NICTA

Operating System Verification for Real Use

Gernot Heiser
NICTA and UNSW and Open Kernel Labs
Sydney, Australia

Australian Government
Department of Broadband, Communications and the Digital Economy
Australian Research Council

NICTA Members

NICTA Partners


Windows
An exception 06 has occured at 0028:C11B3ADC in VxD DiskTSD(03) + 00001660. This was called from 0028:C11B40C8 in VxD voltrack(04) + 00000000. It may be possible to continue normally.
* Press any key to attempt to continue.
* Press CTRL+ALT+RESET to restart your computer. You will lose any unsaved information in all applications.
Press any key to continue

OS Reliability is a Problem

- The consequences of OS failure are getting worse

OS Reliability is a Problem

- Any computer system can only be as reliable/trustworthy as its operating system
- Life-critical systems should have *bullet-proof operating systems*
- Formal verification should be the obvious way to achieve this?
- How does military-grade *assurance* look like?

Common Criteria and Verification

EAL	Requirem.	Funct Spec	HLD	LLD	Implem.
EAL 1	Informal	Informal	Informal	Informal	Informal
EAL 2	Informal	Informal	Informal	Informal	Informal
EAL 3	Informal	Informal	Informal	Informal	Informal
EAL 4	Informal	Informal	Informal	Informal	Informal
EAL 5	Formal	Semiformal	Semiformal	Informal	Informal
EAL 6	Formal	Semiformal	Semiformal	Semiformal	Informal
EAL 7	Formal	Formal	Formal	Semiformal	Informal

→**No certainty of implementation correctness ⇒ not good enough!**

Operating System Verification

OS Verification — Is It Feasible?

- Benefits seem clear, but
  - Can it be done?
  - Can it be done for a system suitable for real use?
  - Is it doable at a reasonable cost?
- Past attempts:
  - UCLA Secure Unix (1980)
    - Pascal kernel, 90% specifications, 20% implementation proofs
  - PSOS (1973–83)
    - 17 specification layers (hardware to apps)
    - some refinement proofs between layers
    - no implementation proofs completed
  - KIT (1987)
    - minimal kernel, some 100 LOC assembly
    - full implementation proof
    - very basic services on idealised machine

Aim of NICTA OS Verification Work

- Formally verify a complete OS kernel
- ... which is suitable for commercial use
  - on real hardware
  - fully functional
  - good performance

Main Challenges

- Size:
  - operating systems tend to be big
  - Linux, Windows: millions of lines of code (LOC)
- Ugly code: low-level and unsafe
  - Low-level language — C is the *lingua franca* of OS
    - could be worse — some use C++
  - Frequently has to bypass type-checking
    - hardware registers are untyped
    - various efficiency tricks, bit fiddling
  - Side effects are unavoidable — hardware is that way
  - Assembler code is unavoidable
    - and used voluntarily for efficiency
- Efficiency: if it's slow, no-one will use it

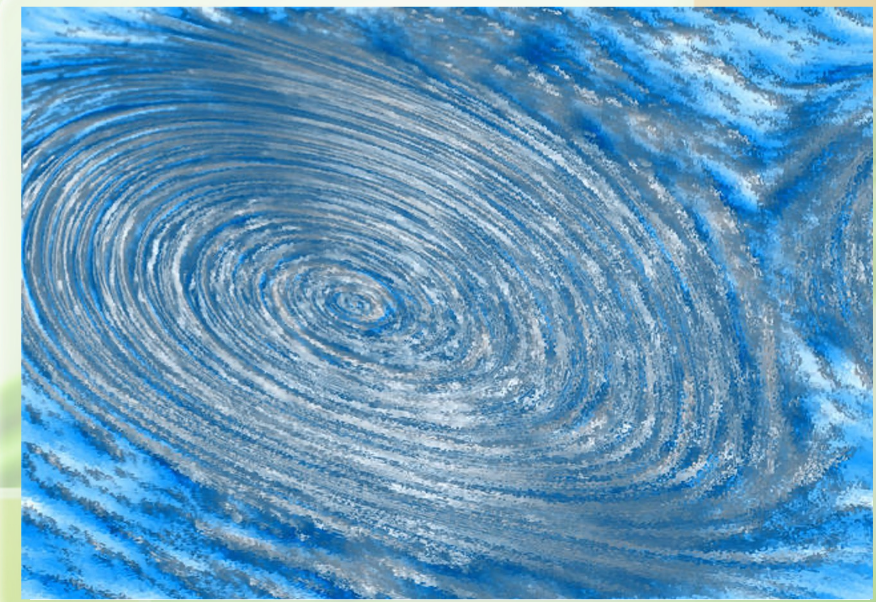
Prerequisites for Success

- Need a target system that is tractable
  - ☐10,000 LOC is about the limit (so they tell me)
- Need a world-class formal-methods team
  - so I suspect, based on historic track record...
- Need a world-class OS team
  - ... or they won't end up with a useful kernel
  - really correct and really slow won't do!
- Need to ensure communication between teams!
  - only practical safeguard against the two teams diverging

