Shared-memory concurrency, out there in the world

Peter Sewell

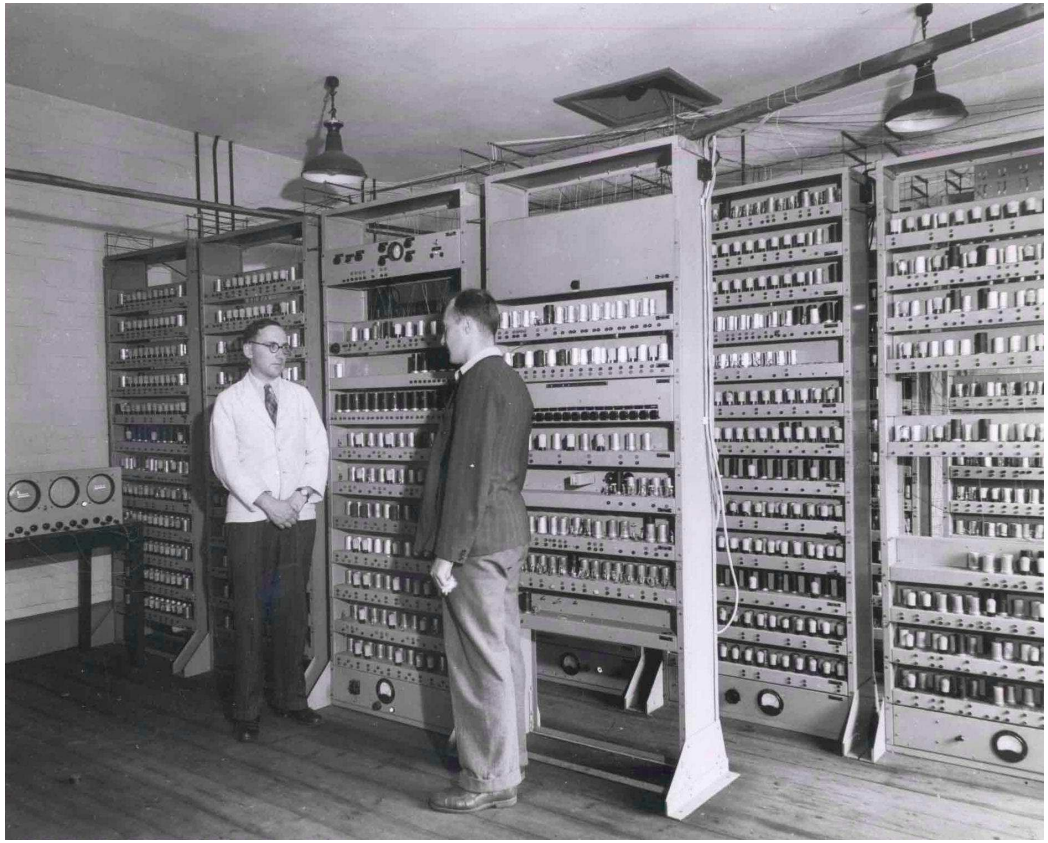
University of Cambridge

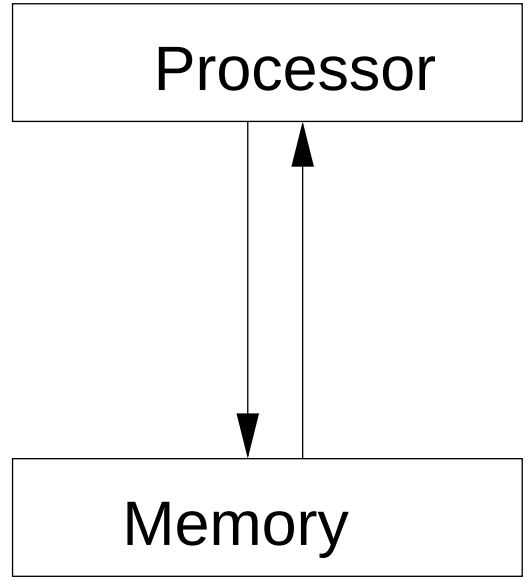
joint work with

Jade Alglave, Mark Batty, Peter Boehm, Suresh Jagannathan, Luc Maranget, Magnus Myreen, Scott Owens, Tom Ridge, Susmit Sarkar, Jaroslav Ševčík, Viktor Vafeiadis, Derek Williams, Francesco Zappa Nardelli

