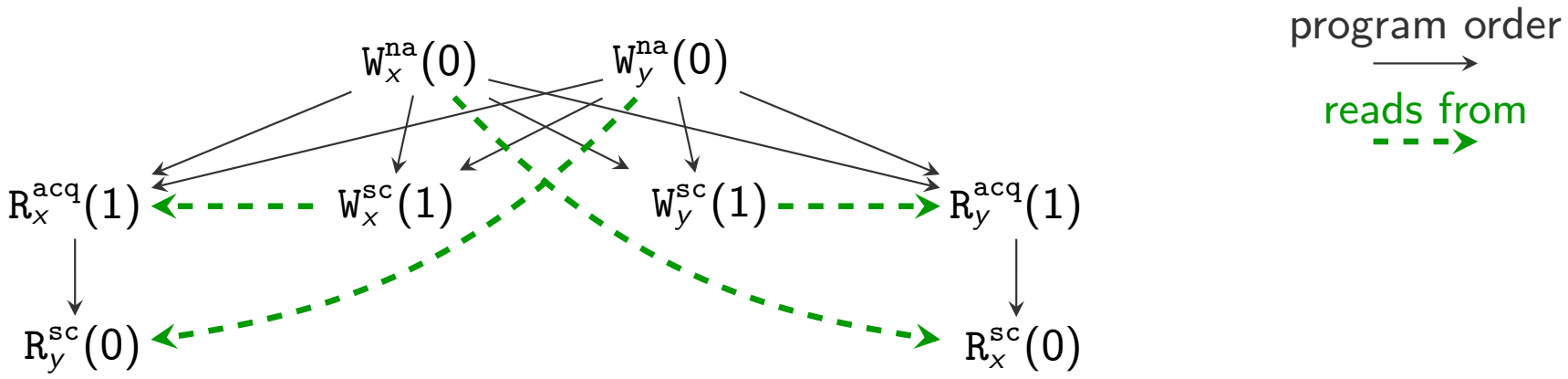
Stage 0: choose *reads-from*

Every read reads from a corresponding write.

C11's declarative semantics 101

Batty et al. [POPL'16]

$$a := x_{\mathtt{acq}}; \quad /\!/ 1 \;\Big\|\; b := y_{\mathtt{sc}}; \quad /\!/ 0 \;\Big\|\; x :=_{\mathtt{sc}} 1; \;\Big\|\; y :=_{\mathtt{sc}} 1; \;\Big\|\; c := y_{\mathtt{acq}}; \quad /\!/ 1 \;\Big\|\; d := x_{\mathtt{sc}}; \quad /\!/ 0$$



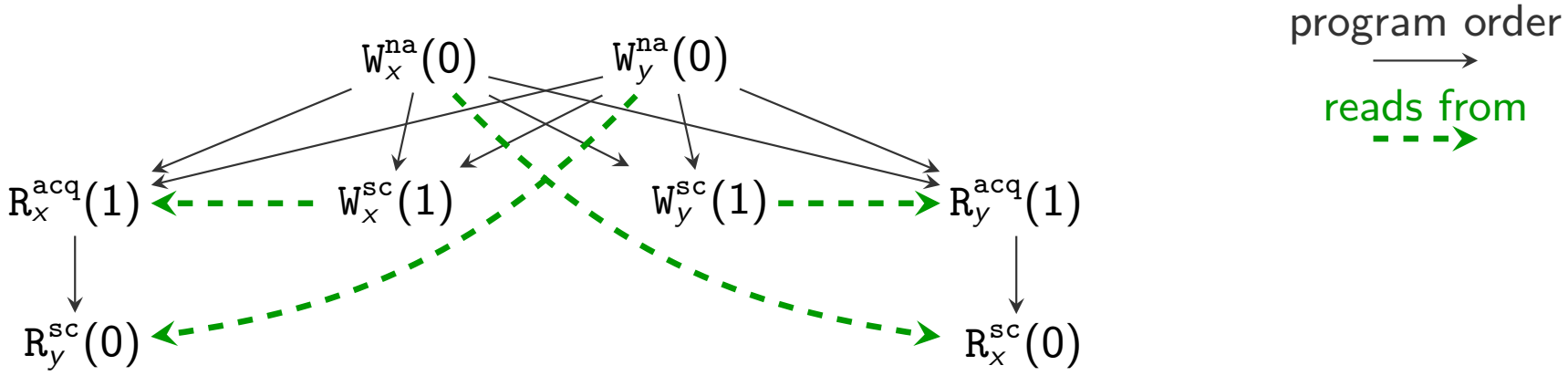
Stage 0: choose *reads-from*

Every read reads from a corresponding write.

C11's declarative semantics 101

Batty et al. [POPL'16]

$$a := x_{\mathtt{acq}}; \quad /\!/ 1 \qquad \Big\| \qquad x :=_{\mathtt{sc}} 1; \qquad \Big\| \qquad y :=_{\mathtt{sc}} 1; \qquad \Big\| \qquad c := y_{\mathtt{acq}}; \quad /\!/ 1$$
$$b := y_{\mathtt{sc}}; \quad /\!/ 0 \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad d := x_{\mathtt{sc}}; \quad /\!/ 0$$



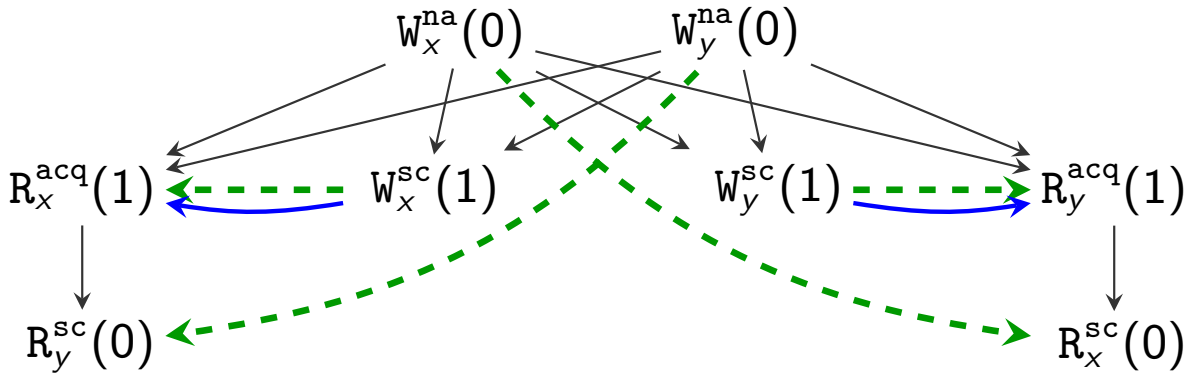
Stage 1: calculate *happens-before*

$$\frac{a \xrightarrow{\mathtt{po}} b}{a \xrightarrow{\mathtt{hb}} b} \qquad \frac{a : \mathtt{W}^{\sqsupseteq \mathtt{rel}} \xrightarrow{\mathtt{rf}} b : \mathtt{R}^{\sqsupseteq \mathtt{acq}}}{a \xrightarrow{\mathtt{hb}} b} \qquad \frac{a \xrightarrow{\mathtt{hb}} b \qquad b \xrightarrow{\mathtt{hb}} c}{a \xrightarrow{\mathtt{hb}} c}$$

C11's declarative semantics 101

Batty et al. [POPL'16]

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



program order

reads from

happens-before

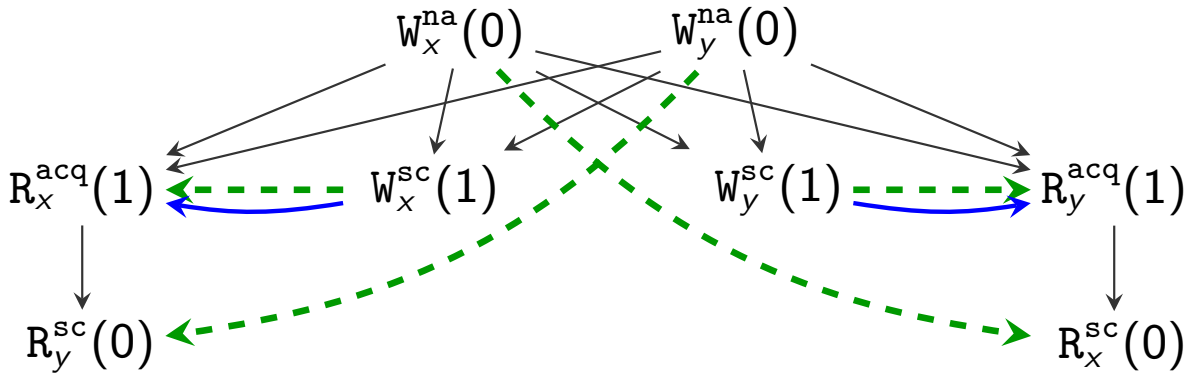
Stage 1: calculate *happens-before*

$$\frac{a \xrightarrow{\mathtt{po}} b}{a \xrightarrow{\mathtt{hb}} b} \qquad \frac{a : \mathtt{W}^{\sqsupseteq \mathtt{rel}} \xrightarrow{\mathtt{rf}} b : \mathtt{R}^{\sqsupseteq \mathtt{acq}}}{a \xrightarrow{\mathtt{hb}} b} \qquad \frac{a \xrightarrow{\mathtt{hb}} b \qquad b \xrightarrow{\mathtt{hb}} c}{a \xrightarrow{\mathtt{hb}} c}$$

C11's declarative semantics 101

Batty et al. [POPL'16]

$$
\begin{array}{l} a := x_{\mathtt{acq}}; \quad /\!/\, 1 \\ b := y_{\mathtt{sc}}; \quad /\!/\, 0 \end{array} \;\Big\|\; x :=_{\mathtt{sc}} 1; \;\Big\|\; y :=_{\mathtt{sc}} 1; \;\Big\|\; \begin{array}{l} c := y_{\mathtt{acq}}; \quad /\!/\, 1 \\ d := x_{\mathtt{sc}}; \quad /\!/\, 0 \end{array}
$$



program order

reads from

happens-before

Stage 2: "SC-per-location"

