# 10 Years of Trustworthy Systems

David Cock

July 29, 2014

# Outline

# seL4

First ever verified kernel.

- Writen in C — high-performance.
- Verified in Isabelle/HOL.

# seL4

First ever verified kernel.

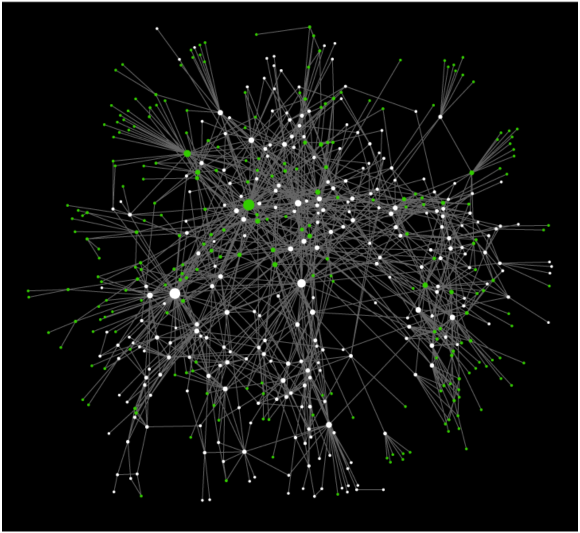
- Writen in C — high-performance.
- Verified in Isabelle/HOL.
- Open source from yesterday!

`http://sel4.systems`

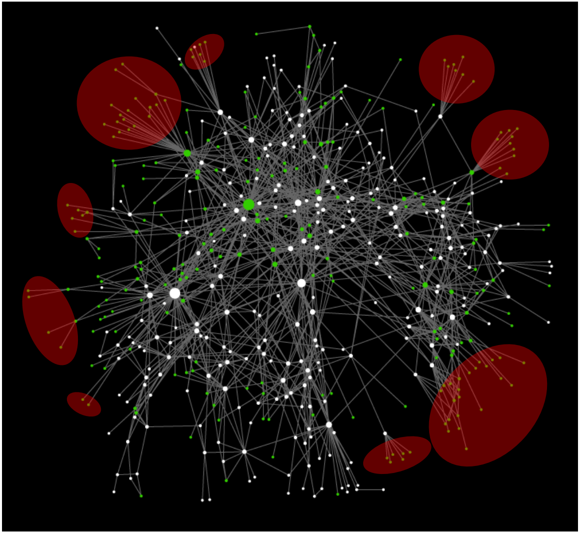
The C kernel implements the high-level specification.

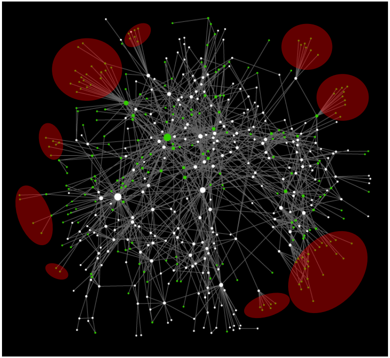
- Initial implementation — 2 weeks.
- Small — 8,700 lines.
- Fast — 224cyc one-way IPC.
- DSL automation — bitfields.

# The seL4 Call Graph

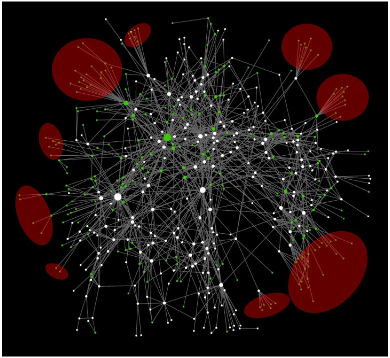


# The seL4 Call Graph

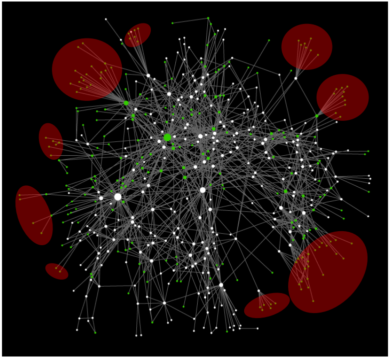




- 573 functions.
- Not modular — No SCCs. . . . . . except those leaves.



- 573 functions.
- Not modular — No SCCs. . . . . . except those leaves.
- 198 of these: 35% of functions, 16% of LOC.



- 573 functions.
- Not modular — No SCCs. . . . . . except those leaves.
- 198 of these: 35% of functions, 16% of LOC.
- Generated and proved automatically from a DSL.

- DSL:

```
base 32
block B { padding 3 field Y 13 field Z 16 }
```

- DSL:

```
base 32
block B { padding 3 field Y 13 field Z 16 }
```

- C:

```
static inline void
B_ptr_set_X(B_t *B_ptr, uint32_t v) {
    B_ptr->words[0] &= ~0x1fff0000;
    B_ptr->words[0] |= (v<<16)&0x1fff0000;
}
```

- DSL:

```
base 32
block B { padding 3 field Y 13 field Z 16 }
```

- C:

```
static inline void
B_ptr_set_X(B_t *B_ptr, uint32_t v) {
    B_ptr->words[0] &= ~0x1fff0000;
    B_ptr->words[0] |= (v<<16)&0x1fff0000;
}
```

- HOL:

$$
\begin{aligned}
&B\_lift\ B \equiv (\!|\\
&\quad B\_CL.X_{C}L = ((index\ (B\_C.words\_C\ \ B)\ 0) >> 16)\ \mathrm{AND}\ 8191,\\
&\quad B\_CL.Y_{C}L = ((index\ (B\_C.words\_C\ \ B)\ 0) >> 0)\ \mathrm{AND}\ 65535|\!)
\end{aligned}
$$

# Automation Helps!

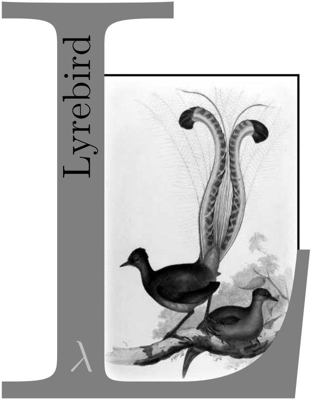
- 35% of the functions in seL4 were proved automatically.
- The tool is now widely used in NICTA.
- It's used by engineers, not formal methods people.
- Many features not mentioned: tagged unions, multilevel decoding, . . . .

