Formal reasoning about the C11 weak memory model

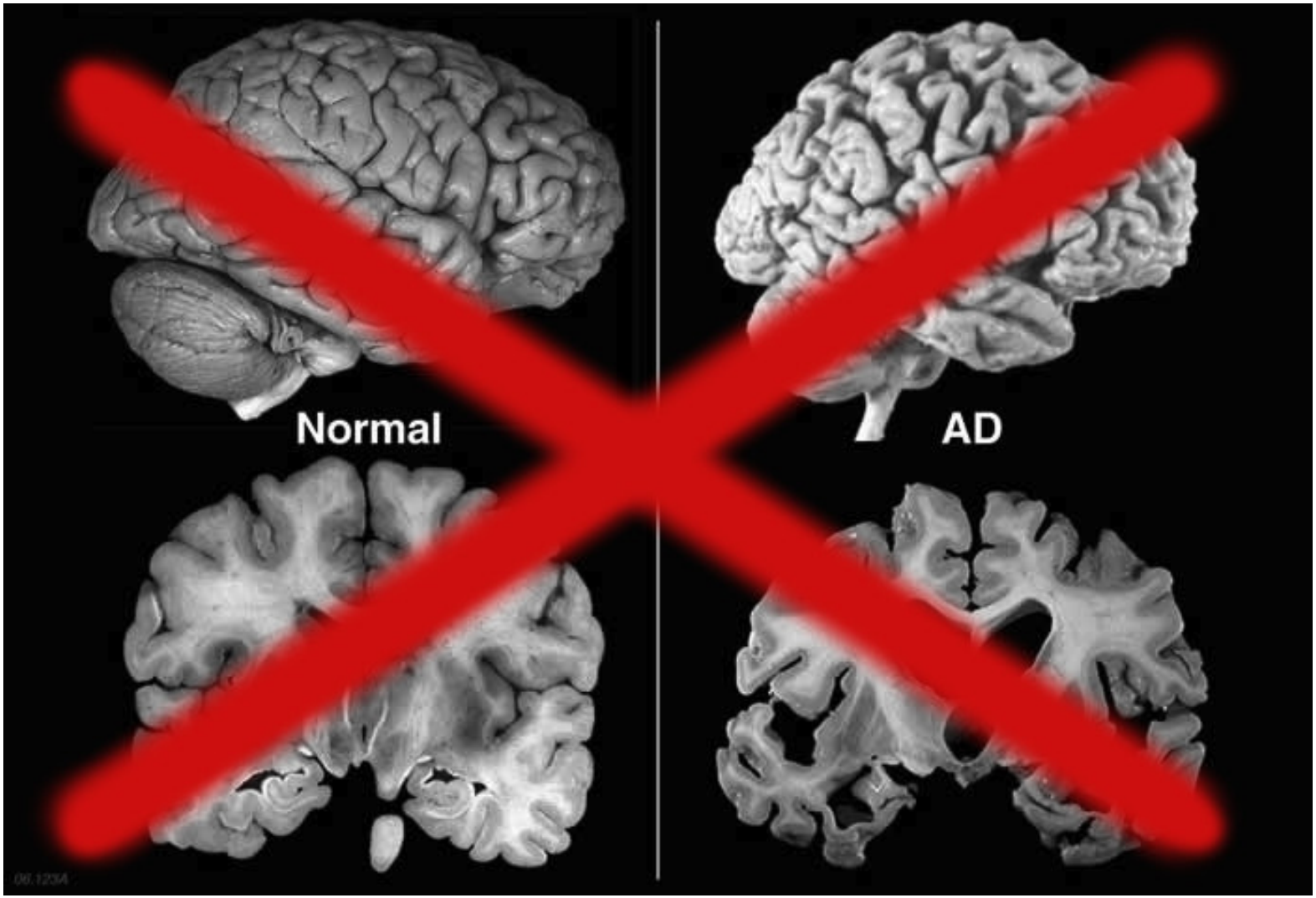
Invited talk @ CPP'15

Viktor Vafeiadis

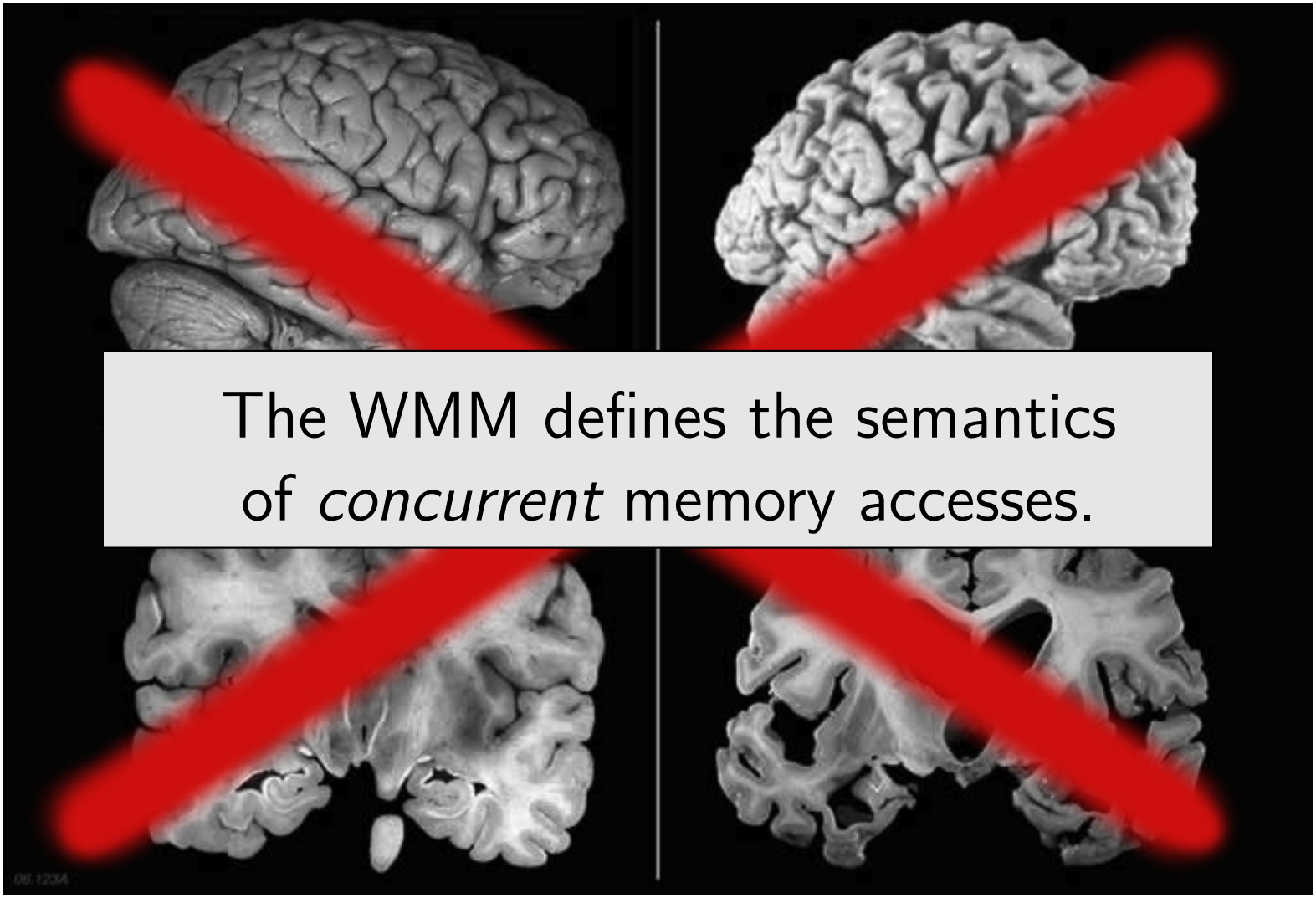
Max Planck Institute for Software Systems (MPI-SWS)