$W_x^{na}(0)$ $W_y^{na}(0)$

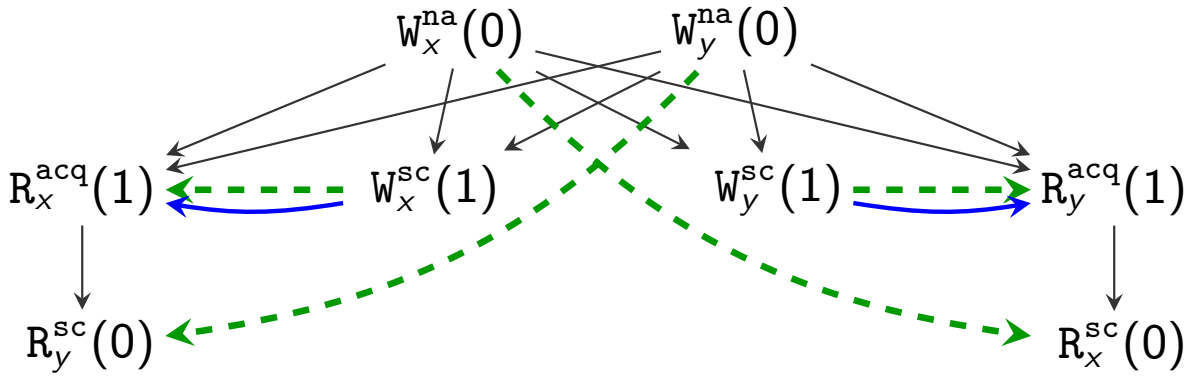
$R_x^{acq}(1)$ $W_x^{sc}(1)$ $W_y^{sc}(1)$ $R_y^{acq}(1)$

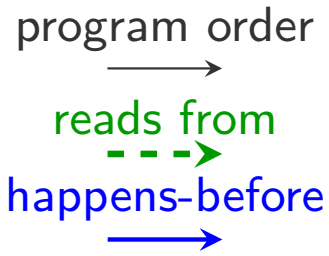
$R_y^{sc}(0)$ $R_x^{sc}(0)$

C11's declarative semantics 101

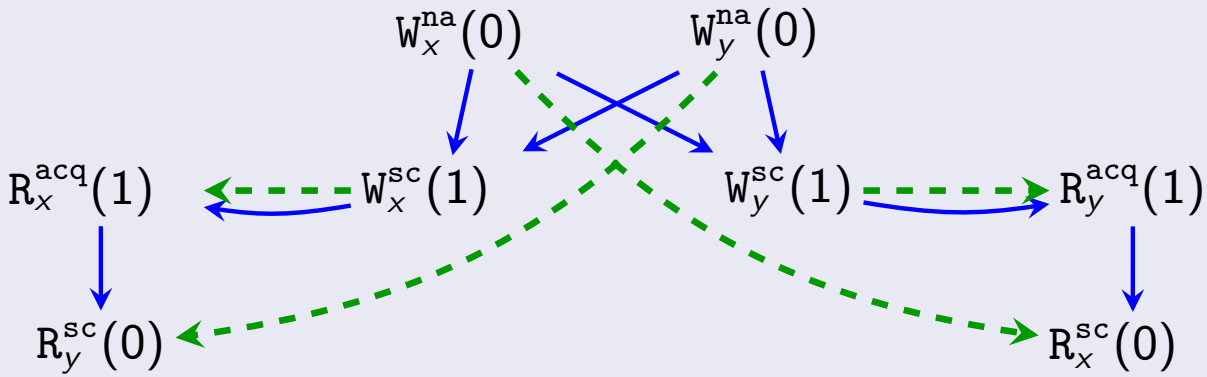
Batty et al. [POPL'16]

$$a := x_{\mathtt{acq}}; \quad /\!/ 1 \qquad b := y_{\mathtt{sc}}; \quad /\!/ 0 \;\Big\|\; x :=_{\mathtt{sc}} 1; \;\Big\|\; y :=_{\mathtt{sc}} 1; \;\Big\|\; c := y_{\mathtt{acq}}; \quad /\!/ 1 \qquad d := x_{\mathtt{sc}}; \quad /\!/ 0$$





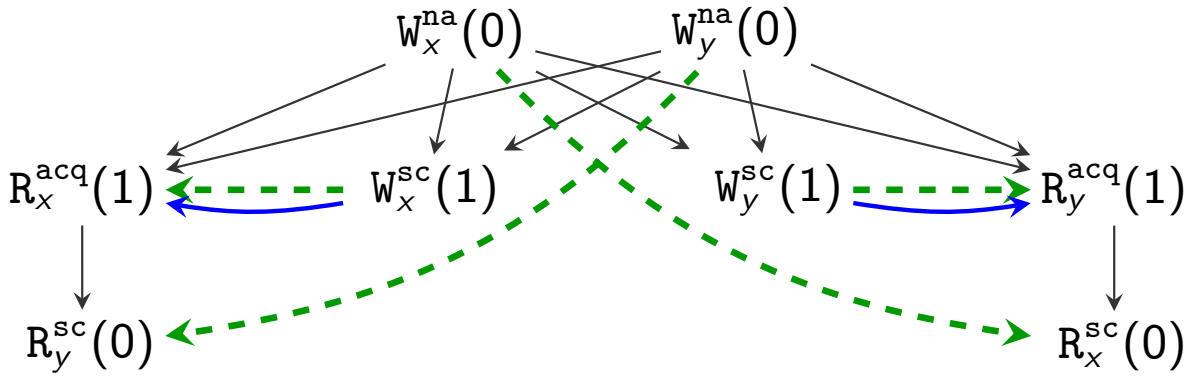
Stage 2: "SC-per-location"

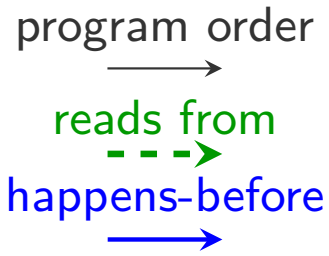


C11's declarative semantics 101

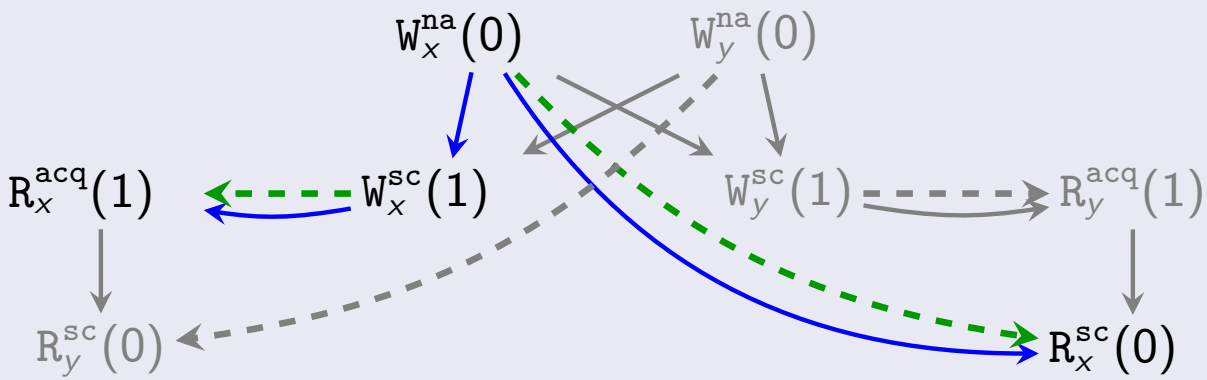
Batty et al. [POPL'16]

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





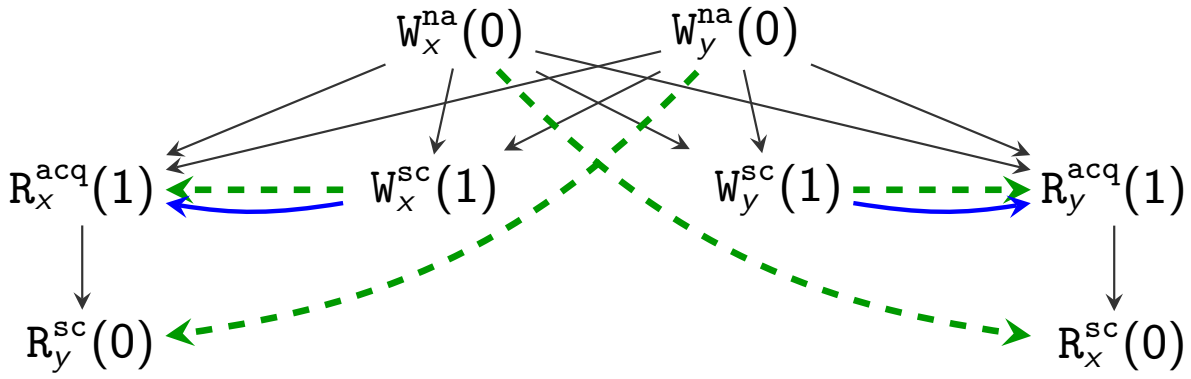
Stage 2: "SC-per-location"



C11's declarative semantics 101

Batty et al. [POPL'16]

$$
\begin{array}{l|l|l|l}
a := x_{\mathtt{acq}}; \quad /\!/ 1 & & & c := y_{\mathtt{acq}}; \quad /\!/ 1 \\
b := y_{\mathtt{sc}}; \quad /\!/ 0 & x :=_{\mathtt{sc}} 1; & y :=_{\mathtt{sc}} 1; & d := x_{\mathtt{sc}}; \quad /\!/ 0
\end{array}
$$