# Further Reading

For more see:

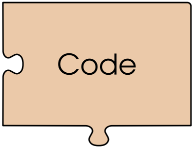
- *Running the manual: An approach to high-assurance microkernel development, Haskell Workshop '06.*
- *Bitfields and tagged unions in C: verification through automatic generation, VERIFY'08.*
- *Secure microkernels, state monads and scalable refinement, TPHOLS'08.*
- *Mind the gap: A verification framework for low-level C, TPHOLS'09.*
- *seL4: Formal verification of an OS kernel, SOSP'09*

# Outline

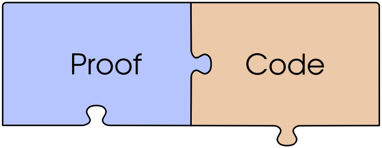



- A DSL for CPU/system modelling.
- High performance simulator.
- Automatic formal model.
- Used to prototype seL4.

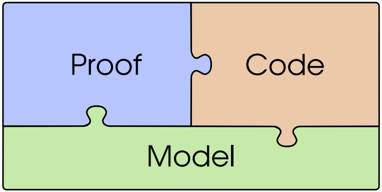
Program proof is important, but there's more to do.



Program proof is important, but there's more to do.



Program proof is important, but there's more to do.

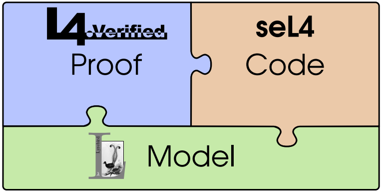



Any statement "P is True" is incomplete:
It must be read as ", under Q - my model of the world".

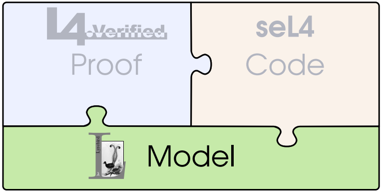
## Goal

*Development outcomes: program, proof and model.*

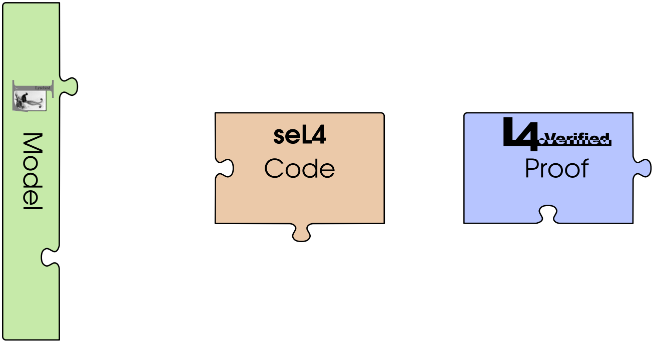
Program proof is important, but there's more to do.

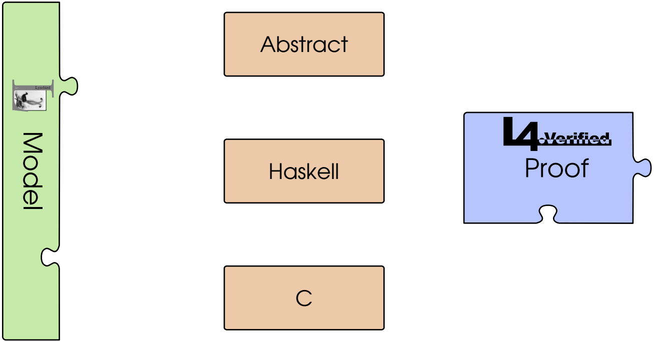


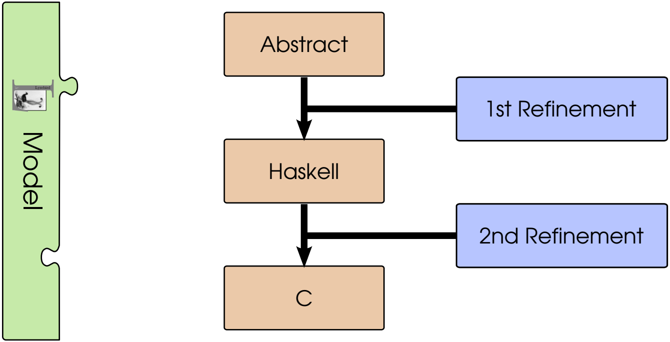
Program proof is important, but there's more to do.

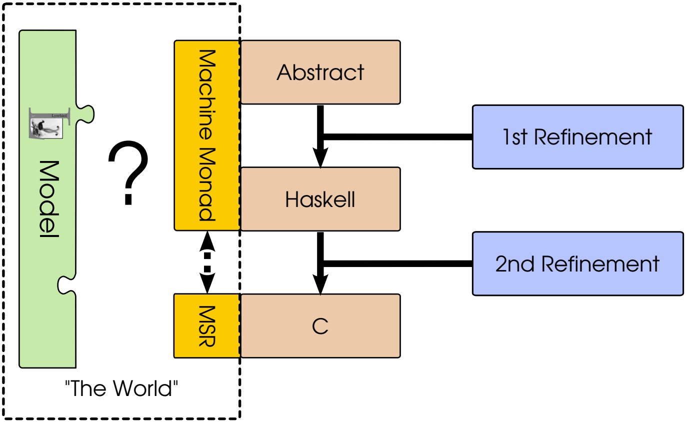


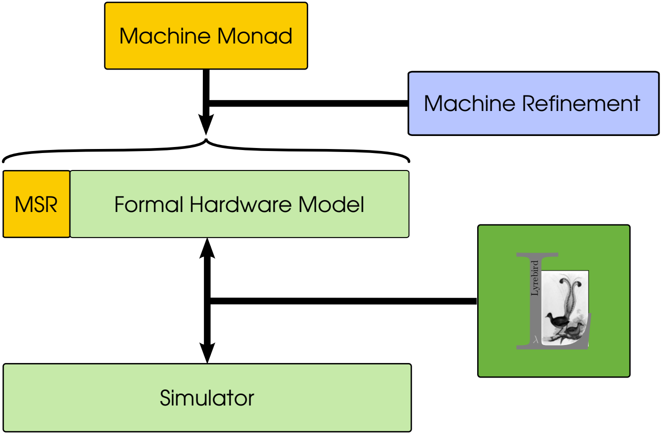

Our approach is a language framework: *Lyrebird*.

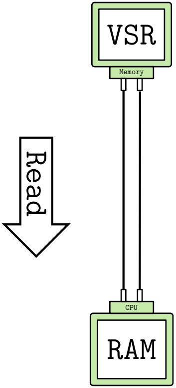

Model
seL4
Code
L4.verified
Proof


Model
Abstract
Haskell
C
L4.verified
Proof

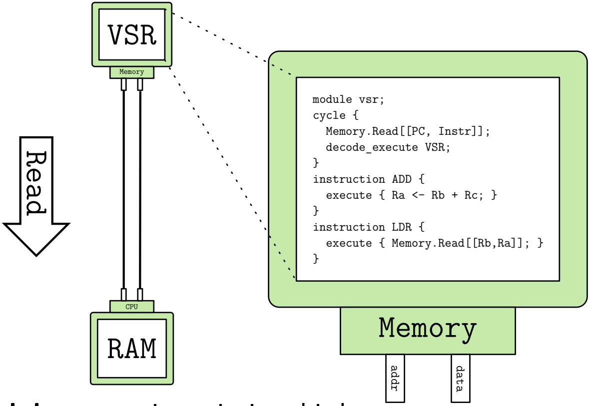

Model
Abstract
1st Refinement
Haskell
2nd Refinement
C


Model
?
Machine Monad
MSR
"The World"
Abstract
Haskell
C
1st Refinement
2nd Refinement

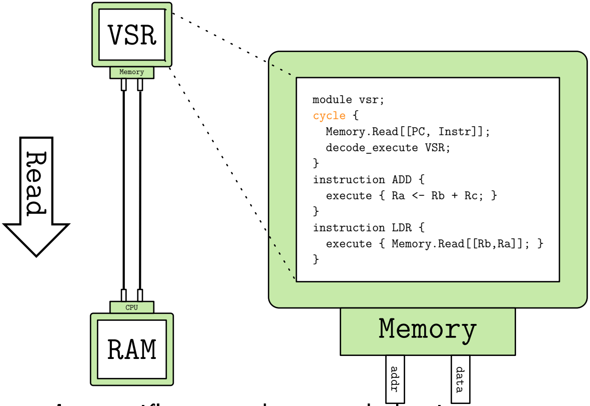

Machine Monad
Machine Refinement
MSR
Formal Hardware Model
Simulator
Lyrebird




A simple model of a CPU connected to RAM.

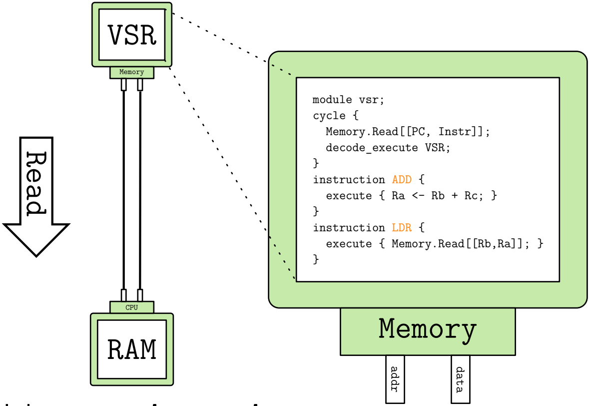



**Modules** are written in Lyrebird.



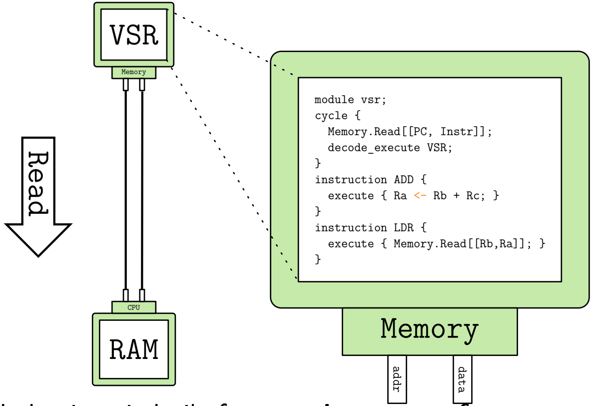

```
module vsr;
cycle {
  Memory.Read[[PC, Instr]];
  decode_execute VSR;
}
instruction ADD {
  execute { Ra <- Rb + Rc; }
}
instruction LDR {
  execute { Memory.Read[[Rb,Ra]]; }
}
```

The **cycle** specifies asynchronous behaviour.

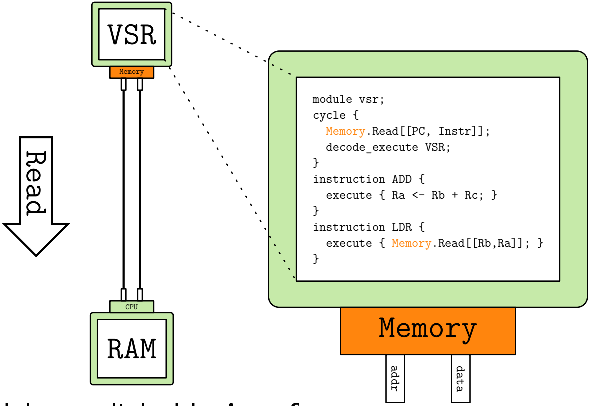



```
module vsr;
cycle {
  Memory.Read[[PC, Instr]];
  decode_execute VSR;
}
instruction ADD {
  execute { Ra <- Rb + Rc; }
}
instruction LDR {
  execute { Memory.Read[[Rb,Ra]]; }
}
```

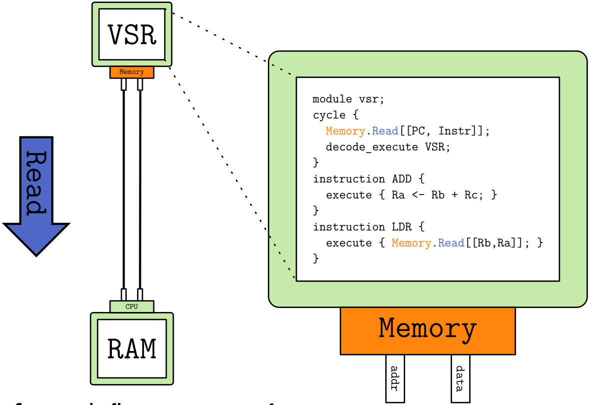
Modules export **instructions**.



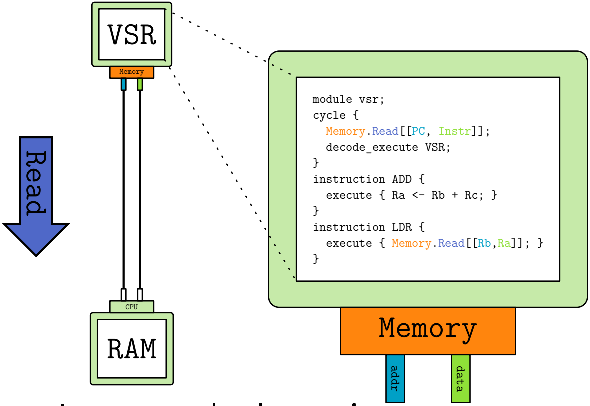

All behaviour is built from **register transfers**.



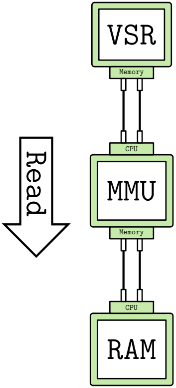

Modules are linked by **interfaces**.



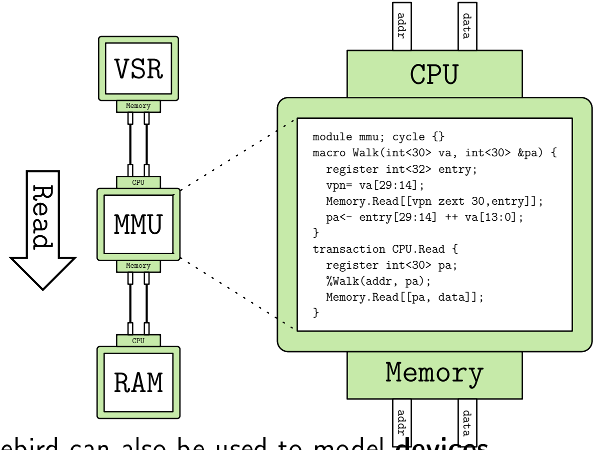

Interfaces define **transactions**.



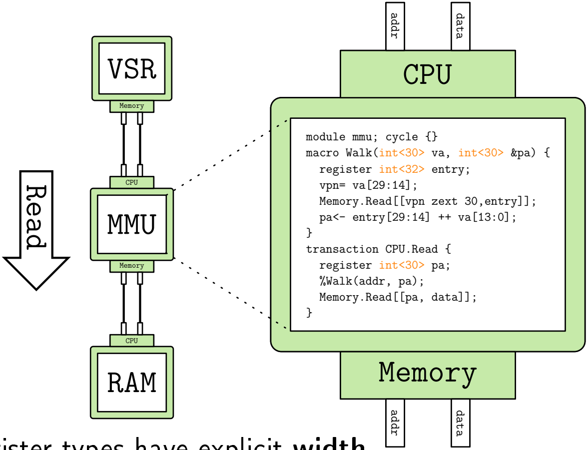

Transactions access the **datapath**.



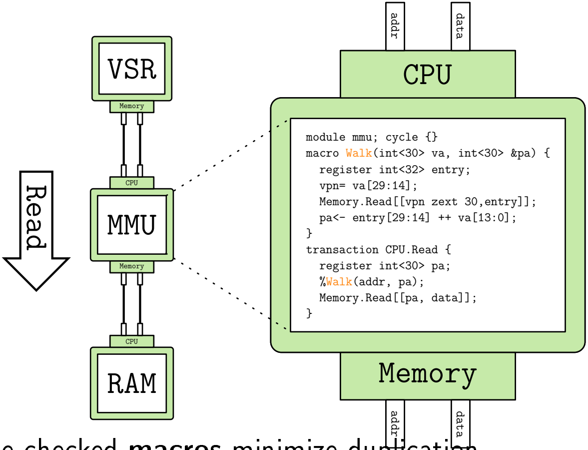

Interfaces and modules allow different implementations.



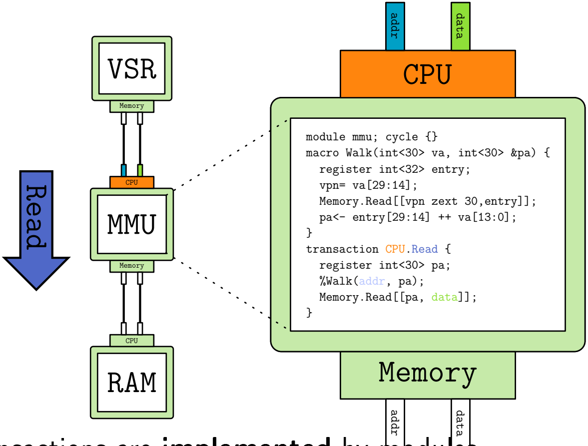

Lyrebird can also be used to model **devices**.




Register types have explicit **width**.




Type-checked **macros** minimize duplication.

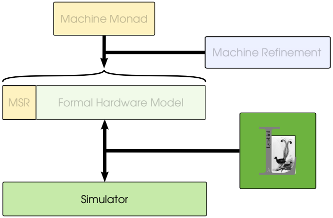



Transactions are **implemented** by modules.

# ARMv6 Model

- We have an ARMv6 user-level integer instruction model.
- Floating-point and vector operations are excluded.
- The complete model is approximately 1600 lines.
- We used it to validate the seL4 Haskell prototype.

# Simulation




Register transfer is easy to simulate.
The simulator is portable and fast — 10MIPS for ARMv6 user.
The output is a single C module;
It is easily incorporated into larger simulations.

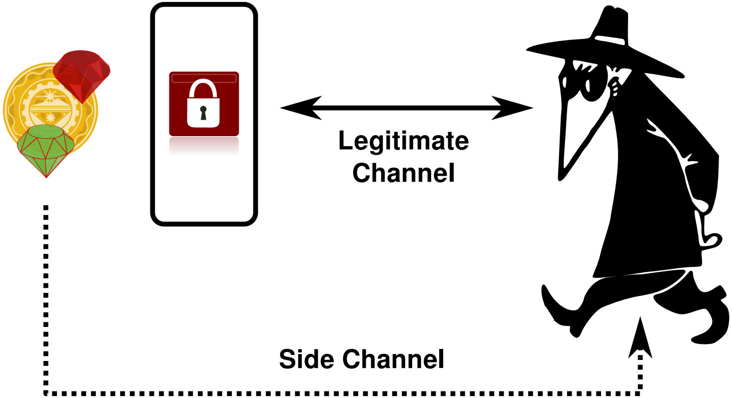
# Further Reading

For more see:

- *Lyrebird — assigning meanings to machines, SSV'10*

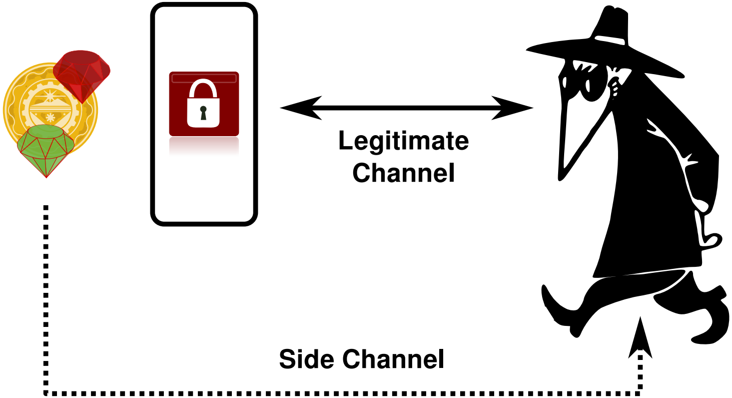
# Outline

# The Problem



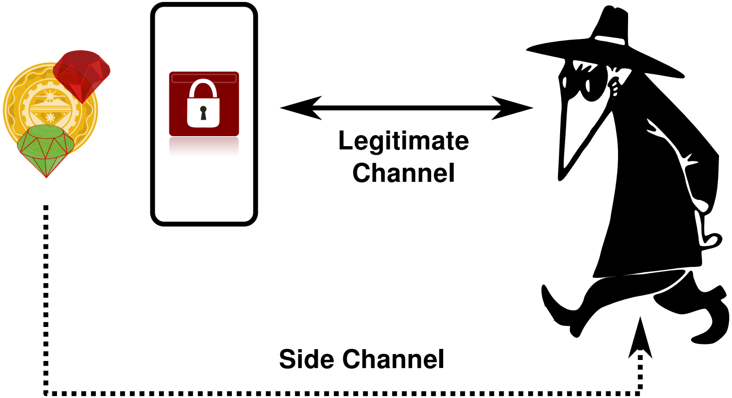

The attacker tries to guess the lock combination.

# The Problem



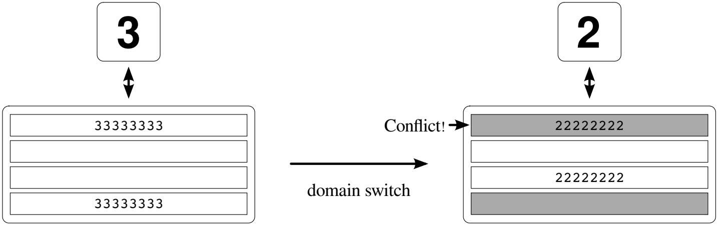

After $n$ tries he's locked out.

# The Problem

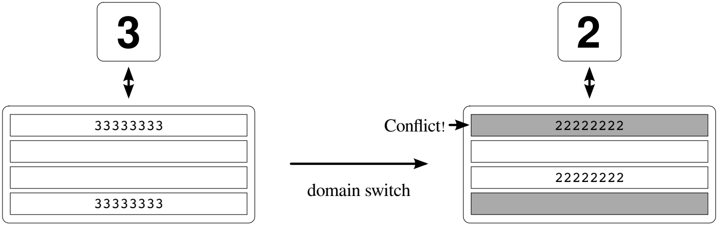



Every guess leaks something about the combination.

# The Cache Channel

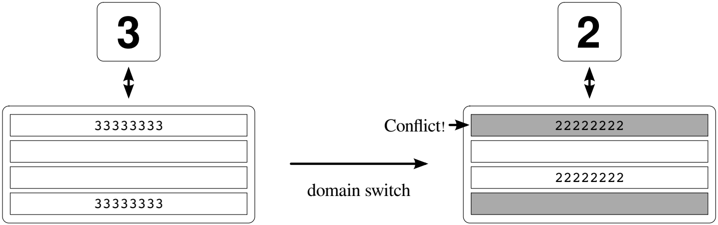


# The Cache Channel



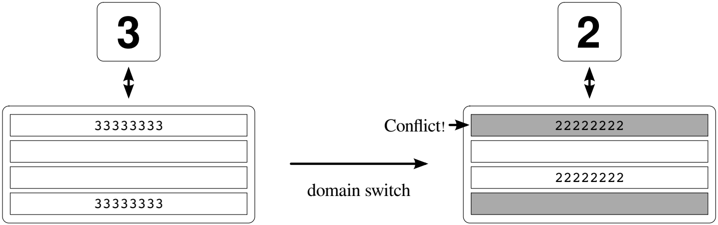

- It's easy to spot a cache miss.

# The Cache Channel



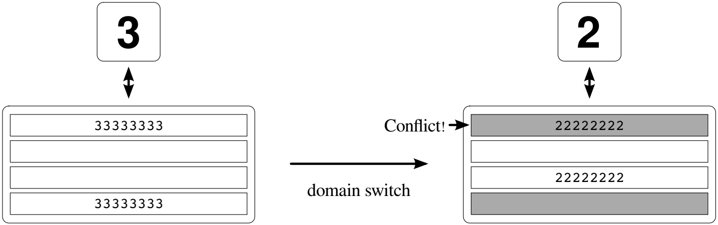

- It's easy to spot a cache miss.
- Cache contention forms a channel.

# The Cache Channel




- It's easy to spot a cache miss.
- Cache contention forms a channel.
- This is a big problem in crypto e.g. AES.

# The Cache Channel




- It's easy to spot a cache miss.
- Cache contention forms a channel.
- This is a big problem in crypto e.g. AES.

We ran a large empirical evaluation:

- 3 channels, 2 countermeasures and 5 platforms.
- 6 months of observations
- Integrated with regression tests.

