DATA 61

Software Verification in the Real World

Hira Taqdees Syeda
Trustworthy Systems Group

August 2019

www.ts.data61.csiro.au

UNSW SYDNEY

CSIRO

Software verification

How

Where

Why should software be verified?

What do you think?

Why should software be verified?

Ariane 5, 4th June 1996, **37 seconds** after lift-off

10 years and **$7 billion** to produce

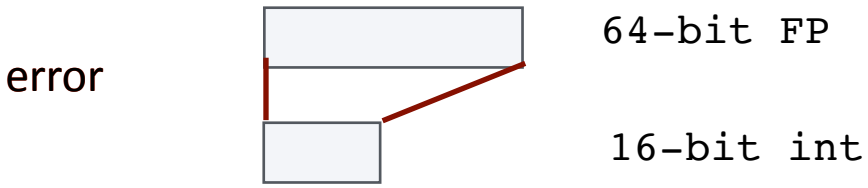
Destroyed rocket valued at **$500 million**

Ariane 5 explosion — cause

Halting of inertial navigation system → Resultantly → disintegration under high aerodynamic forces → Finally → self-destruction by automated flight termination system

Halting of inertial navigation system → Why? → the rocket was making an abrupt course correction commanded by On-board Computer (OBC) software

```
void OBC(){
  ...
  float f = get_velocity();
  int data = compute_data (f);
  correct_course(data);
  ...
}
```



Why should software be verified?

software error ⇒

More recent — car hacking!

Car hacking

1- via the “on-board diagnostic software”

2- using corrupted mp3

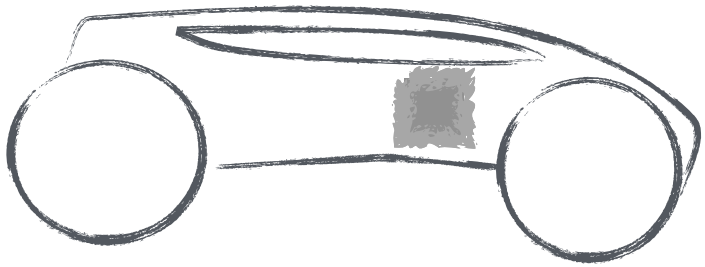
3- wirelessly!!

“we demonstrate the ability to control a wide range of automotive functions”

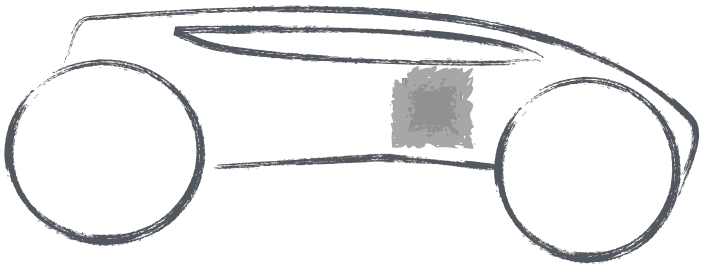
“including disabling the brakes, selectively braking individual wheels on demand, stopping the engine, and so on. “

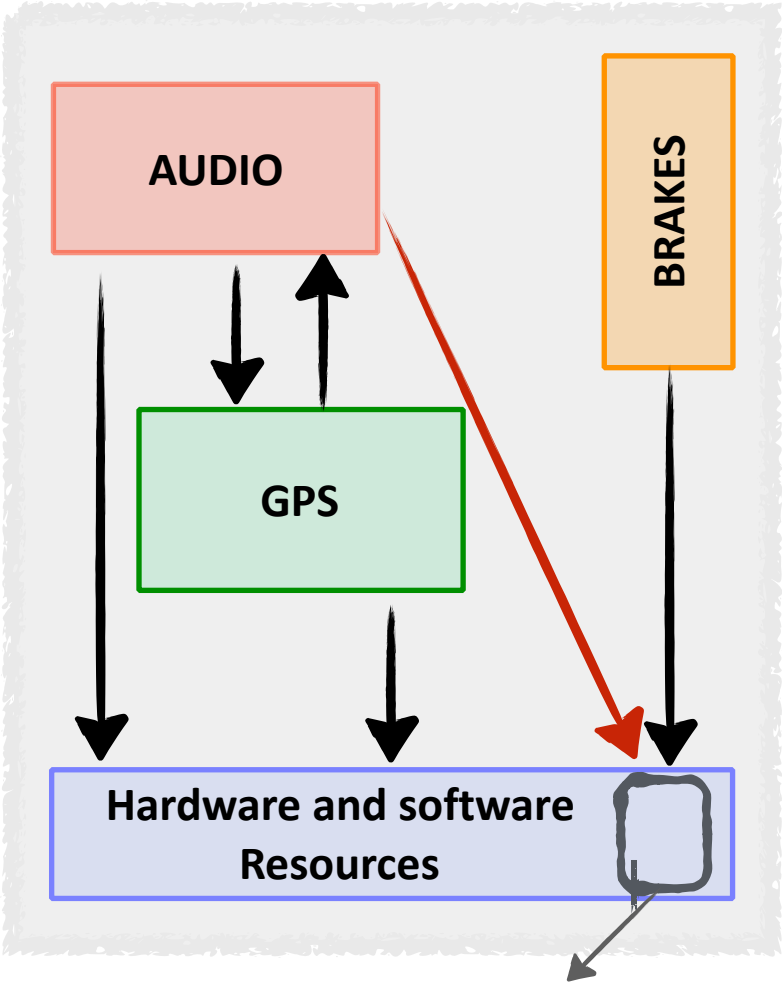
University of Washington and the University of California, San Diego, “Experimental Security Analysis of a Modern Automobile”, IEEE S&P 2010