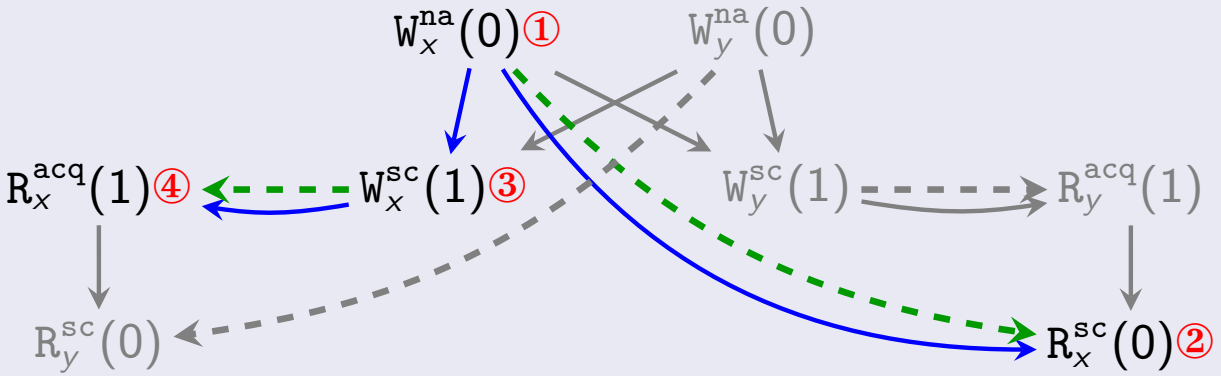


program order

reads from

happens-before

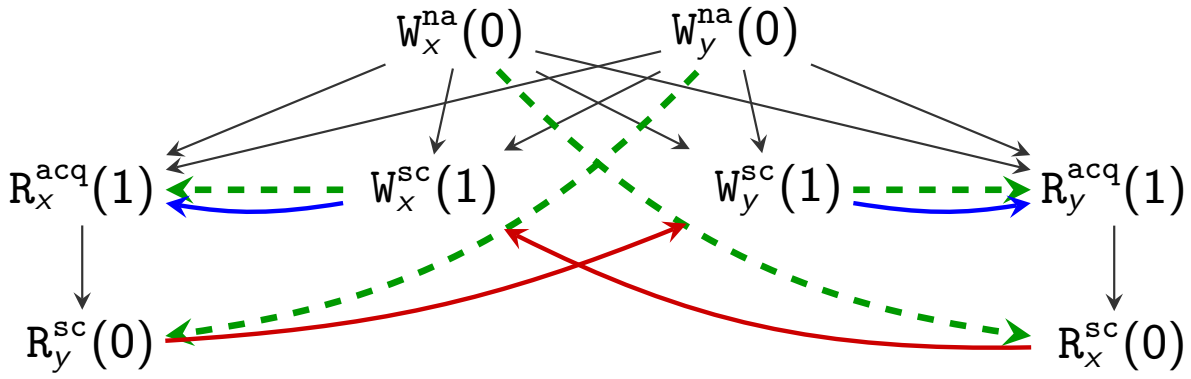
Stage 2: "SC-per-location"



C11's declarative semantics 101

Batty et al. [POPL'16]

$$
\begin{array}{l}
a := x_{\mathtt{acq}}; \quad /\!/\, 1 \\
b := y_{\mathtt{sc}}; \quad /\!/\, 0
\end{array}
\;\Big\|\; x :=_{\mathtt{sc}} 1; \;\Big\|\; y :=_{\mathtt{sc}} 1; \;\Big\|\;
\begin{array}{l}
c := y_{\mathtt{acq}}; \quad /\!/\, 1 \\
d := x_{\mathtt{sc}}; \quad /\!/\, 0
\end{array}
$$



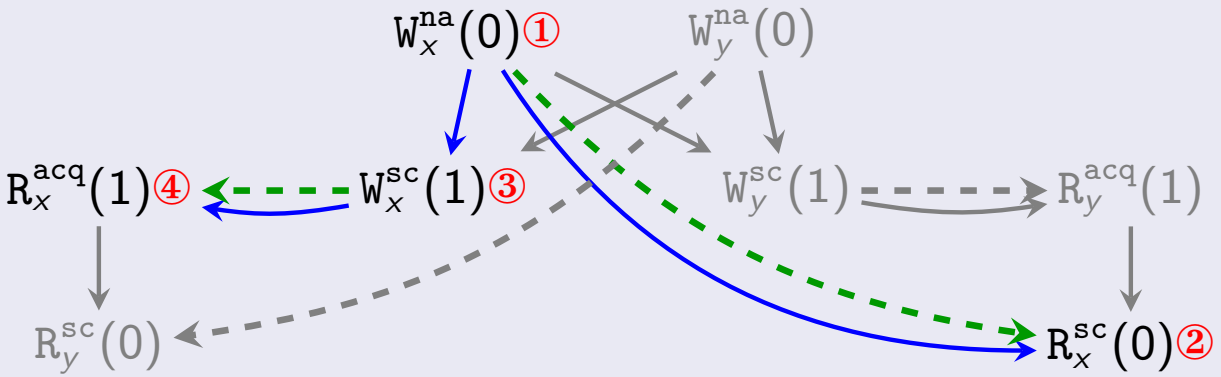
program order

reads from

happens-before

sc-per-loc

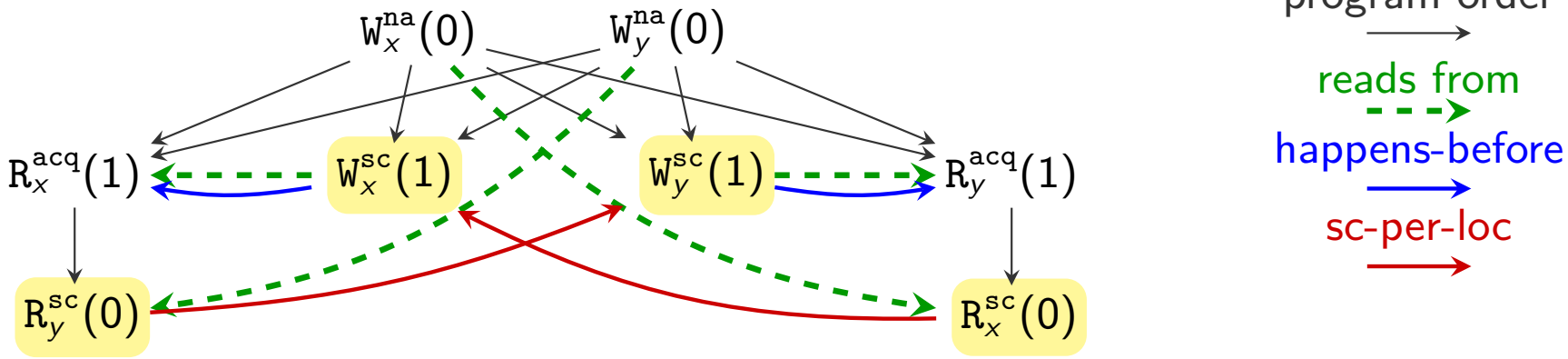
Stage 2: "SC-per-location"



C11's declarative semantics 101

Batty et al. [POPL'16]

$$a := x_{\mathtt{acq}}; \quad /\!/ 1 \qquad b := y_{\mathtt{sc}}; \quad /\!/ 0 \;\Big\|\; x :=_{\mathtt{sc}} 1; \;\Big\|\; y :=_{\mathtt{sc}} 1; \;\Big\|\; c := y_{\mathtt{acq}}; \quad /\!/ 1 \qquad d := x_{\mathtt{sc}}; \quad /\!/ 0$$



Stage 3: global restrictions on SC-accesses

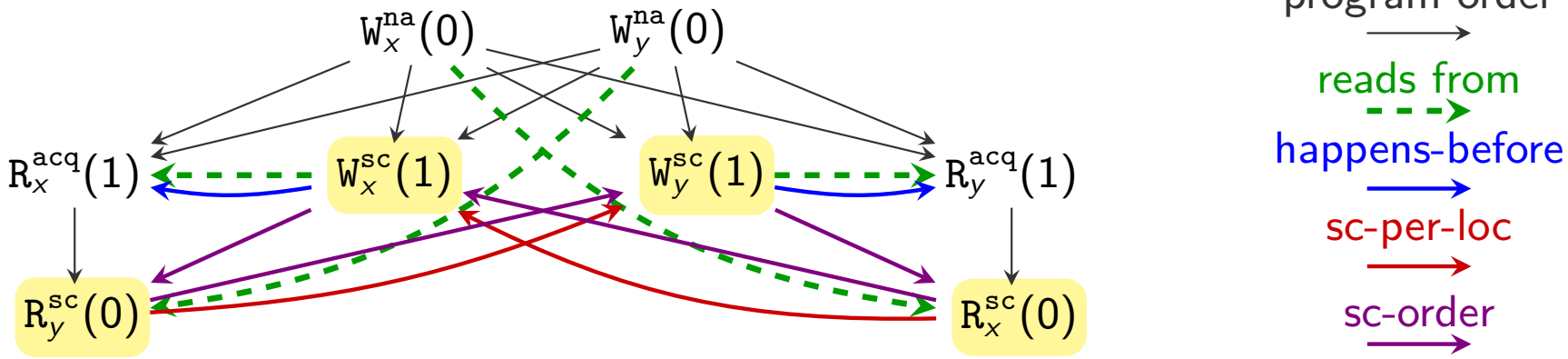
Order all SC-accesses while respecting:

$$\frac{a : *^{\mathtt{sc}} \xrightarrow{\mathtt{hb}} b : *^{\mathtt{sc}}}{a \xrightarrow{\text{sc-order}} b} \qquad \frac{a : *_x^{\mathtt{sc}} \xrightarrow{\text{sc-per-loc}} b : W_x^{\mathtt{sc}}}{a \xrightarrow{\text{sc-order}} b}$$

C11's declarative semantics 101

Batty et al. [POPL'16]

$$a := x_{\mathtt{acq}}; \quad /\!/ 1 \qquad b := y_{\mathtt{sc}}; \quad /\!/ 0 \;\Big\|\; x :=_{\mathtt{sc}} 1; \;\Big\|\; y :=_{\mathtt{sc}} 1; \;\Big\|\; c := y_{\mathtt{acq}}; \quad /\!/ 1 \qquad d := x_{\mathtt{sc}}; \quad /\!/ 0$$