13 January 2015

What is a weak memory model?



What is a weak memory model?



The WMM defines the semantics of *concurrent* memory accesses.

Sequential consistency

Sequential consistency (SC):

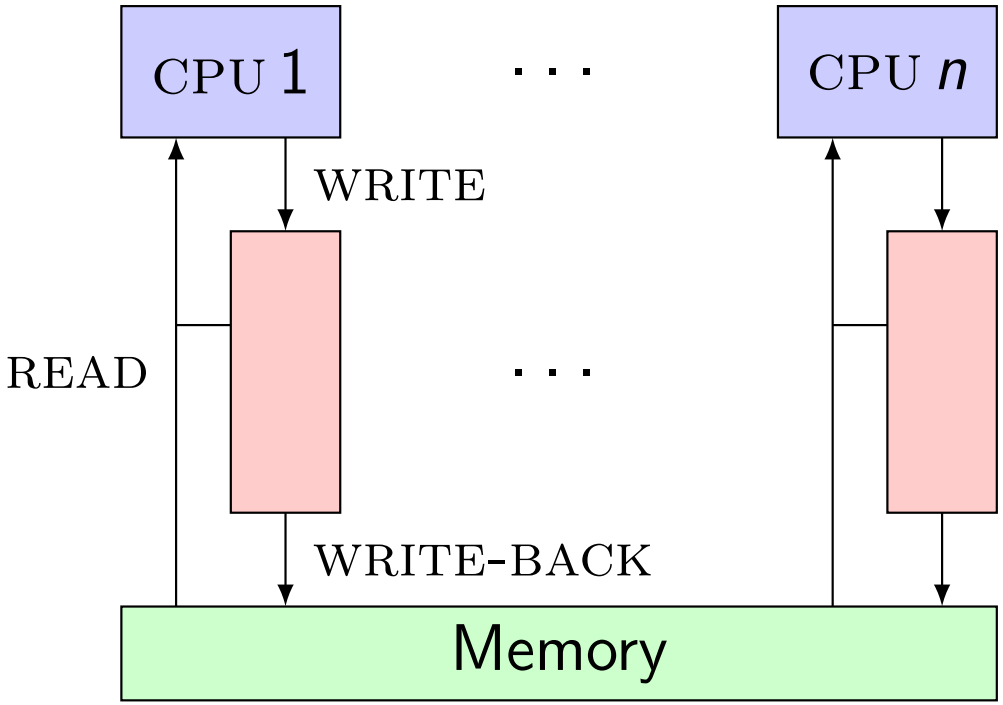
- The standard model for concurrency.
- Interleave each thread's atomic accesses.
- Almost all verification work assumes it.

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, $X = Y = 0$.

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

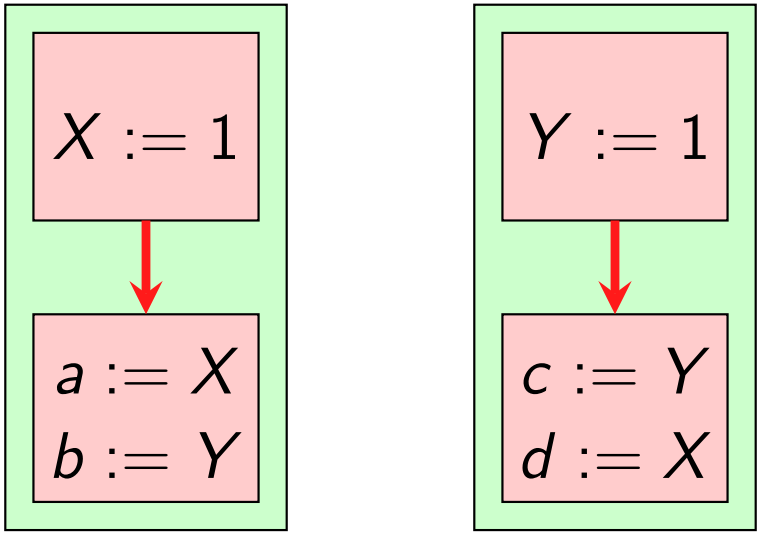
Allowed outcome: $a = b = 0$.

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

The C11 memory model

Two types of locations: ordinary and atomic

- Races on ordinary accesses $\rightsquigarrow$ error

A spectrum of atomic accesses:

- Seq. consistent $\rightsquigarrow$ full memory fence
- Release writes $\rightsquigarrow$ no fence (x86); lwsync (PPC)
- Acquire reads $\rightsquigarrow$ no fence (x86); isync (PPC)
- Consume reads $\rightsquigarrow$ no fence, but preserve deps
- Relaxed $\rightsquigarrow$ no fence

Explicit primitives for fences

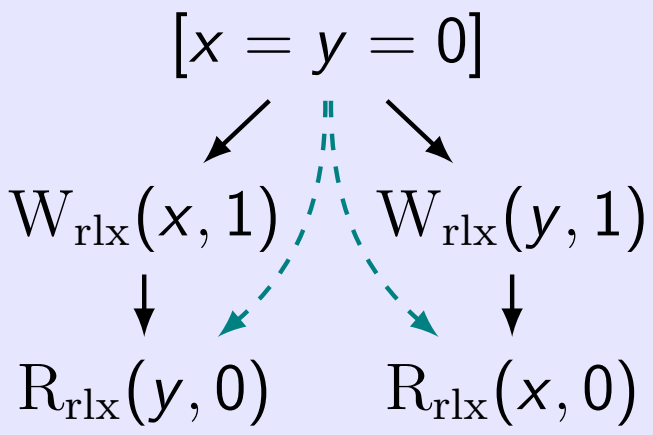
Relaxed behaviour: store buffering

Initially $x = y = 0$.

$$\begin{array}{l|l} x.\text{store}(1, \mathit{rlx}); & y.\text{store}(1, \mathit{rlx}); \\ t_1 = y.\text{load}(\mathit{rlx}); & t_2 = x.\text{load}(\mathit{rlx}); \end{array}$$

This can return $t_1 = t_2 = 0$.