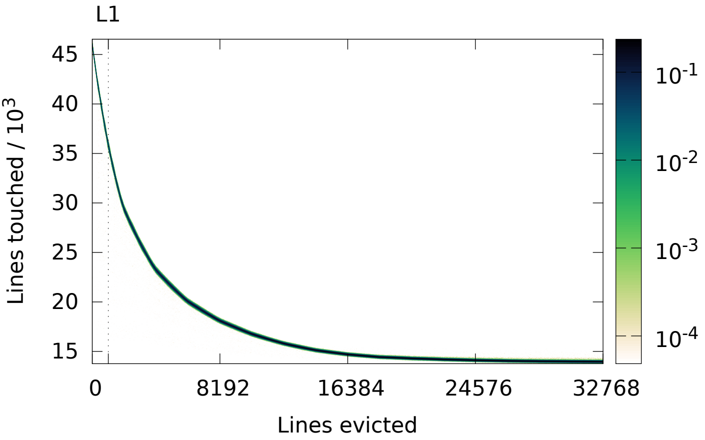
| | | | | | |
|---|---|---|---|---|---|
| Processor | iMX.31 | E6550 | DM3730 | AM3358 | Exynos4412 |
| Manufacturer | Freescale | Intel | TI | TI | Samsung |
| Architecture | ARMv6 | x86-64 | ARMv7 | ARMv7 | ARMv7 |
| Core type | ARM1136JF-S | Conroe | Cortex A8 | Cortex A8 | Cortex A9 |
| Released | 2005 | 2007 | 2010 | 2011 | 2012 |
| Cores | 1 | 2 | 1 | 1 | 4 |
| Clock rate | 532 MHz | 2.33 GHz | 1 GHz | 720 MHz | 1.4 GHz |
| Timeslice | 1 ms | 2 ms | 1 ms | 1 ms | 1 ms |
| RAM | 128 MiB | 1024 MiB | 512 MiB | 256 MiB | 1024 MiB |
| L1 D-cache | | | | | |
| size | 16 KiB | 32 KiB | 32 KiB | 32 KiB | 32 KiB |
| index | virtual | physical | virtual | virtual | virtual |
| tag | physical | physical | physical | physical | physical |
| line size | 32 B | 64 B | 64 B | 64 B | 32 B |
| lines | 512 | 512 | 512 | 1024 | 512 |
| associativity | 4 | 8 | 4 | 4 | 4 |
| sets | 128 | 64 | 128 | 128 | 256 |
| L2 cache | | | | | |
| size | 128 KiB | 4096 KiB | 256 KiB | 256 KiB | 1024 KiB |
| line size | 32 B | 64 B | 64 B | 64 B | 32 B |
| lines | 4096 | 65,536 | 4096 | 4096 | 32,768 |
| associativity | 8 | 16 | 8 | 8 | 16 |
| sets | 512 | 4096 | 512 | 512 | 2048 |
| colours | 4 | 64 | 8 | 8 | 16 |

Table: Experimental platforms.

| | | | | | |
|---|---|---|---|---|---|
| Processor | iMX.31 | E6550 | DM3730 | AM3358 | Exynos4412 |
| Manufacturer | Freescale | Intel | TI | TI | Samsung |
| Architecture | ARMv6 | x86-64 | ARMv7 | ARMv7 | ARMv7 |
| Core type | ARM1136JF-S | Conroe | Cortex A8 | Cortex A8 | Cortex A9 |
| Released | 2005 | 2007 | 2010 | 2011 | 2012 |
| Cores | 1 | 2 | 1 | 1 | 4 |
| Clock rate | 532 MHz | 2.33 GHz | 1 GHz | 720 MHz | 1.4 GHz |
| Timeslice | 1 ms | 2 ms | 1 ms | 1 ms | 1 ms |
| RAM | 128 MiB | 1024 MiB | 512 MiB | 256 MiB | 1024 MiB |
| L1 D-cache | | | | | |
| size | 16 KiB | 32 KiB | 32 KiB | 32 KiB | 32 KiB |
| index | virtual | physical | virtual | virtual | virtual |
| tag | physical | physical | physical | physical | physical |
| line size | 32 B | 64 B | 64 B | 64 B | 32 B |
| lines | 512 | 512 | 512 | 1024 | 512 |
| associativity | 4 | 8 | 4 | 4 | 4 |
| sets | 128 | 64 | 128 | 128 | 256 |
| L2 cache | | | | | |
| size | 128 KiB | 4096 KiB | 256 KiB | 256 KiB | 1024 KiB |
| line size | 32 B | 64 B | 64 B | 64 B | 32 B |
| lines | 4096 | 65,536 | 4096 | 4096 | 32,768 |
| associativity | 8 | 16 | 8 | 8 | 16 |
| sets | 512 | 4096 | 512 | 512 | 2048 |
| colours | 4 | 64 | 8 | 8 | 16 |

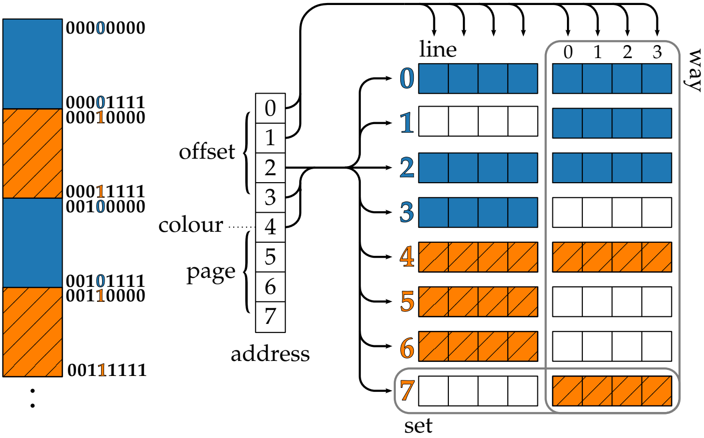
Table: Experimental platforms.