Car hacking explained



Car hacking explained

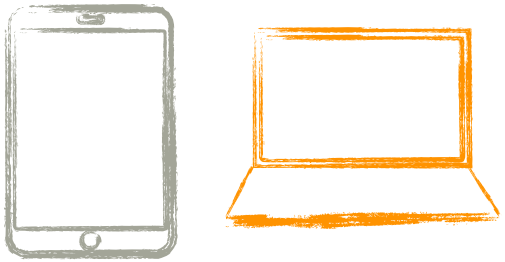




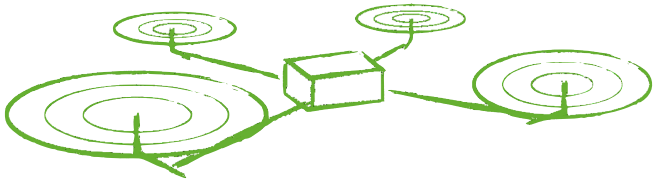
the program
managing the brakes

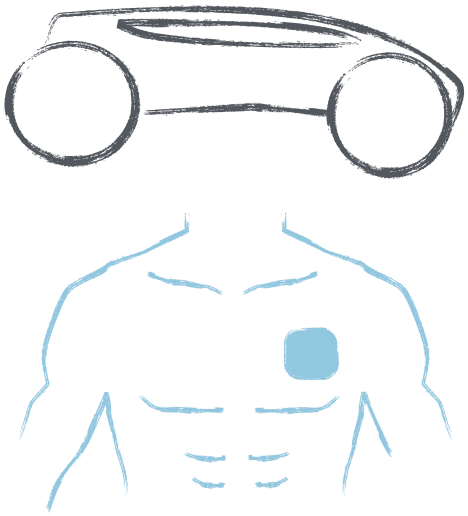
So, why should we verify software?

- To produce **better** programs
- To be able to give **guarantees** about them



- Because these programs are used in **critical** devices!





"failure to verify software will be treated legally as negligence"
(Tony Hoare)

So, imagine…

Imagine a car manufacturer company
asking us to provide with:

- a **software system** for ECUs (electronic control units)
- **guarantees** that hackers can not produce attacks

Software verification

Why

How

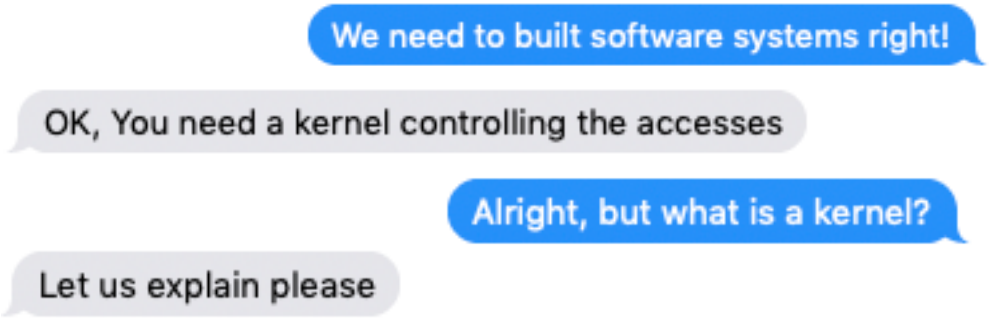
Where

How to create verified software?

