

C Concurrency: Still Tricky

Francesco Zappa Nardelli

Inria, France

Based on work done with Morisset, Pawan, Vafeiadis, Balabonsky, Chakraborty

MPI-SWS and Inria

```
int a = 1;
int b = 0;
```

Thread 1

```
int s;
for (s=0; s!=4; s++) {
  if (a==1)
    return NULL;
  for (b=0; b>=26; ++b)
   ,
```

Thread 2

}

```
int a = 1;
int b = 0;
```

Thread 1

```
int s;
for (s=0; s!=4; s++) {
    if (a==1)
        return NULL;
    for (b=0; b>=26; ++b)
      ;
}
```

```
b = 42;
printf("%d\n", b);
```

```
int a = 1;
int b = 0;
```

Thread 1

```
int s;
for (s=0; s!=4; s++) {
    if (a==1)
        return NULL;
    for (b=0; b>=26; ++b)
      ;
}
```

Thread 2

b = 42;
printf("%d\n", b);

```
int a = 1;
int b = 0;
```

Thread 1

int s;
for (s=0; s!=4; s++) {
 if (a==1)
 return NULL;
 for (b=0; b>=26; ++b)
 ;
}

```
int a = 1;
int b = 0;
```

Thread 1

```
int s;
for (s=0; s!=4; s++) {
    if (a==1)
        return NULL;
    for (b=0; b>=26; ++b)
      ;
}
```

```
b = 42;
printf("%d\n", b);
```

Thread 1

```
int s;
for (s=0; s!=4; s++) {
    if (a==1)
       return NULL;
    for (b=0; b>=26; ++b)
      ;
}
```

```
int a = 1;
int b = 0;
```

Thread 1

Thread 2

```
int s;
for (s=0; s!=4; s++) {
    if (a==1)
        return NULL;
    for (b=0; b>=26; ++b)
      ;
}
```

Thread 1 returns without modifying b

```
int a = 1;
int b = 0;
```

Thread 1

Thread 2

Thread 2 is not affected by Thread 1 and vice-versa

```
int a = 1;
int b = 0;
```

Thread 1

Thread 2

I expect this program to print 42

```
int a = 1;
int b = 0;
```

Thread 1

```
int s;
for (s=0; s!=4; s++) {
  if (a==1)
    return NULL;
  for (b=0; b>=26; ++b)
   ,
```

Thread 2

}

```
int a = 1;
int b = 0;
```

Thread 1

```
int s;
for (s=0; s!=4; s++) {
  if (a==1)
    return NULL;
  for (b=0; b>=26; ++b)
   ,
```

Thread 2

}

```
int a = 1;
int b = 0;
```

Thread 1

```
int s;
for (s=0; s!=4; s++) {
    if (a==1)
      return NULL;
    for (b=0; b>=26; ++b)
    ;
}
```

Thread 2



...sometimes we get 0 on the screen

```
int s;
for (s=0; s!=4; s++) {
    if (a==1)
       return NULL;
    for (b=0; b>=26; ++b)
      ;
}
```

```
movl a(%rip), %eax  # load a into eax
movl b(%rip), %ebx  # load b into ebx
testl %eax, %eax  # if a==1
jne .L2  # jump to .L2
movl $0, b(%rip)
ret
.L2:
movl %ebx, b(%rip)  # store ebx into b
xorl %eax, %eax  # store 0 into eax
ret  # return
```

The outer loop can be (and is) optimised away

mott	$2/2$ r_1 r_2 r_2 r_3 r_2 r_3 r_2 r_3	# 1020 2 $10+0$ 0237
MOVI	a(%rip), %eax	# load a into eax
movl	b(%rip), %ebx	<pre># load b into ebx</pre>
testl	%eax, %eax	# if a==1
jne	.L2	# jump to .L2
movl	\$0, b(%rip)	
ret		
.L2:		
movl	%ebx, b(%rip)	<pre># store ebx into b</pre>
xorl	%eax, %eax	<pre># store 0 into eax</pre>
ret		# return

eax
ebx
L2
nto b
o eax

movl	a(%rip), %eax	<pre># load a into eax</pre>
movl	b(%rip), %ebx	<pre># load b into ebx</pre>
testl	%eax, %eax	# if a==1
jne	.L2	<pre># jump to .L2</pre>
movl	\$0, b(%rip)	
ret		
.L2:		
movl	%ebx, b(%rip)	<pre># store ebx into b</pre>
xorl	%eax, %eax	<pre># store 0 into eax</pre>
ret		# return

```
movl a(%rip), %eax # load a into eax
movl b(%rip), %ebx # load b into ebx
testl %eax, %eax # if a==1
jne .L2 # jump to .L2
movl $0, b(%rip)
ret
.L2:
movl %ebx, b(%rip) # store ebx into b
xorl %eax, %eax # store 0 into eax
ret # return
```

movl	a(%rip), %eax	<pre># load a into eax</pre>
movl	b(%rip), %ebx	<pre># load b into ebx</pre>
testl	%eax, %eax	# if a==1
jne	.L2	<pre># jump to .L2</pre>
movl	\$0, b(%rip)	
ret		
.L2:		
movl	%ebx, b(%rip)	<pre># store ebx into b</pre>
xorl	%eax, %eax	<pre># store 0 into eax</pre>
ret		# return

```
movl a(%rip), %eax # load a into eax
movl b(%rip), %ebx # load b into ebx
testl %eax, %eax # if a==1
jne .L2 # jump to .L2
movl $0, b(%rip)
ret
.L2:
movl %ebx, b(%rip) # store ebx into b
xorl %eax, %eax # store 0 into eax
ret # return
```

The compiled code saves and restores **b** Correct result in a sequential setting

movl	a(%rip), %eax	<pre># load a into eax</pre>
movl	b(%rip), %ebx	<pre># load b into ebx</pre>
testl	%eax, %eax	# if a==1
jne	.L2	# jump to .L2
movl	\$0, b(%rip)	
ret		
.L2:		
movl	%ebx, b(%rip)	<pre># store ebx into b</pre>
xorl	%eax, %eax	<pre># store 0 into eax</pre>
ret		# return

```
int a = 1;
int b = 0;
```

Thread 1

movl a(%rip),%eax movl b(%rip),%ebx testl %eax, %eax jne .L2 movl \$0, b(%rip) ret .L2: movl %ebx, b(%rip) xorl %eax, %eax ret

Thread 2

b = 42;
printf("%d\n", b);

```
int a = 1;
int b = 0;
```

Thread 1

movl	a(%rip),%eax
movl	b(%rip),%ebx
testl	%eax, %eax
jne	.L2
movl	\$0, b(%rip)
ret	
L2:	
movl	%ebx, b(%rip)
xorl	%eax, %eax
ret	•

Thread 2

b = 42;
printf("%d\n", b);

- Read a (1) into eax

```
int a = 1;
int b = 0;
```

Thread 1

mo∨l	a(%rip),%eax
movl	b(%rip),%ebx
testl	%eax, %eax
jne	.L2
movl	\$0, b(%rip)
ret	
L2:	
mo∨l	%ebx, b(%rip)
xorl	%eax, %eax
ret	

- Read a (1) into eax
- Read b (0) into ebx

```
int a = 1;
int b = 0;
```

Thread 1

movl a(%rip),%eax movl b(%rip),%ebx testl %eax, %eax jne .L2 movl \$0, b(%rip) ret .L2: movl %ebx, b(%rip) xorl %eax, %eax ret

Thread 2

b = 42;
printf("%d\n", b);

- Read a (1) into eax
- Read b (0) into ebx
- Store 42 into b

```
int a = 1;
int b = 0;
```

Thread 1

a(%rip),%eax movl movl b(%rip),%ebx testl %eax, %eax jne .L2 \$0, b(%rip) movl ret .L2: %ebx, b(%rip) movl %eax, %eax xorl ret

- Read a (1) into eax
- Read **b** (**0**) into **ebx**
- Store 42 into b
- Store ebx (0) into b

```
int a = 1;
int b = 0;
```

Thread 1

a(%rip),%eax movl movl b(%rip),%ebx testl %eax, %eax jne .L2 \$0, b(%rip) movl ret .L2: movl %ebx, b(%rip) %eax, %eax xorl ret

- Read a (1) into eax
- Read b (0) into ebx
- Store 42 into b
- Store ebx(0) into b
- Print b: 0 is printed

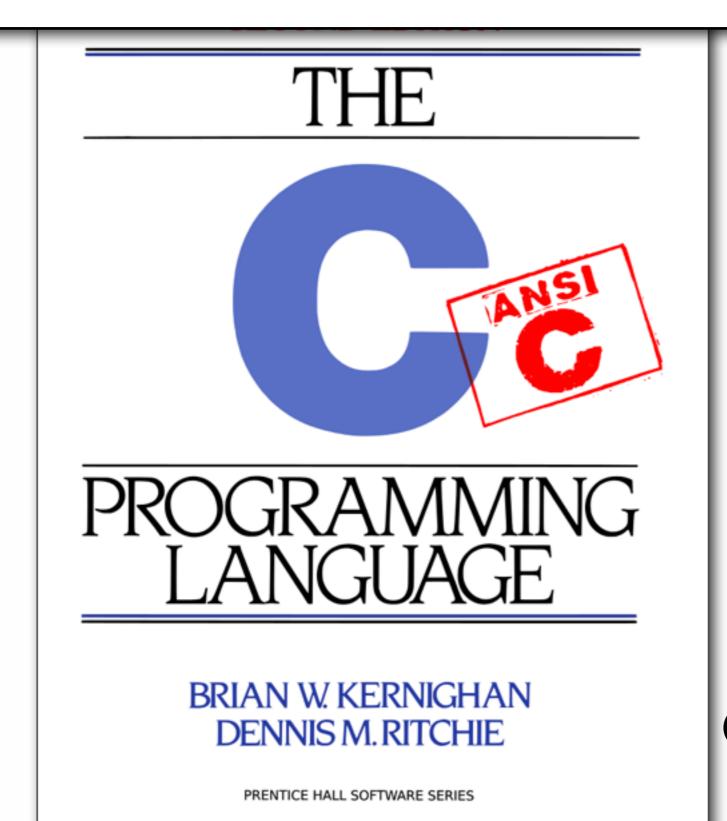


C can't be so nasty! Must be a subtle compiler bug.

Of course C allows this. No news here.

C can't be so nasty! Must be a subtle compiler bug.

What is C?



K&R ANSI C C99 C11

DeFacto C: whatever C compilers implement C programmers rely on

THE K&R Image: Coordinate of the second se

1980 - ... : widespread use of threads, no spec, poor understanding of constraints

2005 onwards: proposals by Boehm, Adve, C++0x concurrency subgroup

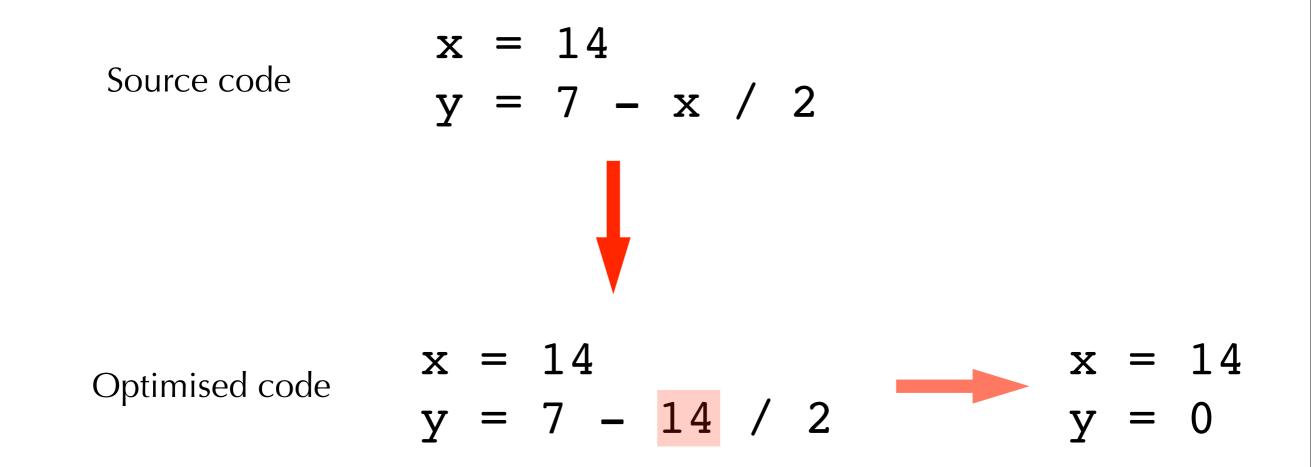
2009-2011: Batty et al., draft standard \Rightarrow math \Rightarrow fixes \Rightarrow C/C++11 standard

PRENTICE HALL SOFTWARE SERIES

Why is it so hard?

Constant propagation

A simple, and *innocuous*, optimisation:



Shared memory concurrency

Shared memory

Shared memory concurrency

Shared memory

Intuitively this program always prints 0

Shared memory concurrency

But if the compiler propagates the *constant* x = 1...

$$x = y = 0$$

Thread 1

Shared memory concurrency

But if the compiler propagates the *constant* x = 1...

Thread 1

...the program always writes 1 rather than 0.

This talk

- 0. Concurrency and optimisations, not so simple
- 1. The layman semantics
- 2. Escape lanes for the expert programmer

3. Compiler testing via a theory of sound optimisations

4. Escape lanes are a Pandora's box

5. The way forward...

The layman solution forbid data-races

Monday 11 May 15

Standard way out: prohibit data races

Two memory accesses conflict if they

- access the same memory location, e.g. variable
- at least one access is a store

A program has a data race if two data accesses • conflict, and

can occur simultaneously in a sequentially consistent execution.

A program data-race-free (on a particular input) if no sequentially consistent execution results in a data race.

ADA 83

[ANSI-STD-1815A-1983, 9.11] For the actions performed by a program that uses shared variables, the following assumptions can always be made:

- If between two synchronization points in a task, this task reads a shared variable whose type is a scalar or access type, then the variable is not updated by any other task at any time between these two points.
- If between two synchronization points in a task, this task updates a shared variable whose task type is a scalar or access type, then the variable is neither read nor updated by any other task at any time between these two points.

The execution of the program is erroneous if any of these assumptions is violated.

Posix Threads Specification

[IEEE 1003.1-2008, Base Definitions 4.11] Applications shall ensure that access to any memory location by more than one thread of control (threads or processes) is restricted such that no thread of control can read or modify a memory location while another thread of control may be modifying it.

C++2011 / C11

[C++ 2011 FDIS (WG21/N3290) 1.10p21] The execution of a program contains a *data* race if it contains two conflicting actions in different threads, at least one of which is not atomic, and neither happens before the other. Any such data race results in undefined behavior.

C++2011 / C11

[C++ 2011 FDIS (WG21/N3290) 1.10p21] The execution of a program contains a *data* race if it contains two conflicting actions in different threads, at least one of which is not atomic, and neither happens before the other. Any such data race results in undefined behavior.

How to use C/C++ to implement low-level system code?

Escape lanes for expert programmers

Low-level atomics in C11/C++11

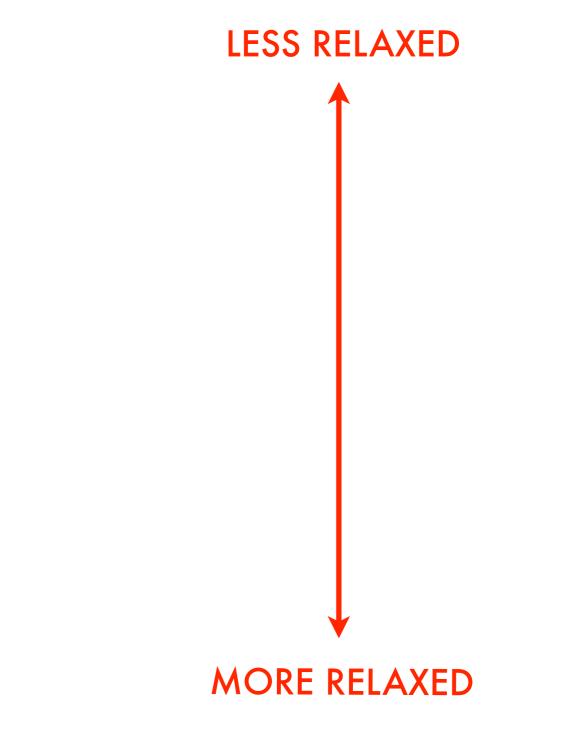
```
std::atomic<int> flag0(0),flag1(0),turn(0);
void lock(unsigned index) {
   if (0 == index) {
                                                       Atomic variable declaration
        flag0.store(1, std::memory_order_relaxed);
        turn.exchange(1, std::memory_order_acq_rel);
       while (flag1.load(std::memory_order_acquire) .
           && 1 == turn.load(std::memory_order_relaxed))
           std::this_thread::yield();
    } else {
        flag1.store(1, std::memory_order_relaxed);
                                                                 New syntax
        turn.exchange(0, std::memory_order_acq_rel);
                                                                 for memory accesses
       while (flag0.load(std::memory_order_acquire)
           && 0 == turn.load(std::memory_order_relaxed))
           std::this_thread::yield();
    }
}
                                                                 Qualifier
void unlock(unsigned index) {
    if (0 == index) {
        flag0.store(0, std::memory_order_release);
    } else {
        flag1.store(0, std::memory_order_release);
    }
```

MO_SEQ_CST

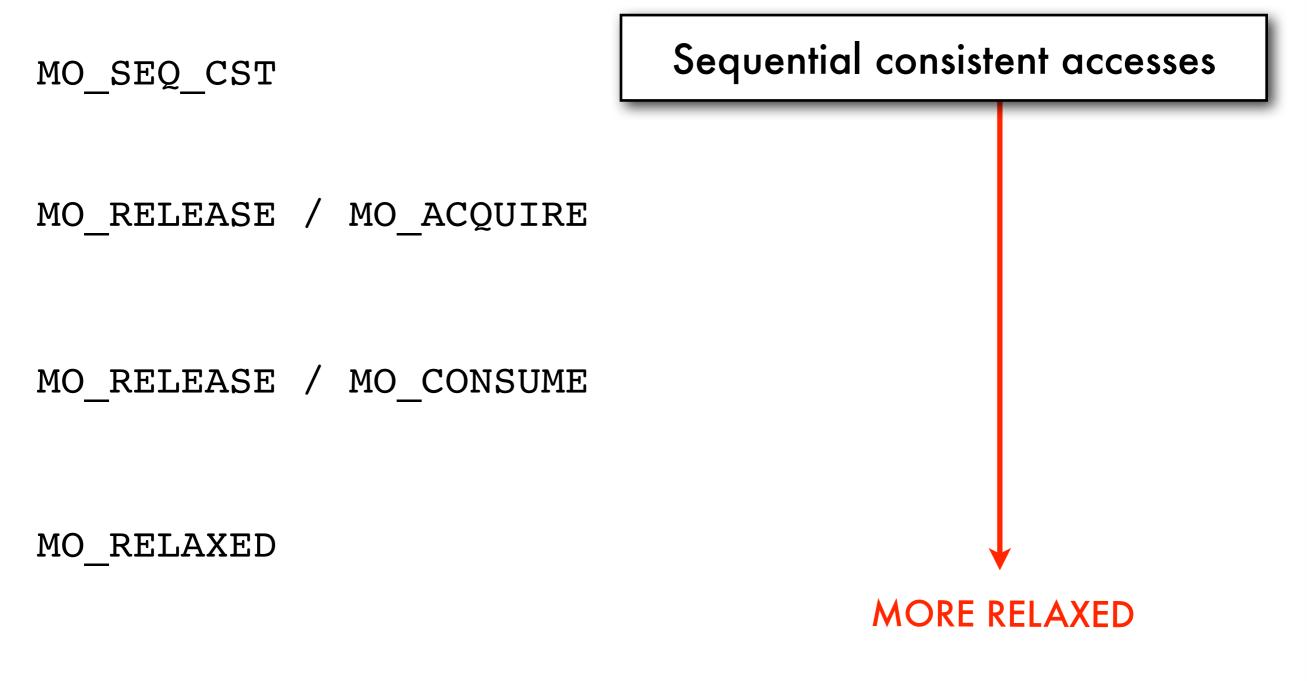
MO_RELEASE / MO_ACQUIRE

MO_RELEASE / MO_CONSUME

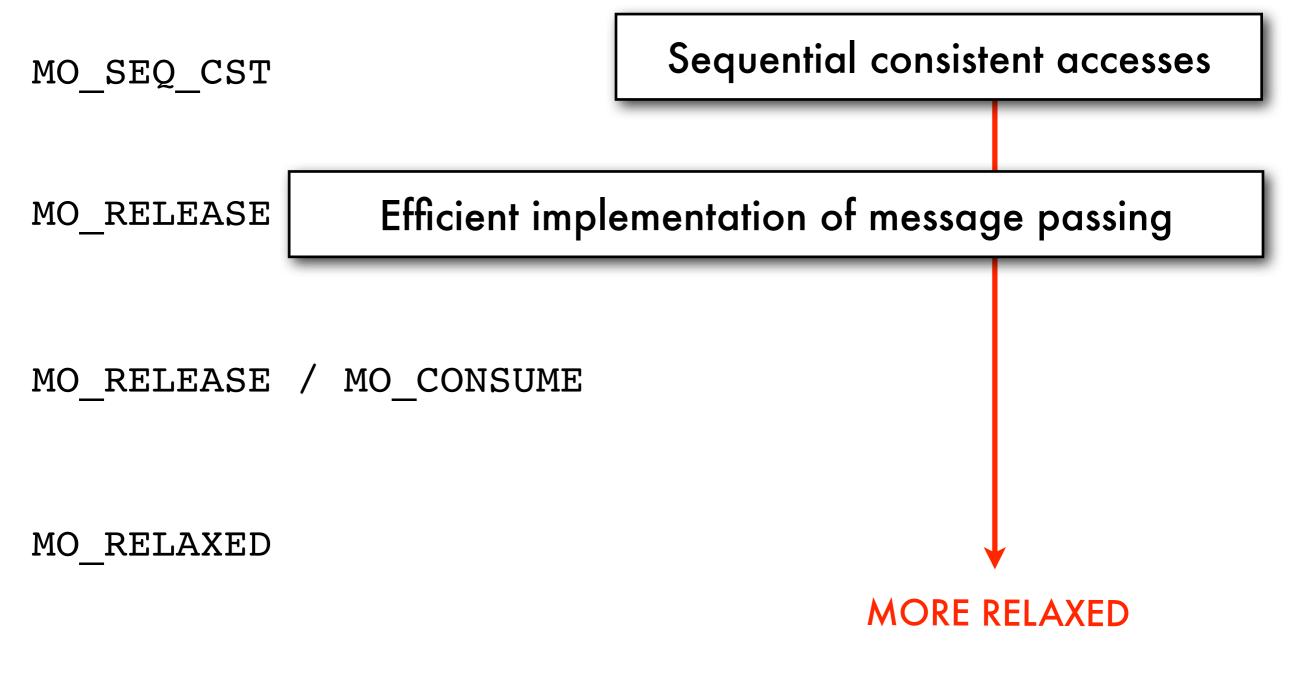
MO RELAXED



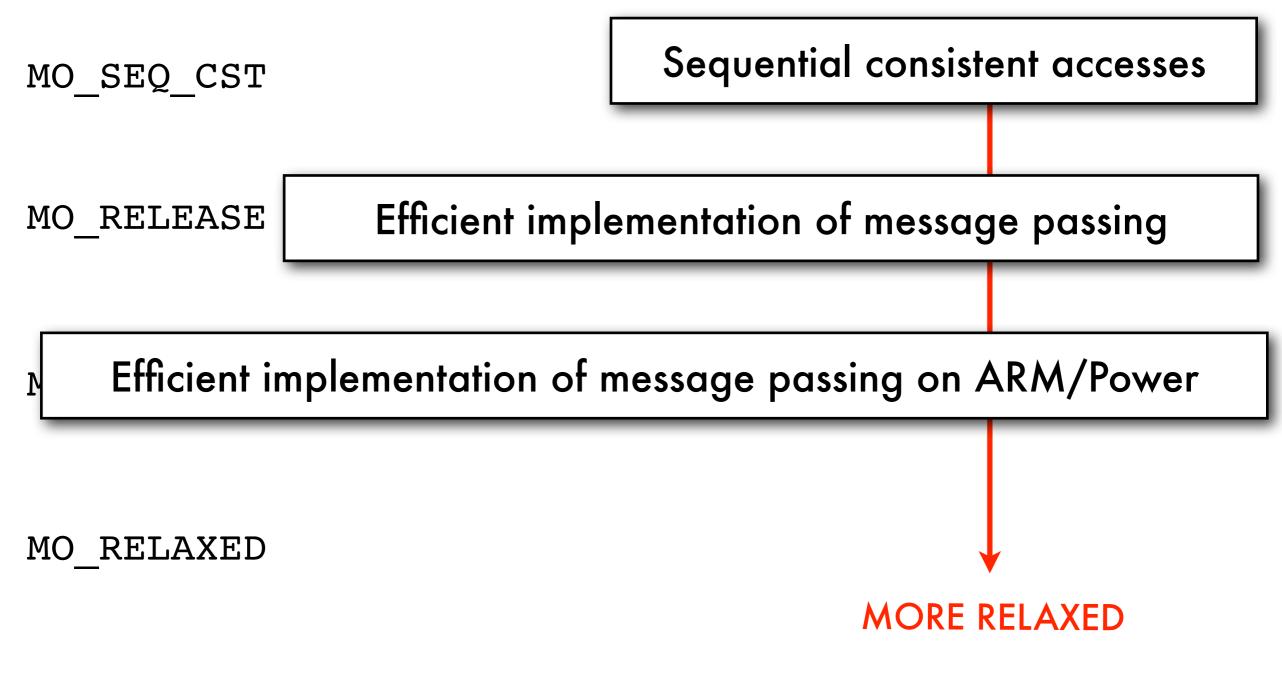




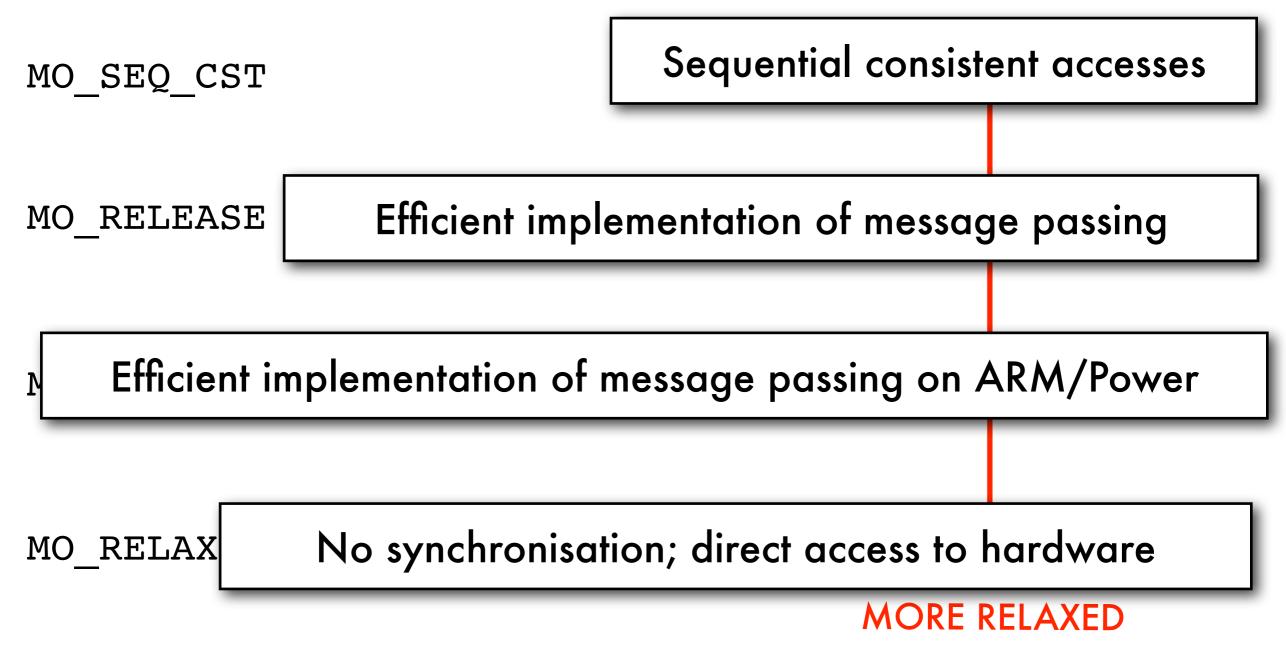




LESS RELAXED



LESS RELAXED



Memory access synchronisation

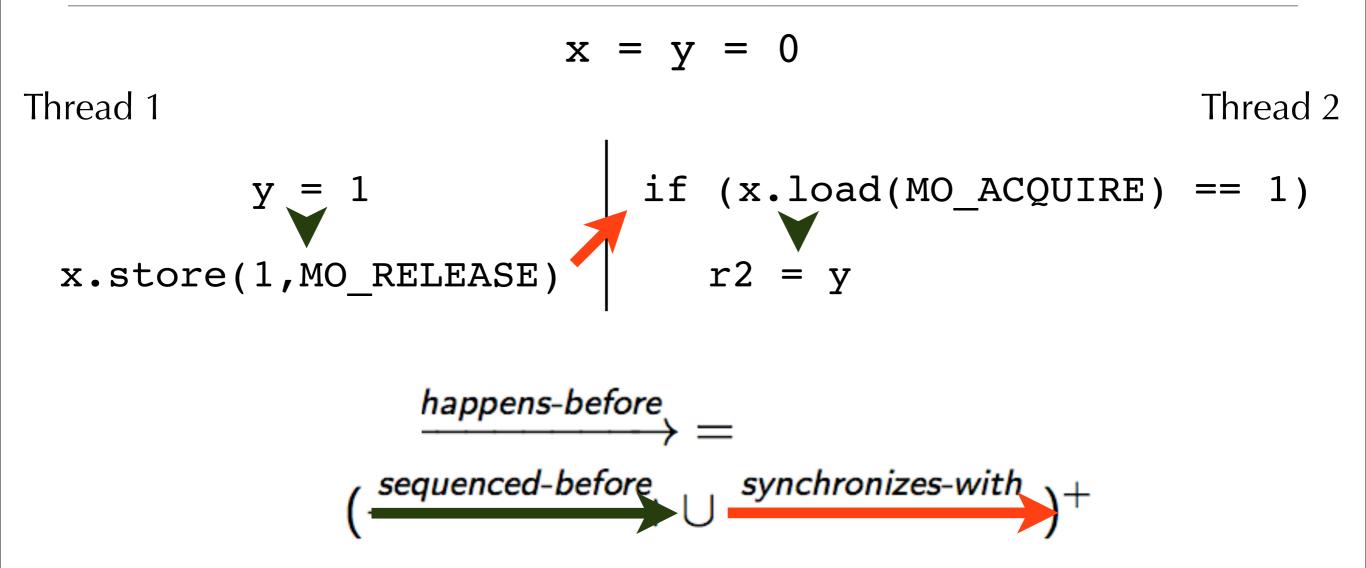
$$x = y = 0$$
Thread 1
$$y = 1$$

$$x.store(1,MO_RELEASE)$$

$$x = y = 0$$
Thread 2
$$if (x.load(MO_ACQUIRE) == 1)$$

$$r2 = y$$

Memory access synchronisation



Non-atomic loads must return the *most recent write* in the happens-before order (unique in a DRF program)