With hindsight:

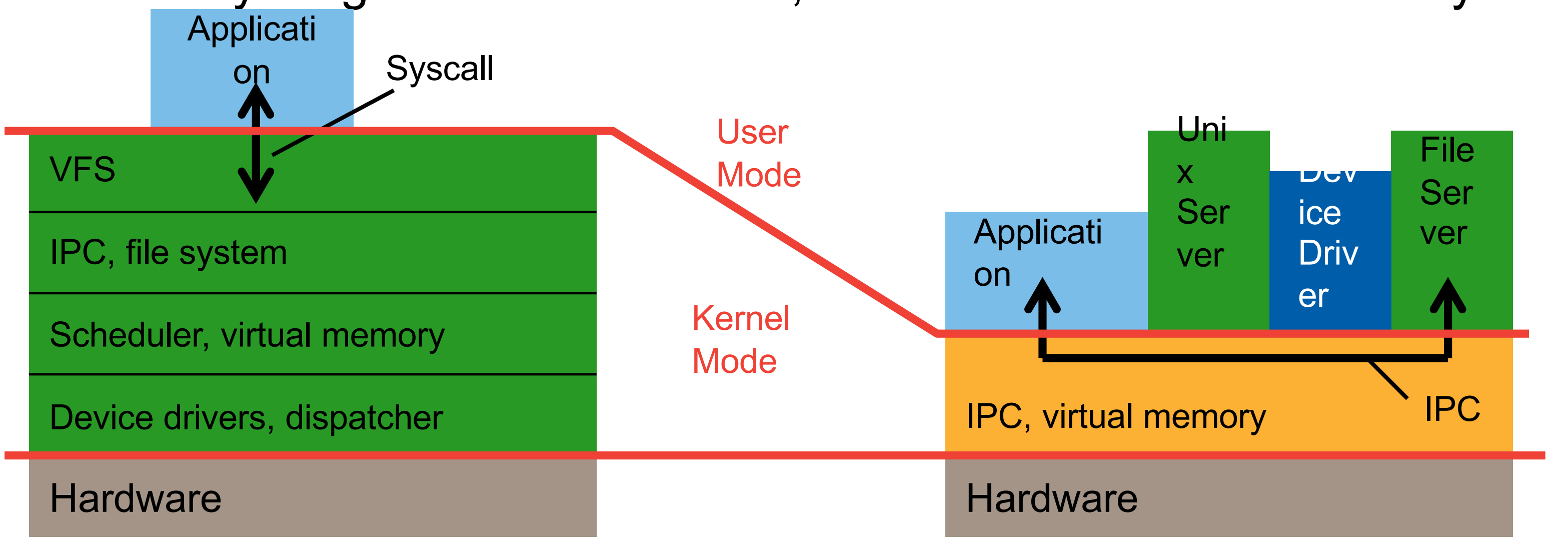
- Need control over tools!

The Target



Tractable System

- 10kLOC and practically usable ⇒ microkernel
  - Very small platform that allows constructing arbitrary systems on top
  - OS functionality reduced to its essence:
    - fundamental mechanisms
    - no policies
  - Only microkernel runs in privileged mode
    - everything else in user mode, incl "normal" OS functionality