Stage 3: global restrictions on SC-accesses

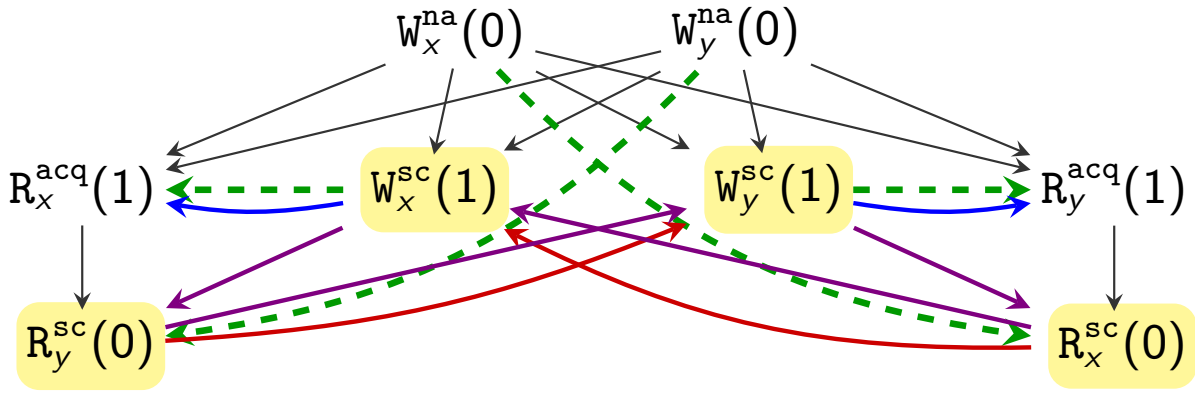
Order all SC-accesses while respecting:

$$\frac{a : *^{\mathtt{sc}} \xrightarrow{\mathtt{hb}} b : *^{\mathtt{sc}}}{a \xrightarrow{\text{sc-order}} b} \qquad \frac{a : *_x^{\mathtt{sc}} \xrightarrow{\text{sc-per-loc}} b : W_x^{\mathtt{sc}}}{a \xrightarrow{\text{sc-order}} b}$$

C11's declarative semantics 101

Batty et al. [POPL'16]

$$\begin{array}{l} a := x_{\mathtt{acq}}; \quad /\!/\, 1 \\ b := y_{\mathtt{sc}}; \quad /\!/\, 0 \end{array} \;\Big\|\; x :=_{\mathtt{sc}} 1; \;\Big\|\; y :=_{\mathtt{sc}} 1; \;\Big\|\; \begin{array}{l} c := y_{\mathtt{acq}}; \quad /\!/\, 1 \\ d := x_{\mathtt{sc}}; \quad /\!/\, 0 \end{array}$$



program order
reads from
happens-before
sc-per-loc
sc-order

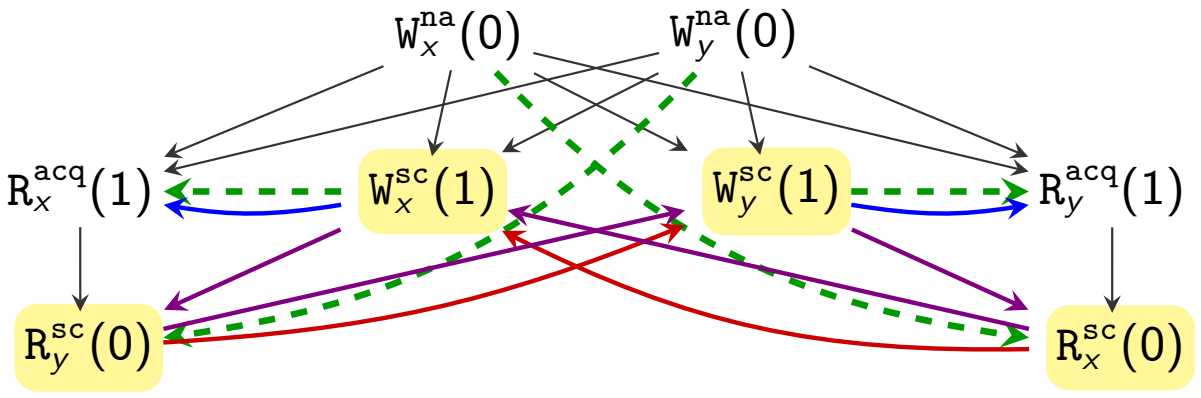
C/C++11: behavior disallowed

Stage 3: global restrictions on SC-accesses

Order all SC-accesses while respecting:

$$\frac{a : *^{\mathtt{sc}} \xrightarrow{\mathtt{hb}} b : *^{\mathtt{sc}}}{a \xrightarrow{\text{sc-order}} b} \qquad \frac{a : *^{\mathtt{sc}}_x \xrightarrow{\text{sc-per-loc}} b : W^{\mathtt{sc}}_x}{a \xrightarrow{\text{sc-order}} b}$$

What went wrong and how to fix it



C/C++11: behavior disallowed

program order

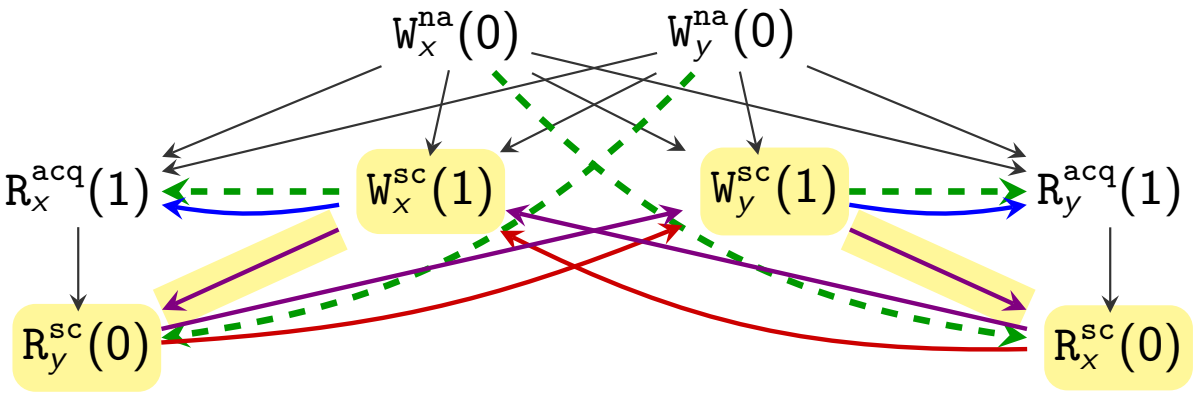
reads from

happens-before

sc-per-loc

sc-order

What went wrong and how to fix it



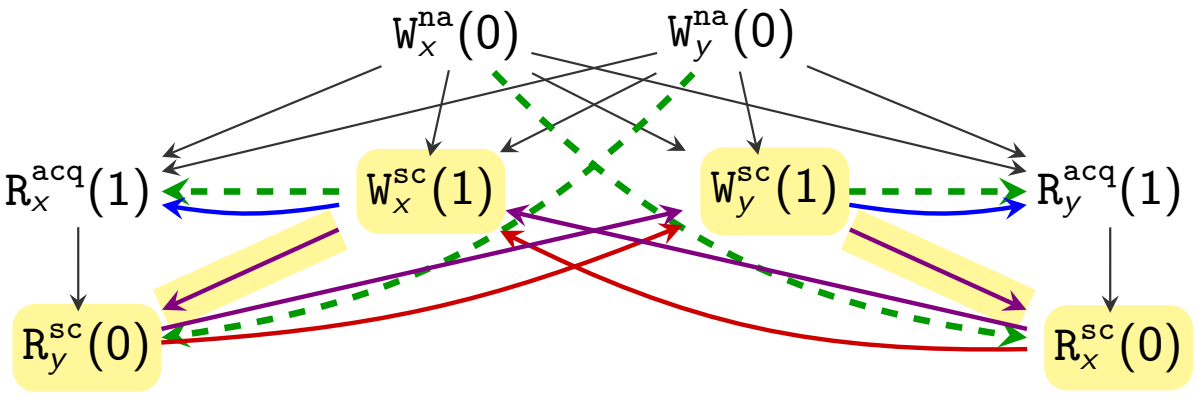
program order
reads from
happens-before
sc-per-loc
sc-order

C/C++11: behavior disallowed

$$\frac{a : *^{sc} \xrightarrow{hb} b : *^{sc}}{a \xrightarrow{sc\text{-}order} b}$$

- There are hb-paths between SC-accesses without sync fence in between.

What went wrong and how to fix it



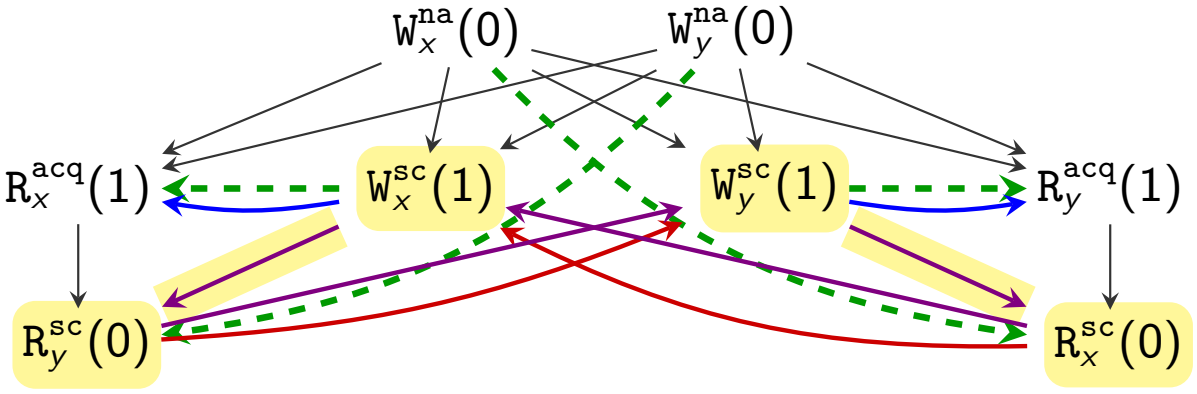
program order
reads from
happens-before
sc-per-loc
sc-order

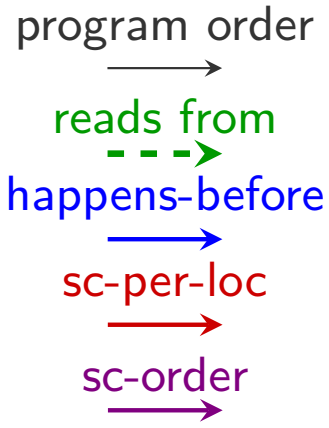
C/C++11: behavior disallowed

$$\frac{a : *^{sc} \xrightarrow{hb} b : *^{sc}}{a \xrightarrow{sc\text{-}order} b}$$

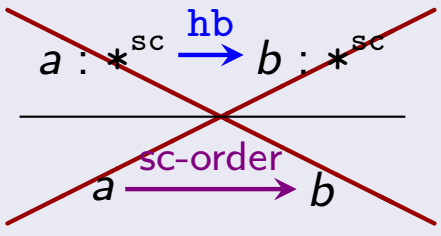
- There are hb-paths between SC-accesses without sync fence in between.
- Both compilation schemes ensure a sync fence on hb-paths between SC-accesses that start and end with "program order".