Justification:



Behaviour observed on x86/Power/ARM

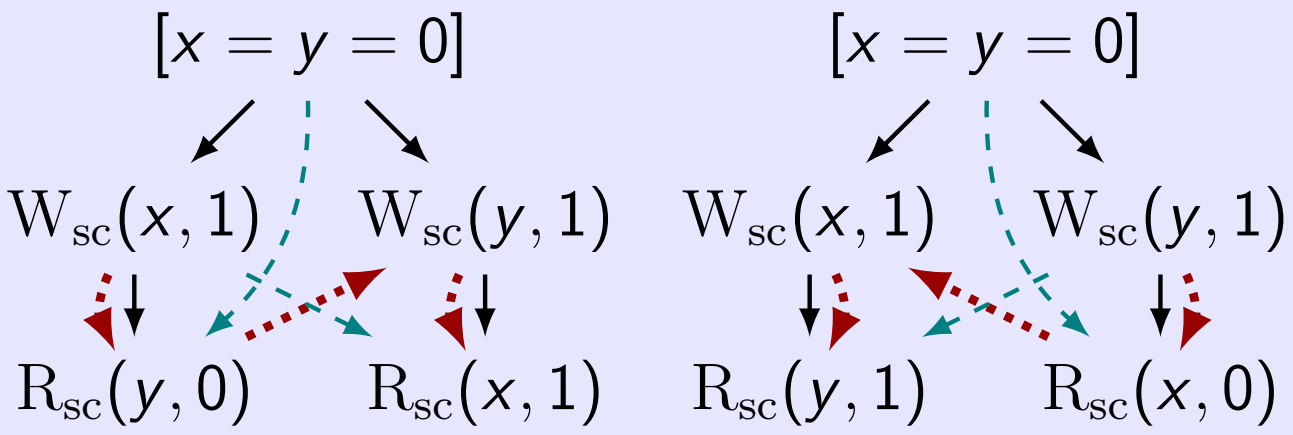
Getting rid of the SB behaviour

Initially $x = y = 0$.

$$\begin{array}{l|l} x.\mathsf{store}(1, sc); & y.\mathsf{store}(1, sc); \\ t_1 = y.\mathsf{load}(sc); & t_2 = x.\mathsf{load}(sc); \end{array}$$

This cannot return $t_1 = t_2 = 0$.

Justification:



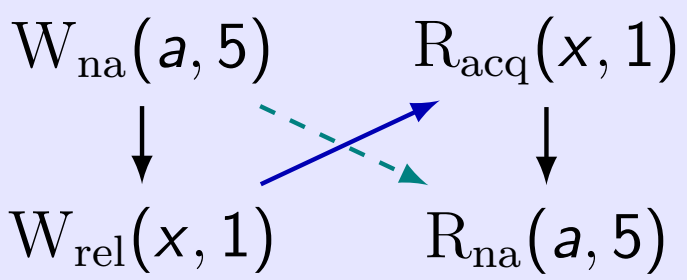
Release-acquire synchronization: message passing

Initially $a = x = 0$.

$a = 5;$	**while** $(x.\text{load}(acq) == 0);$
$x.\text{store}(1, \textit{release});$	$\text{print}(a);$

This will always print 5.

Justification:



Release-acquire synchronization

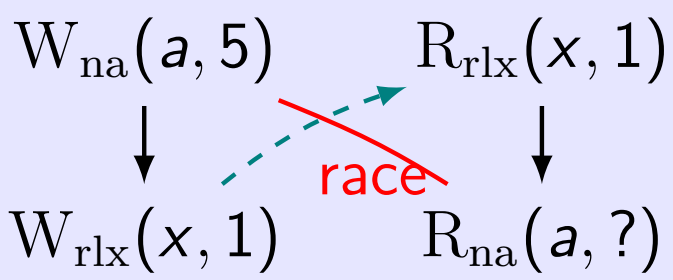
Relaxed accesses don't synchronize

Initially $a = x = 0$.

$a = 5;$	**while** $(x.\text{load}(rlx) == 0);$
$x.\text{store}(1, rlx);$	$\text{print}(a);$

The program is racy $\rightsquigarrow$ undefined semantics.

Justification:



Relaxed accesses don't synchronize

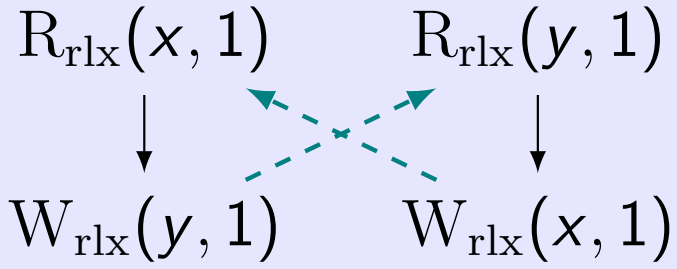
Dependency cycles

Initially $x = y = 0$.

if $(x.load(rlx) == 1)$	**if** $(y.load(rlx) == 1)$
$y.store(1, rlx);$	$x.store(1, rlx);$

C11 allows the outcome $x = y = 1$.

