Why are computers so @#!*, and what can we do about it?

Peter Sewell

University of Cambridge

August 2014

EMF, somewhere near Bletchley

Things I Know About Computers

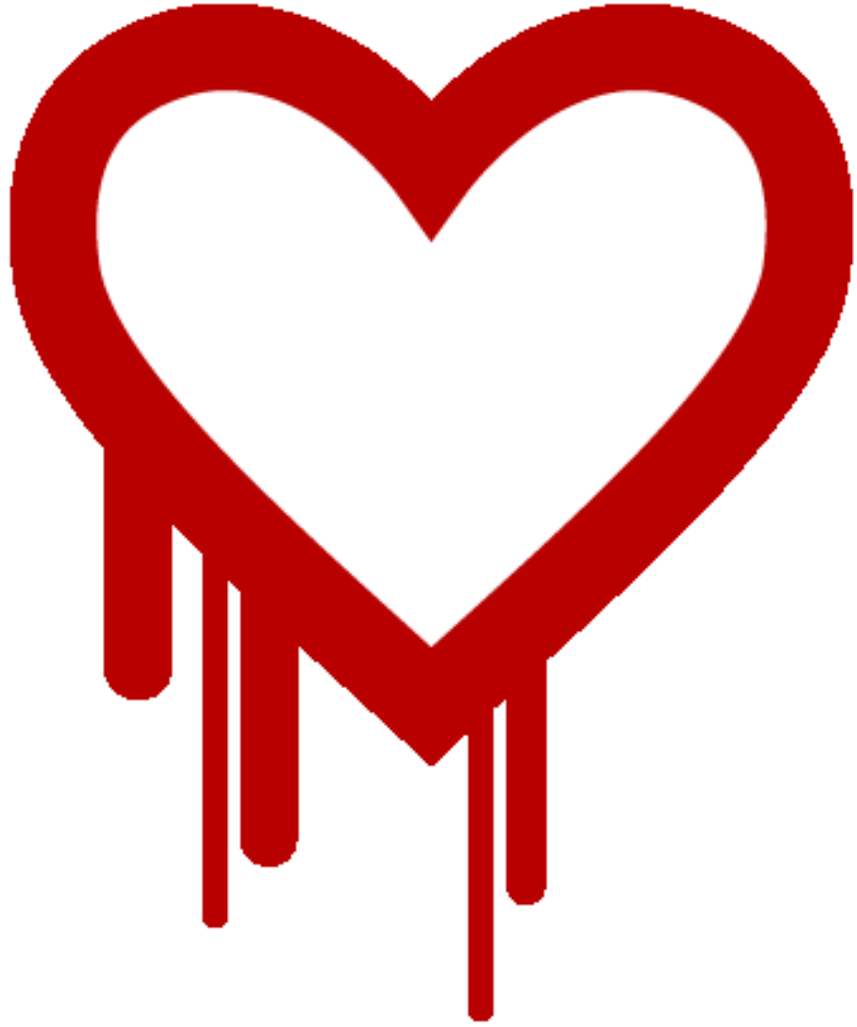
Things I Know About Computers

1. there are a lot of them

Things I Know About Computers

1. there are a lot of them
2. they go wrong a lot

[Ariane 501, $500 million]