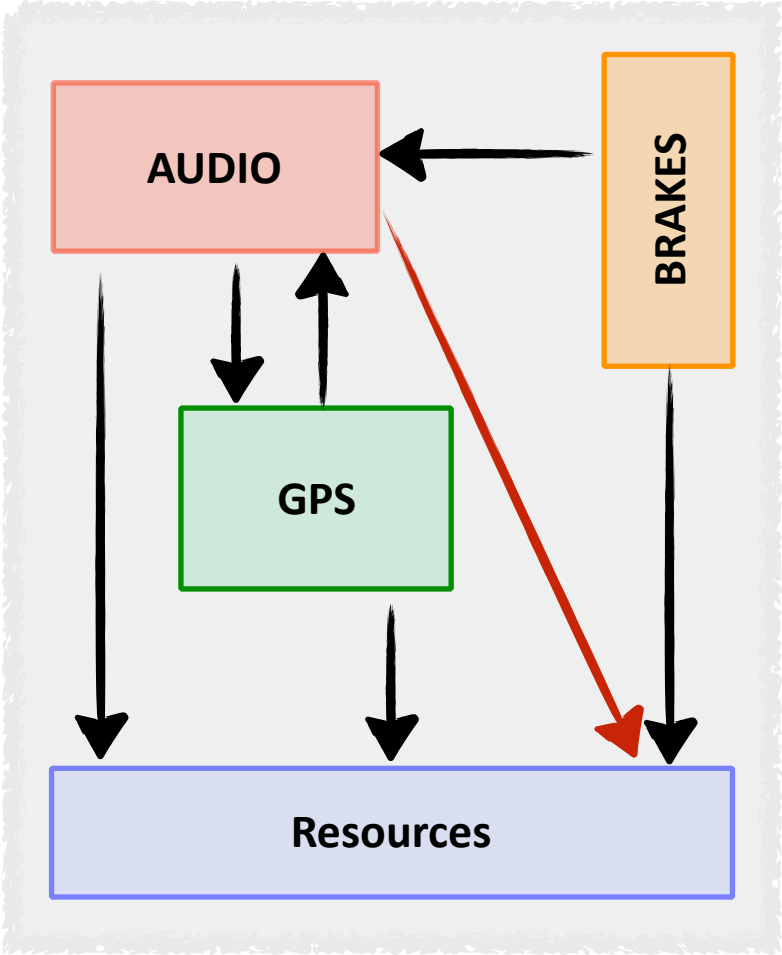
We need

a software system **guarantees**

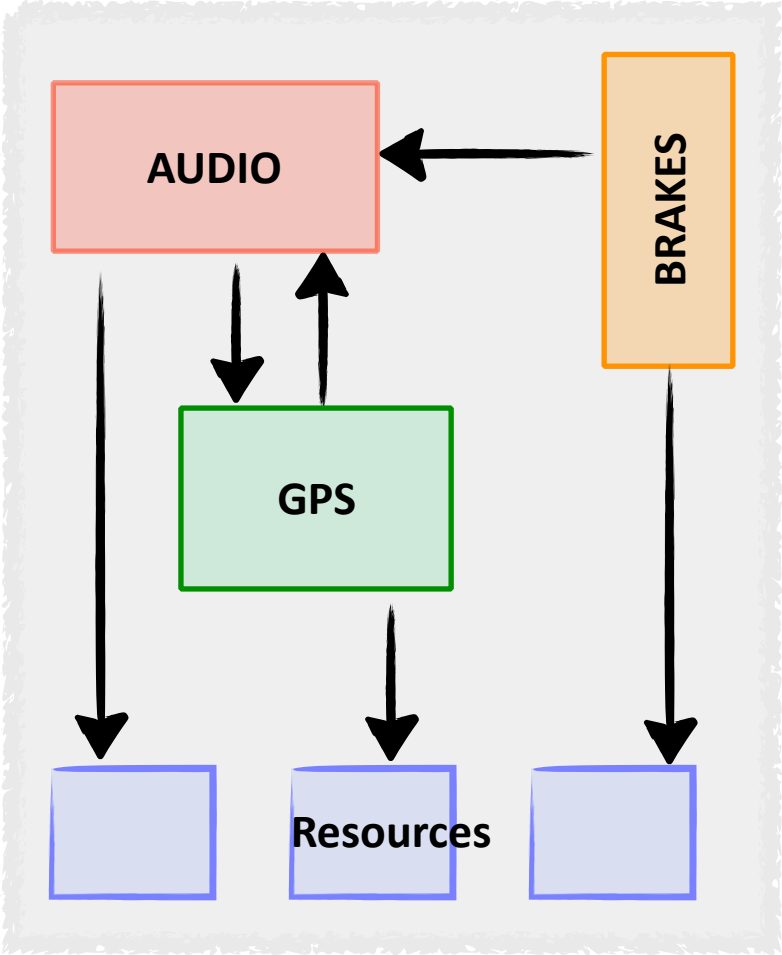
Let's talk to "systems" people:



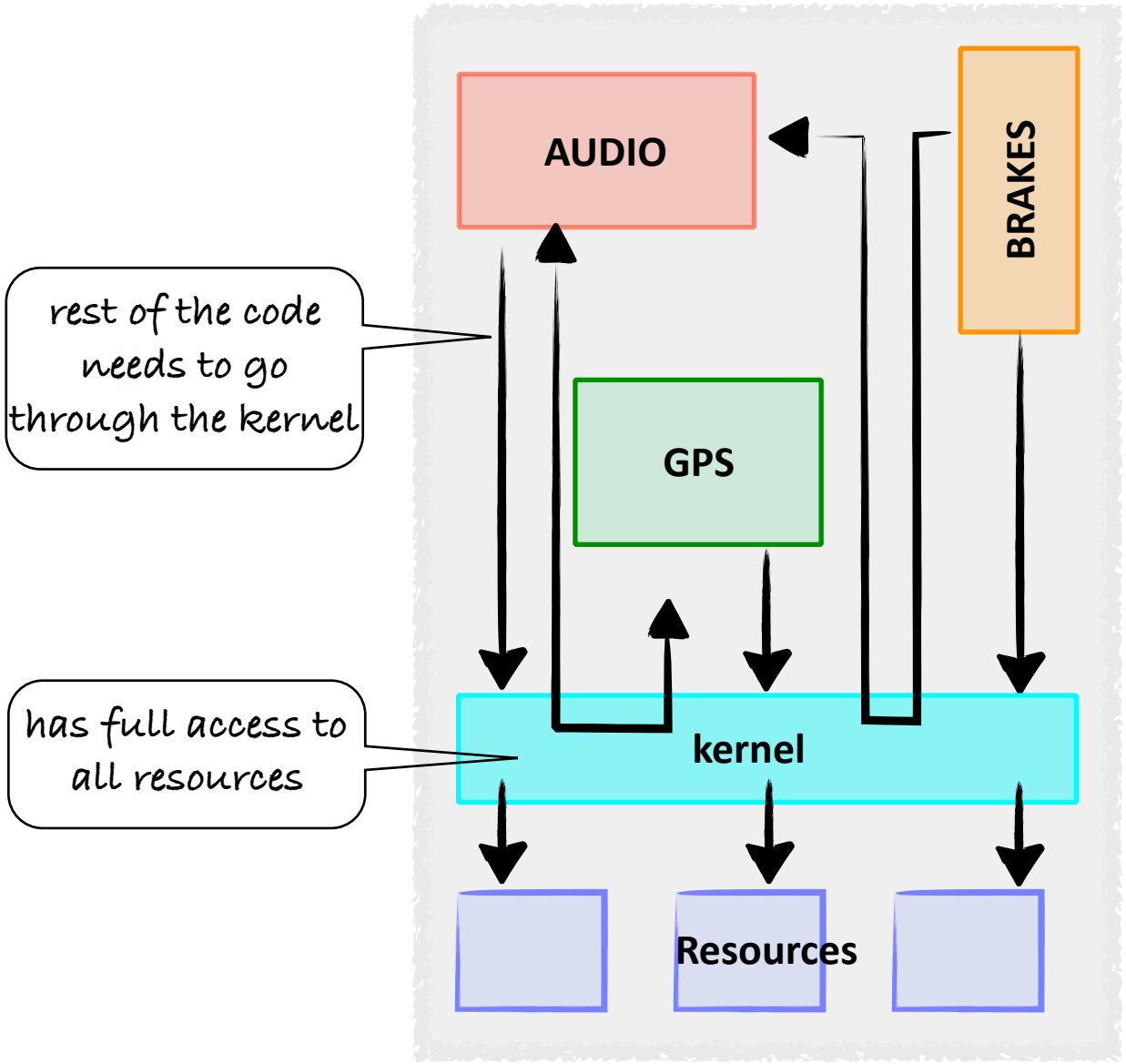
What is a kernel?