Justification:



Relaxed accesses don't synchronize

The C11 weak memory model (simplified)

$$\mathsf{isread}_{\ell,v}(a) \stackrel{\text{def}}{=} \exists X, v'.\ lab(a) \in \{\mathsf{R}_X(\ell, v), \mathsf{C}_X(\ell, v, v')\}$$
$$\mathsf{iswrite}_{\ell,v}(a) \stackrel{\text{def}}{=} \exists X, v'.\ lab(a) \in \{\mathsf{W}_X(\ell, v), \mathsf{C}_X(\ell, v', v)\}$$
$$\mathsf{isfence}(a) \stackrel{\text{def}}{=} lab(a) \in \{\mathsf{F}_{\text{ACQ}}, \mathsf{F}_{\text{REL}}\}$$
$$\mathsf{sameThread}(a, b) \stackrel{\text{def}}{=} \mathsf{tid}(a) = \mathsf{tid}(b)$$
$$\mathsf{rsElem}(a, b) \stackrel{\text{def}}{=} \mathsf{sameThread}(a, b) \vee \mathsf{isrmw}(b)$$

$$\mathsf{isread}_{\ell}(a) \stackrel{\text{def}}{=} \exists v.\ \mathsf{isread}_{\ell,v}(a)$$
$$\mathsf{iswrite}_{\ell}(a) \stackrel{\text{def}}{=} \exists v.\ \mathsf{iswrite}_{\ell,v}(a)$$
$$\mathsf{isaccess}(a) \stackrel{\text{def}}{=} \mathsf{isread}(a) \vee \mathsf{iswrite}(a)$$
$$\mathsf{isrmw}(a) \stackrel{\text{def}}{=} \mathsf{isread}(a) \wedge \mathsf{iswrite}(a)$$
$$\mathsf{isAcq}(a) \stackrel{\text{def}}{=} \mathsf{mode}(a) \sqsupseteq \text{ACQ}$$

$$\mathsf{isread}(a) \stackrel{\text{def}}{=} \exists \ell.\ \mathsf{isread}_{\ell}(a)$$
$$\mathsf{iswrite}(a) \stackrel{\text{def}}{=} \exists \ell.\ \mathsf{iswrite}_{\ell}(a)$$
$$\mathsf{isNA}(a) \stackrel{\text{def}}{=} \mathsf{mode}(a) = \text{NA}$$
$$\mathsf{isSC}(a) \stackrel{\text{def}}{=} \mathsf{mode}(a) = \text{SC}$$
$$\mathsf{isRel}(a) \stackrel{\text{def}}{=} \mathsf{mode}(a) \sqsupseteq \text{REL}$$

$$\mathsf{rseq}(a, b) \stackrel{\text{def}}{=} a = b \vee \mathsf{rsElem}(a, b) \wedge mo(a, b) \wedge (\forall c.\ mo(a, c) \wedge mo(c, b) \Rightarrow \mathsf{rsElem}(a, c))$$
$$\mathsf{sw}(a, b) \stackrel{\text{def}}{=} \exists c, d.\ \begin{array}{l} \neg\mathsf{sameThread}(a, b) \wedge \mathsf{isRel}(a) \wedge \mathsf{isAcq}(b) \wedge \mathsf{rseq}(c, rf(d)) \\ \wedge (a = c \vee \mathsf{isfence}(a) \wedge sb^+(a, c)) \wedge (d = b \vee \mathsf{isfence}(d) \wedge sb^+(d, b)) \end{array}$$
$$\mathsf{hb} \stackrel{\text{def}}{=} (sb \cup \mathsf{sw} \cup asw)^+$$
$$\mathsf{Racy} \stackrel{\text{def}}{=} \exists a, b.\ \begin{array}{l} \mathsf{isaccess}(a) \wedge \mathsf{isaccess}(b) \wedge \mathsf{loc}(a) = \mathsf{loc}(b) \wedge a \neq b \\ \wedge(\mathsf{iswrite}(a) \vee \mathsf{iswrite}(b)) \wedge (\mathsf{isNA}(a) \vee \mathsf{isNA}(b)) \wedge \neg(\mathsf{hb}(a, b) \vee \mathsf{hb}(b, a)) \end{array}$$
$$\mathsf{Observation} \stackrel{\text{def}}{=} \{(a, b) \mid mo(a, b) \wedge \mathsf{loc}(a) = \mathsf{loc}(b) = \texttt{world}\}$$

$$\forall a, b.\ sb(a, b) \implies \mathsf{tid}(a) = \mathsf{tid}(b) \quad \text{(ConsSB)}$$
$$\mathsf{order}(\mathsf{iswrite}, mo) \wedge \forall \ell.\ \mathsf{total}(\mathsf{iswrite}_{\ell}, mo) \quad \text{(ConsMO)}$$
$$\begin{array}{l} \mathsf{order}(\mathsf{isSC}, sc) \wedge \mathsf{total}(\mathsf{isSC}, sc) \\ \wedge (\mathsf{hb} \cup mo) \cap (\mathsf{isSC} \times \mathsf{isSC}) \subseteq sc \end{array} \quad \text{(ConsSC)}$$
$$\begin{array}{l} \forall b.\ (\exists c.\ rf(b) = c) \iff \\ \quad \exists \ell, a.\ \mathsf{iswrite}_{\ell}(a) \wedge \mathsf{isread}_{\ell}(b) \wedge \mathsf{hb}(a, b) \end{array} \quad \text{(ConsRFdom)}$$
$$\forall a, b.\ rf(b) = a \implies \exists \ell, v.\ \mathsf{iswrite}_{\ell,v}(a) \wedge \mathsf{isread}_{\ell,v}(b) \quad \text{(ConsRF)}$$
$$\forall a, b.\ rf(b) = a \wedge (\mathsf{isNA}(a) \vee \mathsf{isNA}(b)) \implies \mathsf{hb}(a, b) \quad \text{(ConsRFna)}$$
$$\begin{array}{l} \forall a, b.\ rf(b) = a \wedge \mathsf{isSC}(b) \implies \\ \mathsf{imm}(\mathsf{scr}, a, b) \vee \neg\mathsf{isSC}(a) \wedge \nexists x.\ \mathsf{hb}(a, x) \wedge \mathsf{imm}(\mathsf{scr}, x, b) \end{array} \quad \text{(SCReads)}$$