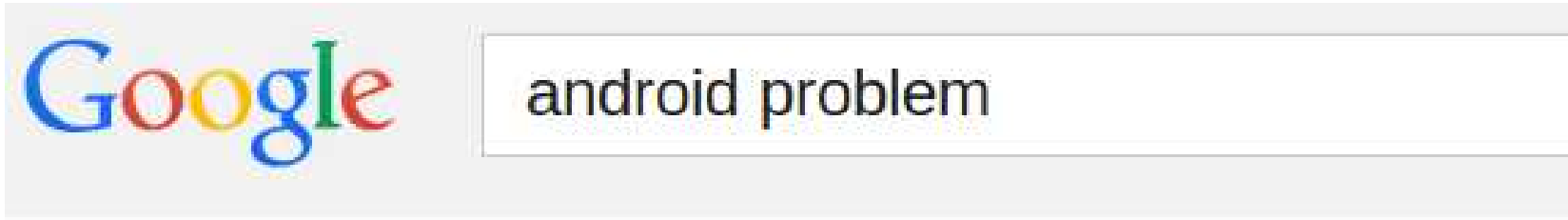
The Register®

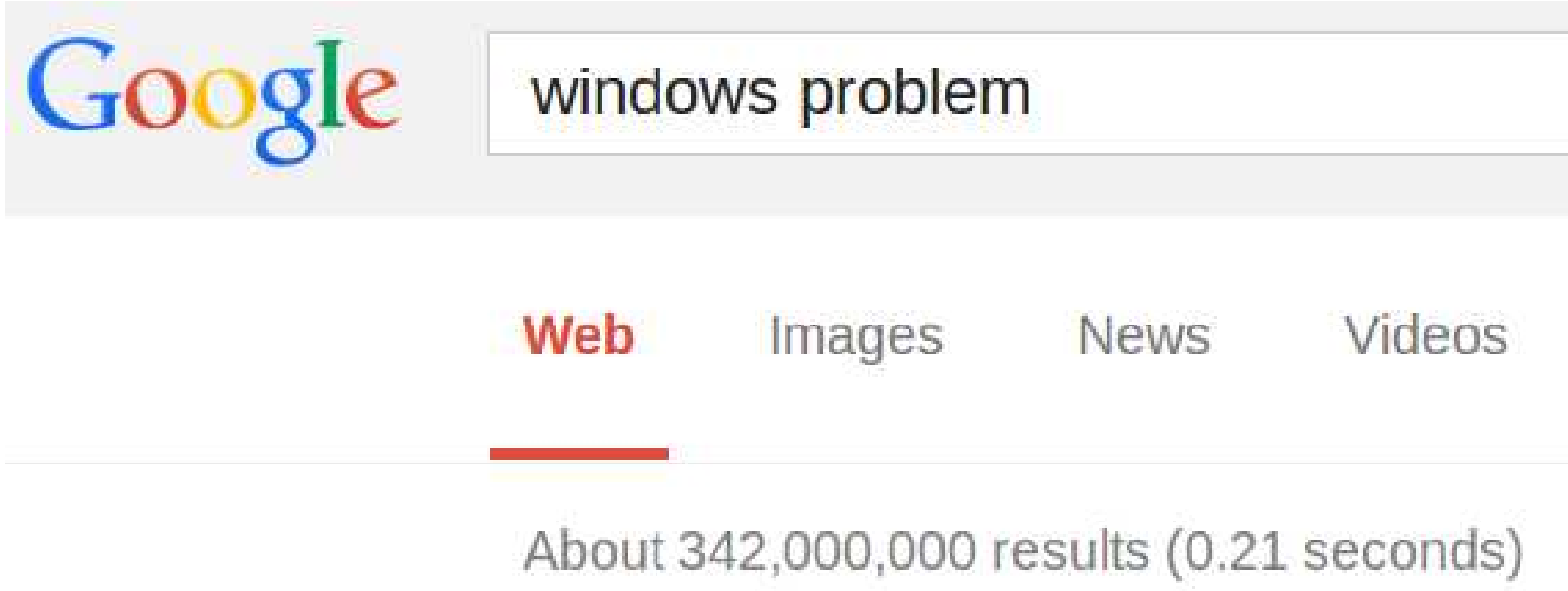
Heartbleed implicated in US hospital megahack