What went wrong and how to fix it



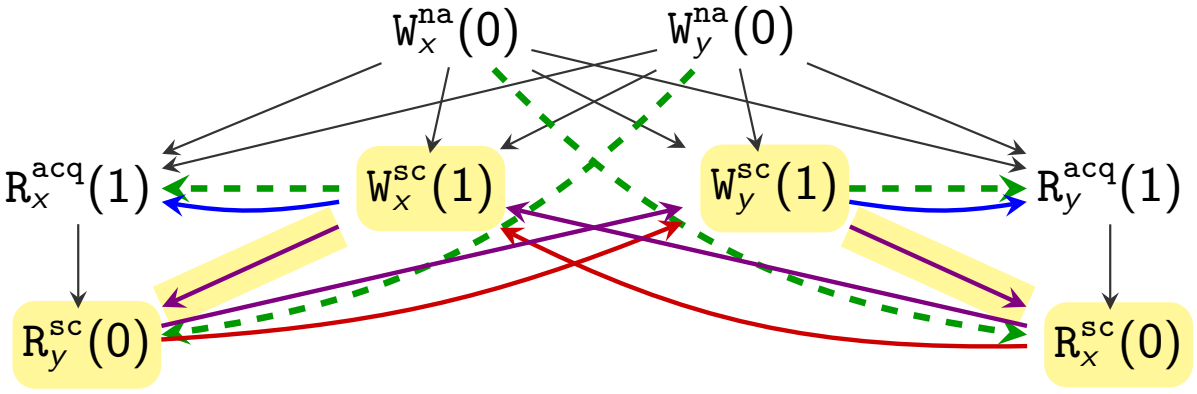


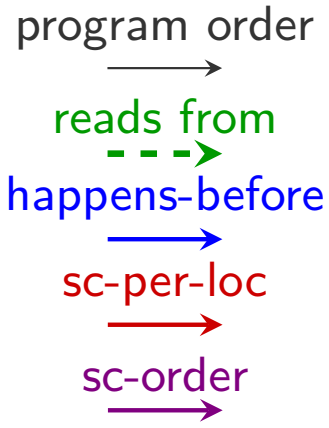
C/C++11: behavior disallowed



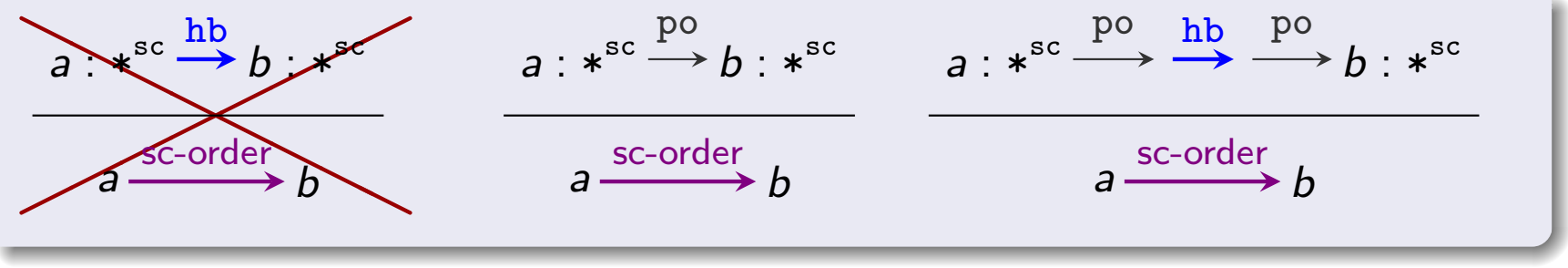
- There are hb-paths between SC-accesses without sync fence in between.
- Both compilation schemes ensure a sync fence on hb-paths between SC-accesses that start and end with "program order".

What went wrong and how to fix it



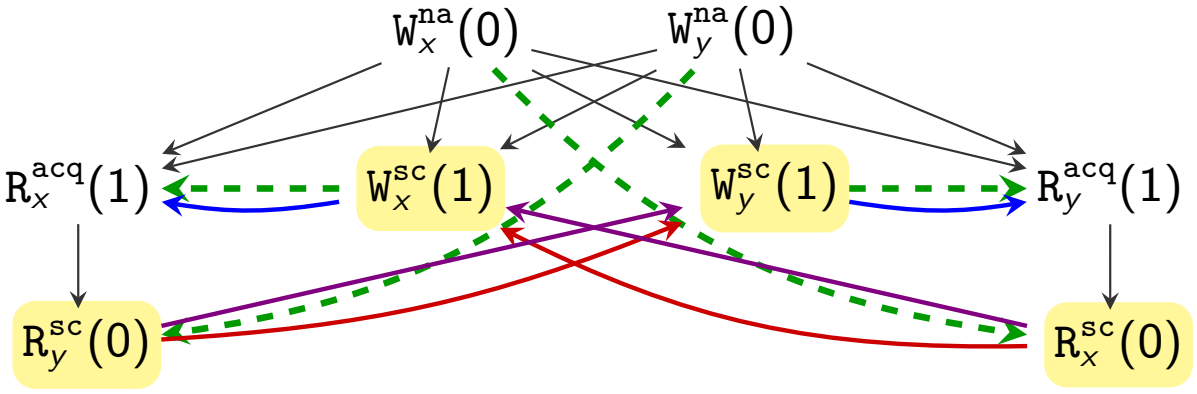


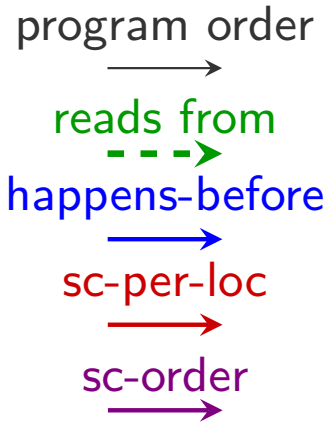
C/C++11: behavior disallowed



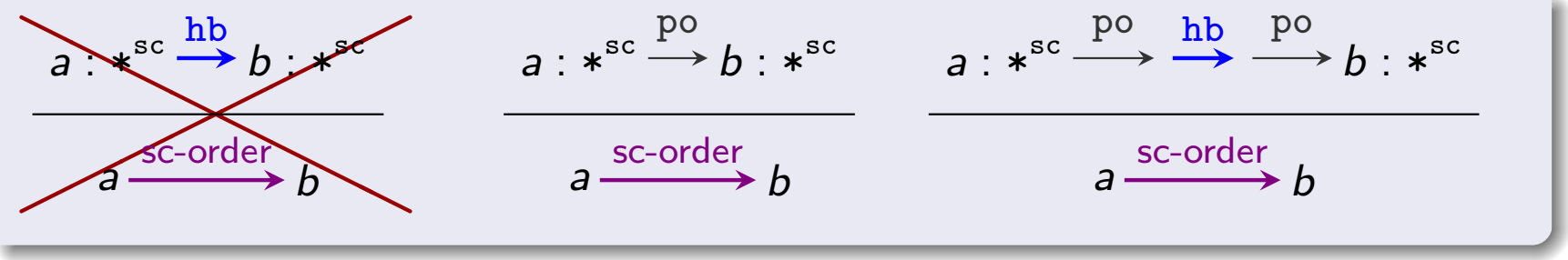
- There are hb-paths between SC-accesses without sync fence in between.
- Both compilation schemes ensure a sync fence on hb-paths between SC-accesses that start and end with "program order".

What went wrong and how to fix it



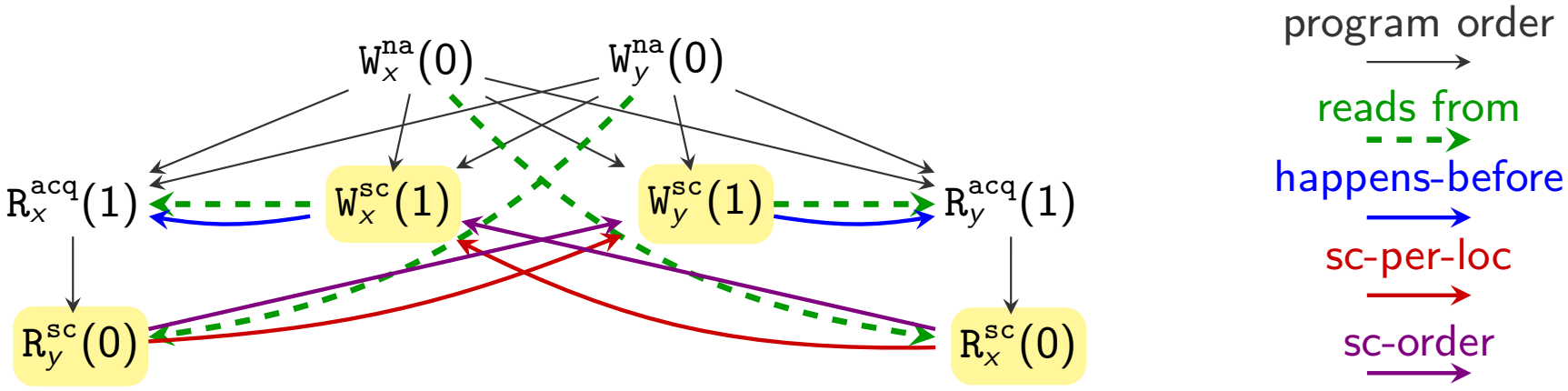


C/C++11: behavior disallowed

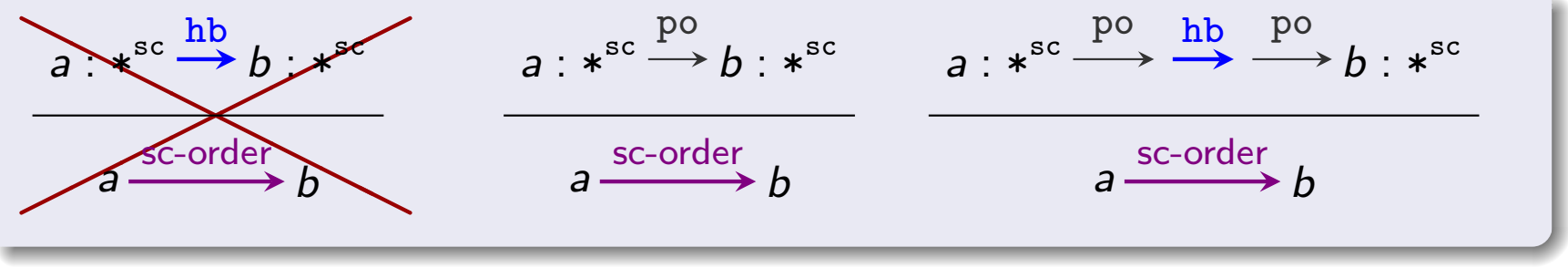


- There are hb-paths between SC-accesses without sync fence in between.
- Both compilation schemes ensure a sync fence on hb-paths between SC-accesses that start and end with "program order".