Tractable System: L4 Microkernel

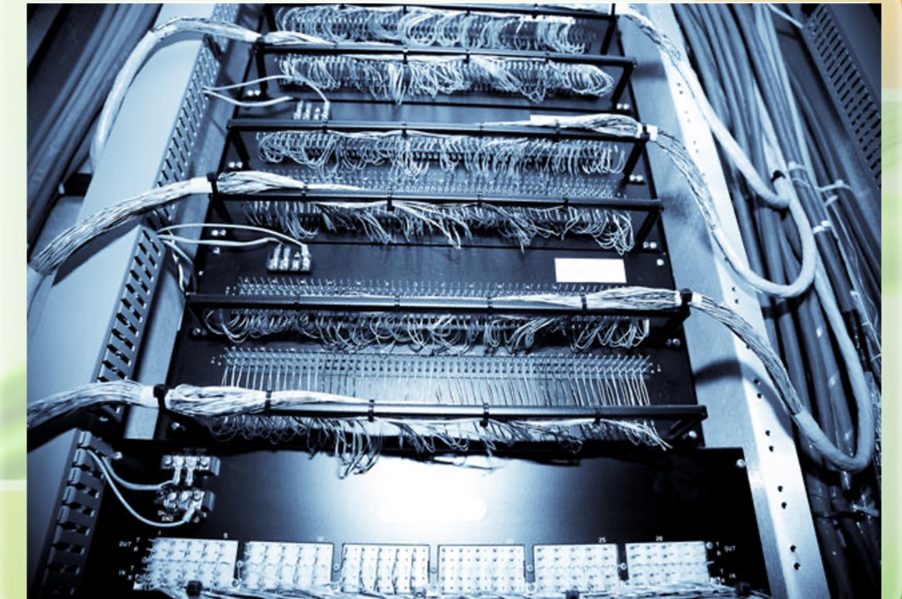
- Right size: ☐10,000 LOC
  - mostly C, some (100's LOC) assembler
- Very general-purpose: supports
  - componentised OSes
  - small embedded environments
  - virtualized main-stream OSes (Linux)
- Famous for performance
  - the benchmark for microkernel performance
- Ready for commercial use
  - ... or so we hoped at the time
  - now deployed on some 150,000,000 devices
- However, not fully suitable
  - several security issues:
    - inefficient communication control mechanisms
    - isolation of user domains broken by management of kernel resources
  - needed API overhaul

The Challenge

- Formally verify a secure version of L4
  - Need to develop a new kernel API as we go
  - Concurrent to verification effort
- Outcome to be suitable for practical use
  - Performance within 10% of existing high-performance L4

- Succeed!
  - Unsuccessful multi-M$ project doesn't look good
  - Some careers were on the line

The Ingredients



Formal-Methods Team

- Gerwin Klein — newly recruited researcher
- Project leader verification
  - *L4.verified* project
- Developed project plan
- Built team of up to 10 people across two cities

OS Team

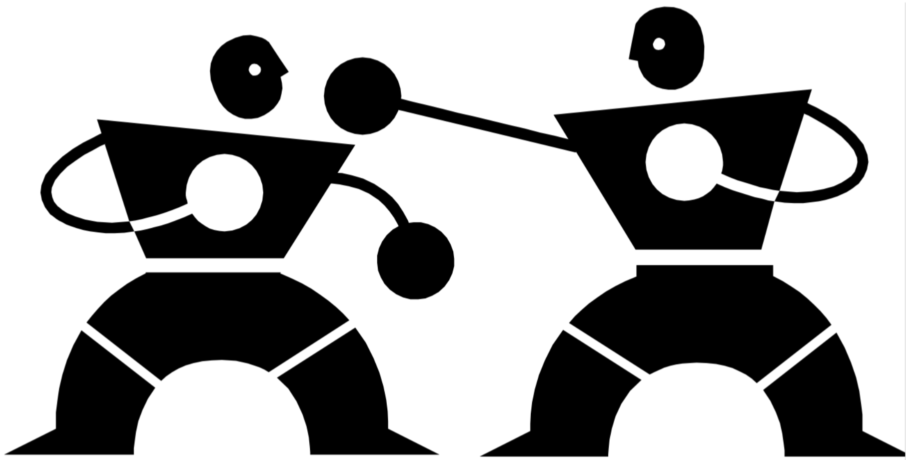
- Kevin Elphinstone — recent recruitment to UNSW
- Project leader kernel API
  - *seL4* project
- Built (smaller) kernel team
- Lead design and implementation of new L4 API

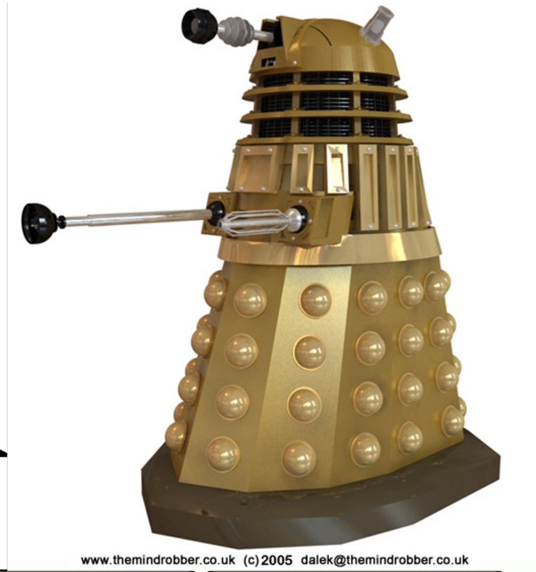
The Interface Part 1: The Human Side

- Needed at least one person I could trust to understand both sides
- Don't have one? Need to bake your own!
  - Take one top class systems hacker with Maths aptitude
  - Let Gerwin work on him for a year
- The result?