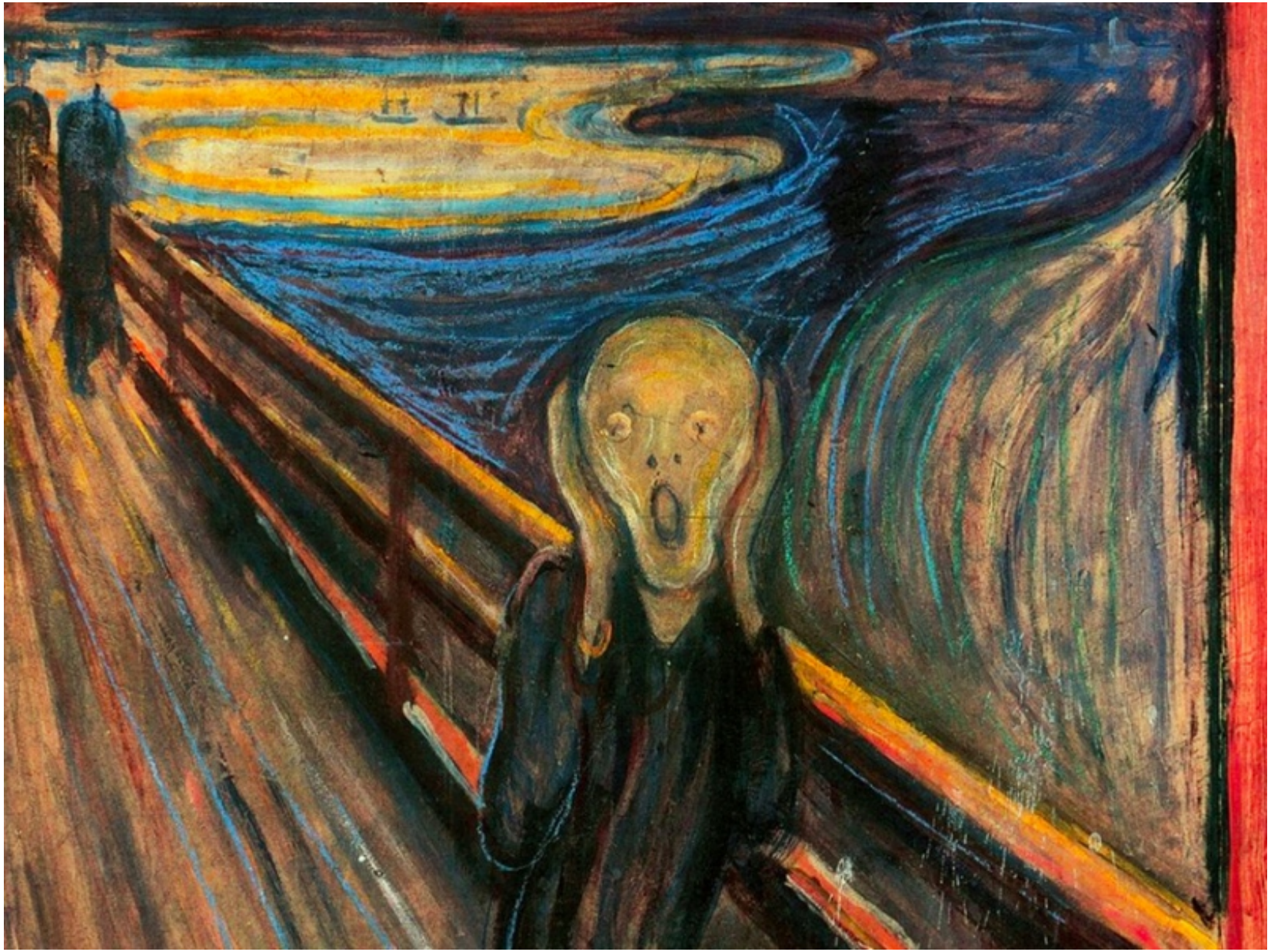
$$\nexists a.\ \mathsf{hb}(a, a) \quad \text{(IrrHB)}$$
$$\nexists a, b.\ rf(b) = a \wedge \mathsf{hb}(b, a) \quad \text{(ConsRFhb)}$$
$$\nexists a, b.\ \mathsf{hb}(a, b) \wedge mo(b, a) \quad \text{(CohWW)}$$
$$\nexists a, b.\ \mathsf{hb}(a, b) \wedge mo(rf(b), rf(a)) \quad \text{(CohRR)}$$
$$\nexists a, b.\ \mathsf{hb}(a, b) \wedge mo(rf(b), a) \quad \text{(CohWR)}$$
$$\nexists a, b.\ \mathsf{hb}(a, b) \wedge mo(b, rf(a)) \quad \text{(CohRW)}$$
$$\begin{array}{l} \forall a, b.\ \mathsf{isrmw}(a) \wedge rf(a) = b \implies \\ \quad \mathsf{imm}(mo, b, a) \end{array} \quad \text{(AtRMW)}$$
$$\forall a, b, \ell.\ lab(a) = lab(b) = \mathsf{A}(\ell) \implies a = b \quad \text{(ConsAlloc)}$$

where $\mathsf{order}(P, R) \stackrel{\text{def}}{=} (\nexists a.\ R(a, a)) \ \wedge\ (R^+ \subseteq R) \ \wedge\ (R \subseteq P \times P)$

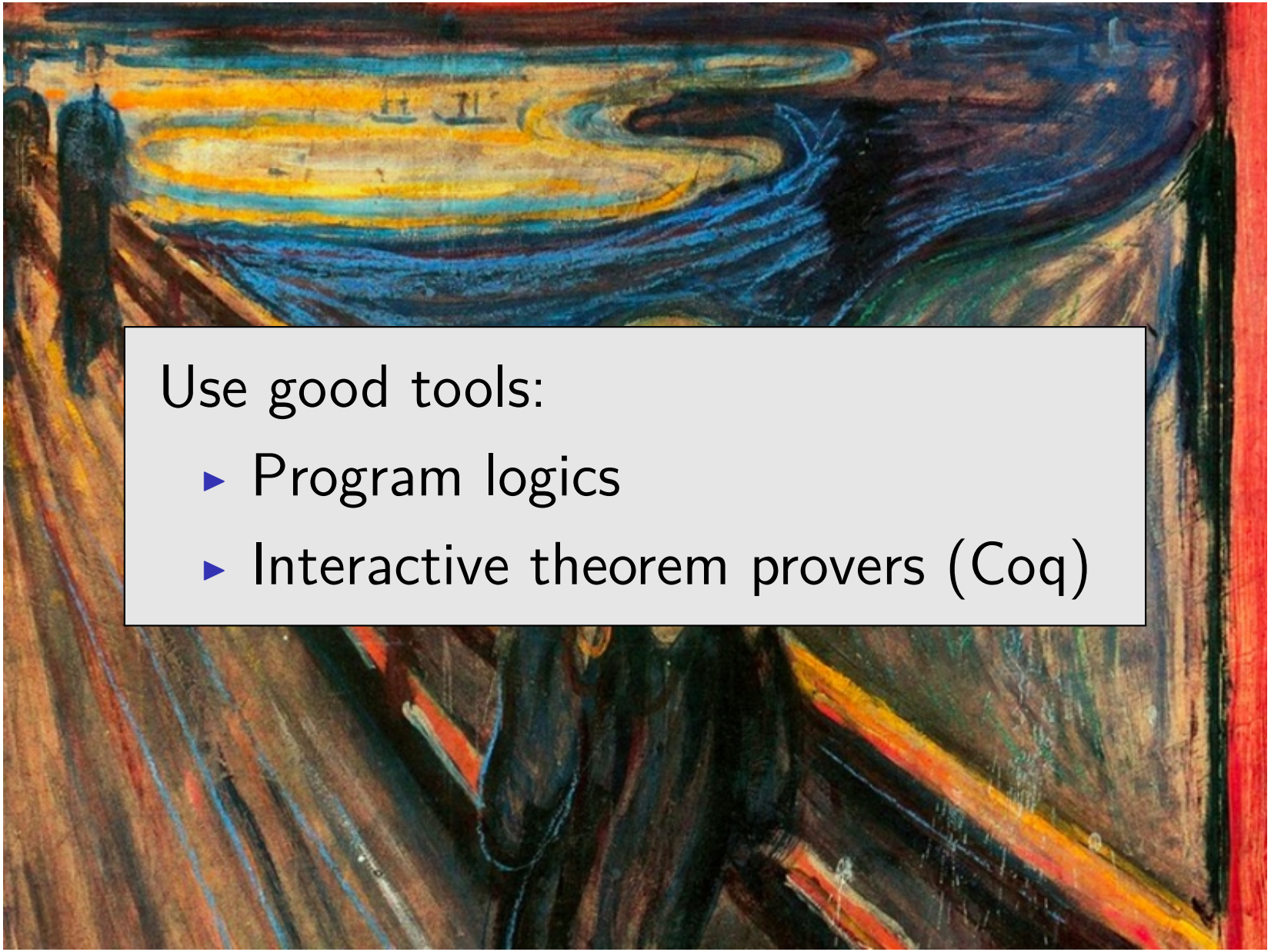
$$\mathsf{total}(P, R) \stackrel{\text{def}}{=} (\forall a, b.\ P(a) \wedge P(b) \implies a = b \vee R(a, b) \vee R(b, a))$$