Understanding MO_RELAXED

$$\mathbf{x} = \mathbf{y} = \mathbf{0}$$

Thread 1

X.S

Understanding MO_RELAXED

$$x = y = 0$$
hread 1
$$y = 1$$

$$if (x.load(MO_RELAXED) == 1)$$

$$x.store(1,MO_RELAXED)$$

$$r2 = y$$

DATA RACE

Two conflicting accesses not related by happens-before

Τ

1)

Understanding MO_RELAXED

$$\mathbf{x} = \mathbf{y} = \mathbf{0}$$

Thread 1

Thread 2

y.store(1,MO_RELAXED) if (x.load(MO_RELAXED) == 1)

x.store(1,MO_RELAXED)

r2 = y.load(MO RELAXED)

WELL DEFINED

but $r^2 = 0$ is possible

Intuition

the compiler (or hardware) can reorder independent accesses

WELL DEFINED

but $r^2 = 0$ is possible

Intuition

the compiler (or hardware) can reorder independent accesses

Allow a RELAXED load to see any store that:

- does not happens-after it
- is not hidden by an intervening store hb-ordered between them

The full model

	is_store $a = case a of STORE \to T \parallel \to F$		
$a \stackrel{r}{\rightarrow} b = (a, b) \in r$]	vible_side_effect_ert actions threads location-kind aquanced lefter additional-spectronized with data-dependency laguest before =
$a r b = (a, b) \in r$	is_fence a = case a of FENCE→ T _ → F	rs_olement rs_head = = same_thread = rs_head V is_stomic_traw =	(a) & Appendishter, ket (x,b) = ab in Visible abide, detect action fluends, because defear additional synchronized with data dependency cantrol-dependency happens before a b)
$a \stackrel{\circ}{\to} b = (a, b) \notin r$	is_lock_or_unlock a = is_lock a∨is_unlock a	$release_{sequence} = a_{seq} \xrightarrow{nhame sequence} b =$	visible.argence.af.side.affets.and = visible.argence.af.side.affets.and + a
$a \neq b = (a, b) \notin r$	is_atomic_action a = is_atomic_load a ∨ is_atomic_store a ∨ is_atomic_rmw a	$b \neq a \text{ control bornion } b \land$ is the control of	(c.vze, lost materials of c.) (b. hagen stars) (c) August atom (c) Augu
$\stackrel{r}{\rightarrow} = r$		$(v - w_0)$, w_0 (regularized by $b \to 0$) $(r_v c)$ -means $a_{ud} \ b \land a_{ud}$ (regularized by $b \land b $	(Viz vsze_badz amilitation volte, c === - {b approx botta; c === - {b approx botta; c}]))
$a \xrightarrow{r} b \xrightarrow{s} c = a \xrightarrow{r} b \wedge b \xrightarrow{s} c$	is_load_or_store a = is_load a V is_store a	(($m_{j}m_{2}p \in f = \{r, 2\pi \in \pi : (r - f \times j)\}$
relation_over s rel = domain rel $\subseteq \ s \wedge$ range rel $\subseteq \ s$	is_read $a = $ is_atomic_load $a \lor$ is_atomic_rmw $a \lor$ is_load a	release_sequence_set actions threads location-kind sequenced-before additional-synchronized-with data-dependency control-dependency modification-order =	visibi_uequences_uf_side_uffects = visibi_uequences_uf_side_uffects =
$\xrightarrow{nl}_{ s} = rel \cap (s \times s)$	is_write a =	release-sequence actions threads location-kind sequenced-before additional-genchronized-with data-dependency control-dependency modification-order a b]	V(virz.host, b). (A if ust.atomic.location b then
$rel _s = rel \cap (s \times s)$	is_atomic_store a∨is_atomic_rmw a∨is_store a		(vsz.had) visible_sequence_ud_uide_stfectu_tail vszhad b eks
$\frac{d}{dt} _{s} = rel \cap (s \times s)$	is_acquire $\mathbf{a} = ($ case memory_order \mathbf{a} of SOME mem_ord \rightarrow	$(b = a) \vee$ (x_{a}) constrained by $a = b \wedge a$ (x_{c}) constrained by $a = b \wedge a$ (x_{c}) constrained by $b = a$	
$\rightarrow _{s} = Rr \mapsto (s \times s)$	(mem_ord ∈ {Mo_ACQUIRE, Mo_ACQ_REL, Mo_SEQ_CST} ∧	rscelement a c)))	vide_sequence_st_side_affect_set actions threads bacino kind sequences before additional performance with data dependency control dependency modification order happens before video kind effect = mystage (video-arrows-st_side).effects actions threads bacina bacine sequences before video-kind effect = mystage (video-arrows-st_side).effects actions threads bacine bacine sequences before video-kind effects = mystage (video-arrows-st_sideo-arrows-state).effects = mystage (video-arrows-state).effects = mystage (video-arro
$rel _s = rel \cap (s \times s)$	(s.cread 2 v is,fence 3)) ∨ (* 29.8:5 states that consume fences are acquire fences. *) ((mem.ord = MO_CONSUME) ∧ is.fence: 2)	hypothetical_release_sequence_set actions threads location-kind sequenced-before additional-synchronized-with data-dependency control-dependency modification-order =	consistent_reads_from_mapping = consistent_reads_from_mapping =
${\rm strict}_{*}{\rm preorder}~{\it ord}={\rm irreflexive}~{\it ord}\wedge{\rm trans}~{\it ord}$	$\parallel \text{NONE} \rightarrow \text{is-lock a}$	hypothetical_release_sequence actions threads location-kind sequenced before additional-synchronized with data-dependency control-dependency modification-order a b}	(b) (sured $h \land i = 1$ constraint $h \land i = 1$) \Rightarrow (if $(3_{i_{1}}, a_{i_{2}}, a_{i_{2}$
total_over s and = relation_over s and ∧	is_consume a = is_read a (memory_order a = Some Mo_consume)	synchronizes, with $a \ge \frac{a_{1}c_{2}c_{2}a_{3}a_{3}}{(s - a_{2}d_{3}b_{3}a_{3})} b = (s - a_{2}d_{3}b_{3}a_{3}b_{3}b_{3}a_{3}b_{3}b_{3}a_{3}b_{3}b_{3}a_{3}b_{3}b_{3}a_{3}b_{3}b_{3}b_{3}b_{3}b_{3}b_{3}b_{3}b$	$\dim (\exists_{\omega_{-}} f_{\omega_{-}} \longrightarrow 0 \land f_{\omega_{-}} \to 0)$ $\dim (\exists_{\omega_{-}} f_{\omega_{-}} \to 0)$
$(\forall x \in s. \forall y \in s. x \xrightarrow{ad} y \lor y \xrightarrow{ad} x \lor (x = y))$	is_release a =	a <u>attituid quaterial estimate</u> (ame_location ≥ b ∧ a ∈ actions ∧ b ∈ actions ∧ ((16 (μ -md b Λ i.u.q.(atomic_boxton b) $\rightarrow \rightarrow $ (if (($I(\Lambda) = 0)$) columb ($I(\Lambda) = 0$))
<pre>strict_total_order_over s and = strict_preorder ord / total_over s and</pre>	(case memory_order a of SOME mem.ord → mem.ord ∈ (Mo_RELEASE, Mo_ACQ_REL, Mo_SEQ_CST) ∧	(* – mutex synchronization – *) (is_umlock $a \land is_b lock b \land a \stackrel{s}{\Longrightarrow} b) \lor$	then $(\exists \langle x', xxy) \in violab expansions of solid effects. (x' = b) \land [\exists x \in vxxs. \in \stackrel{d}{\rightarrow} b])effect (\exists x, y \in [x]) \land (\forall x \in \stackrel{d}{\rightarrow} b])$
	imention < Notherstein Notherstein Notherstein Notherstein N (is_write a V is feace a) NONE → is_unlock a)	(* – release/acquire synchronization – *) (%⊥release → K⊥_exquire b ∧ → same_thread a b ∧	$(\mathbf{v}(\mathbf{x}, \mathbf{z}) \in \vec{\Delta}$.
$\begin{array}{c} x \stackrel{\text{out}}{\longrightarrow} pred \; y = \\ pred \; x \wedge x \stackrel{\text{out}}{\longrightarrow} \; y \wedge \neg (\exists z, pred \; z \wedge x \stackrel{\text{out}}{\longrightarrow} z \stackrel{\text{out}}{\longrightarrow} y) \end{array}$	is_seq_cst a = (memory_order a = SOME MO_SEQ_CST)	$(3c, z, \frac{ndum mpana}{z}, c, \frac{d}{d}, b)) \lor$ (* - 6mc + unchronization - *)	$\forall (\mathbf{r}, \mathbf{s}) \in \mathcal{A},$ \mathbf{s} the second $\mathbf{s}_{\mathbf{s}} \wedge \mathbf{s}$
$x \mapsto y =$		(is fonce λ is precase $\lambda \wedge$ is fonce $b \wedge$ is acquire $b \wedge$ ($\exists x, \exists y, same location x \neq \lambda$	same_location $2 + h \leq u_{+}$ stateming_location b $\implies (x = y) \lor x \xrightarrow{n \text{ influence setup}} y) \land$ (* new GMR *)
$x \xrightarrow{\operatorname{out}} y \land \neg (\exists z. x \xrightarrow{\operatorname{out}} z \xrightarrow{\operatorname{out}} y)$	location_kind = MUTEX NON_NTOMIC	is_atomic_action x , the statemic_action y , is_write $x \land$ x = means that $y_i \land y_i$ = means that $y_i \land f_i$ ($2x, x$ = isotenticity down equation $x_i \in f_i$) ($y) \lor (x \land y)$	$(\gamma(\mathbf{x}, \mathbf{z}) \in \frac{1}{2}$
well_founded $r = wf r$	ATOMIC	J (is_fence a ∧ is_release a ∧	$c \stackrel{d}{\longrightarrow} b \Lambda$, is_vrite $_2 \wedge \text{same_kontain} = b \wedge \text{is_kst}_katomic_kontain} = b$ $\implies (c = 2) \vee 2 \xrightarrow{\text{substantian}} c \Lambda$
type_abbrev action_id : string	actions_respect_location_kinds = actions_respect_location_kinds = ∀z. case location z of SOME / →	is atomic action $b \land is acceptive b \land$ (3r, same location $x \land b \land$ is atomic action $x \land is write x \land$	$ \begin{array}{c} \Longrightarrow (c-a)(v) = & \\ (f \operatorname{neur} GMW^{h}) \\ (f(a,b) \in \frac{happen halow}{a}. \end{array} $
type_abbrev thread_id:string	case location a of SOME I → (case location-kind I of MUTEX → is_lock_or_unlock a	$a = \frac{1}{2} \sum_{k=1}^{n} $	vc c → ∧
type_abbrev location : string	NON_ATOMIC → is_load_or_store a ATOMIC → is_load_or_store a ∨ is_atomic_action a) NONE → T	(active and the set of	is write $h \land \text{manual-contin} \Rightarrow h \land \text{is at atomic location } a \implies c$ matterian edge, $b \land \land$
type_abbrev val:string		(3c, same-location $2 \times h$ is atomic_action $x \wedge x$ x = x = x = x = x = x	$(\gamma(\mathbf{x})) \subset \mathcal{A}$, inclusion of $\mathbf{x} \to 0$ (A)
[is_ut_location_kind = $h_ut_location_kind =$ case location a of SOME $l \rightarrow (location-kind l = hd)$ $\parallel \text{NONE} \rightarrow F$	$(\exists z. 2 \xrightarrow{\text{offere segment}} z \xrightarrow{d'} x))))))$	$(\gamma(x,b) \in \stackrel{d}{\rightarrow}$, isoequat b
memory_order_enum = MO_SEQ_CST MO_RELAXED	NONE → F	synchronizes_with_set actions threads location-kind sequenced before additional-synchronized with data-dependency control-dependency of modification-order sc release-sequence hypothetical-release-sequence =	$= \varphi \left(-i_{k} eq_{k} e_{k} x \cdot x_{k}^{-1} z_{k} \cdot z_{k} e_{k} e_{$
MO_RELEASE MO_ACQUIRE	is_nt_mutex_location ə = is_nt_location_kind ə MUTEX	synchronizes, with actions thanks location-kind sequenced before additional-synchronized with data-dependency control-dependency of modification-order sc release-sequence hypothetical-release-sequence a b}	(* -Fence restrictions *) (* 29.3.3 *)
Mo_consume Mo_acq_rel	is_at_non_atomic_location a =	carries, u, dependency. Lo = $a \xrightarrow{\text{carries}} b = a \left(\left(\frac{d}{d_1} \cap \frac{m_{\text{stands}}}{d_2} \right) \left(\frac{d}{d_2} - \frac{d}{d_2} \right) \right) \left(\frac{d}{d_2} - \frac{d}{d_2} \right) \left(\frac{d}{d_2} + $	$(k_{2}, k_{3}, k_{3}) \in \frac{\text{reward below}}{k_{3}}, \forall p, $ $(k_{3}, k_{3}, k_{3}) \in (k_{3}, k_{3}, k_{3}) \in (k_{3}, k_{3}) $
action = LOCK of action_id thread_id location	is_at_location_kind a NON_ATOMIC	carriesdependency_to_set actions throuts location-kind sequenced-before additional-synchronized-with data-dependency control-dependency rf =	is vertice λ same based on the $\lambda \ge h$ $2 \stackrel{(n)}{\longrightarrow} x \land y \stackrel{(n)}{\longrightarrow} b$ $\approx (y = 2)^{2} \stackrel{(n)}{\longrightarrow} (x = 2)^{2} \stackrel{(n)}{\longrightarrow} y \land h$
UNLOCK of action_id thread_id location ATOMIC_LOAD of action_id thread_id memory_order_enum location val ATOMIC_STORE of action_id thread_id memory_order_enum location val	is_at_atomic_location $a =$ is_at_alocation_kind a ATOMIC	euries_udependency_to actions threads location-kind sequenced-before additional-genchronized with data-dependency control-dependency of a b}	(* 20 % 4 *)
ATOMIC_RMW of action_id thread_id memory_order_enum location val val LOAD of action_id thread_id location val	same_thread $a b = (thread_id_of a = thread_id_of b)$	dependency_contered_before = 2 $\frac{dependency-outlend barlow}{d} d =$	$(\pi(x, x) \in \frac{\text{impartice balling}}{(x, x) \in x}, \forall (y, b) \in \vec{n}.$ (isotomic-action $a \land b$ isotome $x \land b$ isotome $x \land b$ is vertice $a \land b$ and boarding $b \land b$
STORE of action_id thread_id location val FENCE of action_id thread_id memory_order_enum	threadwise_relation_over s rel =	$\mathbf{z} \in \operatorname{actions} \wedge d \in \operatorname{actions} \wedge$ $(\exists k: \operatorname{actions} \wedge h \subseteq \operatorname{actions} h \wedge$ $(\exists k: \operatorname{actions} h \in \operatorname{actions} h)$	$x \approx b$ (statistication during b) $\Rightarrow (y = 2) \lor 2 \xrightarrow{\text{sublick}(x) = 0} y) \land$
$(action_id_of (LOCK aid _) = aid) \land$ $(action_id_of (UNLOCK aid _) = aid) \land$	$\begin{array}{c} \operatorname{Interaction_source} rel \land (\forall (a, b) \in rel. \operatorname{same_thread} a b) \end{array}$	$(b \xrightarrow{\text{crises-dependency}} d \lor (b = d)))$	(* 29.3.5 *) (*($x, y) \in \frac{mparande balan}{2}, \forall (y, b) \in \frac{mparande balan}{2}, \forall x.$
(action_id_of (ATOMIC_LOAD aid) = aid) ∧ (action_id_of (ATOMIC_STORE aid) = aid) ∧ (action_id_of (ATOMIC_RAW aid) = aid) ∧	same_location $a b = (location a = location b)$	dependency_ordered_before_set actions threads location-kind sequenced-before additional-synchronized-with data-dependency control-dependency of modification-order release-sequence carries-a-dependency-to =	(h_stonic_action a/h_sense x/h_sequet x/ h_senter = h_senter y/h_sequet y/
(action_id_of (LOAD aid) = aid) ∧ (action_id_of (STORE aid _) = aid) ∧	locations_of actions = { l . $\exists a$. (location $a = \text{SOME } l$)}	dependency_undered_before actions threads location-lind sequenced before additional-gentronized with data-dependency control-dependency of modification-order release-sequence carries-a-dependency-to a b)	is atomic write ab $h \to ame Accation \Rightarrow b \landx \neq y \land z \neq b\Rightarrow (z \neq z) \land z^{$
(action_id_of (FENCE aid) = aid)	well_formed_action 2 =	simple_happens_before =	
(thread_id_of (LOCK _ tid _) = tid) \land (thread_id_of (UNLOCK _ tid _) = tid) \land (thread_id_of (ATOMIC_LOAD _ tid) = tid) \land	case \Rightarrow of ATOMIC_LOAD mem_ord _ \rightarrow mem_ord \in {MO_RELAXED, MO_ACQUIRE, MO_SEQ_CST, MO_CONSUME}		all data_dependency = $ \begin{pmatrix} d_{1} & d_{2} & d_{3} & d_{$
(thread_id_of (ATOMIC_STORE _ tid) = tid) \land (thread_id_of (ATOMIC_RMW_ tid _) = tid) \land	ATOMIC_STORE _ mem_ord → mem_ord ∈ {MO_RELAXED, MO_RELEASE, MO_SEQ_CST} ATOMIC_RAW _ mem_ord → mem_ord ∈	consistent_simple_happens_before shb = irreflective (^{Abb} →)	
(thread_id_of (LOAD _ $tid) = tid) \land$ (thread_id_of (STORE _ $tid) = tid) \land$ (thread_id_of (FRORE _ $tid) = tid) \land$			consistent_control_dependency = consistent_control_dependency =
	$ \begin{array}{c} \text{Aloside_max} & _ \text{imm_ord} & _ \rightarrow \text{imm_ord} & \subseteq \\ & \{\text{Mo_RELAXED, Mo_RELEASE, Mo_ACQUIRE, Mo_ACQ_REL, Mo_SEQ_CST, Mo_CONSUME} \} \\ & _ \rightarrow T \end{array} $	inter thread harness before a method approaches	consistent_control_dependency = consistent_control_dependency = intelexive ((
	{Mo_BELAXED, Mo_BELEASE, Mo_ACQUIRE, Mo_ACQ_REL, Mo_SEQ_CST, Mo_CONSUME} - + T well_formed_threads = well_formed_threads =	inter-altered_langenet_defere = inter_sections with 	ancient ancies seine thank, horizo lief annand helm obligation control data her despites and an effection of a set of set of a set o
$(memory_order (ATOMIC_LOAD__mem_ord__) =$ SOME: mem_ord \land $(memory_order (ATOMIC_TORE _ mem_ord _) =$	(Mo_attractor, Mo_attractor, Mo_account,	inter.htms.lingpens.hefore =	consister_accusion actions threads location-land sequenced-before additional synchronized with data-dependency control dependency of modification-order sc = with_format_flows status threads location-land sequenced-before additional synchronized with data-dependency of modification-order sc = with_format_flows status = hypothetical-flows acquerate-before additional synchronized with data-dependency control dependency modification-order in https://www.flows.sci.ex.com/sc
(n=mary=ndm (APDMC=JAND - , mem_ed - ,) = South mem_ed) . (memory=ndm (ArDML=Static - , mem_ed - ,) = South mem_ed) . (memory=ndm (APDMC=JANN - , mem_ed - ,) =	(Mo_attractor, Mo_attractor, Mo_account, Mo_account, Mo_account, Mo_account, Mo_account, [] T well_strand_ltrands = well_strand_ltrands = injum attractude(attrand) / (f2x well_strand_ltractor) / (f2x well_strand_ltractor) / fiterables_relation_are attring segmented-before ∧ thread-before failed account attractor action attractory ∧ thread-before failed account attractory ∧ thread-before failed account attractory ∧	$ \begin{array}{l} \label{eq:constraint} \\ inter.Hreal.happens.hefere = \frac{hire straint hypers hefer}{hire - hire straint hypers hefer} = \\ \begin{tabular}{lllllllllllllllllllllllllllllllllll$	consister_percasion actions threads location-line sequenced before additional-spectronized with data dependency control dependency of modification-order sc = well_formad_threads actions threads location-line standards and sequences of the additional-spectronized with data dependency control dependency in the addition of the additional spectronized with data dependency control dependency in the addition of the additional spectronized with data dependency control dependency in the addition of the additional spectronized with data dependency control dependency in the addition of the additional spectronized with data dependency control dependency in the addition of the additional spectronized with data dependency control dependency and factor of the addition of the additional spectronized with data dependency control dependency and factor of the addition of the additional spectronized with a data dependency control dependency and factor of the addition of the additional spectronized with data dependency and factor of the addition of the additional spectronized with data dependency and factor of the addition of the additional spectronized with data dependency and addition of the addition of the additional spectronized with data dependency and additional spectronized with data dependency of additional spectronized with data dependency and additional spectronized with data dependency control dependency and additional spectronized with data dependency and additional spectronized with data dependency control dependency and additional spectronized with data dependency and additional spectroniz
(memory_inder (ATOMIC_iOADmem_ord) = SOUE: mem_ord) ∧ (memory_inder (ATOMIC_ITOMImem_ord) = SOUE: mem_ord) ∧ (memory_inder (ATOMIC_INNmem_ord) = (memory_inder (TPSNCmem_ord) = SOUE: mem_ord) ∧ (memory_inder _ =	[M_m_attractors, Mm_attractors, Mm_accounts, Mm_accounts, Mm_astra_cost, Mm_astractors, Mm_astr	$\label{eq:constraint} \begin{split} & \operatorname{inter-all-read-hypers.defore} = & & \operatorname{inter-all-approx.defore} = \\ & \operatorname{inter-all-read-approx.defore} & & & = \\ & \operatorname{inter-all-read-approx.defore} & & & & = \\ & & \operatorname{inter-all-read-approx.defore} & & & & \\ & & & \operatorname{inter-all-approx.defore} & & & & \\ & & & & \operatorname{inter-all-approx.defore} & & & & \\ & & & & & \operatorname{inter-all-approx.defore} & & & & \\ & & & & & & \operatorname{inter-all-approx.defore} & & & \\ & & & & & & \operatorname{inter-all-approx.defore} & & & \\ & & & & & & \operatorname{inter-all-approx.defore} & & & \\ & & & & & & & \\ & & & & & & & $	consistent actuation scient timule, braches hiel sequences before additional prochonistical with data dependency or modification order tx = modification during the science of the scien
(mmorp_wolet (Arrows_wolet) - mmm_ord _) = Sould mem_wole). (mmorp_wolet (Arrows_wolet) - mmm_ord _) = Sould mem_wole). (mmorp_wolet (Arrows_wolet) - mmm_ord _) = Sould mem_wole). (Sould mem_wole).	[M=_art_XXIII, M=_articAstr, M=_acqc(HIR, HIR, HIR, HIR, HIR, HIR, HIR, HIR,	$ \begin{array}{l} \label{eq:constraint} \\ inter.Hreal.happens.hefere = \frac{hire straint hypers hefer}{hire - hire straint hypers hefer} = \\ \begin{tabular}{lllllllllllllllllllllllllllllllllll$	consistent_according sections likely sequenced before additional-perchanged with data-dependency or motification-order as = well_Strand_likely data for threads in the location-likely dependency control dependency or motification-order as = well_Strand_likely data for threads in the location-likely dependency control dependency or (consistent_according according data data data dependency control dependency or (consistent_according according data data data data data data dependency control dependency or difficultion-order in left spectration-likely data data data data data data data dat
$\begin{array}{l} (nemery:acder (AVDMC:JADA) = nem_{acd} - a) = \\ SOLH::nem_{acd} / h \\ (nemery:acder (AVDML:Total: = nem_{acd} - a) = \\ SOLH::nem_{acd} / h \\ (source nem_{acd} / h \\ (source nem_{acd} - a) = \\ Solid::nem_{acd} / h \\ (source nem_{acd} - a, nem_{acd} - a) = \\ Solid::nem_{acd} / h \\ (source nem_{acd} - a, nem_{acd} - b) = \\ Solid::nem_{acd} / h \\ (source nem_{acd} - a) = \\ Solid::nem_{acd} / h \\ (source nem_{acd} - a) = \\ Solid::nem_{acd} / h \\ (source nem_{acd} - b) = \\ Solid::nem_{acd} - b \\ (source nem_{acd} - b) = \\ Solid::nem_{acd} - b \\ (source nem_{acd} - b) = \\ Solid::nem_{acd} - b \\ (source nem_{acd} - b) = \\ Solid::nem_{acd} - b \\ (source nem_{acd} - b) = \\ Solid::nem_{acd} - b \\ (source nem_{acd} - b) = \\ Solid::nem_{acd} - b \\ (source nem_{acd} - b) = \\ Solid::nem_{acd} - b \\ (source nem_{acd} - b) = \\ Solid::nem_{acd} - b \\ (source nem_{acd} - b) = \\ Solid::nem_{acd} - b \\ (source nem_{acd} - b) = \\ Solid::nem_{acd} - b \\ (source nem_{acd} - b) = \\ Solid::nem_{acd} $	[M_m_attractors, M_m_attractors, M_m_accounts, M_m_accounts, M_m_accounts, M_m_attractors, M_m_accounts, M_m_attractors, M_m_m_attractors, M_m_attractors, M_	$\label{eq:constraint} \begin{split} & \operatorname{interal_happens_leftere} = \underset{\substack{ \operatorname{intermatrix} \operatorname{higher} \operatorname{higher} \\ \ \ \ \ \ \ \ \ \ \ \ \ \$	consistent_presention actions threads location-laid sequenced before additional-productional with data-dependency control-dependency of modification-order sc = well_brands_brands_brands_actions threads_brands_br
$ \begin{array}{l} (\operatorname{memory}_{\mathcal{A}}\operatorname{rdefr}\left(A^{(1)}(M^{(1)}_{\mathcal{A}},M^{(1)}_{\mathcal{A}},\dots,\operatorname{mem}_{\mathcal{A}}\mathcal{A}_{\mathcal{A}}^{-}\right) =\\ \operatorname{Soure}_{\mathcal{A}}\operatorname{memory}_{\mathcal{A}}\operatorname{rdefr}\left(A^{(1)}(M^{(1)}_{\mathcal{A}},\dots,\operatorname{mem}_{\mathcal{A}}\mathcal{A}_{\mathcal{A}}^{-})\right) =\\ \operatorname{Soure}_{\mathcal{A}}\operatorname{soure}_{\mathcal{A}}\operatorname{rdefr}\left(F^{(1)}_{\mathcal{A}}(M^{(1)}_{\mathcal{A}},\dots,\operatorname{mem}_{\mathcal{A}}\mathcal{A}_{\mathcal{A}}^{-})\right) =\\ \operatorname{Soure}_{\mathcal{A}}\operatorname{soure}_{\mathcal{A}}\operatorname{rdefr}\left(F^{(1)}_{\mathcal{A}}(M^{(1)}_{\mathcal{A}},\dots,\operatorname{mem}_{\mathcal{A}}\mathcal{A}_{\mathcal{A}}^{-})\right) =\\ \operatorname{Soure}_{\mathcal{A}}\operatorname{soure}_{\mathcal{A}}\operatorname{rdefr}\left(F^{(1)}_{\mathcal{A}}(M^{(1)}_{\mathcal{A}},\dots,\operatorname{mem}_{\mathcal{A}}\mathcal{A}_{\mathcal{A}}^{-})\right) =\\ \operatorname{Soure}_{\mathcal{A}}\operatorname{soure}_{\mathcal{A}}\operatorname{rdefr}\left(F^{(1)}_{\mathcal{A}}(M^{(1)}_{\mathcal{A}},\dots,\operatorname{mem}_{\mathcal{A}}\mathcal{A}_{\mathcal{A}}^{-})\right) =\\ \operatorname{Soure}_{\mathcal{A}}\operatorname{soure}_{\mathcal{A}}\operatorname{rdefr}\left(F^{(1)}_{\mathcal{A}}(M^{(1)}_{\mathcal{A}},\dots,\operatorname{mem}_{\mathcal{A}}\mathcal{A}_{\mathcal{A}}^{-})\right) \\\\ \left(\operatorname{Soure}_{\mathcal{A}}\operatorname{soure}_{\mathcal{A}}\operatorname{rdefr}\left(F^{(1)}_{\mathcal{A}}(M^{(1)}_{\mathcal{A}},\dots,\operatorname{mem}_{\mathcal{A}}\mathcal{A}_{\mathcal{A}}^{-})\right) \\\\ \left(\operatorname{Soure}_{\mathcal{A}}\operatorname{soure}_{\mathcal{A}}\operatorname{rdefr}\left(F^{(1)}_{\mathcal{A}}(M^{(1)}_{\mathcal{A}},\dots,\operatorname{mem}_{\mathcal{A}}\mathcal{A}_{\mathcal{A}}^{-})\right) \\\\ \left(\operatorname{Soure}_{\mathcal{A}}\operatorname{soure}_{\mathcal{A}}\operatorname{rdefr}\left(F^{(1)}_{\mathcal{A}}(M^{(1)}_{\mathcal{A}},\dots,\operatorname{mem}_{\mathcal{A}}\mathcal{A}_{\mathcal{A}}^{-})\right) \\\\ \left(\operatorname{Soure}_{\mathcal{A}}\operatorname{soure}_{\mathcal{A}}\operatorname{soure}_{\mathcal{A}}\operatorname{soure}_{\mathcal{A}},\dots, \left(\operatorname{Soure}_{\mathcal{A}}\operatorname{soure}_{\mathcal{A}}^{-}\right)\right) \\\\ \left(\operatorname{Soure}_{\mathcal{A}}\operatorname{soure}_{\mathcal{A}}\operatorname{soure}_{\mathcal{A}}\operatorname{soure}_{\mathcal{A}}\operatorname{soure}_{\mathcal{A}}^{-}\right) \\\\ \left(\operatorname{Soure}_{\mathcal{A}}\operatorname{soure}_{\mathcal{A}}\operatorname{soure}_{\mathcal{A}}\operatorname{soure}_{\mathcal{A}}\operatorname{soure}_{\mathcal{A}}^{-}\right) \\\\ \left(\operatorname{Soure}_{\mathcal{A}}\operatorname{soure}_{\mathcal{A}}\operatorname{soure}_{\mathcal{A}}\operatorname{soure}_{\mathcal{A}}\operatorname{soure}_{\mathcal{A}}^{-}\right) \\\\ \left(\operatorname{Soure}_{\mathcal{A}}\operatorname{soure}_{\mathcal{A}}\operatorname{soure}_{\mathcal{A}}\operatorname{soure}_{\mathcal{A}}\operatorname{soure}_{\mathcal{A}}^{-}\right) \\\\ \left(\operatorname{Soure}_{\mathcal{A}}\operatorname{soure}_{\mathcal{A}}so$	[M_m_attractors, M_m_attractors, M_m_acquiring, M_m_acquiring, M_m_attractors, M_m_acquiring, M_m_attractors, M_m_acquiring, M_m_attractors, M_m_acquiring, M_m_attractors, M_m_m_attractors, M_m_attractors, M_m_m_attractors, M_m_m_attractors, M_m_m_attractors, M_m_m_attractors, M_m_m_m_m_m_m_m_m_m_m_m_m_m_m_m_m_m_m_m	$\label{eq:constraint} \begin{split} & \operatorname{inter-all-read-hypers.defore} = & & \operatorname{inter-all-approx.defore} = \\ & \operatorname{inter-all-read-approx.defore} & & & = \\ & \operatorname{inter-all-read-approx.defore} & & & & = \\ & & \operatorname{inter-all-read-approx.defore} & & & & \\ & & & \operatorname{inter-all-approx.defore} & & & & \\ & & & & \operatorname{inter-all-approx.defore} & & & & \\ & & & & & \operatorname{inter-all-approx.defore} & & & & \\ & & & & & & \operatorname{inter-all-approx.defore} & & & \\ & & & & & & \operatorname{inter-all-approx.defore} & & & \\ & & & & & & \operatorname{inter-all-approx.defore} & & & \\ & & & & & & & \\ & & & & & & & $	consister_presention actions threads location kind sequenced before additional-prochonologic with data-dependency control dependency of medication-order m well_present_interaction actions threads actions kind sequenced before additional-prochonologic with data-dependency control dependency or (consister_processing actions threads location kind sequenced before additional-prochonologic with data-dependency control dependency or (location actions threads location kind sequenced before additional-prochonologic with data-dependency control dependency or (location actions threads location kind sequenced before additional-prochonologic with data-dependency control dependency or (location actions appendence) actions and sequence dependence control dependency control dependency or molification-order in let dependency order before additional-prochonologic with backs location kind sequences before additional prochonologic with data-dependency control dependency modification-order in let dependency control dependency modification-order material data-dependency control dependency or molification-order in let dependency control dependency modification-order material data-dependency control dependency or molification-order in let dependency control dependency modification-order material data-dependency control dependency or molification-order in let single-dependency actions and expenses data data-dependency control dependency or molification-order in let single-dependency actions and expenses data additional dependence additional dependence data additional dependence and additional dependence and additional dependence additional dependence additional dependence additional dependence additional dependence and additional dependence additional dependence additional depende
	$ \begin{split} & \left[\left[h_{-} \text{urt}, \text{XUR}, M_{0} \text{urt}, \text{XUR}, M_{0} \text{urt}, \text{XUR}, M_{0} \text{urt}, \text{UN}_{-} \text{urc}, \text{URL}, M_{0} \text{urt}, \text{UN}_{-} \text{urc}, \text{UN}$	Intra Altrack Mappen Lefter =	consistent_presention actions threads location-laid sequenced before additional-productional with data-dependency control-dependency of modification-order sc = well_brands_brands_brands_actions threads_brands_br
$ \begin{array}{l} \left(\operatorname{snemser}_{j \in \operatorname{odd}} \left(A^{(j)_{M} (j)_{M}} (j, A^{(j)_{M}} (j, $	$ \begin{split} & \left[\left[h_{i} \rightarrow \text{ITA}_{i} \text{XER}_{i} \text{XER}_{i} \text{Average} = \text{well_formal_threads} = \\ & \left[h_{i} \rightarrow \text{IT}_{i} \text{XER}_{i} \text{Average}_{i} + \text{well_formal_threads} = \\ & \left[h_{i} = h_{i} \text{XER}_{i} \text{Average}_{i} \text{Average}_{i} + h_{i} + h_{i$	inter.ubroad.happens.defere = inter.ubroad.happens.defere = inter.ubroad.happens.defere = inter.ubroad.happens.defere = (for (consistent_presention actions threads location laid sequenced before additional-gendenoised with data-dependency control dependency of medification-order is well_formul_linesis actions threads location-laid sequenced before additional-gendenoised with data-dependency control dependency or in the sequence = relative sequence = hypothetical relative sequences defore additional-gendenoised with data-dependency control dependency or in the sequence = relative sequence = hypothetical relative sequences defore additional-gendenoised with data-dependency control dependency control dependency control dependency or in the sequence = relative sequence = hypothetical relative sequences defore additional-gendenoised with data-dependency control dependency control dependency or in the sequence = relative sequence = hypothetical relative sequences defore additional-gendenoised with data-dependency control dependency or in the sequence = relative sequence = hypothetical relative sequences are actions threads holds and dependency control dependency or in the sequence = relative sequence = hypothetical relative sequences are actions threads holds additional-gendenoised with data-dependency control dependency or in the sequences of the sequences defore additional sequences defore additional sequences control dependency relatives well and the sequences or sequences in the sequences of the sequences defore additional sequences defore additional sequences control dependency control dependency control dependency in the section has a sequence defore additional sequences or sequences are advected before additional sequences defore additional sequences defore additional sequences or sequences control dependency control dependency control dependency or sequences are advected before in the sequences of setters = wells advected before additional sectors has assessed before additional sectors has appendent befor
	$ \begin{split} & \left[\left[h_{-} \text{urt}, \text{XUR}, M_{0} \text{urt}, \text{XUR}, M_{0} \text{urt}, \text{XUR}, M_{0} \text{urt}, \text{UR}, \text{UN}_{-} \text{urt}, \text{UR}_{-} \text{urt}, $	Intra Altrack Mappen Lefter =	consister_presention actions threads location laid sequenced before additional synchronized with data dependency control dependency of m collication order is real-presention actions threads location laid sequenced before additional synchronized with data dependency control dependency or in the dependency or antibia sequence dependency actions and sequenced before additional synchronized with data dependency control dependency or in the dependency or antibia sequence dependency actions and sequenced before additional synchronized with data dependency control dependency or in the dependency or antibia sequence dependency actions and sequenced before additional synchronized with data dependency control dependency or the in the dependency or antibia sequence actions threads locations have actions threads locations and sequences before additional synchronized with data dependency control dependency or in the dependency or antibia sequence actions and sequence dependency actions and sequence before additional synchronized with data dependency control dependency or in the dependency or antibia sequence dependency actions and sequence devices additional synchronized with data dependency control dependency or in the dependency or antibia sequence devices additional synchronized with data dependency control dependency or in the single sequence or single sequence devices additional synchronized with data dependency control dependency or in the single sequence or devices and sequence device additional synchronized with data dependency control dependency modification order is sequences additional synchronized with data dependency control dependency and dependency modification order has dependency and dependency and depe
	$ \begin{split} & \left[\ h_{i} = \text{trans.trans.} M_{i} = \text{trans.trans.trans.} M_{i} = trans.tr$	$\begin{aligned} & intra-diagonal-being - intra-diagonal$	consistent_devention actions threads heating squared before additional productional with data dependency control dependency of modification order m =
$ \begin{array}{l} \left(\operatorname{memory}_{\mathcal{A}} \operatorname{mem}_{\mathcal{A}} \mathcal{A}(\Lambda) \\ \operatorname{Solid} & \operatorname{solid} (\Gamma_{\mathcal{A}} \mathcal{A}(\Lambda), -1) = \operatorname{Solid} (\Lambda) \\ \operatorname{Solid} & \operatorname{solid} (\Gamma_{\mathcal{A}} \mathcal{A}(\Lambda), -1) = \operatorname{Solid} (\Lambda) \\ \operatorname{Solid} & \operatorname{Solid} (\Gamma_{\mathcal{A}} \mathcal{A}(\Lambda), -1) = \operatorname{Solid} (\Lambda) \\ \operatorname{Solid} & \operatorname{Solid} (\Gamma_{\mathcal{A}} \mathcal{A}(\Lambda), -1) = \operatorname{Solid} (\Lambda) \\ \operatorname{Solid} & \operatorname{Solid} (\Gamma_{\mathcal{A}} \mathcal{A}(\Lambda), -1) = \operatorname{Solid} (\Lambda) \\ \operatorname{Solid} & \operatorname{Solid} (\Gamma_{\mathcal{A}} \mathcal{A}(\Lambda), -1) = \operatorname{Solid} (\Lambda) \\ \operatorname{Solid} & \operatorname{Solid} (\Gamma_{\mathcal{A}} \mathcal{A}(\Lambda), -1) = \operatorname{Solid} (\Lambda) \\ \operatorname{Solid} & \operatorname{Solid} (\Gamma_{\mathcal{A}} \mathcal{A}(\Lambda), -1) = \operatorname{Solid} (\Lambda) \\ \operatorname{Solid} & \operatorname{Solid} (\Gamma_{\mathcal{A}} \mathcal{A}(\Lambda), -1) = \operatorname{Solid} (\Lambda) \\ \operatorname{Solid} & \operatorname{Solid} (\Gamma_{\mathcal{A}} \mathcal{A}(\Lambda), -1) = \operatorname{Solid} (\Lambda) \\ \operatorname{Solid} & \operatorname{Solid} (\Gamma_{\mathcal{A}} \mathcal{A}(\Lambda), -1) = \operatorname{Solid} (\Lambda) \\ \operatorname{Solid} & \operatorname{Solid} (\Gamma_{\mathcal{A}} \mathcal{A}(\Lambda), -1) = \operatorname{Solid} (\Lambda) \\ \operatorname{Solid} & \operatorname{Solid} (\Gamma_{\mathcal{A}} \mathcal{A}(\Lambda), -1) = \operatorname{Solid} (\Lambda) \\ \operatorname{Solid} & \operatorname{Solid} (\Gamma_{\mathcal{A}} \mathcal{A}(\Lambda), -1) = \operatorname{Solid} (\Lambda) \\ \operatorname{Solid} & \operatorname{Solid} (\Gamma_{\mathcal{A}} \mathcal{A}(\Lambda), -1) = \operatorname{Solid} (\Lambda) \\ \operatorname{Solid} & \operatorname{Solid} (\Gamma_{\mathcal{A}} \mathcal{A}(\Lambda), -1) = \operatorname{Solid} (\Lambda) \\ \operatorname{Solid} & \operatorname{Solid} (\Lambda) \\ \\ \operatorname{Solid} & \operatorname{Solid} (\Lambda) \\ \\ \operatorname{Solid} & \operatorname{Solid} (\Lambda) \\ \\ \operatorname$	$ \begin{split} & \left[\left[h_{-} \text{urt}_{-} u$	inter.ukraukhappens.deter = inter.ukraukhappens.deter = inter.ukraukhappens.deter = inter.ukraukhappens.deter = (<u>sementation</u> , ; <u>sementation</u> ;) in (<u>sementation</u> ; <u>sementation</u> ;) in (<u>sementation</u> ; <u>sementation</u> ;) in (<u>sementation</u> ; <u>sementation</u> ;) interture sementation; (<u>sementation</u> ; <u>sementation</u> ; (<u></u>	$\begin{aligned} & consister_exercise actions threads hockins hist sequenced before additional synchronized with data dependency control dependency of m collications order m c = control dependency of the collication order m control dependency of the collication order additional synchronized with data dependency control dependency control dependency of the collication order m control dependency of the collication order control dependency of the collication order control dependency of the collication order m control dependency of the collication order control dependency of the collication order control dependency of the collication order m control dependency of the collication order control dependency of the control dependency of the control dependency $
	$ \begin{split} & \left[\ h_{i} = \text{trans.term.} M h_{i} = \text{trans.term.term.term.} M h_{i} = \text{trans.term.term.} M h_{i} = \text{trans.term.term.term.} M h_{i} = \text{trans.term.term.term.} M h_{i} = \text{trans.term.term.term.} M h_{i} = \text{trans.term.term.term.term.term.} M h_{i} = \text{trans.term.term.term.} M h_{i} = trans.term.term.term.term.term.term.term.term$	$\begin{aligned} & \text{intra-linequal-topone-local segmentation}_{i} & = \\ & \text{intra-linequal-topone-local segmentation}_{i} & = \\ & \text{intra-linequal-topone-local}_{i} & = \\ & \text{(attractive set)}_{i} & = \\ & (\text{(attractive set)}_{i}) & = \\ & \text{(attractive set)}_{i} & = \\ & \text{(attractive set)}_{$	$\begin{aligned} & constant_period by the sequence of before additional spectromical with data dependency control dependency of modification order m = \\ while the sequence of the sequence of the sequence additional spectromical with data dependency control dependency of modification order m = \\ while the sequence of the sequence of the sequence additional spectromical with data dependency control dependency of modification order m = \\ while the sequence of the sequence of the sequence additional spectromical with data dependency control dependency of modification order m = \\ while the sequence of the sequence of the sequence additional spectromical with data dependency control dependency of modification order m = \\ while the sequence of the sequence$
$ \begin{array}{l} \left(\operatorname{memory}_{\mathcal{A}} \operatorname{mem}_{\mathcal{A}} \mathcal{A}(\Lambda) \\ \operatorname{Solid} & \operatorname{solid} (\Gamma_{\mathcal{A}} \mathcal{A}(\Lambda), -1) = \operatorname{Solid} (\Lambda) \\ \operatorname{Solid} & \operatorname{solid} (\Gamma_{\mathcal{A}} \mathcal{A}(\Lambda), -1) = \operatorname{Solid} (\Lambda) \\ \operatorname{Solid} & \operatorname{Solid} (\Gamma_{\mathcal{A}} \mathcal{A}(\Lambda), -1) = \operatorname{Solid} (\Lambda) \\ \operatorname{Solid} & \operatorname{Solid} (\Gamma_{\mathcal{A}} \mathcal{A}(\Lambda), -1) = \operatorname{Solid} (\Lambda) \\ \operatorname{Solid} & \operatorname{Solid} (\Gamma_{\mathcal{A}} \mathcal{A}(\Lambda), -1) = \operatorname{Solid} (\Lambda) \\ \operatorname{Solid} & \operatorname{Solid} (\Gamma_{\mathcal{A}} \mathcal{A}(\Lambda), -1) = \operatorname{Solid} (\Lambda) \\ \operatorname{Solid} & \operatorname{Solid} (\Gamma_{\mathcal{A}} \mathcal{A}(\Lambda), -1) = \operatorname{Solid} (\Lambda) \\ \operatorname{Solid} & \operatorname{Solid} (\Gamma_{\mathcal{A}} \mathcal{A}(\Lambda), -1) = \operatorname{Solid} (\Lambda) \\ \operatorname{Solid} & \operatorname{Solid} (\Gamma_{\mathcal{A}} \mathcal{A}(\Lambda), -1) = \operatorname{Solid} (\Lambda) \\ \operatorname{Solid} & \operatorname{Solid} (\Gamma_{\mathcal{A}} \mathcal{A}(\Lambda), -1) = \operatorname{Solid} (\Lambda) \\ \operatorname{Solid} & \operatorname{Solid} (\Gamma_{\mathcal{A}} \mathcal{A}(\Lambda), -1) = \operatorname{Solid} (\Lambda) \\ \operatorname{Solid} & \operatorname{Solid} (\Gamma_{\mathcal{A}} \mathcal{A}(\Lambda), -1) = \operatorname{Solid} (\Lambda) \\ \operatorname{Solid} & \operatorname{Solid} (\Gamma_{\mathcal{A}} \mathcal{A}(\Lambda), -1) = \operatorname{Solid} (\Lambda) \\ \operatorname{Solid} & \operatorname{Solid} (\Gamma_{\mathcal{A}} \mathcal{A}(\Lambda), -1) = \operatorname{Solid} (\Lambda) \\ \operatorname{Solid} & \operatorname{Solid} (\Gamma_{\mathcal{A}} \mathcal{A}(\Lambda), -1) = \operatorname{Solid} (\Lambda) \\ \operatorname{Solid} & \operatorname{Solid} (\Gamma_{\mathcal{A}} \mathcal{A}(\Lambda), -1) = \operatorname{Solid} (\Lambda) \\ \operatorname{Solid} & \operatorname{Solid} (\Lambda) \\ \\ \operatorname{Solid} & \operatorname{Solid} (\Lambda) \\ \\ \operatorname{Solid} & \operatorname{Solid} (\Lambda) \\ \\ \operatorname$	$ \begin{split} & \left[\left[h_{i} \rightarrow \text{TT}_{i} \text{XER}_{i} \text{M}_{i} \text{ATEAXER}_{i} \text{M}_{i} \text{ACQUERE, M}_{i} ACQUERE$	$\begin{aligned} & intra-diagonal-being - intra-mail-integrations into the second segment data - integration of the second segment data - integration - integration (- integration - in$	<pre>cmainter_encution actions threads hockins hist sequenced before additional productioned with data dependency control dependency of modification order m = """ """ """ """ """ """ """ """ """ """ """ """ ""</pre>
	$\begin{split} & \left(\ h_{n-\text{structure}}^{-1} \text{Area}_{n-\text{structure}}^{-1} \text$	$\begin{aligned} & \text{intra-langement-lefter = \frac{\text{intermating segment lefter }}{\text{inter-langement lefter }} = \frac{\text{intermating }}{\text{inter-langement lefter }} = \frac{\text{intermating }}{\text{intermating }} \\ & \text{for } = \frac{\text{intermating }}{\text{intermating }} = \frac{\text{intermating }}{\text{intermating }} \\ & (\frac{\text{intermating }}{\text{intermating }} = \frac{\text{intermating }}{\text{intermating }} \right) \text{intermating }} \\ & (\frac{\text{intermating }}{\text{intermating }} = \frac{\text{intermating }}{\text{intermating }} \right) \text{intermating }} \\ & consistent inter direct discussed lefter = consistent inter_intered_langement lefter = $	consistent_exercise scient threads heating handle h
$ \begin{cases} (\operatorname{sermer}_{j \to defr} (A^{(V)MC-JAM} - \ldots, \operatorname{serm}_{j \to d} -) = \\ Solit: \operatorname{serm}_{j \to d} (A^{(j)} -) = \\ Solit: \operatorname{serm}_{j \to d} (A^{(j)} -) = \\ Solit: \operatorname{serm}_{j \to d} (A^{(j)} -) = \\ Solit: \operatorname{serm}_{j \to d} (A^{(j)} -) = \\ (\operatorname{sermer}_{j \to d} -) = \\ Solit: \operatorname{serm}_{j \to d} (A^{(j)} -) = \\ Solit: \operatorname{serm}_{j \to d} (A^{(j)} -) = \\ Solit: \operatorname{serm}_{j \to d} (A^{(j)} -) = \\ (\operatorname{serm}_{j \to d} -) = \\ (\operatorname{sermer}_{j \to d} -) = \\ (\operatorname{serm}_{j \to d} -) \\ (\operatorname{serm}_{j \to d} -) = \\ (\operatorname{serm}_{j \to d} -) $	$ \begin{split} & \left(M_{0} \text{-untractions}, M_{0} \text{-untractions}, M_{0} \text{-ucc}_{untractions}, M_{0} \text{-ucc}_{unt$	$\begin{aligned} \inf_{x \in A} \operatorname{Irres} \operatorname{Lingence}_{x \in X} \operatorname{Irres}_{x \in $	<pre>cmaintary_exercision actions threads heatings have additional productional with the dependency control dependency of modification order x = with_control Actions have approach the thread heating approach after a displaced production of with the dependency control dependency of modification order x is with actions have approach the thread heating approach after a displaced production of with the dependency control dependency of modification order x is with actions have approach the thread heating approach after additional approximated with the dependency control dependency of modification order x is is approach at the action have approach at the action have approach after additional approximated with the dependency control dependency of modification order x is is approach at the action have approach at the action have approach approac</pre>
$ \begin{aligned} & \left[(\operatorname{serem} \gamma_{inc} \operatorname{selec} (\operatorname{VOME}_{i,k} \operatorname{DAM}_{i} \dots \operatorname{seme}_{inc} \operatorname{selec}_{i}) - \\ & \operatorname{Sourd}_{i} \operatorname{seme}_{i} \operatorname{selec} (\operatorname{VOME}_{i} \operatorname{VOME}_{i} \dots \operatorname{seme}_{inc} \operatorname{selec}_{i}) - \\ & \operatorname{Sourd}_{i} \operatorname{seme}_{i} \operatorname{selec} (\operatorname{VOME}_{i} \dots \operatorname{seme}_{i} \operatorname{seme}_{i}) - \\ & \operatorname{Sourd}_{i} \operatorname{seme}_{i} \operatorname{seme}$	$ \begin{split} & \left(M_{n-\text{MELSERS}}, M_$	$\begin{aligned} & Intra-diagonal-being - intra-main degrees being - intra-main degree$	consister, exercise actions threads hockets hid sequence before additional spectromical with data dependency control dependency of modification order m = with furned, lational actions hidrogeneous deform additional spectromical with data dependency control dependency of modification order m is with threads actions threads hockets hidrogeneous deformation and the data dependency control dependency of modification order m is the production with threads hockets hidrogeneous deformation and the data dependency control dependency of modification order m is the production with threads hockets hidrogeneous deformation and the data dependency control dependency of modification order m is the production with threads hockets hidrogeneous deformation and the data dependency control dependency of modification order m is the production with threads hockets hidrogeneous deformation and the data dependency control dependency of modification order m is the production with threads hockets hidrogeneous deformation and the data dependency control dependency of modification order is and expenses deformation and the data dependency control dependency of modification order is and expenses deformation and the data dependency control dependency of modification order is and expenses deformation and the data dependency control dependency of modification order is and expenses deformation and the data dependency control dependency modification order is and expenses of modification order is and expenses deformation and the data dependency control dependency modification order is and expenses deformation and the data dependency control dependency modification order is and expenses deformation and the data dependency control dependency modification order is and expenses deformation and the data dependency control dependency or modification order is and expenses deformation and the data dependency control dependency or modification order is a dependency or deformation modification order is a dependency or deformation modification orde
$ \begin{cases} (\operatorname{semacry}_{\mathcal{A}} \operatorname{cder} (\operatorname{ACMME}_{\mathcal{A}} \operatorname{ALM}) \dots \operatorname{semagrad}_{\mathcal{A}}) = \\ \operatorname{South} \operatorname{macry}_{\mathcal{A}} \operatorname{ACM} (\operatorname{ACMME}_{\mathcal{A}} \operatorname{ALM}) \dots \operatorname{macry}_{\mathcal{A}} = \\ \operatorname{South} \operatorname{macry}_{\mathcal{A}} \operatorname{ACM} (\operatorname{ACMME}_{\mathcal{A}} \operatorname{ALM}) \dots \operatorname{macry}_{\mathcal{A}} = \\ \operatorname{South} \operatorname{macry}_{\mathcal{A}} \operatorname{ACM} (\operatorname{ACMME}_{\mathcal{A}} \operatorname{ALM}) \dots \operatorname{macry}_{\mathcal{A}} = \\ \operatorname{South} \operatorname{macry}_{\mathcal{A}} \operatorname{ACM} (\operatorname{ACMME}_{\mathcal{A}} \operatorname{ALM}) = \\ \operatorname{South} \operatorname{macry}_{\mathcal{A}} \operatorname{ACM} (\operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM}) = \\ \operatorname{South} \operatorname{ACMME}_{\mathcal{A}} \operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM} (\operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM}) = \\ \operatorname{South} \operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM} \operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM} (\operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM}) = \\ \operatorname{South} \operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM} \operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM} (\operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM}) = \\ \operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM} \operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM} (\operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM}) = \\ \operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM} \operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM} (\operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM}) = \\ \operatorname{ACMME}_{\mathcal{A}} \operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM} (\operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM}) = \\ \operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM} \operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM} (\operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM}) = \\ \operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM} \operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM} (\operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM}) = \\ \operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM} \operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM} \operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM} (\operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM}) = \\ \\ \operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM} \operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM} = \\ \\ \operatorname{ACMME}_{\mathcal{A}} \operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM} \operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM} = \\ \\ \operatorname{ACMME}_{\mathcal{A}} \operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM} = \\ \\ \operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM} \operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM} = \\ \\ \operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM} = \\ \\ \operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM} = \\ \\ \operatorname{ACMME}_{\mathcal{A} \operatorname{ACM} = \\ \\ \operatorname{ACMME}_{\mathcal{A}} \operatorname{ACM} = \\ \\ \\ \operatorname{ACMME}$	$ \begin{split} & \left(M_{n-\text{MELSERS}}, M_{n-\text{MELSERS}}, M_{n-\text{MEQCHER}}, M_$	$\begin{aligned} \inf_{x \in A} \operatorname{Irres} \operatorname{Lingence}_{x \in X} \operatorname{Irres}_{x \in $	<pre>mattermachine actions threak location kind sequences before additional quoteconical with data dependency control dependency if medification over x = = with_concu_threak actions threak location kind sequences down additional quoteconical with data dependency control dependency if medification over x = with_concu_threak actions threak location kind sequences down additional quoteconical with data dependency control dependency if medification over x = with_concu_threak actions threak location kind sequences down additional quoteconical with data dependency control dependency if medification over x = with_concu_threak actions threak location kind sequences down additional quoteconical with data dependency control dependency if medification over x = with_concu_threak actions threak location kind sequences down additional quoteconical with data dependency control dependency if medification over x = with_concu_threak actions threak location kind sequences down additional quoteconical with data dependency control dependency if medification over x = with_concu_threak actions threak location kind sequences down additional quoteconical with data dependency control dependency if medification over x = with_concu_threak actions the data dependency control dependency if medification over x = with sequences down additional quoteconical with data dependency control dependency if medification over x = with sequences down additional quoteconical with data dependency control dependency if medification over x = with sequences down additional quoteconical with data dependency control dependency if medification over x = with sequences down additional quoteconical with data dependency control dependency if medification over x = with sequences down additional quoteconical with data dependency control dependency if medification over x = with sequences down additional quoteconical with data dependency control dependency if medification over x = with sequences down additional down additional dependence down additional dependency d</pre>
$ \begin{aligned} & \left[(\operatorname{serem} \gamma_{inc} \operatorname{selec} (\operatorname{VOME}_{i,k} \operatorname{DAM}_{i} \dots \operatorname{seme}_{inc} \operatorname{selec}_{i}) - \\ & \operatorname{Sourd}_{i} \operatorname{seme}_{i} \operatorname{selec} (\operatorname{VOME}_{i} \operatorname{VOME}_{i} \dots \operatorname{seme}_{inc} \operatorname{selec}_{i}) - \\ & \operatorname{Sourd}_{i} \operatorname{seme}_{i} \operatorname{selec} (\operatorname{VOME}_{i} \dots \operatorname{seme}_{i} \operatorname{seme}_{i}) - \\ & \operatorname{Sourd}_{i} \operatorname{seme}_{i} \operatorname{seme}$	$ \begin{split} & \left(M_{n-\text{structure}}, M_{n-structure$	$\begin{aligned} & intraching productions intraching productions interview of the production interview of the production interview of the productions interview of the productionsinterview of the productions$	<pre>instant_ancies action threads tacking instant devices bid approved bider additional production with the dependency of medification order is = """."</pre>
	$ \begin{split} & \left(M_{n-\text{structure}}, M_{n-structure$	<pre>inter.strendstagenes.deter =:</pre>	<pre>emission_action action threads back addition in the back back addition in the back addition in the back addition in the back bac</pre>
$ \begin{cases} (\operatorname{semacry}_{\mathcal{A}} \operatorname{cder} (\operatorname{AVDME}_{\mathcal{A}} \operatorname{ADM} : \operatorname{mem}_{\mathcal{A}} \operatorname{cder}), \\ (\operatorname{semacry}_{\mathcal{A}} \operatorname{cder} (\operatorname{AVDME}_{\mathcal{A}} \operatorname{ADM} : \operatorname{mem}_{\mathcal{A}} \operatorname{cder}), \\ (\operatorname{semacr}_{\mathcal{A}} \operatorname{cder} (\operatorname{AVDME}_{\mathcal{A}} \operatorname{cder} (\operatorname{adm} : \operatorname{mem}_{\mathcal{A}} \operatorname{cder}), \\ (\operatorname{South} : \operatorname{mem}_{\mathcal{A}} \operatorname{cder} (\operatorname{Fiscle} : \operatorname{mem}_{\mathcal{A}} \operatorname{cder}), \\ (\operatorname{South} : \operatorname{mem}_{\mathcal{A}} \operatorname{cder} (\operatorname{Fiscle} : \operatorname{mem}_{\mathcal{A}} \operatorname{cder}), \\ (\operatorname{South} : \operatorname{mem}_{\mathcal{A}} \operatorname{cder} (\operatorname{Fiscle} : \operatorname{mem}_{\mathcal{A}} \operatorname{cder}), \\ (\operatorname{South} : \\ (\operatorname{South} :), \\ (\operatorname{South} :), \\ (\operatorname{South} :), \\ (\operatorname{South} :), \\ (Sout$	$ \begin{array}{l} \left(M_{n-\text{HILARDEN}}, M_$	<pre>inter.structure_inter</pre>	<pre>mmisstar_accelute action therea. Notice had aqueous badre additional genome badre additional geno</pre>
$ \begin{cases} (\operatorname{serem} \gamma_{inc}\operatorname{refer} (A^{int}(\operatorname{Sint} i = \operatorname{sens} d) \land \\ (\operatorname{serem} \gamma_{inc} \operatorname{refer} (A^{int}(\operatorname{Sint} i = \operatorname{sens} d) \land \\ (\operatorname{serem} \gamma_{inc} \operatorname{refer} (A^{int}(\operatorname{Sint} i = \operatorname{sens} d) \land \\ (\operatorname{Sint} i = \operatorname{sens} d) \land \\ (\operatorname{sene} \gamma_{inc} \operatorname{sens} d) \land \\ (\operatorname{Sint} i = \operatorname{sens} d) \circ \\ ($	$ \begin{split} & \left(M_{n-\text{structure}}, M_{n-structure$	$ \begin{aligned} & \text{intra-line large matching , } & \text{intra-line large matching , } & \\ & \text{intra-line large matching , } & \\ & \text{intra-line large matching , } & \\ & (\text{denomination } , \text{denomination }) & \\ & (\text{denomination } , \text{denomination }) & \\ & (\text{denomination } , \text{denomination }) & \\ & (\text{denomination } , \text{denomination }) & \\ & (\text{denomination } , \text{denomination }) & \\ & (\text{denomination } , \text{denomination }) & \\ & (\text{denomination } , \text{denomination }) & \\ & (\text{denomination } , \text{denomination }) & \\ & (\text{denomination } , \text{denomination }) & \\ & (\text{denomination } , \text{denomination }) & \\ & (\text{denomination } , \text{denomination }) & \\ & (\text{denomination } $	<pre>senter</pre>

The full model

	is_store a = case a of Store $\to T \parallel . \to F$		
$a \xrightarrow{r} b = (a, b) \in r$	is fine $a = case a$ of FENCE $\dots \rightarrow T \parallel \rightarrow F$		visible_side_stfice_set actions threads location-kind sequenced before additional-genchronized-with data-dependency tappens before =
$a r b = (a, b) \in r$	BUELLE 2 = Case 2 OF FENCE	rs_element rs_head > = same_thread > rs_head V is_atomic_rmw >	(20 < https://doi.org/10.001/2000 (2010) - 20 and visition.public.p
	is_lock_or_unlock a = is_lock a V is_unlock a	$release_{sequence} = \lambda_{seq} \xrightarrow{nlass-squares} b =$	visite-sequenced_electrical = visite-sequence_d_electrical vescherd b =
$a \stackrel{q}{\not \rightarrow} b = (a, b) \notin r$	is_atomic_action 2 =	is at atomic location $b \land$ is $a_{ad} \land f_{ad} \land f_{ad}$	(c may head minimum e.s.)
$\xrightarrow{r} = r$	is_atomic_load a∨is_atomic_store a∨is_atomic_rmw a	$(b = a_{sd}) \lor$ (rs_chement $a_{sd} b \land a_{sd}$ -mathing $b \land$	$\langle 1 \rangle = \frac{1}{1000} \left[\frac{1}{10000000000000000000000000000000000$
	is_load_or_store a = is_load a V is_store a	$(Yc. \lambda_{W} \xrightarrow{\text{molfaction-order}} c \xrightarrow{\text{molfaction-order}} b \Longrightarrow$	-((c
$a \stackrel{\prime}{\rightarrow} b \stackrel{a}{\rightarrow} c = a \stackrel{\prime}{\rightarrow} b \wedge b \stackrel{a}{\rightarrow} c$			$\operatorname{avinage} t = (y, \exists x \in x (y = f \cdot x))$
relation_over $s \ rel = {\rm domain} \ rel \ \subseteq \ s \wedge {\rm range} \ rel \ \subseteq \ s$	is_read a = is_atomic_load a∨is_atomic_rmw a∨is_load a	release_sequence_set actions threads location-kind sequenced-before additional-synchronized-with data-dependency control-dependency modification-order =	visibirepresent
$\frac{nl}{ s } = rel \cap (s \times s)$	is write a =	release-asequence: actions threads: location-kind sequenced-before additional-synchronized-with data-dependency control-dependency modification-order a b)	The second
	is_atomic_store a∨is_atomic_rmw a∨is_store a	hypothetical_release_sequence = a hypothetical-inhum-sequence = b =	{vsz.hzd}\u00e4
$ref _s = ref \cap (s \times s)$	is_acquire a =	is_at_atomic_location $b \land ($ $(b = 2) \lor$ motification-unity	else ())
$\frac{nl}{m} _s = rel \cap (s \times s)$	(case memory order a of SOME mem_ord →	$(x_{\mathcal{A}})$ determined $a > b \land a$ and the following $b \land (y \in a)$ modification only $b \rightarrow b \implies b$	
	$(mem_acd \in \{Mo_acquire, Mo_acq_rel, Mo_seq_cst\} \land$	rs_clement a c)))	widek-auspancesd. ideaeffectset actions lurants. location-kind supported softer additionalsetterheide tables additionalsetterheide a
$rel _s = rel \cap (s \times s)$	(is_read a ∨ is_fence a)) ∨ (* 29.8:5 states that consume fences are acquire fences. *) ((mem.ord = Mo_CONSUME) ∧ is_fence. a)	hypothetical_release_sequence_set actions threads location-kind sequenced-before additional-synchronized-with data-dependency control-dependency modification-order =	omierat study francustoire = omierat study francustoire =
$strict_preorder ord = irreflexive ord \land trans ord$	$((mem_ord = Mo_CONSUME) \land is_fence a)$ $\parallel NONE \rightarrow is_lock a)$	hypothetical_release_sequence actions through location-kind sequenced-before additional-synchronized-with data-dependency control-dependency modification-order a b}	$(\forall b \ (is real \ b \land is at non atomic location \ b) \Longrightarrow$
	is.consume a =	synchronizes, with $= 2 \frac{\text{such assiment}}{b} b =$	$\begin{array}{l} (\mathbf{f} \in [\mathbf{z}_{m}, -\mathbf{z}_{m}, \frac{\mathrm{disk} \mathrm{disk} \mathrm{disk} \mathbf{f}}{\mathrm{disk}}, \mathbf{b}) \\ \mathbf{then} (\mathbf{z}_{m}, -\mathbf{z}_{m}, \frac{\mathrm{disk} \mathrm{disk} \mathrm{disk} \mathrm{disk}}{\mathrm{disk}}, \mathbf{b} \wedge \mathbf{z}_{m}, \overset{\mathcal{I}}{\mathrm{disk}}, \mathbf{b}) \end{array}$
total_over s ord = relation_over s ord ∧	is_read a ∧ (memory_order a = SOME MO_CONSUME)	(* = additions/synchronization, from thread create etc. = *) a additions/synchronization → b ∨	ene - (∃z, ≤ − \$))) ∧
$(\forall x \in s. \forall y \in s. x \xrightarrow{ood} y \lor y \xrightarrow{ood} x \lor (x = y))$	is_release a =	$a b \vee b \vee$ (sume_location $a b \land a \in actions \land b \in actions \land ($	(vb. [is read b h is at statistic location b) \Longrightarrow (if $[2[t], vas) \in viable sequences of eider effects. (b' = b))them [(2t], vas) \in viable secure of eider effects.$
strict_total_order_over s and =	(case memory_order ≥ of SOME mem_ord →	(* - matex synchronization - *) (is_umlock $a \land is_lock \land a \stackrel{s}{\longrightarrow} b) \lor$	$(b' = b) \land (\exists c \in vose, c \stackrel{d}{\to} b))$
strict_preorder ord ∧ total_over s ord	$mem_ord \in \{Mo_RELEASE, Mo_ACQ_REL, Mo_SEQ_CST\} \land$ (is_write $a \lor is_fence a$)	(* - release/acquire synchronization - *)	etae - (2z → 4))) ∧
$x \xrightarrow{\text{ord}} p_{\text{pred}} y =$	$ NONE \rightarrow is_{unlock} a$	$(\text{is-release } a \land h \Rightarrow \text{output} b \land -\text{same-thread} a b \land (3c_2 - a \text{is-released}) \land (3c_2 - a \text{is-released}) \land (3c_1 - a \text{is-released}$	$(\mathbf{r}(\mathbf{r}_{i}) \in \vec{A}, \mathbf{r}_{i}) \in \vec{A}, \mathbf{r}_{i}$
$pred \times \wedge \times \xrightarrow{ord} y \wedge \neg (\exists z. pred \ z \wedge \times \xrightarrow{ord} z \xrightarrow{ord} y)$	is_seq_cst a = (memory_order a = SOME MO_SEQ_CST)	(* - fence synchronization - *)	a happen halong b A. happen halo
$x \xrightarrow{\text{out}} y =$	location.kind =	(is frace $2 \wedge is$ release $2 \wedge is$ frace $b \wedge is$ acquire $b \wedge$ ($\exists x, \exists y$, same location $x \neq y \wedge$	\rightarrow $(x - y) \lor x$ multiplication \rightarrow $y) \land$ (* enc CoUNF*)
$x \xrightarrow{\text{odd}} y \land \neg(\exists z. x \xrightarrow{\text{odd}} z \xrightarrow{\text{odd}} y)$	MUTEX NON-ATOMIC	is atomic action $x \land b$ is atomic action $y \land b$ survite $x \land a$ $a = \frac{mpenced delay}{mpenced delay} \times A y = \frac{mpenced delay}{mpenced delay} \ge b \land$	(∀(ria)) ≥ months here, Ye,
well_founded $r = wf r$	ATOMIC	$(\exists x. \times \xrightarrow{hpathetical-observe magnetical} x \xrightarrow{d} y))) \lor$	$c \stackrel{d}{\to} b \wedge$ is write a Λ same location $a \ge h \wedge i_{1}a_{1}$ atomic location b
type_abbrev action_id : string	actions_respect_location_kinds = actions_respect_location_kinds =	[b_fines → Λ, b_schesse → Λ is_stomic_action b ∧ is_schemine b ∧ [3c, sum-Contin w b ∧	$\Rightarrow (c = 2) \vee 2$ and for a solution only $c > 0 \land$
typeanorev acconduit string	$\forall z.$ case location z of SOME $l \rightarrow$	(a. same-octain z b∧ is_stome_action x ∧ is_write x ∧ 	(∀(x)) ≥ www.htm x, x, x, x, x, x, x, x, x, x, x, x, x,
type_abbrev thread_id : string	(case location-kind / of MUTEX → is_lock_or_intlock a NON_ATOMIC → is_load_or_store a	$(\underline{z}, \times \underline{hpatherizations equations}_{\underline{z}} \in \underline{z} \stackrel{d}{\to} b))) \lor$	$c \stackrel{<}{\rightarrow} a_A$ is write $b \land anne-location a \land b \land i_{a+1} -storic location a$
type_abbrev location : string	ATOMIC → is_load_or_store a ∨ is_atomic_action a) NONE → T	(is_atomic_action ≥ ∧ is_arefense ≥ ∧ is_fence b ∧ is_arequire b ∧	inter c medicinentes, b) ∧
		(ik: same of isolation x ∧ is_atomic_action x ∧ x sequence between b ∧	((n k a) c ≤ h, agamin, ruw b → 2
type_abbrev val:string	is_at_location_kind = is_at_location_kind = case location a of	$\begin{array}{c} x & & x \\ (\exists z, z & x & z & x))))) \end{array}$	$\longrightarrow z \longrightarrow b) \land$ ($(i(z,b) \in \stackrel{d}{\rightarrow}, iz, exp.ext b$
memory_order_enum =	Some $I \rightarrow (location-kind I = lk0)$ NONE $\rightarrow F$		\Rightarrow (-is sequent $a \land (\forall x. x \xrightarrow{\kappa})_{h,c}$ is write commutation by $b \Rightarrow x \xrightarrow{\text{modification order}} a)) \lor$
MO_SEQ_CST MO_RELAXED		spectronizes_with_set actions threads location-kind sequenced-before additional-spectronized-with data-dependency or modification-order sc release-sequence hypothetical-release-sequence a by synchronized-with data-dependency control-dependency of modification-order sc release-sequence hypothetical-release-sequence a b)	$a \stackrel{d}{=} \sum_{\lambda_{i} \in \mathcal{M}} a_{i} = b_{i} \wedge b_{i} \wedge$
MO_RELEASE MO_ACQUIRE MO_CONSIME	is_at_mutex_location a = is_at_location_kind a MUTEX		(* -fence restrictions-*) (* 29.33 *)
MO_ACQ_REL	is at non-atomic location 2 =	carries_u_dependency_to = $a \xrightarrow{\text{submark}(u)} b = a = a = a = a = a = b = b = b = a = b = b$	(ra T(x)) = Constantia , Vy. (kilene x August x Augusti x Augusti a A
action =	is_at_non_atomic_location a = is_at_location_kind a NON_ATOMIC		$[h_{\mu\nu}rrln + J$ remains $\mu \neq J$ and $\mu = J$ $\mu = \frac{1}{2} \sum_{i=1}^{n} J_{\mu\nu} = \frac{1}{2} \sum_{i=1}^{n} J$
LOCK of action_id thread_id location UNLOCK of action_id thread_id location	is_at_atomic_location a =	carries_a_dependency_to_set actions threads location-kind sequenced before additional-synchronized-with data-dependency control-dependency rf =	$x \to (x, y, y)$ $\to (y, z)$ sublicities only $y \land X$
ATOMIC_LOAD of action_id thread_id memory_order_enum location val ATOMIC_STORE of action_id thread_id memory_order_enum location val ATOMIC_RIMN of action_id thread_id memory_order_enum location val val	is_at_location_kind > ATOMIC	curries_a_dependency_to_actions threads location-kind sequenced-before additional-synchronized-with data-dependency control-dependency rf a b}	$(2,3)3,4^{\dagger}$ $(f_{4,4}) \in maximula balan, \forall (p, k) \in \vec{a}$,
ATOMIC_RMW of action_id thread_id memory_order_enum location val val LOAD of action_id thread_id location val STORE of action_id thread_id location val	same_thread $a b = (thread_id_of a = thread_id_of b)$	dependency_ordered_before = $a \frac{dependecy-ordered_before}{dependency_ordered_before} d =$	(Vectoring a Construction of a
FENCE of action_id thread_id memory_order_enum		$\mathbf{a} \in \operatorname{arding} \wedge d \in \operatorname{arding} \wedge$ (3b, increasing $\mathbf{b} \wedge$ (3c, $\mathbf{a}, \mathbf{a}, \mathbf{a}, \mathbf{a}, \mathbf{a}, \mathbf{a}, \mathbf{a}, \mathbf{b}) \wedge$	$x = b \land b = a tomore action b)$ $\rightarrow (v = c) x = a tomore action b = a tomore action $
(action_id_of (LOCK aid) = aid) ∧	threadwise_relation_over $s rel =$ relation_over $s rel \land (\forall(a, b) \in rel. same_thread a b)$	$(\exists x, x \xrightarrow{\text{subservations}} e \xrightarrow{\rightarrow} b) \land$ (b) $(\exists x \text{-arrise-adjunction}) \xrightarrow{a} d \lor (b = d))))$	(* 2015 *)
$(action_id_of (UNLOCK aid _) = aid) \land$ $(action_id_of (ATOMIC_LOAD aid) = aid) \land$	same_location $z b = (location z = location b)$		$(v (x,x) \in \underline{waxardada}, y (x,b) \in \underline{waxardada}, yz.$ $(b_{k})tomik = that a \land h_{k} = that x \land h_{k} = that x \land h_{k}$
(action_id_of (ATOMIC_STORE aid) = aid) ∧ (action_id_of (ATOMIC_RMW aid) = aid) ∧ (action_id_of (ATOMIC_RMW aid) = aid) ∧	Sume_ocation 2 = (acation 2 = tocation B)	dependency_contend_before_set actions thanks location-kind sequenced-before additional-synchronized-with data-dependency control-dependency of modification-order release-sequence carries-a-dependency-to= dependency_contend_before_actions threads location-kind sequenced-before additional-synchronized with data-dependency control-dependency of modification-order release-sequence carries-a-dependency-to=b	is write a /sisfing y /sisequat y / is stating-action b / sime-focultion a b /
(action_id_of (LOAD aid) = aid) ∧ (action_id_of (STORE aid) = aid) ∧ (action_id_of (FENCE aid _) = aid)	locations of actions = { l . $\exists a. (location a = SOME \ l)}$		$x \neq y_1, z \neq y_2$ $\rightarrow (z = 2) \sqrt{2 - 2(z + 2) + 2(z + 2)} + 2(z + 2)$
	well_formed_action a =	simple_happens_before = ^{simple_happens_before = (separated bafors extension = simple_happens_before = = = = = = = = = = = = = = = = = = =}	
(thread_id_of (LOCK _ tid _) = tid) \land (thread_id_of (UNLOCK _ tid _) = tid) \land	case \Rightarrow of ATOMIC_LOAD mem_ord _ \rightarrow mem_ord \in		3] d.t.d.s.depeddery =
(thread_id_of (ATOMIC_LOAD_tid) = tid) \land (thread_id_of (ATOMIC_STORE_tid) = tid) \land	{Mo_RELAXED, Mo_ACQUIRE, Mo_SEQ_CST, Mo_CONSUME} ATOMIC_STORE mem_ord> mem_ord \in	consistent_simple_lappens_before abb =	
(thread id of (Arouno pure tid) - tid) a	II IMO RELAXED MO RELEASE MO SEO OST3	B B1525335(1-7)	

We can reason about C concurrency!

(value_written (ATOMIC_RMW v) = SOME v) \land (value_written (STORE v) = SOME v) \land		$(\forall l \in locations_of actions. case locations.ind l of$	(2≠4) ∧ same docation 2b ∧ (is write 2) ∧ (is write 2) ∧
(value_written _= NONE)	all_lock_or_unlock_actions_at lopt as = $\{a \in as: is_lock_or_unlock \land (location a = lopt)\}$	Arounc \rightarrow (let actions.st. $J = \{z, \text{location } a = \text{Soute } I\}\}$ in let write.st. $J = \{z, z, L, z, z, L, z, z, z = N \}$	$\neg \max_{z \in Z} thread z \geq b \land$ $\neg \langle f_{z} \downarrow torins (z trian > \Lambda is a toriniz action b) \land$ $\neg \langle z = \frac{hapten-haltin}{z} \rightarrow b \lor b = \frac{hapten-haltin}{z} \rightarrow 0 \rangle$
$ \begin{array}{l} \mathrm{is_lock} \ a = \\ \mbox{case } a \ of \ \mathrm{Lock}___\to T \parallel _\to F \end{array} $	consistent_locks = consistent_locks = $\forall I \in \text{ locations_of actions. (location-kind I = MUTEX)} \implies ($	is_atomic_ators = 4' is_atomic_ators = 3) in strict_tabularder_arrer = wines_z_d = <u>(unification-win)_(unification-win</u>	data_mos ¹ actions threads location-kind sequenced before additional-spectromized-with data-dependency control-dependency of modification-order sc == left release-sequence = release_sequence_set actions threads. location-land sequenced-before additional-spectromized-with data-dependency control-dependency control-
is_unlock a = case a of UNLOCK \rightarrow T _ \rightarrow F	Ist fock-unlock-actions = all-lock-arriance-actions-art (SOME /)actions in Ist fock-arriance-actions-art (SOME /)actions in Ist fock-artistics actions-art (SOME /)actions in (* 30.41.5 - The imdemnitation shall serialize those (lock and unlock) operations. *)	¹ augum holm ₁ _{intern} ⊂ mathematication and r, (* Mu ₂ N ₂ N ₂) ⊂ function inputs modification on order *) (minum holm ₂ ⊂ (function) = minum holmm ₁ (minum holm ₂) (minum) = minum holmm ₁ (minum holm ₂) = (function) = minum holmm ₁ (minum holm ₂) = (function) = minum holm ₁ (minum holm ₁) = (function) = minum holm ₁	Let hypothetical-relaxes sequence = relaxes sequence at actions threads location-laid responsed before additional spectromized with data-dependency control-dependency modification order in Let spectromize with = spectromizes: relaxes actions threads location-laid responsed before additional spectromized with data-dependency control-dependency or final Let dependency control-dependency control-dependency or final Let dependency control-dependency control-dependency round-dependency or final Let dependency control-dependency control-dependency round-dependency control-dependency round-dependency round-dependen
is_atomic_load $\mathfrak{a} =$ case \mathfrak{a} of Atomic_load $\to F$	strict_total_order_over fock_unlock_actions fock_order ∧ (* 30.4.1.1 A thread owns a mutex from the time it successfully calls one of the lock functions until it calls unlock.)	$\subseteq \frac{\operatorname{contractions of }_{ij}}{\operatorname{contract}_{ij}} = <0.$ (contracting $a = 5.0$ cm I) in $\left(\frac{\operatorname{contractions of }_{ij}}{\operatorname{contract}_{ij}} = 0.0$ (i))	ket inter strusst Augense before = linter, linteruk Luppen, J-dene z-krisen thrusks bezation-kind sugenende before additional-synchronized with data gendenose; control-dependency repetitionalises - with data gendenose; control-dependency inter-thrush Augense before in data_nuces actions thrush. location-kind sequenced before additional-synchronized-with data dependency control-dependency inter-thrush Augense before in data_nuces actions thrush. location-kind sequenced before additional-synchronized-with data dependency restructions.
is_atomic_store a = case a of Atomic_Store → T $\ $ _ → F	(* 30.4.120 Requires: The calling thread shall ono the mates. *) (* 30.4.121 Effects: Relases the calling threads ownership of the mates.*) (Va ₁ ∈ look_undock_actions: logindlock a _µ ⇒→ (Ja ₁ ∈ look_undock_actions:	visible discuttor = $a^{-\text{min}} a^{bh} a^{bh} a^{bh} b = a^{-bh} b^{bh} a^{bh} a^{bh} b^{bh}$	cgs_memory_model open ((= c'program) = let exercutivas = ((actions, threads, location-kind, asquenced-before additional-genchronized with data-dependency, centrol-dependency, cf, modification-order, sc). open particular to thread asquenced-before additional-genchronized with data-dependency centrol-dependency - consistent_execution actions: threads, location-kind sequenced-before additional-genchronized with data-dependency - modification-order sc) in
is_atomic_rmw a = case a of Atomic_RMW \to T $\ $ - \to F	a) ^{backyatter} a ₂ ∧ same_thread a ₁ a ₂ ∧ is_lock a ₃)) ∧ (* 30.4.17 Effects: Blocks the calling thread until ownership of the mutex can be obtained for the calling thread.*) (* 30.4.18 Potocondition: The calling thread owns the mutex. *)	is write $a \land h_{a} read \land h_{a} man_{a} location a \land h_{a} Ca : (c \neq a) \land (c \neq b) \landis write c \land sume_{b} coston (c \land h)a \ sume_{b} coston (c \land h)$	If "jartion, threak, locato-kind, sequences before, additional-genetonicari-with, data-dependency, control-dependency, c modification-order, g \in exerctions. (inderturning results are constrained in threak location-kind sequences before additional genetonicari with data-dependency control-dependency of \neq []]) \vee (insequence), zeros actions threads location-kind sequences before additional-genetionicari with data-dependency control-dependency of \neq []]) \vee (distances actions threads location-kind sequences before additional-genetion control-dependency of the diffication-order \neq (]) (distances actions threads location-kind sequences before additional-genetion control-dependency of the diffication-order \neq (])
is load a – case a of Load \rightarrow T $_{*}$ \rightarrow F	$ \begin{array}{c} (Y_{2i} \in [oci_{i}, \operatorname{and} c_{i}, \operatorname{actions} \ i_{k}] ock \ a_{i} \rightarrow \\ (Y_{2i} \in [oci_{i}, \operatorname{and} c_{i}, \operatorname{actions} \ i_{k}] ock \ a_{i}]))) \end{array} $		etc escutions

Shared memory

```
int a = 1;
int b = 0;
```

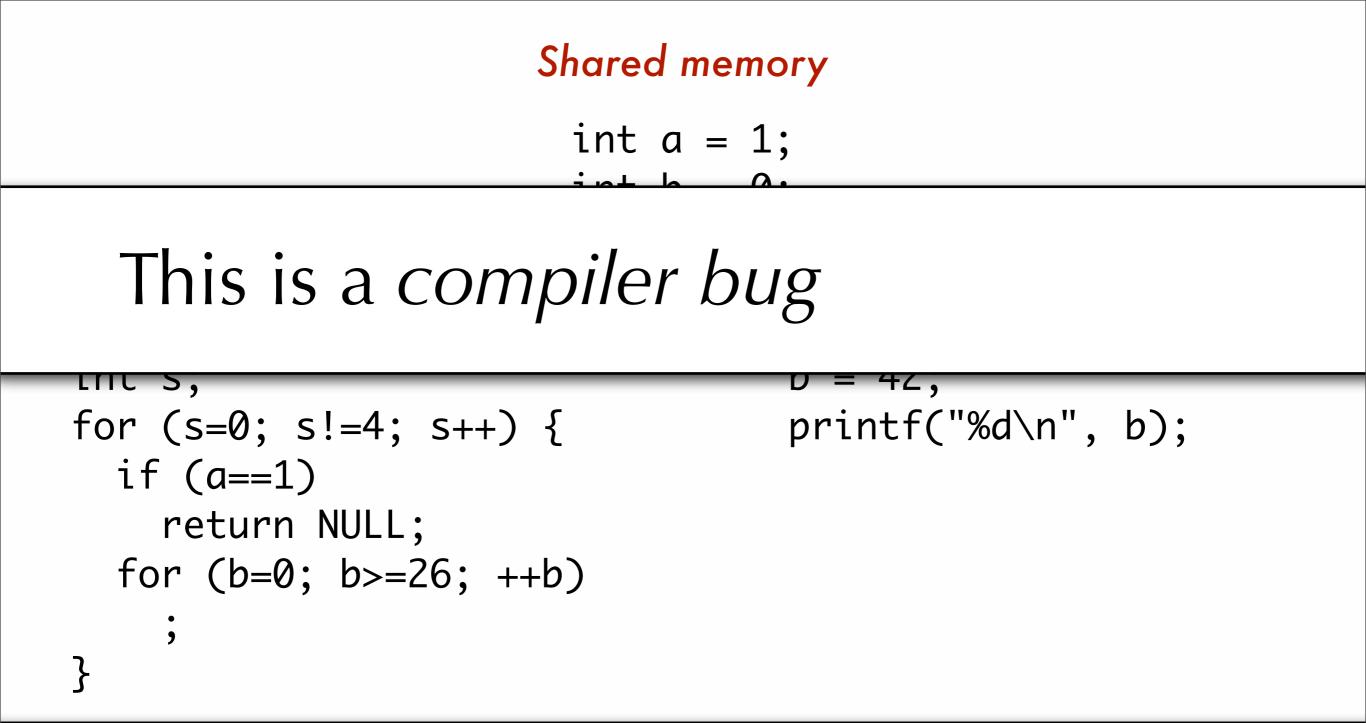
Thread 1

Thread 2

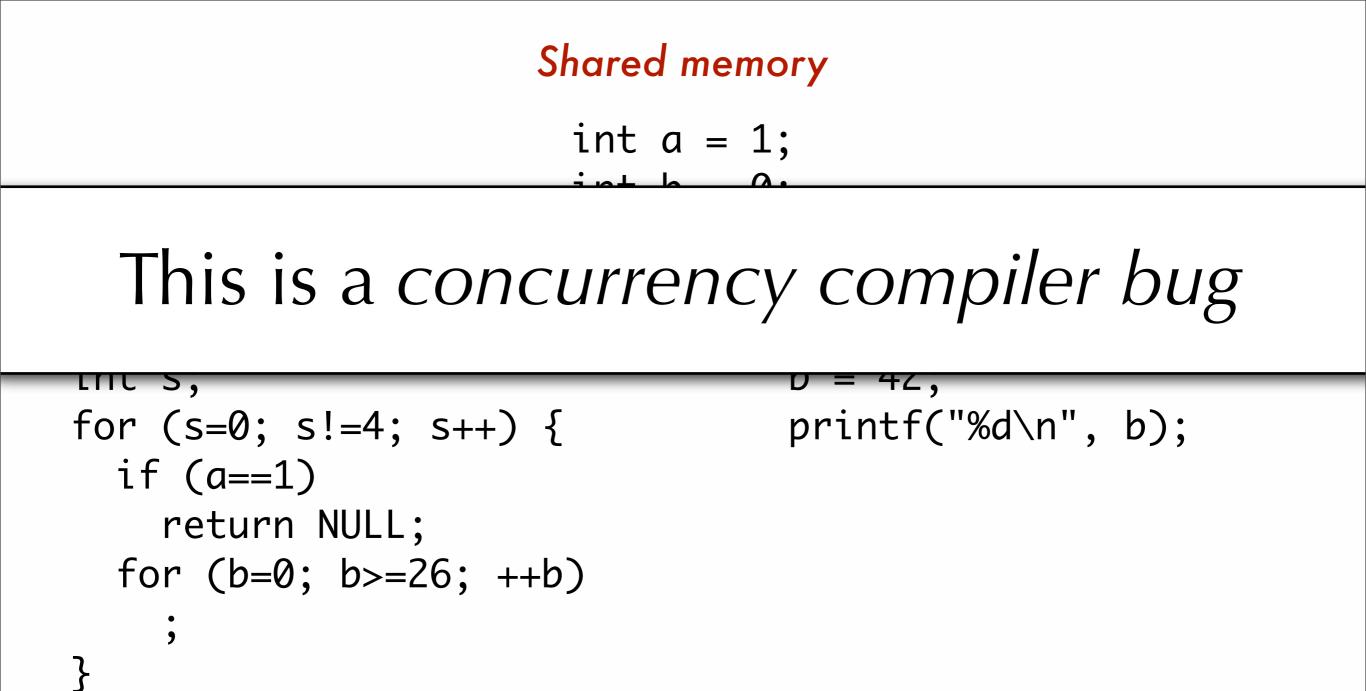
```
int s;
for (s=0; s!=4; s++) {
    if (a==1)
      return NULL;
    for (b=0; b>=26; ++b)
      ;
}
```

```
b = 42;
printf("%d\n", b);
```

