

Shared memory: an elusive abstraction

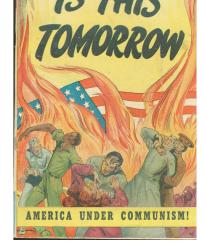
Francesco Zappa Nardelli

Inria Paris

http://www.di.ens.fr/~zappa/projects/weakmemory

Based on work done by or with

Peter Sewell, Jaroslav Ševčík, Susmit Sarkar, Tom Ridge, Scott Owens, Viktor Vafeiadis, Magnus O. Myreen, Kayvan Memarian, Luc Maranget, Derek Williams, Pankaj Pawan, Thomas Braibant, Mark Batty, Jade Alglave.



High-level languages, compilers, multiprocessors... an elusive mix?

Francesco Zappa Nardelli

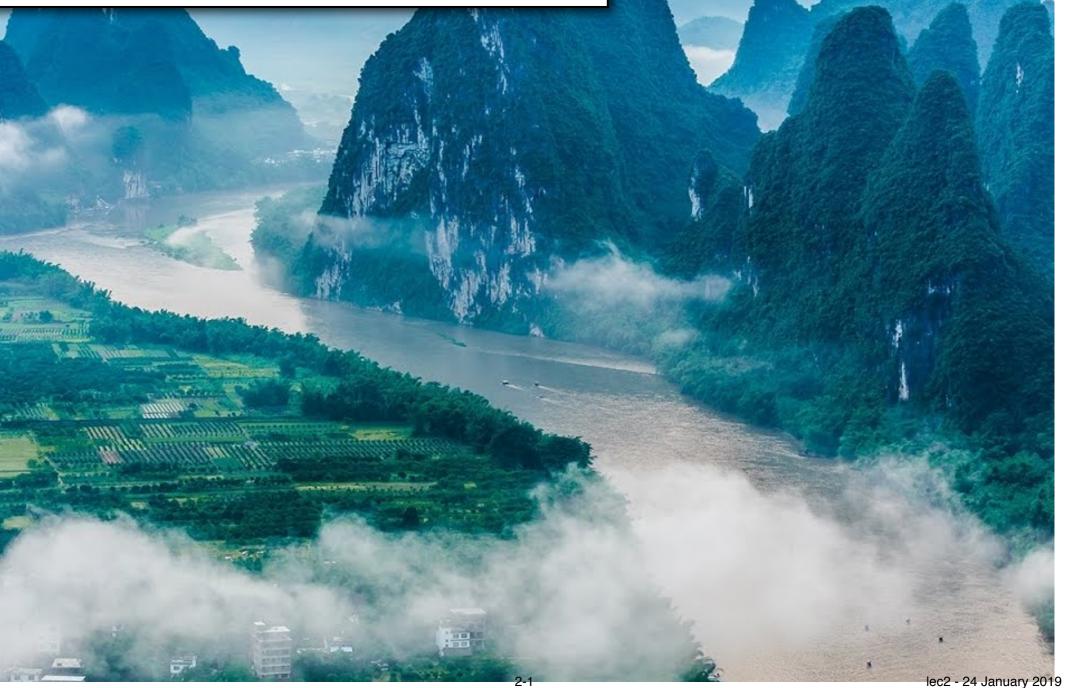
Inria Paris

http://www.di.ens.fr/~zappa/projects/weakmemory

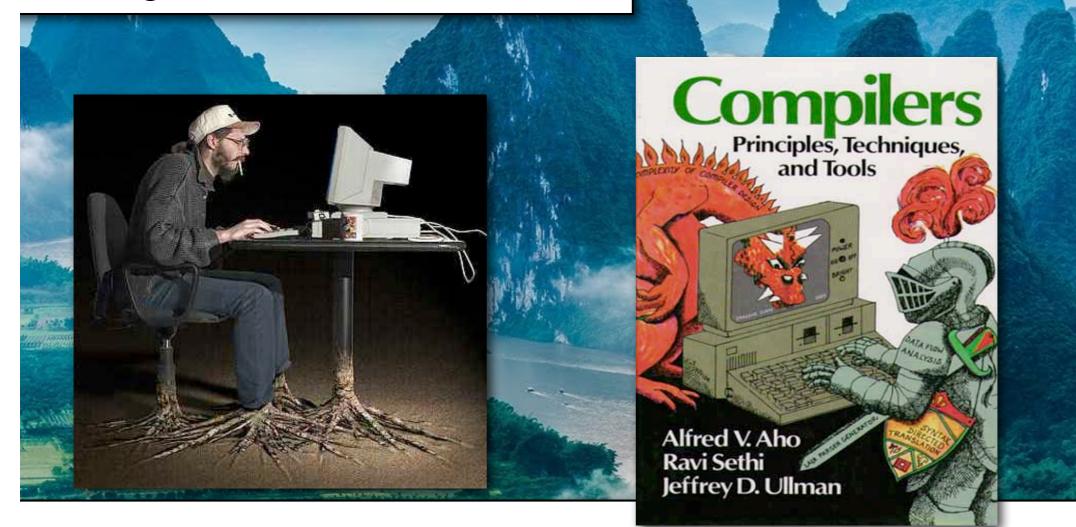
Based on work done by or with

Peter Sewell, Jaroslav Ševčík, Susmit Sarkar, Tom Ridge, Scott Owens, Viktor Vafeiadis, Magnus O. Myreen, Kayvan Memarian, Luc Maranget, Derek Williams, Pankaj Pawan, Thomas Braibant, Mark Batty, Jade Alglave.

Imagine an ideal world



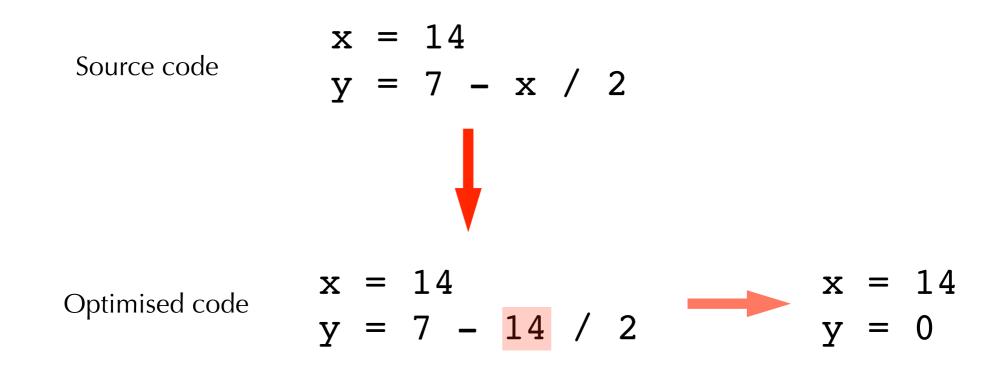
Imagine an ideal world



Programmers and compilers cooperate to make great software

Constant propagation

A simple, and *innocuous*, optimisation:



Shared memory

Thread 1

Shared memory

Τ

Intuitively this program always prints 0

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

$$x = y = 0$$

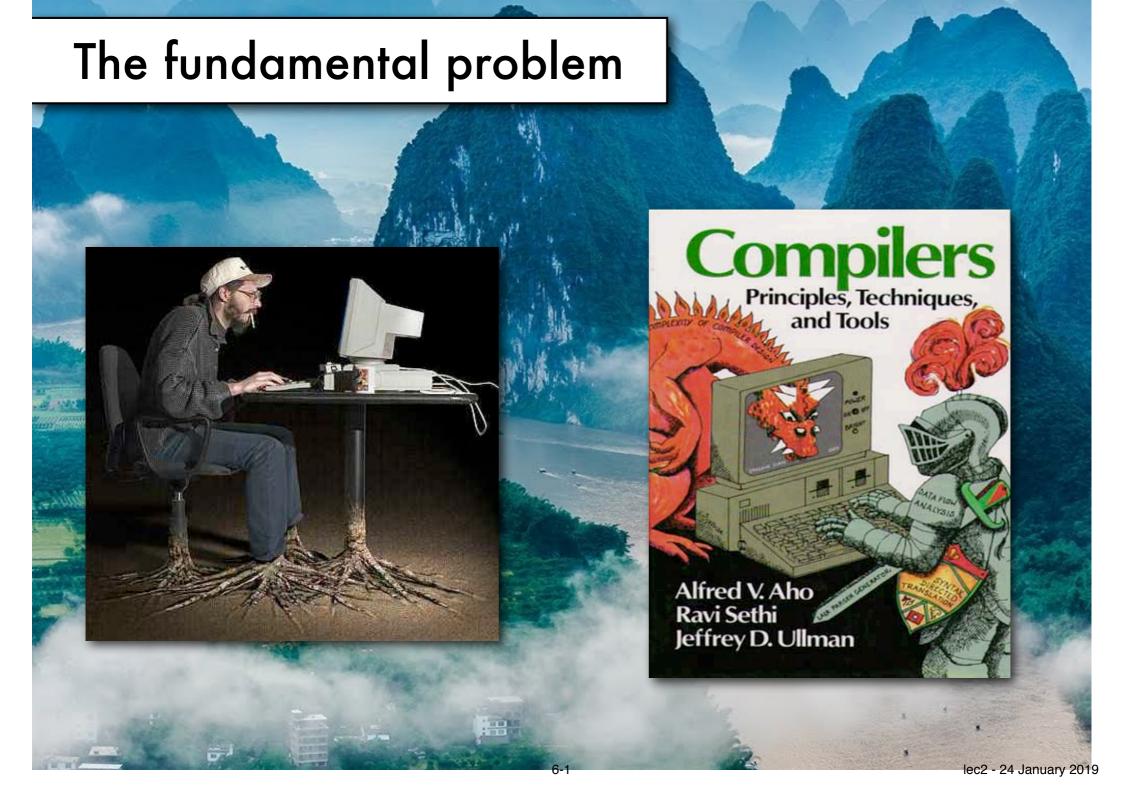
Thread 1

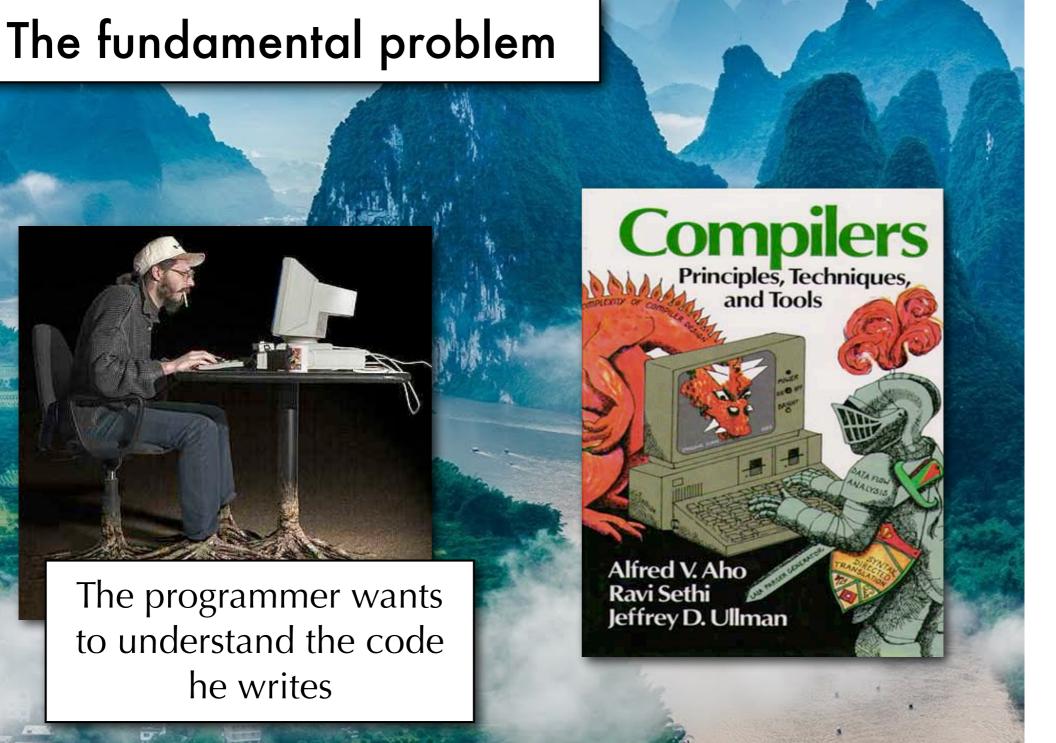
1

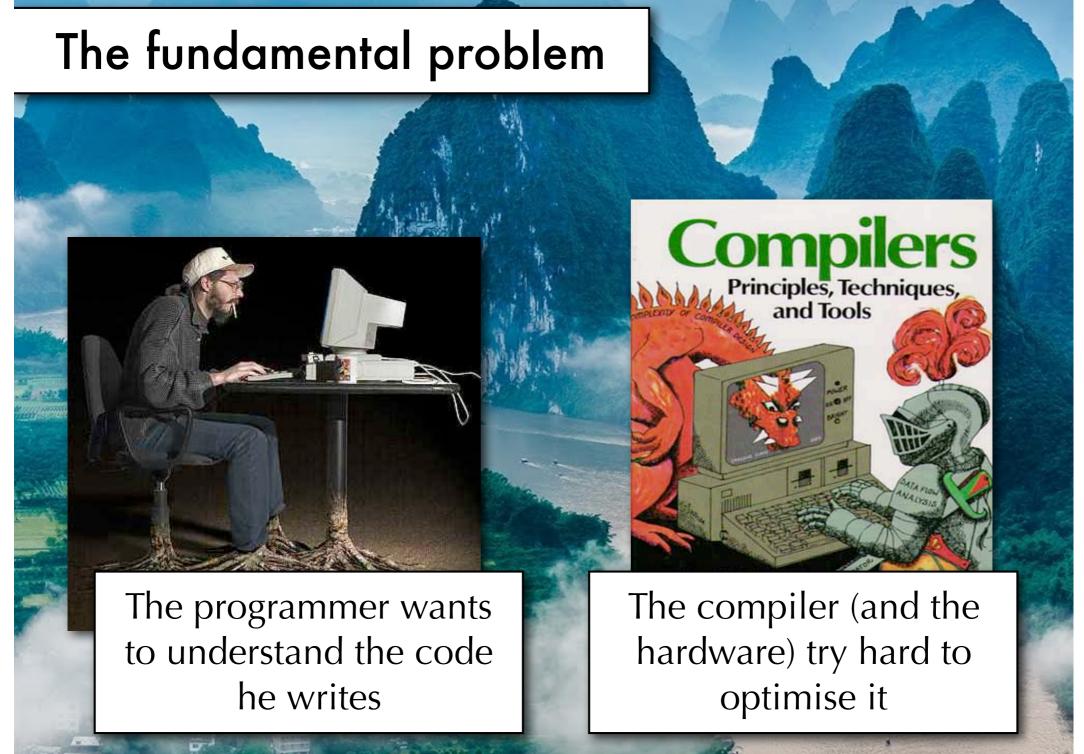
But if the compiler propagates the *constant* x = 1...

Thread 1

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







Background: lock and unlock

• Suppose that two threads increment a shared memory location:

• If both threads read 0, (even in an ideal world) x = 1 is possible:

Background: lock and unlock

• Lock and unlock are primitives that prevent the two threads from interleaving their actions.

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

lock();	<pre>lock(); tmp2 = *x; *x = tmp2 + 1;</pre>
tmp1 = *x;	tmp2 = *x;
*x = tmp1 + 1;	*x = tmp2 + 1;
unlock();	unlock();

• In this case, the interleaving below is forbidden, and we are guaranteed that x == 2 at the end of the execution.

Lazy initialisation (an unoptimising compiler breaks your program)

Deferring an object's initialisation util first use: a big win if an object is never used (e.g. device drivers code). Compare:

```
int x = computeInitValue(); // eager initialization
...
// clients refer to x
```

with:

```
int xValue() {
   static int x = computeInitValue(); // lazy initialization
   return x;
} ... // clients refer to xValue()
```

The singleton pattern

Lazy initialisation is a pattern commonly used. In C++ you would write:

```
class Singleton {
public:
  static Singleton *instance (void) {
    if (instance_ == NULL)
     instance = new Singleton;
    return instance ;
  }
                                 // other methods omitted
private:
  static Singleton *instance ; // other fields omitted
};
...
Singleton::instance () -> method ();
```

But this code is not thread safe! Why?

Making the singleton pattern thread safe

A simple thread safe version:

```
class Singleton {
public:
    static Singleton *instance (void) {
    Guard<Mutex> guard (lock_); // only one thread at a time
    if (instance_ == NULL)
        instance_ = new Singleton;
    return instance_;
    }
private:
    static Mutex lock_;
    static Singleton *instance_;
};
```

Every call to instance must acquire and release the lock: excessive overhead.

Obvious (broken) optimisation

```
class Singleton {
public:
  static Singleton *instance (void) {
  if (instance == NULL) {
    Guard<Mutex> guard (lock ); // lock only if unitialised
    instance_ = new Singleton; }
  return instance ;
  }
private:
 static Mutex lock ;
  static Singleton *instance ;
};
```

Exercise: why is it broken?

Clever programmers use double-check locking

```
class Singleton {
public:
 static Singleton *instance (void) {
  // First check
  if (instance == NULL) {
    // Ensure serialization
    Guard<Mutex> guard (lock_);
    // Double check
    if (instance == NULL)
       instance = new Singleton;
  }
  return instance ;
  }
private: [..]
};
```

Idea: re-check that the Singleton has not been created after acquiring the lock.

Double-check locking: clever but broken

```
The instruction
```

```
instance_ = new Singleton;
```

does three things:

- 1) allocate memory
- 2) construct the object
- 3) assign to instance_ the address of the memory

Not necessarily in this order! For example:

```
instance_ = // 3
operator new(sizeof(Singleton)); // 1
new (instance_) Singleton // 2
```

If this code is generated, the order is 1,3,2.

Broken...

```
if (instance_ == NULL) { // Line 1
Guard<Mutex> guard (lock_);
if (instance_ == NULL) {
    instance_ =
        operator new(sizeof(Singleton)); // Line 2
        new (instance_) Singleton; }}
```

Thread 1:

executes through Line 2 and is suspended; at this point, instance_ is non-NULL, but no singleton has been constructed.

Thread 2:

executes Line 1, sees instance_ as non-NULL, returns, and dereferences the pointer returned by Singleton (i.e., instance_).

Thread 2 attempts to reference an object that is not there yet!

Problem: You need a way to specify that step 3 come after steps 1 and 2.

There is no way to specify this in C++

Similar examples can be built for any programming language...

That pesky hardware (1)

Consider misaligned 4-byte accesses

(Disclaimer: compiler will normally ensure alignment)

Intel SDM x86 atomic accesses:

- *n*-bytes on an *n*-byte boundary (n = 1, 2, 4, 16)
- P6 or later: ... or if unaligned but within a cache line

Question: what about multi-word high-level language values?

That pesky hardware (1)

Consider misaligned 4-byte accesses

	$int32_t a = 0$			
	a =	0x44332211	if (a == 0x00002211) print "error"	
(Discla	aime			
Intel S	SDN	This is called a <i>out-of-thin air read</i> :		
• <i>n</i> -k	oyte	the program reads a value		
• P6	orl	that the programmer never wrote.		
Question. what about multi-word high-leven language values?				

That pesky hardware (2)

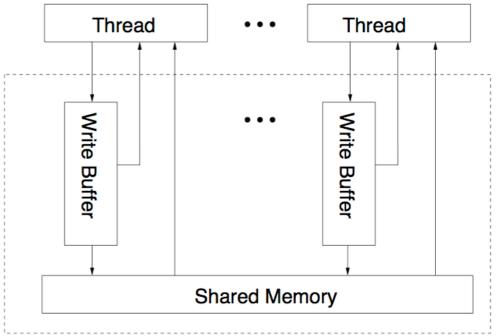
Hardware optimisations can be observed by concurrent code:

Thread 0	Thread 1
x = 1	y = 1
print y	print x

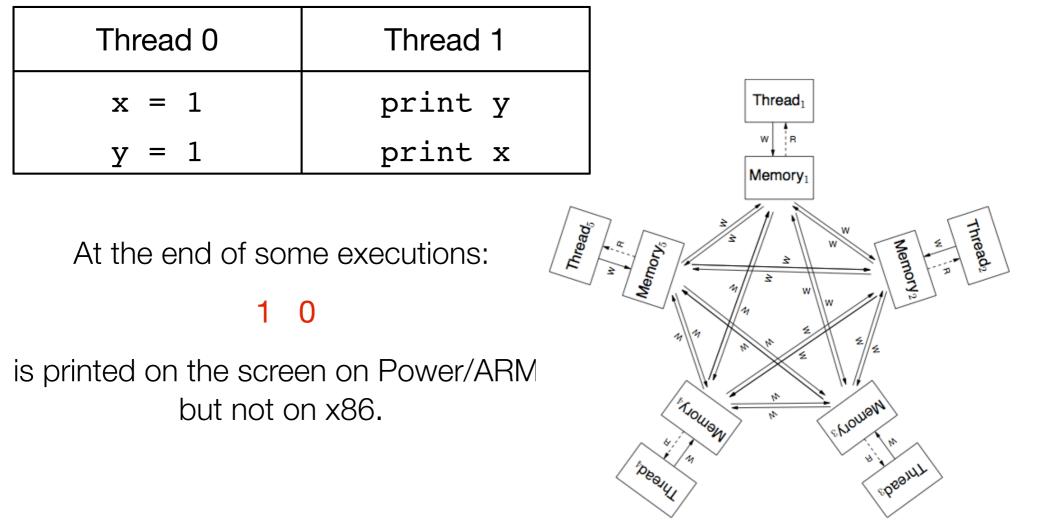
At the end of some executions:

0 0

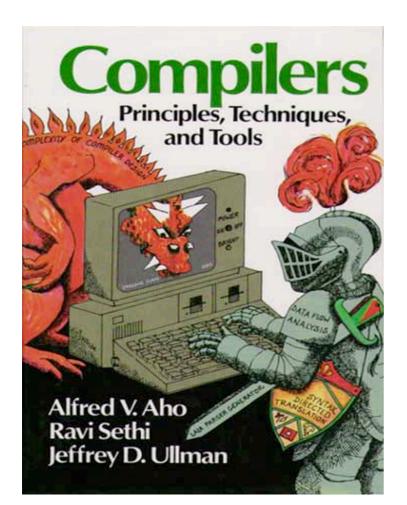
is printed on the screen, both on x86 and Power/ARM).



...and differ between architectures...



Compilers vs. programmers





Compilers vs. programmers

Tension:

- the programmer wants to understand the code he writes
- the compiler and the hardware want to optimise it.

Which are the valid optimisations that the compiler or the hardware can perform without breaking the expected semantics of a concurrent program?

Which is the semantics of a concurrent program?

This lecture

Programming language models

- 1) defining the semantics of a concurrent programming language
- 2) data-race freedom
- 3) soundness of compiler optimisations

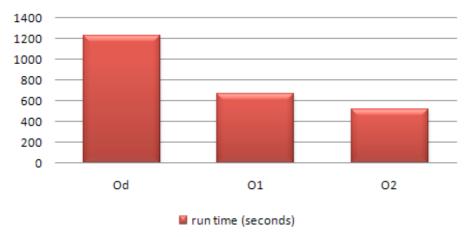
Previous lecture: hardware models

1) why are industrial specs so often flawed?

focus on x86, with a glimpse of Power/ARM

2) usable models: x86-TSO, PowerARM

effect of VS2005 compiler optimisations on speed



A brief tour of compiler optimisations



effect of additional VS2005 optimisations on speed

run time (seconds)

A typical compiler performs many optimisations.

gcc 4.4.1. with –O2 option goes through 147 compilation passes.

computed using -fdump-tree-all and -fdump-rtl-all

Sun Hotspot Server JVM has 18 high-level passes with each pass composed of one or more smaller passes.

http://www.azulsystems.com/blog/cliff-click/2009-04-14-odds-ends

World of optimisations

A typical compiler performs many optimisations.

- Common subexpression elimination
 - (copy propagation, partial redundancy elimination, value numbering)
- (conditional) constant propagation
- dead code elimination
- loop optimisations

(loop invariant code motion, loop splitting/peeling, loop unrolling, etc.)

- vectorisation
- peephole optimisations
- tail duplication removal
- building graph representations/graph linearisation
- register allocation
- call inlining
- local memory to registers promotion
- spilling
- instruction scheduling

World of optimisations

However only some optimisations change shared-memory traces:

- Common subexpression elimination

(copy propagation, partial redundancy elimination, value numbering)

- (conditional) constant propagation
- dead code elimination
- loop optimisations

(loop invariant code motion, loop splitting/peeling, loop unrolling, etc.)

- vectorisation
- peephole optimisations
- tail duplication removal
- building graph representations/graph linearisation
- register allocation
- call inlining
- local memory to registers promotion
- spilling
- instruction scheduling

What is an optimisation?

Compiler Writer



Semanticist



What is an optimisation?

Compiler Writer



Sophisticated program analyses Fancy algorithms Source code or IR *Operations on AST*

Semanticist



What is an optimisation?

Compiler Writer



Sophisticated program analyses Fancy algorithms Source code or IR *Operations on AST*

Semanticist



Compiler Writer



Sophisticated program analyses Fancy algorithms Source code or IR *Operations on AST*

Semanticist



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



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

Load y 42 Store z 0

Store x[0] 43 Store z 1

Store x[1] 43

Eliminations

This includes common subexpression elimination, dead read elimination, overwritten write elimination, redundant write elimination.

Irrelevant read elimination:

 $r=*x; C \rightarrow C$

where \mathbf{r} is not free in \mathbf{C} .

Redundant read after read elimination:

 $r1=*x; r2=*x \rightarrow r1=*x; r2=r1$

Redundant read after write elimination:

*x=r1; r2=*x → *x=r1; r2=r1

Reordering

Common subexpression elimination, some loop optimisations, code motion.

Normal memory access reordering:

Roach motel reordering:

memop; lock m → lock m; memop unlock m; memop → memop; unlock m where memop is *x=r1 or r1=*x Memory access introduction

Can an optimisation introduce memory accesses?

Yes, but rarely:

i = 0; while (i != 0) { j = *x + 1; i = i-1 } i = i-1 } i = i-1 } i = i-1 } i = 0; tmp = *x; while (i != 0) { j = tmp + 1;i = i-1 }

Note that the loop body is not executed.

Memory access introduction

Back to our question now:

Which is the semantics of a concurrent program?

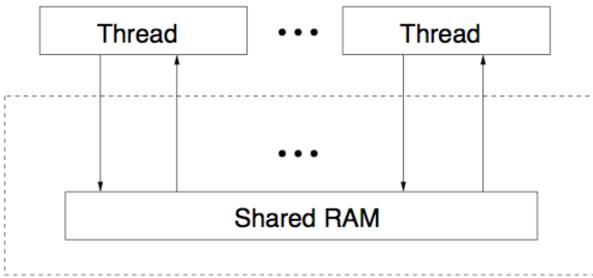
Mule lhat the loop body is hot executed.