$$\mathsf{imm}(R, a, b) \stackrel{\text{def}}{=} R(a, b) \wedge \nexists c.\ R(a, c) \wedge R(c, b)$$
$$\mathsf{scr}(a, b) \stackrel{\text{def}}{=} sc(a, b) \wedge \mathsf{iswrite}_{\mathsf{loc}(b)}(a)$$

The C11 weak memory model (simplified)



The C11 weak memory model (simplified)



Use good tools:

- Program logics
- Interactive theorem provers (Coq)

Two research directions

Verify compilation of C11:

- Compilation of the atomics to hardware (Batty et al.'11, Sarkar et al.'12)
- Source-to-source transformations (see POPL'15)
- An actual compiler (future work)

Verify concurrent C11 programs:

- **Using program logics**
- By reduction to SC (robustness)
- ~~*Don't verify, just find bugs.*~~

Understanding C11 using relaxed program logics

When should we care about relaxed memory?

C11 satisfies the DRF-SC property:

Theorem (DRF-SC)

If $[\![Prg]\!]_{\mathrm{SC}}$ *contains no data races and no weak atomics, then* $[\![Prg]\!]_{\mathrm{C11}} = [\![Prg]\!]_{\mathrm{SC}}$.

- Program logics that disallow data races are trivially sound for the NA+SC fragment of C11.

Separation logic

Key concept of *ownership* :

- Resourceful reading of Hoare triples.

$$\{P\}\ C\ \{Q\}$$

- Disjoint parallelism:

$$\frac{\{P_1\}\ C_1\ \{Q_1\} \qquad \{P_2\}\ C_2\ \{Q_2\}}{\{P_1 * P_2\}\ C_1 \| C_2\ \{Q_1 * Q_2\}}$$

Separation logic rules for non-atomic accesses

- Allocation gives you permission to access x.

$$\{\mathsf{emp}\}\ x = \mathsf{alloc}();\ \{x \mapsto \_\}$$

- To access a normal location, you must own it:

$$\{x \mapsto v\}\ t = *x;\ \{x \mapsto v \wedge t = v\}$$
$$\{x \mapsto v\}\ *x = v';\ \{x \mapsto v'\}$$

Reasoning about SC accesses

- Model SC accesses as non-atomic accesses inside a CCR.
- Use concurrent separation logic (CSL)

$$J \vdash \{P\}\ C\ \{Q\}$$

- Rule for SC-atomic reads:

$$\frac{emp \vdash \{J * P\}\ t = *x;\ \{J * Q\}}{J \vdash \{P\}\ t = x.load(sc);\ \{Q\}}$$

Rules for release/acquire accesses

Relaxed separation logic [OOPSLA'13]

Ownership transfer by rel-acq synchronizations.

- Atomic allocation $\rightsquigarrow$ pick loc. invariant $\mathcal{Q}$.

$$\left\{\mathcal{Q}(v)\right\}\ x = \mathsf{alloc}(v);\ \left\{\mathbf{W}_{\mathcal{Q}}(x) * \mathbf{R}_{\mathcal{Q}}(x)\right\}$$

- Release write $\rightsquigarrow$ give away permissions.

$$\left\{\mathcal{Q}(v) * \mathbf{W}_{\mathcal{Q}}(x)\right\}\ x.\mathsf{store}(v, \mathit{rel});\ \left\{\mathbf{W}_{\mathcal{Q}}(x)\right\}$$

- Acquire read $\rightsquigarrow$ gain permissions.

$$\left\{\mathbf{R}_{\mathcal{Q}}(x)\right\}\ t = x.\mathsf{load}(\mathit{acq});\ \left\{\mathcal{Q}(t) * \mathbf{R}_{\mathcal{Q}[t:=\mathsf{emp}]}(x)\right\}$$

Release-acquire synchronization: message passing

Initially $a = x = 0$. Let $J(v) \stackrel{\text{def}}{=} v = 0 \vee \&a \mapsto 5$.

$\{\&a \mapsto 0 * \mathbf{W}_J(x)\}$	$\{\mathbf{R}_J(x)\}$
$a = 5;$	**while** $(x.\text{load}(acq) == 0);$
$\{\&a \mapsto 5 * \mathbf{W}_J(x)\}$	$\{\&a \mapsto 5\}$
$x.\text{store}(release, 1);$	$\text{print}(a);$
$\{\mathbf{W}_J(x)\}$	$\{\&a \mapsto 5\}$

PL consequences:

Ownership transfer works!

Relaxed accesses

Basically, disallow ownership transfer.

- Relaxed reads:

$$\left\{\mathbf{R}_{\mathcal{Q}}(x)\right\}\ t := x.\mathsf{load}(rlx)\ \left\{\mathbf{R}_{\mathcal{Q}}(x)\right\}$$

- Relaxed writes:

$$\frac{\mathcal{Q}(v) = \mathsf{emp}}{\left\{\mathbf{W}_{\mathcal{Q}}(x)\right\}\ x.\mathsf{store}(v, rlx)\ \left\{\mathbf{W}_{\mathcal{Q}}(x)\right\}}$$

Unsound because of dependency cycles!