News › World news › NSA

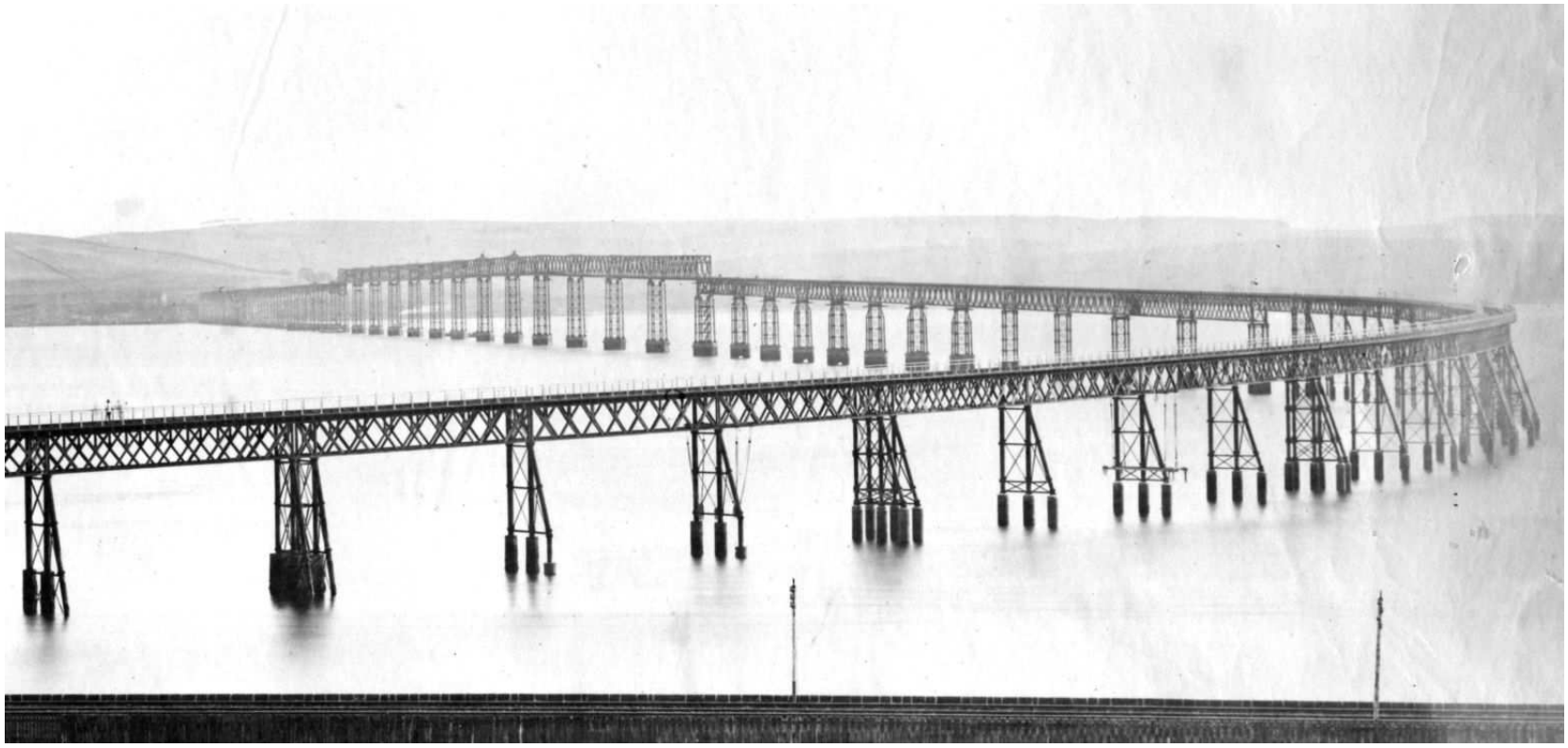
NSA 'hacking unit' infiltrates computers around the world – report

• NSA: Tailored Access Operations a 'unique national asset'


Google
android problem
Web
Images
News
Videos
About 304,000,000 results (0.39 seconds)


Google
windows problem
Web
Images
News
Videos
About 342,000,000 results (0.21 seconds)

Back in the 19th Century...



Beautiful Railway Bridge of the Silvery Tay!
With your numerous arches and pillars in so grand array


HOGWARTS EXPRESS
5972

FIG. 5. AN ILLUSTRATION OF WHAT EXPLOSION DID TO STAYS AND BRACES

Now

100 years later: decent mechanical and civil engineering

Now

100 years later: decent mechanical and civil engineering

What does that mean?

enough thermodynamics, materials science, quality control, etc. to *model* designs well enough to *predict* whether they'll do what we want