- Approach tested in 1-year pilot project
  - Seemed to work, resulting in 3-year main project
  - Learned some lessons on approach
  - Developed idea of overall effort required

The Interface Part 2: The Language



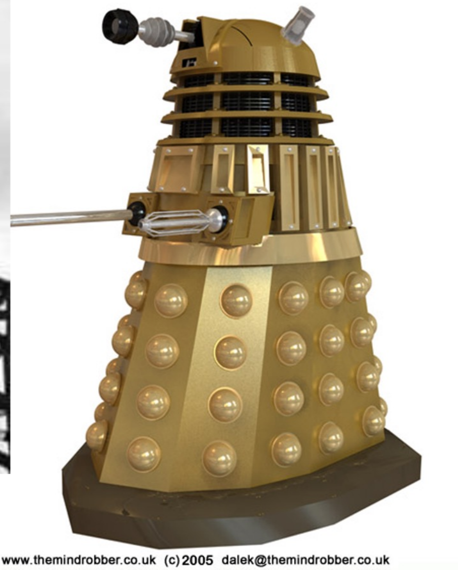


Abstract Model

C Code | HW

Where is the common ground?

Bridging the Gap

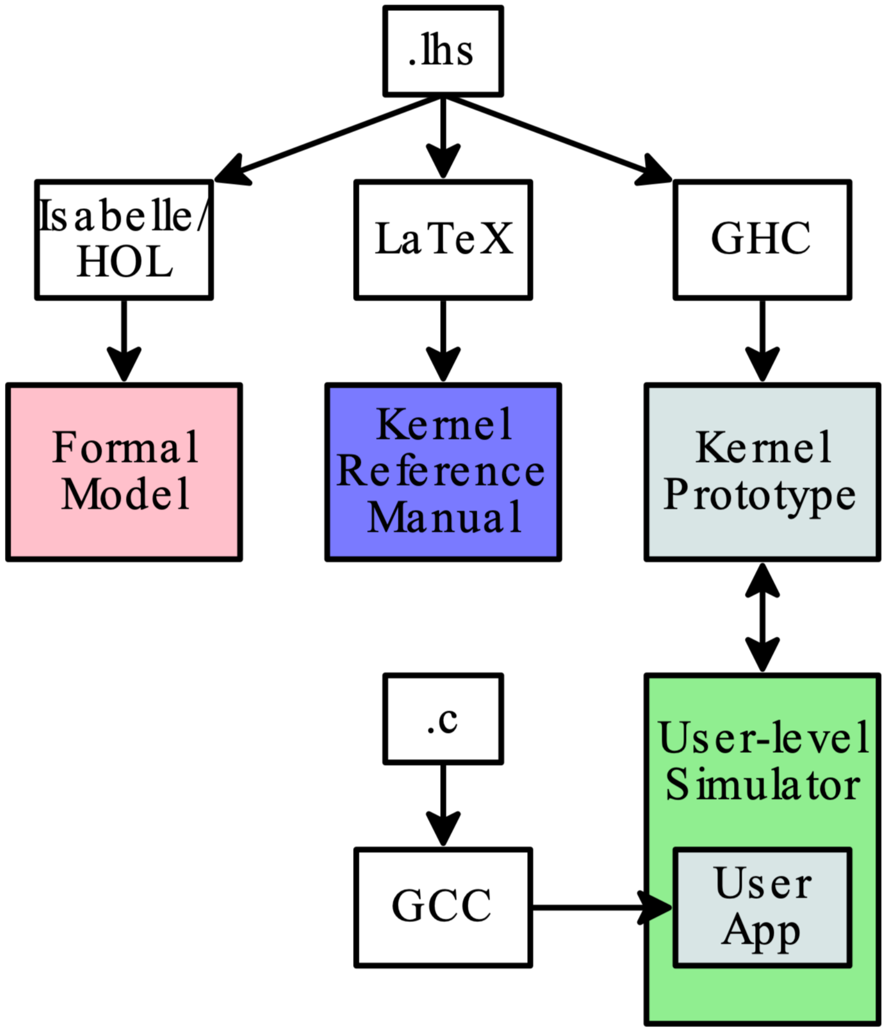


Modelling?