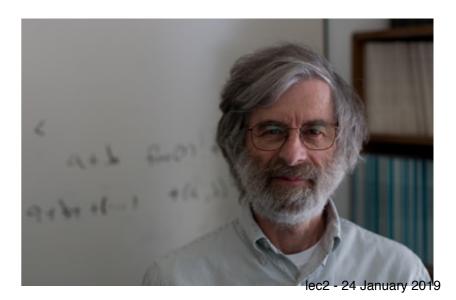
Naive answer: enforce sequential consistency

Multiprocessors have a *sequentially consistent* shared memory when:

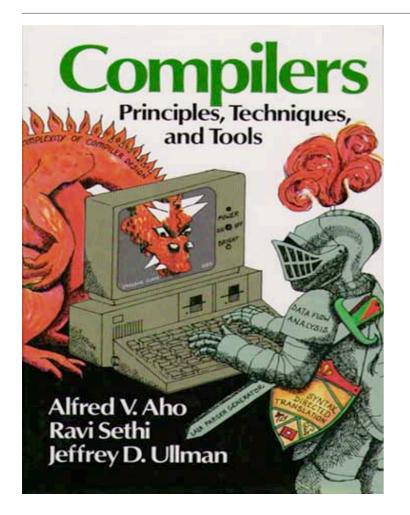
...the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program...

Lamport, 1979.



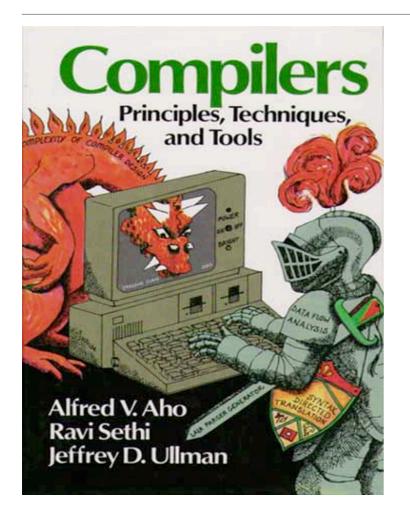


Compilers, programmers & sequential consistency





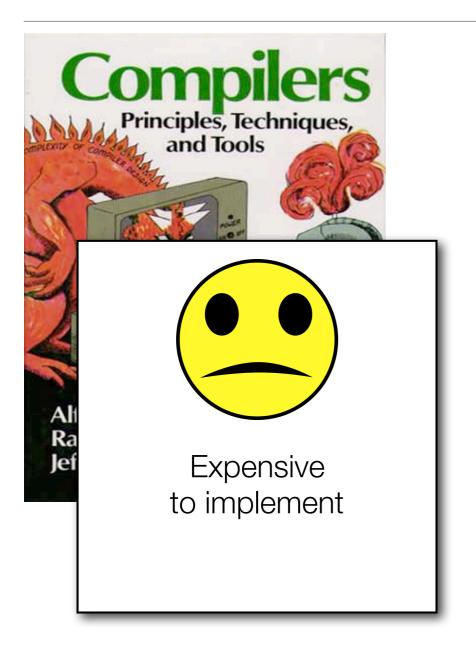
Compilers, programmers & sequential consistency

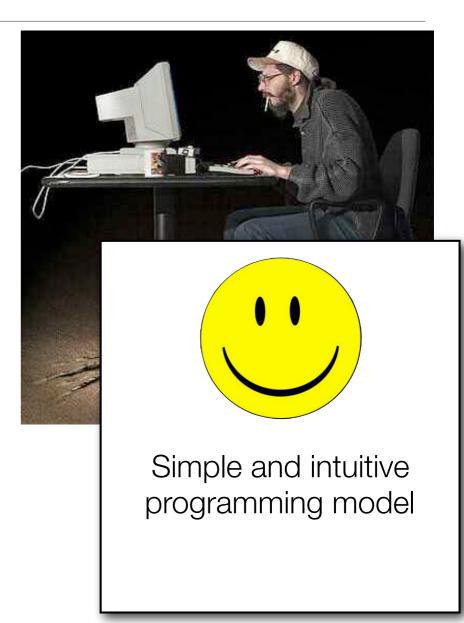




Simple and intuitive programming model

Compilers, programmers & sequential consistency



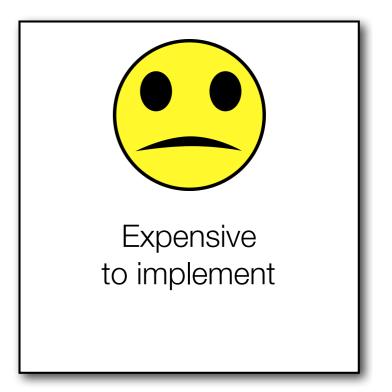


A Case for an SC-Preserving Compiler

Daniel Marino[†] Abhayendra Singh^{*} Todd Millstein[†] Madanlal Musuvathi[‡] Satish Narayanasamy^{*} [†]University of California, Los Angeles ^{*}University of Michigan, Ann Arbor [‡]Microsoft Research, Redmond

An SC-preserving compiler, obtained by restricting the optimization phases in LLVM, a state-of-the-art C/C++ compiler, incurs an average slowdown of 3.8% and a maximum slowdown of 34% on a set of 30 programs from the SPLASH-2, PARSEC, and SPEC CINT2006 benchmark suites.

And this study supposes that the hardware is SC.



SC and hardware

The compiler must insert enough synchronising instructions to prevent hardware reorderings. On x86 we have:

• MFENCE

flush the local write buffer

• LOCK prefix (e.g. CMPXCHG) flush the local write buffer globally lock the memory

Initial: [x]=0 ^ [y]=0		
proc 0	proc 1	
MOV [x]←\$1	MOV [y]←\$1	
MFENCE	MFENCE	
MOV EAX←[y]	MOV EBX←[x]	
Forbid: EAX=0 \land EBX=0		

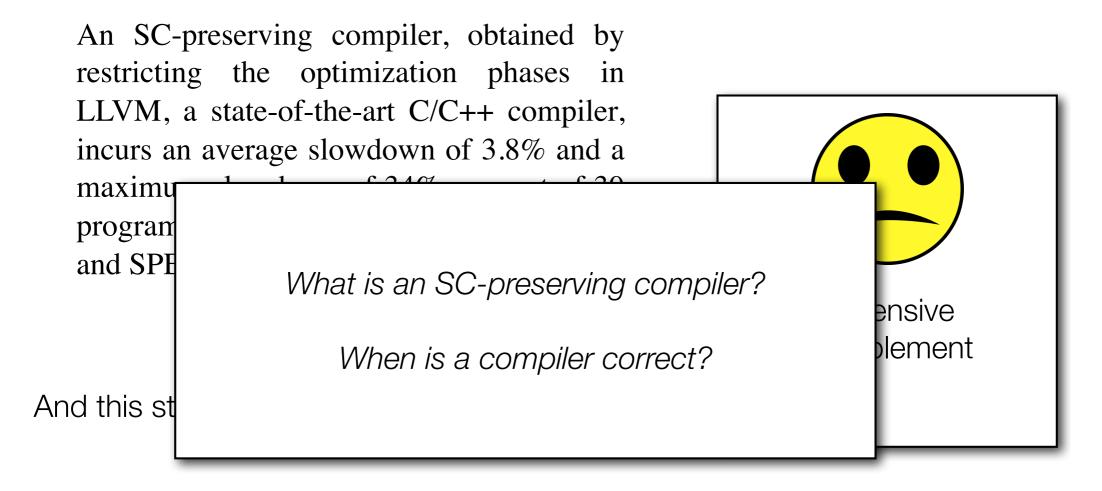
Initally, $[100] = 0$	proc:0	proc:1
At the end, [100] = 2	LOCK; INC [100]	LOCK; INC [100]

These consumes hundreds of cycles... ideally should be avoided.

Naively recovering SC on x86 incurs in a ~40% overhead.

A Case for an SC-Preserving Compiler

Daniel Marino[†] Abhayendra Singh^{*} Todd Millstein[†] Madanlal Musuvathi[‡] Satish Narayanasamy^{*} [†]University of California, Los Angeles ^{*}University of Michigan, Ann Arbor [‡]Microsoft Research, Redmond



A compiler is correct if any behaviour of the compiled program could be exhibited by the original program.

i.e. for any execution of the compiled program, there is an execution of the source program with the same observable behaviour.

Intuition: we represent programs as sets of memory action traces, where the trace is a sequence of memory actions of a single thread.

Intuition: the observable behaviour of an execution is the subtrace of external actions.

$$P_1 = *x = 1 \begin{vmatrix} r1 = *x; r2 = *x; \\ if r1=r2 \text{ then print 1 else print 2} \end{vmatrix}$$
$$P_2 = *x = 1 \begin{vmatrix} r1 = *x; r2 = r1; \\ if r1=r2 \text{ then print 1 else print 2} \end{vmatrix}$$

Is the transformation from P1 to P2 correct (in an SC semantics)?

$$P_1 = *x = 1 \begin{vmatrix} r1 = *x; r2 = *x; \\ if r1=r2 \text{ then print 1 else print 2} \end{vmatrix}$$
$$P_2 = *x = 1 \begin{vmatrix} r1 = *x; r2 = r1; \\ if r1=r2 \text{ then print 1 else print 2} \end{vmatrix}$$

$$P_1 = *x = 1 \begin{vmatrix} r1 = *x; r2 = *x; \\ if r1=r2 \text{ then print 1 else print 2} \end{vmatrix}$$

$$P_2 = *x = 1 \begin{vmatrix} r1 = *x; r2 = r1; \\ if r1=r2 \text{ then print 1 else print 2} \end{vmatrix}$$

Executions of P1:

$$\begin{split} & \mathsf{W}_{t_1} \, x{=}1, \mathsf{R}_{t_2} \, x{=}1, \mathsf{R}_{t_2} \, x{=}1, \mathsf{P}_{t_2} \, 1 \\ & \mathsf{R}_{t_2} \, x{=}0, \mathsf{W}_{t_1} \, x{=}1, \mathsf{R}_{t_2} \, x{=}1, \mathsf{P}_{t_2} \, 2 \\ & \mathsf{R}_{t_2} \, x{=}0, \mathsf{R}_{t_2} \, x{=}0, \mathsf{W}_{t_1} \, x{=}1, \mathsf{P}_{t_2} \, 1 \\ & \mathsf{R}_{t_2} \, x{=}0, \mathsf{R}_{t_2} \, x{=}0, \mathsf{P}_{t_2} \, 1, \mathsf{W}_{t_1} \, x{=}1 \end{split}$$

$$P_1 = \mathbf{*x} = 1 \begin{vmatrix} \mathbf{r1} = \mathbf{*x}; & \mathbf{r2} = \mathbf{*x}; \\ \text{if } \mathbf{r1} = \mathbf{r2} & \text{then print 1 else print 2} \end{vmatrix}$$

$$P_2 = \mathbf{*x} = 1 \begin{vmatrix} \mathbf{r1} = \mathbf{*x}; & \mathbf{r2} = \mathbf{r1}; \\ \text{if } \mathbf{r1} = \mathbf{r2} & \text{then print 1 else print 2} \end{vmatrix}$$

Executions of P1:

Executions of P2:

$$\begin{split} & \mathsf{W}_{t_1} \, x{=}1, \mathsf{R}_{t_2} \, x{=}1, \mathsf{R}_{t_2} \, x{=}1, \mathsf{P}_{t_2} \, 1 \\ & \mathsf{R}_{t_2} \, x{=}0, \mathsf{W}_{t_1} \, x{=}1, \mathsf{R}_{t_2} \, x{=}1, \mathsf{P}_{t_2} \, 2 \\ & \mathsf{R}_{t_2} \, x{=}0, \mathsf{R}_{t_2} \, x{=}0, \mathsf{W}_{t_1} \, x{=}1, \mathsf{P}_{t_2} \, 1 \\ & \mathsf{R}_{t_2} \, x{=}0, \mathsf{R}_{t_2} \, x{=}0, \mathsf{P}_{t_2} \, 1, \mathsf{W}_{t_1} \, x{=}1 \end{split}$$

Т

$$\begin{split} & \mathsf{W}_{t_1} \, x{=}1, \mathsf{R}_{t_2} \, x{=}1, \mathsf{P}_{t_2} \, 1 \\ & \mathsf{R}_{t_2} \, x{=}0, \mathsf{W}_{t_1} \, x{=}1, \mathsf{P}_{t_2} \, 1 \\ & \mathsf{R}_{t_2} \, x{=}0, \mathsf{P}_{t_2} \, 1, \mathsf{W}_{t_1} \, x{=}1 \end{split}$$

$$P_1 = \mathbf{*x} = 1 \begin{vmatrix} \mathbf{r1} = \mathbf{*x}; & \mathbf{r2} = \mathbf{*x}; \\ \text{if } \mathbf{r1} = \mathbf{r2} \text{ then print 1 else print 2} \\ P_2 = \mathbf{*x} = 1 \begin{vmatrix} \mathbf{r1} = \mathbf{*x}; & \mathbf{r2} = \mathbf{r1}; \\ \text{if } \mathbf{r1} = \mathbf{r2} \text{ then print 1 else print 2} \end{vmatrix}$$

Executions of P1:

Executions of P2:

$$\begin{split} & \mathsf{W}_{t_1} \, x{=}1, \mathsf{R}_{t_2} \, x{=}1, \mathsf{R}_{t_2} \, x{=}1, \mathsf{P}_{t_2} \, 1 \\ & \mathsf{R}_{t_2} \, x{=}0, \mathsf{W}_{t_1} \, x{=}1, \mathsf{R}_{t_2} \, x{=}1, \mathsf{P}_{t_2} \, 2 \\ & \mathsf{R}_{t_2} \, x{=}0, \mathsf{R}_{t_2} \, x{=}0, \mathsf{W}_{t_1} \, x{=}1, \mathsf{P}_{t_2} \, 1 \\ & \mathsf{R}_{t_2} \, x{=}0, \mathsf{R}_{t_2} \, x{=}0, \mathsf{P}_{t_2} \, 1, \mathsf{W}_{t_1} \, x{=}1 \end{split}$$

Т

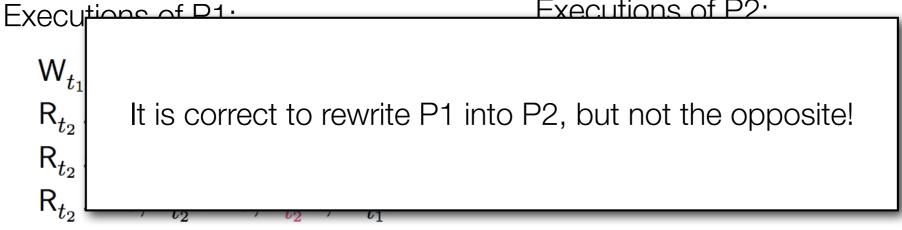
$$\begin{split} & \mathsf{W}_{t_1} \, x{=}1, \mathsf{R}_{t_2} \, x{=}1, \mathsf{P}_{t_2} \, 1 \\ & \mathsf{R}_{t_2} \, x{=}0, \mathsf{W}_{t_1} \, x{=}1, \mathsf{P}_{t_2} \, 1 \\ & \mathsf{R}_{t_2} \, x{=}0, \mathsf{P}_{t_2} \, 1, \mathsf{W}_{t_1} \, x{=}1 \end{split}$$

Behaviours of P1: $[P_{t_2} 1], [P_{t_2} 2]$

Behaviours of P2: $[P_{t_2} 1]$

$$P_{1} = *x = 1 \begin{vmatrix} r1 = *x; r2 = *x; \\ if r1 = r2 \text{ then print 1 else print 2} \end{vmatrix}$$

$$P_{2} = *x = 1 \begin{vmatrix} r1 = *x; r2 = r1; \\ if r1 = r2 \text{ then print 1 else print 2} \end{vmatrix}$$
Executions of P1:



Behaviours of P1: $[P_{t_2} 1], [P_{t_2} 2]$

Behaviours of P2: $[P_{t_2} 1]$

There is only one execution with a printing behaviour:

$$\mathsf{W}_{t_1} x = 1, \mathsf{W}_{t_1} y = 1, \mathsf{R}_{t_2} x = 1, \mathsf{W}_{t_2} x = 2, \mathsf{W}_{t_2} y = 2, \mathsf{R}_{t_1} y = 2, \mathsf{R}_{t_1} x = 2, \mathsf{P}_{t_1} 2 = 2, \mathsf{P}_{t_1}$$

But a compiler would optimise to:

The only execution with a printing behaviour in the optimised code is:

$$\mathsf{W}_{t_1} \, x{=}1, \mathsf{W}_{t_1} \, y{=}1, \mathsf{R}_{t_2} \, x{=}1, \mathsf{W}_{t_2} \, x{=}2, \mathsf{W}_{t_2} \, y{=}2, \mathsf{R}_{t_1} \, y{=}2, \mathsf{P}_{t_1} \, 1$$

So the optimisation is not correct.

*x = 1;	r = *x;	
*y = 1;	print r	

Our first example highlighted that CSE is incorrect in SC.

Here is another example.

 $\begin{bmatrix} \mathsf{P}_{t_2} \, \mathsf{1}, \, \mathsf{P}_{t_2} \, \mathsf{0}, \, \mathsf{P}_{t_2} \, \mathsf{1} \end{bmatrix}$ $\begin{bmatrix} \mathsf{P}_{t_2} \, \mathsf{0}, \, \mathsf{P}_{t_2} \, \mathsf{1}, \, \mathsf{P}_{t_2} \, \mathsf{1} \end{bmatrix}$ $\begin{bmatrix} \mathsf{P}_{t_2} \, \mathsf{0}, \, \mathsf{P}_{t_2} \, \mathsf{0}, \, \mathsf{P}_{t_2} \, \mathsf{1} \end{bmatrix}$ $\begin{bmatrix} \mathsf{P}_{t_2} \, \mathsf{0}, \, \mathsf{P}_{t_2} \, \mathsf{0}, \, \mathsf{P}_{t_2} \, \mathsf{0} \end{bmatrix}$

The observable behaviours are (note that 0 - 1 - 0 is not observable):

$$\begin{split} & [\mathsf{P}_{t_2} \, 1, \mathsf{P}_{t_2} \, 1, \mathsf{P}_{t_2} \, 1] \\ & [\mathsf{P}_{t_2} \, 1, \mathsf{P}_{t_2} \, 0, \mathsf{P}_{t_2} \, 1] \\ & [\mathsf{P}_{t_2} \, 0, \mathsf{P}_{t_2} \, 1, \mathsf{P}_{t_2} \, 1] \\ & [\mathsf{P}_{t_2} \, 0, \mathsf{P}_{t_2} \, 0, \mathsf{P}_{t_2} \, 1] \\ & [\mathsf{P}_{t_2} \, 0, \mathsf{P}_{t_2} \, 0, \mathsf{P}_{t_2} \, 0] \end{split}$$

But a compiler would optimise as:

*x = 1;	r = *x;	*x = 1;	r = *x;
*y = 1;	print r;	*y = 1;	print r;
	<pre>print *y;</pre>		<pre>print *y;</pre>
	<pre>print *x;</pre>		<pre>print r;</pre>

Let's compare the behaviours of the two programs:

$$\begin{bmatrix} \mathsf{P}_{t_2} \ 1, \, \mathsf{P}_{t_2} \ 1, \, \mathsf{P}_{t_2} \ 1, \, \mathsf{P}_{t_2} \ 1, \\ \begin{bmatrix} \mathsf{P}_{t_2} \ 1, \, \mathsf{P}_{t_2} \ 0, \, \mathsf{P}_{t_2} \ 1 \end{bmatrix} \\ \begin{bmatrix} \mathsf{P}_{t_2} \ 0, \, \mathsf{P}_{t_2} \ 1, \, \mathsf{P}_{t_2} \ 1 \end{bmatrix} \\ \begin{bmatrix} \mathsf{P}_{t_2} \ 0, \, \mathsf{P}_{t_2} \ 0, \, \mathsf{P}_{t_2} \ 1 \end{bmatrix} \\ \begin{bmatrix} \mathsf{P}_{t_2} \ 0, \, \mathsf{P}_{t_2} \ 0, \, \mathsf{P}_{t_2} \ 0 \end{bmatrix}$$

$$\begin{split} & [\mathsf{P}_{t_2} \, \mathsf{l}, \mathsf{P}_{t_2} \, \mathsf{l}, \mathsf{P}_{t_2} \, \mathsf{l}] \\ & [\mathsf{P}_{t_2} \, \mathsf{l}, \mathsf{P}_{t_2} \, \mathsf{0}, \mathsf{P}_{t_2} \, \mathsf{1}] \\ & [\mathsf{P}_{t_2} \, \mathsf{0}, \mathsf{P}_{t_2} \, \mathsf{1}, \mathsf{P}_{t_2} \, \mathsf{0}] \\ & [\mathsf{P}_{t_2} \, \mathsf{0}, \mathsf{P}_{t_2} \, \mathsf{0}, \mathsf{P}_{t_2} \, \mathsf{0}] \end{split}$$

$$\begin{bmatrix} \mathsf{P}_{t_2} \ 1, \, \mathsf{P}_{t_2} \ 1, \, \mathsf{P}_{t_2} \ 1, \\ [\mathsf{P}_{t_2} \ 1, \, \mathsf{P}_{t_2} \ 0, \, \mathsf{P}_{t_2} \ 1] \\ \begin{bmatrix} \mathsf{P}_{t_2} \ 0, \, \mathsf{P}_{t_2} \ 1, \, \mathsf{P}_{t_2} \ 1] \\ [\mathsf{P}_{t_2} \ 0, \, \mathsf{P}_{t_2} \ 0, \, \mathsf{P}_{t_2} \ 1] \\ \begin{bmatrix} \mathsf{P}_{t_2} \ 0, \, \mathsf{P}_{t_2} \ 0, \, \mathsf{P}_{t_2} \ 0] \end{bmatrix}$$

$$\begin{bmatrix} \mathsf{P}_{t_2} \ 1, \mathsf{P}_{t_2} \ 1, \mathsf{P}_{t_2} \ 1 \end{bmatrix}$$
$$\begin{bmatrix} \mathsf{P}_{t_2} \ 1, \mathsf{P}_{t_2} \ 0, \mathsf{P}_{t_2} \ 1 \end{bmatrix}$$
$$\begin{bmatrix} \mathsf{P}_{t_2} \ 0, \mathsf{P}_{t_2} \ 1, \mathsf{P}_{t_2} \ 0 \end{bmatrix}$$
$$\begin{bmatrix} \mathsf{P}_{t_2} \ 0, \mathsf{P}_{t_2} \ 1, \mathsf{P}_{t_2} \ 0 \end{bmatrix}$$

Reordering incorrect

$$*x = 1;$$
 $*y = 1;$ $r1 = *y$ $*y = 1;$ $r1 = *y$ $r2 = *x;$ $\Rightarrow *x = 1;$ $r2 = *x;$ print r1print r2print r1print r2

Again, the optimised program exhibits a new behaviour:

$$\begin{split} & [\mathsf{P}_{t_1} \, 0, \mathsf{P}_{t_2} \, 1] \\ & [\mathsf{P}_{t_1} \, 1, \mathsf{P}_{t_2} \, 0] \\ & [\mathsf{P}_{t_1} \, 1, \mathsf{P}_{t_2} \, 1] \end{split}$$

$$\begin{split} & [\mathsf{P}_{t_1} \, 0, \mathsf{P}_{t_2} \, 1] \\ & [\mathsf{P}_{t_1} \, 1, \mathsf{P}_{t_2} \, 0] \\ & [\mathsf{P}_{t_1} \, 1, \mathsf{P}_{t_2} \, 1] \\ & [\mathsf{P}_{t_1} \, 0, \mathsf{P}_{t_2} \, 0] \end{split}$$

Elimination of adjacent accesses

There are some correct optimisations under SC. For example it is correct to rewrite:

 $r1 = *x; r2 = *x \rightarrow r1 = *x; r2 = r1$

The basic idea: whenever we perform the read r1 = *x in the optimised program, we perform *both* reads in the source program.

(More on this later)

Elimination of adjacent accesses

There are some correct optimisations under SC. For example it is correct to rewrite:

 $r1 = *x; r2 = *x \rightarrow r1 = *x; r2 = r1$

Can we define a model that:

1) enables more optimisations than SC, and

2) retains the simplicity of SC?

(IVIORE OF LITIS TALE)

The layman solution forbid data-races



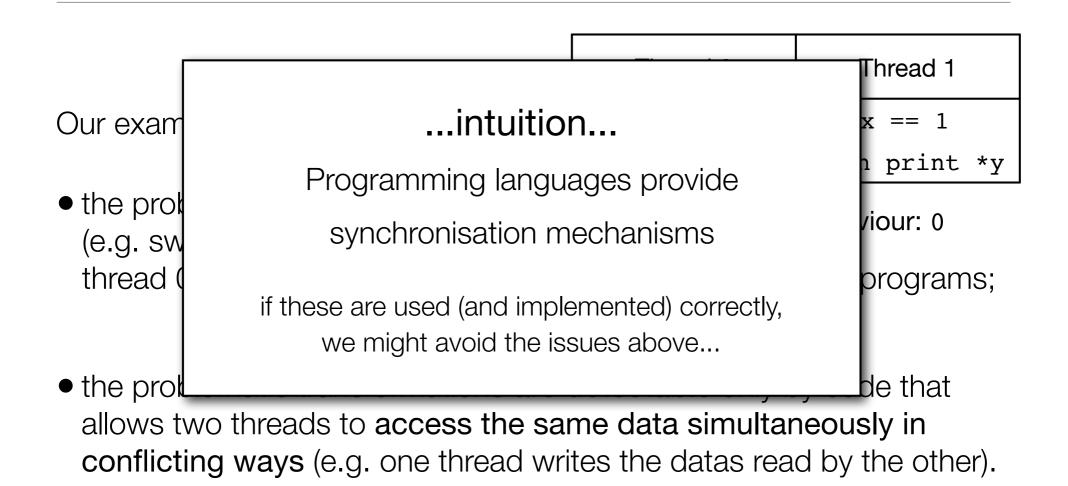
Data-race freedom

Our examples again:

Thread 0	Thread 1
*y = 1	if *x == 1
*x = 1	then print *y

- the problematic transformations (e.g. swapping the two writes in thread 0) do not change the meaning of single-threaded programs;
- the problematic transformations are detectable only by code that allows two threads to access the same data simultaneously in conflicting ways (e.g. one thread writes the datas read by the other).

Data-race freedom



The basic solution	Thread 0	Thread 1
	*y = 1	if *x == 1
Prohibit data races	*x = 1	then print *y

Observable behaviour: 0

Defined as follows:

- two memory operations **conflict** if they access the same memory location and at least one is a store operation;
- a SC execution (interleaving) contains a data race if two conflicting operations corresponding to different threads are adjacent (maybe executed concurrently).

Example: a data race in the example above:

$$\mathsf{W}_{t_1} y{=}1, \mathsf{W}_{t_1} x{=}1, \mathsf{R}_{t_2} x{=}1, \mathsf{R}_{t_2} y{=}1, \mathsf{P}_{t_2} 1$$

The basic sc	olution	Thread 0	Thread 1
Prohibit d	ata races	*y = 1	if $*x == 1$
		*x = 1	then print *y
		Observable	behaviour: 0
Defined as follows:			
 two men location Th 			
operatio	over the sequential consistent executions iflicting t (maybe		

executed concurrently).

Example: a data race in the example above:

$$W_{t_1} y=1, W_{t_1} x=1, R_{t_2} x=1, R_{t_2} y=1, P_{t_2} 1$$

How do we avoid data races? (focus on high-level languages)

• Locks

No lock(I) can appear in the interleaving unless prior lock(I) and unlock(I) calls from other threads balance.

• Atomic variables

Allow concurrent access "exempt" from data races. Called volatile in Java.