What went wrong and how to fix it



C/C++11: behavior disallowed Fixed model: behavior allowed



- There are hb-paths between SC-accesses without sync fence in between.
- Both compilation schemes ensure a sync fence on hb-paths between SC-accesses that start and end with "program order".

Results

The fixed model is not too strong:

- correctness of existing compilation schemes
  - Power/ARMv7 (Alglave et al. '14): leading/trailing `sync`
  - x86-TSO: `mfence` after-SC-writes/before-SC-reads
- soundness of compiler optimizations

The fixed model is not too weak:

- DRF theorem (without relaxed accesses)
- coincides with C11 when SC-accesses are to distinguished locations

SC-fences

Store buffer

$$\begin{array}{l|l} x := 1; & y := 1; \\ a := y; \quad /\!/\, 0 & b := x; \quad /\!/\, 0 \end{array}$$

How to guarantee only SC behaviors (*i.e.*, $a = 1 \vee b = 1$)?

$$\begin{array}{l|l} x :=_{\mathtt{sc}} 1; & y :=_{\mathtt{sc}} 1; \\ a := y_{\mathtt{sc}}; & b := x_{\mathtt{sc}}; \end{array} \quad \cong \quad \begin{array}{l|l} x :=_{\mathtt{rlx}} 1; & y :=_{\mathtt{rlx}} 1; \\ \mathbf{fence}_{\mathtt{sc}}; & \mathbf{fence}_{\mathtt{sc}}; \\ a := y_{\mathtt{rlx}}; & b := x_{\mathtt{rlx}}; \end{array}$$

Semantics of SC-fences is too weak!

- SC-fences, even when placed between every two accesses, do not restore SC.

Example

$$\begin{array}{l} a := x_{\texttt{rlx}};\ \#\,1 \\ \textbf{fence}_{\texttt{sc}}; \\ b := y_{\texttt{rlx}};\ \#\,0 \end{array} \;\Big\|\; x :=_{\texttt{rlx}} 1; \;\Big\|\; y :=_{\texttt{rlx}} 1; \;\Big\|\; \begin{array}{l} c := y_{\texttt{rlx}};\ \#\,1 \\ \textbf{fence}_{\texttt{sc}}; \\ d := x_{\texttt{rlx}};\ \#\,0 \end{array}$$

C/C++11: behavior allowed!

- Algorithm designers may have to unnecessarily strengthen access modes, leading to redundant hardware fences.
  - Chase-Lev concurrent deque [Lê et al. '13]: "unrecoverable overheads" in the interaction between atomic operations and memory barriers in C11.

Stronger semantics for SC fences

Global restrictions on SC-fences

Order all SC-fences while respecting:

$$\frac{a : \mathtt{F}^{\mathtt{sc}} \xrightarrow{\mathtt{hb}} b : \mathtt{F}^{\mathtt{sc}}}{a \xrightarrow{\text{sc-order}} b} \qquad \frac{a : \mathtt{F}^{\mathtt{sc}} \xrightarrow{\mathtt{hb}} *_x \xrightarrow{\text{sc-per-loc}} *_x \xrightarrow{\mathtt{hb}} b : \mathtt{F}^{\mathtt{sc}}}{a \xrightarrow{\text{sc-order}} b}$$

- We prove the correctness of existing compilation schemes and compiler optimizations for the strengthened model.
- SC-fences between every two accesses suffice to restore SC (assuming no data races on non-atomics).

Thin-air conservative solution

$$\texttt{non-atomic} \sqsubset \boxed{\texttt{relaxed}} \sqsubset \texttt{release/acquire} \sqsubset \texttt{sc}$$

The out-of-thin-air problem

Relaxed accesses are overly weak:

- Values appear out-of-thin-air
- DRF is broken

Load-buffering + data dependency

$a :=_{\texttt{rlx}} x$; // 1	$\parallel$	$b :=_{\texttt{rlx}} y$; // 1
$y :=_{\texttt{rlx}} a$;		$x :=_{\texttt{rlx}} b$;

Load-buffering + control dependency

$a :=_{\texttt{rlx}} x$; // 1	$\parallel$	$b :=_{\texttt{rlx}} y$; // 1
if $(a = 1)$		if $(b = 1)$
$y :=_{\texttt{rlx}} 1$;		$x :=_{\texttt{rlx}} 1$;

Thin-air conservative solution

$$\texttt{non-atomic} \sqsubset \boxed{\texttt{relaxed}} \sqsubset \texttt{release/acquire} \sqsubset \texttt{sc}$$

The out-of-thin-air problem

Relaxed accesses are overly weak:

- Values appear out-of-thin-air
- DRF is broken

Load-buffering + data dependency

$a := x_{\texttt{rlx}}$; // 1	$\parallel$	$b := y_{\texttt{rlx}}$; // 1
$y :=_{\texttt{rlx}} a$;		$x :=_{\texttt{rlx}} b$;

Load-buffering + control dependency

$a := x_{\texttt{rlx}}$; // 1	$\parallel$	$b := y_{\texttt{rlx}}$; // 1
if $(a = 1)$		if $(b = 1)$
$y :=_{\texttt{rlx}} 1$;		$x :=_{\texttt{rlx}} 1$;

Conservative solution [Boehm&Demsky '14]

- Require acyclicity of (program order $\cup$ reads-from)
- More expensive compilation:
  1. (fake) control dependency after relaxed reads
  2. or: (lightweight) fence before relaxed writes

Correctness of conservative solution

Conservative solution [Boehm&Demsky '14]

- Require acyclicity of (`program order` ∪ `reads-from`)
- More expensive compilation:
  1. (fake) control dependency after relaxed reads
  2. or: (lightweight) fence before relaxed writes

We proved correctness of compilation to Power/ARMv7 for scheme (1).

Main challenge

- Hardware models allow (`program order` ∪ `reads-from`) cycles (involving non-atomic reads in the source).
- We have to show that such cycles can be untangled to produce a racy consistent execution.

Summary

We presented **RC11**, a repaired model for C/C++11 concurrency:

- weaker semantics for SC-accesses
- stronger semantics for SC-fences
- disallow ($\texttt{program order} \cup \texttt{reads-from}$) cycles

We proved:

- correctness of compilation schemes
- soundness of compiler optimizations
- programming guarantees (DRF, SC-fences can restore SC)

Future Work

- Mechanize our proofs
- ARMv8

Summary

We presented **RC11**, a repaired model for C/C++11 concurrency:

- weaker semantics for SC-accesses
- stronger semantics for SC-fences
- disallow ($\texttt{program order} \cup \texttt{reads-from}$) cycles

We proved:

- correctness of compilation schemes
- soundness of compiler optimizations
- programming guarantees (DRF, SC-fences can restore SC)

Future Work

- Mechanize our proofs
- ARMv8

Thank you!

Correctness of compilation to different hardware

	C/C++11	Batty *et al.* [POPL'16]	RC11	Strong RC11	Strongest
x86-TSO	✓	✓	✓	✗	✗
POWER	✗	✗	✓	✓	✗
ARMv7 (no `isb`)	✓	✓	✓	✓	✓
ARMv7 (with `isb`)	✗	✗	✓	✓	✗
ARMv8 POP	✗	✗	✓ *	?	✗
ARMv8.2 (with `STLR`,`LDAR`)	✓ *	✓ *	✓ *	✓ *	✓ *

$$\mathtt{eco} \stackrel{\text{def}}{=} (\mathtt{rf} \cup \mathtt{mo} \cup \mathtt{rb})^{+}$$

$$\mathtt{pohbpo} \stackrel{\text{def}}{=} \mathtt{po}|_{\neq \mathtt{loc}}; \mathtt{hb}; \mathtt{po}|_{\neq \mathtt{loc}}$$

$$\text{RC11} \stackrel{\text{def}}{=} \text{acyclic}(([\mathtt{E}^{\mathtt{sc}}] \cup [\mathtt{F}^{\mathtt{sc}}]; \mathtt{hb}^{?}); (\mathtt{po} \cup \mathtt{pohbpo} \cup \mathtt{rf} \cup \mathtt{mo} \cup \mathtt{rb}); ([\mathtt{E}^{\mathtt{sc}}] \cup \mathtt{hb}^{?}; [\mathtt{F}^{\mathtt{sc}}]) \cup [\mathtt{F}^{\mathtt{sc}}]; \mathtt{hb}^{?}; (\mathtt{hb} \cup \mathtt{eco}); \mathtt{hb}^{?}; [\mathtt{F}^{\mathtt{sc}}])$$

$$\text{Strong-RC11} \stackrel{\text{def}}{=} \text{acyclic}(([\mathtt{E}^{\mathtt{sc}}] \cup [\mathtt{F}^{\mathtt{sc}}]; \mathtt{hb}^{?}); (\mathtt{po} \cup \mathtt{pohbpo} \cup \mathtt{eco}); ([\mathtt{E}^{\mathtt{sc}}] \cup \mathtt{hb}^{?}; [\mathtt{F}^{\mathtt{sc}}]) \cup [\mathtt{F}^{\mathtt{sc}}]; \mathtt{hb}^{?}; (\mathtt{hb} \cup \mathtt{eco}); \mathtt{hb}^{?}; [\mathtt{F}^{\mathtt{sc}}])$$