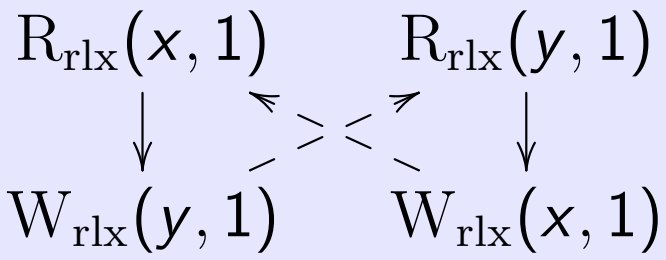
Dependency cycles

Initially $x = y = 0$.

$$\begin{array}{r|l} \textbf{if}\ (x.\mathit{load}(\mathit{rlx}) == 1) & \textbf{if}\ (y.\mathit{load}(\mathit{rlx}) == 1) \\ y.\mathit{store}(1, \mathit{rlx}); & x.\mathit{store}(1, \mathit{rlx}); \end{array}$$

C11 allows the outcome $x = y = 1$.

Justification:



Relaxed accesses don't synchronize

Dependency cycles

Initially $x = y = 0$.

$$\begin{array}{c|c} \textbf{if } (x.load(rlx) == 1) & \textbf{if } (y.load(rlx) == 1) \\ \quad y.store(1, rlx); & \quad x.store(1, rlx); \end{array}$$

C11 allows the outcome $x = y = 1$.

What goes wrong:

Non-relational invariants are unsound.

$$x = 0 \wedge y = 0$$

The DRF-property does not hold.

Dependency cycles

Initially $x = y = 0$.

$$\begin{array}{r||l} \textbf{if } (x.load(rlx) == 1) & \textbf{if } (y.load(rlx) == 1) \\ y.store(1, rlx); & x.store(1, rlx); \end{array}$$

C11 allows the outcome $x = y = 1$.

How to fix this:

Don't use relaxed writes

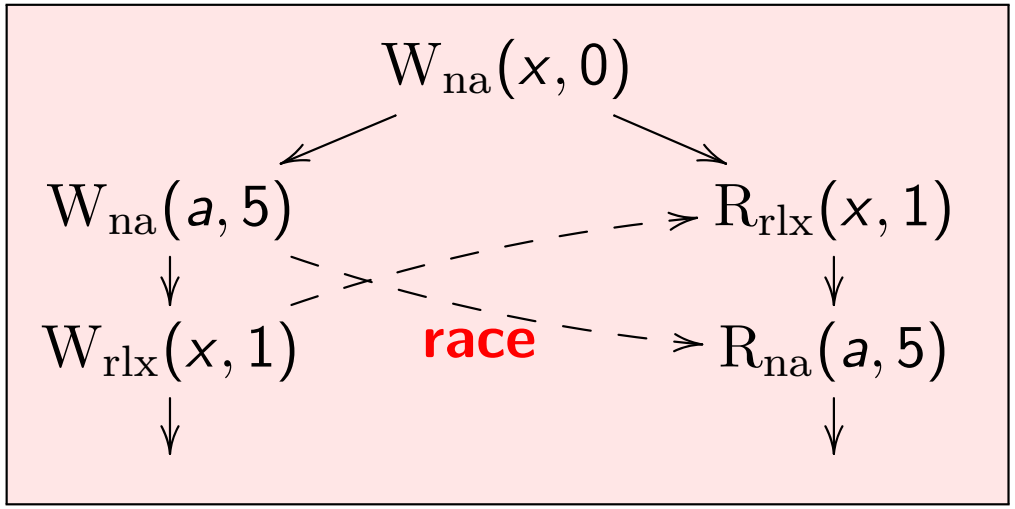
$\vee$

Strengthen the model

Incorrect message passing

$$\textbf{int } a;\ \textbf{atomic\_int } x = 0;$$

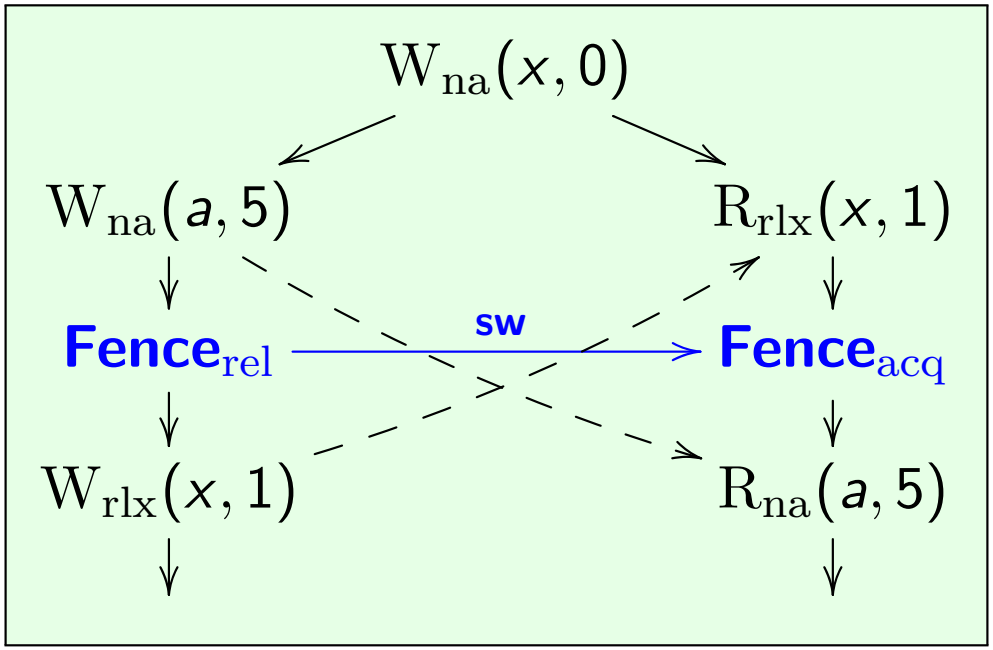
$$\left(\begin{array}{l} a = 5; \\ x.\mathsf{store}(1, rlx); \end{array} \middle\| \begin{array}{c} \textbf{if } (x.\mathsf{load}(rlx) \neq 0)\{ \\ \textbf{print}(a);\ \} \end{array} \right)$$



Message passing with C11 memory fences

$$\textbf{int } a;\ \textbf{atomic\_int } x = 0;$$

$$\left(\begin{array}{l|l} a = 5; & \textbf{if } (x.\mathsf{load}(rlx) \neq 0)\{ \\ \mathsf{fence}(release); & \quad \mathsf{fence}(acq); \\ x.\mathsf{store}(1, rlx); & \quad \textbf{print}(a);\ \} \end{array}\right)$$



Reasoning about fences

Joint work with Marko Doko. In progress.