Thread 2 is not affected by Thread 1 and vice-versa This program is data-race free *This program must print 42*



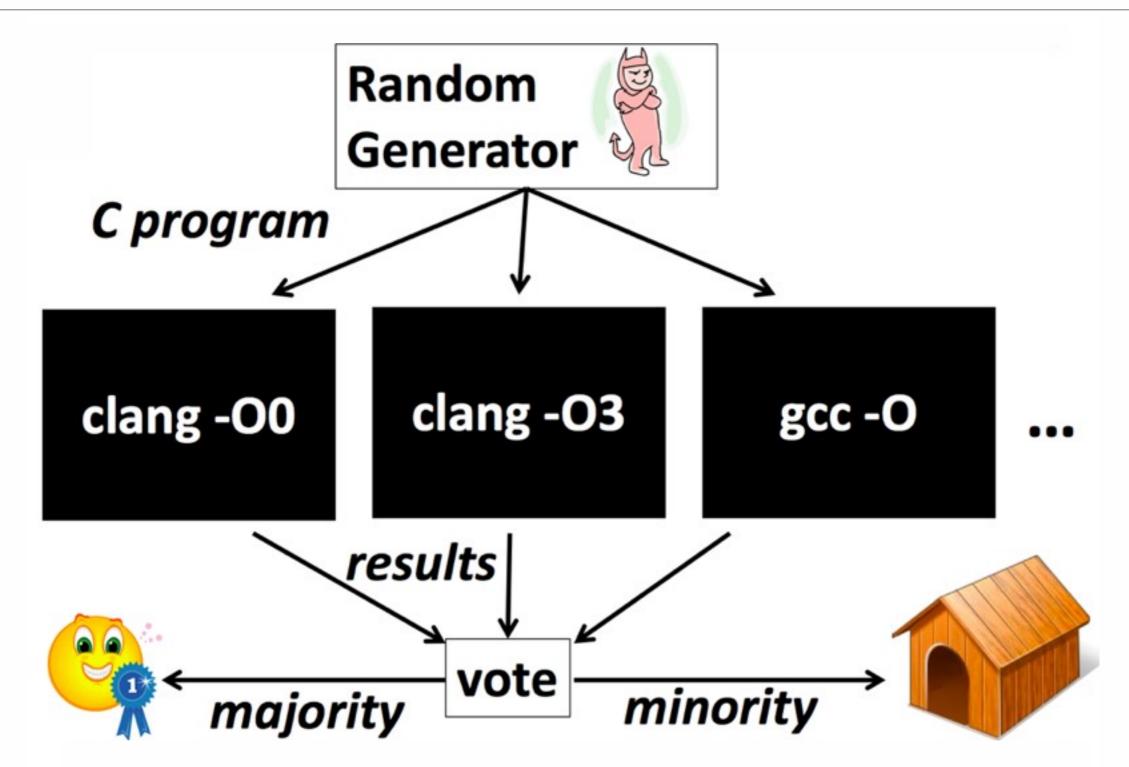
Thread 2 is not affected by Thread 1 and vice-versa This program is data-race free *This program must print 42*



Thread 2 is not affected by Thread 1 and vice-versa This program is data-race free *This program must print 42*

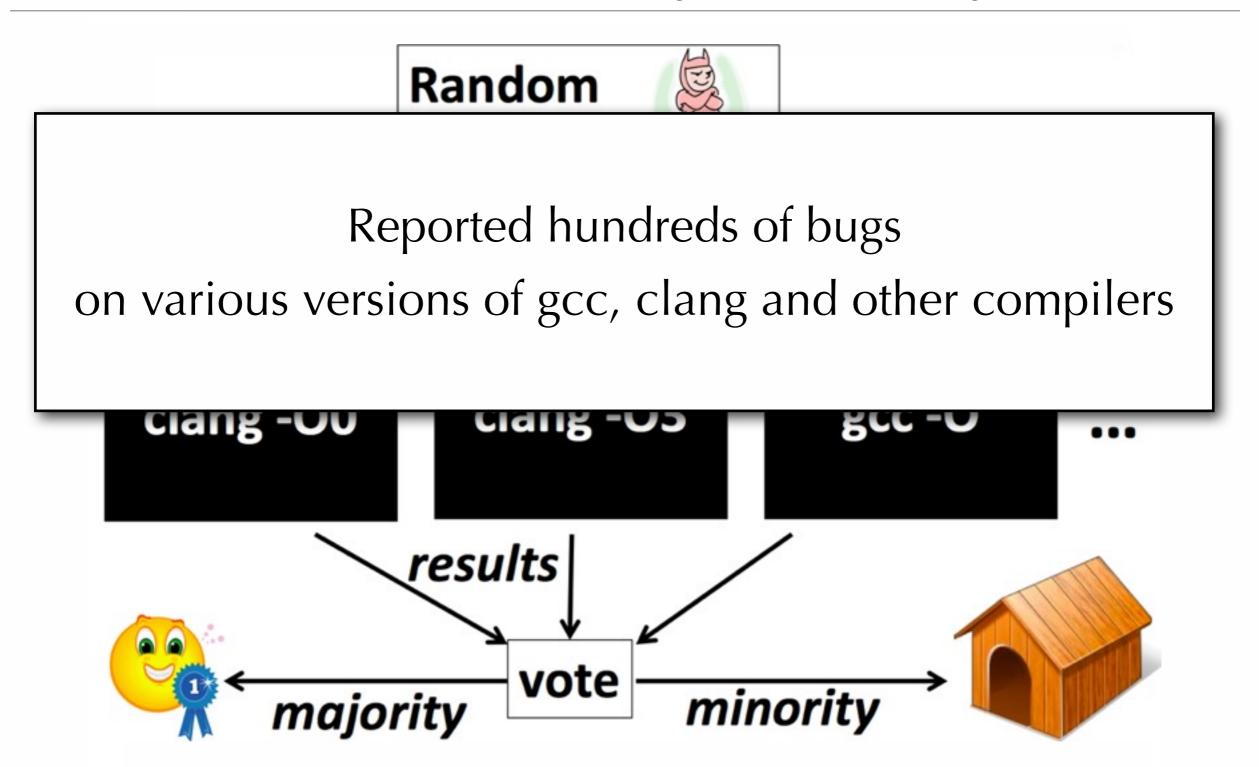
Compiler testing: state of the art

Yang, Chen, Eide, Regehr - PLDI 2011



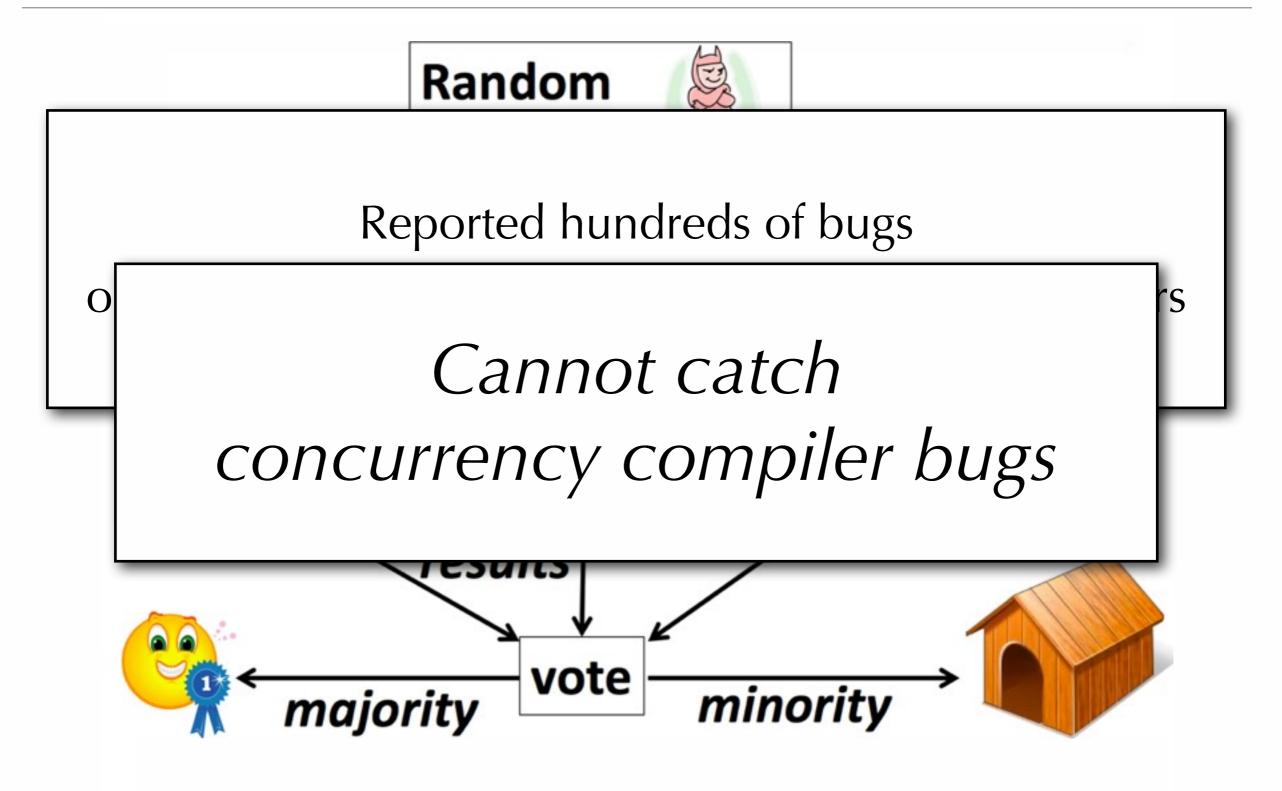
Compiler testing: state of the art

Yang, Chen, Eide, Regehr - PLDI 2011



Compiler testing: state of the art

Yang, Chen, Eide, Regehr - PLDI 2011



Hunting concurrency compiler bugs?

How to deal with non-determinism?

How to generate non-racy interesting programs?

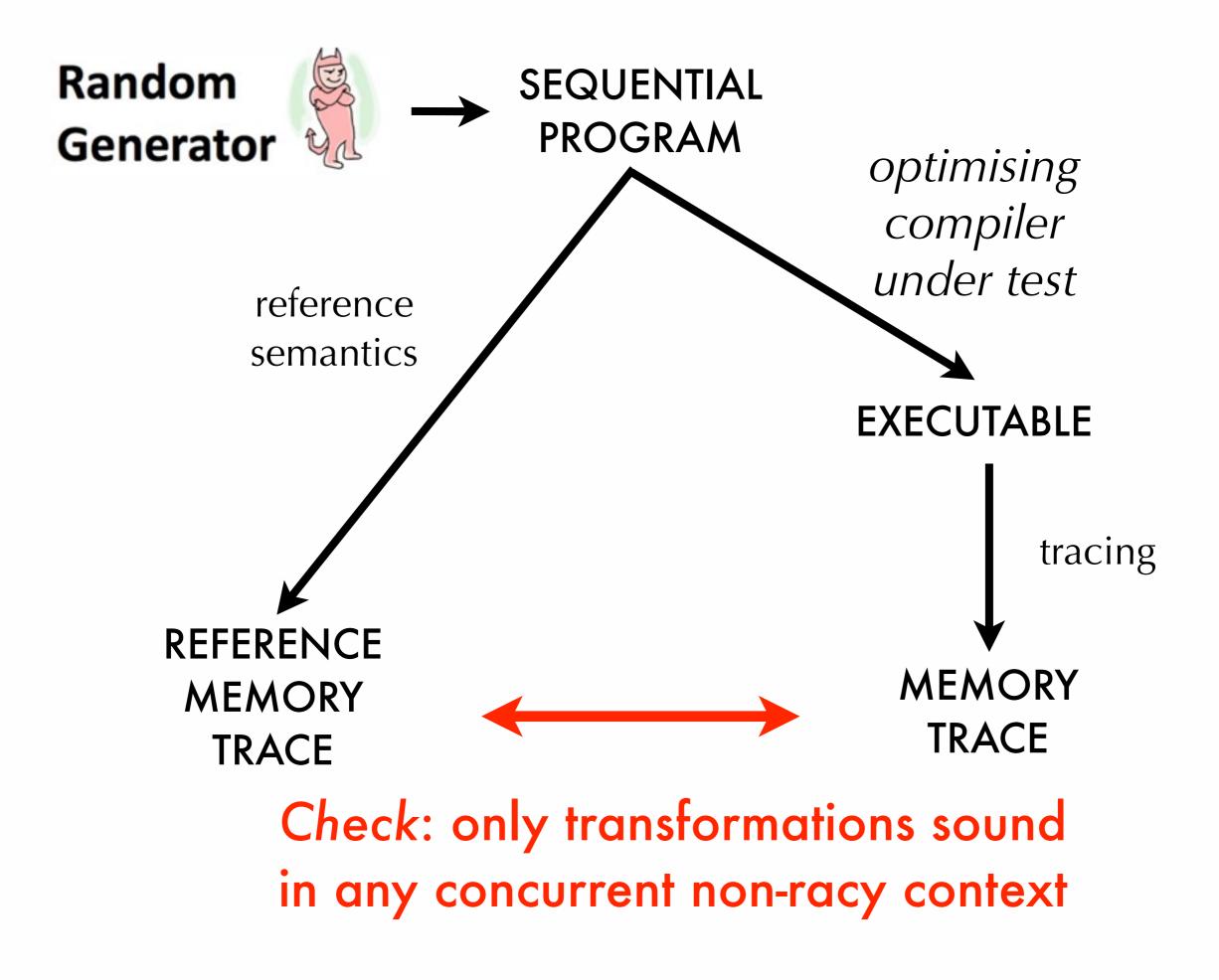
How to capture all the behaviours of concurrent programs?

A compiler can optimise away behaviours: *how to test for correctness? limit case*: two compilers generate correct code with disjoint final states C/C++ compilers support separate compilation Functions can be called in arbitrary non-racy concurrent contexts

C/C++ compilers can only apply transformations sound with respect to an arbitrary non-racy concurrent context

Hunt concurrency compiler bugs

search for transformations of sequential code not sound in an arbitrary non-racy context



Soundness of compiler optimisations in the C11/C++11 memory model



Compiler Writer





Compiler Writer



Sophisticated program analyses Fancy algorithms Source code or IR

Operations on AST



Compiler Writer



Sophisticated program analyses Fancy algorithms Source code or IR

Operations on AST



```
for (int i=0; i<2; i++) {
   z = i;
   x[i] += y+1;
}</pre>
```

Compiler Writer



Sophisticated program analyses Fancy algorithms Source code or IR

Operations on AST



```
tmp = y+1 ;
for (int i=0; i<2; i++) {
   z = i;
   x[i] +=tmp;
}</pre>
```

Compiler Writer



Sophisticated program analyses Fancy algorithms Source code or IR

Operations on AST

Semanticist



Elimination of run-time events Reordering of run-time events Introduction of run-time events *Operations on sets of events*

```
tmp = y+1 ;
for (int i=0; i<2; i++) {
   z = i;
   x[i] +=tmp;
}</pre>
```

Compiler Writer



Sophisticated program analyses Fancy algorithms Source code or IR

Operations on AST

Semanticist



Elimination of run-time events Reordering of run-time events Introduction of run-time events *Operations on sets of events*

```
Store z 0
Load y 42
Store x[0] 43
Store z 1
Load y 42
Store x[1] 43
```

Compiler Writer



Operations on AST

Semanticist



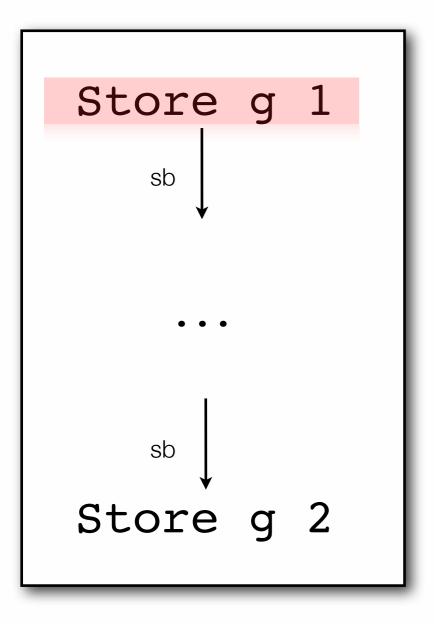
Elimination of run-time events Reordering of run-time events Introduction of run-time events *Operations on sets of events*

> Load y 42 Store z 0

Store x[0] 43 Store z 1

Store x[1] 43

Elimination of overwritten writes



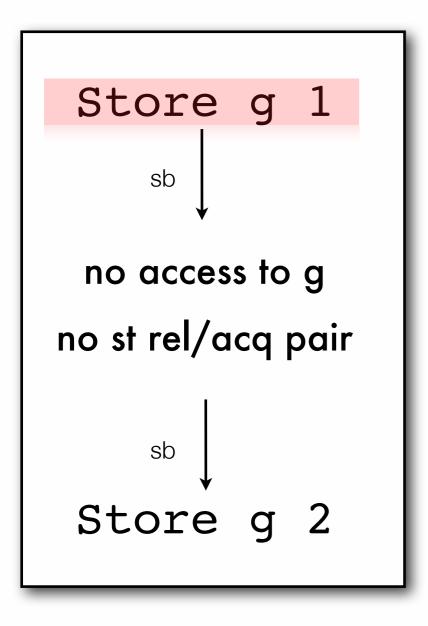
Under which conditions is it correct to eliminate the first store?

A same-thread release-acquire pair is a pair of a release action followed by an acquire action in program order.

An action is a *release* if it is a possible source of a synchronisation *unlock mutex, release or seq_cst atomic write*

An action is an *acquire* if it is a possible target of a synchronisation lock mutex, acquire or seq_cst atomic read

Elimination of overwritten writes



It is safe to eliminate the first store if there are:

 no intervening accesses to **g** no intervening same-thread release-acquire pair

Shared memory

$$g = 0;$$
 atomic f1 = f2 = 0;

Thread 1

```
g = 1;
f1.store(1,RELEASE);
while(f2.load(ACQUIRE)==0);
g = 2;
```

Shared memory

$$g = 0;$$
 atomic f1 = f2 = 0;

Thread 1 candidate overwritten write
g = 1;
f1.store(1,RELEASE);
while(f2.load(ACQUIRE)==0);
g = 2;

Shared memory

$$g = 0;$$
 atomic f1 = f2 = 0;

Thread 1 candidate overwritten write
g = 1;
f1.store(1,RELEASE); same-thread release-acquire pair
while(f2.load(ACQUIRE)==0);
g = 2;

Shared memory

$$g = 0$$
; atomic f1 = f2 = 0;

Thread 1

```
g = 1;
f1.store(1,RELEASE);
while(f2.load(ACQUIRE)==0);
g = 2;
```

Thread 2

while(f1.load(ACQUIRE)==0);
printf("%d", g);
f2.store(1,RELEASE);

Shared memory

$$g = 0;$$
 atomic f1 = f2 = 0;

Thread 1

Thread 2

g = 1; f1.store(1,RELEASE); while(f1.load(ACQUIRE)==0); while(f2.load(ACQUIRE)==0); g = 2; while(f1.load(ACQUIRE)==0); f2.store(1,RELEASE);

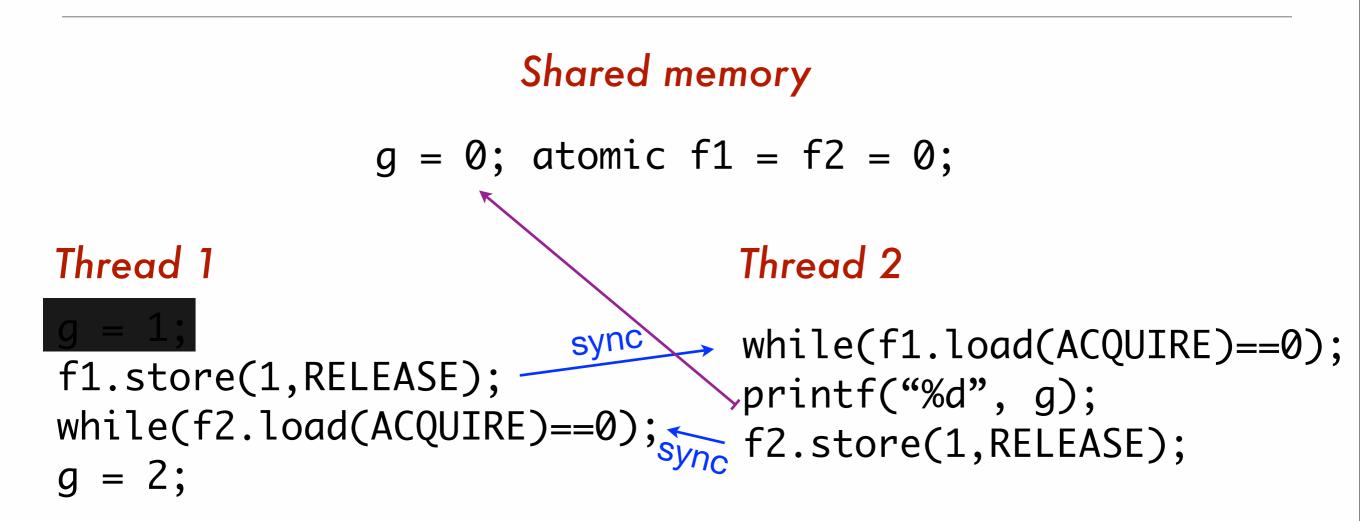
Thread 2 is non-racy

Shared memory

$$g = 0;$$
 atomic f1 = f2 = 0;

Thread 1 Thread 2 g = 1; f1.store(1,RELEASE); while(f1.load(ACQUIRE)==0); while(f2.load(ACQUIRE)==0); f2.store(1,RELEASE); g = 2;

Thread 2 is non-racy The program should only print **1**



Thread 2 is non-racy The program should only print **1**

If we perform overwritten write elimination it prints 0

Shared memory

$$g = 0;$$
 atomic f1 = f2 = 0;

Thread 1

Thread 2

g = 1;
f1.store(1,RELEASE);
while(f2.load(ACQUIRE)==0);
g = 2;

while(f1.load(ACQUIRE)==0);
printf("%d", g);
f2.store(1,RELEASE);

Shared memory

$$g = 0;$$
 atomic f1 = f2 = 0;

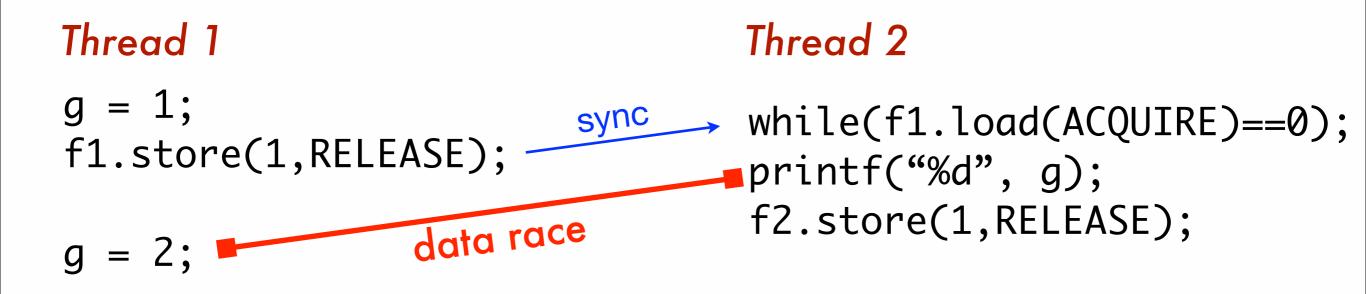
Thread 1

Thread 2

while(f1.load(ACQUIRE)==0);
printf("%d", g);
f2.store(1,RELEASE);

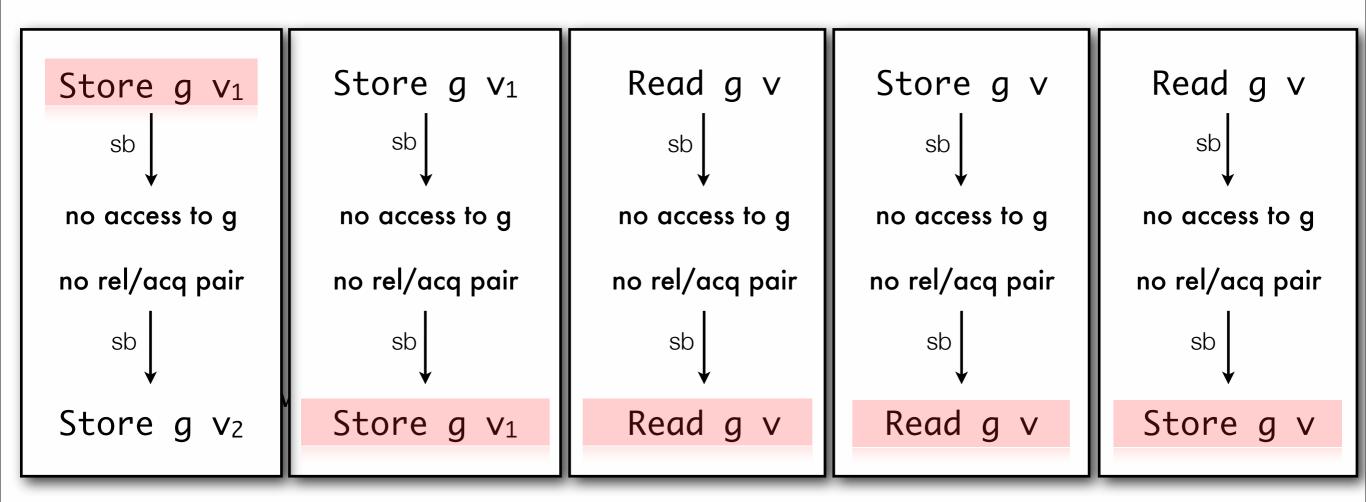
Shared memory

$$g = 0;$$
 atomic f1 = f2 = 0;



If only a release (or acquire) is present, then all discriminating contexts *are racy*.It is sound to optimise the overwritten write.