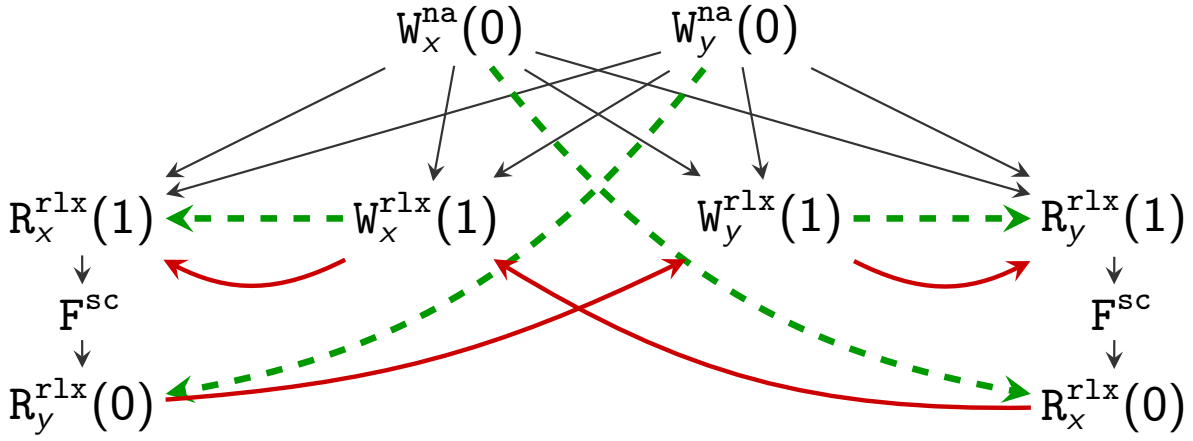
$$\text{Strongest} \stackrel{\text{def}}{=} \text{acyclic}([\mathtt{E}^{\mathtt{sc}}] \cup [\mathtt{F}^{\mathtt{sc}}]; \mathtt{hb}^{?}); (\mathtt{hb} \cup \mathtt{eco}); ([\mathtt{E}^{\mathtt{sc}}] \cup \mathtt{hb}^{?}; [\mathtt{F}^{\mathtt{sc}}])$$

Strengthening C11's declarative semantics for SC-fences

$$
\begin{array}{l}
a := x_{\texttt{rlx}};\ \ /\!/\ 1 \\
\textbf{fence}_{\texttt{sc}}; \\
b := y_{\texttt{rlx}};\ \ /\!/\ 0
\end{array}
\ \Big\|\ x :=_{\texttt{rlx}} 1;\ \Big\|\ y :=_{\texttt{rlx}} 1;\ \Big\|\
\begin{array}{l}
c := y_{\texttt{rlx}};\ \ /\!/\ 1 \\
\textbf{fence}_{\texttt{sc}}; \\
d := x_{\texttt{rlx}};\ \ /\!/\ 0
\end{array}
$$

Strengthening C11's declarative semantics for SC-fences

$$
\begin{array}{l}
a := x_{\mathtt{rlx}}; \ // \ 1 \\
\mathbf{fence}_{\mathtt{sc}}; \\
b := y_{\mathtt{rlx}}; \ // \ 0
\end{array}
\quad \Big\| \quad
x :=_{\mathtt{rlx}} 1; \quad \Big\| \quad
y :=_{\mathtt{rlx}} 1; \quad \Big\| \quad
\begin{array}{l}
c := y_{\mathtt{rlx}}; \ // \ 1 \\
\mathbf{fence}_{\mathtt{sc}}; \\
d := x_{\mathtt{rlx}}; \ // \ 0
\end{array}
$$



program order

reads from

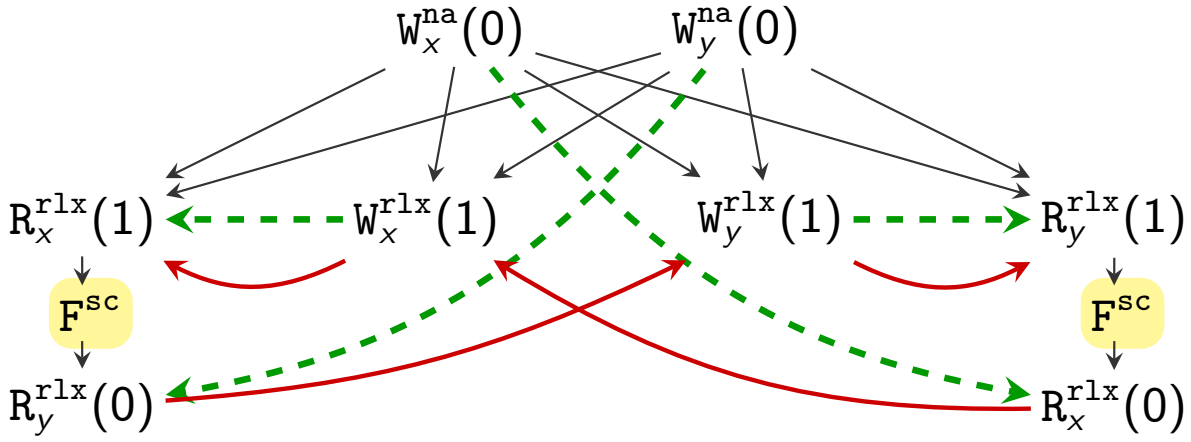
happens-before

sc-per-loc

sc-order

Strengthening C11's declarative semantics for SC-fences

$$a := x_{\texttt{rlx}};\ //\ 1 \\ \textbf{fence}_{\texttt{sc}}; \\ b := y_{\texttt{rlx}};\ //\ 0 \quad \Big\| \quad x :=_{\texttt{rlx}} 1; \quad \Big\| \quad y :=_{\texttt{rlx}} 1; \quad \Big\| \quad c := y_{\texttt{rlx}};\ //\ 1 \\ \textbf{fence}_{\texttt{sc}}; \\ d := x_{\texttt{rlx}};\ //\ 0$$



program order
reads from
happens-before
sc-per-loc
sc-order

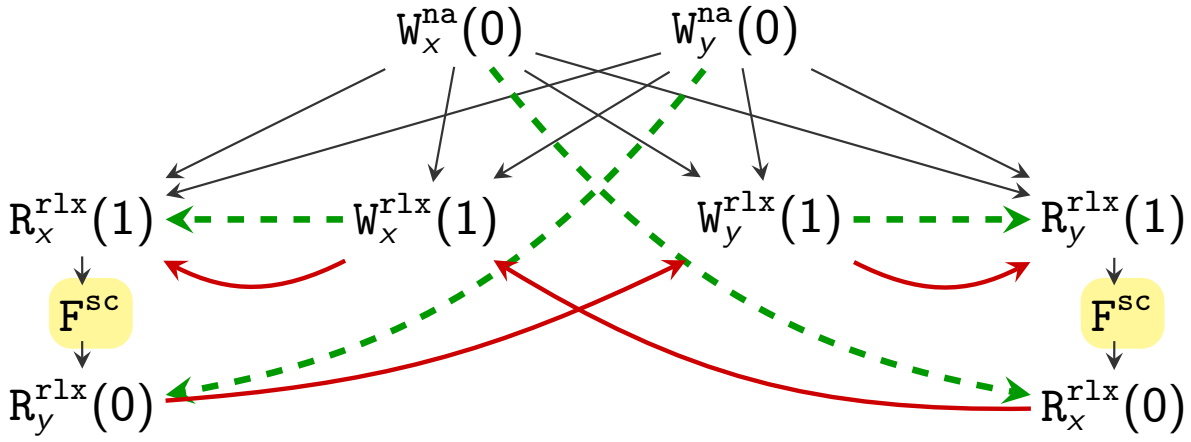
Global restrictions on SC-fences

Order all SC-fences while respecting:

$$\frac{a : \mathtt{F}^{\mathtt{sc}} \xrightarrow{\ \mathtt{hb}\ } b : \mathtt{F}^{\mathtt{sc}}}{a \xrightarrow{\ \mathtt{sc\text{-}order}\ } b} \qquad \frac{a : \mathtt{F}^{\mathtt{sc}} \xrightarrow{\ \mathtt{po}\ } *_x \xrightarrow{\ \mathtt{sc\text{-}per\text{-}loc}\ } \mathtt{W}_x \xrightarrow{\ \mathtt{po}\ } b : \mathtt{F}^{\mathtt{sc}}}{a \xrightarrow{\ \mathtt{sc\text{-}order}\ } b}$$

Strengthening C11's declarative semantics for SC-fences

$$a := x_{\texttt{rlx}};\ //\ 1 \quad \textbf{fence}_{\texttt{sc}};\quad b := y_{\texttt{rlx}};\ //\ 0 \ \Big\|\ x :=_{\texttt{rlx}} 1; \ \Big\|\ y :=_{\texttt{rlx}} 1; \ \Big\|\ c := y_{\texttt{rlx}};\ //\ 1 \quad \textbf{fence}_{\texttt{sc}};\quad d := x_{\texttt{rlx}};\ //\ 0$$



program order
reads from
happens-before
sc-per-loc
sc-order

C/C++11: behavior allowed!