# Exynos4 Cache Channel

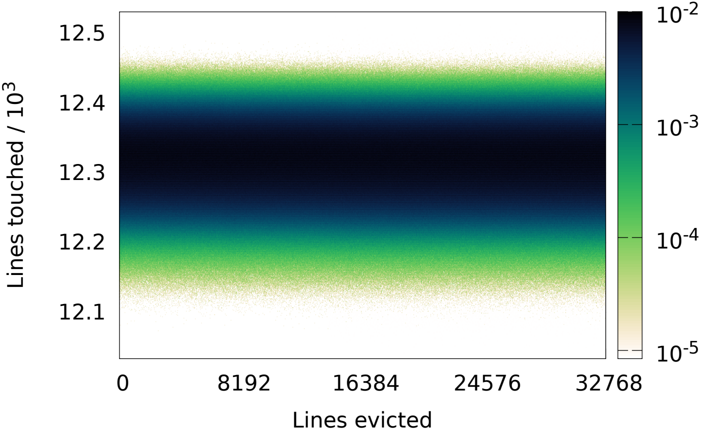



Bandwidth: 7.04kb/s

# Cache Colouring

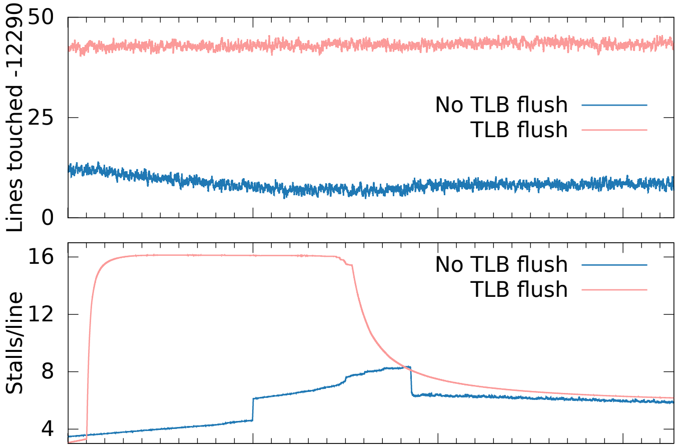


# Coloured Cache Channel




Bandwidth: 81.3b/s

# Residual TLB Channel



# The Lucky-13 Attack

This is a recent vulnerability in OpenSSL TLS.

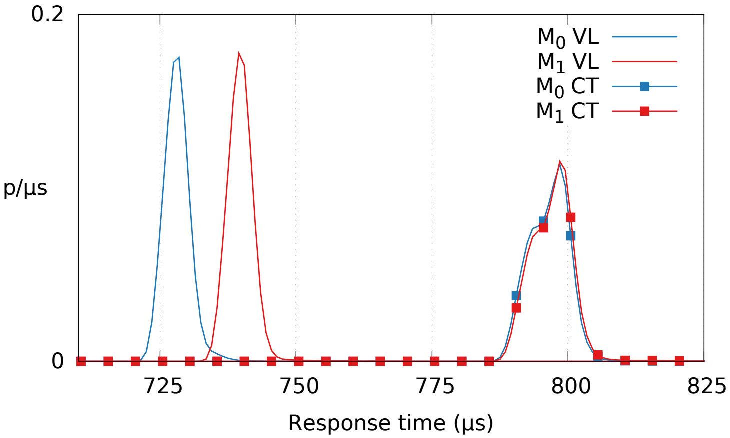
- Runtime depends on *unvalidated* user input.
- Can be used as a decryption oracle.
- ‘Fixed’ with a constant-time algorithm.
- We reproduced the attack on seL4...

# The Lucky-13 Attack

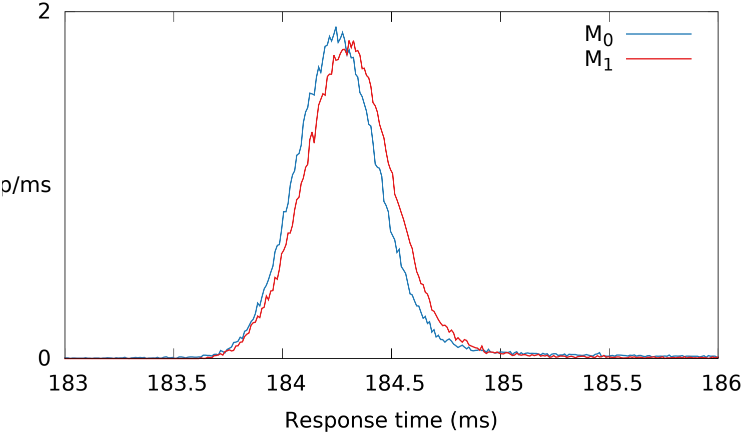
This is a recent vulnerability in OpenSSL TLS.

- Runtime depends on *unvalidated* user input.
- Can be used as a decryption oracle.
- 'Fixed' with a constant-time algorithm.
- We reproduced the attack on seL4...
- ...and fixed it with better performance!
- Required no modifications to OpenSSL.