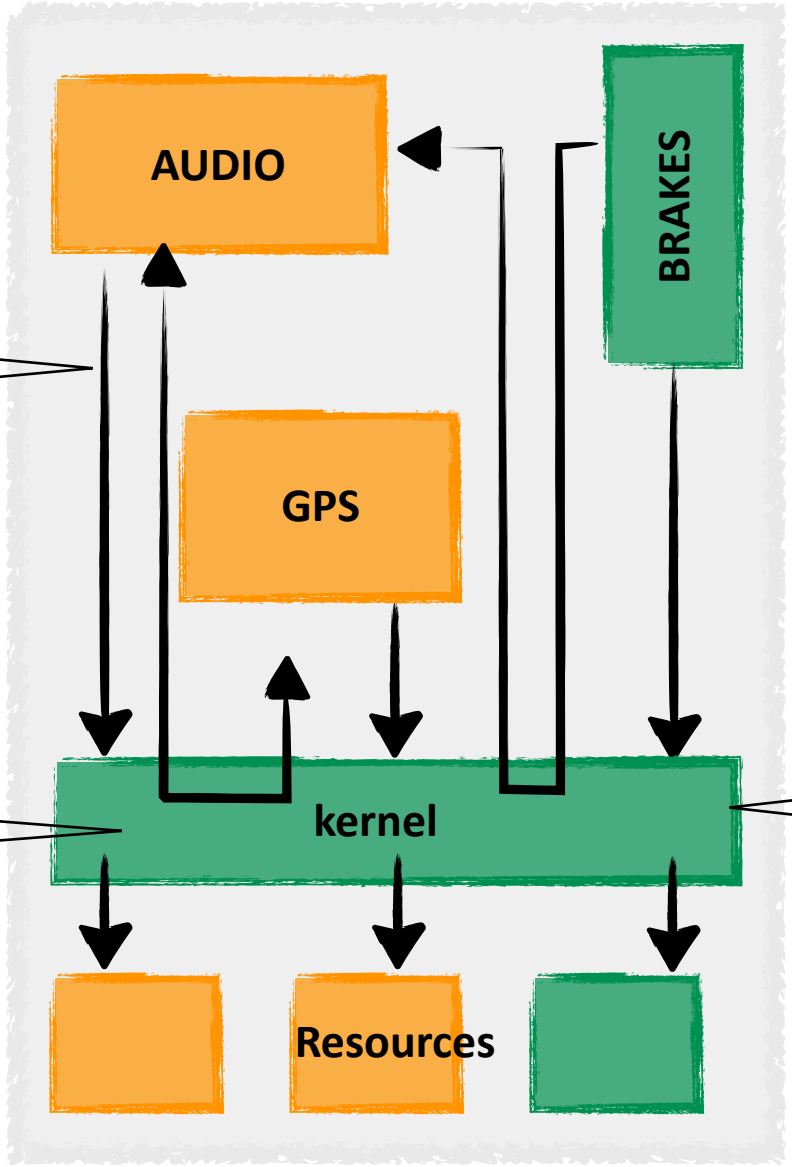
What is a kernel?



What is a kernel?



What is a kernel?



rest of the code needs to go through the kernel

has full access to all resources

critical

uncritical

provides isolation

How to create verified software?

We need

a software system **guarantees**

Let's talk to "systems" people:

We need to built software systems right!

OK, You need a kernel controlling the accesses

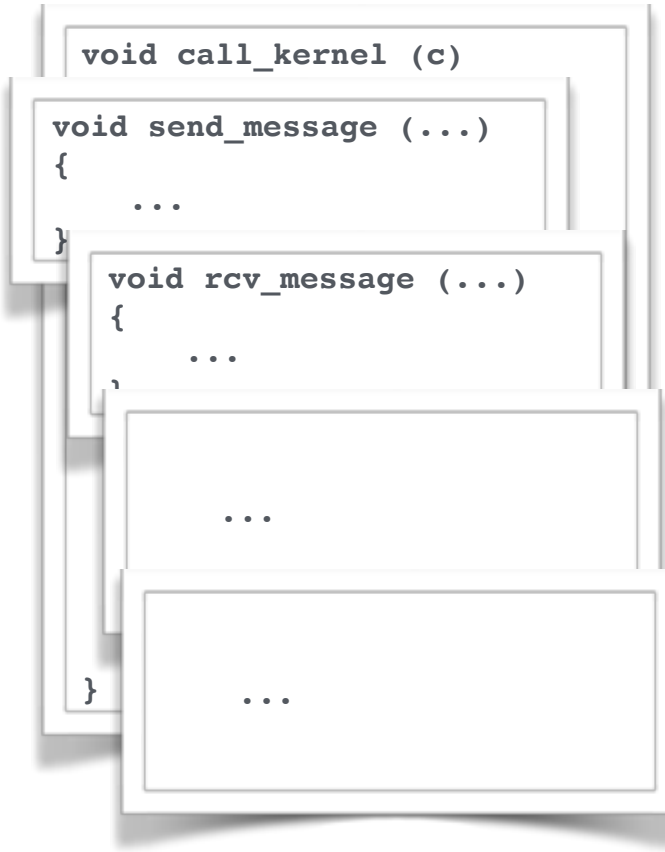
Alright, but what is a kernel?

Let us explain please

So, this is a kernel... [busy writing a kernel]

Now, how do we get guarantees about it?

Oh, for this you need to talk to verification people

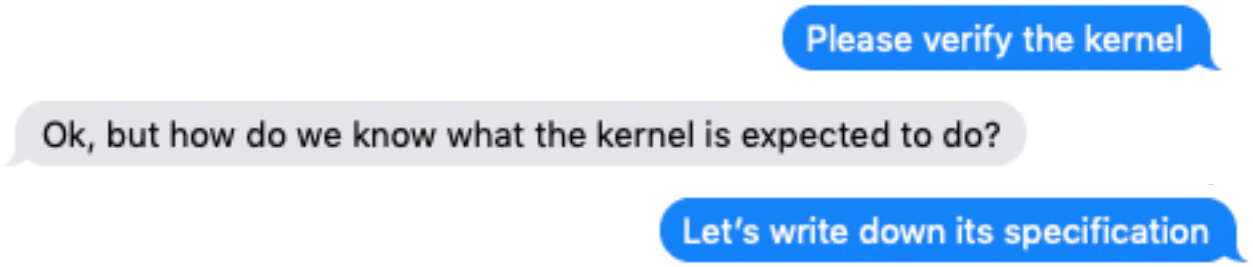


How to create verified software?

We need

a software system **guarantees**

Let's talk to "verification" people:



Program Specification

- Functional specification
  - e.g. memory manager should provide address translation
- High-level Properties
  - e.g. kernel should provide security, isolation, confidentiality
- Pre- and post- execution conditions
  - e.g. `x` holds value `0`, execute `x = 5`, `x` holds value `5`

pre-condition

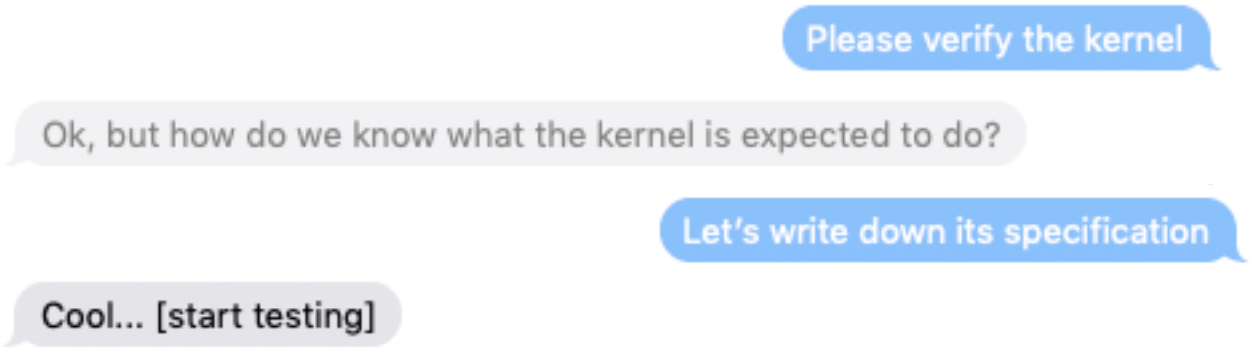
post-condition

How to create verified software?

We need

a software system **guarantees**

Let's talk to "verification" people:



Software testing

- Unit testing
  - generate input test vectors, examine the results
- Property based testing
  - QuickCheck of Haskell!
- Program Analysis
  - static analysis, type checking, etc
- and many more…
- But, testing is **incomplete**…!
  - we cannot test the program behaviour for all possible test cases
  - hence we cannot generalise on the validated properties

Software testing

- But, testing is **incomplete**…!

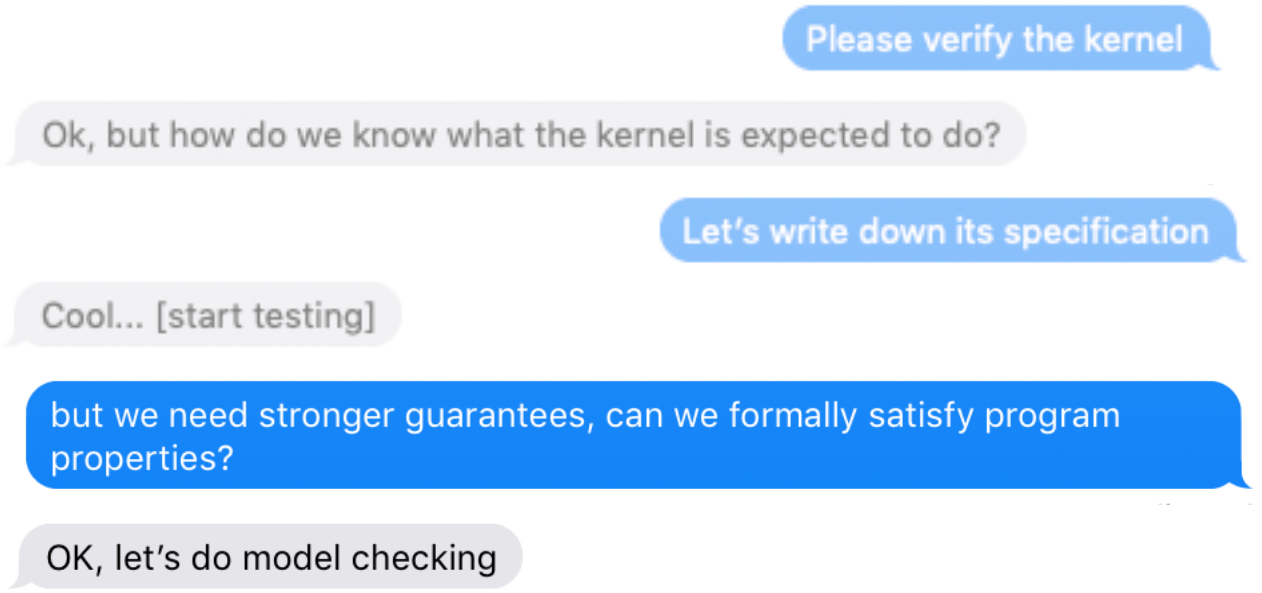
"Testing shows the presence, not the absence of bugs"
(Dijkstra)

How to create verified software?

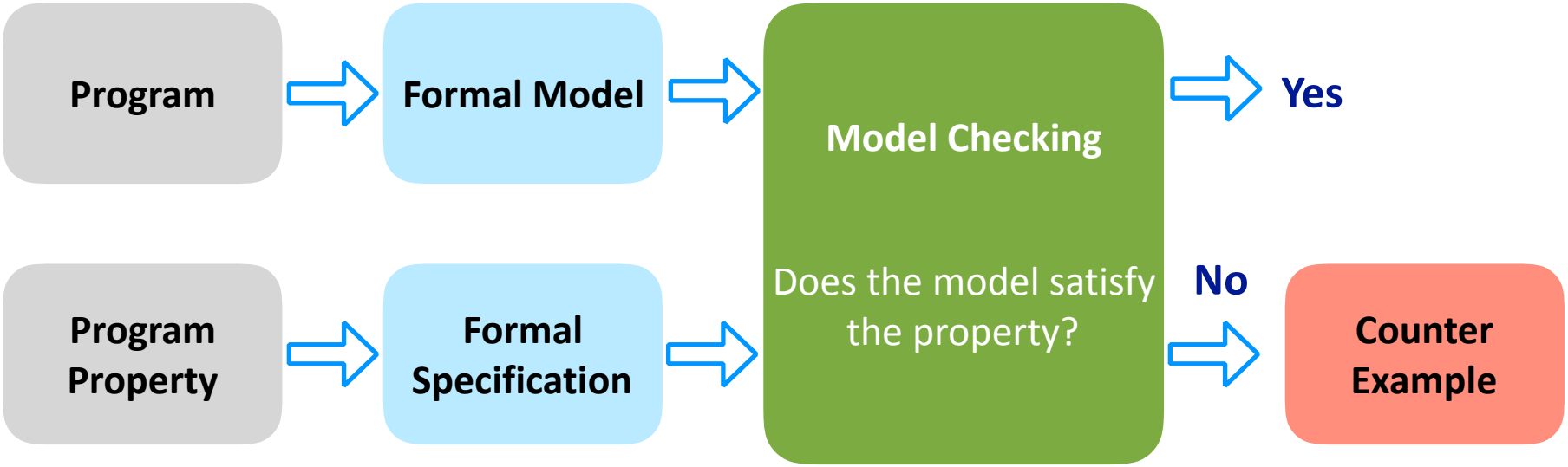
We need

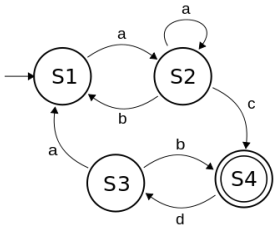
a software system **guarantees**

Let's talk to "verification" people:



Formal verification—Model checking





Finite state machine

`Eventually`, address will be translated
`Always`, kernel will provide isolation

But, what is the **catch** here?

state space explosion!!!

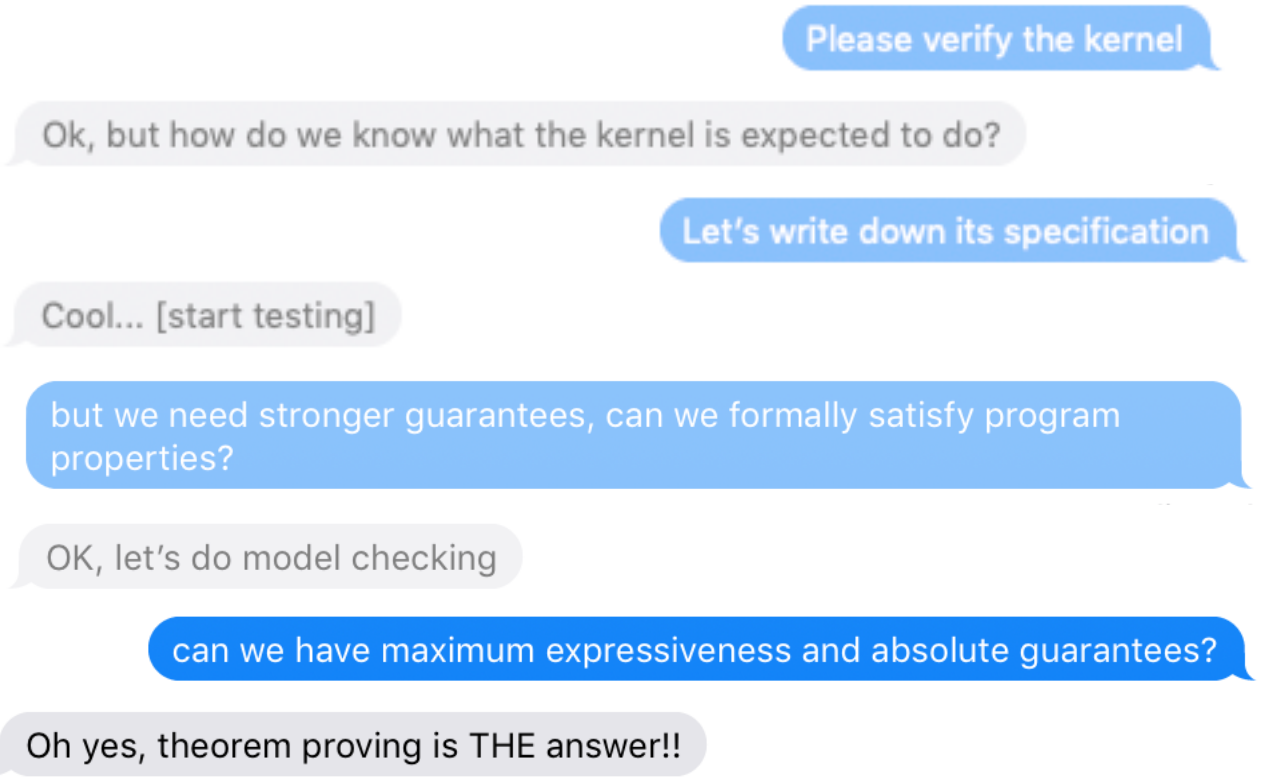
Model checking:
accurate but inexpressive

How to create verified software?

We need

a software system **guarantees**

Let's talk to "verification" people:



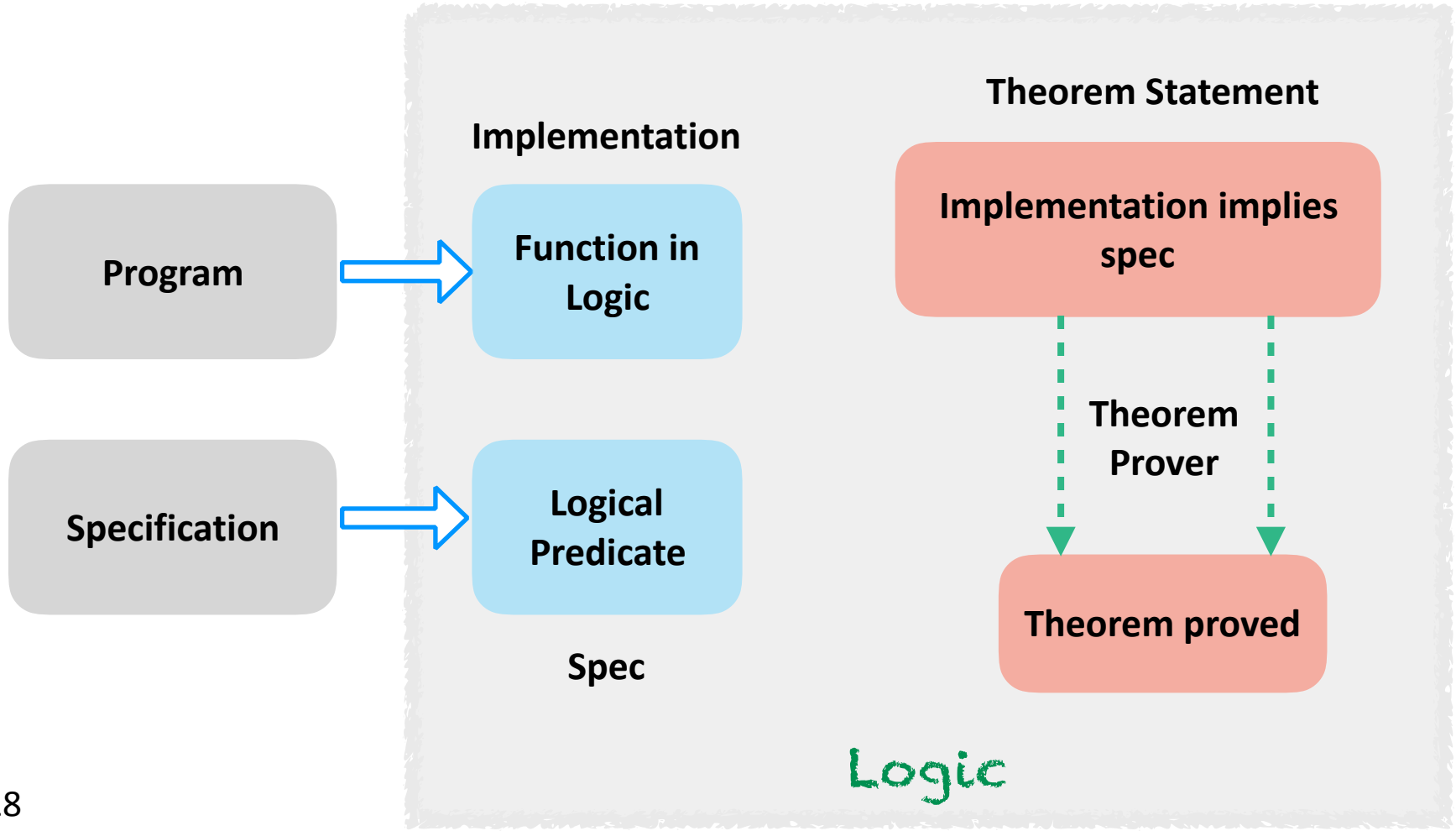
Formal verification—
Theorem proving

- Logical reasoning for program verification

All humans are mortal
Socrates is a human
Therefore, Socrates is mortal

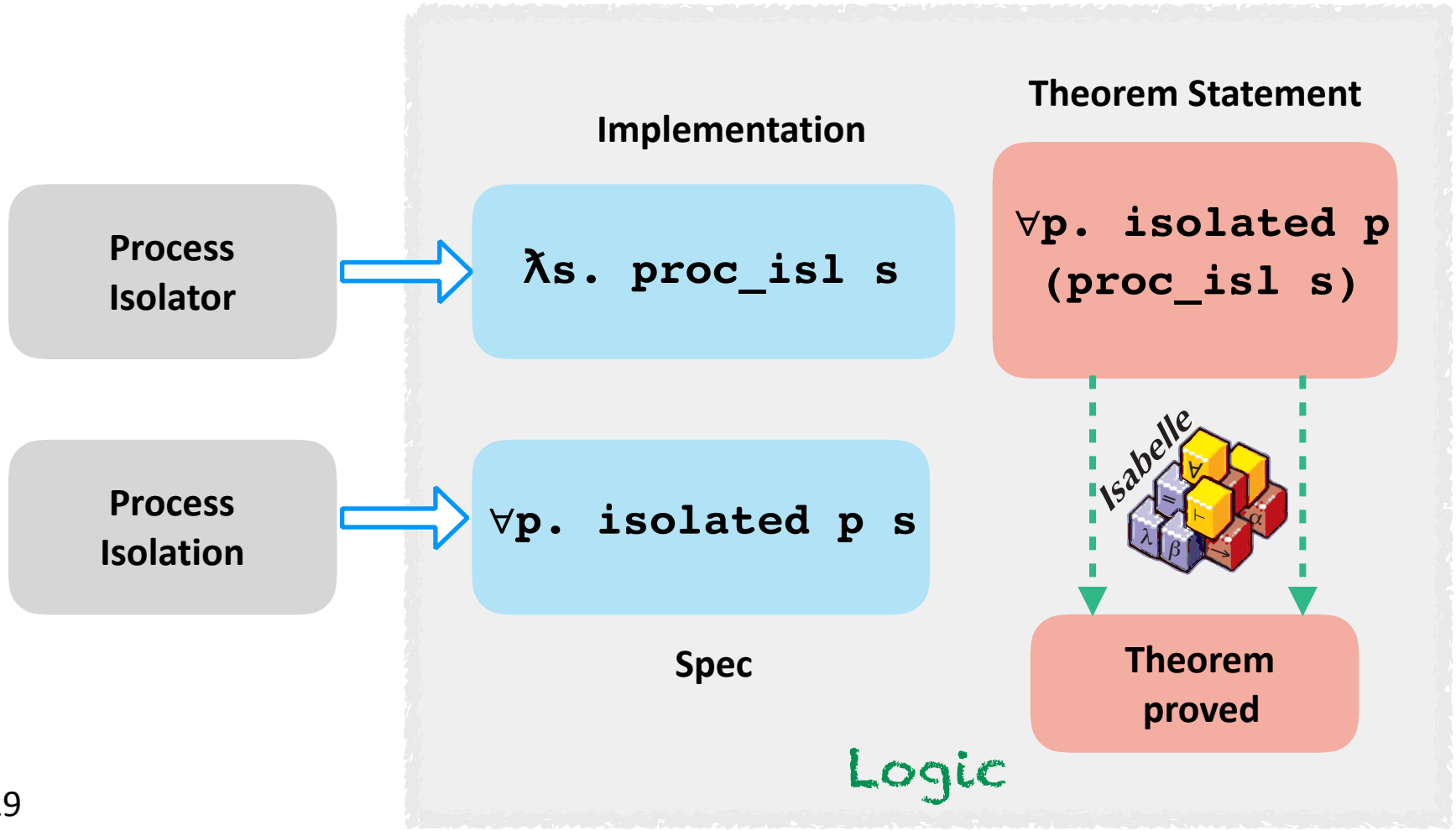
Formal verification—Theorem proving

- Logical reasoning for program verification



Formal verification—
Theorem proving

- Logical reasoning for program verification

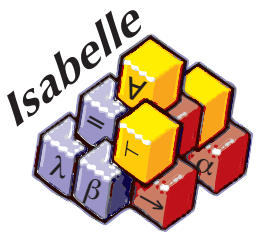


Summary —
Software verification methods

	Testing	Model Checking	Theorem Proving
Expressiveness	☑	☒	☑
Accuracy	☒	☑	☑
Automation	☑	☑	☒

Software verification

Why



How

software

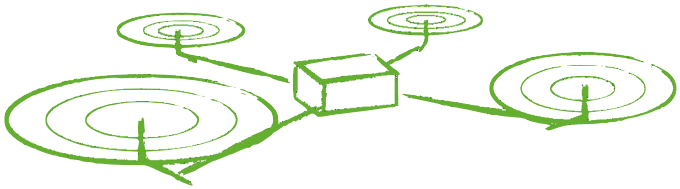
implies

specification

Where

Where is software verification used?

In quadcopters!



At Data61!

Where is software verification used?

Where is software verification used?

seL4 of Trustworthy Systems

1 microkernel

8,700 lines of C code

0 bugs *

QED

*conditions apply

seL4 — functional c
*conditions apply

Expectation

Specification

Proof

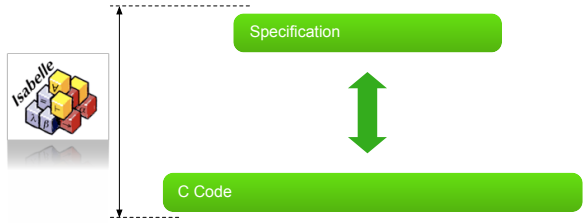
Assume correct:

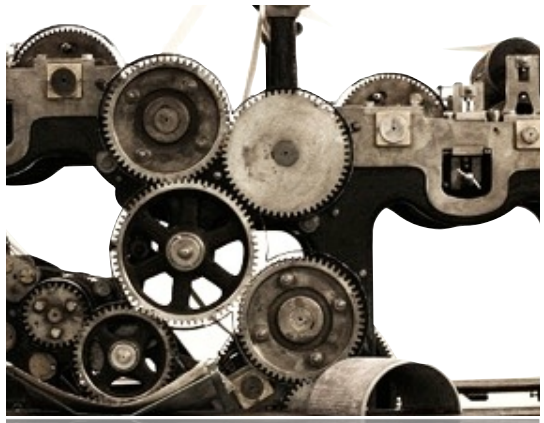
- assembly code (600 loc)
- hardware (ARMv6)
- cache and TLB management
- boot code (1,200 loc)

seL4 — guarantees

Execution always defined:

- no null pointer de-reference
- no buffer overflows
- no code injection
- no memory leaks/out of kernel memory
- no div by zero, no undefined shift
- no undefined execution
- no infinite loops/recursion





seL4 —big deal?

Slashdot

Stories Recent Popular

Slashdot is powered by your

Technology: World

Posted by Soulskill on Thursda
from the wait-for-it dept.

An anonymous reader writes

> "Operating systems usual
> and so forth are known by
> to prove that a particular C
> formally verified, and as s
> researchers used an exec
> the Isabelle theorem prove
> matches the executable a