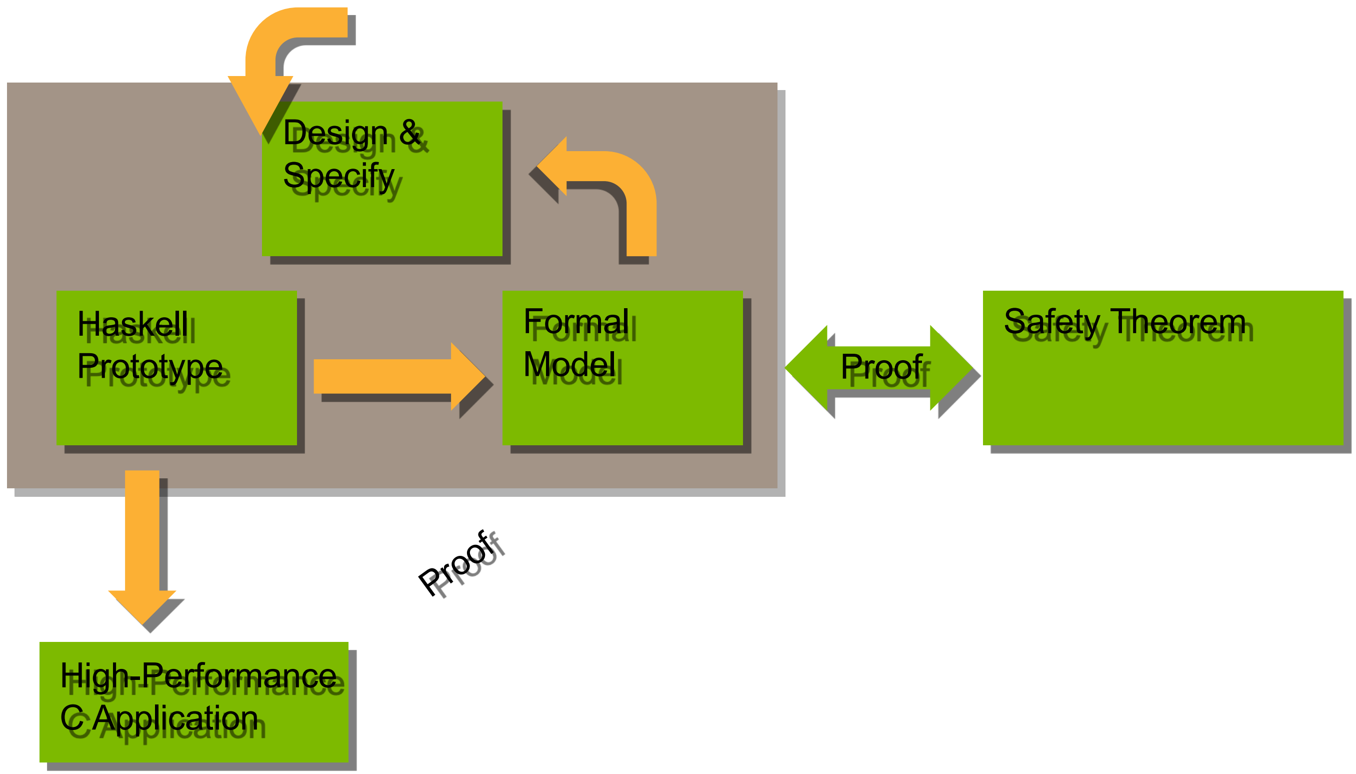
- Well defined semantics
- Readily formalisable
- Exposed implementation details
- Programming language

Haskell — The New *Lingua Franca*?

- Used *Literate Haskell* as modelling language
  - pure functional language
  - embedded documentation
  - close to Isabelle/HOL
- Familiar to most kernel hackers
  - First-year teaching language at UNSW
- Executable
  - Supports running user-level code
  - Useful for exercising the API
    - gain experience with API
    - port user-level software
- Used to model kernel in detail



Iterative Design and Formalisation



- Haskell kernel executes native binaries on simulator
- Exposes usability issues early
- Tight formal design integration

Kernel Modelling

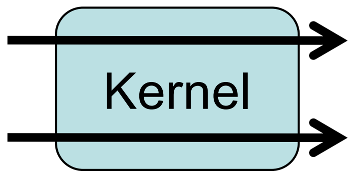
Kernel Modelling in Haskell

Kernel API is event-based, mostly atomic

- Kernels are big state machines with events as input
  - Imperative
  - Rely on side-effects all the time
    - P(s), make_runnable(tcb)
- Kernels manipulate the low-level machine
  - Interrupts, TLBs, caches

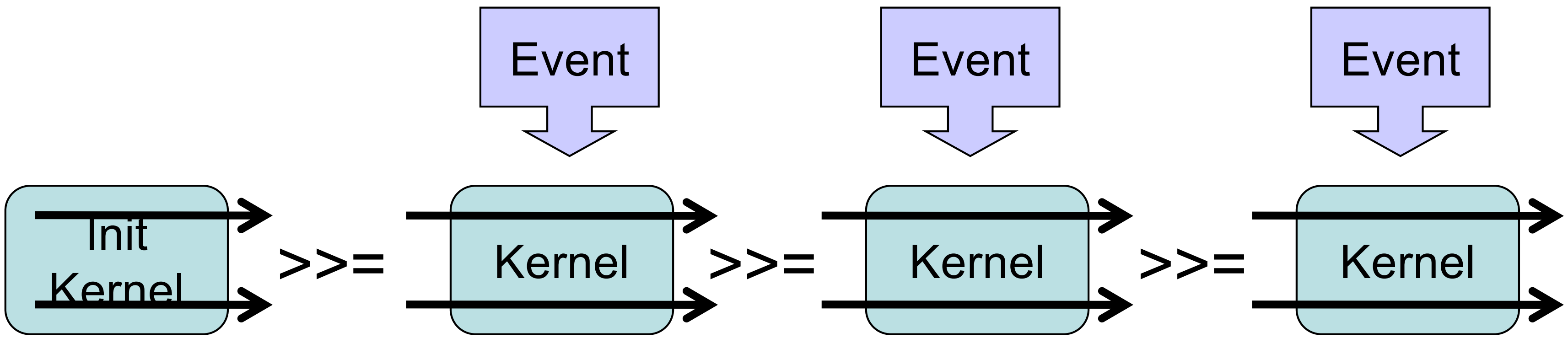
- Preemption required
  - Kernels can't always perform operations to completion

Kernel Code in a State Monad



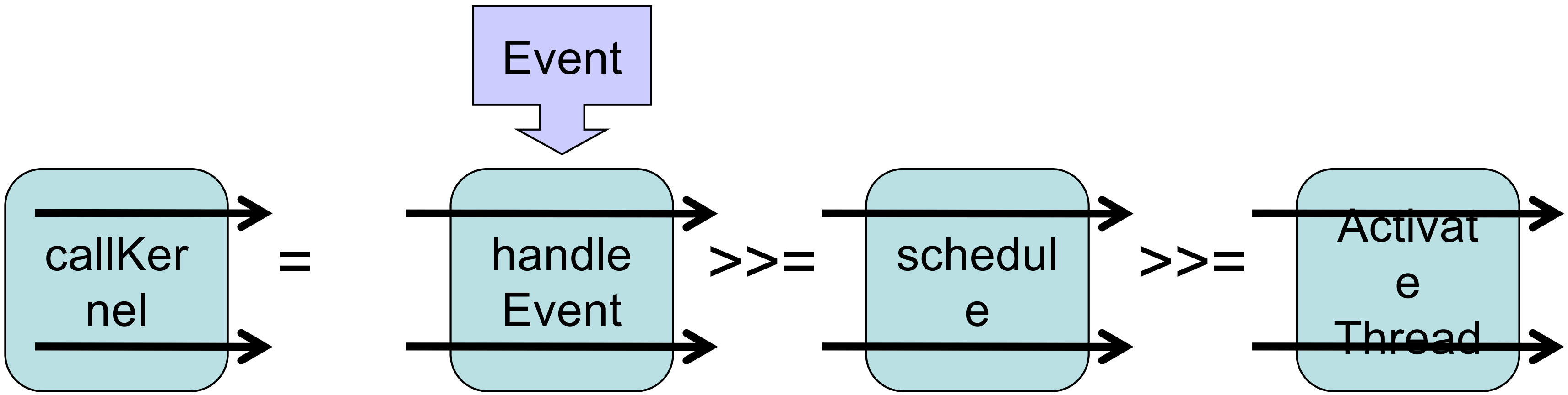
- State monads are *units* of computation which consume and produ
  - state transformers
- Kernel monad encapsulates a state transformer of the kernel and

- Monads can be bound together using the *bind* operator
  - sequencing the computation
  - connects the plumbing to pass the state along



Kernel Code in a Monad

```
type Kernel = StateT KernelState MachineMonad
callKernel :: Event -> Kernel ()
callKernel ev =
    handleEvent ev >>= (\x -> schedule >>=
                              (\y -> activateThread))
```

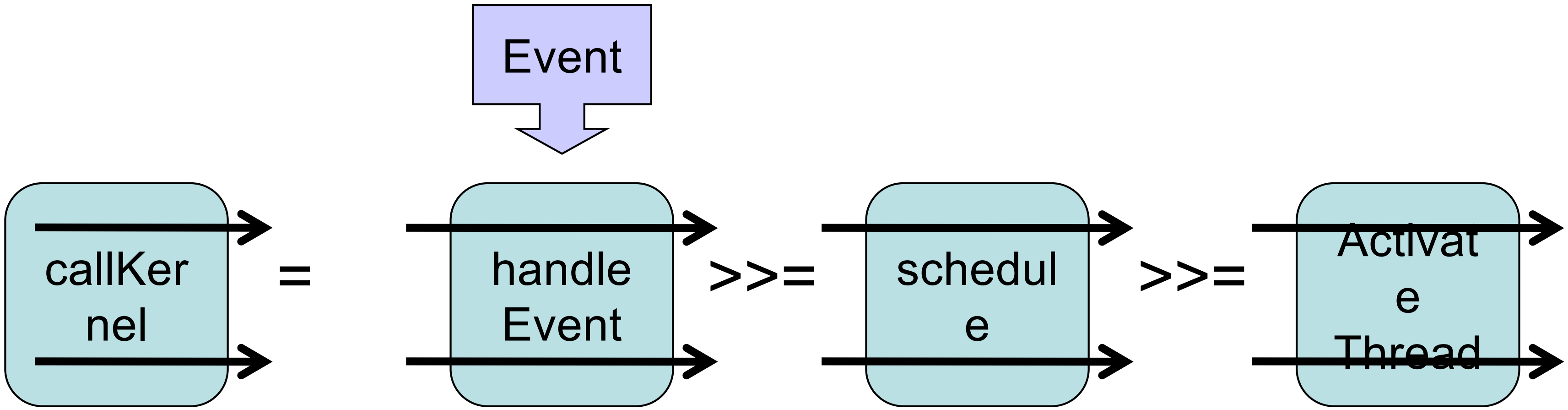


Kernel Code in a Monad

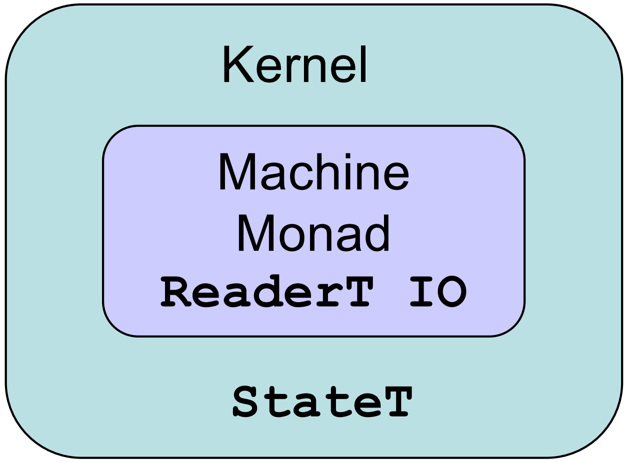
```
type Kernel = StateT KernelState MachineMonad
callKernel :: Event -> Kernel ()
callKernel ev = do
    handleEvent ev
    schedule
    activateThread
```

Imperative in “style”

Lowers barrier to entry for kernel developers



Kernel Monad



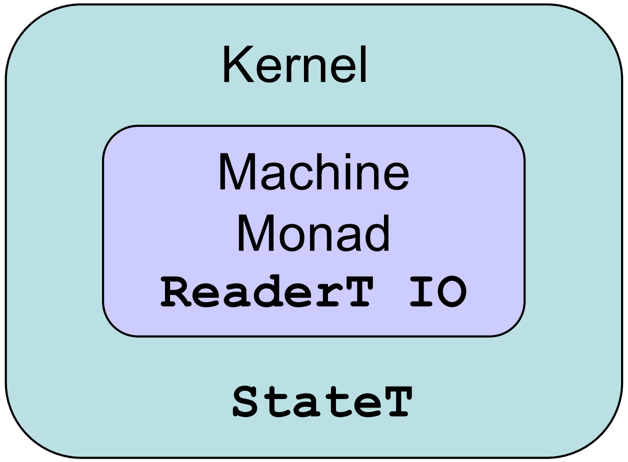
- Machine monad contains state to interface to
- Kernel contains the state of physical memory

Machine Monad — Lowest Level of Model

- `getMemoryTop :: MachineMonad (PPtr ())`
- `getDeviceRegions :: MachineMonad [(PPtr (), Int)]`
- `loadWord :: PPtr Word -> MachineMonad Word`
- `storeWord :: PPtr Word -> Word -> MachineMonad ()`
- `insertMapping :: PPtr Word -> VPtr -> Int -> Bool ->`
- `flushCaches :: MachineMonad ()`
- `getActiveIRQ :: MachineMonad (Maybe IRQ)`
- `maskInterrupt :: Bool -> IRQ -> MachineMonad ()`
- `ackInterrupt :: IRQ -> MachineMonad ()`
- `waitForInterrupt :: MachineMonad IRQ`
- `configureTimer :: MachineMonad IRQ`
- `resetTimer :: MachineMonad ()`

- Foreign Function Interface (FFI)
- Approximate machine-level C functions
- Close to “real” as possible
  - Forces us to manage “hardware”

KernelState Monad



- Statically allocated global kernel data
  - Current thread
  - Scheduler queues
- Physical Memory