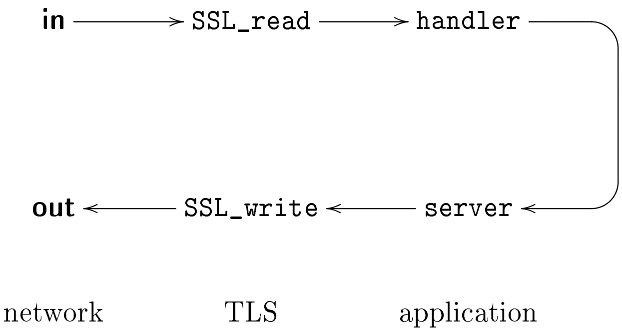
# The Lucky-13 Attack



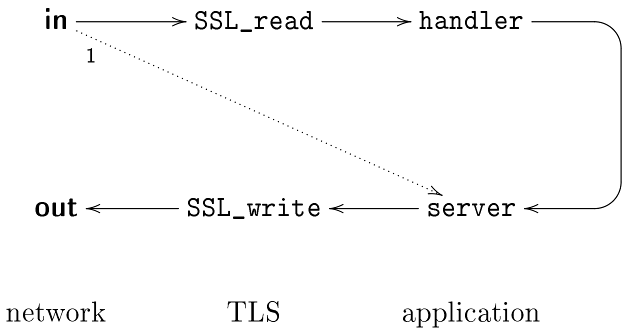
# Intercontinental Attack



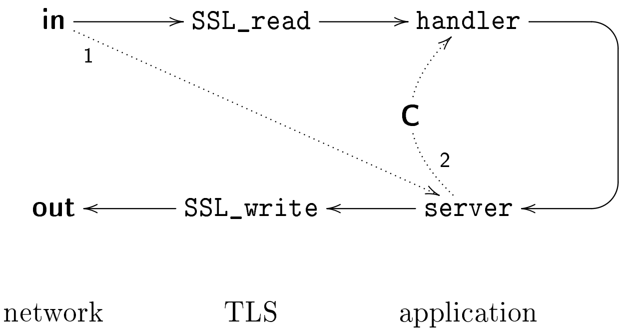
# Scheduled Delivery



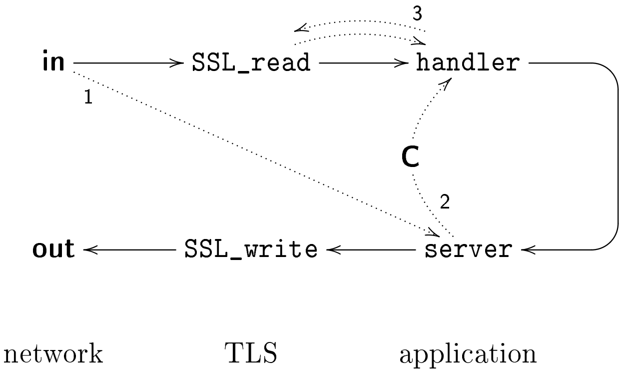
# Scheduled Delivery



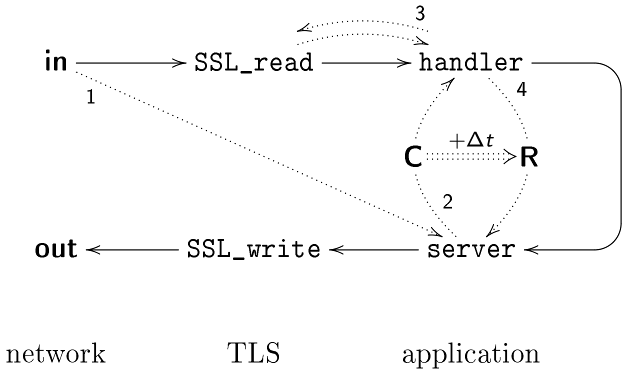
# Scheduled Delivery



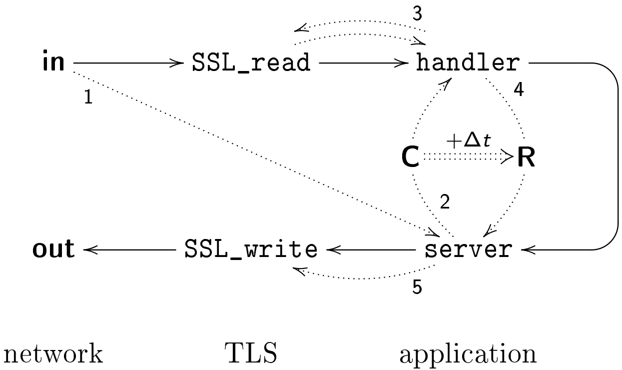
# Scheduled Delivery



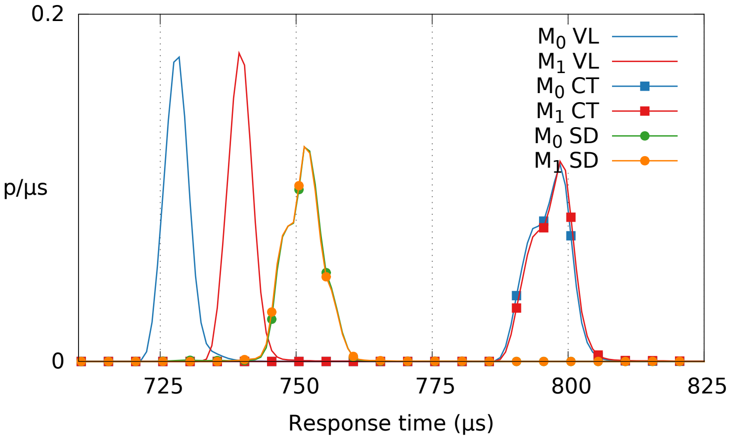
# Scheduled Delivery



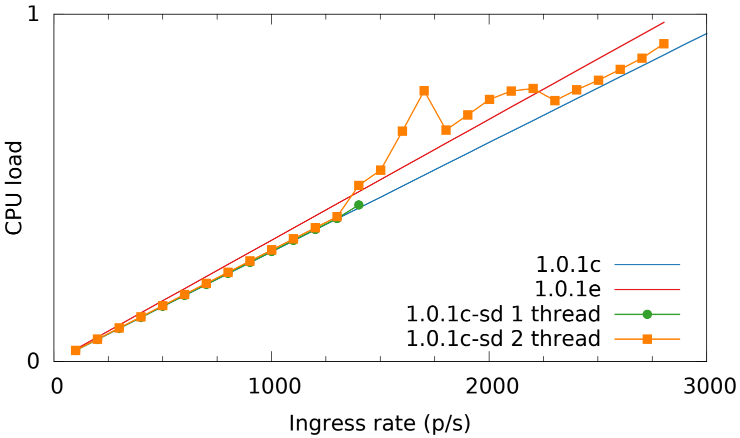
# Scheduled Delivery



# Lucky-13 Mitigated



# Load Performance

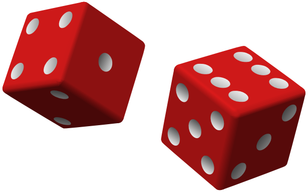


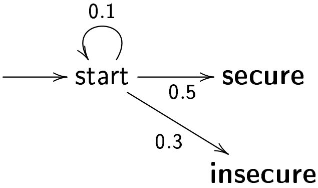
# Further Reading

For more see:

- *Exploitation as an inference problem, AISEC,11.*
- *The Last Mile: An Empirical Study of Some Timing Channels on seL4, CCS'13.*

# pGCL






- pGCL is a language of probabilistic automata.
- It models both demonic and probabilistic choice.
- My Isabelle/HOL formalisation is now in the Archive of Formal Proofs.
- Used to formally verify probabilistic security properties e.g. side channel leakage.

# pGCL

For more see:

- *Verifying probabilistic correctness in Isabelle with pGCL, SSV'12*
- *From probabilistic operational semantics to information theory - side channels with pGCL in isabelle, ITP'14*
- *pGCL for Isabelle, Archive of Formal Proofs, 2014*

# Outline

# Can We Verify the C Library?

An open project:

- Work in a public repository.
- Code only accepted with proof.
- Self-contained student projects.

# Can We Verify the C Library?

An open project:

- Work in a public repository.
- Code only accepted with proof.
- Self-contained student projects.

Applications:

- Systems on a verified kernel.
- Library spec for symbolic execution (no tracing libc!).
- Verified compiler (CompCert) needs a verified runtime.

# Getting Value out of FM

You don't have to do seL4 to benefit from FM:

- Go for bang/buck.
- Focus on things likely to be wrong.
- Provide a toolset to programmers.

# Getting Value out of FM

You don't have to do seL4 to benefit from FM:

- Go for bang/buck.
- Focus on things likely to be wrong.
- Provide a toolset to programmers.

DSLs provide a convenient interface:

- We've seen examples: Bitfields, Lyrebird, ...
- Match to tool to the job.
- Full formalism isn't exposed to programmers.
- Don't force everything into a single framework: provide tools!

# Outline

# Questions?