Eliminations: bestiary

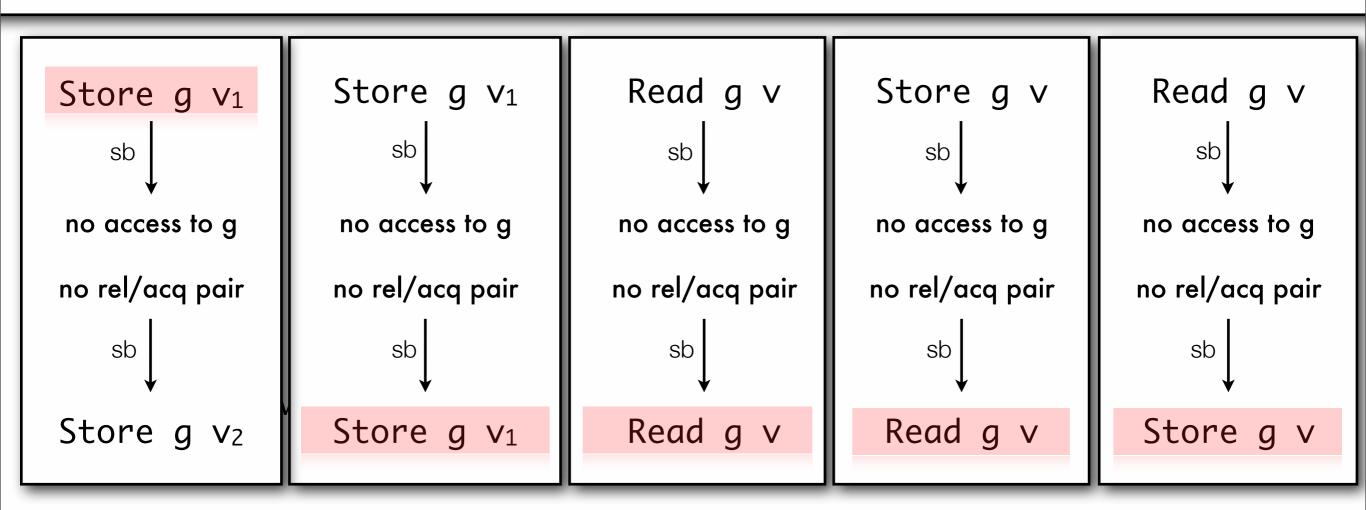


Overwritten-Write Write-after-Write Read-after-Read Read-after-Write Write-after-Read

Reads which are not used (via data or control dependencies) to decide a write or synchronisation event are also eliminable (*irrelevant reads*).

Also correctness statements for

reorderings, merging, and introductions of events.



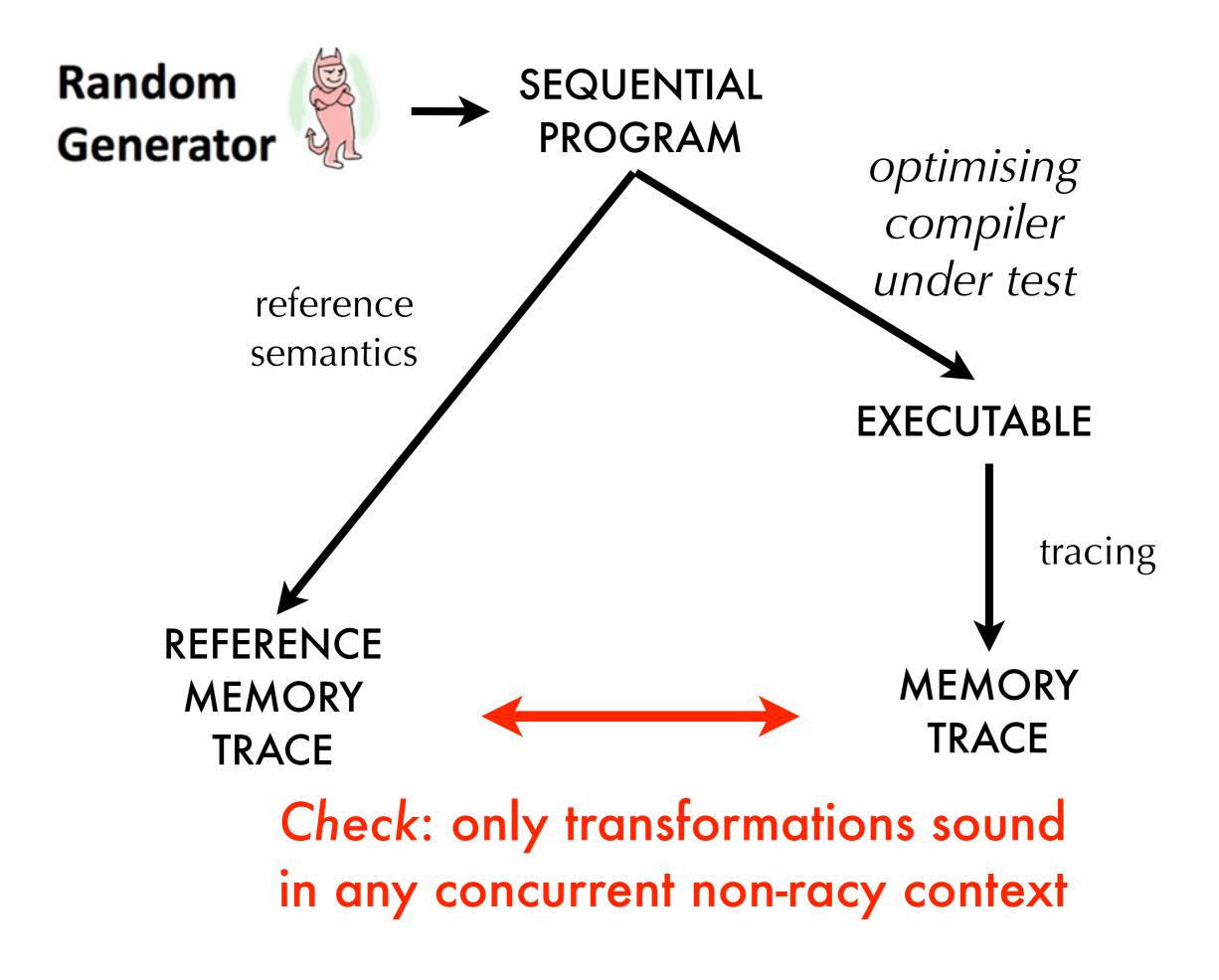
Overwritten-Write Write-after-Write Read-after-Read Read-after-Write Write-after-Read

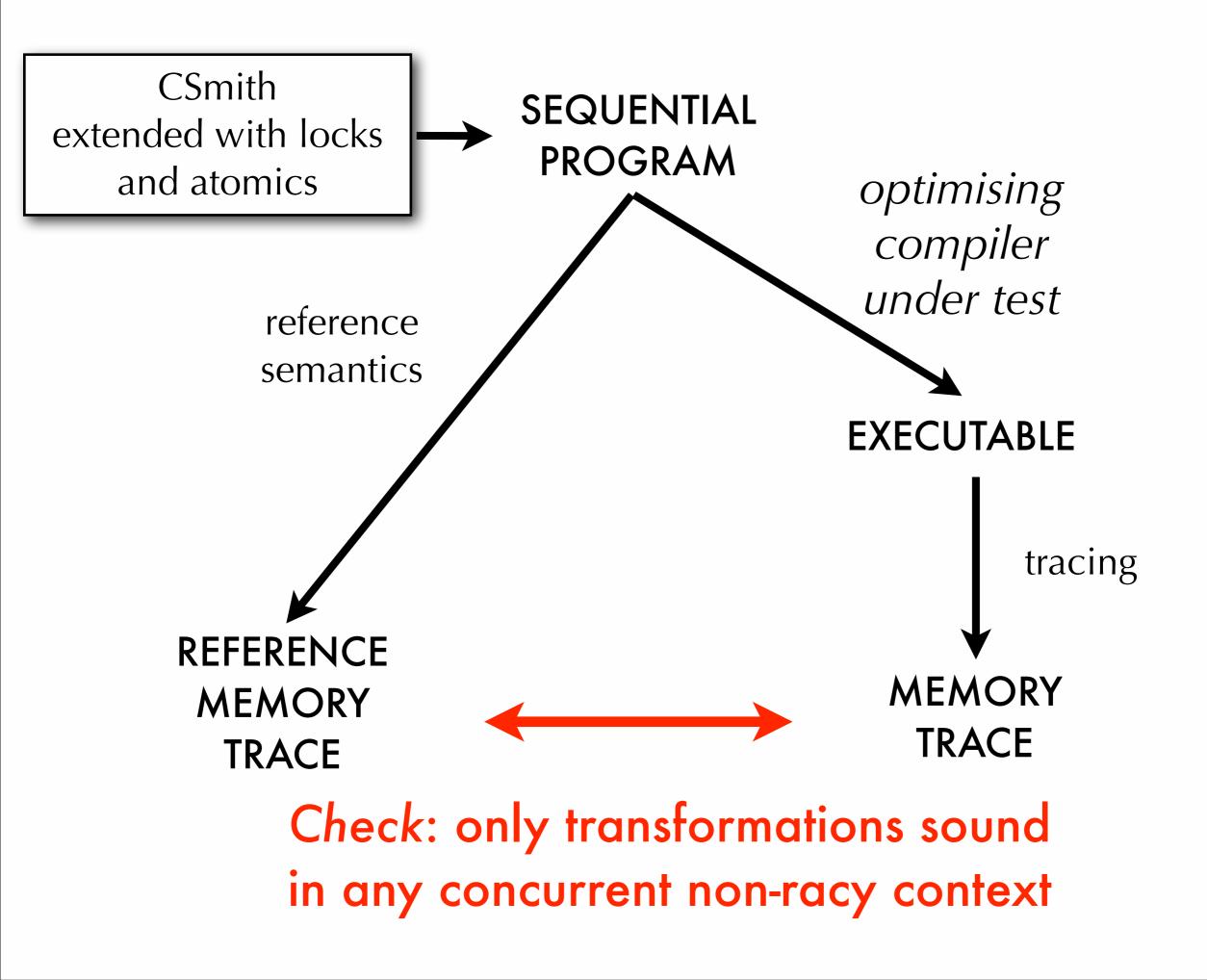
Reads which are not used (via data or control dependencies) to decide a write or synchronisation event are also eliminable (*irrelevant reads*).

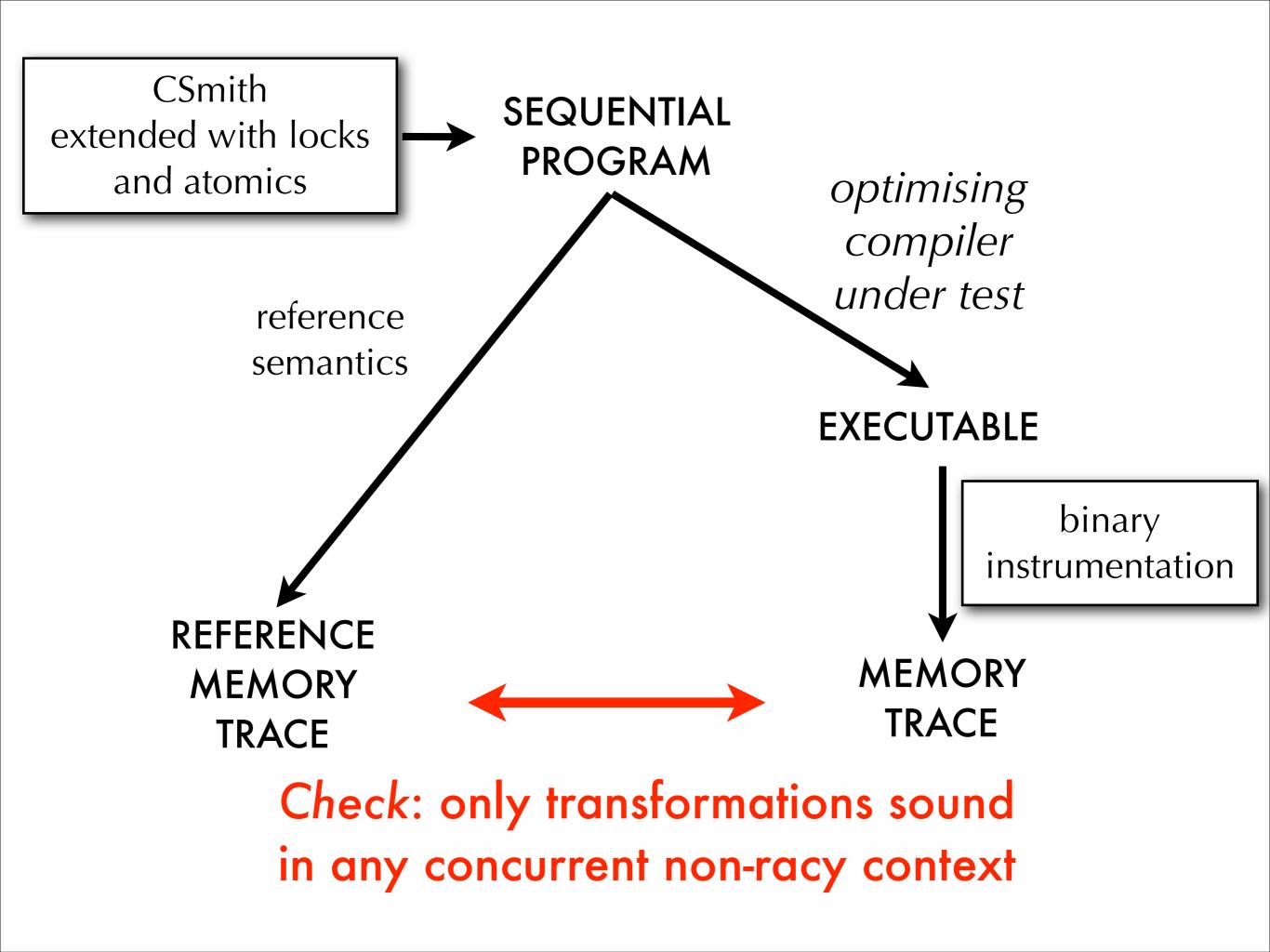
From theory to the Cmmtest tool

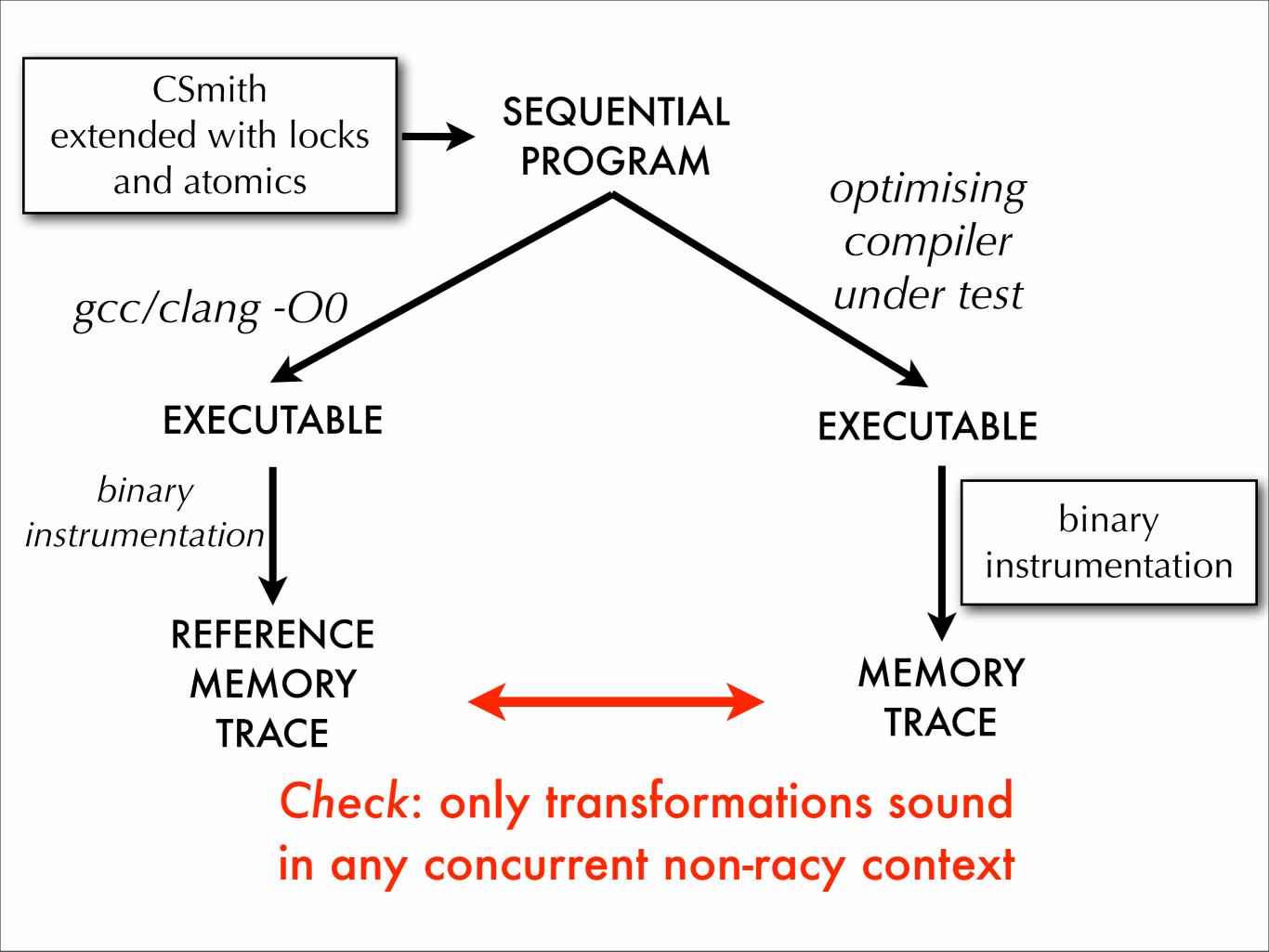


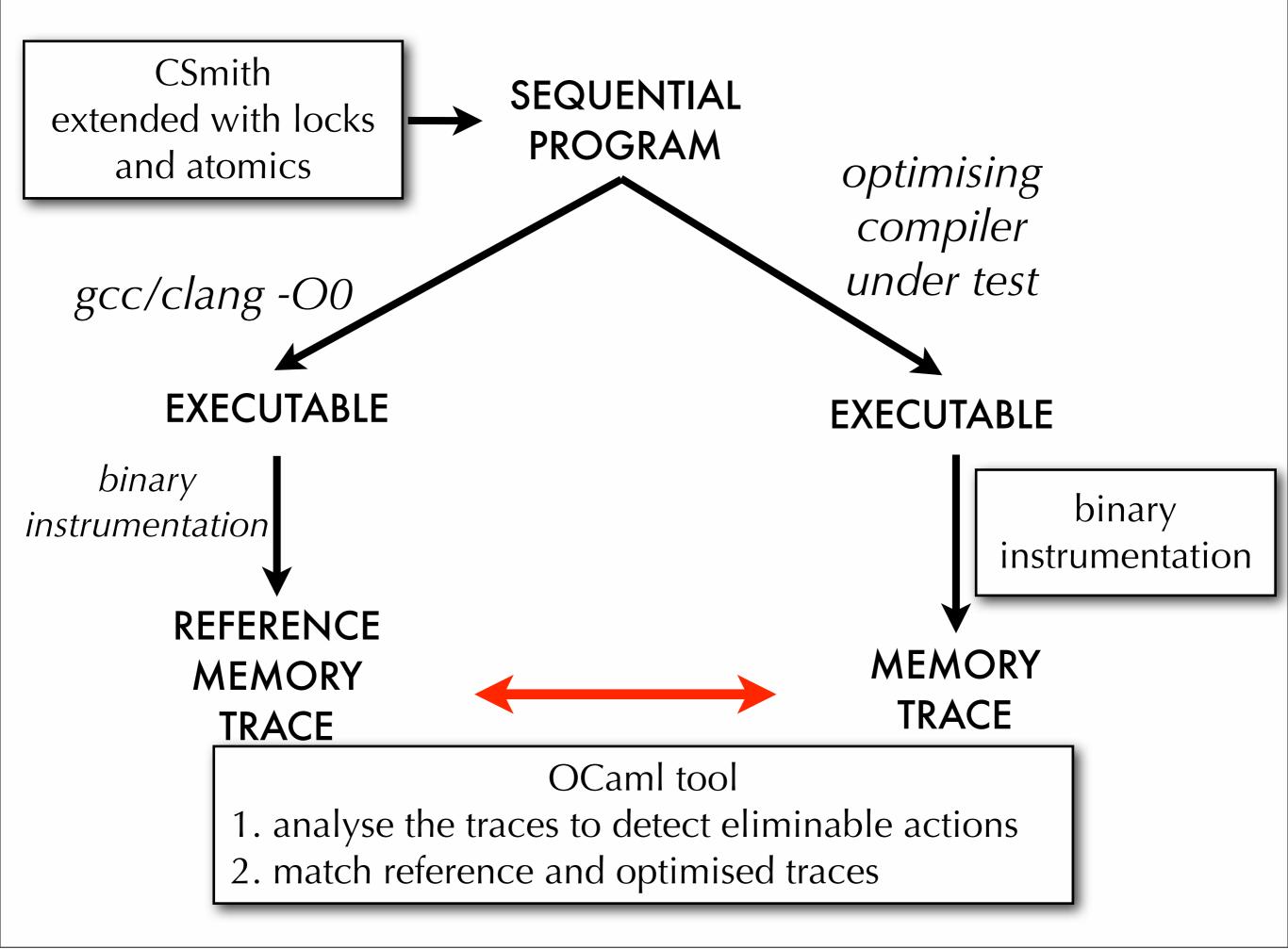
Monday 11 May 15











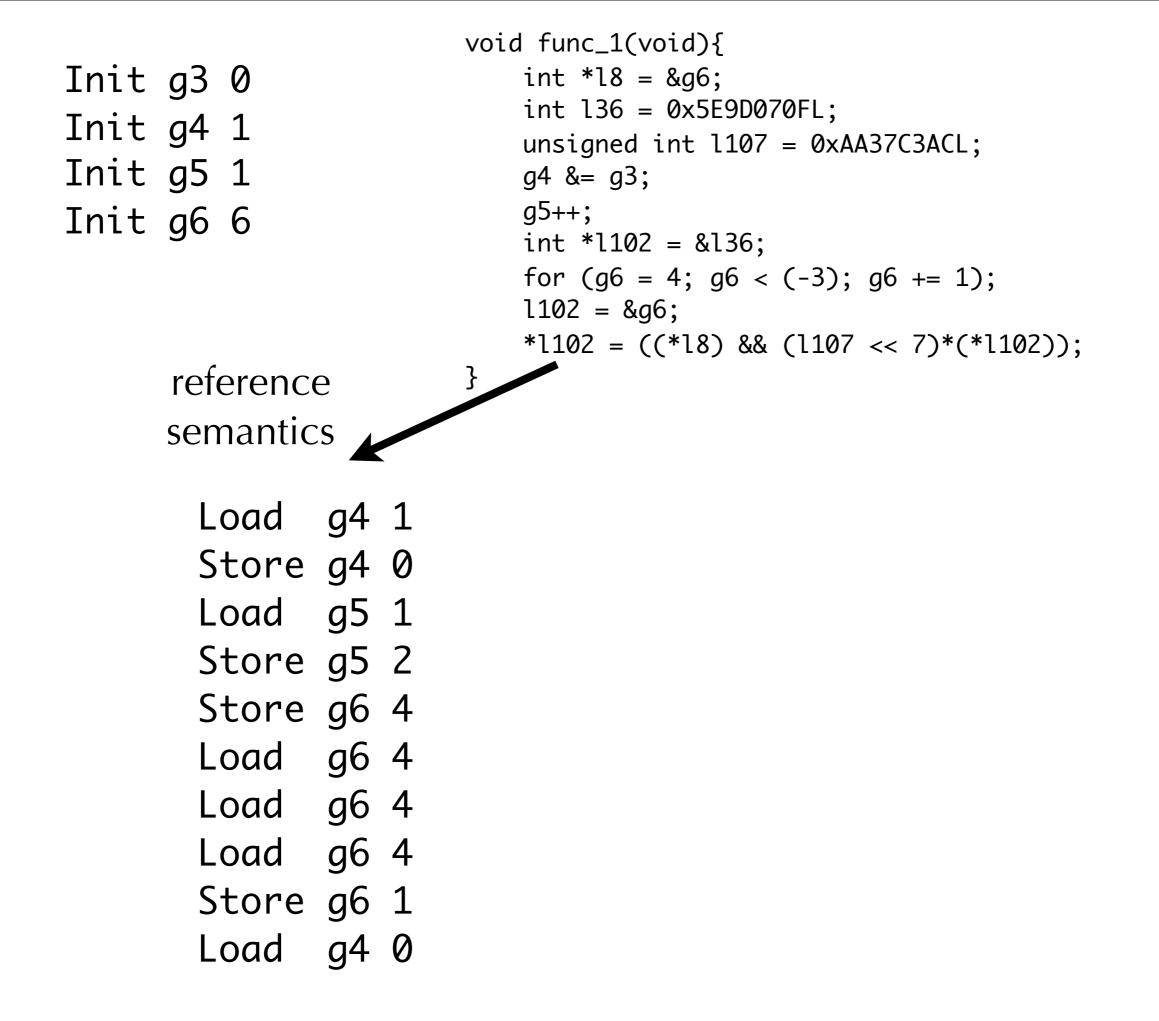
```
const unsigned int g3 = 0UL;
long long g4 = 0x1;
int g6 = 6L;
volatile unsigned int g5 = 1UL;
void func_1(void){
     int *18 = \&g6;
     int 136 = 0 \times 5E9D070FL;
     unsigned int 1107 = 0xAA37C3ACL;
     q4 \&= q3;
     g5++;
     int *1102 = \&136;
     for (g6 = 4; g6 < (-3); g6 += 1);
     1102 = \&g6;
     *1102 = ((*18) && (1107 << 7)*(*1102));
}
```

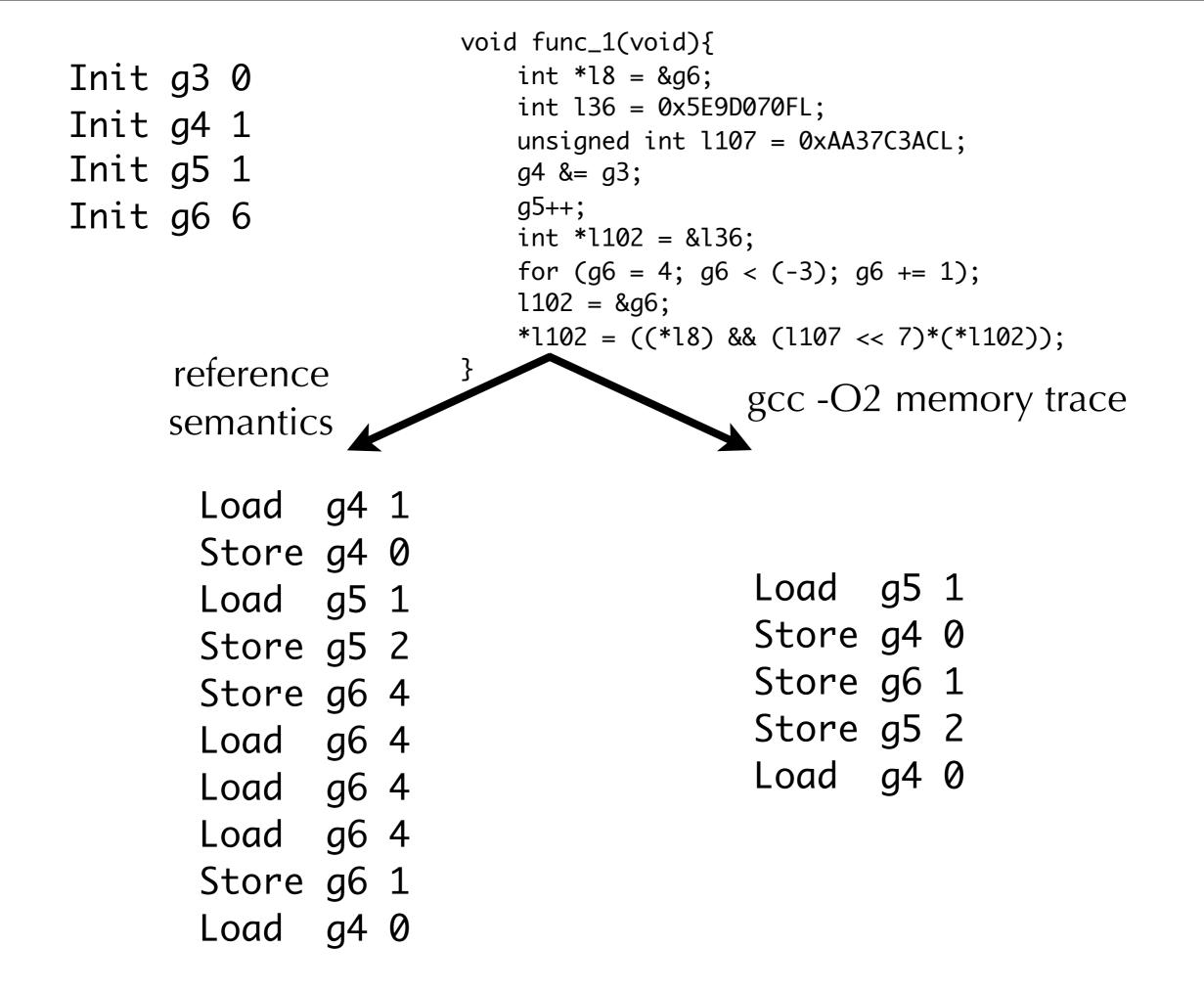
Start with a randomly generated well-defined program