Thread 0	Thread 1
<pre>*y = 1 lock();</pre>	<pre>lock(); tmp = *x;</pre>
<pre>*x = 1 unlock();</pre>	unlock(); if tmp = 1 then print *y

Example:

How do we avoid data races? (focus on high-level languages)

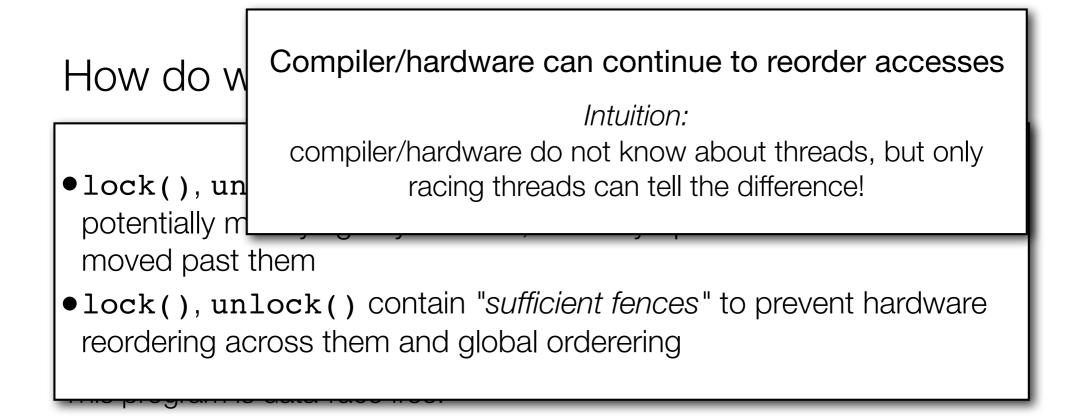
Thread 0	Thread 1
lock();	<pre>lock(); tmp = *x;</pre>
<pre>*x = 1 unlock();</pre>	unlock(); if tmp = 1 then print *y

This program is data-race free:

*y = 1; lock();*x = 1;unlock(); lock();tmp = *x;unlock(); if tmp=1 then print *y *y = 1; lock(); tmp = *x; unlock(); lock(); *x = 1; unlock(); if tmp=1 *y = 1; lock(); tmp = *x; unlock(); if tmp=1; lock(); *x = 1; unlock(); lock();tmp = *x;unlock(); *y = 1; lock(); *x = 1; unlock(); if tmp=1 lock(); tmp = *x; unlock(); if tmp=1; *y = 1; lock();*x = 1;unlock(); lock();tmp = *x;unlock(); *y = 1; if tmp=1; lock(); *x = 1; unlock();

How do we avoid data races? (focus on high-level languages)

- lock(), unlock() are opaque for the compiler: viewed as potentially modifying any location, memory operations cannot be moved past them
- •lock(), unlock() contain "sufficient fences" to prevent hardware reordering across them and global orderering



*y = 1; lock(); tmp = *x; unlock(); lock(); *x = 1; unlock(); if tmp=1

lock();tmp = *x;unlock(); *y = 1; lock(); *x = 1; unlock(); if tmp=1

lock();tmp = *x;unlock(); *y = 1; if tmp=1; lock(); *x = 1; unlock();

Another example of DRF program

Exercise: is this program DRF?

Thread 0	Thread 1
if *x == 1	if *y == 1
then $*y = 1$	then *x = 1

Another example of DRF program

Exercise: is this program DRF?

Thread 0	Thread 1
if *x == 1	if *y == 1
then *y = 1	then *x = 1

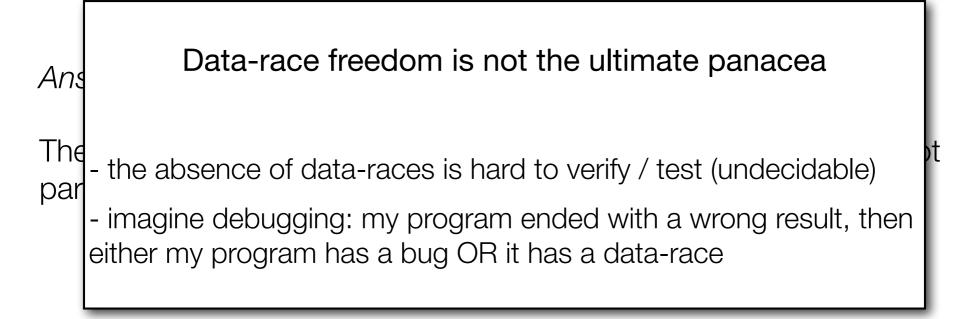
Answer: yes!

The writes cannot be executed in any SC execution, so they cannot participate in a data race.

Another example of DRF program

Exercise: is this program DRF?

Thread 0	Thread 1
if *x == 1	if *y == 1
then *y = 1	then $*x = 1$



Validity of compiler optimisations, summary

Transformation	SC	DRF
Memory trace preserving transformations	\checkmark	\checkmark
Redundant read after read elimination	✓*	\checkmark
Redundant read after write elimination	✓*	\checkmark
Irrelevant read elimination	\checkmark	\checkmark
Redundant write before write elimination	✓*	\checkmark
Redundant write after read elimination	✓*	\checkmark
Irrelevant read introduction	\checkmark	×
Normal memory accesses reordering	×	\checkmark
Roach-motel reordering	\times (\checkmark for locks)	\checkmark
External action reordering	×	\checkmark

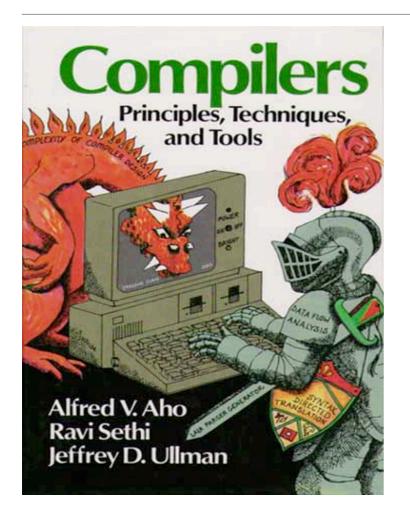
* Optimisations legal only on adjacent statements.

Validity of compiler optimisations, summary

Transformation	SC	
Memory trace preserving transformations	\checkmark	
 slav Sevcik Optimisations for Shared-Memory Cond	current Prograi	ms
	P	LDI 2011
Roach-motel reordering	$\times (\sqrt{\text{for locks}})$	\checkmark

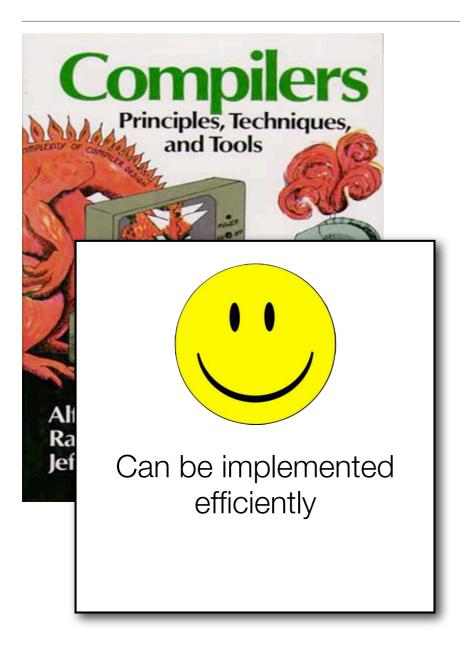
* Optimisations legal only on adjacent statements.

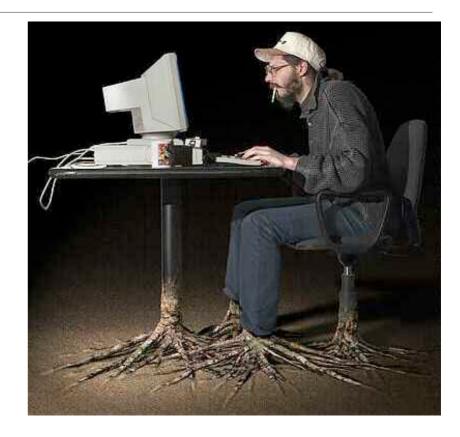
Compilers, programmers & data-race freedom



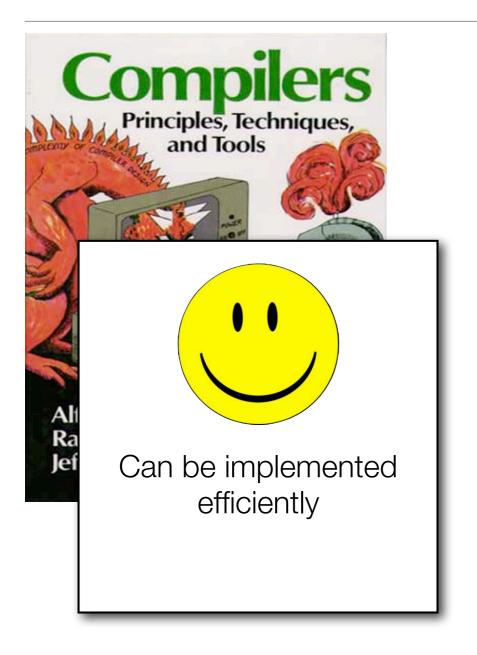


Compilers, programmers & data-race freedom

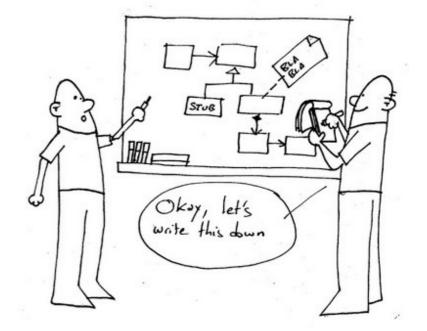




Compilers, programmers & data-race freedom







Data-race freedom, formalisation

A toy language: semantics

location, x register, r integer, n thread_id, t	shared memory location thread-local variable integers thread identifier
<pre>statement, s ::= r := x x := r r := n lock unlock print r</pre>	statements read from memory write to memory load constant into register lock unlock output
program, p ::= s;	; s a program is a sequence of statements
system ::= cor	ncurrent system
$ t_0:p_0 t_n:$	p n parallel composition of n threads

A toy language: semantics

location, x register, r in th:	shared memory location thread-local variable	
We work with a toy language, but this approach scales to the full Java Memory Model or C11/C++11.		
lock	lock	
unlock	unlock	
print r	output	
program, p ::= s;	a program is a sequence of statements	
system ::= co	ncurrent system	

Definition [trace]: a sequence of memory operations (read, write, thread start, I/O, synchronisation). Thread start is always the first action of thread. All actions in a trace belong to the same thread.

Definition [traceset]: a traceset is a prefix-closed set of traces.

Sample traceset:

Thread 0	Thread 1
r1:=x	r2:=y
	x:=1
y:=r1	print r2

$$\{ [S(0), R[x=v], W[y=v]] \mid v \in V \} \\ \cup \{ [S(1), R[y=v], W[x=1], X(v)] \mid v \in V \} \}$$

Remarks:

1. Reads can read arbitrary values from memory.

- 2. Tracesets should not be confused with interleavings.
- sta 3. Tracesets do not enforce receptiveness or determinism: $\{[S(0)], [S(0), R[x=1]], [S(0), W[y=1]]\}$

is also a valid traceset for the example below.

Sample traceset:

Tr

De

thr

De

Thread 0	Thread 1
r1:=x y:=r1	r2:=y
	x:=1
	print r2

$$\{ [S(0), R[x=v], W[y=v]] \mid v \in V \} \\ \cup \{ [S(1), R[y=v], W[x=1], X(v)] \mid v \in V \} \}$$

ad

Associate tracesets to toy language programs

$$\langle S, r := x; s \rangle \xrightarrow{R[x=v]} \langle S[r=v], s \rangle$$

$$\langle S, x := r; s \rangle \xrightarrow{W[x=S(r)]} \langle S, s \rangle$$

$$\langle S, r := n; s \rangle \xrightarrow{T} \langle S[r=n], s \rangle$$

$$\langle S, lock; s \rangle \xrightarrow{L} \langle S, s \rangle$$

$$\langle S, unlock; s \rangle \xrightarrow{U} \langle S, s \rangle$$

$$\langle S, print r; s \rangle \xrightarrow{X(S(r))} \langle S, s \rangle$$

$$\langle S, t_0:p_0 \mid ... \mid t_n:p_n \rangle \xrightarrow{S(i)} \langle S, p_i \rangle$$

Tracesets and interleavings

Definition [interleaving]: an interleaving is a sequence of thread-identifieraction pairs.

 $I' = \left[\left< 0, \mathbf{S}(0) \right>, \left< 1, \mathbf{S}(1) \right>, \left< 0, \mathbf{W}[\mathbf{y}=1] \right>, \left< 1, \mathbf{R}[\mathbf{v}=0] \right>, \left< 1, \mathbf{X}(0) \right> \right]$

Given an interleaving *I*, the trace of *tid* in *I* is the sequence of actions of thread *tid* in *I*, e.g.:

trace 1 I' = [S(1), R[v=0], X(0)].

Conversely, given a traceset, we can compute all the well-formed interleavings (called *executions*)... (next slide)

Tracesets and interleavings

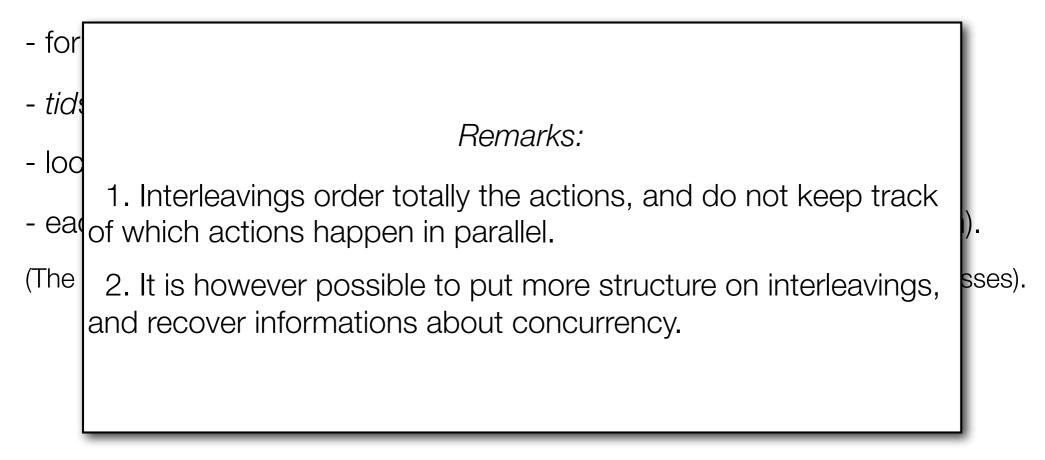
An interleaving *I* is an *execution* of a traceset *T* if:

- for all *tid*, trace *tid* $I \in T$ (traces belong to the traceset)
- *tids* correspond to entry points S(*tid*)
- lock / unlock alternates correctly
- each read sees the most recent write to the same location (read/from).

(The last property enforce the sequentially consistent semantics for memory accesses).

Tracesets and interleavings

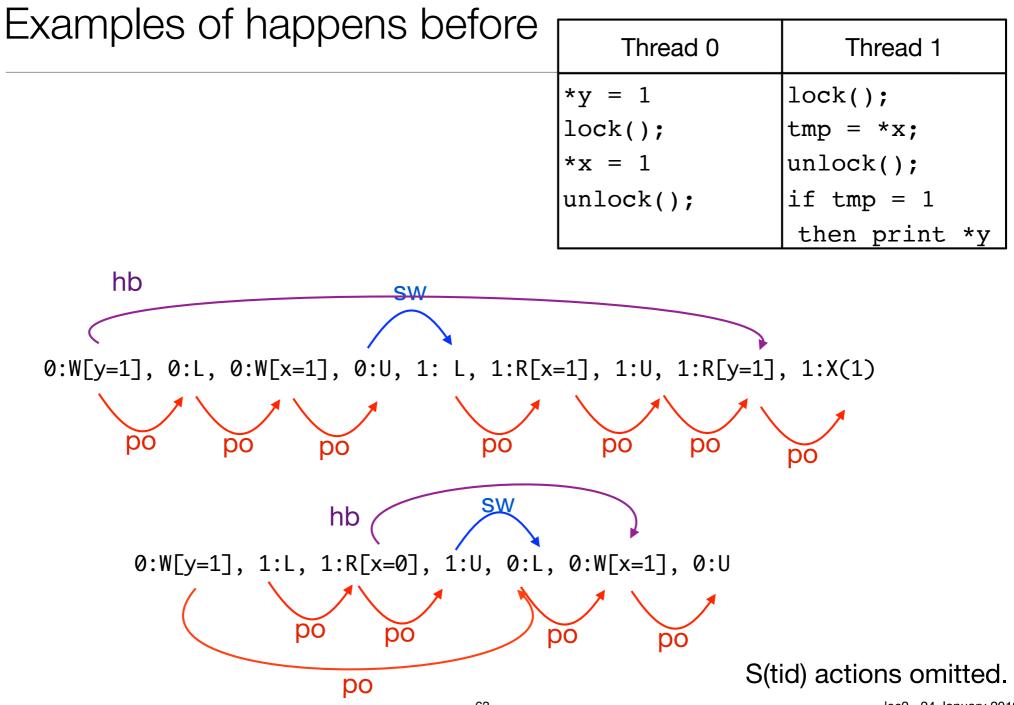
An interleaving *I* is an *execution* of a traceset *T* if:



Definition [program order]: program order, <ppo, is a total order over the actions of the same thread in an interleaving.

Definition [synchronises with]: in an interleaving I, index i synchroniseswith index j, i $<_{sw}$ j, if i < j and A(I_i) = U (unlock), A(I_j) = L (lock).

Definition [happens-before]: Happens-before is the transitive closure of program order and synchronises with.

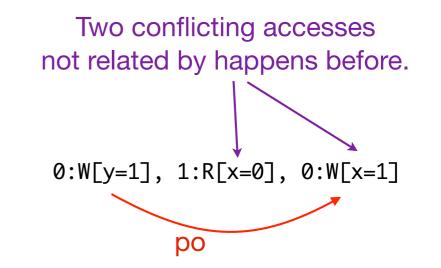


Definition [data-race-freedom]: A traceset is **data-race free** if none of its executions has two adjacent conflicting actions from different threads.

Equivalently, a traceset is data-race free if in all its executions all pairs of conflicting actions are ordered by happens-before.

Thread 0Thread 1*y = 1if *x == 1*x = 1then print *y

A racy program



Data-race freedom: equivalence of definitions

Given an execution

$$\alpha$$
 ++ [a] ++ β ++ [b]

of a traceset *T* where [a] and [b] are the first conflicting actions not related by happen-before, we build the interleaving

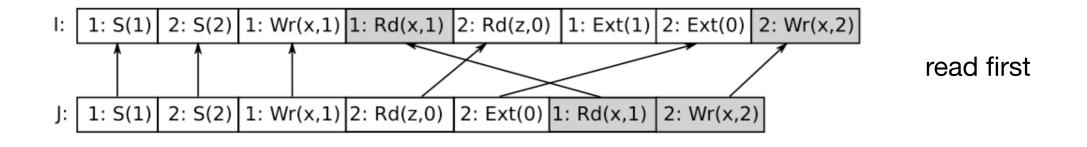
α ++ β' ++ [a] ++ [b]

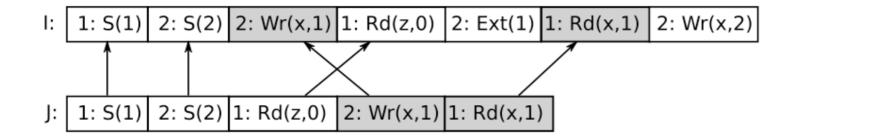
where β ' are all the actions from β that strictly happen-before [b].

It remains to show that $\alpha ++ \beta' ++ [a] ++ [b]$ is an execution of *T*.

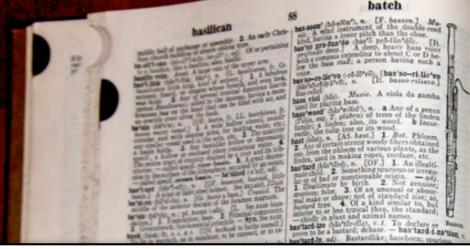
The formal proof is tedious and not easy (see Boyland 2008, Bohem & Adve 2008, Sevcik), here will give the intuitions of the construction on an example.

Data-race freedom: equivalence of definitions





write first



nice lie've (-rê-li've), bas'so-ri-lie've has so-re-lie've (-rê-li've), n. []t. basso-riliere.]

s'tard-ly, adj. Bastardlike; baschorn; an

bate bate (blit), e. L. (From ABATE.) To lesses by retrenching, deducting, or reducing; abate; hence, to lower, moderate, etc.; as, to bate one's breath. - v. s. To waste away. Shak.

and the set of the set o air, sand, or oil, for regulating the temperature of anything

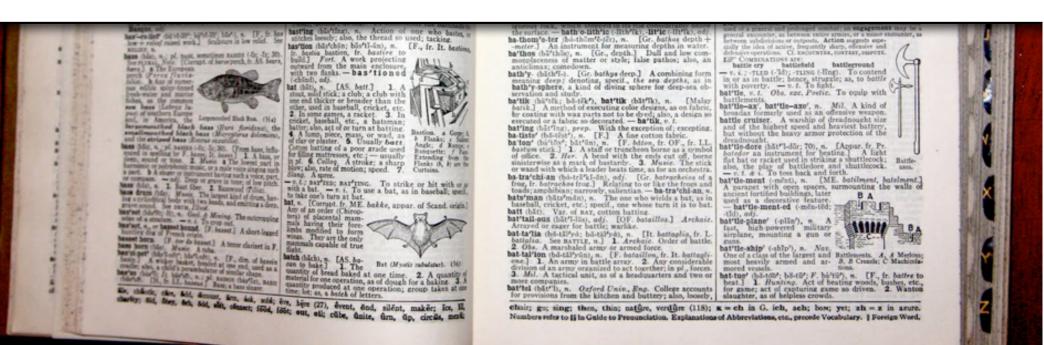
the whole college accounts: — only in pl., except adjective-by. — e. 4. To have such an account. — bat'tel-er. n. bat'ten (bla'n, e. 4. [ON. bat'se to grow better.] To thrive; grow lat; also, to grow further mail. — e. 1. To make fat: faiten. bat'ten n. [F. baten stick, staff.] 1. A strip of sawed timber, used for flooring, etc. 2. A strip of wood used for mailing acrois two other pieces, to cover a crack, stiffen a spar, etc. — e. f. To furnish of fasten with battens; as, to bat'ten a base; batten down the hat thes. — bat'tener; n. bat'ter (bla'fir), s. f. [OF. batrs, battes. The Enz. word is prob. in part freq. from bat to strike.] 1. To beat with successive blows; beat so as to bruise, shatter, or demeliab. 2. To wear or impair as by hard unage. — s. i. To beat

successive blows: beat so as to bruine, shatter, or demolish.
2. To wear or impair as by hard usare. — e. i. To beat repeatedly, eso. with violence. — m. 1. A semiliquid mixture, as for cake or biscuit, of four, liquid, etc. 2. Print. A bruise on the face of a plate or of type in the form; also, the faces or type so injured.

bat'ter y (bis'fel), n.; pl. -rnz-izs (·Iz), (F. batterie, ir. batter to beat.] 1. Act of battering or beating. 2. Apparatus used in battering. 3. A number of

1662 - 24 January 2019

Defining programming language memory models



Option 1

Don't. No concurrency.

Implemented by highly-successful programming languages (OCaml)

Poor match for current trends

Option 2

Don't. No shared memory

A good match for some problems (see Erlang, MPI, ...)



Don't.

But language ensures data-race freedom

Possible:

- syntactically ensuring data accesses protected by associated locks
- fancy effect type systems

Not suitable for general purpose programming.



Don't. Leave it (sort of) up to the hardware

Example:

MLton, a high performance ML-to-x86 compiler with concurrency extensions

Accesses to ML refs exhibit the underlying x86-TSO behaviour (atomicity is guaranteed though)



Do. Use data race freedom as a definition

- 1. Programs that race-free have only sequentially consistent behaviours
- 2. Programs that have a race in some execution can behave in any way

Sarita Adve & Mark Hill, 1990





Do.

Use data race freedom as a definition

Pro:

- simple

- strong guarantees for most code
- allows lots of freedom for compiler and hardware optimisations

Cons:

- undecidable premise
- can't write racy programs (escape mechanisms?)

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.

Data-races are errors

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.

Data-races are errors

C++ 2011

[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.

Data-races are errors

Data race freedom as a definition

• Core of the C11/C++11 standard.

Hans Boehm & Sarita Adve, PLDI 2008.



• Part of the JSR-133 standard.

Jeremy Manson & Bill Pugh & Sarita Adve, PLDI 2008.



Data race freedom as a definition

• Core of the C11/C++11 standard.

Hans Boehm & Sarita Adve, PLDI 2008.

with some escape mechanism called "low level atomics".

Mark Batty & al., POPL 2011.

• Part of the JSR-133 standard.

Jeremy Manson & Bill Pugh & Sarita Adve, PLDI 2008.

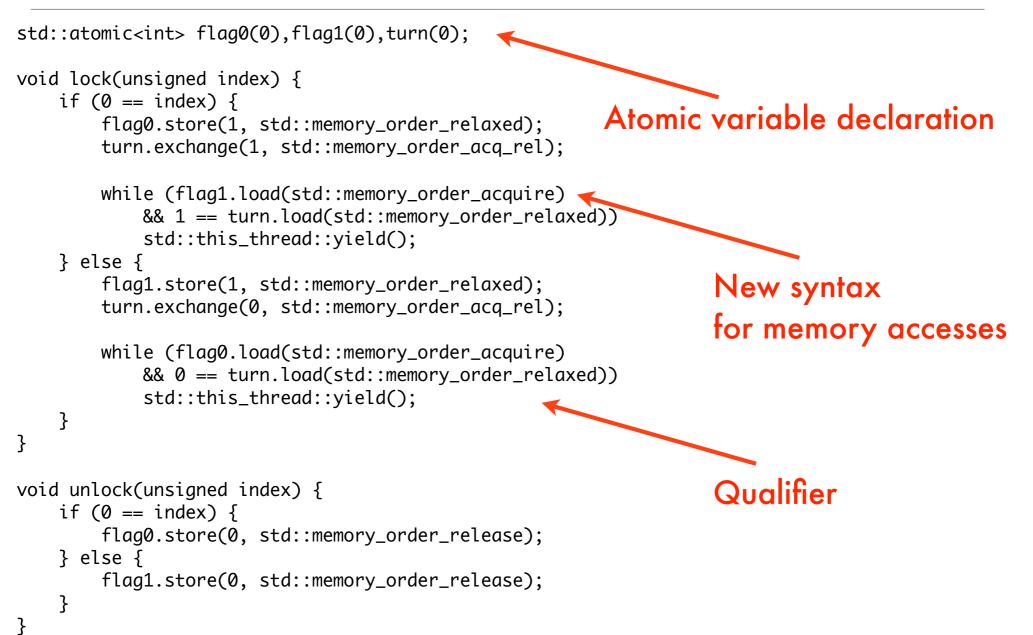
DRF gives no guarantees for untrusted code: a disaster for Java, which relies on unforgeable pointers for its security guarantees.