Now

100 years later: decent mechanical and civil engineering

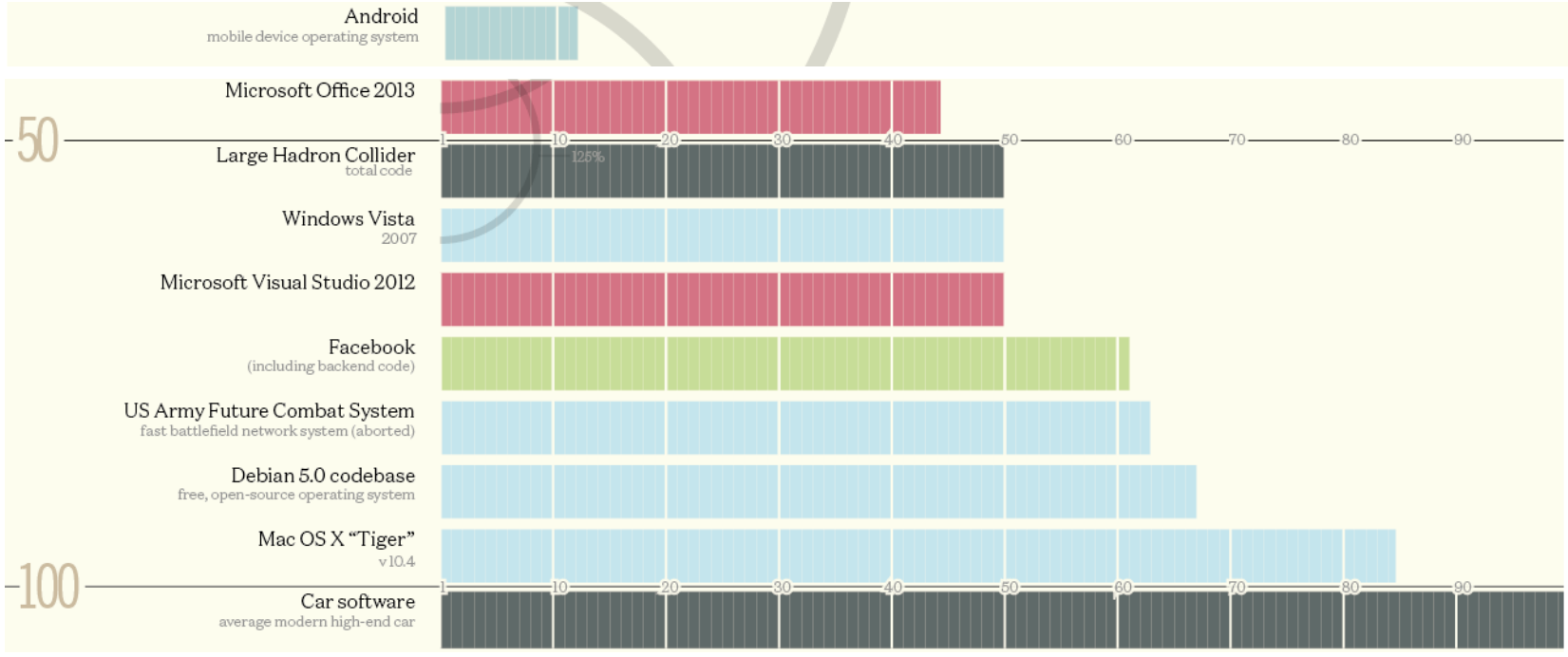
What does that mean?

enough thermodynamics, materials science, quality control, etc. to *model* designs well enough to *predict* whether they'll do what we want

And for Computing?

Why is it so hard?

Too Much Code



[from `www.informationisbeautiful.net`]

But computer systems are built by

- smart people
- in big groups
- subject to commercial pressures
- using the best components and tools they know....


Sunlight
Little Caesars
HOT-N-READY
PIZZA
HOT-N-READY
POCKETS
POCHETTES
Minute Rice
Beatrice
OASIS
Kraft Dinner
12 PACK
HOT-N-READY
BBQ
pepsi


BANKING
S/W
BROWSER
JS
TLS
C
FILESYSTEM
TCP
ARCH
H/W

How do we build those pieces?

1. (sometimes, at best) specify in prose
2. write code
3. write some ad hoc tests
4. test-and-fix-and-extend until marketable
5. test-and-fix-and-extend until no longer marketable
6. use until too bitrotted, device breaks, or obsolete

How do we build those pieces?

1. (sometimes, at best) specify in prose
2. write code
3. write some ad hoc tests
4. test-and-fix-and-extend until marketable
5. test-and-fix-and-extend until no longer marketable
6. use until too bitrotted, device breaks, or obsolete

Too Many...

Execution paths
... scales (at least!) exponentially with code size

States
... scales exponentially with amount of data

Possible inputs

Discrete Systems

What can we do?

1: improve s/w engineering processes

more unit+system testing, more assertions, better coordination...

2: use 1980s languages instead of 1970s

- expressive type systems
- guarantees of type- and memory-safety
- enforcement of abstraction boundaries
- user-defined inductive types, pattern matching, functions, ...

In 2014? No excuse...

2b: use languages that have been *designed*

2b: use languages that have been *designed*

(we can now *precisely define* the intended semantics of real-world PLs...)