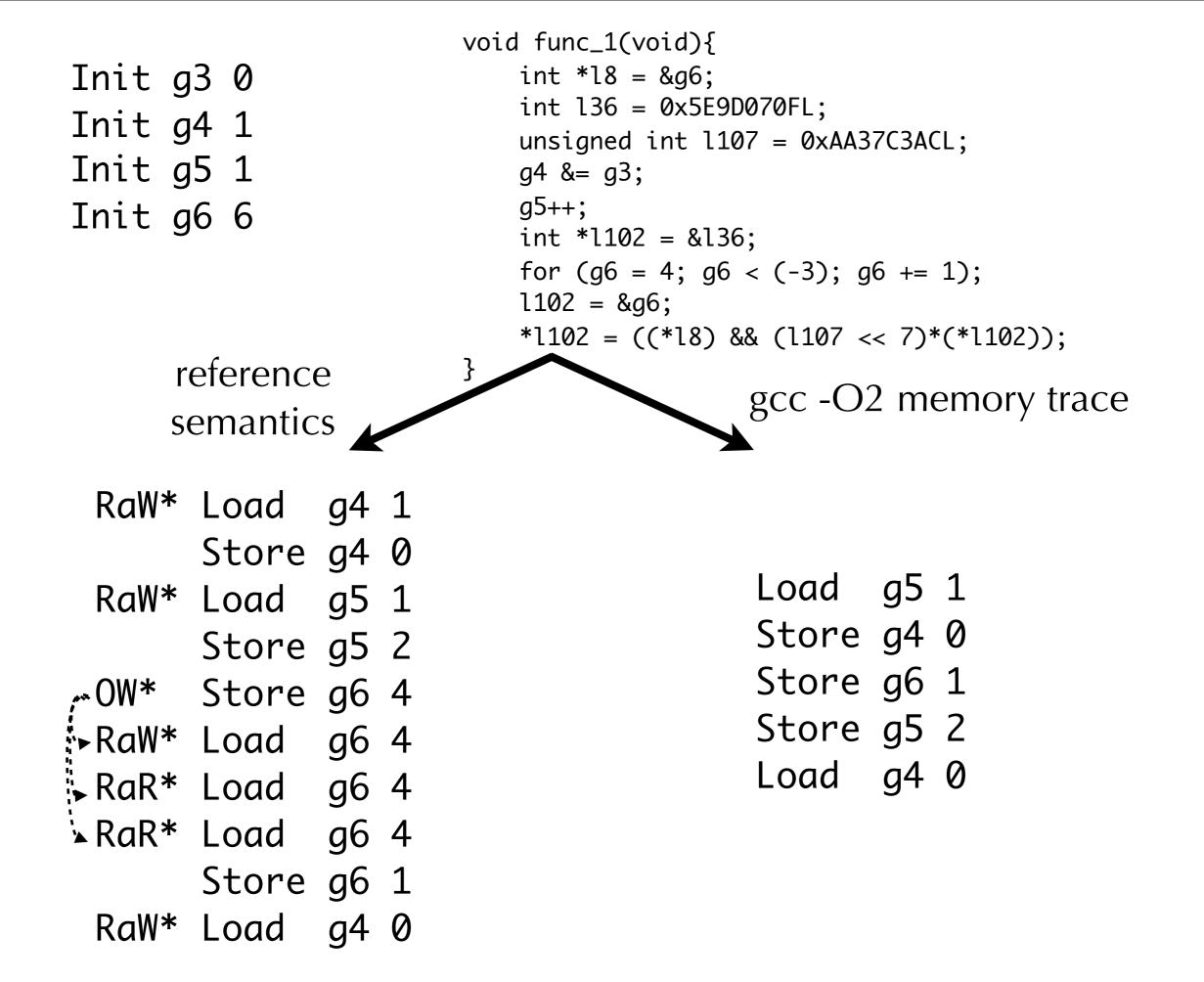
Init g3 0 Init g4 1 Init g5 1 Init g6 6

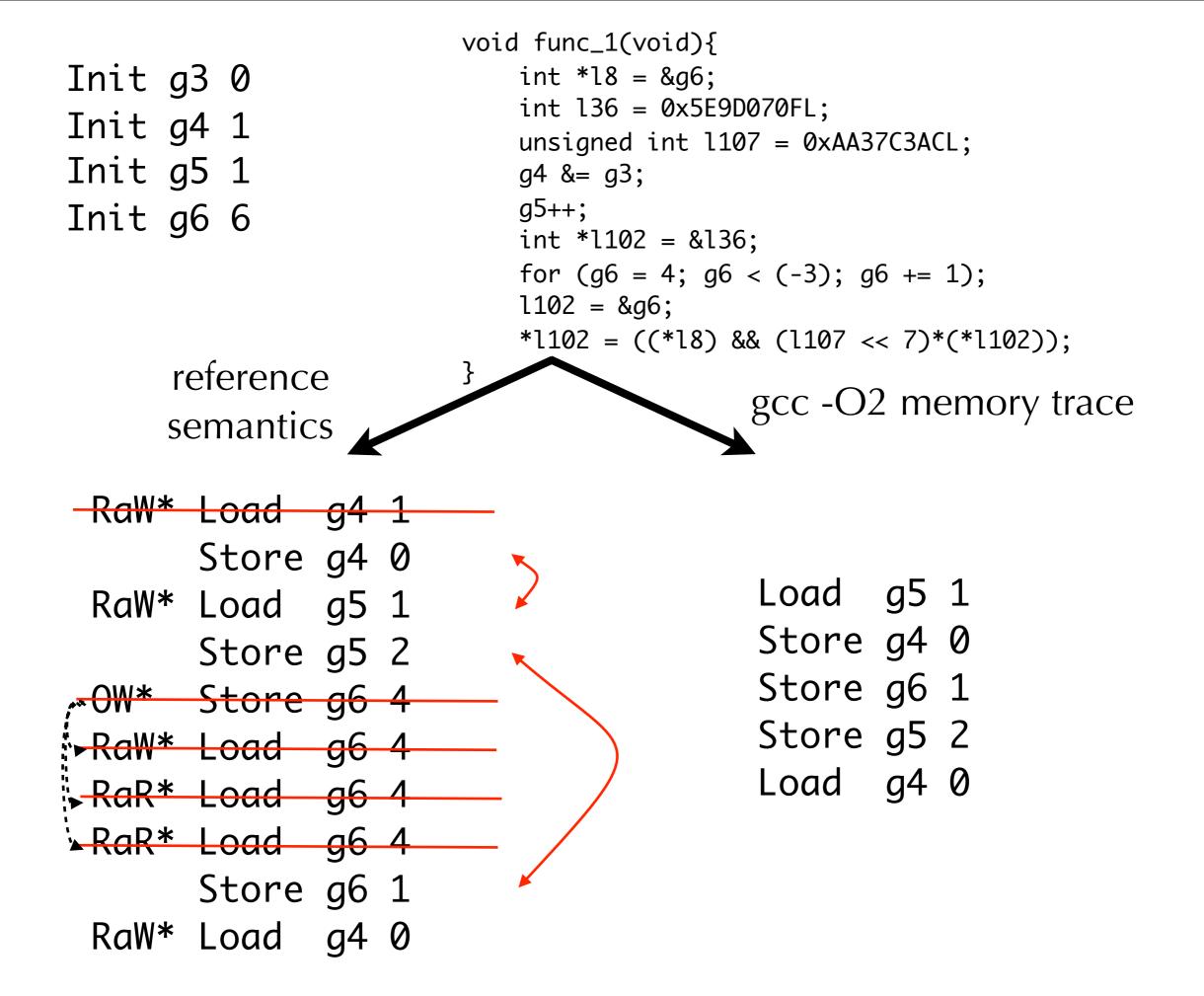
```
void func_1(void){
    int *18 = &g6;
    int 136 = 0 \times 5E9D070FL;
    unsigned int l107 = 0xAA37C3ACL;
    g4 &= g3;
    g5++;
    int *1102 = &136;
    for (g6 = 4; g6 < (-3); g6 += 1);
    1102 = \&g6;
    *1102 = ((*18) && (1107 << 7)*(*1102));
```

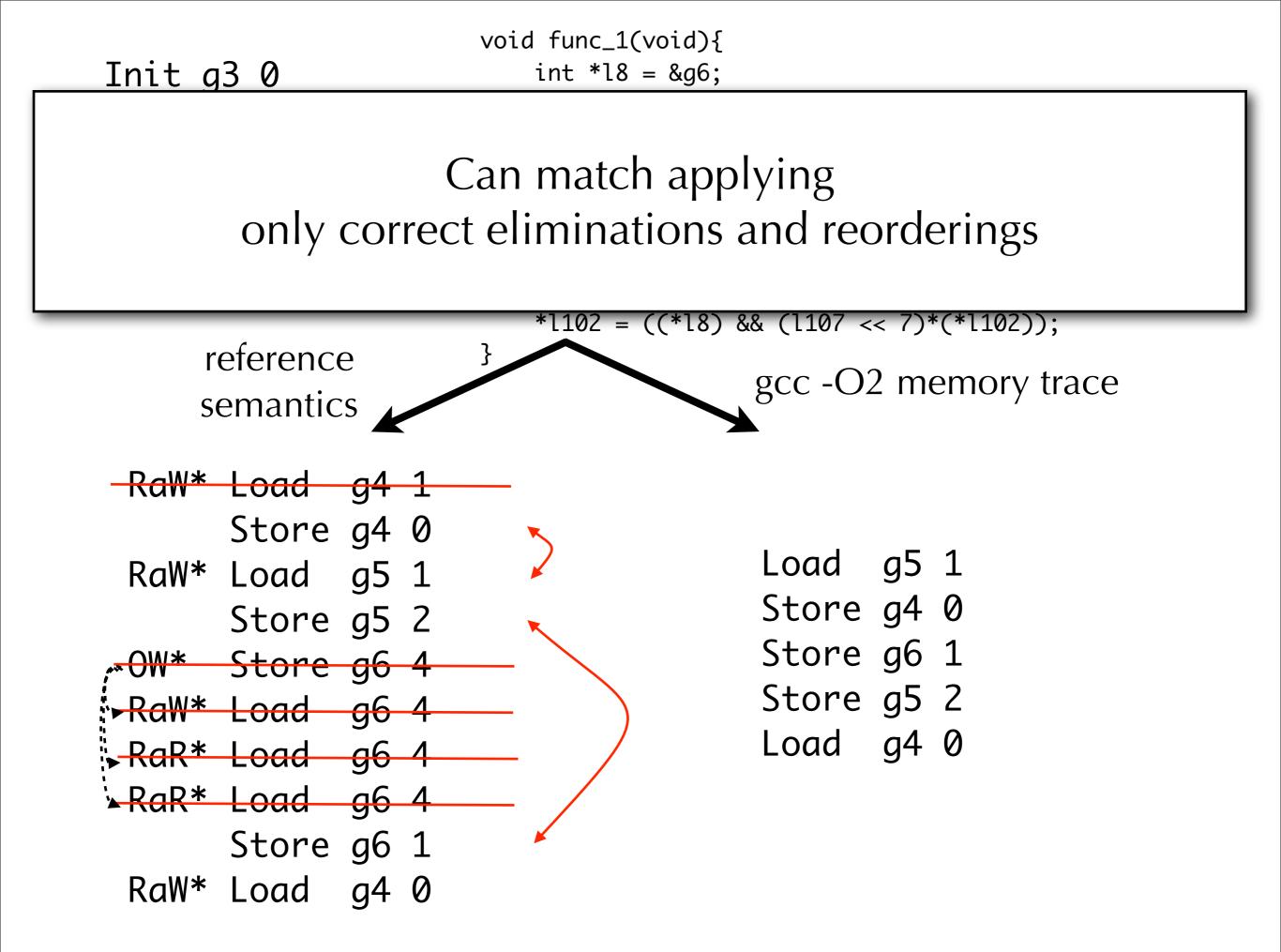
}



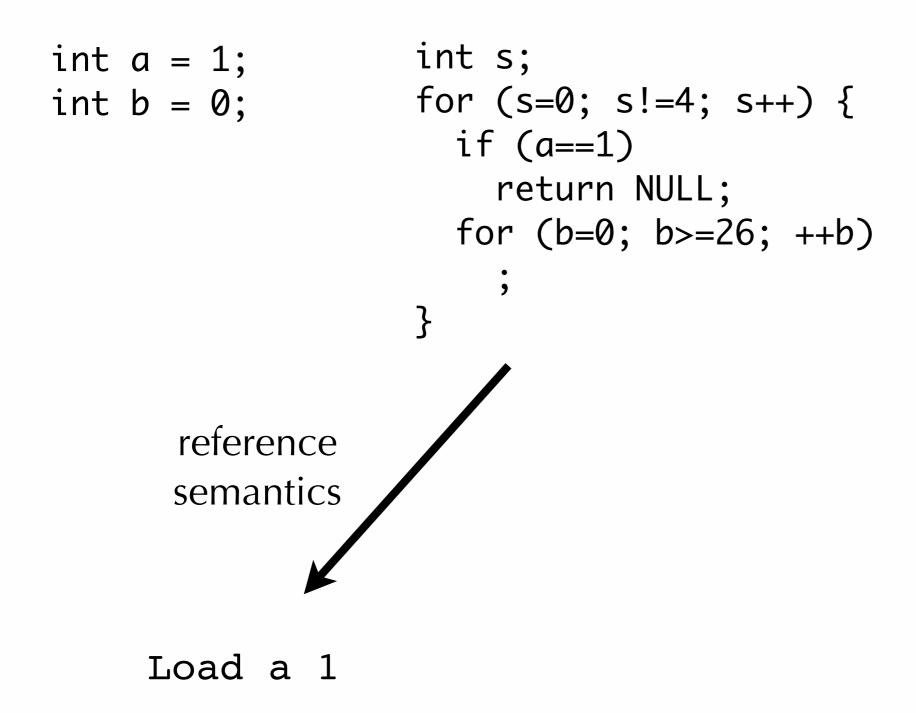


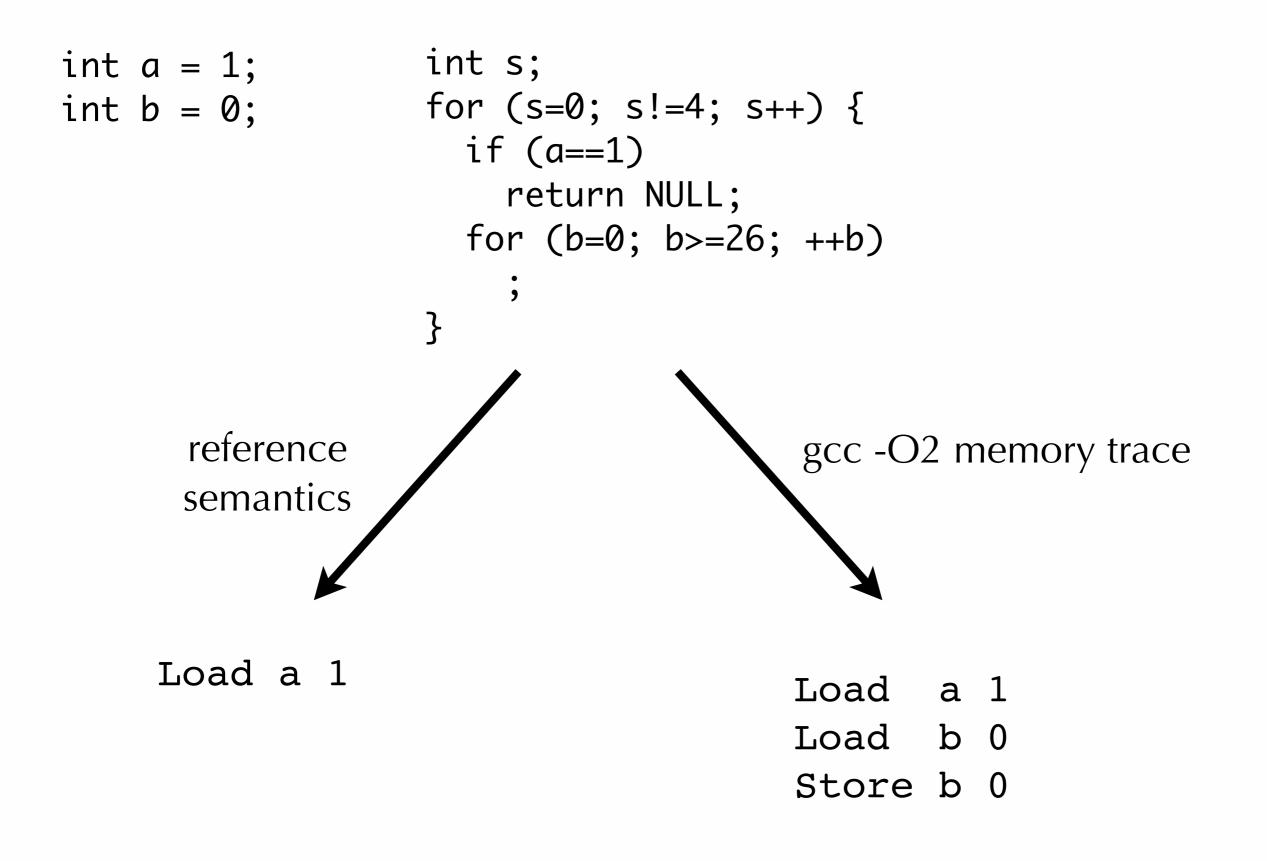




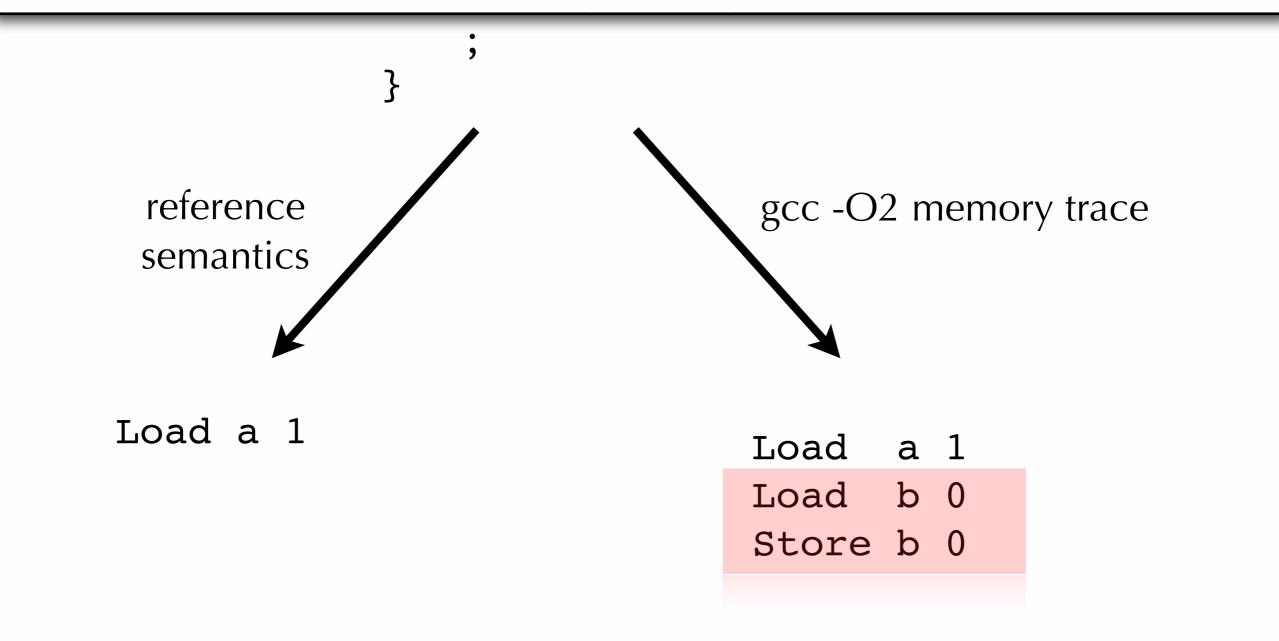


If we focus on the miscompiled initial example...





Cannot match some events — detect compiler bug



Applications



1. Testing C compilers (GCC, Clang, ICC)

Some concurrency compiler bugs found in the latest version of GCC.

Store introductions performed by loop invariant motion or if-conversion optimisations.

Remark: these bugs break the Posix thread model too.

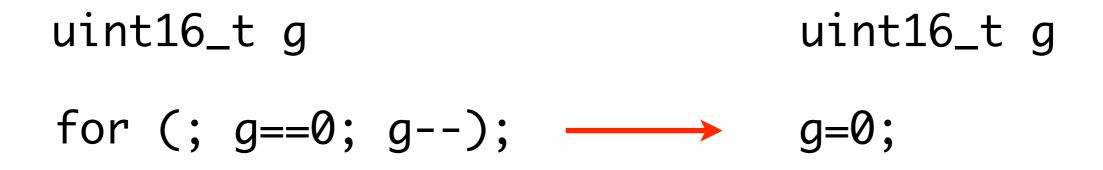
All promptly fixed.

2. Checking compiler invariants

GCC internal invariant: never reorder with an atomic access

Baked this invariant into the tool and found a counterexample... ...not a bug, but fixed anyway

3. Detecting unexpected behaviours



Correct or not?

3. Detecting unexpected behaviours

uint16_t g uint16_t g for (; g==0; g--); \longrightarrow g=0; If g is initialised with 0, a load gets replaced by a store:

Load g 0 ? (Store g 0

The introduced store cannot be observed by a non-racy context. Still, arguable if a compiler should do this or not.

3. Detecting unexpected behaviours

uint16_t g uint16_t g for (; g==0; g--); \longrightarrow g=0; If g is initialised with 0, a load gets replaced by a store: Load g 0 ? (Store g 0

False positives in Thread Sanitizer

The formalisation of the C11 memory model enables compiler testing... what else?



Proving the correctness of mappings for atomics

https://www.cl.cam.ac.uk/~pes20/cpp/cpp0xmappings.html

C/C++11 Operation	ARM implementation	
Load Relaxed:	ldr	
Load Consume:	ldr + preserve dependencies until next kill_dependency OR ldr; teq; beq; isb OR ldr; dmb	
Load Acquire:	ldr; teq; beq; isb OR ldr; dmb	
Load Seq Cst:	ldr; dmb	
Store Relaxed:	str	
Store Release:	dmb; str	
Store Seq Cst:	dmb; str; dmb	
Cmpxchg Relaxed (32 bit):	_loop: ldrex roldval, [rptr]; mov rres, 0; teq roldval, rold; strexeq rres, rnewval, [rptr]; teq rres, 0; bne _loop	
Cmpxchg Acquire (32 bit):	_loop: ldrex roldval, [rptr]; mov rres, 0; teq roldval, rold; strexeq rres, rnewval, [rptr]; teq rres, 0; bne _loop; isb	
Cmpxchg Release (32 bit):	dmb; _loop: ldrex roldval, [rptr]; mov rres, 0; teq roldval, rold; strexeq rres, rnewval, [rptr]; teq rres, 0; bne _loop;	
Cmpxchg AcqRel (32 bit):	dmb; _loop: ldrex roldval, [rptr]; mov rres, 0; teq roldval, rold; strexeq rres, rnewval, [rptr]; teq rres, 0; bne _loop; isb	
Cmpxchg SeqCst (32 bit):	dmb; _loop: ldrex roldval, [rptr]; mov rres, 0; teq roldval, rold; strexeq rres, rnewval, [rptr]; teq rres, 0; bne _loop; dmb	
Acquire Fence:	dmb	
Release Fence:	dmb	
	dmb	
SeqCst Fence:	dmb	

Inform new optimisations e.g. the work by Robin Morisset on the Arm LLVM backend

while (flag.load(acquire))
{}

.loop ldr r0, [r1] dmb ish

bnz .loop

```
.loop
ldr r0, [r1]
bnz .loop
dmb ish
```



Inform new optimisations e.g. the work by Robin Morisset on the Arm LLVM backend

while (flag.load(acquire))
{}

.loop ldr r0, [r1] dmb ish

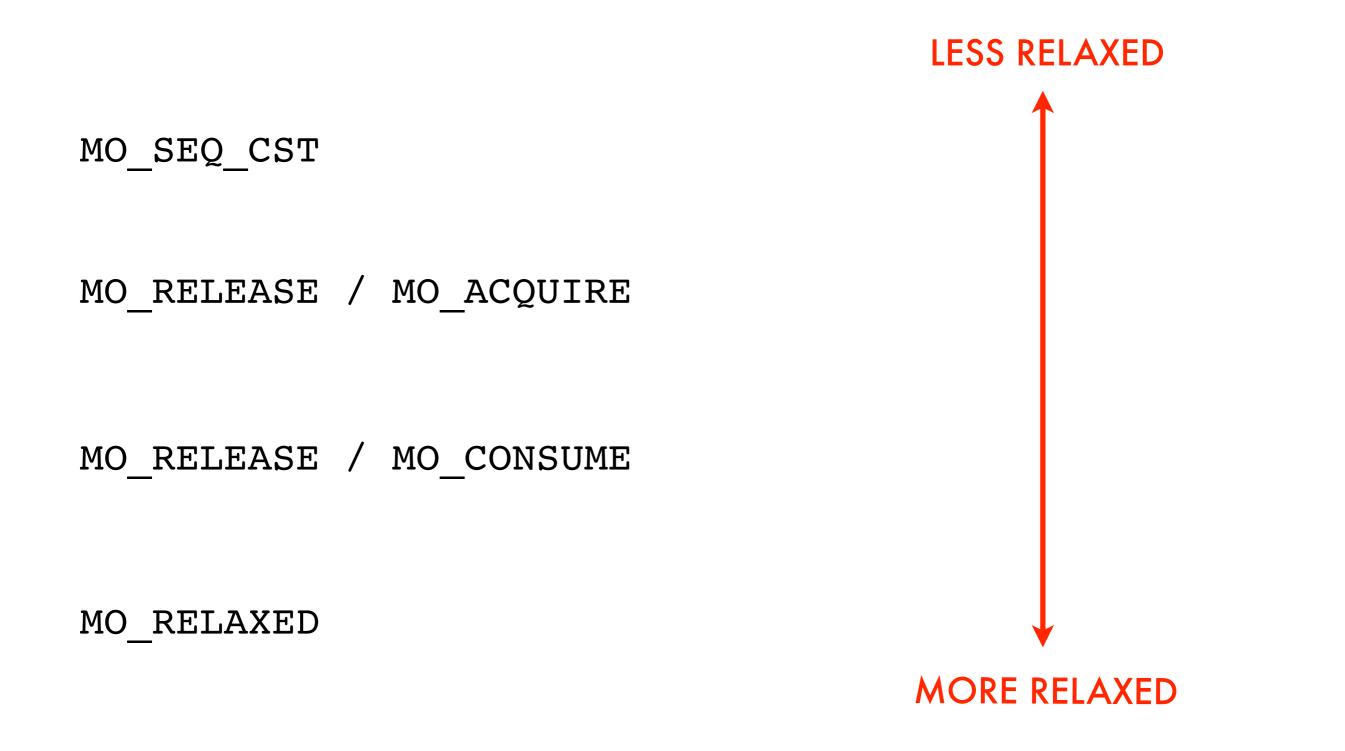
bnz .loop

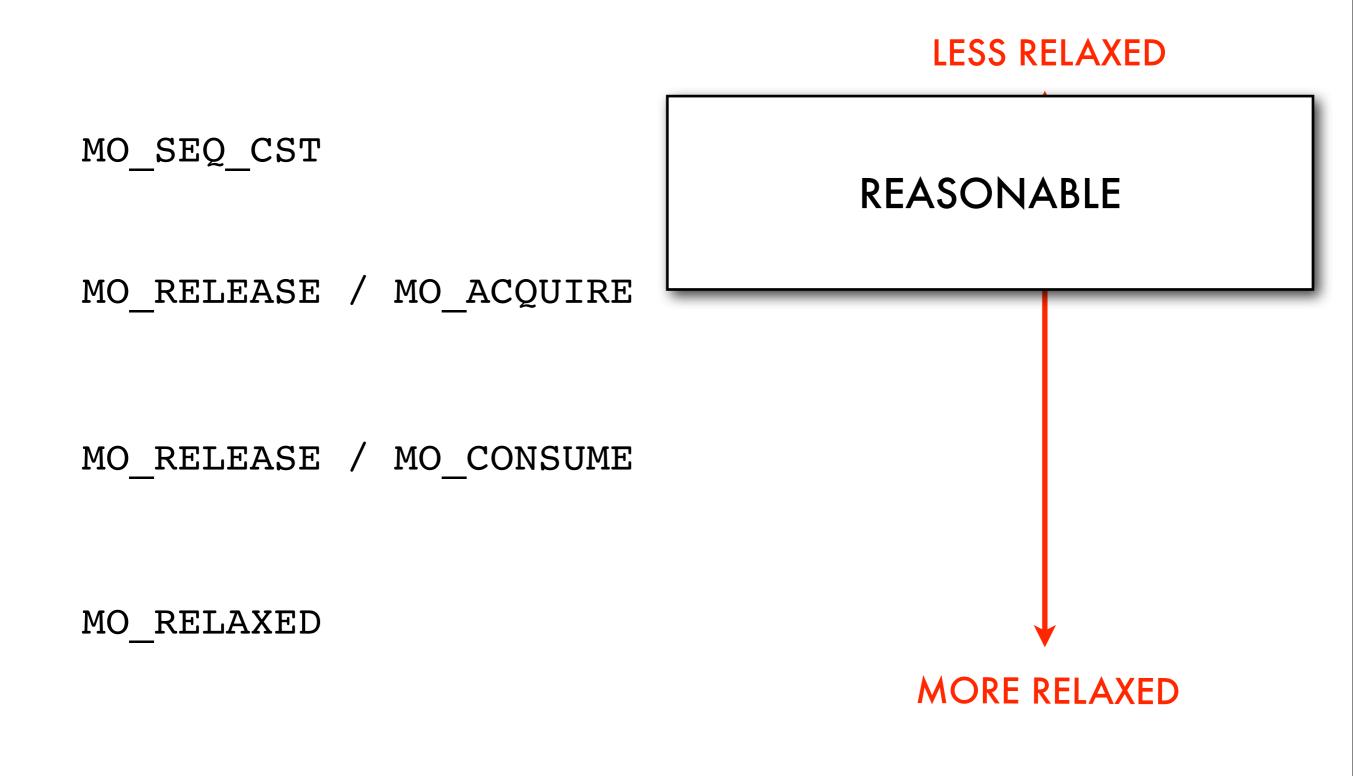
```
.loop
ldr r0, [r1]
bnz .loop
dmb ish
```