The Proof

NICTA

```
tcb_t * scheduler_t::find_next_thread(prio_queue_t * prio_queue)
{
    ASSERT(DEBUG, prio_queue);

    if (prio_queue->index_bitmap) {
        word_t top_word = msb(prio_queue->index_bitmap);
        word_t offset = BITS_WORD * top_word;

        for (long i = top_word; i >= 0; i--)
        {
            word_t bitmap = prio_queue->prio_bitmap[i];

            if (bitmap == 0)
                goto update;

            do {
                word_t bit = msb(bitmap);
                word_t prio = bit + offset;
                tcb_t *tcb = prio_queue->get(prio);
```

```
hread
ooseThread
```

Abstract Model

Manual System Specification

(Isabelle/HOL)

Formal proof: concrete behaviour captured at abstract level

Monadic functional programs

Executable Model

Haskell Prototype

Hoare Logic
Separation Logic

C Code

HW

High Performance Implementation (C/asm)
Hardware model

Common Criteria and L4.verified

EAL	Requirem.	Funct Spec	HLD	LLD	Implem.
EAL 1	Informal	Informal	Informal	Informal	Informal
EAL 2	Informal	Informal	Informal	Informal	Informal
EAL 3	Informal	Informal	Informal	Informal	Informal
EAL 4	Informal	Informal	Informal	Informal	Informal
EAL 5	Formal	Semiformal	Semiformal	Informal	Informal
EAL 6	Formal	Semiformal	Semiformal	Semiformal	Informal
EAL 7	Formal	Formal	Formal	Semiformal	Informal
L4.verified	Formal	Formal	Formal	Formal	*Formal*

Lessons Learned

The Tools

- Isabelle/HOL generally worked well
  - Gerwin's experience clearly helped
- But we were frequently pushing the boundaries
  - of techniques
  - of tools
- Crucially important to have control over tools
  - Need to be able to fix a limitation you run into
  - Source-code access is essential
  - Open-source tool is ideal
- Good support from tool supplier helps massively
  - The TUM folks were great!

It's Not Quite Over Yet!

- Refinement to low-level design (Haskell) complete
  - most formally analysed general-purpose kernel
- source-code level refinement in progress
  - due December '08
  - no-one doubts that it will succeed
- Work on proving security properties on-going

Verification Summary

Statistics

- 3.5 kLOC abstract, 7 kLOC concrete spec (about 3k Haskell)
- Abstract to Haskell: 100 kLOP (more features coming)
- Access control model + initial security proofs: 1 kLOP
- Haskell to C/asm: expect 80kLOP
- 109 patches to Haskell kernel, 132 to abstract spec
- Performance in line with other L4 kernels

Kinds of properties proved

- Well typed references, aligned objects, ..
- Well formed thread states, endpoint and scheduler queues, ...
- All syscalls terminate, reclaiming memory is safe, ...
- Authority is distributed by caps only
- Access control is decidable

Proof Maintenance

- Challenge: adapting proofs to changes in implementation
  - Will minor changes result in massive reworking of proofs?
- Inevitably tested as a result of project structure:
  - concurrent work on proofs and kernel design
  - frequently verification work progressed on frozen kernel
  - merging of source trees required update of proofs
- Experience: depends on how changes affect invariants
  - Some changes took weeks to port
    - new syscall,
    - additional parameters to syscall decoded deep down
  - Others, breaking existing invariants, took months
    - fundamentally changing operation of IPC, eg. reply caps
    - required discovering new invariants
- Experience increases confidence in practicability of OS verification

Cost: Is OS Verification Affordable?

- Estimated cost of complete project: A$4–5M
  - seL4 and L4.verified combined, until December '08
- Estimated cost of re-doing on latest kernel: A$2M
  - on commercial OKL4 kernel
- Cost of traditional assurance: US$10k/LOC
  - industry estimate for Common Criteria EAL6 certification
  - means *US$100M* for L4-like kernel!
- Challenge: Convince authorities that verification is superior!

Conclusion

- Complete verification of a fully-functional OS kernel seems doable
- Cost seems small compared to traditional assurance schemes
- However, probably can't succeed without:
  - top-notch verifiers
  - top-notch kernel experts
  - excellent communication between the two sides
    - need some people who understand both
    - good languages and tools help
- A microkernel is only the start!
  - Need to work on actual OS services
  - Multiple independent levels of security (MILS)

The Team

- **Gerwin Klein**
- June Andronick
- David Cock
- Philip Derrin
- Kai Engelhardt
- Jia Meng
- Michael Norrish
- David Tsai
- Simon Winwood

- **Kevin Elphinstone**
- Andrew Boyton
- Jeremy Dawson
- Dhammika Elkaduwe
- Rafal Kolanski
- Catherine Menon
- Thomas Sewell
- Harvey Tuch

NICTA

From **imagination** to **impact**