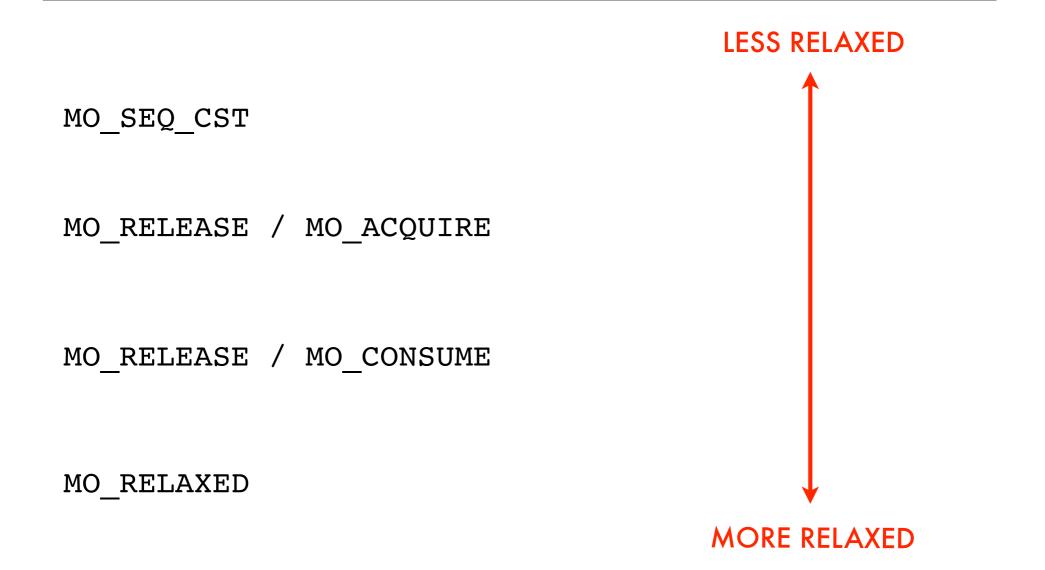
JSR-133 is DRF + some out-of-thin-air guarantees for all code.

Escape lanes for expert programmers

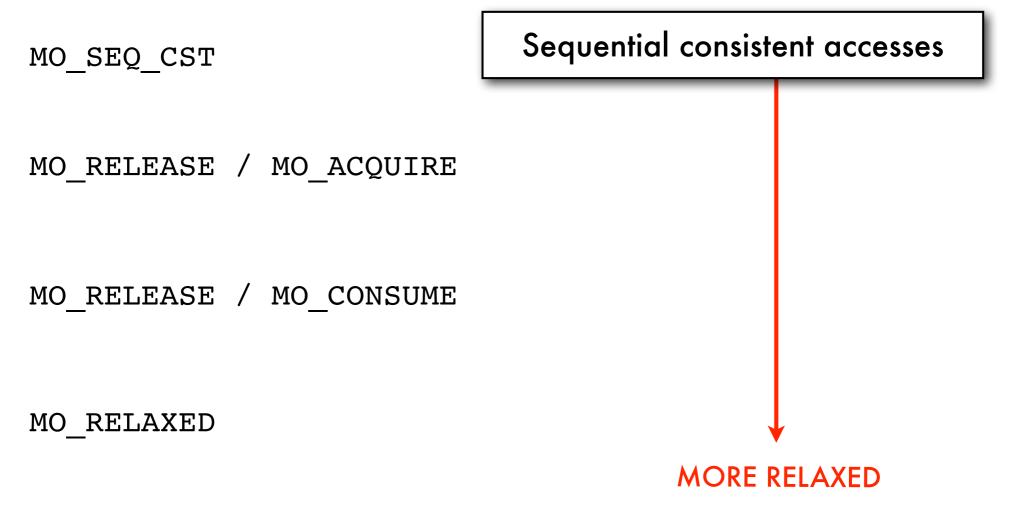


Low-level atomics in C11/C++11

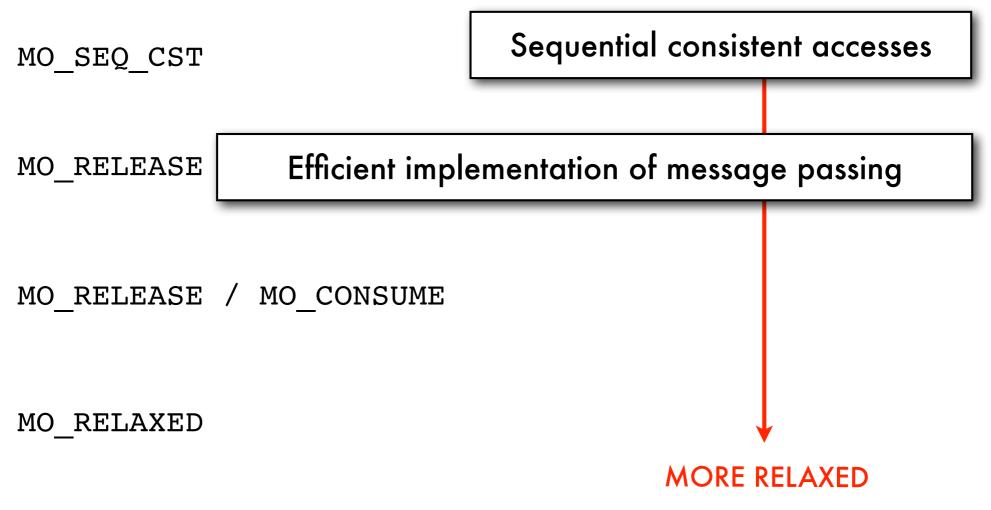




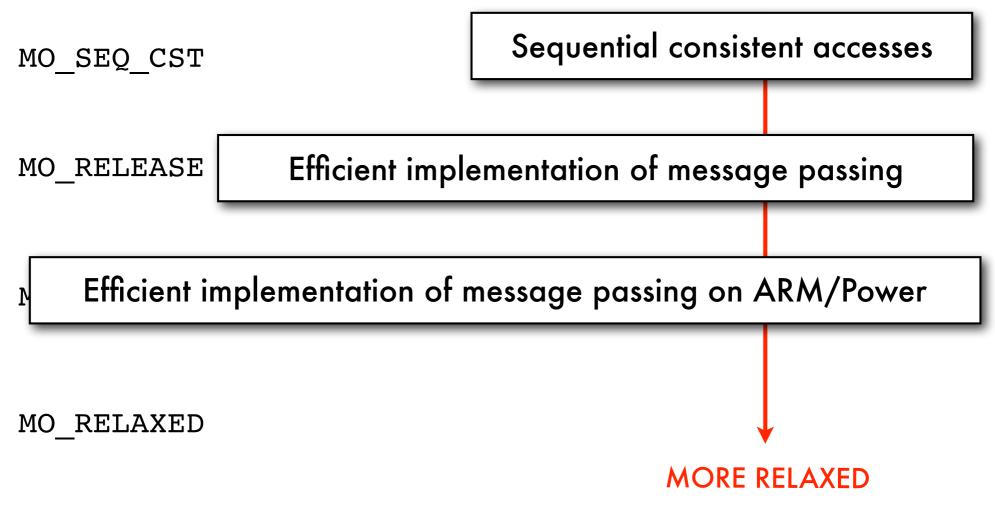








LESS RELAXED



LESS RELAXED

Sequential consistent accesses MO SEQ CST MO RELEASE Efficient implementation of message passing Efficient implementation of message passing on ARM/Power No synchronisation; direct access to hardware MO RELAX **MORE RELAXED**



The compiler must ensure that MO_SEQ_CST accesses have sequentially consistent semantics.

Thread 0	Thread 1
x.store(1,MO_SEQ_CST)	y.store(1,MO_SEQ_CST)
$r1 = y.load(MO_SEQ_CST)$	$r2 = x.load(MO_SEQ_CST)$

The program above cannot end with r1 = r2 = 0.

Sample compilation on x86:Sample compilation on Power:store: MOV; MFENCEstore: HWSYNC; STload: MOVload: HWSYNC; LD; CMP; BC; ISYNC

MO_RELEASE / MO_ACQUIRE

Supports a fast implementation of the message passing idiom:

Thread 0	Thread 1
x.store(1,MO_RELAXED) 7	r1 = y.load(MO_ACQUIRE)
y.store(1,MO_RELEASE)	$r2 = x.load(MO_RELAXED)$

The program above cannot end with r1 = 1 and r2 = 0.

Accesses to the data structure can be reordered/optimised (MO_RELAXED).

Sample compilation on x86:Sample compilation on Power:store: MOVstore: LWSYNC; STload: MOVload: LD; CMP; BC; ISYNC

MO_RELEASE / MO_CONSUME

Supports a fast implementation of the message passing idiom on Power:

Thread 0	Thread 1
x.store(1,MO_RELAXED)	<pre>r1 = y.load(x,MO_CONSUME)</pre>
y.store(&x,MO_RELEASE)	$r2 = (*x).load(MO_RELAXED)$

The program above cannot end with r1 = 1 and r2 = 0.

The two loads have an address dependency, Power won't reorder them.

Sample compilation on x86:Sample compilation on Power:store: MOVstore: LWSYNC; STload: MOVload: LD

Memory access synchronisation

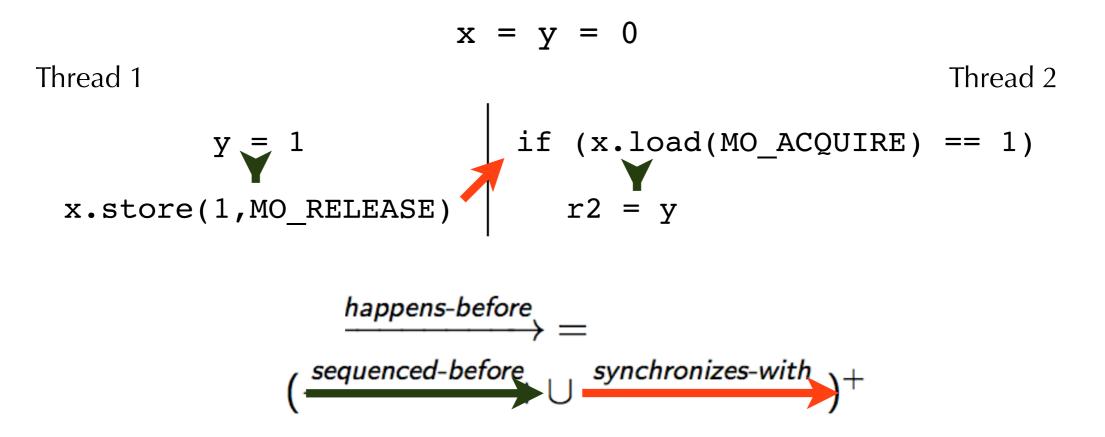
$$\mathbf{x} = \mathbf{y} = \mathbf{0}$$

Thread 1

$$\mathbf{y} = \mathbf{1}$$

x.store(1,MO_RELEASE)

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

$$x = y = 0$$

Thread 1

Understanding MO_RELAXED

Thread 1

$$x = y = 0$$

Т

Thread 2

DATA RACE

Two conflicting accesses not related by happens-before

Understanding MO_RELAXED

$$x = y = 0$$

Thread 1

Thread 2

y.store(1,MO_RELAXED) i

x.store(1,MO_RELAXED)

WELL DEFINED

but $r^2 = 0$ is possible

Intuition

the compiler (or hardware) can reorder independent accesses

$$x = y = 0$$

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

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

Thread 1

Thread 2

y.store(1,MO_RELAXED)	<pre>if (x.load(MO_RELAXED) == 1)</pre>
<pre>x.store(1,MO_RELAXED)</pre>	$r2 = y.load(MO_RELAXED)$

н

Allow a RELAXED load to see any store that:

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

I he full model

$x \xrightarrow{i} b = (x, b) \in r$			
	isotore $a=case\ a$ of STORE \rightarrow T $\ $. \rightarrow F		vikle-ald-affer_art zeiten throte keztek kird segmend lader additional approximated all data dependency tappana-kefer =
$a \ r \ b = (a, b) \in r$	is fince a = case a of $\operatorname{Fexce}_{} \to T \parallel_{-} \to F$	ncchement <i>mchand</i> a = somecheme a somecheme a	vihila david series et savien t translu konisien inde superved solver additional-genetonisme with data dependency toppens-before = (vihila-sub-supervised (k + 1) = n in in intervised (k + 1) = n in intervised (k + 1) = n intervi
	is dock or much $s = \mathrm{is}_{s}\mathrm{lock}\ s \vee \mathrm{is}_{s}\mathrm{mlock}\ s$		
$x \stackrel{\sim}{\rightarrow} b = (x, b) \notin r$	ic_stomic_action a = ic_stomic_load a V is_atomic_store a V ic_atomic_ymw a	$\label{eq:alpha} \begin{split} &holds = \operatorname{scalar}_{A} (a) &holds = \operatorname{scalar}_{A} (a) \\ &holds = \operatorname{scalar}_{A} (A)$	with expressed of determined in which expressed of the determination of
		$(b \sim a)^{\vee}$ $(c \sim a)^{-1}$ $(c \sim a)^{-1}$ $a \sim a)^{-1}$ a	$\gamma = \frac{1}{2} \sum_{i=1}^{n-1} \frac{1}{i} \sum_{i=1}^{n-1} \sum_{j=1}^{n-1} \frac{1}{i} \sum_{i=1}^{n-1} \sum_{j=1}^{n-1} \frac{1}{i} \sum_{i=1}^{n-1} \frac{1}{i} \sum_{j=1}^{n-1} \frac{1}{i} \sum_$
$a \xrightarrow{i}{\rightarrow} b \xrightarrow{h}{\rightarrow} c = a \xrightarrow{i}{\rightarrow} b \wedge b \xrightarrow{h}{\rightarrow} c$	is load or store $a = i$ load $a \vee i$ s-tore a	(vs. sai and c)))	$ \min_{x \in [x_1, x_2] \in \mathcal{X}_{x_1}(x_2 \in \mathcal{X}_{x_2}(x_2 \in \mathcal{X}_{x_2})] } $
relation_over s sel = domain sel $\subseteq s \land$ range sel $\subseteq s$	is_read a = is_atomic_load a∨is_atomic_ruw a∨is_load a	release_arquerce_art actions threads location-kind sequenced-before additional-synchronized-with data-dependency control-dependency modification-order =	
$\frac{m}{ \mathbf{x} } = m \cap (\mathbf{x} \times \mathbf{z})$		release-sequence actions threads location-kind sequenced-before additional-synchronized-with data-dependency control-dependency modification-order a b)	vidio-opportuni - also-also in vidio-opportuni - also also also in vidio-opportuni - also also also in vidio-opportuni - also also also also also also also also
	is write $a \equiv b_{a,a}$ tomic_store $a \vee is_{a,a}$ tomic_store $a \vee i_{a,a}$ tore a	hypothetical release sequence = a hypothetical observations $b = b$ is a statemic location $b, b \in C$	hills_approxed_abs_ficts = hills_approxed_abs_ficts = //are_data3 (b) //are_bata3) (b) //are_bata3) (b) //are_bata3) (b) //are_bata3) (b) //are_bata3)(b) //
$ as = as \cap (s \times s)$	is_acquire a =	$b \neq 1$ and $b \neq 0$ (b = a) \lor (b = a) \lor (m ≥ 1 multiplication under $b \land a$	due ())
$\stackrel{ni}{\longrightarrow}_{ s} = nl \cap (s \times s)$	Source memored → (even_ord ∈	$ \begin{array}{l} \mbox{Approximation} \mbox{approx} = \sigma & \mbox{Approx} \mbox{approx} \mbox{b} = \\ \mbox{(b)} (b)$	vihle gegeneer of side affects get actions throuls. Incution-lind negative deform additional-synchronized-with data-dependency motification-order happens-before wible side affect = myiming (wills) generated affect a scions through a location-lind responses deform additional-synchronized-with data-dependency motification-order happens-before wible side affect = myiming (wills) generated affect a scions through a location-lind responses affect and a dependency motification-order happens-before wible side affect = myiming (wills) generated affect a scions through a location-lind responses affect and a dependency motification-order happens-before wible side affect = myiming (wills) generated affect = myiming (w
$m _s = ml \cap (s \times s)$	A contrain the s' is further a' is further a ' dependence a' is a difference of the state of th	hypothetical_relases_sequence_set_actions_threads_location-kind sequenced-balane additional-synchronized-with data-dependency control-dependency modification-order ==	
strict_preorder and = irreflexive and / trans and	$((mem_end = MO_CONSUME) \land is_fence a)$ NONE $\rightarrow is_lock a$	hypothetical_release_sequence actions threads location-kind sequenced-before additional-synchronized-with data-dependency control-dependency modification-order a b)	consistent ended, from empring $m = consistent a mode domain a mapping m = (b_1, b_2, \dots, b_n) = b_n(b) (b) (b) (b) (b) (b) (b) (b) (b) (b) $
total_cror s orf =	is_constitute a = is_read a ∧ (memory_order a = Soure Mo_constitute)	synchronizes, with $= s \frac{q_{maximum with}}{b} b = 0$	$\begin{array}{l} \mbox{construt-mode_lines_mapping} = & \mbox{construt_mode_lines_mapping} = & \mbox{(blue lines line)} & \mbox{(blue lines)} & \mbox{(constrution)} & \mb$
total_waver x and $=$ relation_waver x and \land ($\forall x \in x \forall y \in x. x \xrightarrow{and} y \lor y \xrightarrow{and} x \lor (x = y)$)		synchronizes, with $= x \xrightarrow{q - maximum} h = 1$ $(^{2} - sidical synchronizes, the these desires etc*)$ $= x \xrightarrow{q - maximum} h = y \lor$	
	is_release a = (case memory_order a of	(some_bontion s b ∧ s ∈ actions ∧ b ∈ actions ∧ ((* - mutex synchronization - *)	$ \begin{array}{l} (2a \left(p_{ab} + k + h_{ab} - attribute (attribute (k)) \right) \\ (f \in \{2d, m_{ab}\}, (k) \in \{2d, m_{ab}\},$
<pre>strict_total_ender_over x ord = strict_preorder ord / total_ever x ord</pre>	landness an Gome samourder a of Some meaned – meaned (Marinesse Maringlet, Maringlet, Maringlet) ∧ (arith a's lands) [Note: - is antick a)	$(-mates systematics - \bullet)$ $(s-mates a \wedge k - lock b \wedge a \xrightarrow{-} b) \vee(mates a \wedge mates methods) = 0$	$\frac{1}{1} \frac{1}{1} \frac{1}$
$x \xrightarrow{ed}_{point} y =$ pred $x \land x \xrightarrow{ed}_{y} \land \neg(\exists x. pred x \land x \xrightarrow{ed}_{x} x \xrightarrow{ed}_{y} y)$	$\ $ None \rightarrow is analock a)	$(2 - \operatorname{sharm})$ incurs to the share $(2 - 1)^{-1}$ $(2 - 1)^{-1}$ ($2 - 1)^{-1}$ ($2 - 1)^{-1}$ ($2 - 1)^{-1}$) ($2 - 1$) $(2 - 1)^{-1}$ ($2 - 1)^{-1}$ ($2 - 1)^{-1}$) $(2 - 1)^{-1}$ ($2 - 1$	$(\forall x, y) \in \mathcal{L},$ $(\forall x$
	is sequent $s = (memory_order \ s = Soure \ Mo_ssiq_cost)$	· · · · · · · · · · · · · · · · · · ·	a langendar, β.∧ ansmi-kenne δ∧ Laga-ansmir-kenne δ
$x \xrightarrow{nd} y = x \xrightarrow{nd} y \wedge \neg (\exists x. x \xrightarrow{nd} x \xrightarrow{nd} y)$	location,lond = MUTEX	$P \rightarrow harm productions = 7$ (c), dense $r, k + h_{0}$ and $r, k + h_{0}$ and $r, k + h_{0}$ (c), dense $r, k + h_{0}$ and $r, r + h_{0}$ (c), dense $r, r + h_{0}$ and $r, r + h_{0}$ (c), $r + h_{0}$ and $r + h_{0}$ (c), $r $	$\begin{array}{c} \mbox{matcharm} a > h_{1} h_{2} h_{3} m_{1} h_{3} m_{1} h_{3} h_{$
wildforded r = wf r	MUTEX NON_KTOMIC ATCOMIC	$a \frac{\text{superconducting}}{(3x, x, by \text{distribution indexe superconducting}, b \land (3x, x, by \text{distribution indexe superconducting}, z \xrightarrow{di} y))) \lor$	
		[iu_fence: a Λ iu_release: a Λ iu_atemit_actions b Λ iu_acquire b Λ	V: $c = \frac{1}{2} b $ hydrother is non-algorithm of b (logitationic-logithm b) $c = 0 c = a c = a c = \frac{a c = a $
type_abbrev action_id:string	actions_respect_location_kinds = actions_respect_location_kinds = $\forall s$. case location s of SOME $l \rightarrow$	(buffers at handhow at) handhow (buffers at) (buffers and buffers at) (buffers at) (buff	(* was CAW 7) (4, a) (; martine,
type_abbrev thread_id : string	Var emer incuring a of Schurg (→) (effective Schurg) (MTTRX - is include a seminole a NOTRX - To is include a seminole a NOTRX - to is include a set of a V is stromin-artifion a) NOTRX - T NOTRX - T	$(\exists \pi \times Appendix (a) + base sequence, \pi \neq b)) \cup$	Vec. $c = d = d$. Nearth to home-boostine $d b$ histochemic-boostine $d = d = d = d = d = d = d = d = d = d $
type_abbrev location : string	ATOMIC → is_load or_store a ∨ is_atomic_action a) NONE → T	(k_stomic_action a ∧ is_robuse a ∧ _s_frace b ∧ is_acquire b ∧	
type_abbeev val: string		(antonio actino a A lacebone a A koline b A lacebone b A (b) musi p Alone A Alagoning action a A a musication b p A (ize a musication b a A ($(v(, \mathbf{A}) \in \mathcal{A}, \text{ isotomic-arrow } b) = x^{-\text{methods modes}} b) \wedge$
	is_st_location_kind = is_st_location_kind = case location s of $SOME I \rightarrow (bcather-kind I = Rd)$ $\ NONE \rightarrow F$		(r(a,b)) ⊆ 5 kompan b (for a B).
пентор_обе дила т М ₀ _202_077 М0_383_4075 1.00_483_4075 1.00_483_407 1.00_483_		synchronizm_with set actions through location-kind sequenced-bafere additional-synchronized-with data-dependency control-dependency of modification-order as release-sequence hypothetical-selase-sequence =	$\langle 0(A) \in -h_{comparts} A$ $\Rightarrow = (-h_{comp}(A) \langle A \times \pi^{-h_{com}}_{-h_{com}}, h_{com}, h_{com}) \times A \Rightarrow \pi^{-m(h_{com}, h_{com}, h_{com})} A \rangle \rangle \vee$
Mo_RELEASE Mo_ACQUIRE Mo_COMPUTE	is_at_matex_location_s = is_at_location_kind_s MUTEX		(*-finor metricion=*) (*20+1*)
Mo_ACQ_REL	ie_as_rom_stornic_location a =	$\operatorname{curris}_{a} \mathcal{A}_{b}$ productory $t_{0} = a^{-simin + a \cdot b min + a \cdot b min + b} b = a \left(\left(\frac{1}{2} - m m m m m m m m m m m m m m m m m m m$	$ \begin{array}{l} (2.323) \\ (5.53) \\ (5.$
action = LOCK of action_id thread_id location	scat_location_kind # NON_ATOMIC	carriadependency_to_aut actions threads location-kind sequenced-before additional-spectronized-with data-dependency control-dependency of =	$b_{perito} = A \log m_{perito} = A \log m_{p$
LOCK or of encycl breaked pictures LOCK or of encycl breaked pictures LATORIC_LOCK of a sciencid breaked memory-order_arean location vol ATORIC_TORIS of a sciencid breaked memory-order_arean location vol vol ATORIC_TORIS of a sciencid breaked memory-order_arean location vol vol STORE of artificial breaked location vol EFORCE of artificial breaked location vol EFORCE of artificial breaked location vol	is_st_intenie_location s = is_st_location_kind s ATOMIC	carring_i_depandency_lo_aut actions threads location-ind sequenced-batter additional-synchronized-with data-dependency control-dependency if = carring_i_dependency_lo_actions threads location-kind sequenced-batter additional-synchronized-with data-dependency control-dependency (r a b)	⇒ (y = 4) / 2 → → → y) ∧ (234 × 1) (234 × 1)
ACOMIC_RIVE or account mean of memory_order_mean location val ACOMIC_RIVE of action.id thread.id memory_order_mean location val val [LOAD of action.id thread.id location val	some_thread $s b = (thread_id_of s = thread_id_of b)$	$d = \frac{1}{x \in \operatorname{action} \land d \in \operatorname{action} \land} d = \frac{1}{x \in \operatorname{action} \land d \in \operatorname{action} \land d \in \operatorname{action} \land d \in \operatorname{action} \land d \in \operatorname{action} \land$	$(\gamma_{k+1}) \in \frac{\tau_{k+1}}{\tau_{k+1}} (\gamma_{k+1}) \in \mathcal{A},$ (Castingting the λ_{k+1} by
STORE of action_id thread_id location val FEXCE of action_id thread_id memory_order_enum		a ∈ actions ∧ d ∈ actions ∧ (3b. isznehose a / iszonemen b ∧	$ \begin{array}{l} (2 \times 3) \times 1_{[n]} & \\ (p_{1,n}) \in \mathcal{I}_{n} & \\ (p_{1,n}) \in \mathcal$
	threadwise relation over a $nd =$ relation over a $nd \wedge (\forall (a, b) \in nd.$ same thread $a b$)	$h \in \operatorname{actions} A \in \operatorname{actions} A$ $(B_h \operatorname{bachours} + \operatorname{bachourses} h \land$ $(B_h - \operatorname{constanting} a \leq b_j) \land$ $(B_h - \operatorname{constanting} a \leq b_j) \land$ $(B_h - \operatorname{constanting} a \leq b_j) \land$	(235)
(action_id_of (LOCK aid) = aid) ∧			
$(\operatorname{action}_{\operatorname{act}}\operatorname{act}_{\operatorname{act}}(\operatorname{LOCK} \operatorname{ast}_{-}) = \operatorname{ast}) \land$ $(\operatorname{action}_{\operatorname{act}}\operatorname{act}_{\operatorname{act}}(\operatorname{LOCK} \operatorname{ast}_{-}) = \operatorname{ast}) \land$ $(\operatorname{action}_{\operatorname{act}}\operatorname{act}_{\operatorname{act}}(\operatorname{LOCK} \operatorname{act}_{-}) = \operatorname{ast}) \land$ $(\operatorname{action}_{\operatorname{act}}\operatorname{act}_{\operatorname{act}}(\operatorname{Act}_{\operatorname{act}}(\operatorname{Cact}_{\operatorname{act}} \operatorname{cact}_{-}) = \operatorname{ast}) \land$	same_location $a \ b = (\text{location } a = \text{location } b)$	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 =	$(v_{k,n}) \in , \forall p_i p_i \in , \forall p_i$ (basing index of k-black ext k-blac
$\begin{array}{l} (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{actional Lie} (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{Lock} u = ,] = u \in), \\ (\operatorname{Lock} u = ,] = u $		dependency-andread_before-ant-actions through location-bind sequenced-before additional-synchronized-with data-dependency cantrol-dependency of multifaction-order release sequence carries a dependency to = dependency-antrend_before actions through to because before additional-synchronized with data-dependency cantrol-dependency of multifaction-order release sequence carries a dependency to a b]	$(\psi_{n,k}) \in \frac{1}{2} + $
	locations_of actions = {l. is. (location $a = \text{Sout: } l$)}		$\begin{array}{l} (*2.85) \\ (*4.8) \leq \frac{1}{2} \left(\frac{1}{2} \left(\frac{1}{2} \right) \left(\frac{1}{2} \right) \left(\frac{1}{2} \left(\frac{1}{2} \right) \left($
	locations_of actions = {l. is. (location $a = \text{Sout: } l$)}	Appendrug weberd Jahren ut in steam from de forste for informed officer all protocols with data dependency served dependency of model stations and we share sequence series a dependency to : dependency weberd dependency of model stations and weberd before additional quadratical data dependency series dependency of model stations and weber sequence series a dependency to a 3 [
	locations_of actions = {l. is. (location $a = \text{Sout: } l$)}	Employage and the second secon	Al-data-dependency = Addread-top-tage = (\$\l_1\
	locations_of actions = {l. is. (location $a = \text{Sout: } l$)}	A melokakapan Jakar a melokamataka, a (melokakapan Jakar a) ender (melokakapan Jakar a) ender (melokakapan Jakar a) ender (melokakapan Jakar a)	d data depador + American a - A
$ \begin{array}{l} (\operatorname{bban}_{d,d,d}(f(\omega_{n},\omega_{d}),\ldots,\omega_{d}), \alpha(\omega_{n},\omega_{d}), \alpha(\omega_{n},\omega_{d$	Institute of actions of U.S. (Contine of States (1)) with formation of the state o	Finisk-Lappen Safety = United management Image Safety = Safety	d data depador + American a - A
$\begin{split} & \{ \max_{i=1}^{k} (i,j,a,i), (i,j,a,i) \in [n] \}, \\ & \{ \max_{i=1}^{k} (i,j,a,i), (i,j,a$	Institute of actions of U.S. (Contine of States (1)) with formation of the state o	Finisk-Lappen Safety = United management Image Safety = Safety	d data depador + American a - A
$\begin{split} & \{ \max_{i=1}^{k} (i,j,a,i), (i,j,a,i) \in [n] \}, \\ & \{ \max_{i=1}^{k} (i,j,a,i), (i,j,a$	Institute of actions of U.S. (Contine of States (1)) with formation of the state o	A melokakapan Jakar a melokamataka, a (melokakapan Jakar a) ender (melokakapan Jakar a) ender (melokakapan Jakar a) ender (melokakapan Jakar a)	d data depador + American a - A
$\begin{split} & \{ \max_{i=1}^{k} (i_{i_{i_{i_{i_{i_{i_{i_{i_{i_{i_{i_{i_{i$	Institute of actions of U.S. (Contine of States (1)) with formation of the state o	Finisk-Lappen Safety = United management Image: Control of the same state of the	d data depador + American a - A
$\begin{split} & \{ \max_{i=1}^{k} (i_{i_{i_{i_{i_{i_{i_{i_{i_{i_{i_{i_{i_{i$	Institute of actions of U.S. a (Institute of Statistic) settlemend-action of Statistics (Statistics) (Institute of Statistics) (Institute of Statistics)	Angele Augence Aufors = minimum and angele angele and angele and angele and angele angele and angele ange	d data depador + American a - A
	Institute of actions of U.S. a (Institute of Statistic) settlemend-action of Statistics (Statistics) (Institute of Statistics) (Institute of Statistics)	New Address State Address (minimal Address </td <td>d data depador + American a - A</td>	d data depador + American a - A
	$\label{eq:second} kenter \in \{1,2,\{kenter s = 8000, 0\}\}$ will demonstrating s = $\label{eq:second} kenter s = \\ Monstration kenter s = manual s = \\ Monstration kenter s = \\ Mons$	Angle Approx. Justice = Stationarching = (minimum data); - (minimum data); - Anshare (Sh) - Stationarching : Stationarching = (minimum data); - Stationarching : (minimum data); -	d data depador + American a - A
	hentime (f attern = (f a (hentin = 5000. 0)) will demondering = f attern = f a	Imple Augment Aufors = Streammanday, = (minima augment Aufors = Streammanday, = (minima augment Aufors = Augment Aufors = Augment Augment Aufors = Augment Augment Aufors = Augment	A default of a second s
	hentime (f attern = (f a (hentin = 5000. 0)) will demondering = f attern = f a	<pre>inforkupper_lation = methamatikes = inforkupper_lation = meth</pre>	d data depador + American a - A
$ \begin{cases} \label{eq:constraint} \left\{ \begin{array}{llllllllllllllllllllllllllllllllllll$	hentime (f attern = (f a (hentin = 5000. 0)) will demondering = f attern = f a	<pre>inforkupper_lation = methamatikes = inforkupper_lation = meth</pre>	d. documents of documents :
$ \begin{cases} \label{eq:constraint} \left\{ \begin{array}{llllllllllllllllllllllllllllllllllll$	$\label{eq:constraint} \begin{split} & \operatorname{Instraint}_{max}(d \ artism \in \{1, 2 \ (b \ order in \ s = 5 \ order \ s = 1 \ or$	ImplexApprox.lation = minimized and minim	Added Agendance of Added Added and Added A
$ \begin{cases} \label{eq:constraints} \left\{ \begin{array}{llllllllllllllllllllllllllllllllllll$	$\label{eq:constraint} \begin{split} & \operatorname{Instraint}_{max}(d \ artism \in \{1, 2 \ (b \ order in \ s = 5 \ order \ s = 1 \ or$	Single Algorization - Single Algori	A default of the second
$ \begin{cases} \label{eq:constraints} \left\{ \begin{array}{llllllllllllllllllllllllllllllllllll$	hentime (f attern = (f a (hentin = 5000. 0)) will demondering = f attern = f a	Instrume_instrume	A default of the second
$ \begin{cases} \label{eq:constraint} \left\{ \begin{array}{llllllllllllllllllllllllllllllllllll$	$\label{eq:second} \begin{array}{l} \mbox{linear setum} (\mbox{linear setum}$	Instrume_instrume	A default of the second
$ \begin{cases} \underset{\substack{\text{there } (A, C, C,$	$\label{eq:second} a status = (l.ta (bostim s = bost 0)) \\ \mbox{withdrawalextim s = } \\ \mbox{withdrawalextim s = magnet = - magnet (bostim s = bost 0) } \\ \mbox{withdrawalextim s = magnet = - magnet (bostim s = bost 0) } \\ \mbox{Withdrawalextim s = magnet = - magnet (bostim s = bost 0) } \\ \mbox{Withdrawalextim s = magnet = - magnet (bostim s = bost 0) } \\ \mbox{Withdrawalextim s = magnet = - magnet (bostim s = bost 0) } \\ \mbox{Withdrawalextim s = magnet = - magnet (bostim s = bost 0) } \\ Withdrawalextim s = magnet = - magnet (bostim s = bostim s $	Instrume_integrations Instrume_integrations Instrume_integratintegrations Instrume_integrat	<pre>d.dc.getaparties_checkenomes_checkenomes_checkenomes de la setation de production and de de dependency and de de de dependency and de de de de dependency and de de de dependency and de de de</pre>
$ \begin{cases} \label{eq:second} \left\{ \begin{array}{llllllllllllllllllllllllllllllllllll$	$\label{eq:second} a status = (l.ta (bostim s = bost 0)) \\ \mbox{withdrawalextim s = } \\ \mbox{withdrawalextim s = magnet = - magnet (bostim s = bost 0) } \\ \mbox{withdrawalextim s = magnet = - magnet (bostim s = bost 0) } \\ \mbox{Withdrawalextim s = magnet = - magnet (bostim s = bost 0) } \\ \mbox{Withdrawalextim s = magnet = - magnet (bostim s = bost 0) } \\ \mbox{Withdrawalextim s = magnet = - magnet (bostim s = bost 0) } \\ \mbox{Withdrawalextim s = magnet = - magnet (bostim s = bost 0) } \\ Withdrawalextim s = magnet = - magnet (bostim s = bostim s $	Imple Augmentation = minimum and a second and second and a second and a second and a second	<pre>d.dc.getaparties_checkenomes_checkenomes_checkenomes de la setation de production and de de dependency and de de de dependency and de de de de dependency and de de de dependency and de de de</pre>
$ \begin{cases} \underset{\substack{\text{there } (A, C, C,$	$\label{eq:second} a status = (l.ta (bostim s = bost 0)) \\ \mbox{withdrawalextim s = } \\ \mbox{withdrawalextim s = magnet = - magnet (bostim s = bost 0) } \\ \mbox{withdrawalextim s = magnet = - magnet (bostim s = bost 0) } \\ \mbox{Withdrawalextim s = magnet = - magnet (bostim s = bost 0) } \\ \mbox{Withdrawalextim s = magnet = - magnet (bostim s = bost 0) } \\ \mbox{Withdrawalextim s = magnet = - magnet (bostim s = bost 0) } \\ \mbox{Withdrawalextim s = magnet = - magnet (bostim s = bost 0) } \\ Withdrawalextim s = magnet = - magnet (bostim s = bostim s $	Imple Augmentation = minimum and a second and second and a second and a second and a second	<pre>d.dc.getaparties_checkenomesta_control de production de la control de la control</pre>
$ \begin{aligned} & \left \begin{array}{l} \left \begin{array}{l} \left $	$\label{eq:constrained} artime ((2a (localin a = 5000. 0)) \\ \mbox{withdrawedsettin a = 1 \\ Marca 200. 0 = mage a^{-1} a^{-1} mage a^{-1} (2a (localin a = 5000. 0)) \\ \mbox{withdrawedsettin a = 1 \\ Marca 200. 0 = mage a^{-1} a^{-1} mage a^{-1} (2a (localin a = 1000. 0)) \\ withdrawedsettin a = 1 \\ Marca 200. 0 = 1 \\ \mbox{withdrawedsettin a = 1 \\ Marca 200. 0 = 1 \\ \mbox{withdrawedsettin a = 1 \\ Marca 200. 0 = 1 \\ \mbox{withdrawedsettin a = 1 \\ \mbox{wit$	Implementation = method managementation	add Approx Approx Addition and Approx App
$ \begin{cases} \label{eq:second} \left\{ \begin{array}{llllllllllllllllllllllllllllllllllll$	$\label{eq:constrained} a status = (1 a (bostim s = biost. 0))$ will demonstration s =	Imple Augmentation = minimum and a second and second and a second and a second and a second	<pre>d.d.d.g.u.g.t.d.d.d.d.d.d.d.g.u.g.t.d.d.g.u.g.t.d.d.g.u.g.t.d.g.u.g.u.g.t.d.g.u.g.t.d.g.u.g.t.d.g.u.g.t.d.g.u.g.u.g.t.d.g.u.g.u.g.t.d.g.u.g.u.g.t.d.g.u.g.u.g.t.d.g.u.g.u.g.t.d.g.u.g.u.g.u.g.t.d.g.u.g.u.g.u.g.u.g.u.g.u.g.u.g.u.g.u.g</pre>
$ \begin{cases} p_{max}(d_{1}d_{2}d_{2}d_{2}d_{2}d_{2}d_{3}d_{2}d_{3}d_{3}d_{3}d_{3}d_{3}d_{3}d_{3}d_{3$	$\label{eq:constrained} a status = (1 a (bostim s = biost. 0))$ will demonstration s =	Imple Augment Aufors - Station and and a station and a	<pre>d.d.d.g.u.g.t.d.d.d.d.d.d.d.g.u.g.t.d.d.g.u.g.t.d.d.g.u.g.t.d.g.u.g.u.g.t.d.g.u.g.t.d.g.u.g.t.d.g.u.g.t.d.g.u.g.u.g.t.d.g.u.g.u.g.t.d.g.u.g.u.g.t.d.g.u.g.u.g.t.d.g.u.g.u.g.t.d.g.u.g.u.g.u.g.t.d.g.u.g.u.g.u.g.u.g.u.g.u.g.u.g.u.g.u.g</pre>
$ \left \begin{array}{c} \left \begin{array}{c} \left $	$\label{eq:constraint} \begin{tabular}{lllllllllllllllllllllllllllllllllll$	Imple Augment Aufors - Station and and a station and a	<pre>d.d.d.g.u.g.t.d.d.d.d.d.d.d.g.u.g.t.d.d.g.u.g.t.d.d.g.u.g.t.d.g.u.g.u.g.t.d.g.u.g.t.d.g.u.g.t.d.g.u.g.t.d.g.u.g.u.g.t.d.g.u.g.u.g.t.d.g.u.g.u.g.t.d.g.u.g.u.g.t.d.g.u.g.u.g.t.d.g.u.g.u.g.u.g.t.d.g.u.g.u.g.u.g.u.g.u.g.u.g.u.g.u.g.u.g</pre>
$ \begin{cases} p_{max}(d_{1}d_{2}d_{2}d_{2}d_{2}d_{2}d_{3}d_{2}d_{3}d_{3}d_{3}d_{3}d_{3}d_{3}d_{3}d_{3$	<pre>heating_d attract = { 12 k [cosing s = hose ()] will_annul_set = s All the set of the set of</pre>	Implementation = Stationarchics = (Stationarchics : Stationarch	<pre>d.d.d.actions::::::::::::::::::::::::::::::::::::</pre>
$ \left \begin{array}{c} \left \begin{array}{c} \left $	$\label{eq:constraint} \begin{tabular}{lllllllllllllllllllllllllllllllllll$	Imple Augment Aufors - Station and and a station and a	add approximation and intermediation of the second approximation approximation of the second approximation of the second approximation of the second approximation appr