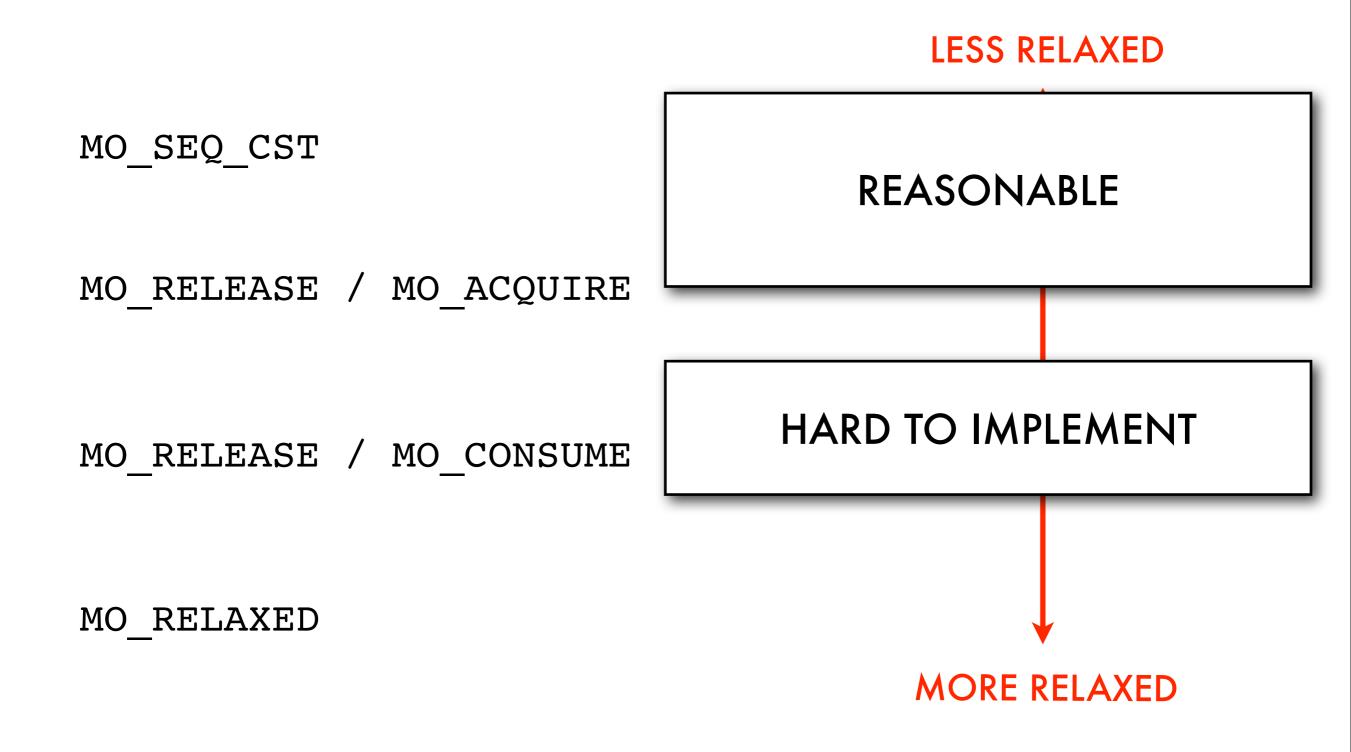


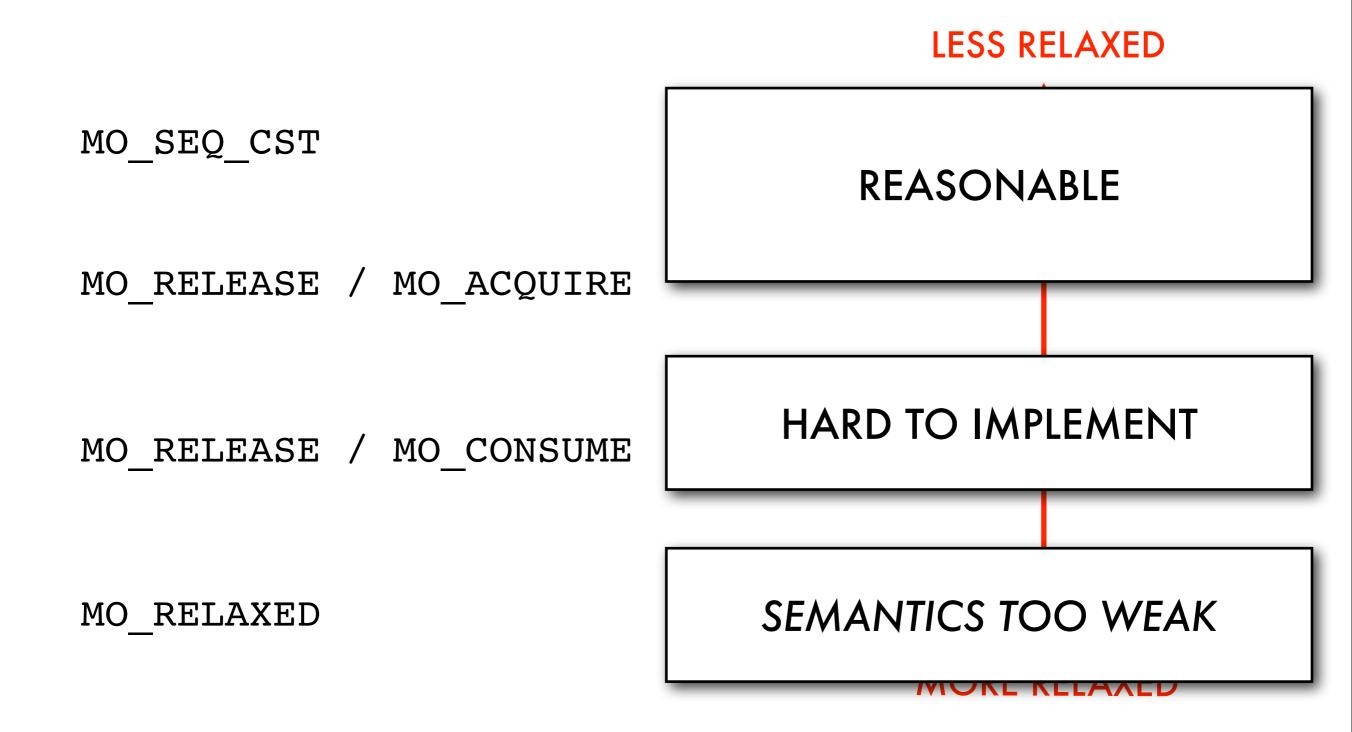
Not all of C/C++11 is good





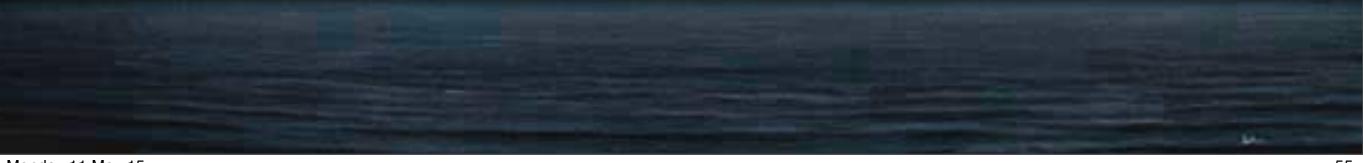








Out of thin air reads



Shorthand

from now on, all the memory accesses are atomic with MO_RELAXED semantics

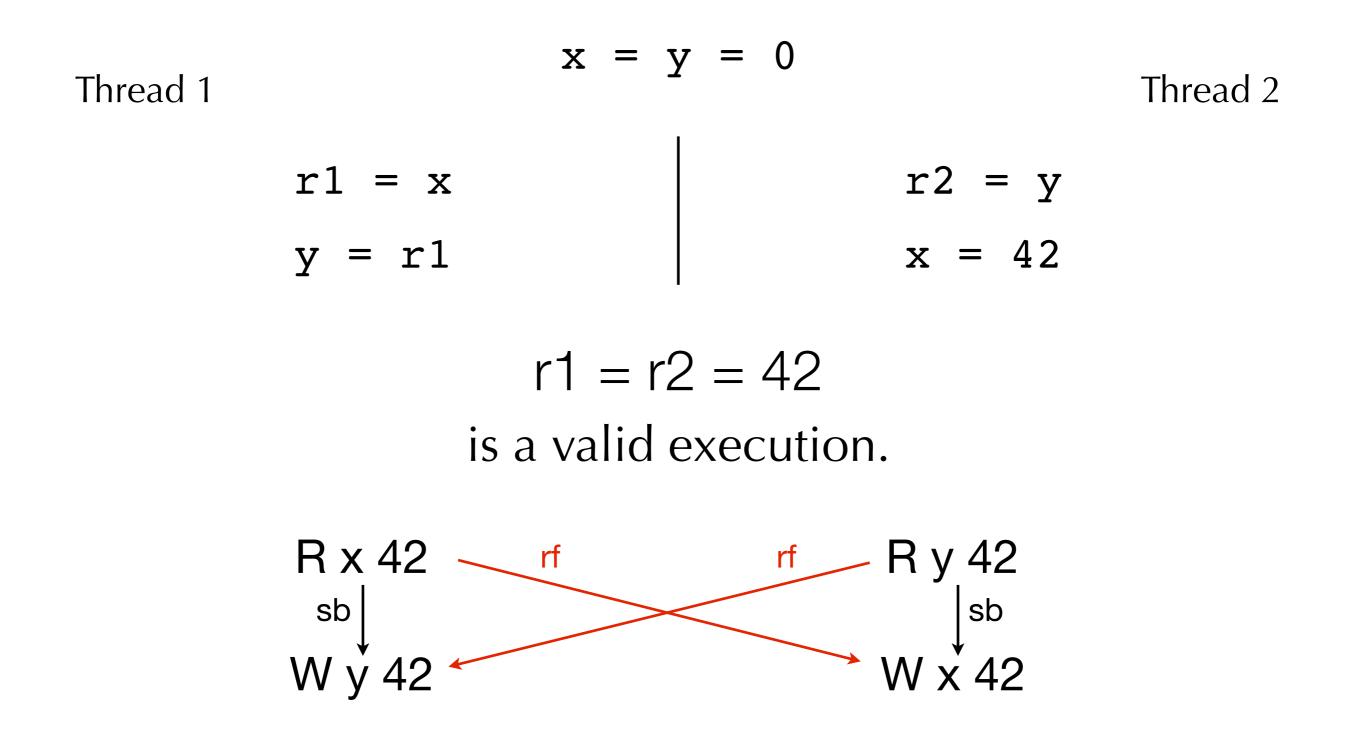
Relaxed atomics

Thread 1

$$r1 = x$$

 $y = r1$
 $x = y = 0$
 $r2 = y$
 $x = 42$
Thread 2

Relaxed atomics



Out-of-thin-air reads

Thread 1 x = y = 0 Thread 2 r1 = x r2 = yy = r1 x = r2

Out-of-thin-air reads

Thread 1

$$x = y = 0$$
Thread
$$r1 = x$$

$$y = r1$$

$$r2 = y$$

$$x = r2$$

$$r1 = r2 = 42$$

is also an allowed execution
$$R \times 42 \xrightarrow{rf} R \times 42 \xrightarrow{rf} W \times 42$$

2

the value 42 appears out-of-thin-air

Thread 1

$$x = y = 0$$
Thread 2
$$r1 = x$$

$$y = r1$$

$$x = y$$

$$r2 = y$$

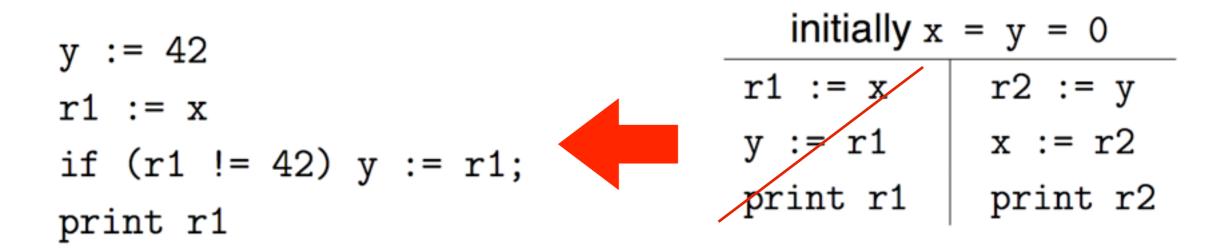
$$x = r2$$

$$r1 = r2 = 42$$

is also an allowed execution
$$R \times 42 \xrightarrow{rf} R \times 42 \xrightarrow{rf} W \times 42 \xrightarrow{sb} W \times 42$$

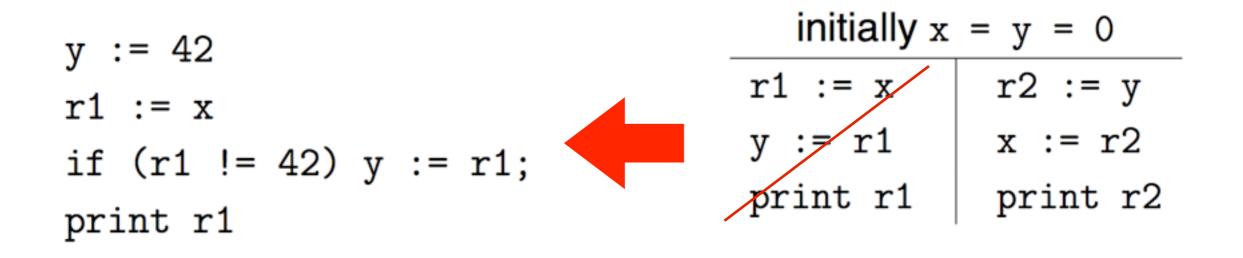
Speculation can justify out-of-thin-air reads

If the compiler states that x is likely to hold 42...



Speculation can justify out-of-thin-air reads

If the compiler states that x is likely to hold 42...



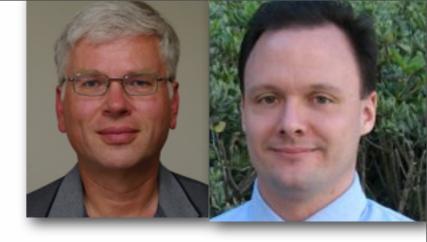
It does not happen in practice...

(a big thank you to compiler and hardware developers) ...but allowed by the standard



Consequences of out-of-thin-air reads

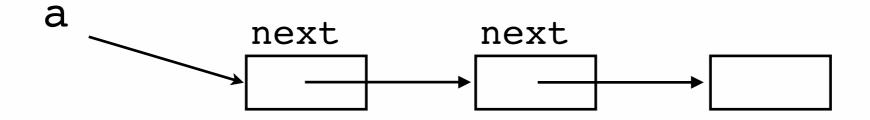
Monday 11 May 15

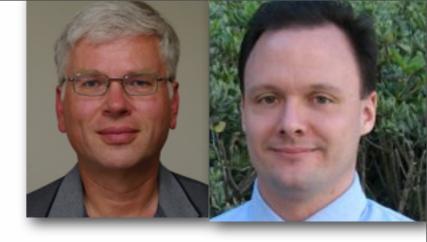


```
struct foo {
   atomic<struct foo *> next;
}
struct foo *a;
```

Thread 1

r1 = a - nextr1 - next = a

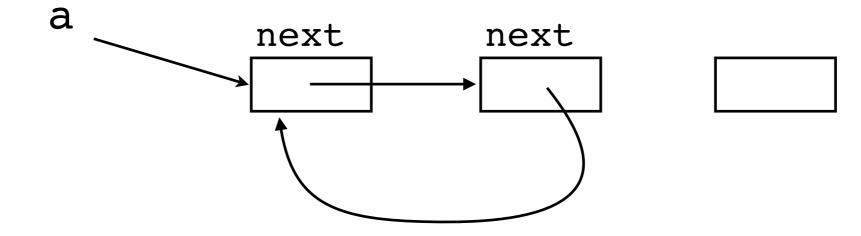


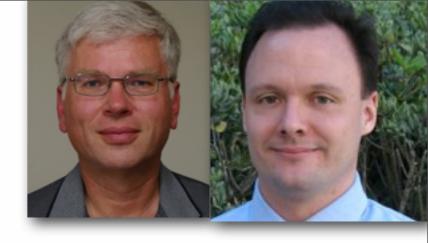


```
struct foo {
   atomic<struct foo *> next;
}
struct foo *a;
```

Thread 1

r1 = a - nextr1 - next = a





```
struct foo {
   atomic<struct foo *> next;
}
struct foo *a, *b;
```

Thread 1

Thread 2

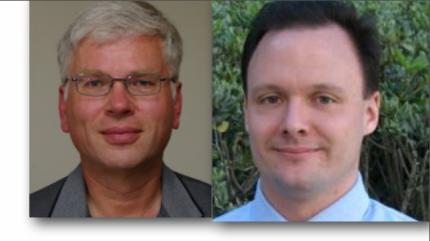
r1 = a - nextr1 - next = a r2 = b - next

r2 - next = b



<pre>struct foo { atomic<struct *="" foo=""> next; }</struct></pre>	
<pre>struct foo *a, *b;</pre>	
Thread 1	Thread 2
r1 = a->next	r2 = b - next
r1->next = a	r2 - next = b

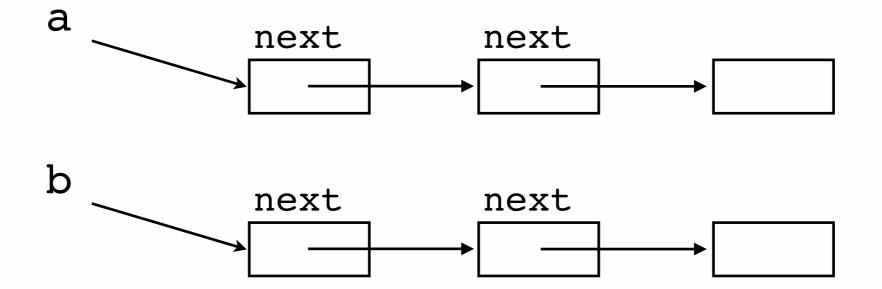
If a and b initially reference disjoint data-structures we expect a and b to remain disjoint



```
struct foo {
   atomic<struct foo *> next;
}
struct foo *a, *b;
```

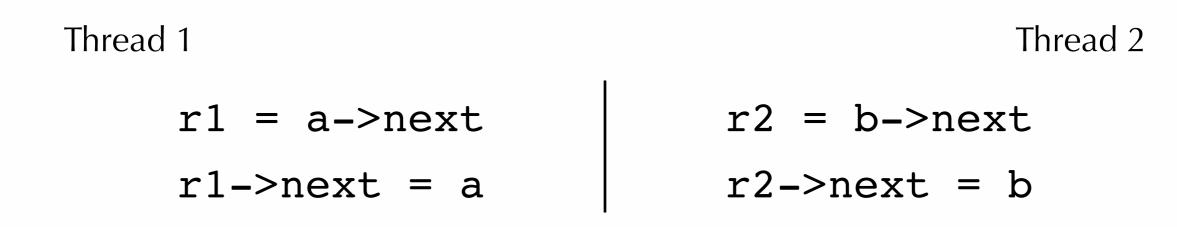
Thread 1 r1 = a - next r2 = b - next

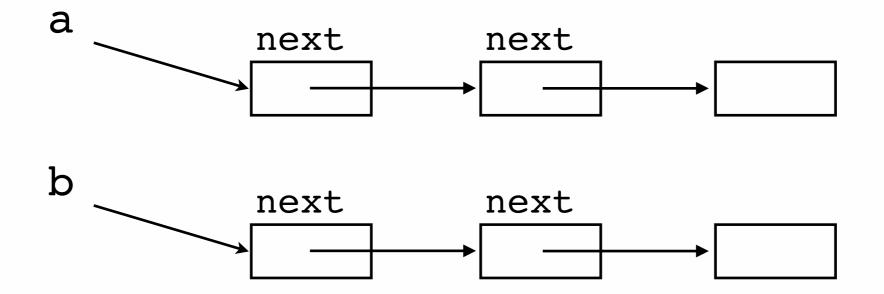
$$r1 \rightarrow next = a$$
 $r2 \rightarrow next = b$



If the compiler speculates r1=b and r2=a, then

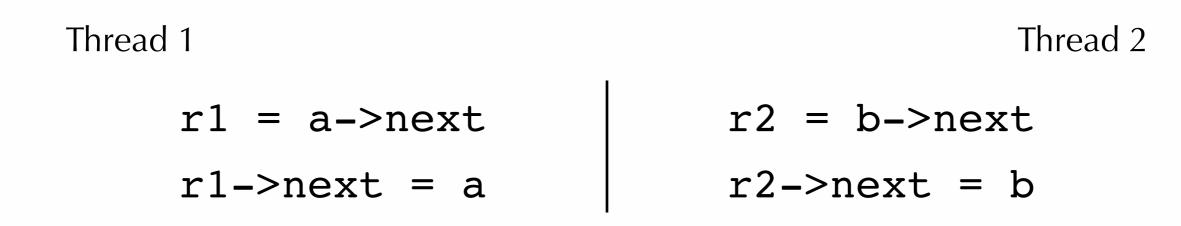
the store r1 - next = a justifies r2 = b - next assigning r2 = a(and symmetrically to justify r1 = b)

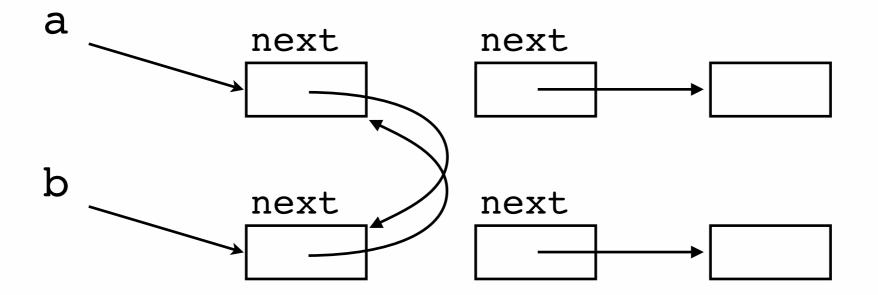




If the compiler speculates r1=b and r2=a, then

the store r1 - next = a justifies r2 = b - next assigning r2 = a(and symmetrically to justify r1 = b)



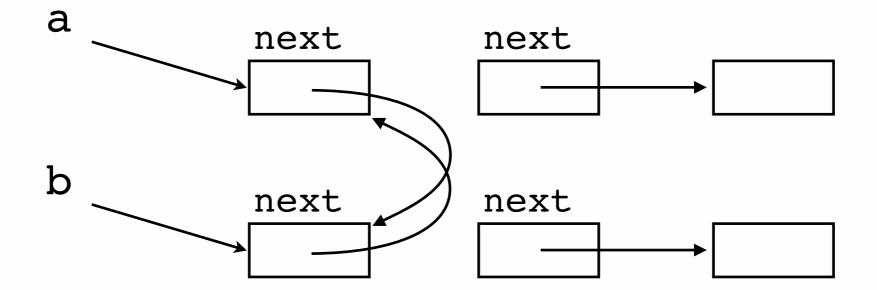


If the compiler speculates r1=b and r2=a, then

the store r1 - next = a justifies r2 = b - next assigning r2 = a

(and symmetrically to justify r1=b)

Break our basic intuitions about memory and sharing!



$$x = y = a = 0$$

н

Т

$$x = y = a = 0$$

Remark 1 This code is not racy!

There is no consistent execution in which the read of **a** occurs.

$$x = y = a = 0$$

Remark 2

$$\mathbf{a} = 1 \land \mathbf{x} = \mathbf{y} = 0$$

is the only possible final state

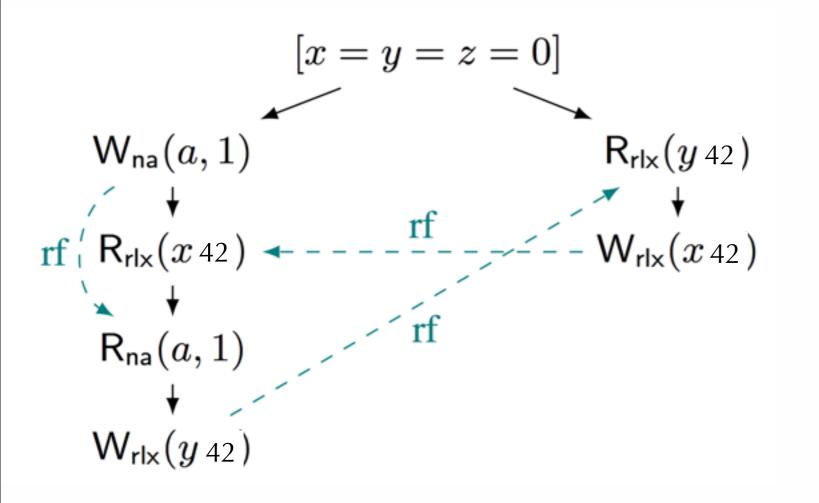
$$\mathbf{x} = \mathbf{y} = \mathbf{a} = \mathbf{0}$$

if (x.load(rlx)==42) if (y.load(rlx)==42) a = 1
y.write(42,rlx) if (a==1)
x.write(42,rlx)

Consider sequentialisation: $C \mid \mid D \implies C; D$ (ought to be correct)

$$x = y = a = 0$$

$$x = y = a = 0$$
if (x.load(rlx)==42)
y.write(42,rlx)
if (a==1)
x.write(42,rlx)



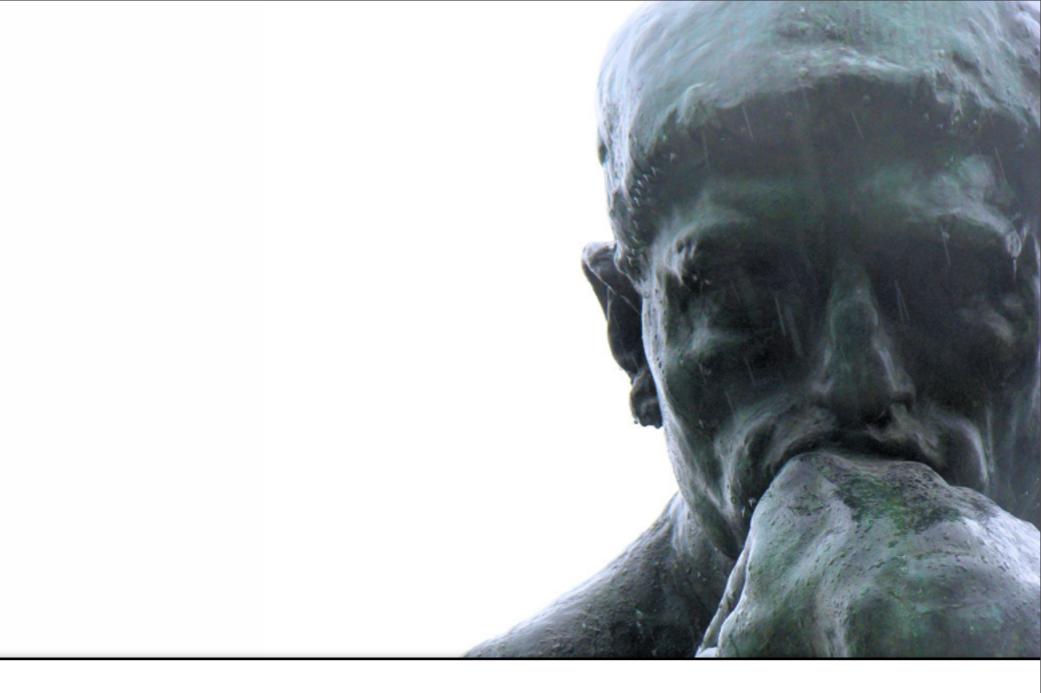
a = 1x = y = 42

is also possible

$$x = y = a = 0$$
if (x.load(rlx)==42)
y.write(42,rlx)
if (a==1)
x.write(42,rlx)

Break common source-to-source (or LLVM IR - to - LLVM IR) compiler optimisations

including expression linearisation and roach-motel reorderings



We still lack a really satisfactory proposal for the semantics of a general-purpose shared-memory concurrent programming language.



The way forward

3



Understand the effects of what compilers implement and programmers rely on

Build on that...

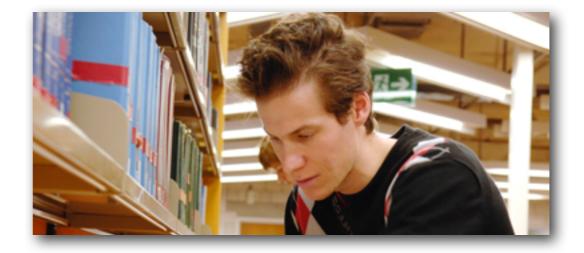


Beyond concurrency

Can one do < comparison or pointer arithmetic between pointers to separately allocated objects?

Routinely done in Linux kernel Forbidden by ISO standard





tinyurl.com/csurvey2

A web survey of 15 questions to investigate what C is in current practice: what behaviour is implemented by mainstream compilers and relied on by systems programmers





tinyurl.com/csurvey2

Eventual outcome: clear descriptions of what people can rely on and what compilers *in practice* should implement, what alias analysis and optimisation passes should (and should not) be allowed to do, etc.



tinyurl.com/csurvey2



Thank you. Questions?