New Scientist
Saturday 29/8/2009
Page: 21
Section: General News
Region: National
Type: Magazines Science / Technology
Size: 196.31 sq.cms.
Published: -----S-

The ultimate way to keep your computer safe from harm

FLAWS in the code, or "kernel", that sits at the heart of modern computers leave them prone to occasional malfunction and vulnerable to attack by worms and viruses. So the development of a secure general-purpose microkernel could pave the

just mathematics, and you can reason about them mathematically," says Klein.

His team formulated a model with more than 200,000 logical steps which allowed them to prove that the program would always behave as its

Does it run Linux? "We're pleased to say that it does. Presently, we have a para-virtualized ver

> of a general-purpose operati

kernelét alkotó 7 500 sornyi C forráskód helyességét igazolni egyedülálló teljesítmé
eredményeképpen pedig egy olyan megbízhatóságot kapnak a szoftvertől, amely e

Trustworthy Systems is much more now!

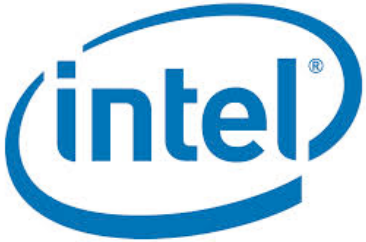
- Verified code generation
- Concurrent computing systems
- Verified compilers
- Verified file system

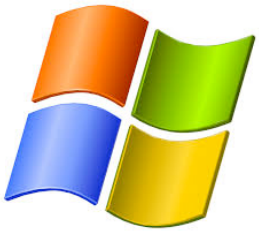
and more!

Formal verification elsewhere?

- Intel
  - formal verification replacing testing for hardware
- INRIA
  - certified compiler CompCert
- Microsoft
  - Terminator
- Airbus A380
  - code modelling and generation

and many more!





Software verification

Why

Isabelle

How

software

implies

specification

Where

What's next for YOU?

- Comp4161
  - Advanced Topics in Software Verification
- Taste of Research (ToR) Summer Projects
  - Formal methods
  - Systems design
  - Programming languages research
- Internship and casual work
  - or simply visit us!

we are at level-3 CSE, K-17 building

https://ts.data61.csiro.au

Thank You! :-)