4: prove correctness

4: prove correctness

- **CompCert** verified compiler from C-like language to assembly
- **CompCertTSO** ...plus concurrency
- **CakeML** verified compiler from core ML to binary
- **Vellvm** verified LLVM optimisation passes
- **RockSalt** verified SFI
- **seL4** verified hypervisor

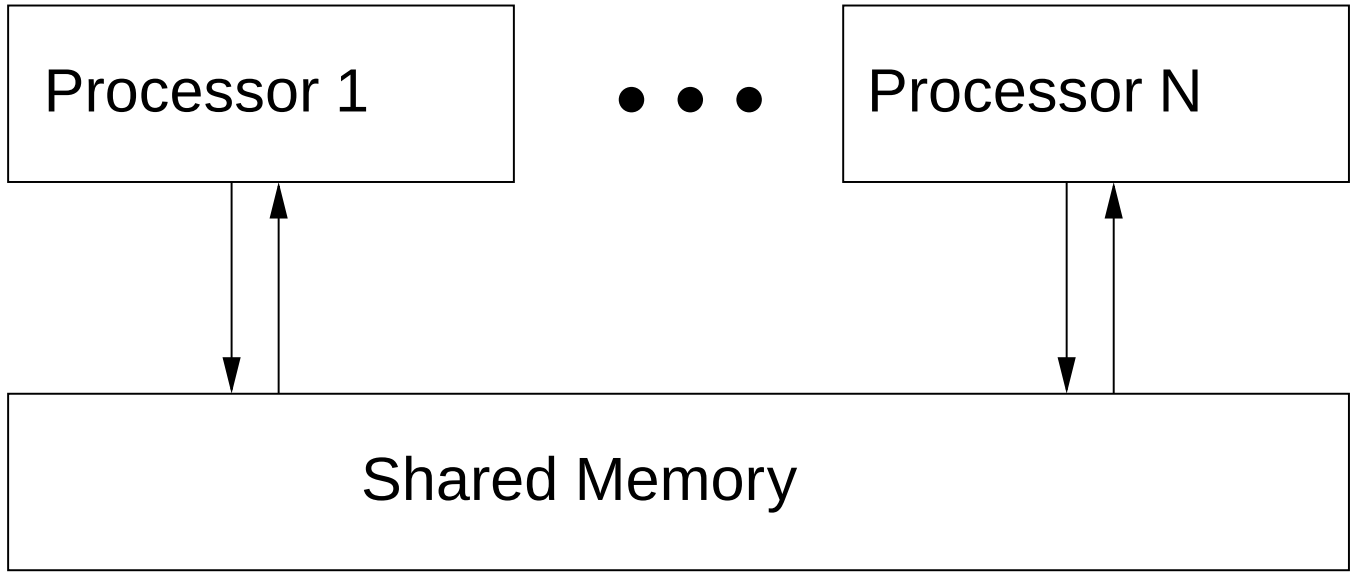
3: specify+test behaviour of key interfaces


BANKING
S/W
BROWSER
JS
C
TLS
FILESYSTEM
TCP
ARCH
H/W

3: specify+test behaviour of key interfaces

- TCP and Sockets API
- x86, ARM, and IBM Power multiprocessor behaviour
- C/C++11 concurrency models (+ gcc/clang testing)
- OCaml core language
- C language
- TLS stack

Multiprocessors



The Ghost of Multiprocessors Past

BURROUGHS D825, 1962

“Outstanding features include truly modular hardware with parallel processing throughout”

FUTURE PLANS
The complement of compiling languages is to be expanded.

Example 1 (SB)

Initial shared memory values: x=0 and y=0

Thread 0	Thread 1
write x := 1	write y := 1
read y	read x

Might expect x=1,y=1, x=1,y=0, or x=0,y=1.

Example 1 (SB)

Initial shared memory values: x=0 and y=0

Thread 0	Thread 1
write x := 1	write y := 1
read y	read x

Might expect x=1,y=1, x=1,y=0, or x=0,y=1.

Experimentally, on x86 (Intel Core i7):

x=1, y=1	79
x=1, y=0	499683
x=0, y=1	499889
x=0, y=0	349

Example 2 (MP, Message Passing)

Initial shared memory values: x=0 and y=0

Thread 0	Thread 1
write x := 1	read y // until gets 1
write y := 1	read x

Might expect to always see x=1

Example 2 (MP, Message Passing)

Initial shared memory values: x=0 and y=0

Thread 0	Thread 1
write x := 1	read y // until gets 1
write y := 1	read x

Might expect to always see x=1

Experimentally:

x86	x=1
ARM	x=0 and x=1
IBM Power	x=0 and x=1

Programmer must enforce ordering with *memory barriers*

How are architectures expressed?