I he full model

Land Answer (Control of Control of		is store $a=case$ a of Store $\ldots \to T \parallel \ldots \to F$		
Internation Ander and and an an and an an an and an a	$x \stackrel{\prime}{\rightarrow} b = (x, b) \in r$	in form on some of Freezen in Till in F		vide-uide-uide-affect-uit actions timule location-kind sequenced-bidre additional-specherolycestrate dependency lappens-kefor =
NumberNumberNumberNumberNumberImage: State S	$a r b = (a, b) \in r$	Particle of a Mark of the Constant of the Cons	real-ment real-hard a = school a real-hard V icatomic more a	visible_side_statics extinus lances latera kind sequences balow additional-spectromined-with data-dependency control dependency lagrams before a b)
Interface Name Note of the second se		is_lock_or_unlock $a = is_u lock \ a \lor is_u m lock \ a$	release severence = s , severe segments h =	visibio-expressional-advantieve visibia-expressional-advantieve to an advantieve to a state of the second
	$a \stackrel{_{a}}{\rightarrow} b = (a, b) \notin r$	is_stomic_action a =	is_at_atomic_location $b \land$	{ ⊂, ruse-had windows with c ∧ _L, k winn in
Number Mathematical Margina Margina Margina Margina Margina Section Margina Margina Margina Margina Margina Margina Section Margina Margina Margina Margina Margina Margina Section Margina Margina Margina Margina Section Margina Margina Margina Margina Section Margina Margina Margina Margina Section Margina Margina Margina Margina Section Margina Margina Margina Margina Section Margina Margina Margina Margina Section Margina Margina Margina Margina Section Margina Margina Margina Margina Section Margina Margina Margina Margina Section Margina Margina Margina Margina Section Margina Margina Margina Margina Section Margina Margina Margina Margina Section Margina Margina Margina Margina Section Margina Margina Margina Margina	$\dot{\rightarrow} = r$	is_atomic_load a Vis_atomic_store a Vis_atomic_rmw a	$(b = a_{\alpha}) \lor$ $(reduction a_{\alpha}, b \land a_{\alpha}, mathematics, b \land$	(%, van, hat manhitum and hat manhitum a subfactur and c
Junch Junch <th< td=""><td>A. 1. A</td><td>is load or store $a = is$ load $a \lor is$ store a</td><td>(Vc. au management) c management b =></td><td></td></th<>	A. 1. A	is load or store $a = is$ load $a \lor is$ store a	(Vc. au management) c management b =>	
Betware Ausside Ausside Ausside Ausside Carlor Second Second Second Second Second Second Second Second Second Second Second Second <td< td=""><td>$a \rightarrow b \rightarrow c = a \rightarrow b \land b \rightarrow c$</td><td></td><td></td><td>$myimage \ f \ s = \{p, 2k \in \kappa \ (p = f \ s)\}$</td></td<>	$a \rightarrow b \rightarrow c = a \rightarrow b \land b \rightarrow c$			$ myimage \ f \ s = \{p, 2k \in \kappa \ (p = f \ s)\} $
Note of a set	relation_over s rel = domain rel $\subseteq s \wedge \mathrm{range}$ rel $\subseteq s$		J J	vide annous of vide efforts - vide monrow of vide efforts -
Notice Notice Notice Notice 1 - 1 - 1 Notice Notice Notice <td>$\frac{m!}{m} _{x} = m! \cap (x \times x)$</td> <td>is_write a =</td> <td>relense_sequence actions threads location-kind sequenced-before additional-synchronized-with data-dependency control-dependency modification-order a b)</td> <td>X (une-knot, b). (6) if iso-intrinsic-torinsic b then</td>	$\frac{m!}{m} _{x} = m! \cap (x \times x)$	is_write a =	relense_sequence actions threads location-kind sequenced-before additional-synchronized-with data-dependency control-dependency modification-order a b)	X (une-knot, b). (6) if iso-intrinsic-torinsic b then
		is_atomic_store a∨is_atomic_mw a∨is_store a	$by post-brief a functional sequence = s^{-1} y multiplicative angumes, b = 1$	visible wearsnes of side, effects, tail was , baad b
Instruction Instruction Instruction Instruction Instruction Instruction Instruction <td< td=""><td>$rol_s = rol \cap (s \times s)$</td><td>is sequire a =</td><td>$(b = a) \vee$</td><td>die ()</td></td<>	$rol_s = rol \cap (s \times s)$	is sequire a =	$(b = a) \vee$	die ()
19.13 Implementation Implementation Implementation 19.14 Implementation Implementation Implementation 19.14 Implementation Implementation Implementation 19.15 Impl	$\xrightarrow{m _{s}} = ml \cap (s \times s)$	Some mem_ord \rightarrow	(Vc. a modification order or modification order b ==>	in the mean of the first set wine these busines intermediate additional methods and the dependence and a basis of the mean busines in the set of the set o
		{Mo_ACQUIRE, Mo_ACQ_REL, Mo_SEQ_CST} ^	relations a c())	my image (visible sequences, of solis actions through location-kind sequenced-before additional spectrominal with data dependency motifications only happens before with be side-effect) with a side side officet)
Instrument Instrument Instrument Instrument Instrument Instrum	$nol_s = nol \cap (s \times s)$	(* 29.8:5 states that consume fences are acquire fences. *)	hypothetical_release_wepance_and actions threads location-kind aspanced-before additional-synchronized-with data-depandency control-depandency modification-order =	zowistat zash fom mozing = conistant mak fom mozing =
Provide Lange Statistics Provide Statistics Provid Statistics Provide Statistics Provide	strict-preorder and = irreflexive and $\wedge {\rm trans}$ and	$\ $ NONE \rightarrow is lock a)	bypothetical_release_sequence actions threads location-kind sequenced-before additional-synchronized-with data-dependency control-dependency modification-order a b}	(b) (a wind b is defined to be a set of the
Name House House House Name House House		is_consume a =	synchronizes, with $= a \xrightarrow{\text{synchronizes with}} b =$	then $(3s_{\mu\nu}, s_{\mu\nu}, \frac{1}{2}) = b \wedge s_{\mu\nu} + \frac{1}{2} b$
LandMain State </td <td>relation_over s and ∧</td> <td>is_read $a \wedge (memory_order a = Soure Mo_constant)$</td> <td>(* – additional synchronization, from thread create etc. – *) a ^{dditional synchronization} b ∨</td> <td></td>	relation_over s and ∧	is_read $a \wedge (memory_order a = Soure Mo_constant)$	(* – additional synchronization, from thread create etc. – *) a ^{dditional synchronization} b ∨	
Image: state	$(\forall x \in x, \forall y \in x, x \longrightarrow y \lor y \longrightarrow x \lor (x \equiv y))$			$(if (\exists (b', vaue) \in viuble-sequences-of-side-effects, (b' = b))$
Image: State	strict_total_order_over s ord =	SOME mem_ord ->		
Party of the state of the st	station and a consigned a dis-		(* - release/acquire synchronization - *)	
Image: section of the section of th	$x : \stackrel{\text{red}}{\longrightarrow}_{pred} y =$ and $x \ge -\frac{\text{red}}{2} x \ge -\frac{1}{2} x$ much $x \ge -\frac{\text{red}}{2} x = \frac{\text{red}}{2} x \ge \frac{1}{2} x = \frac{1}{2} x \ge \frac{1}{2} x = \frac{1}{2} x = \frac{1}{2} x = \frac{1}{2} x =$	TOAL OF BEINGE 3	(a) reasons a > (b) (c) (c) (c) (c) (c) (c) (c) (c) (c) (c	$\forall x, b \in \mathcal{A}$
Lange (a) (a) (b) (b)Instance (b) (b) (b)Instance (b) 	processory and the state of the	is_sequest a = (memory_order a = Soure Mo_stoq_cort)		same location $a b \wedge \log at atomic location b$
Linear diamage Image diamage Image diamage diamag	$x \xrightarrow{\text{ord}} y = $	location.kind =	(∃x, ∃y, same_location x y ∧	(* new CoWR *)
index of the state of the s	$x \longrightarrow y \land \neg (\exists x. x \longrightarrow x \longrightarrow y)$	MUTEX NON_ATOMIC	$a \xrightarrow{sequenced intervent} x \land y \xrightarrow{sequenced intervent} b \land$	Vc.
Image: state	well, founded $r = w f \ r$	Arounc		is made a 4 years burstion a \$4.5 is at atomic burstion \$
$$ $$ $$ $$ $$ $$ $$ $$ $$ $$ $$ $$ $$	type_ableev action_id : string	actions_respect_location_kinds = actions_respect_location_kinds =	is atomic action $b \wedge is acquire b \wedge$	
IncludeInclu		va. case location a of Source $I \rightarrow$	sequenced before	$(\gamma_{i} \in A_{i}) \in \frac{Lymma holor_{i}}{V_{i}}$.
Image:	type_abhrev thread_id : string	$MUTEX \rightarrow is lock or unlock a$	$(3x, x, \frac{by which wild enhance sequence}{2}, x, \frac{d}{b} b))) \lor$	is write b / same location a b / is at atomic location a
	type_abhrev location : string	ATOMIC \rightarrow is load or store $a \lor is$ atomic action a)	(is a tomic action of a backware of a	$\implies c \text{number of the set of th$
Interfact Part of the state of			$(\exists x, same location x \land is atomic action x \land$	d = m + m + m + m + m + m + m + m + m + m
$\left \left \left$	type_abbrev val : string	case location a of	$(\exists x. x \xrightarrow{\text{release subsection}} x \xrightarrow{d} x)))))$	
Image: Single		Some $l \rightarrow (location-kind l = lk0)$ None $\rightarrow F$	with the action through location-kind answered before additional-withronized-with data-dependency control-dependency of modification-order as release-sequence broathetical-sequence =	\Rightarrow (-is used of a \land (Vi.x. $\xrightarrow{m}_{b,b}$, in order scatter instance $b \in b \Rightarrow x$ medification order, a)) \lor
Image: Mode and State a	MO_RELAXED	is at water busiles an		
$\left \begin{array}{c c c c c } \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\$	Mo_acquire	is at location kind a MUTEX		(*2833*)
$ \left \begin{array}{c} \text{and } \text{and } \text{derived} deri$	Mo_ACQ_REL	is_st_non_stonic_location a =	$carries_{ij}dependency to = s$ $= \frac{1}{1-1} \int b = \frac{1}{1-1} $	$(i_{x}j_{mon} \times \land i_{y})_{x} = 0$
$\left \begin{array}{c} \left \begin{array}{c} \left $		is_at_location_kind a NON_ATOMIC		$a \rightarrow x \wedge x \rightarrow b$
$\left \begin{array}{c} \left \begin{array}{c} \left \begin{array}{c} \left $	UNLOCK of action, id thread, id location	is_st_stomic_location a =		$\Longrightarrow (y = a) \lor a \xrightarrow{f = d(a) = a = a} y) \land$
$\left \begin{array}{c} \int_{\Omega} drugt drugt drugt grugt drugt grugt drugt grugt drugt drugt grugt drugt drugt grugt drugt drugt grugt grugt drugt grugt gr$	ATOMIC_STORE of action_id thread_id memory_order_enum location val	is_nt_location_kind a ATOMIC	earning_appendency to action threads location-kind acquired-before additional-synchronized-with data-dependency control-dependency of a b}	(223.4) $(3_{12}, \gamma) \equiv maximizing M_{12} \otimes M_{22} = \frac{1}{2}$
$\left \begin{array}{c} \left[\begin{array}{c} \left[$	LOAD of action id thread.id location val STORE of action id thread.id location val	same_thread s b = (thread_id_of s = thread_id_of b)		[b_citonic_action a / h_k=report a / b_cryster a / harm_boation a / h /
(minute)	FEECE of action id thread id memory order enum		(3b, is release a / is comment b /	$x \xrightarrow{a} b \wedge is_{a}$ dominant in b
$\begin{bmatrix} $	$(action.id.of (LOCK aid) = aid) \land$	threaderse_relation_over s ref = relation_over s ref \wedge ($v(s, b) \in ref.$ same_thread s b)	$(26, 2 \rightarrow \cdots \rightarrow 0) \land$ $(b \xrightarrow{invises a dispatchery in} d \lor (b = d)))$	(* 20.3.5 *)
$\begin{bmatrix} $		some location $a, b = flocation a = location b$		figuration a \wedge indexe $x \wedge$ indexe to $x \wedge$
(a) (b) (c) (c) <td>(action_id_of (ATOMIC_RMW aid aid) ∧</td> <td></td> <td></td> <td>is stomic setion b \wedge same location a b \wedge</td>	(action_id_of (ATOMIC_RMW aid aid) ∧			is stomic setion b \wedge same location a b \wedge
Image: Second	$(action_id_of (STORE aid = =) = aid) \land$	locations of actions = { l . $\exists s$. {location $s = \text{SOME } l$ }		$J = \frac{1}{\sqrt{2}} \sqrt{2 + \frac{1}{\sqrt{2}}} \frac{1}{\sqrt{2}} \frac{1}{\sqrt{2}$
(bind)(i/(10x; i/d) = 0)/ are of a log (i/d)		will formed_action a =	simple_happen_bhfore = simple_happen_bhfore = simple_happen_bhfore = = // simple_happen_bhfore = simple_happen_bhf	
	(thread_id_of (UNLOCK _ tid _) = tid) ∧	case s of ATOMIC_LOAD mem.ord → mem.ord ∈		Al defactored production = defactored and a second
	(thread_id_of (ATOMIC_LOAD_tid) = tid) ∧	{Mo_BELAXED, Mo_ACQUIRE, Mo_SEQ_CST, Mo_CONSUME} ATOMIC_STOREmem_and → mem_and ∈	consistent_aimple_happens_before whb = imple_size(^{Ab})	

We can reason about C concurrency!