- Introduce two 'modalities' in the logic

$$\{P\}\ \mathsf{fence}(\mathit{release})\ \{\triangle P\}$$

$$\{\nabla P\}\ \mathsf{fence}(\mathit{acq})\ \{P\}$$

$$\{\mathbf{R}_{\mathcal{Q}}(x)\}\ t := x.\mathsf{load}(\mathit{rlx})\ \{\mathbf{R}_{\mathcal{Q}[t:=\mathsf{emp}]}(x) * \nabla\mathcal{Q}(t)\}$$

$$\{\mathbf{W}_{\mathcal{Q}}(x) * \triangle\mathcal{Q}(v)\}\ x.\mathsf{store}(v, \mathit{rlx})\ \{\mathbf{W}_{\mathcal{Q}}(x)\}$$

Reasoning about fences

Let $\mathcal{Q}(v) \stackrel{\text{def}}{=} v = 0 \lor \&a \mapsto 5$.

$$\{\&a \mapsto 0 * \mathbf{W}_{\mathcal{Q}}(x) * \mathbf{R}_{\mathcal{Q}}(x)\}$$

$$\left(\begin{array}{l|l}
\{\&a \mapsto 0 * \mathbf{W}_{\mathcal{Q}}(x)\} & t = x.\mathrm{load}(rlx); \\
a = 5; & \{\nabla(t = 0 \lor \&a \mapsto 5)\} \\
\{\&a \mapsto 5 * \mathbf{W}_{\mathcal{Q}}(x)\} & \textbf{if } (t \neq 0) \\
\mathrm{fence}(release); & \quad \mathrm{fence}(acq); \\
\{\triangle(\&a \mapsto 5) * \mathbf{W}_{\mathcal{Q}}(x)\} & \quad \{\&a \mapsto 5\} \\
x.\mathrm{store}(1, rlx); & \quad \textbf{print}(a);\ \} \\
\{\mathrm{true}\} & \{\mathrm{true}\}
\end{array}\right)$$

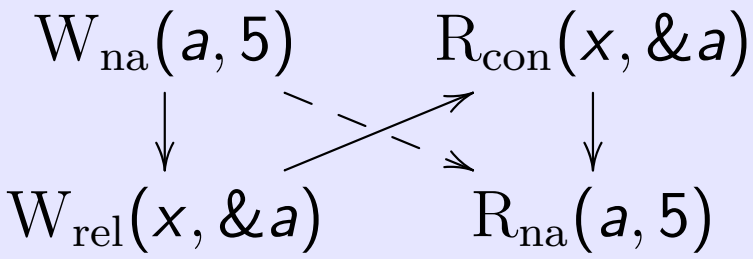
Release-consume synchronization

Initially $a = x = 0$.

$$\begin{array}{l} a = 5; \\ x.\mathit{store}(\mathit{release}, \&a); \end{array} \quad \Bigg\| \quad \begin{array}{l} t = x.\mathit{load}(\mathit{consume}); \\ \textbf{if}\ (t \neq 0)\ \mathit{print}(*t); \end{array}$$

This program cannot crash nor print 0.

Justification:



Release-consume synchronization

Release-consume synchronization

Initially $a = x = 0$. Let $J(t) \stackrel{\text{def}}{=} t = 0 \lor t \mapsto 5$.

$\{\&a \mapsto 0 * \mathbf{W}_J(x)\}$	$\{\mathbf{R}_J(x)\}$
$a = 5;$	$t = x.load(consume);$
$\{\&a \mapsto 5 * \mathbf{W}_J(x)\}$	$\{\nabla_t(t = 0 \lor t \mapsto 5)\}$
$x.store(release, \&a);$	$\mathbf{if}\ (t \neq 0)\ print(*t);$

This program cannot crash nor print 0.

PL consequences:
Needs funny modality, but otherwise OK.

Proposed rules for consume accesses

$$\left\{\mathbf{R}_{\mathcal{Q}}(x)\right\}\ t := x.\mathsf{load}(\mathit{cons})\ \left\{\mathbf{R}_{\mathcal{Q}[t:=\mathsf{emp}]}(x) * \nabla_t\, \mathcal{Q}(t)\right\}$$

$$\frac{\begin{array}{c}\left\{P\right\}\ C\ \left\{Q\right\} \\ C \text{ is basic command mentioning } t\end{array}}{\left\{\nabla_t\, P\right\}\ C\ \left\{\nabla_t\, Q\right\}}$$

Mutual exclusion locks

Let $\mathcal{Q}_J(v) \stackrel{\text{def}}{=} (v = 0 \wedge \text{emp}) \vee (v = 1 \wedge J)$

$Lock(x, J) \stackrel{\text{def}}{=} \mathbf{W}_{\mathcal{Q}_J}(x) * \mathbf{R}^{\text{CAS}}_{\mathcal{Q}_J}(x)$

$new\text{-}lock() \stackrel{\text{def}}{=}$

$\{J\}$

$res = \mathbf{alloc}(1)$

$\{Lock(res, J)\}$

$unlock(x) \stackrel{\text{def}}{=}$

$\{J * Lock(x, J)\}$

$x.\text{store}(1, rel)$

$\{Lock(x, J)\}$

$lock(x) \stackrel{\text{def}}{=}$

$\{Lock(x, J)\}$

repeat

$\{Lock(x, J)\}$

$y = x.\text{CAS}(1, 0, acq, rlx)$

$\left\{Lock(x, J) * \begin{pmatrix} y{=}0 \wedge \text{emp} \\ \vee\ y{=}1 \wedge J \end{pmatrix}\right\}$

until $y \neq 0$

$\{J * Lock(x, J)\}$

Summary of program logic features

Access kind	Program logic features
non-atomic	normal SL $\mapsto$
SC-atomic	normal CSL invariants
release/ acquire	single-location invariants unidirectional ownership transfer
relaxed	send only $\triangle P$; receive ∇P
consume	receive only $\nabla_t P$

Fence kind	Program logic effect
release	introduces $\triangle P$
acquire	eliminates ∇P and $\nabla_t P$

Slogan

Relaxed program logics
are good tools for
understanding
weak memory models