In prose:

For each applicable pair a_i,b_j the memory barrier ensures that a_i will be performed with respect to any processor or mechanism, to the extent required by the associated Memory Coherence Required attributes, before b_j is performed with respect to that processor or mechanism.

- A includes all applicable storage accesses by any such processor or mechanism that have been performed with respect to P1 before the memory barrier is created.
- B includes all applicable storage accesses by any such processor or mechanism that are performed after a *Load* instruction executed by that processor or mechanism has returned the value stored by a store that is in B.

How are architectures expressed?

In prose:

For each applicable pair a_i,b_j the memory barrier ensures that a_i will be performed with respect to any processor or mechanism, to the extent required by the Memory Coherence Required attribute, if a_i is performed with respect to that processor or mechanism:

- A includes all accesses by any such processor or mechanism performed with respect to P1 before the memory barrier is created.
- B includes all accesses by any such processor or mechanism performed after a *Load* instruction executed by that processor or mechanism has returned the value stored by a store that is in B.

How are architectures expressed?

“all that horrible horribly incomprehensible and confusing [...] text that no-one can parse or reason with — not even the people who wrote it”

Anonymous Processor Architect, 2011

Architectural Fiction

Architecture texts (and mainstream language standards too): *informal prose* attempts at subtle loose specifications.

We pretend programmers can “write to the spec” — but we develop software by testing.

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 specs don’t *exist*

Empirical Science to the Rescue!

Empirical Science to the Rescue!

1. invent mathematically precise model of multiprocessor behaviour

   (but abstract, avoiding microarchitectural detail)

Empirical Science to the Rescue!

1. invent mathematically precise model of multiprocessor behaviour

   (but abstract, avoiding microarchitectural detail)

2. make tool to find all model-allowed behaviour of tests

   (and interactive web interface)

Empirical Science to the Rescue!

1. invent mathematically precise model of multiprocessor behaviour

   (but abstract, avoiding microarchitectural detail)

2. make tool to find all model-allowed behaviour of tests

   (and interactive web interface)

3. compare against experimental data from production h/w

   (fixing model and finding h/w bugs)

Empirical Science to the Rescue!

1. invent mathematically precise model of multiprocessor behaviour

   (but abstract, avoiding microarchitectural detail)

2. make tool to find all model-allowed behaviour of tests

   (and interactive web interface)

3. compare against experimental data from production h/w

   (fixing model and finding h/w bugs)

4. discuss with architects

Empirical Science to the Rescue!

1. invent mathematically precise model of multiprocessor behaviour

   (but abstract, avoiding microarchitectural detail)

2. make tool to find all model-allowed behaviour of tests

   (and interactive web interface)

3. compare against experimental data from production h/w

   (fixing model and finding h/w bugs)

4. discuss with architects

5. prove correctness of C/C++11 concurrency compilation scheme

Empirical Science to the Rescue!

1. invent mathematically precise model of multiprocessor behaviour

   (but abstract, avoiding microarchitectural detail)

2. make tool to find all model-allowed behaviour of tests

   (and interactive web interface)

3. compare against experimental data from production h/w

   (fixing model and finding h/w bugs)

4. discuss with architects

5. prove correctness of C/C++11 concurrency compilation scheme

6. goto 1

Rocket Science?

Rocket Science?

specifying the intended behaviour of a system

- in some precise and clear language

Rocket Science?

specifying the intended behaviour of a system

- in some precise and clear language
- in a form that is

  executable as a test oracle

  to decide whether some experimentally observed behaviour is allowed by the model

Rocket Science?

specifying the intended behaviour of a system

- in some precise and clear language
- in a form that is

  executable as a test oracle

  to decide whether some experimentally observed behaviour is allowed by the model

Key Question: any nondeterminism or loose specification?

Not appropriate for everything — but for key infrastructure, yes!

Conclusion

Reasons why building robust systems is hard

Things we can do about it:

- use better PL tools
- specify for test
- work towards full verification

Conclusion

Reasons why building robust systems is hard

Things we can do about it:

- use better PL tools
- specify for test
- work towards full verification

Cautious optimism?

Conclusion

Reasons why building robust systems is hard

Things we can do about it:

- use better PL tools
- specify for test
- work towards full verification

Cautious optimism?

or we’re doomed!

The End