(value, written (ATOHIC, RHW	∧ (is_atomic_load b∨is_atomic_rmw b∨is_load b)))	$(\forall a, \forall b, a \xrightarrow{(d)} b \implies same_{a}$ between $b \models A$ $(\forall l \in \text{locations}, c \rightarrow d$ for $b \models A$ $A \cap Out c \rightarrow d$	 Market and the second se
(value_written _ = NONE)	all_lock_or_unlock_actions_at fopt $au = \{a \in au. is_lock_or_unlock a \land \{location a = lopt\}\}$	let $articular = J = \{a, \ articular a = SOME 1\}$ in let $articular = \{a, art, articular a \lor \}$ $ articular = \{a, art, articular a \lor \}$ in	(a, b) = a = b = b = b = b = b = b = b = b = b
sclock $a = case a$ of Lock $\ldots \rightarrow T \parallel \ldots \rightarrow F$	consistent_locks = consistent_locks = $\forall l \in locations_of extract_location-bind l = MUTEX) \implies ($	entrational status of a segmentation of a segment of the segmento	atu, puor' action tanuto facuitos kiel segureos befora additional-pretorninal with data dependency cantel dependency ef modification-andre ac = ht. elem-secures = minus anamest ant action tanuto la location-injet anamesta befora defendency cantel dependency effected dependency entrel dependency ent
which $a=$ case a of UNLOCK _ , , , , , , , , , , , , , , , , , ,	v) ∈ incutioned actions. [Incataos-knot] = MUTEX) == (the light of the light of	Augenite later, manufacture of motion interest, A (* MAQ_SIGE_CTS' forces impose modification order *) (* manufacture of motion impose modification order *) (* manufacture of motion impose modification order *)	Mt. hypothetic-indexe-square: n fuez, gamper_art action: their asparsed before additional-protocological distances optimized by the spectra opt
unconstruction of a set of Atomic_LOAD \rightarrow T _ \rightarrow F	vitrici_statul_ander_aviver lock_antl	$ \sum_{i = 1 \\ i \neq i \neq i \\ (a \neq b \neq i \neq i \neq i \\ i \neq i \neq i \neq i \neq i \\ (a \neq b \neq i \neq i$	M inter betward supports hafter wind approximately and a statistication of the approximate statistication
case a of ATOMIC_STORE $\to T \parallel_\to F$	(* 30.4.2.0 Requires: The calling thread shall can the metate: *) (* 30.4.2.12 fifter: Knass the calling thread consention of the mater.*) (*A _n ∈ fock-mitod-actions. is unlock a _n ⇒> (² A _n ∈ fock-mitod-actions. is unlock a _n ⇒> (² A _n ∈ fock-mitod-actions.	$\frac{1}{ab_{1}a_{2}a_{3}b_{2}a_{3}b_{3}a_{4}b_{3}a_{4}b_{4}a_{5}b_{5}}b=\frac{1}{a_{1}a_{2}a_{3}a_{3}b_{3}a_{4}b_{5}b_{5}}b_{5}b_{5}b_{5}b_{5}b_{5}b_{$	<pre>exp_memory_model quess (pc / program) =</pre>
when it case a of Atomic_Raw $\ldots \to T \parallel \: \to F$	(* 30.4.1.7 Effects: Blocks the calling thread until ownership of the mutex can be obtained for the calling thread.*) (* 30.4.13 Postcondition: The calling thread owne the mutex. *)	is vertice a $h = h_{min}$ db h same basis as $b \wedge \frac{1}{(2\pi)^{m}} \left(\frac{1}{2\pi} \left(\frac{1}{2\pi} \right) \left(\frac{1}{2\pi} \left(\frac{1}{2\pi} \right) \right) \left(\frac{1}{2\pi} \left(\frac{1}{2\pi} \right) \right) \right)$ is vertice $c \wedge \text{same}(-\text{densities } c \wedge 1)$ a <u>sum vertice $(c + 1)$</u>	Conference on actions: them the learnine intermediation of approximately due to a for
is lead a = case a of Lead $\ldots \rightarrow T \parallel _ \rightarrow F$	$(\forall a_i \in lock_{antlock_{actions.}} \cup lock a_i \implies$ $(\forall a_i \in lock_{antlock_{actions.}} = a_i \implies i_{i,unlock_{actions.}} = a_i \implies$		dia secular

```
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*

int a = 1;

This is a compiler bug

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

int a = 1;

This is a concurrency compiler bug

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

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

Thread 1

}

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

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

Thread 1

int a = 1; int b = 0;

Thread 1

```
int s;
for (s=0; s!=4; s++) {
    if (a==1)
        return 0;
    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 0;
  for (b=0; b>=26; ++b)
    ,
}
            Thread 1 returns without modifying b
```

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

Thread 1

```
b = 42;
int s;
for (s=0; s!=4; s++) {
                                   printf("%d\n", b);
  if (a==1)
    return 0;
  for (b=0; b>=26; ++b)
    ,
}
            Thread 1 returns without modifying b
     Thread 2 is not affected by Thread 1 and vice-versa
               (this program is data-race free)
            This program must always print 42
```

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

Thread 1

Thread 2

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

Typical system code!

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

Thread 1

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



... in some executions might print 0

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

g

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

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

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

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

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

g

The compiler has introduced the prefetch and restore of **b**

Surprising but correct in sequential executions

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

```
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) movl \$0, %eax ret

```
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)
movl	\$0, %eax
ret	

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

Thread 1

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

- 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) movl \$0, %eax ret

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

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

Introduces unexpected behaviours in some concurrent context

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

Introduces unexpected behaviours in some concurrent context

This is a concurrency compiler bug

ret

.L2:

- movl %ebx, b(%rip)
- movl \$0, %eax

ret

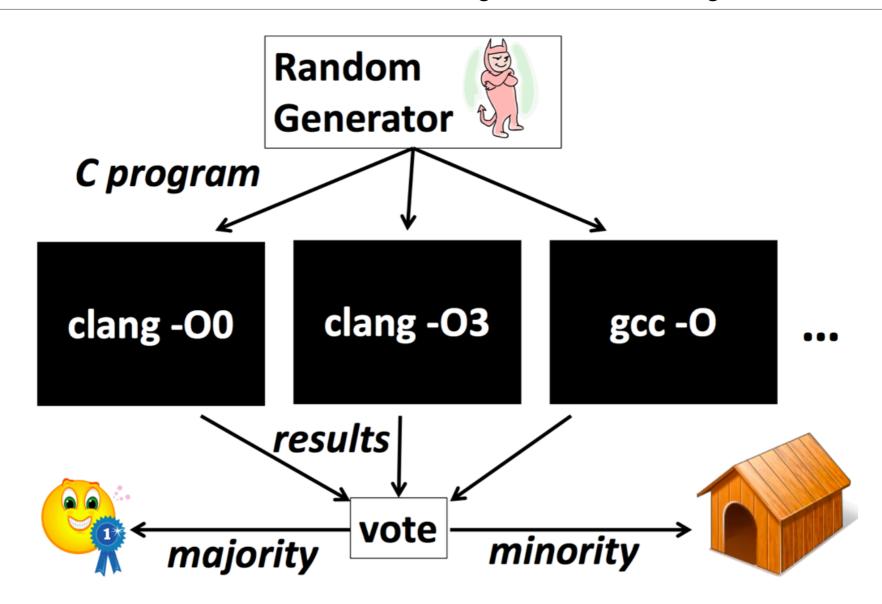
- Read a (1) into eax
- Read b (0) into ebx
- Store 42 into b
- Store ebx (0) into b
- Print b... 0 is printed

Introduces unexpected behaviours A bug report is not research. A *technique* to identify concurrency compiler bugs in existing compilers is! - Store edx (ש) into u

- Print b... 0 is printed

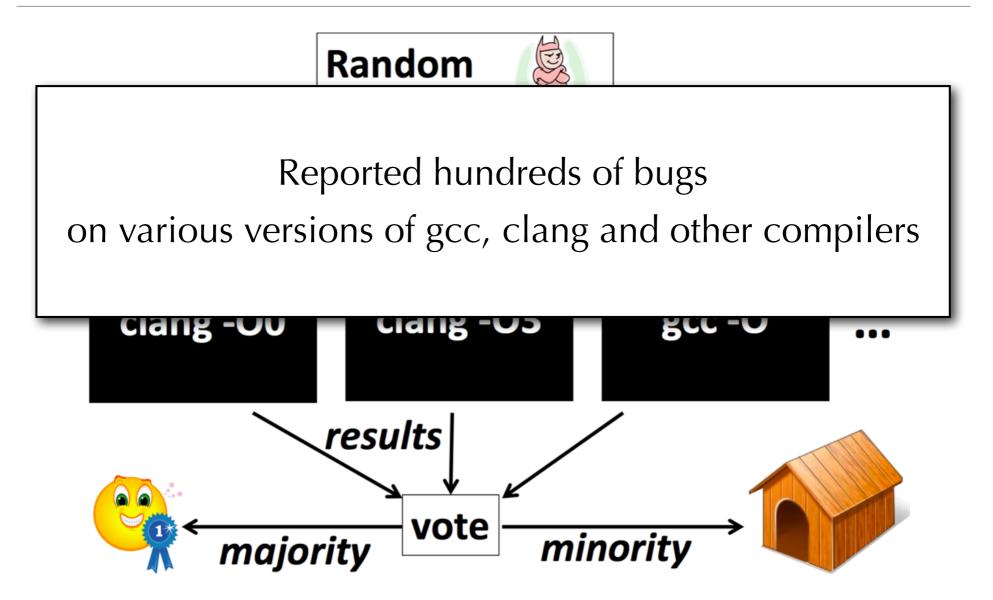
Compiler testing: state of the art

Yang, Chen, Eide, Regehr - PLDI 2011



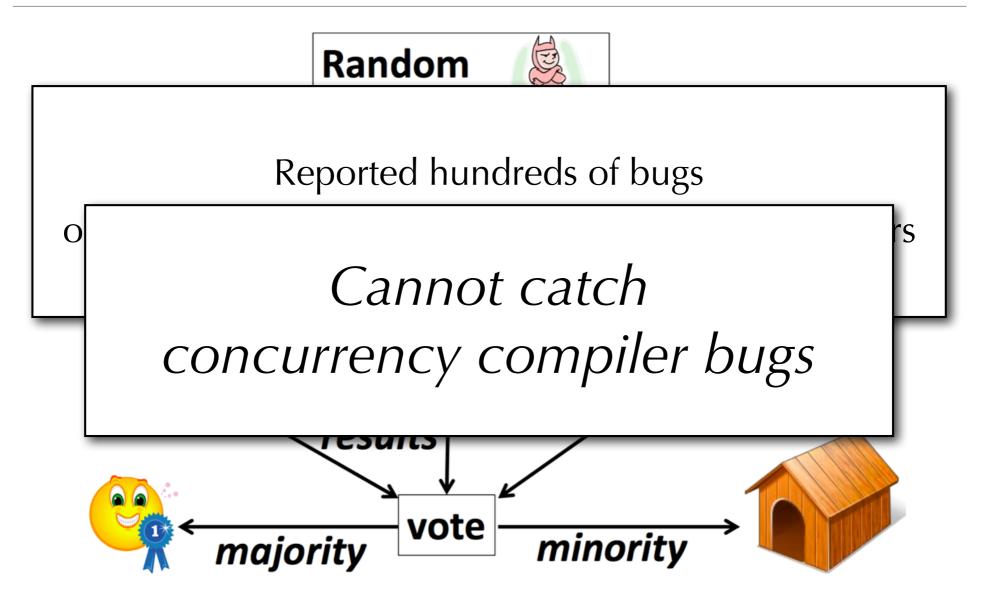
Compiler testing: state of the art

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?

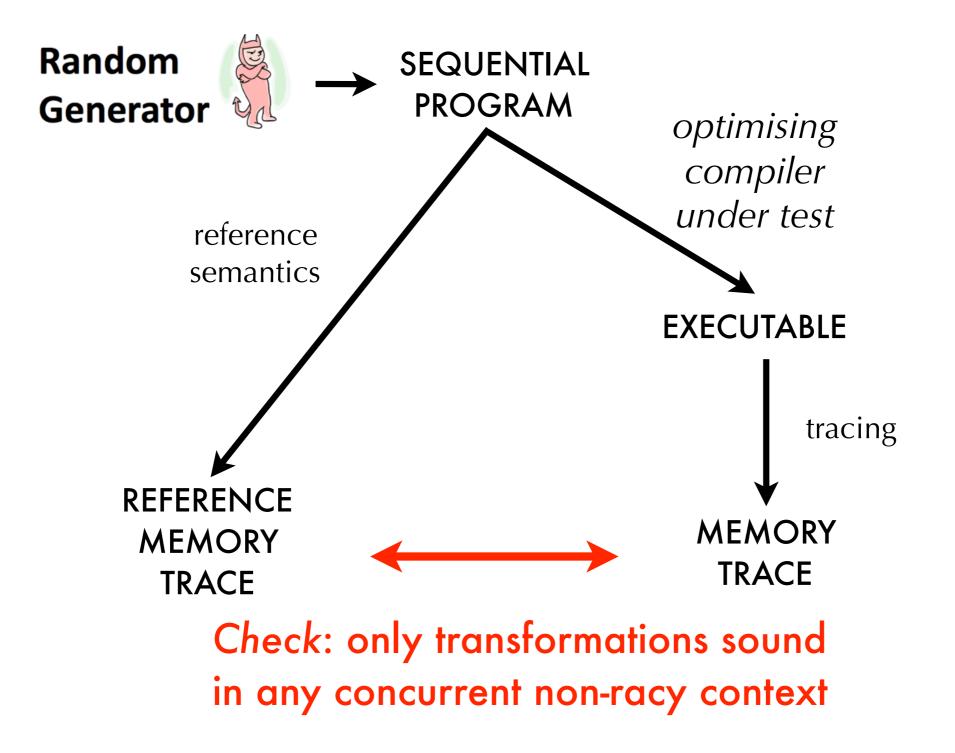
How to capture all the behaviours of concurrent programs?

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

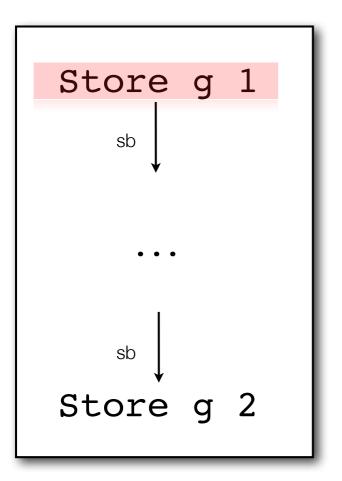
search for transformations of sequential code not sound in an arbitrary non-racy context



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



Elimination of overwritten writes



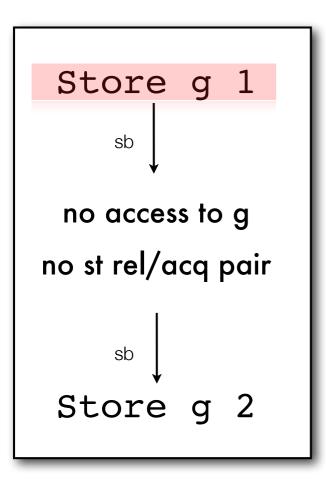
Under which conditions is it correct to eliminate the first store?

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

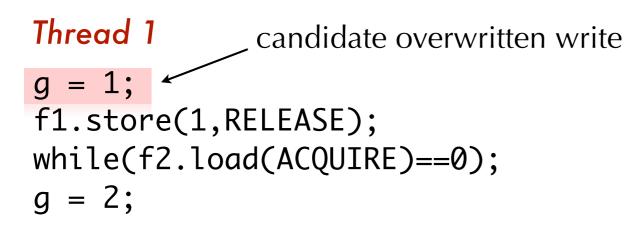
Shared memory

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

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

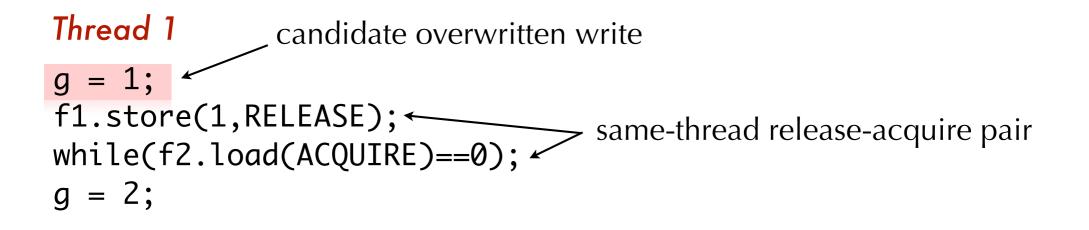
Shared memory

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



Shared memory

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

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);

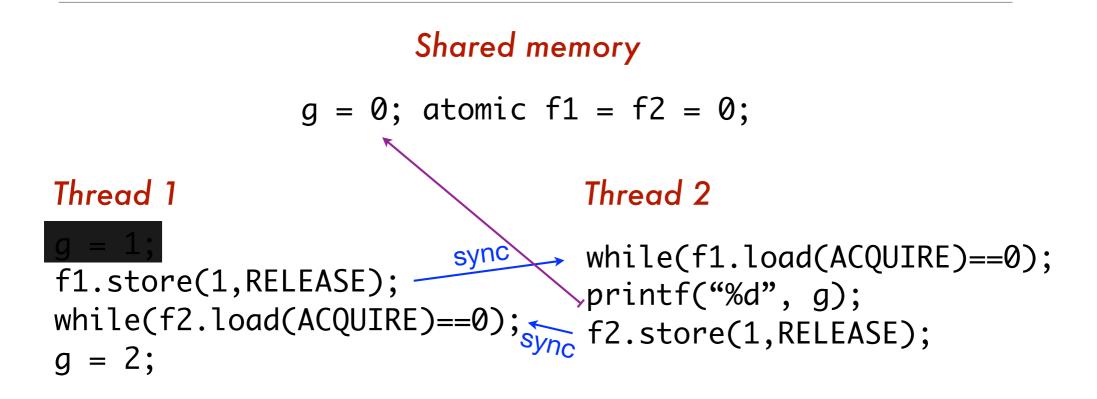
Thread 2 is non-racy

Shared memory

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

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

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); sync while(f2.load(ACQUIRE)==0); g = 2;

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

The soundness condition

Shared memory

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

Thread 1

Thread 2



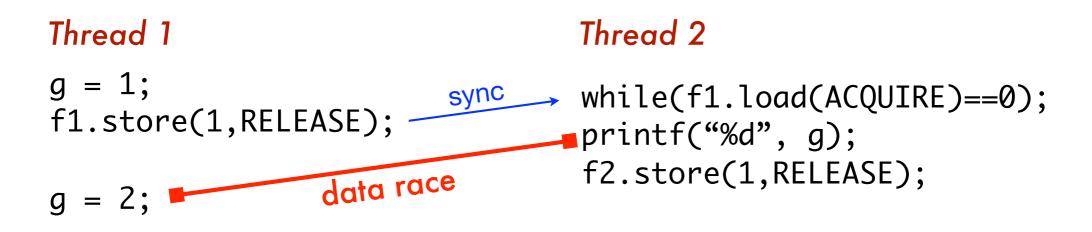
while(f1.load(ACQUIRE)==0);
printf("%d", g);
f2.store(1,RELEASE);

g = 2;

The soundness condition

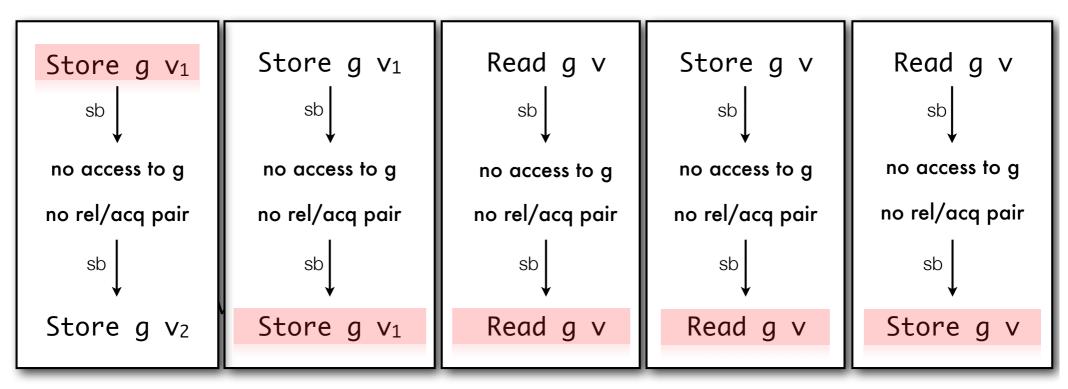
Shared memory

$$g = 0;$$
 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

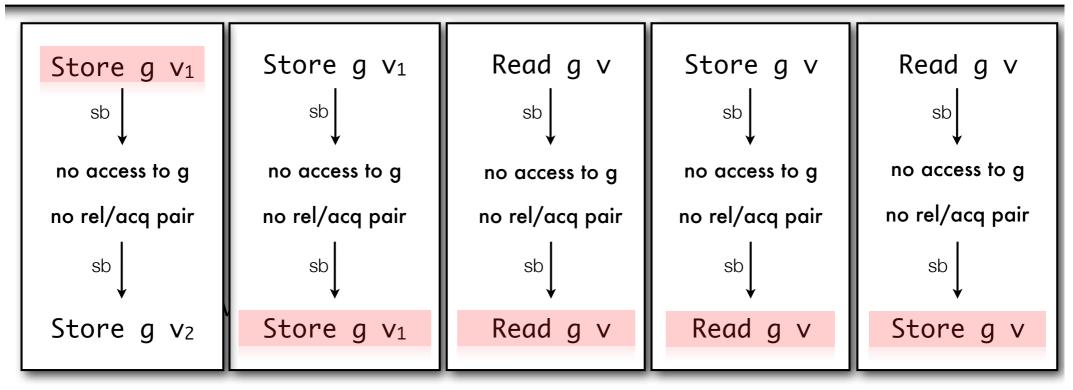


Overwritten-Write Write-after-Write Read-after-Read Read-after-Write Write-after-Read

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.

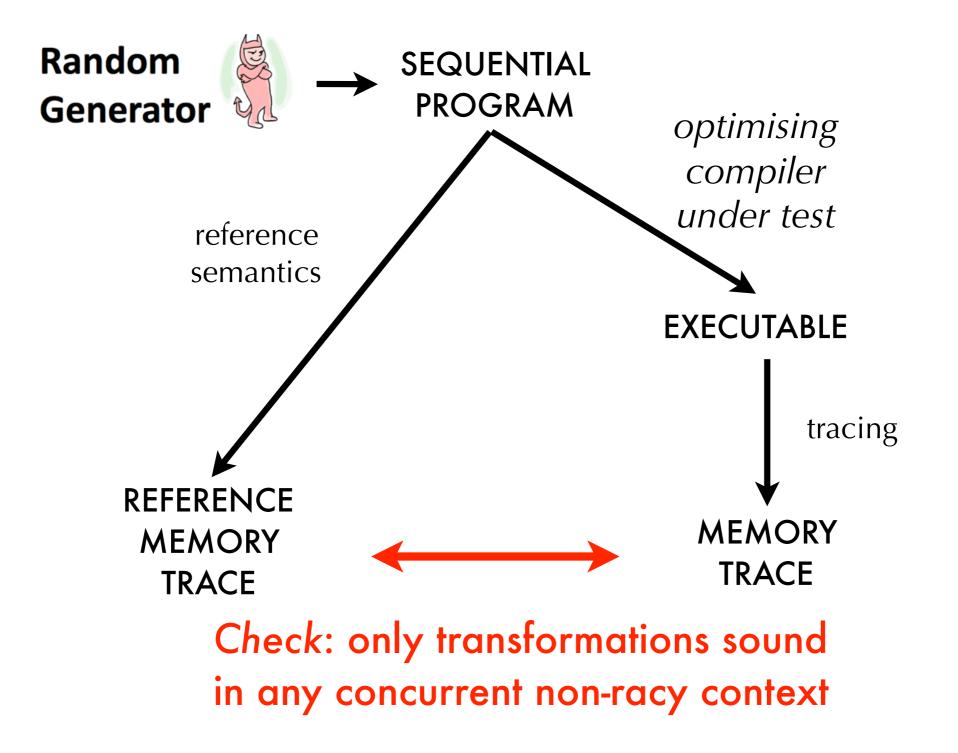


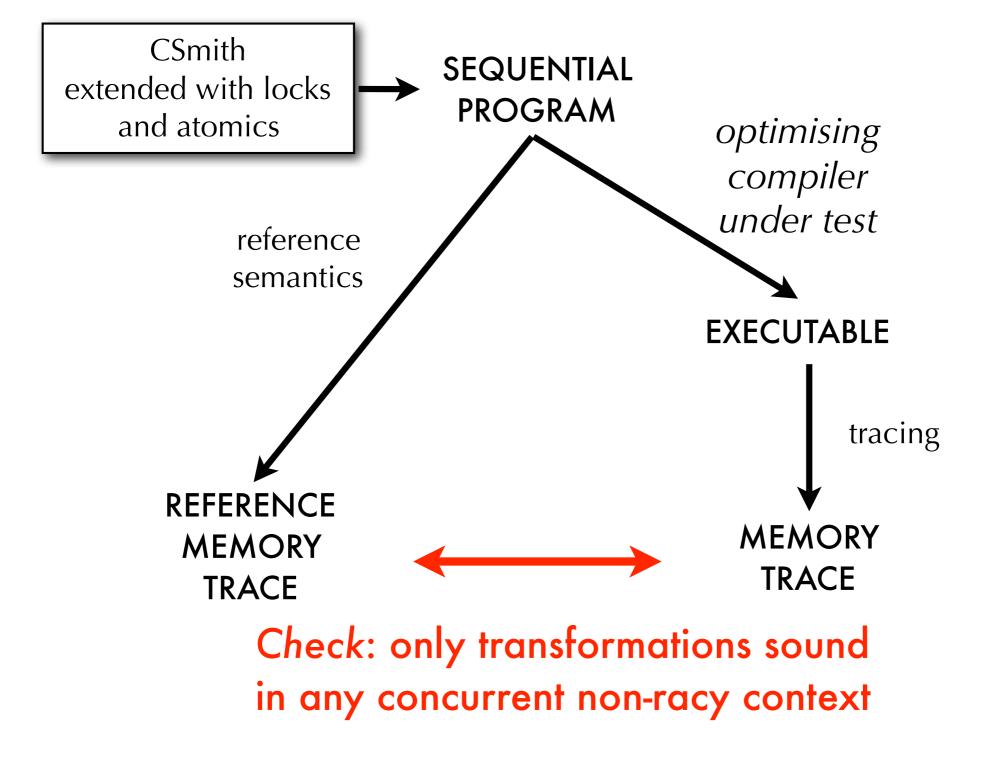
Overwritten-Write Write-after-Write Read-after-Read Read-after-Write Write-after-Read

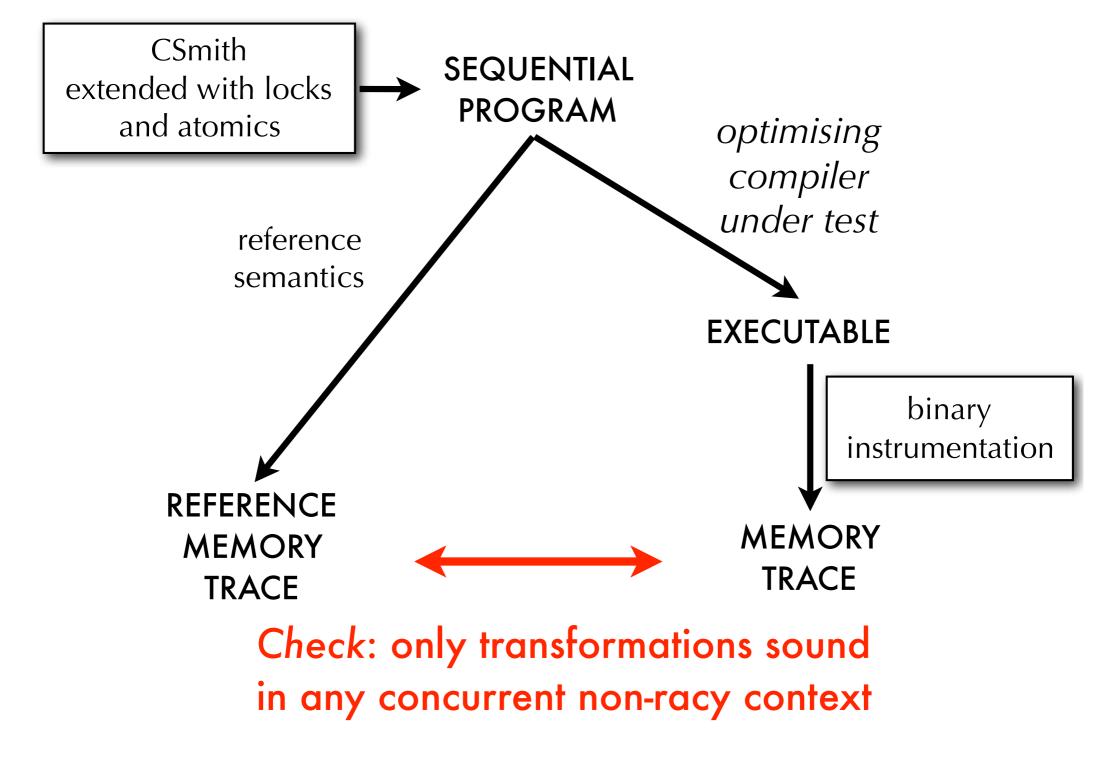
Reads which are not used (via data or control dependencies) to decide a write or synchronisation event are also eliminable (*irrelevant reads*).

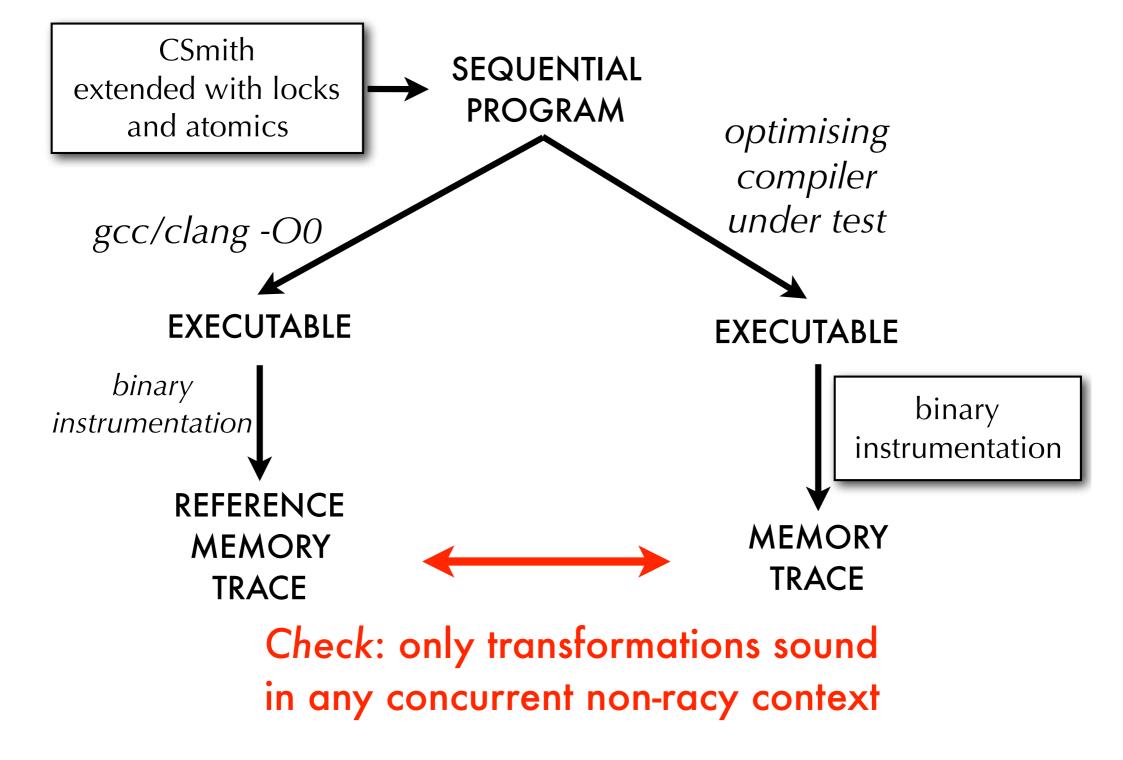
From theory to the Cmmtest tool

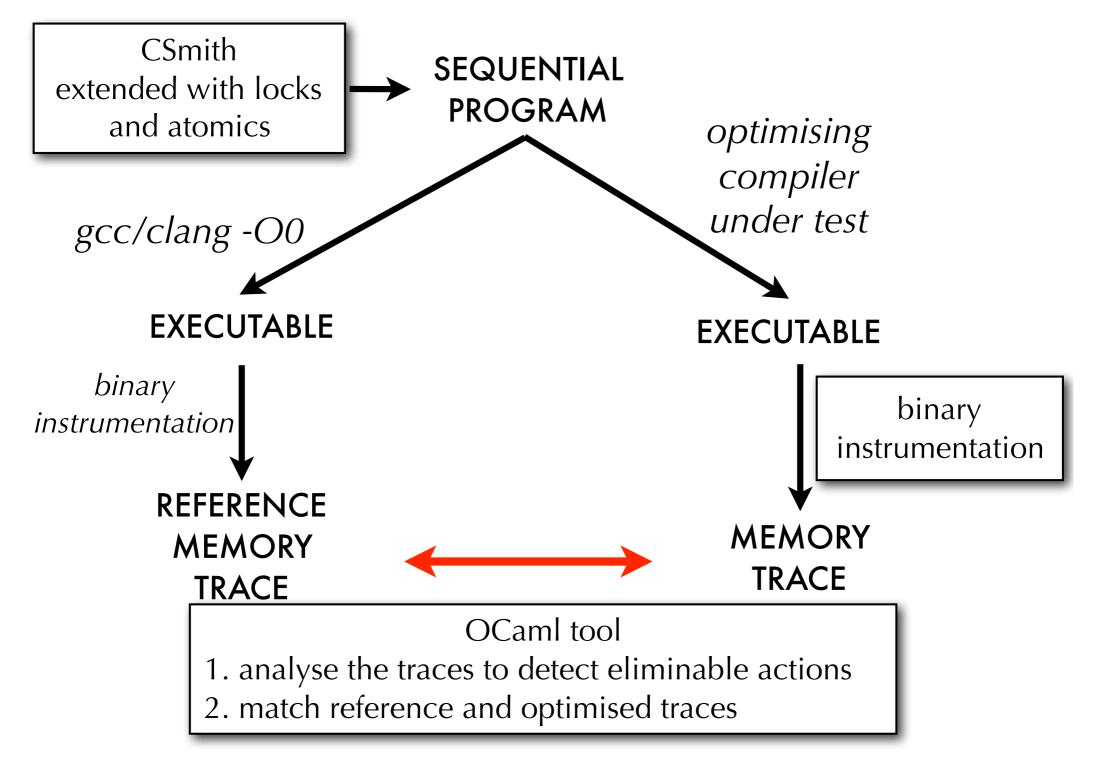












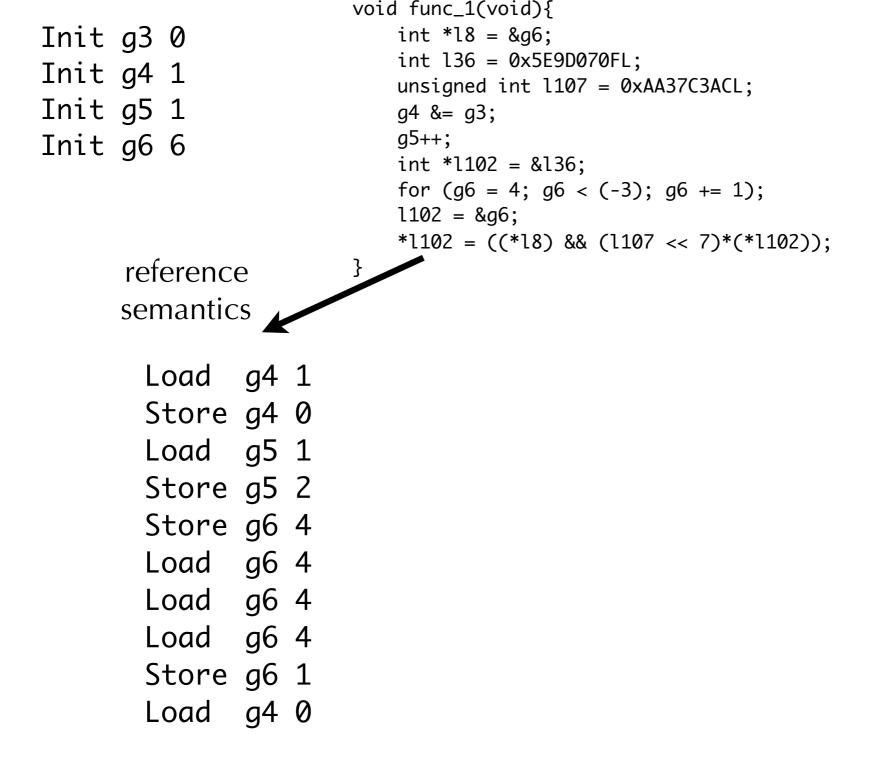
```
const unsigned int g3 = 0UL;
long long g4 = 0x1;
int g6 = 6L;
volatile unsigned int q5 = 1UL;
void func_1(void){
     int *18 = \&g6;
     int 136 = 0 \times 5E9D070FL;
    unsigned int 1107 = 0xAA37C3ACL;
    q4 \&= q3;
    g5++;
     int *1102 = \&136;
     for (g6 = 4; g6 < (-3); g6 += 1);
     1102 = \& g6;
     *1102 = ((*18) && (1107 << 7)*(*1102));
}
```

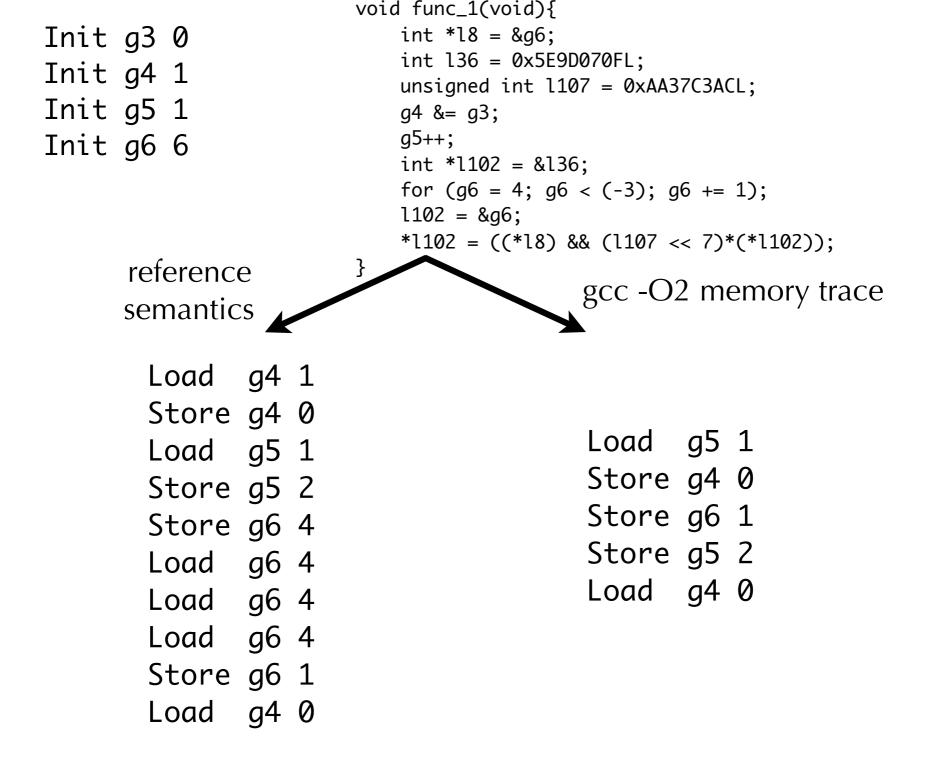
Start with a randomly generated well-defined program

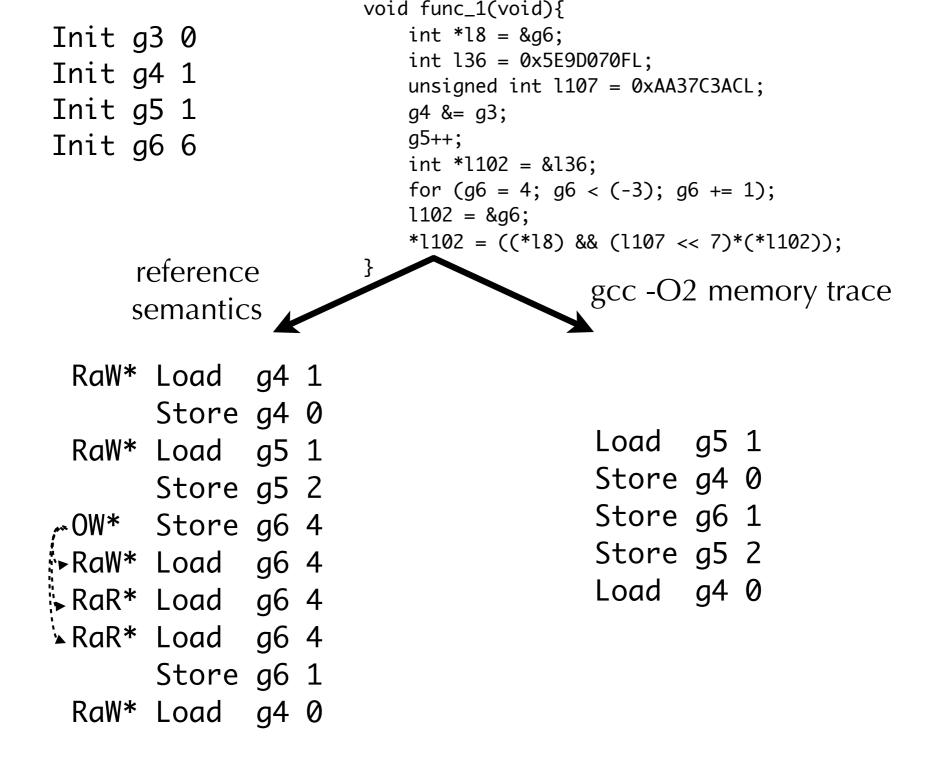
Init g3 0 Init g4 1 Init g5 1 Init g6 6

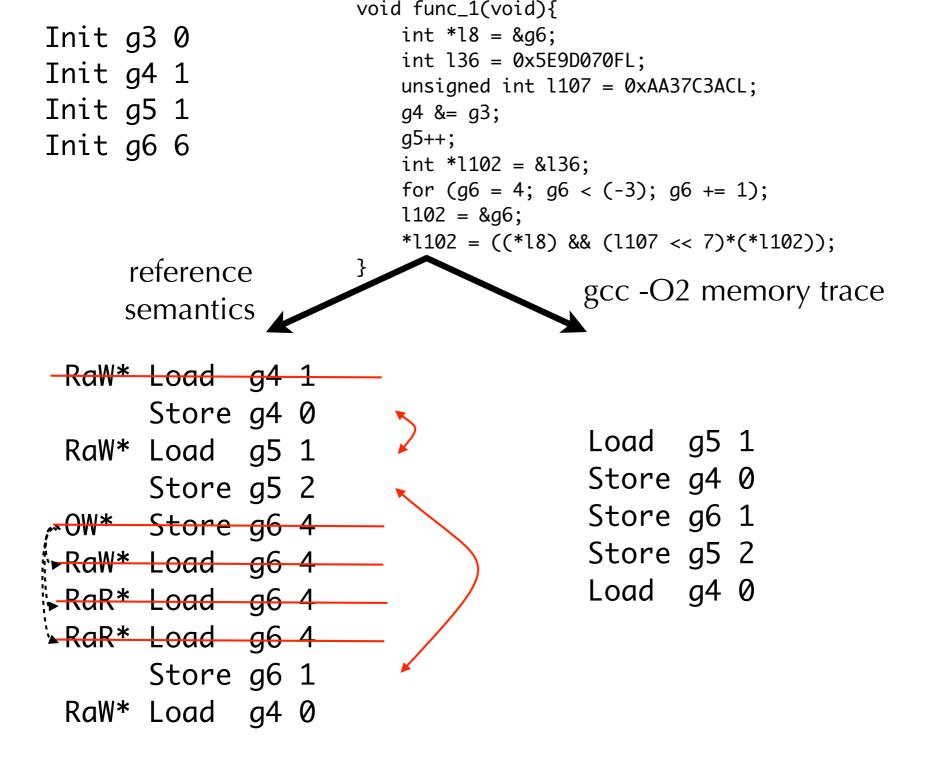
```
void func_1(void){
    int *18 = &g6;
    int 136 = 0 \times 5E9D070FL;
    unsigned int l107 = 0xAA37C3ACL;
    g4 &= g3;
    g5++;
    int *1102 = &136;
    for (g6 = 4; g6 < (-3); g6 += 1);
    1102 = \&g6;
    *1102 = ((*18) && (1107 << 7)*(*1102));
```

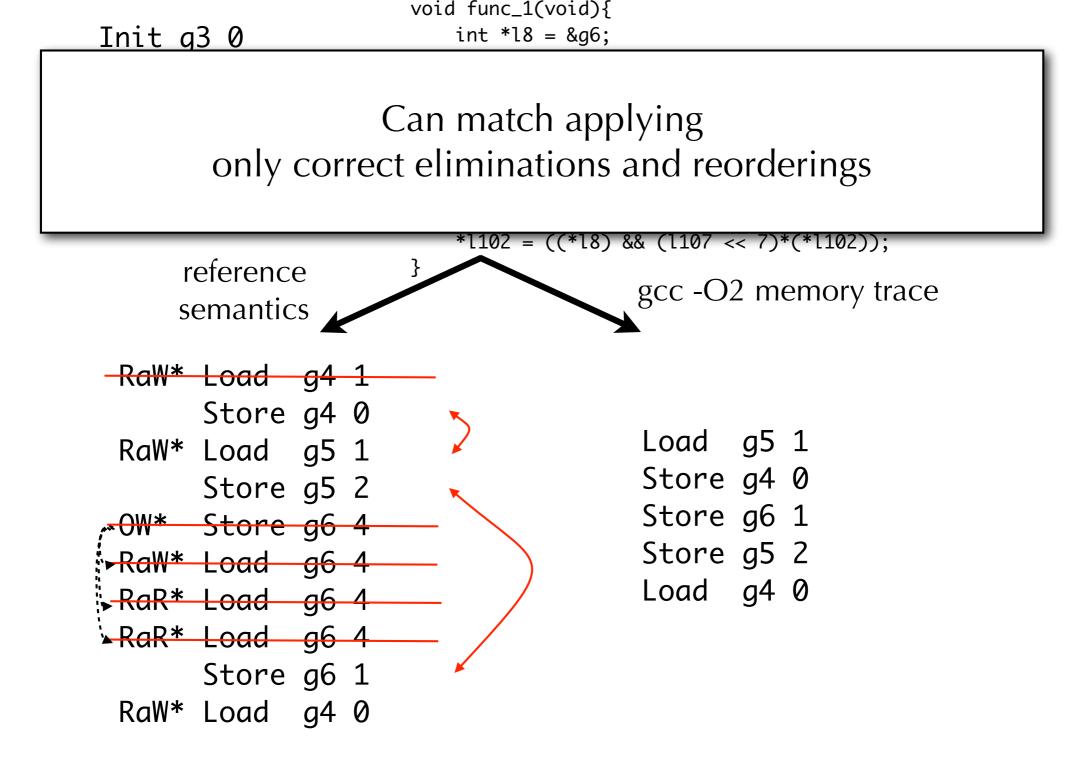
}

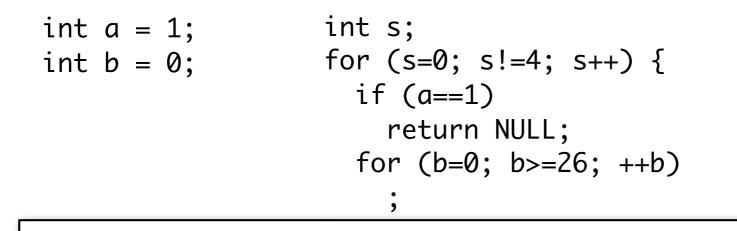






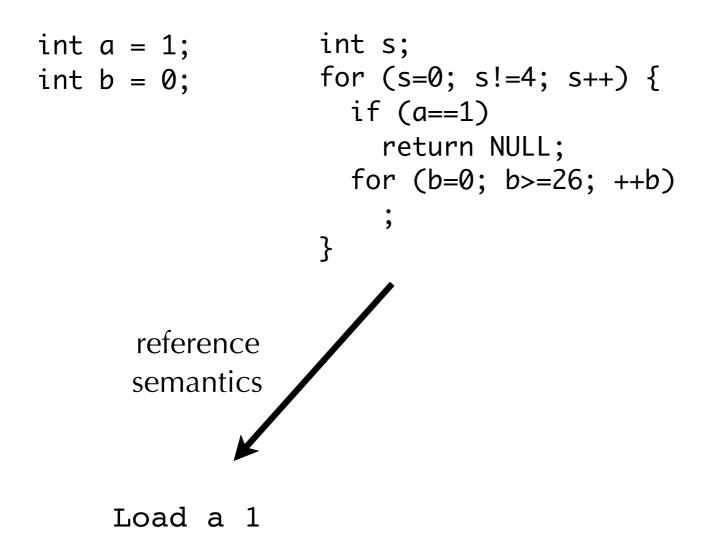


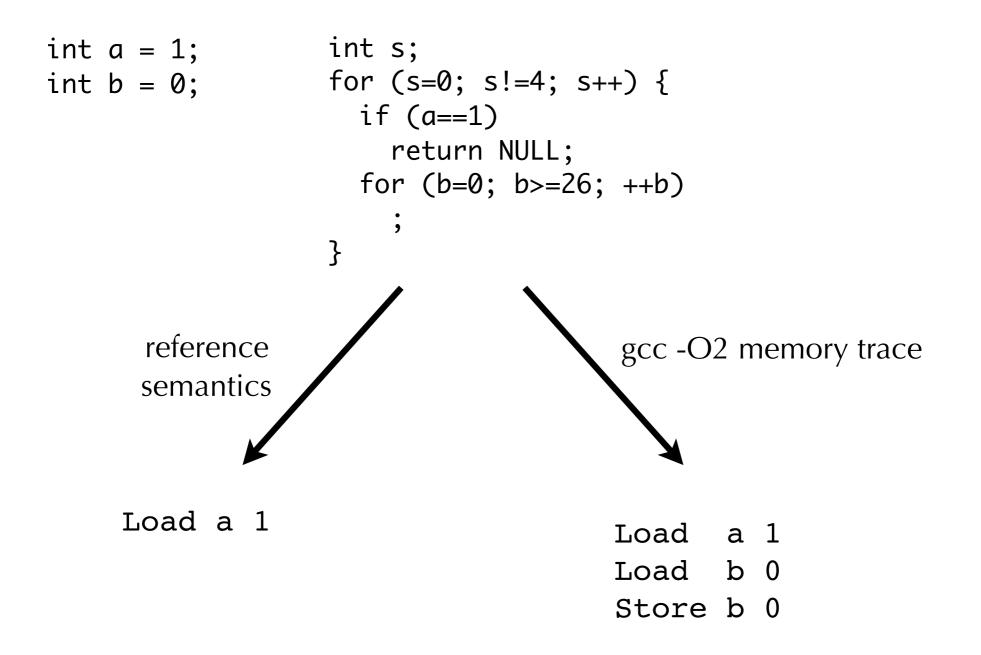


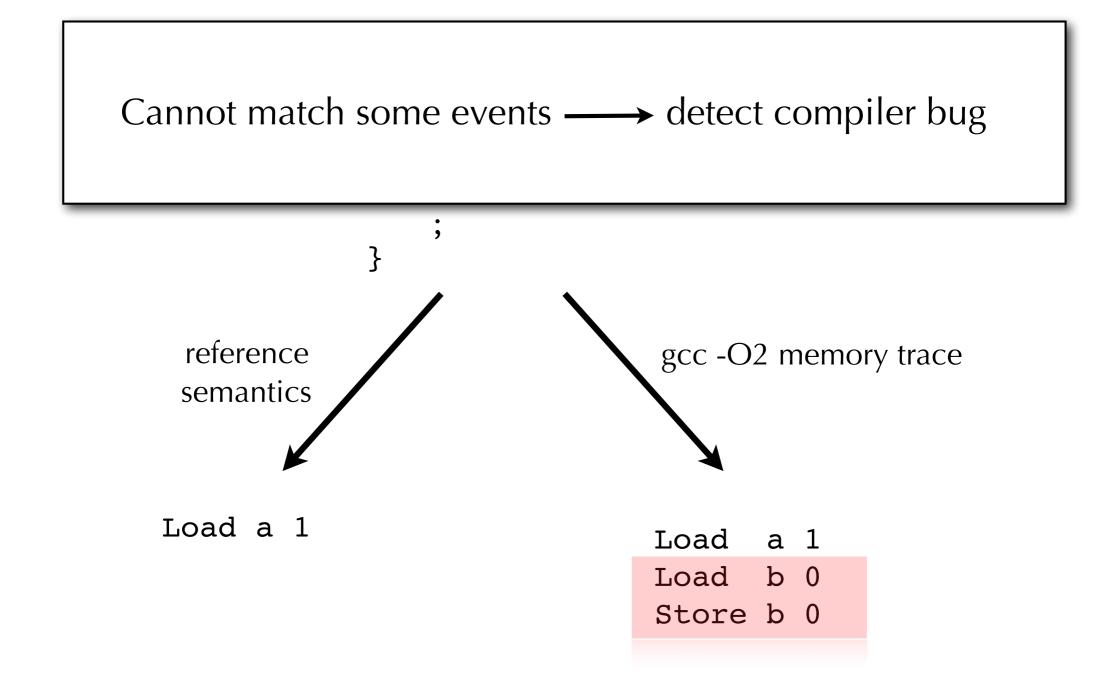


If we focus on the miscompiled initial example...

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







Applications 2013 - 2016



lec2 - 24 January 2019

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.

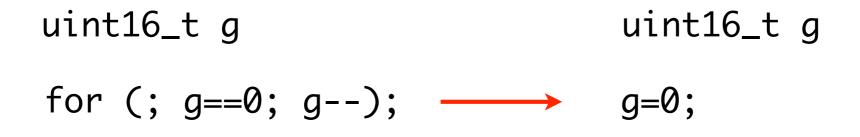
2. Checking compiler invariants

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

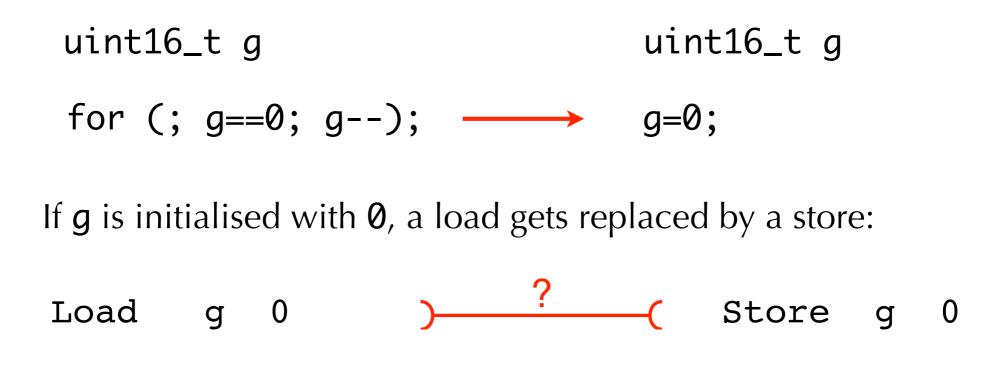
atomic_uint a; int32_t g1, g2;			int main (int, char *[]) { a.load() & a.load (); g2 = g1 != 0;	-
			}	
ALoad	a	0	• Load g1 (0
ALoad	a	0	• • • ALoad a (0
Load	g1	0	o ALoad a (0
Store	g2	0	 Store g2 (0

3. Detecting unexpected behaviours



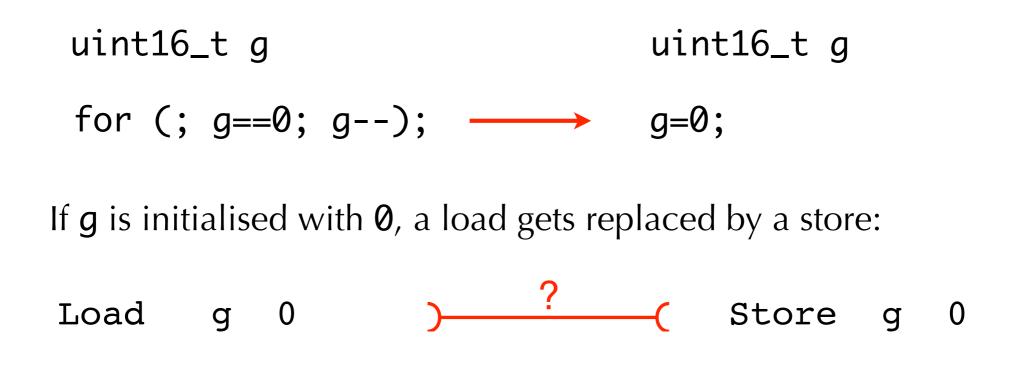
Correct or not?

3. Detecting unexpected behaviours



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



False positives in Thread Sanitizer

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



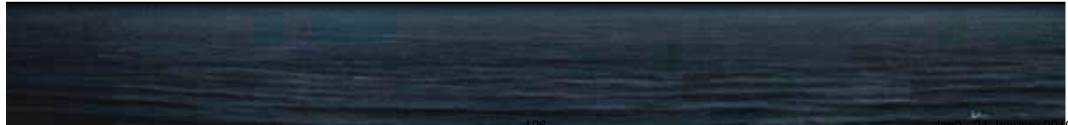
Proving the correctness of mappings for atomics https://www.cl.cam.ac.uk/~pes20/cpp/cpp0xmappings.html

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		
AcqRel Fence:	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



Out of thin-air reads



Memory access synchronisation

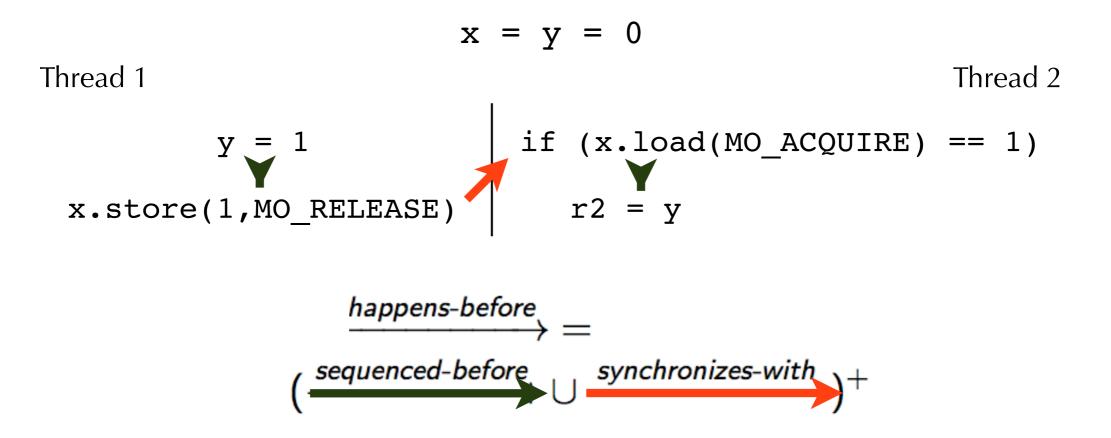
$$x = y = 0$$

Thread 1

$$\mathbf{y} = \mathbf{1}$$

x.store(1,MO_RELEASE)

Memory access synchronisation



Non-atomic loads must return the most recent write in the happens-before order

Understanding MO_RELAXED

$$x = y = 0$$

Thread 1

Thread 2

Understanding MO_RELAXED

Thread 1

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

Thread 2

DATA RACE

Two conflicting accesses not related by happens-before

Understanding MO_RELAXED

$$x = y = 0$$

Thread 1

Thread 2

y.store(1,MO_RELAXED) i

x.store(1,MO_RELAXED)

WELL DEFINED

but $r^2 = 0$ is possible

Understanding MO_RELAXED

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

Thread 1

Thread 2

y.store(1,MO_RELAXED)	<pre>if (x.load(MO_RELAXED) == 1)</pre>
<pre>x.store(1,MO_RELAXED)</pre>	<pre>r2 = y.load(MO_RELAXED)</pre>

Т

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

Intuition

the compiler (or hardware) can reorder independent accesses

$$x = y = 0$$

Thread 1

Thread 2

y.store(1,MO_RELAXED)	<pre>if (x.load(MO_RELAXED) == 1)</pre>
<pre>x.store(1,MO_RELAXED)</pre>	<pre>r2 = y.load(MO_RELAXED)</pre>

Т

Allow a RELAXED load to see any store that:

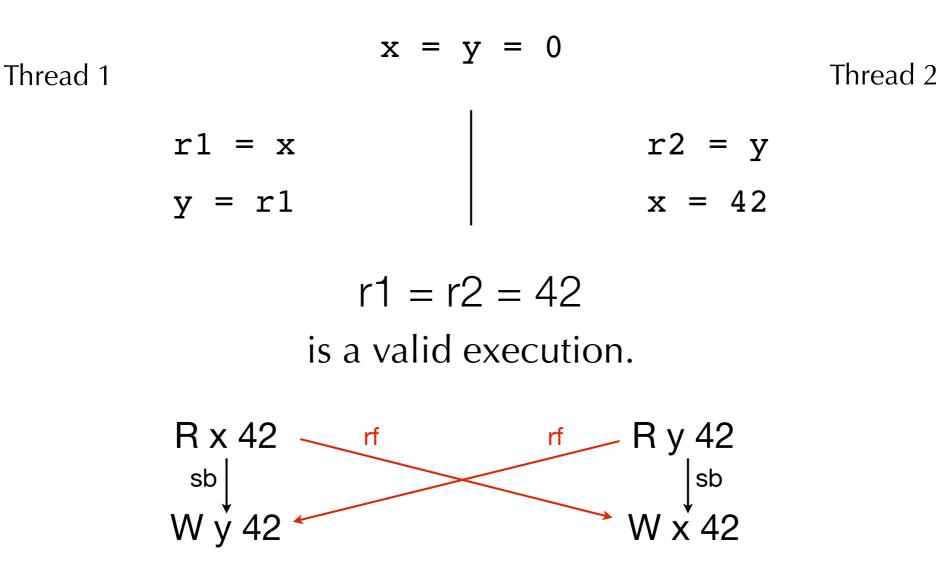
- does not happens-after it
- is not hidden by an intervening store hb-ordered between them

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

Out-of-thin-air

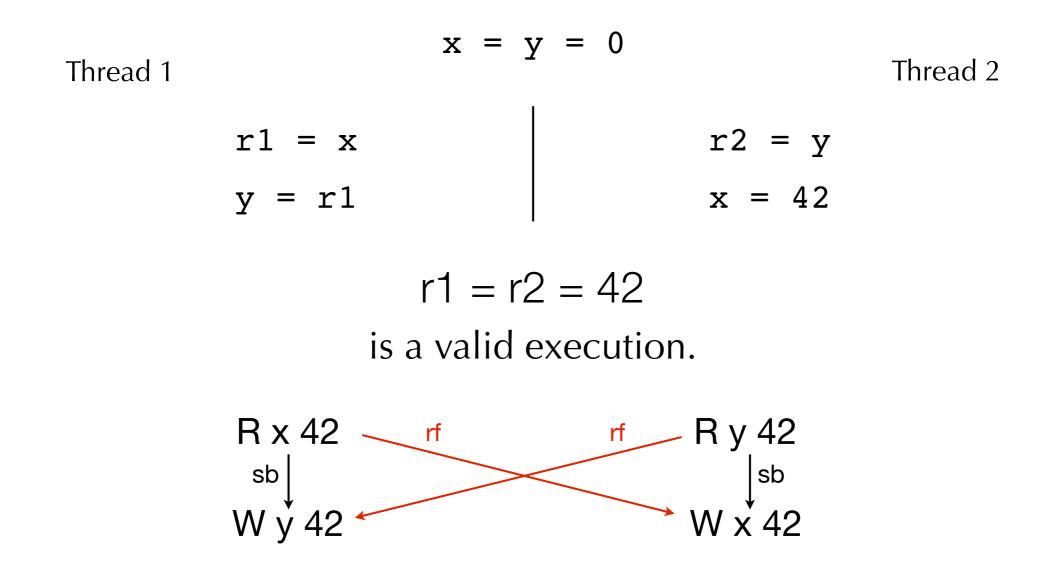
Thread 1 x = y = 0 r1 = x y = r1 r2 = y x = 42Thread 2

Out-of-thin-air



Intuition

the compiler (or hardware) can reorder independent accesses



Out-of-thin-air reads

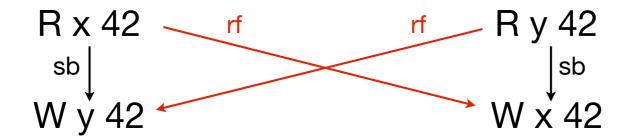
Thread 1 x = y = 0 Thread 2 r1 = x r2 = yy = r1 x = r2

Out-of-thin-air reads

Thread 1

$$r1 = r2 = 42$$

is also an allowed execution

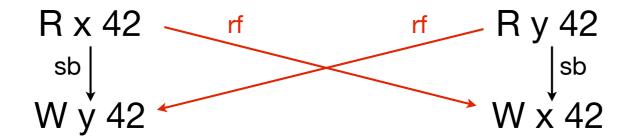


the value 42 appears out-of-thin-air

Thread 1

$$r1 = r2 = 42$$

is also an allowed execution



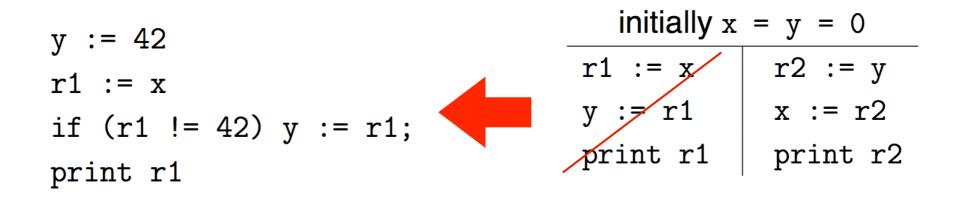
Speculation can justify out-of-thin-air reads

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



Speculation can justify out-of-thin-air reads

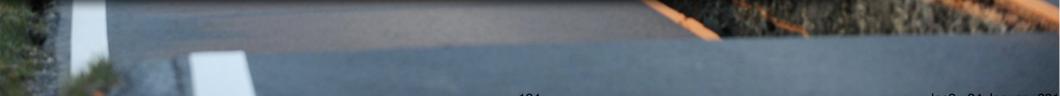
If the compiler states that x is likely to hold 42...



It does not happen in practice... even if it might!



Consequences of out-of-thin-air reads

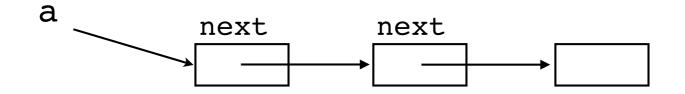




```
struct foo {
   atomic<struct foo *> next;
}
struct foo *a;
```

Thread 1

r1 = a - nextr1 - next = a

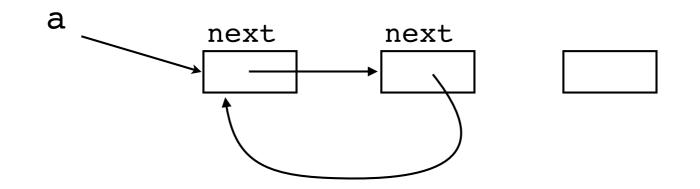




struct foo {
 atomic<struct foo *> next;
}
struct foo *a;

Thread 1

r1 = a - nextr1 - next = a





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

```
Thread 1

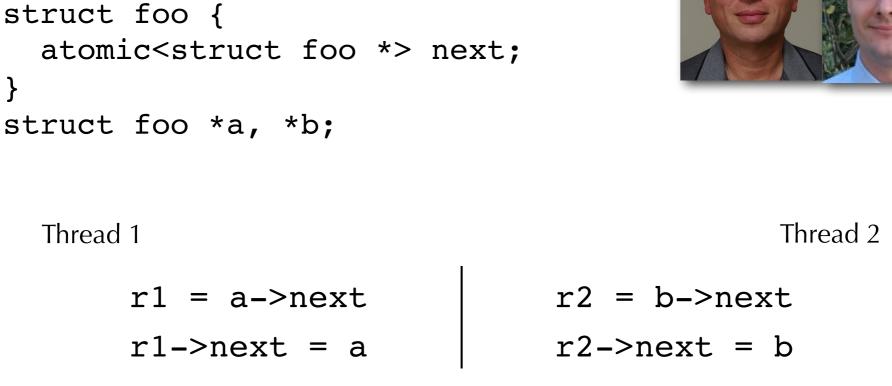
r1 = a - next

r1 = a - next

r2 = b - next

r2 - next = b
```





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

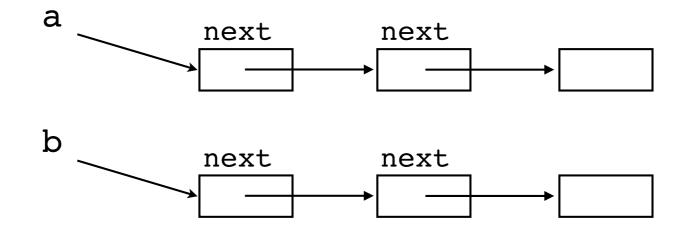


```
struct foo {
   atomic<struct foo *> next;
}
struct foo *a, *b;
```

Thread 1

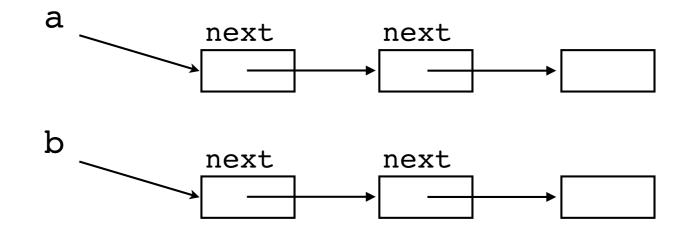
$$r1 = a - next$$

 $r1 - next = a$
Thread 2
 $r2 = b - next$
 $r2 - next = b$



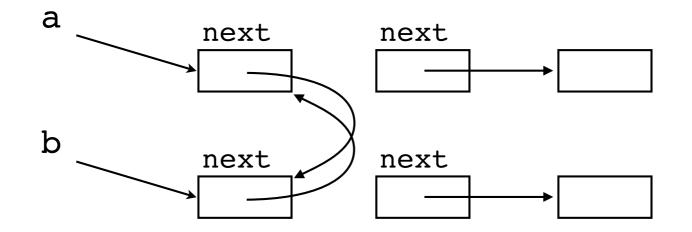
If the compiler speculates r1=b and r2=a, then the store r1->next=a justifies r2=b->next assigning r2=a (and symmetrically to justify r1=b)





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)



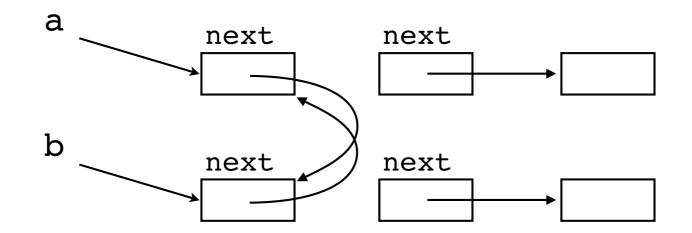


If the compiler speculates r1=b and r2=a, then

the store $r1 \rightarrow next = a$ justifies $r2 = b \rightarrow next$ assigning r2 = a

(and symmetrically to justify r1=b)

Break our basic intuitions about memory and sharing!





Common compiler optimisations are unsound in C11

XINHUA

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

$$x = y = a = 0$$

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

$$a = 1 \land x = y = 0$$

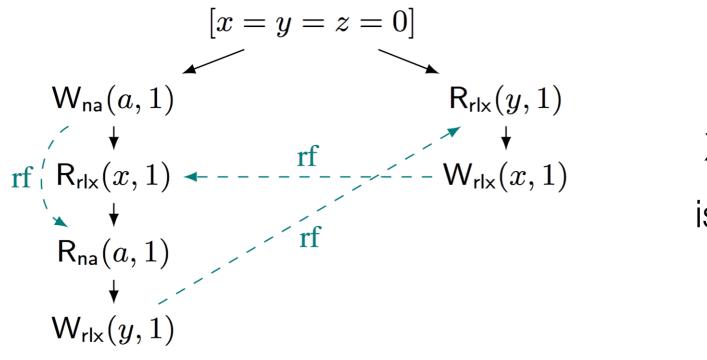
is the only possible final state

$$x = y = a = 0$$

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

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

if



a = 1x = y = 42

is also possible

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

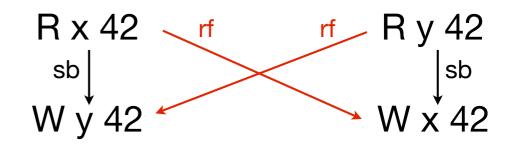
including expression linearisation and roach-motel reorderings



Are there any solutions?

Thread 0	Thread 1	R x 42 _ rf _ R y 42
r1 = x	r2 = y	sb sb sb $W \times 42$
y = r1	x = 42	W y 42 W x 42

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



Thread 0	Thread 1	R x 42 _ rf _ R y 42
r1 = x	r2 = y	sb sb
y = r1	x = 42	W y 42 W x 42

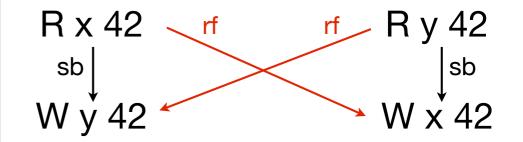
r1 = r2 = 42. Can you spot the difference?

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

Thread 0	Thread 1	R x 42 _ rf _ R y 42
r1 = x	r2 = y	sb sb
y = r1	x = 42	W y 42 W x 42

The "bad" example has a *cycle of dependencies*.

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

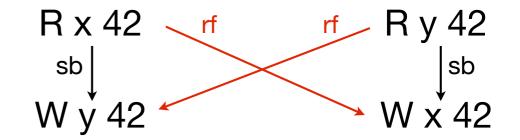


Solution 1.

Prohibit executions with dependency cycles

The "bad" example has a cycle of dependencies.

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



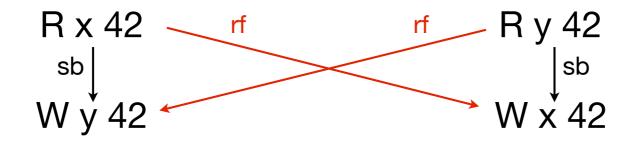
Compiler writers do not want to track all dependencies

Compiler writers do not want to track all dependencies

Does the store to i depend on the load of x?

Solution 2. Brute force

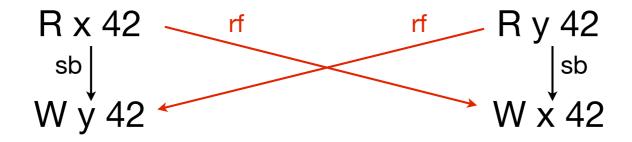
Disallow cycles altogether



 $\mathsf{acyclic}(\mathsf{hb} \cup \{(a,b) \mid rf(b) = a\})$

Allows all source-to-source optimisations (except for r/w reordering on atomics) but expensive on ARM and GPUs

Disallow cycles altogether



 $\mathsf{acyclic}(\mathsf{hb} \cup \{(a,b) \mid rf(b) = a\})$

Solution 3. less brute force

Allow cycles but make this racy by allowing a to read 1

Efficient implementation of atomics on ARM/GPUs but all R/W reorderings are unsound

Allow cycles but make this racy by allowing a to read 1

State of the art

"Implementations should ensure that no "out-of-thin-air" values are computed that circularly depend on their own computation."

Current proposal for C++XX



Goal 1: data-race free programs are sequentially consistent;

Goal 2: all programs satisfy some memory safety requirements;

Goal 3: common compiler optimisations are sound.

Out-of-thin-air

Out-of-thin-air is not so benign for references. Compare:

initially $x = y = 0$		-	initially x = y = null	
r1 := x	r2 := y	and	r1 := x	r2 := y
y := r1	x := r2	and	y := r1	x := r2
print r1	print r2			r2.run()

What should r2.run() call?

If we allow out-of-thin-air, then it could do anything!



Goal 1: data-race free programs are sequentially consistent;

Goal 2: all programs satisfy some memory safety requirements;

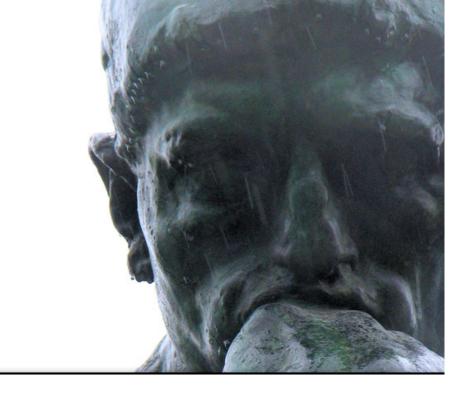
Goal 3: common compiler optimisations are sound.

The model is intricate, and fails to meet goal 3.

An example: should the source program print 1? can the optimised program print 1?

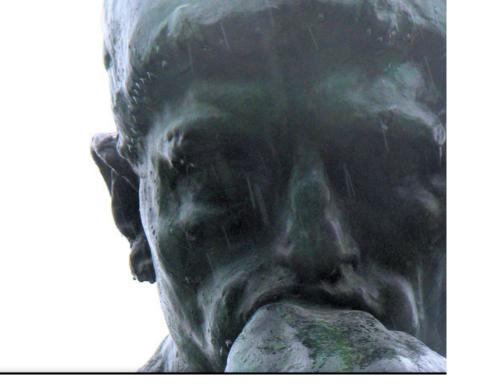
X	= y = 0	lotSpot Optimization $\mathbf{x} = \mathbf{y}$	y = 0
r1 = x y = r1	r2 = y x=(r2==1)?y:1 print r2	r1 = x y = r1	x = 1 r2 = y print r2

Jaroslav Ševčík, David Aspinall, ECOOP 2008



Currently, there is no really satisfactory proposal for the semantics of a general-purpose shared-memory concurrent programming language.





Currently, there is no really satisfactory proposal for the semantics of a general-purpose shared-memory concurrent programming language.

Remarkable and disturbing.







http://www.cl.cam.ac.uk/~pes20/weakmemory/index.html

Starting point:

J. Sevcik

Safe Optimisations for Shared Memory Concurrent Programs

PLDI 2011

H. Bohem

Threads Cannot Be Implemented as a Library

PLDI 2005

Conclusion





In these lectures we have covered the hardware models of two modern computer architectures (x86 and Power/ARM - at least for a large subset of their instruction set).

We have seen how compiler optimisations can also break concurrent programs and the importance of defining the memory model of highlevel programming languages.

We have also introduced some proof methods to reason about concurrency.

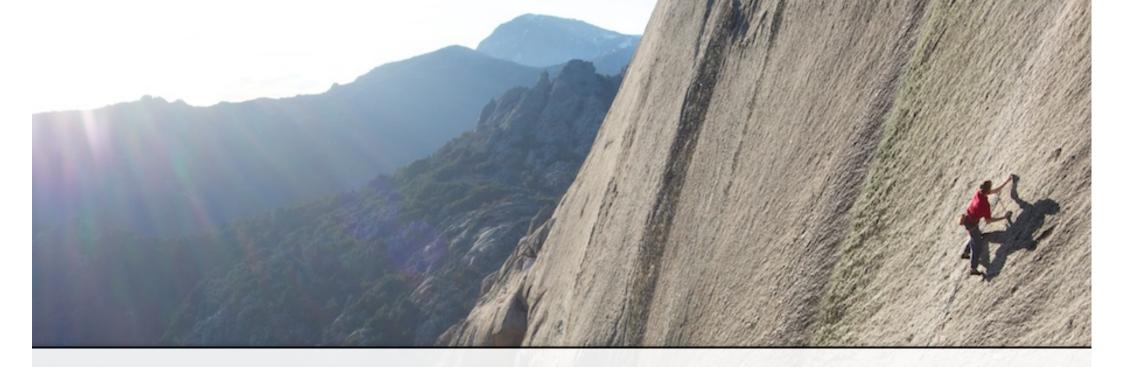
After these lectures, you might have the feeling that multicore programming is a mess and things can't just work.



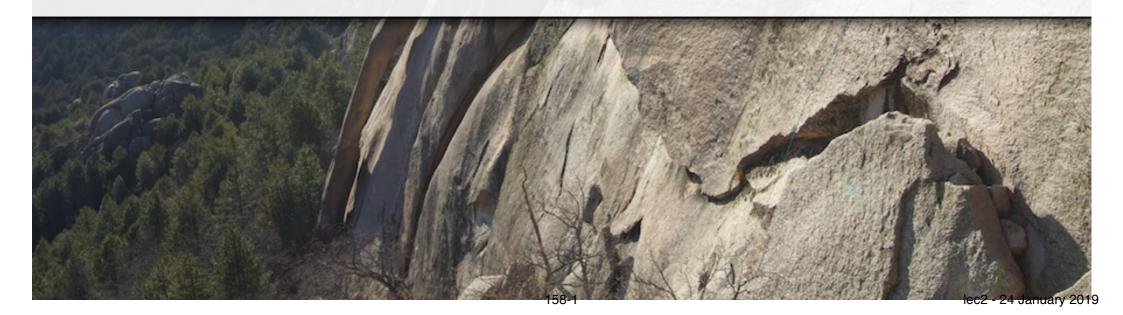
The memory models of modern hardware are better understood.

Programming languages attempt to specify and implement reasonable memory models.

Researchers and programmers are now interested in these problems.



Still, many open problems...





Still, many research opportunities!



lec2 - 24 January 2019