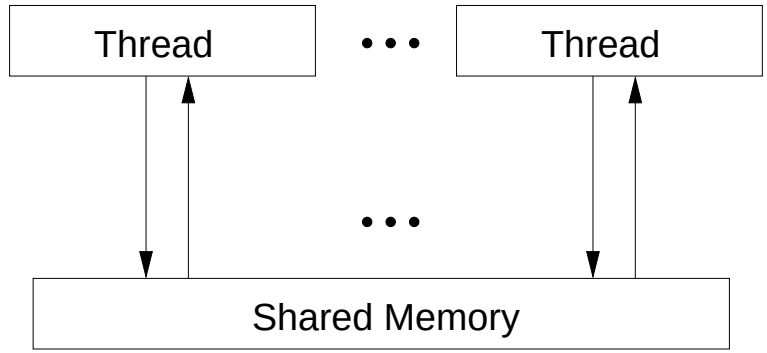
Cambridge, INRIA, Purdue, MPI-SWS, IBM

The Golden Age, 1945–1959





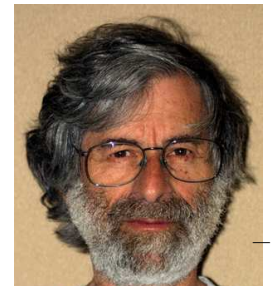
Sequentially Consistent (SC) Shared Memory



Multiple threads, but memory is still an array of values, or a *sequentially consistent* (SC) shared memory:

> *"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".*

Leslie Lamport, 1979



Open Problem: Observational Congruence for SC

Simple case: pure-interleaving, finite executions, parallel contexts, final-state observation

Locations $x, y, z \in L$ Values $v \in \{0, 1\}$ States $s : L \to \{0, 1\}$

Actions $a ::= \mathrm{R}xv \mid \mathrm{W}xv \mid \tau$

Take threads P in some while-language, or receptive LTSs

Take parallel composition $P \mid Q$ to be free interleaving, and the obvious SC semantics for a process with a store $\langle P, s \rangle$.

Define the final states $\mathrm{fs}(P, s_0) = \{s \mid \exists Q.\ \langle P, s_0 \rangle \to^* \langle Q, s \rangle \not\to\}$

Define an observational preorder:

$$P \leqslant_c P' \quad \text{iff} \quad \forall s_0, Q.\ \mathrm{fs}(P|Q, s_0) \subseteq \mathrm{fs}(P'|Q, s_0)$$

Open Problem: Observational Congruence for SC

Problem: give a 'good' characterisation of $\leqslant_c$ or $=_c$.

Some reads are 'irrelevant':

$$\mathrm{R}x0.\mathrm{W}zv + \mathrm{R}x1.\mathrm{W}zv =_c \mathrm{W}zv$$

Some 'write non-stuttering' is unobservable:

$$\mathrm{W}x1.\mathrm{W}x1 + \mathrm{W}x1 =_c \mathrm{W}x1.\mathrm{W}x1$$

Read and write values are interrelated:

$$\mathrm{W}y1 =_c \mathrm{R}y0.\mathrm{W}y1 + \mathrm{R}y1$$

(c.f. Brookes 96, but there a program can atomically check all locations)

Living in an Ideal World

Such a *sequentially consistent* shared memory is taken for granted, by almost all

- programming language semantics
- program logics
- concurrency verification tools
- programmers

False, since 1972

IBM System 370/158MP

And in x86, ARM, POWER, Itanium, Sparc

And in C, C++, Java, ...

And moreover, most of those specs are seriously flawed

This Talk

Mainstream shared-memory concurrency, in multiprocessors and programming languages

...just a flavour, by example

Cuts across CS: hardware, compiler optimisations, concurrent algorithms, programming languages, loose specification and semantics, verification, pragmatic and commercial issues

x86

Intel/AMD/VIA

The Typical TSO Example

At the heart of a mutual exclusion algorithm, e.g. Dekker's, you might find x86 code like this.

Two memory locations x and y, initially 0

Thread 0		Thread 1	
MOV [x]←1	(write x=1)	MOV [y]←1	(write y=1)
MOV EAX←[y]	(read y)	MOV EBX←[x]	(read x)

What final states are allowed?

In SC: What are the possible sequential orders?

The Typical TSO Example

Conclusion:

0,1 and 1,1 and 1,0 can happen, but 0,0 is impossible

The Typical TSO Example

Conclusion:

0,1 and 1,1 and 1,0 can happen, but 0,0 is impossible

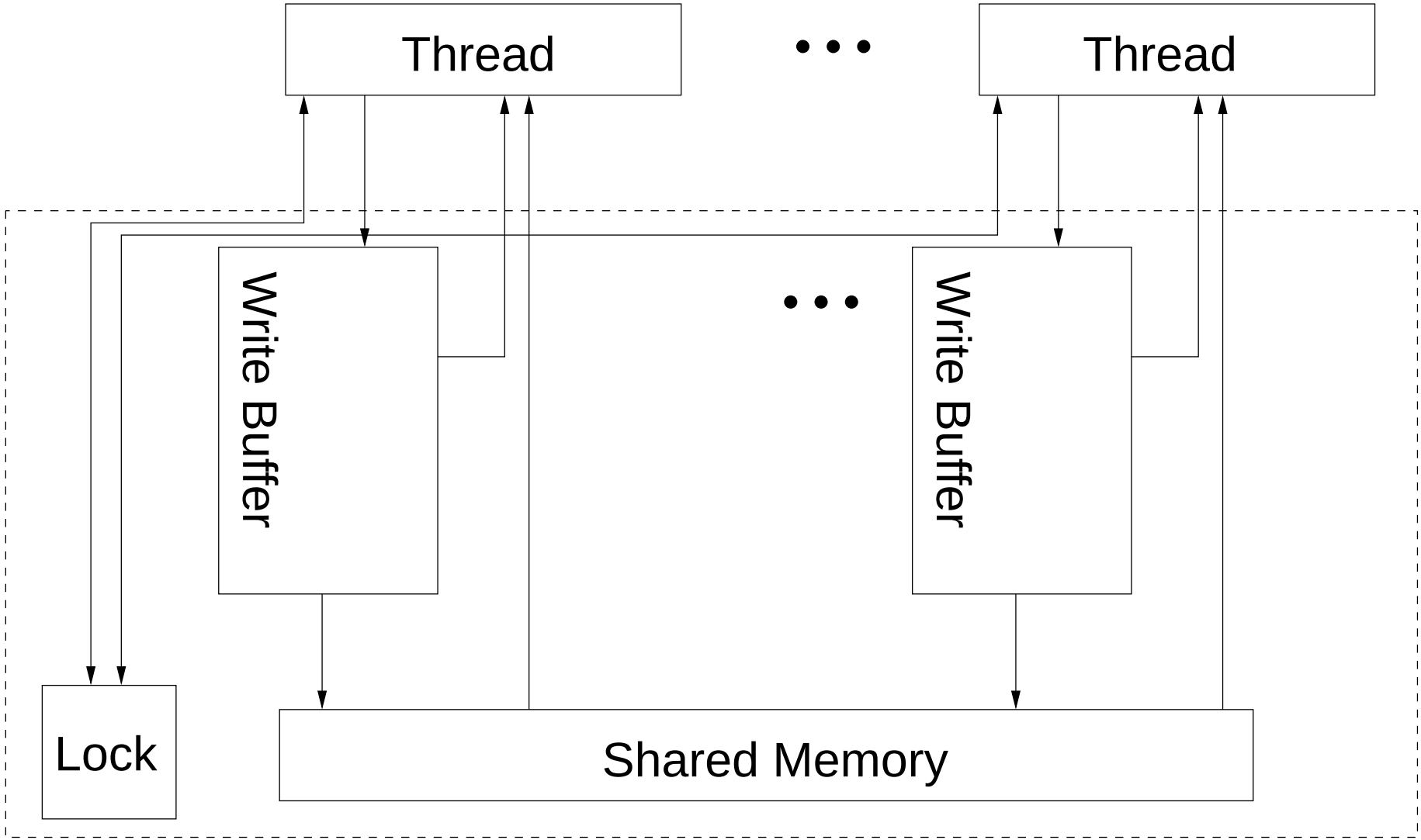
In fact, in the real world, experimentally:

we observe 0,0 every 630/100 000 runs

(on an Intel Core Duo x86)

(and so Dekker's algorithm will fail)

Our x86-TSO model



Our x86-TSO model

- Unambiguous (in HOL4)
- Sound w.r.t. experimentally observable behaviour
- Easy to understand (abstract machine)
- Consistent with what we know of vendors intentions
- Consistent with expert-programmer reasoning

Reason about it:

- Equivalence between abstract machine and axiomatic model
- TRF theory, correctness of locks
- correctness of compilation *to* x86-TSO

Architecture?

Hardware manufacturers document *architectures*:

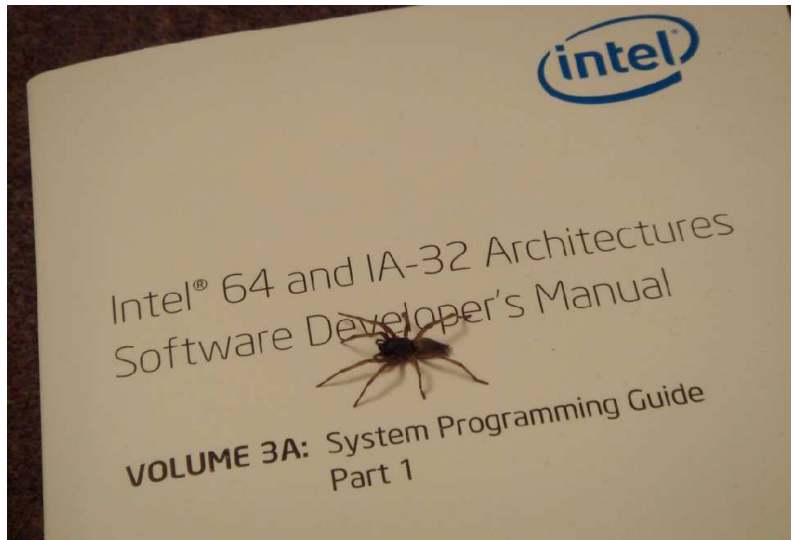
Intel 64 and IA-32 Architectures Software Developer's Manual

AMD64 Architecture Programmer's Manual

Power ISA specification

ARM Architecture Reference Manual

- loose specifications,
- claimed to cover a wide range of past and future processor implementations.



Fundamental Problem

Architecture texts: *informal prose* attempts at subtle loose specifications

Fundamental problem: prose specifications cannot be used

- to *test programs against*, or
- to *test processor implementations*, or
- to *prove* properties of either, or even
- to *communicate precisely*.

In a real sense, the architectures don't *exist*

POWER and ARM

IBM PowerG5, Power 5, Power 6, Power 7
ARM Cortex A9,...

Ubiquitous Multiprocessors, 2010–


Home > Mobile & Wireless > News

Mobile & Wireles

ARM

ARM: £99 dual-core budget smartphone out this year

12Mp+ camera superphones only two years away

By Rosemary Hattersley | PC Advisor | Published 16:27, 15 February 11


1.2GHz Dual-Core ARM CPU Now Certified For Blazing Through Android

Tuesday, February 16, 2010 - by Shawn Oliver


Gadgets

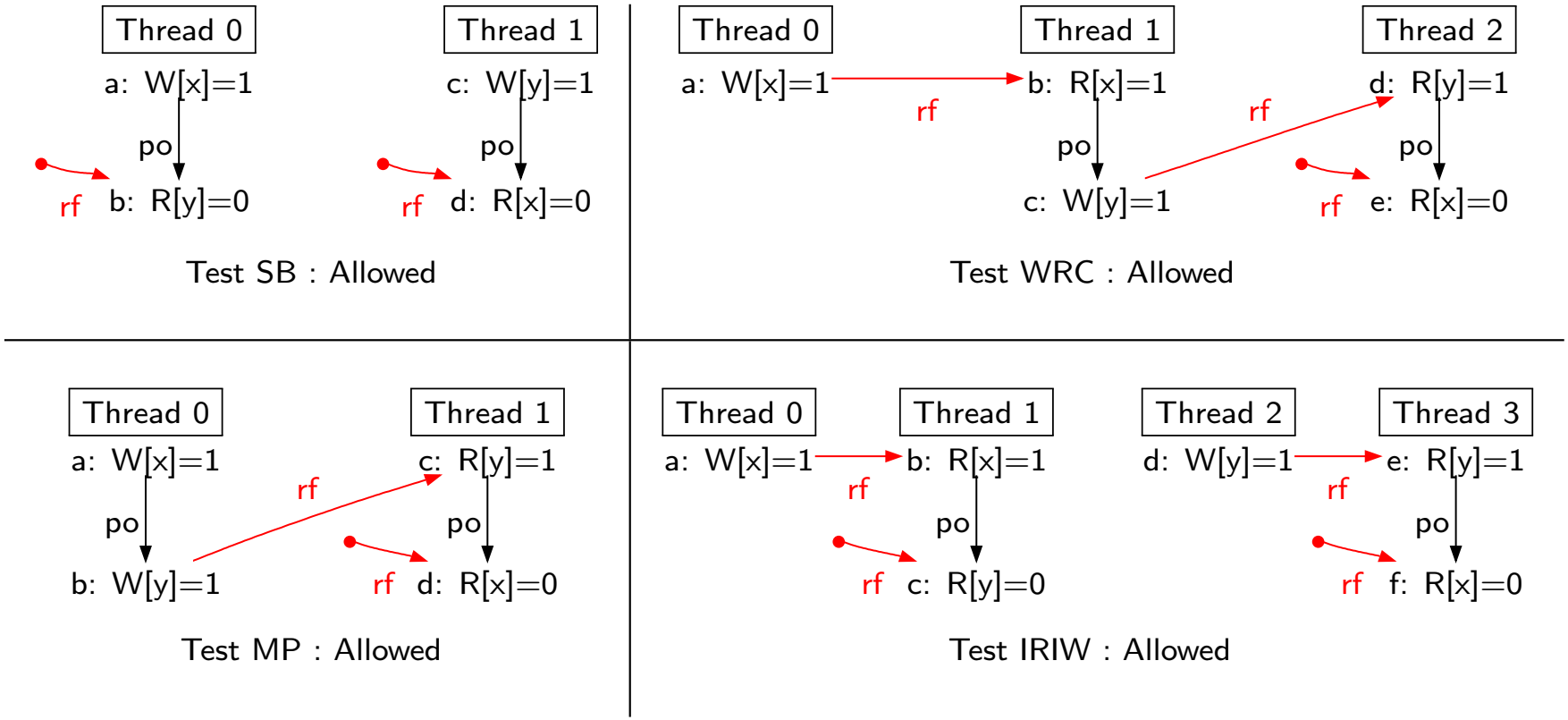
Qualcomm Unleashes Dual and Quad-core 2.5GHz Mobile ARM SoCs

Jason Mick (Blog) - February 14, 2011 10:01 AM

Better performance/watt

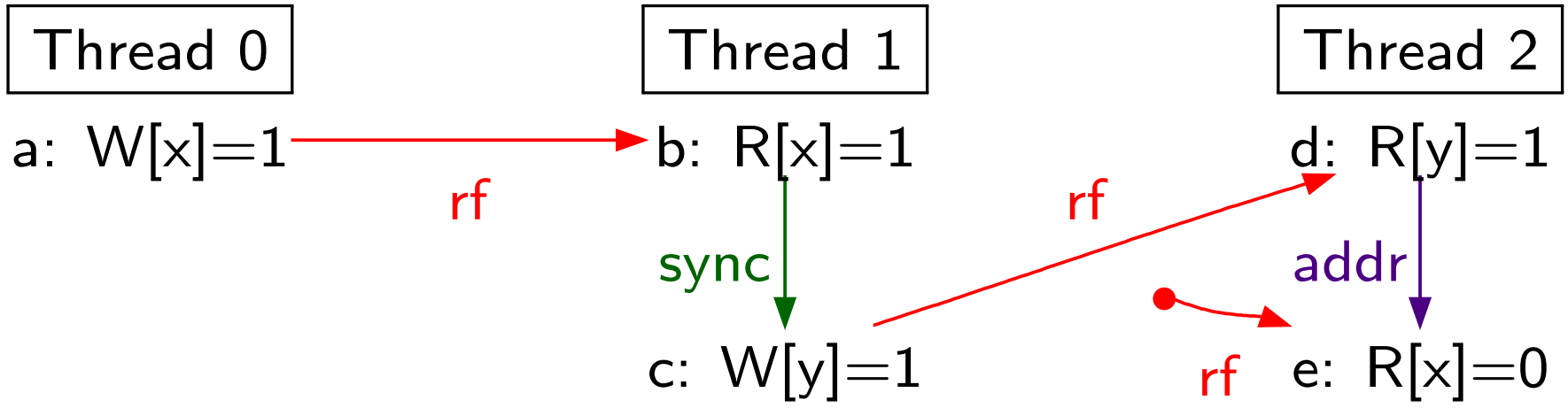
More Relaxed than TSO

(to scale, e.g. to 1024 h/w threads, and for power efficiency)



Use dependencies and various barriers to enforce ordering.

Example: WRC+sync+addr



Test WRC+sync+addr : Forbidden

(and many more subtle issues)

Our work

- much experimental testing of actual h/w
- much discussion with an IBM Power designer
- automatic test generation
- building model(s)
- model exploration tools

Model is *abstract microarchitecture*

Explains all observed behaviour.

Matches intended architecture (intentionally looser than current h/w).

Java

Relaxed Memory from Compiler Optimisations

Example: In x86-TSO, message passing should work as expected:

Thread 1	Thread 2
`data = 1`	
`ready = 1`	`if (ready == 1)`
	`  print data`

In TSO, the program should only print `1`.

Relaxed Memory from Compiler Optimisations

Thread 1	Thread 2
`data = 1`	`int r1 = data`
`ready = 1`	`if (ready == 1)`
	`  print data`

In TSO, the program should only print `1`.

Regardless of other reads.

Relaxed Memory from Compiler Optimisations

Thread 1	Thread 2
`data = 1`	`int r1 = data`
`ready = 1`	`if (ready == 1)`
	`  print data`

In TSO, the program should only print `1`.

But common subexpression elimination (as in `gcc -O1` and `CompCert`) will rewrite

`print data` $\Longrightarrow$ `print r1`

Relaxed Memory from Compiler Optimisations

Thread 1	Thread 2
`data = 1`	`int r1 = data`
`ready = 1`	`if (ready == 1)`
	`  print r1`

In TSO, the program should only print `1`.

But common subexpression elimination (as in `gcc -O1` and `CompCert`) will rewrite

`print data` $\Longrightarrow$ `print r1`

So the compiled program can print `0`!

Broken Specs Again

C/C++ with Posix threads: unclear

JMM original: Broken (Pugh,...)

JMM 2005: Broken (Cenciarelli, Ševčík & Aspinall)

C++0x and C1X

DRF+catch-fire semantics

1. a program that has no races in any SC execution is guaranteed to behave as if running in an SC semantics
2. other programs can behave in any way at all

Promoted by Hans Boehm & Sarita Adve 

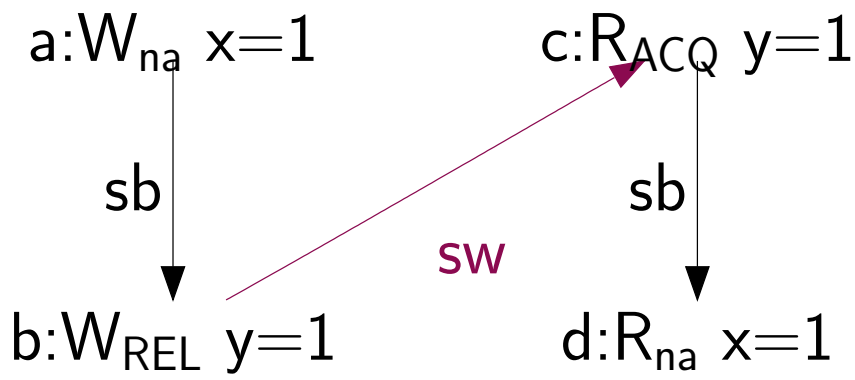
At the heart of proposed C++0x and C1X standards

Permits (pretty) arbitrary compiler optimisation and hardware reordering *between* synchronisation operations

Downside: how do you know code is race-free? And concurrent algorithms are often not.

The release-acquire idiom

```
// sender                          | // receiver
x = ...                            | while (0 == y.load(mo_acquire));
y.store(1,mo_release);             | r = x;
```



More Details

Establishing precise and usable h/w models:

- x86-TSO (CACM, Owens, Sarkar, Sewell, Zappa Nardelli, Myreen)
- Power/ARM (PLDI 2011, Sarkar, Sewell, Alglave, Maranget, Williams)

Establishing precise and usable language models:

- C++0x/C1X (POPL 2011, Batty, Owens, Sarkar, Sewell, Weber)

Reasoning about concurrent code:

- x86-TSO TRF theory, locks etc. (ECOOP 2010, Owens)

Verified compilation:

- CompCertTSO: from ClightTSO to x86-TSO
  (POPL 2011, Ševčík, Vafeiadis, Zappa Nardelli, Jagannathan, Sewell)
- from C++0x executions to x86-TSO executions
  (in above, Owens)
- soundness of optimisations in DRF (PLDI 2011, Ševčík)

Stepping back

Technical abstraction-design challenge:
balance usability from above vs implementability from below

(in some cases still don't know good solutions, e.g. JMM)

Loose specification really bites

Deliberate lack of clarity — loose specification by vague specification

Subtle concurrent behaviour — prose specs not up to it

Challenge for Concurrency Theory: Effective Reasoning

The End

Thanks to:
Jade Alglave, Mark Batty, Peter Boehm, Suresh Jagannathan, Luc Maranget, Magnus Myreen, Scott Owens, Tom Ridge, Susmit Sarkar, Jaroslav Ševčík, Viktor Vafeiadis, Derek Williams, Francesco Zappa Nardelli