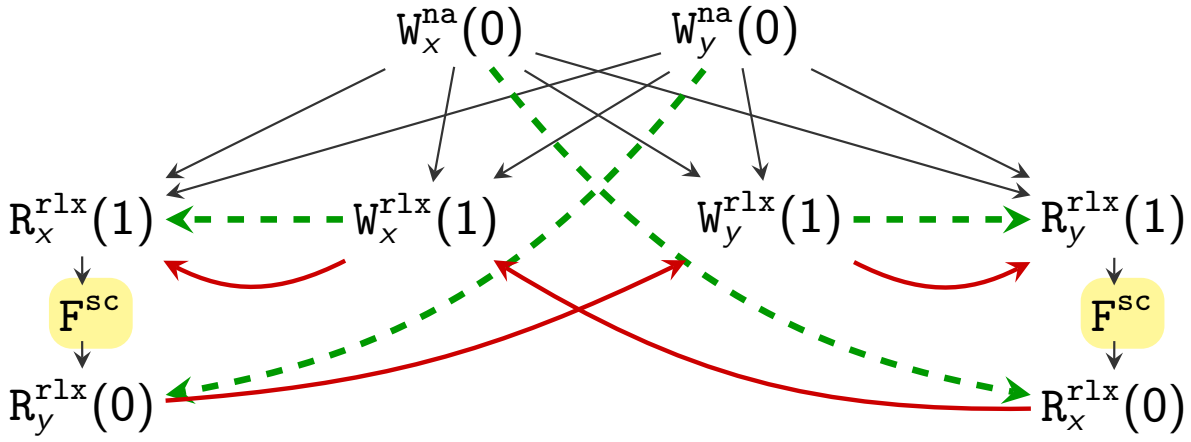
Global restrictions on SC-fences

Order all SC-fences while respecting:

$$\frac{a : \mathtt{F}^{\mathtt{sc}} \xrightarrow{\ \mathtt{hb}\ } b : \mathtt{F}^{\mathtt{sc}}}{a \xrightarrow{\ \text{sc-order}\ } b} \qquad \frac{a : \mathtt{F}^{\mathtt{sc}} \xrightarrow{\ \mathtt{po}\ } *_x \xrightarrow{\ \text{sc-per-loc}\ } \mathtt{W}_x \xrightarrow{\ \mathtt{po}\ } b : \mathtt{F}^{\mathtt{sc}}}{a \xrightarrow{\ \text{sc-order}\ } b}$$

Strengthening C11's declarative semantics for SC-fences

$a := x_{\texttt{rlx}}$; // 1
fence$_{\texttt{sc}}$;
$b := y_{\texttt{rlx}}$; // 0
	$x :=_{\texttt{rlx}} 1$;	$y :=_{\texttt{rlx}} 1$;	$c := y_{\texttt{rlx}}$; // 1
			fence$_{\texttt{sc}}$;
			$d := x_{\texttt{rlx}}$; // 0



program order
reads from
happens-before
sc-per-loc
sc-order

C/C++11: behavior allowed!

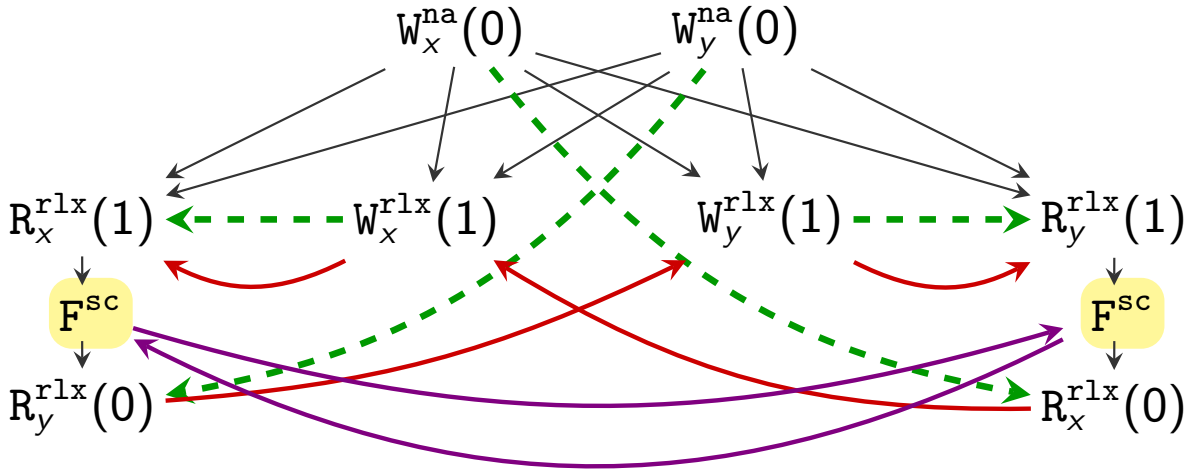
Global restrictions on SC-fences

Order all SC-fences while respecting:

$$\frac{a : \mathtt{F}^{\mathtt{sc}} \xrightarrow{\texttt{hb}} b : \mathtt{F}^{\mathtt{sc}}}{a \xrightarrow{\texttt{sc-order}} b} \qquad \frac{a : \mathtt{F}^{\mathtt{sc}} \xrightarrow{\texttt{po}} *_x \xrightarrow{\texttt{sc-per-loc}} \cancel{W_x} \xrightarrow{\texttt{po}} b : \mathtt{F}^{\mathtt{sc}}}{a \xrightarrow{\texttt{sc-order}} b}$$

Strengthening C11's declarative semantics for SC-fences

$$a := x_{\texttt{rlx}};\ //\ 1 \quad \textbf{fence}_{\texttt{sc}};\quad b := y_{\texttt{rlx}};\ //\ 0 \ \Big\|\ x :=_{\texttt{rlx}} 1; \ \Big\|\ y :=_{\texttt{rlx}} 1; \ \Big\|\ c := y_{\texttt{rlx}};\ //\ 1 \quad \textbf{fence}_{\texttt{sc}};\quad d := x_{\texttt{rlx}};\ //\ 0$$



program order
reads from
happens-before
sc-per-loc
sc-order

C/C++11: behavior allowed!

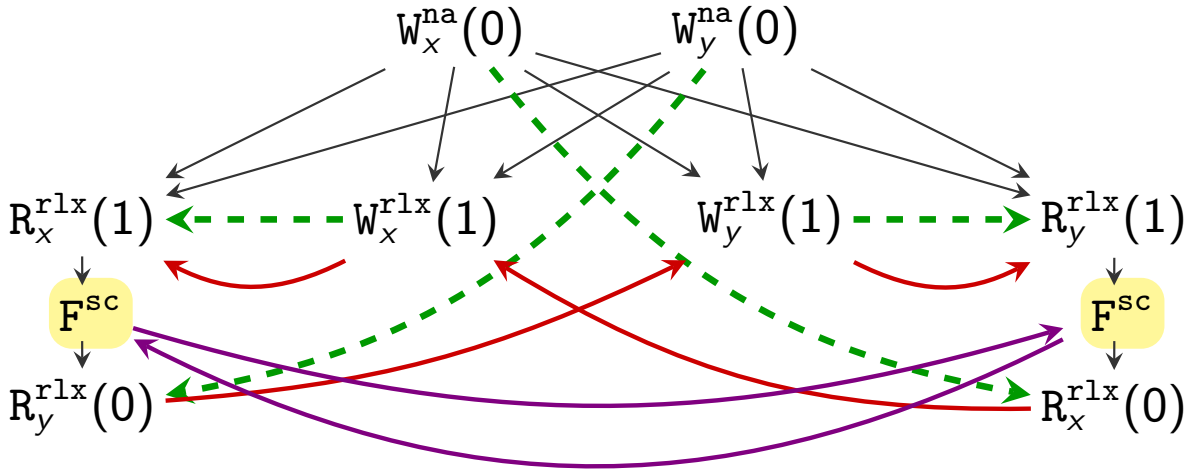
Global restrictions on SC-fences

Order all SC-fences while respecting:

$$\frac{a : \mathrm{F}^{\mathrm{sc}} \xrightarrow{\ \mathrm{hb}\ } b : \mathrm{F}^{\mathrm{sc}}}{a \xrightarrow{\ \text{sc-order}\ } b} \qquad \frac{a : \mathrm{F}^{\mathrm{sc}} \xrightarrow{\ \mathrm{po}\ } *_x \xrightarrow{\ \text{sc-per-loc}\ } W_x \xrightarrow{\ \mathrm{po}\ } b : \mathrm{F}^{\mathrm{sc}}}{a \xrightarrow{\ \text{sc-order}\ } b}$$

Strengthening C11's declarative semantics for SC-fences

$$a := x_{\texttt{rlx}};\ //\ 1 \quad \textbf{fence}_{\texttt{sc}};\quad b := y_{\texttt{rlx}};\ //\ 0 \ \Big\|\ x :=_{\texttt{rlx}} 1; \ \Big\|\ y :=_{\texttt{rlx}} 1; \ \Big\|\ c := y_{\texttt{rlx}};\ //\ 1 \quad \textbf{fence}_{\texttt{sc}};\quad d := x_{\texttt{rlx}};\ //\ 0$$



program order
reads from
happens-before
sc-per-loc
sc-order

C/C++11: behavior allowed!

Fixed model: behavior disallowed!

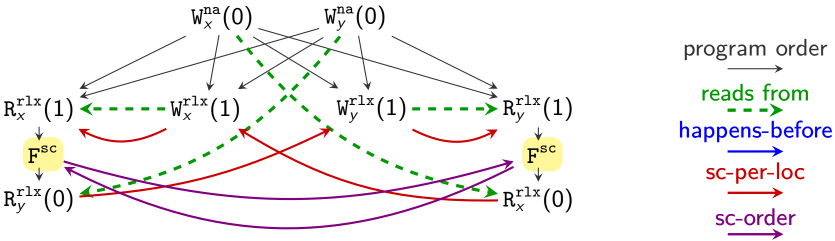
Global restrictions on SC-fences

Order all SC-fences while respecting:

$$\frac{a : \mathrm{F}^{\mathrm{sc}} \xrightarrow{\ \mathrm{hb}\ } b : \mathrm{F}^{\mathrm{sc}}}{a \xrightarrow{\ \text{sc-order}\ } b} \qquad \frac{a : \mathrm{F}^{\mathrm{sc}} \xrightarrow{\ \mathrm{po}\ } *_x \xrightarrow{\ \text{sc-per-loc}\ } W_x \xrightarrow{\ \mathrm{po}\ } b : \mathrm{F}^{\mathrm{sc}}}{a \xrightarrow{\ \text{sc-order}\ } b}$$

Strengthening C11's declarative semantics for SC-fences

$a := x_{\texttt{rlx}};$ // 1 **fence**$_{\texttt{sc}};$ $b := y_{\texttt{rlx}};$ // 0	$x :=_{\texttt{rlx}} 1;$	$y :=_{\texttt{rlx}} 1;$	$c := y_{\texttt{rlx}};$ // 1 **fence**$_{\texttt{sc}};$ $d := x_{\texttt{rlx}};$ // 0



C/C++11: behavior allowed!

Fixed model: behavior disallowed!

Global restrictions on SC-fences

Order all SC-fences while respecting: