

seL4

Formal Verification of an OS Kernel

Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick,
David Cock, Philip Derrin, Dhammadika Elkaduve, Kai Engelhardt,
Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch,
Simon Winwood (SOSP '09)

Presented by Ernest Ng (CS 6410)



(seL4 call graph, Klein et al. TOCS '10)

CrowdStrike Outage (July 2024)



CrowdStrike IT outage affected 8.5 million Windows devices, Microsoft says

Billions in Damages From CrowdStrike Outage to Go Uninsured

- Insured losses estimated between \$300 million and \$1.5 billion
- The days-long outage took a \$5.4 billion toll on Fortune 500

(Images from BBC & Bloomberg)

“The sensor expected 20 input fields, while the update provided 21 input fields ... the mismatch resulted in an **out-of-bounds memory read**, causing a system crash.”

- CrowdStrike Root Cause Analysis Report (Aug 2024)

OS Verification — Now!

Harvey Tuch

Gerwin Klein

Gernot Heiser

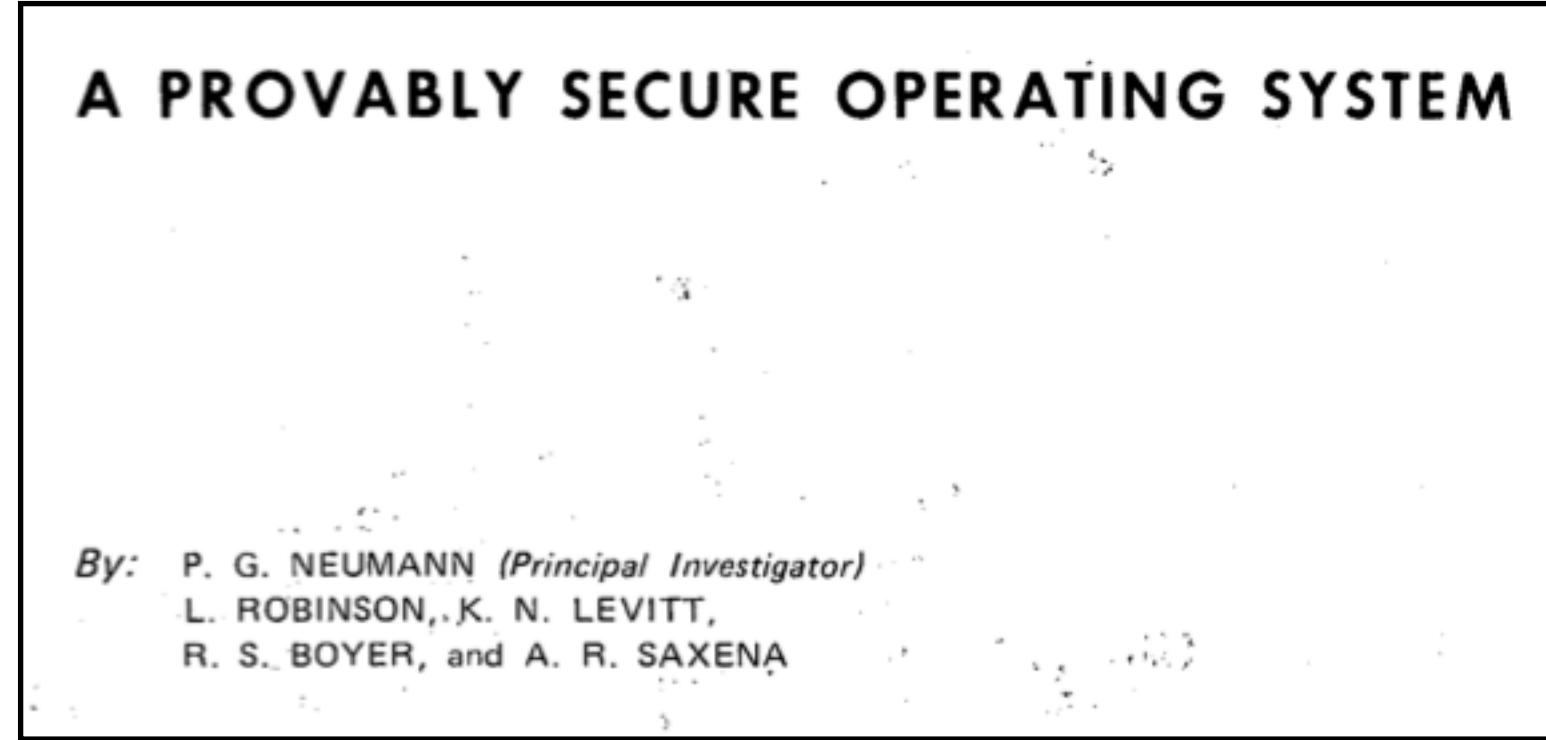
National ICT Australia

University of New South Wales

HotOS '05
(from the creators of seL4)

Formally verified OSes, a history

PSOS (Stanford), 1973



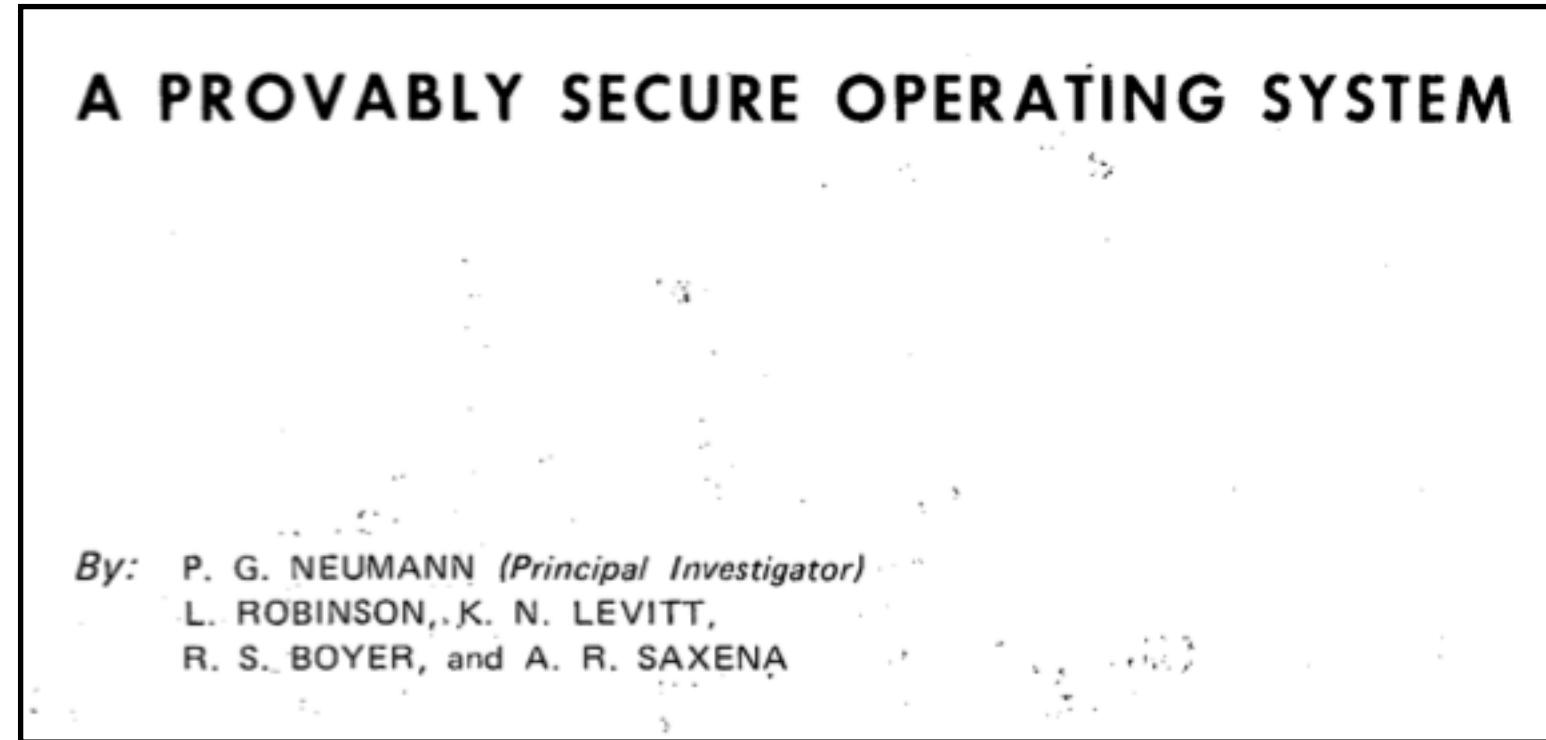
1980

Specification and
Verification of the
UCLA Unix† Security
Kernel

Bruce J. Walker, Richard A. Kemmerer, and
Gerald J. Popek
University of California, Los Angeles

Formally verified OSes, a history

PSOS (Stanford), 1973



1980

Specification and
Verification of the
UCLA Unix† Security
Kernel

Bruce J. Walker, Richard A. Kemmerer, and
Gerald J. Popek
University of California, Los Angeles

- Theorem provers still in their infancy
 - (limited support for pointers!)
- Mostly focused on formalizing OS design, but didn't prove ***implementations*** correct

PSOS Revisited
(Neumann & Feiertag, 2003)

Although some simple illustrative proofs were carried out, it would be incorrect to say that PSOS was a *proven* secure operating system.

Implementing OSes using type-safe languages, a history

(this was before Rust!)

Implementing OSes using type-safe languages, a history

(this was before Rust!)

Extensibility, Safety and Performance in the
SPIN Operating System

Brian N. Bershad Stefan Savage Przemysław Pardyak Emin Gün Sirer
Marc E. Fiuczynski David Becker Craig Chambers Susan Eggers

Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195

SOSP 1995

Modula-3 compiler enforces
module boundaries



Implementing OSes using type-safe languages, a history

(this was before Rust!)

Extensibility, Safety and Performance in the
SPIN Operating System

Brian N. Bershad Stefan Savage Przemysław Pardyak Emin Gün Sirer
Marc E. Fiuczynski David Becker Craig Chambers Susan Eggers

Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195

SOSP 1995

**Language Support for Fast and Reliable Message-based
Communication in Singularity OS**

Manuel Fähndrich, Mark Aiken, Chris Hawblitzel,
Orion Hodson, Galen Hunt, James R. Larus, and Steven Levi

Microsoft Research

EuroSys 2006

Modula-3 compiler enforces
module boundaries



Used a variant of C# with
Rust-like ownership types



Implementing OSes using type-safe languages, a history

(this was before Rust!)

Extensibility, Safety and Performance in the
SPIN Operating System

Brian N. Bershad Stefan Savage Przemysław Pardyak Emin Gün Sirer
Marc E. Fiuczynski David Becker Craig Chambers Susan Eggers

Department of Computer Science and Engineering
University of Washington
Seattle, WA 98195

SOSP 1995

**Language Support for Fast and Reliable Message-based
Communication in Singularity OS**

Manuel Fähndrich, Mark Aiken, Chris Hawblitzel,
Orion Hodson, Galen Hunt, James R. Larus, and Steven Levi

Microsoft Research

EuroSys 2006

Modula-3 compiler enforces
module boundaries



Used a variant of C# with
Rust-like ownership types



seL4 authors:

- Their language runtimes are “substantially bigger” than the seL4 kernel
- Type safety is not strong enough!

**“Complete formal verification is the
only known way to guarantee that a
system is free of programming errors”**

- *seL4: Formal Verification of an OS Kernel (SOSP '09)*

“Complete formal verification is the only known way to guarantee that a system is free of programming errors”

- *seL4: Formal Verification of an OS Kernel (SOSP '09)*

seL4 is the first formally verified OS kernel *implementation*

Authors of seL4

Gernot Heiser

Leads the Trustworthy Systems Group (UNSW Sydney)

{[Gerwin Klein](#), [June Andronick](#), [Rafal Kolanski](#)}, Proofcraft



company started by the seL4 proof team

[Harvey Tuch](#), Google

[Kevin Elphinstone](#), University of New South Wales

[David Cock](#), ETH Zurich

[Philip Derrin](#), Qualcomm

[Dhammadika Elkaduwe](#), University of Peradeniya

{[Kai Engelhardt](#); [Toby Murray](#)}, University of Melbourne

[Michael Norrish](#), Australian National University

[Thomas Sewell](#), University of Cambridge

[Simon Winwood](#), Galois



seL4 Awards

SOSP Best Paper (2009)

CACM Research Highlight (2010)

ACM SIGOPS Hall of Fame (2019)

ACM Software System Award (2022)

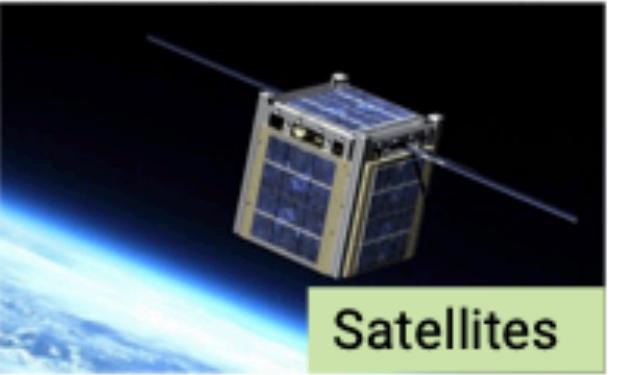
...

seL4's Impact

• Used in Real-World Systems



Autonomous vehicles



Satellites



Critical infrastructure protection



Secure communication device
In use in multiple defense forces



Cars



(From Gernot Heiner's EuroSys '25 keynote)

seL4's Impact

• Used in Real-World Systems



Autonomous vehicles



Satellites



Critical infrastructure protection



Secure communication device
In use in multiple defense forces



Cars



(From Gernot Heiner's EuroSys '25 keynote)

Various DARPA-funded projects

Architecture-Driven Assurance

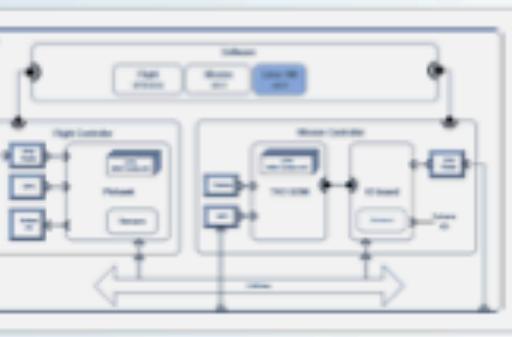
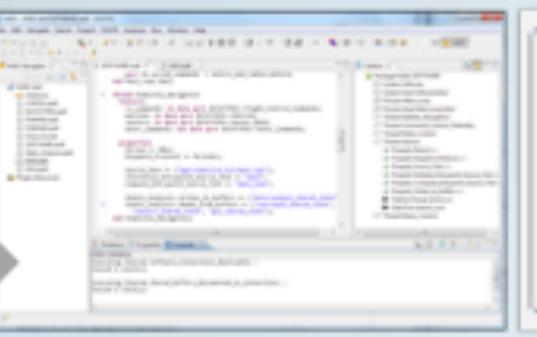
Rockwell Collins, University of Minnesota, Galois, Data61

- Comprehensive use of *formal methods* throughout the development process is needed to ensure that vulnerabilities are eliminated from critical military assets.
- Integrated tools for architectural modeling, analysis, and synthesis make this approach practical and effective.

1 System Architecture modeled in AADL



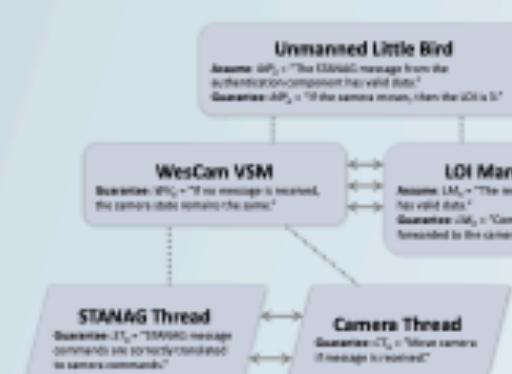
Architecture Analysis and Design Language



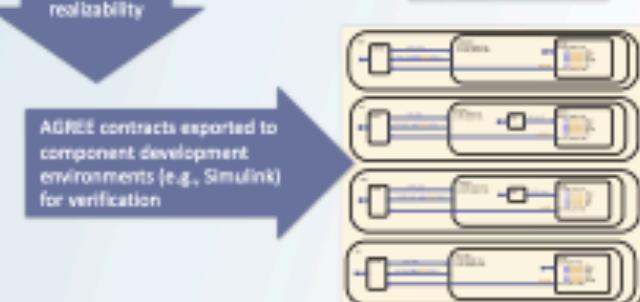
2 Architecture model is correct



Assume-Guarantee Reasoning Environment (AGREE)
Compositional reasoning about system behavior based on formal contracts added to AADL model elements



Resolve
Logic and tool for generating assurance cases from structure of AADL model and claims added to model



3 Software components are correct



Ivory/Tower DSL
• Language prevents common C errors
• Generates memory-safe code
• Embeds checks to detect arithmetic/interface errors

Autopilot Code
*.c, *.h

4 System does what the model says



Component contracts checked for consistency and realizability

AGREE contracts exported to component development environments (e.g., Simulink) for verification

Formally verified from specification to binary



- Secure kernel guarantees isolation between components.
- No information flows other than those explicitly defined in the architecture.

5 Software implementation corresponds to model



Trusted Build
Automatically generates implementation code from architecture model, component specifications, and kernel/OS build system

CMkES
Build tool for seL4 systems

Linux
VxWorks
eChronos

Build for other OS/RTOS but with reduced assurance

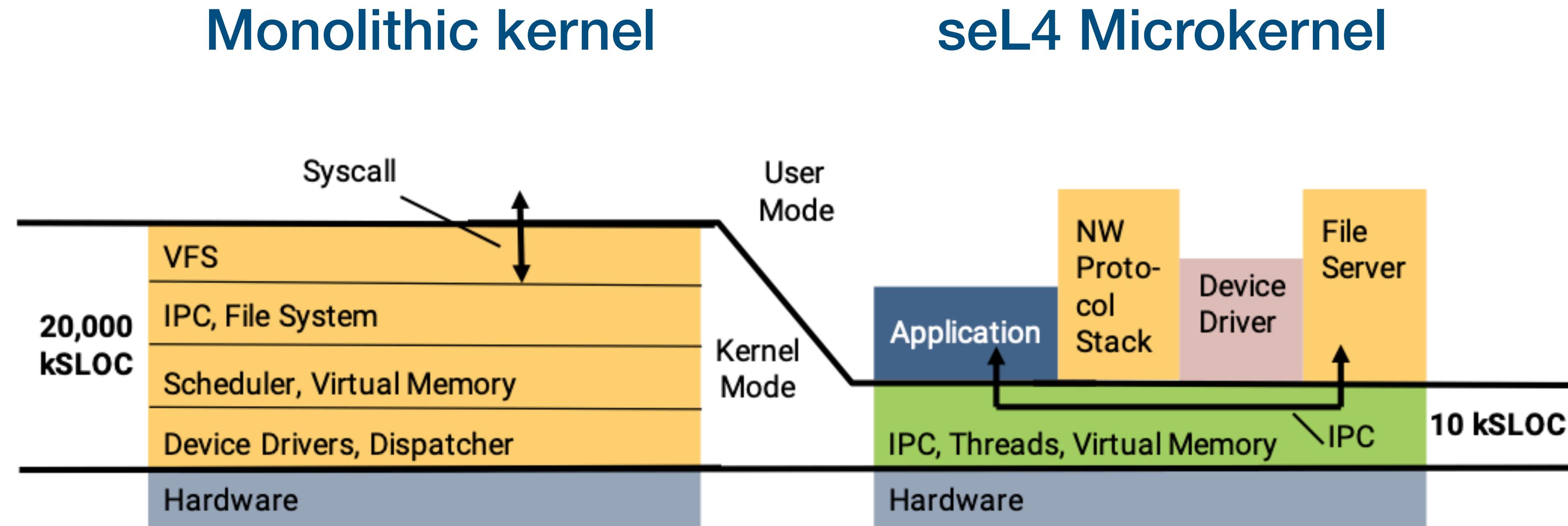


www.darpa.mil

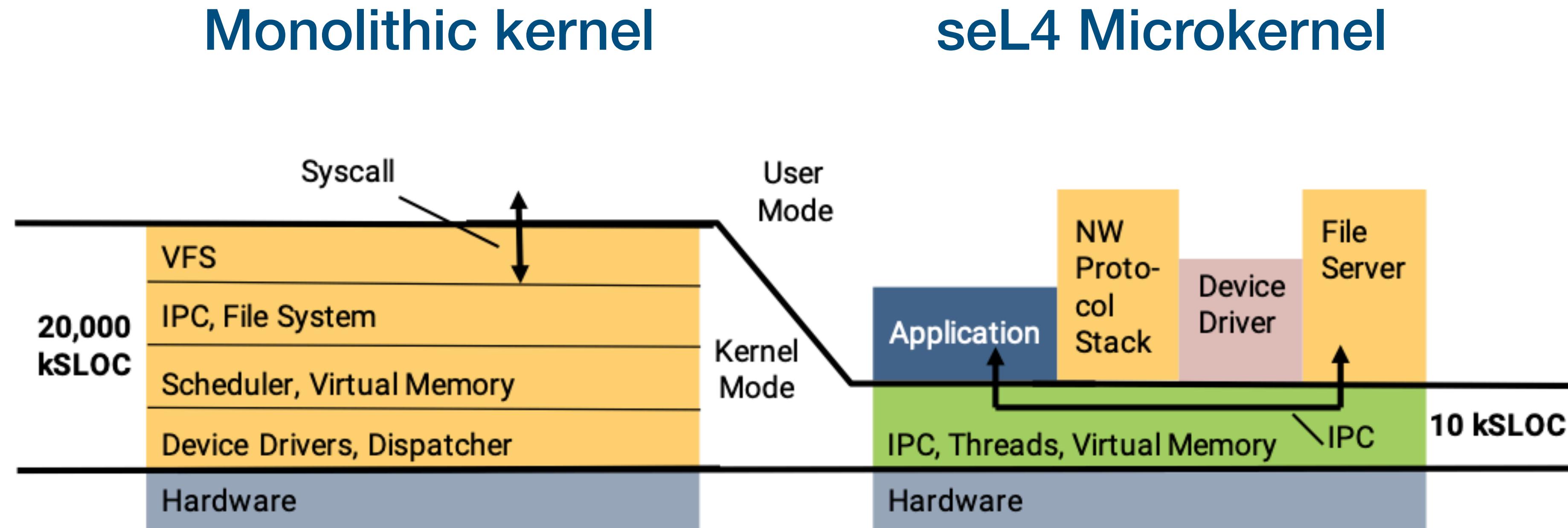
Distribution Statement A - Approved for Public Release, Distribution Unlimited

Formal Verification in seL4

seL4 is a Microkernel



seL4 is a Microkernel



- Small codebase (more amenable to verification)
- No application-oriented services
 - BYO file system, memory manager, device drivers, ...

Aside: Are VMs Microkernels Done Right?

Are Virtual Machine Monitors Microkernels Done Right?

*Steven Hand, Andrew Warfield, Keir Fraser,
Evangelos Kotsovinos, Dan Magenheimer[†]*
University of Cambridge Computer Laboratory
[†] HP Labs, Fort Collins, USA

- VMs work with legacy systems, microkernels don't!
- Microkernels suffer from liability inversion, VMs don't!

HotOS '05

(From the team behind the Xen hypervisor)

Aside: Are VMs Microkernels Done Right?

Are Virtual Machine Monitors Microkernels Done Right?

*Steven Hand, Andrew Warfield, Keir Fraser,
Evangelos Kotsovinos, Dan Magenheimer[†]*
University of Cambridge Computer Laboratory
[†] HP Labs, Fort Collins, USA

- VMs work with legacy systems, microkernels don't!
- Microkernels suffer from liability inversion, VMs don't!

HotOS '05

(From the team behind the Xen hypervisor)

Are Virtual-Machine Monitors Microkernels Done Right?

Gernot Heiser
National ICT Australia* and University of New South Wales
Sydney, Australia
gernot@nicta.com.au

Volkmar Uhlig
IBM T.J. Watson Research Center, Yorktown Heights, NY
vuhlig@us.ibm.com

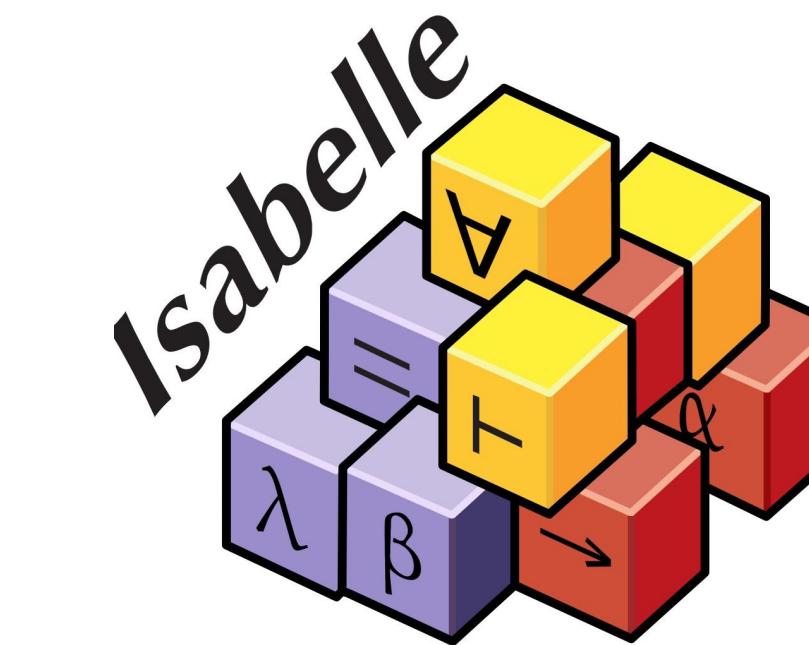
Joshua LeVasseur
University of Karlsruhe, Germany
jtl@ira.uka.de

- Microkernels also work with legacy systems!
- VMs also suffer from liability inversion!

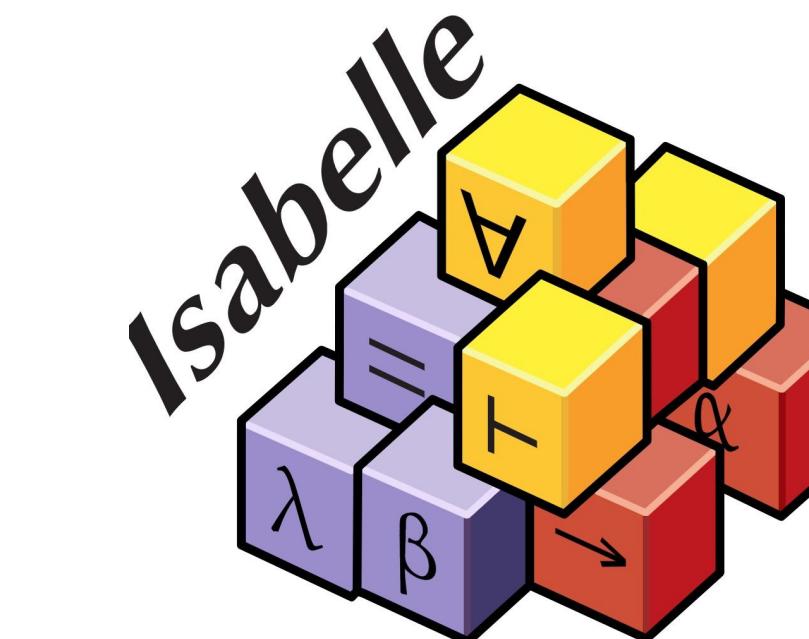
HotOS '06

(From the team behind the L4 microkernel)

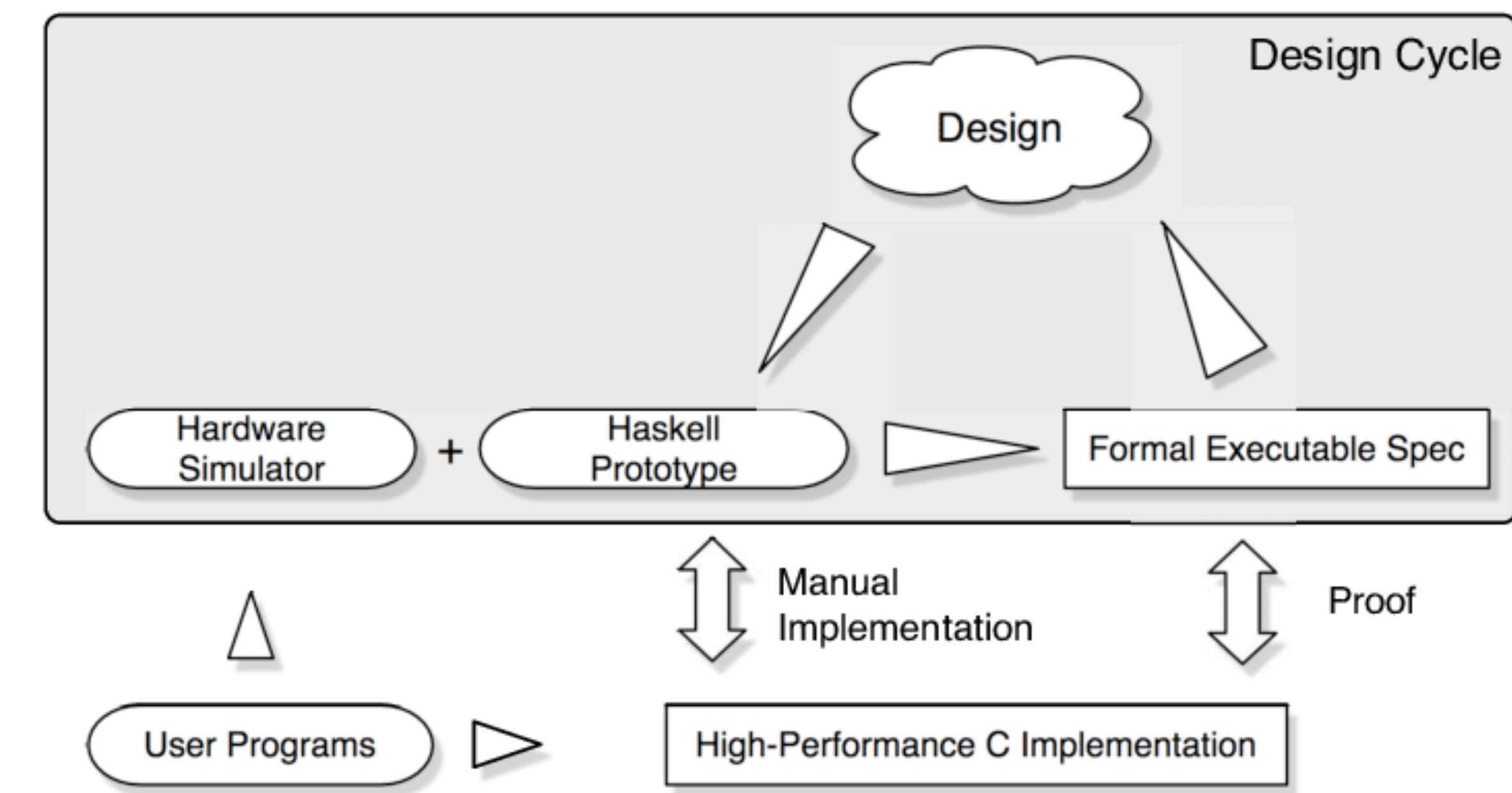
The seL4 development process



The seL4 development process



- Haskell prototype linked w/ a hardware simulator
- Haskell executable spec is automatically translated to the Isabelle theorem prover
- Model re-implemented manually in C for optimization purposes
- NB: final proofs are about the C implementation, not the (intermediate) Haskell prototype
- C compiler & hardware are trusted (assumed to be correct)



Aside: What can be trusted?

Reflections on Trusting Trust

To what extent should one trust a statement that a program is free of Trojan horses? Perhaps it is more important to trust the people who wrote the software.

“You can't trust code that you did not totally create yourself [...] No amount of source-level verification or scrutiny will protect you from using untrusted code.”

From Ken Thompson's (of Unix fame)
1984 Turing Award Lecture

The subset of C used in seL4

Everything from the C99 standard
+ GCC assumptions on data layout, endianness etc.

The subset of C used in seL4

Everything from the C99 standard
+ GCC assumptions on data layout, endianness etc.

except:

- goto, switch statements w/ fall-through cases
- & (address-of) operator on local variables
 - Local variables assumed to be separate from the heap
- Side-effects in expressions
 - (by virtue of the fact that the C code is translated from Haskell)
- Function pointers, untagged unions

What do they prove?

Functional correctness via refinement:

i.e. all behaviors of the C implementation
are captured by the abstract spec

(“Behaviors” are properties specified in Hoare logic)

Hoare Logic, briefly

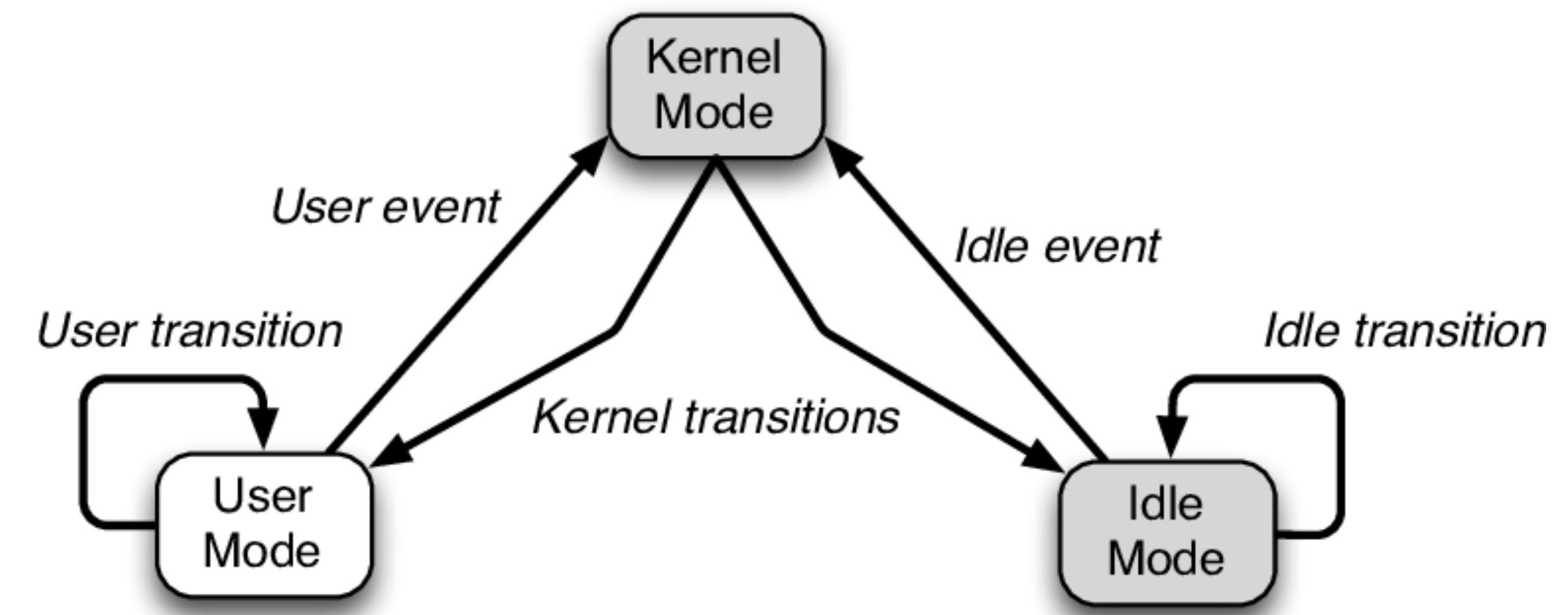
$$\{ P \} \quad c \quad \{ Q \}$$

- If command **c** begins execution in a state satisfying *precondition P*
- and if **c** *terminates* in some final state,
- then the *postcondition Q* is satisfied

Representing seL4 using state machines

Types of transitions:

- *Kernel transitions*: what the kernel does
- *User events*: kernel entry (trap, faults, interrupts)
- *Idle transitions*: what the idle thread does
- *Idle events*: interrupts occurring during idle time



Invariants in the seL4 proof

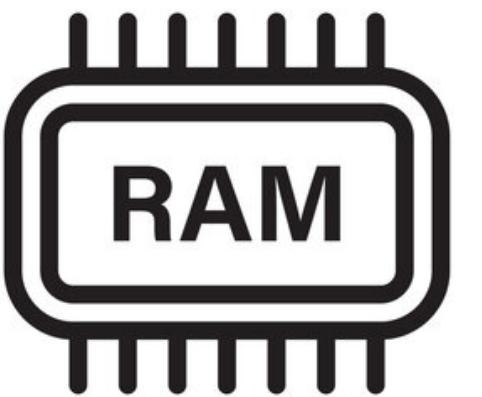
The authors prove 4 types of invariants:

Invariants in the seL4 proof

The authors prove 4 types of invariants:

Memory invariants

no object is at address 0,
kernel objects don't
overlap in memory

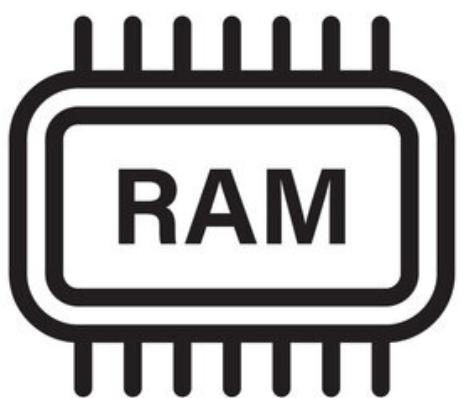


Invariants in the seL4 proof

The authors prove 4 types of invariants:

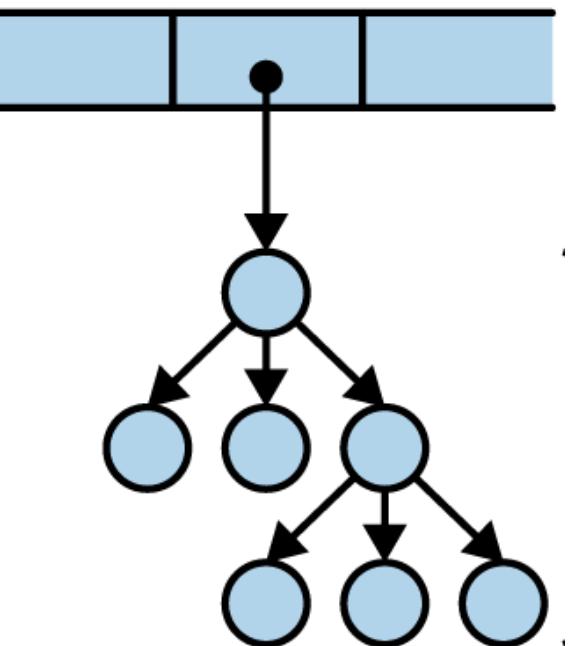
Memory invariants

no object is at address 0,
kernel objects don't
overlap in memory



Typing invariants

references point to objects of
the right type whose values
are in a specified range

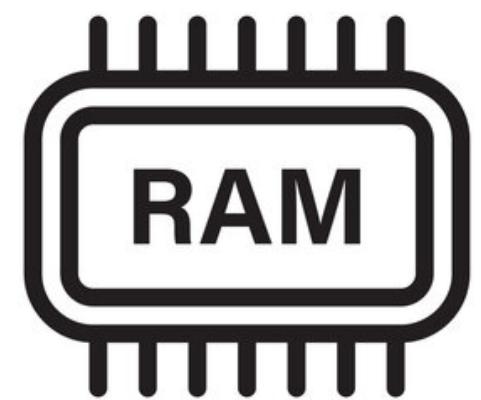


Invariants in the seL4 proof

The authors prove 4 types of invariants:

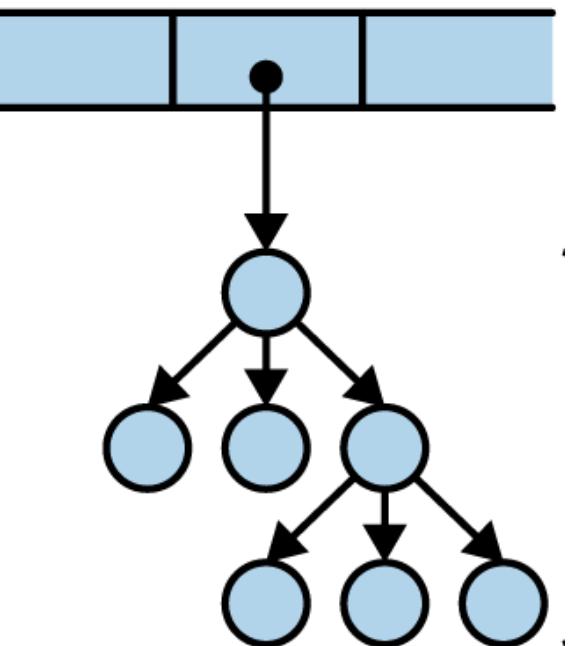
Memory invariants

no object is at address 0,
kernel objects don't
overlap in memory



Typing invariants

references point to objects of
the right type whose values
are in a specified range



Data structure invariants

(e.g. correct back-links in
doubly-linked lists)

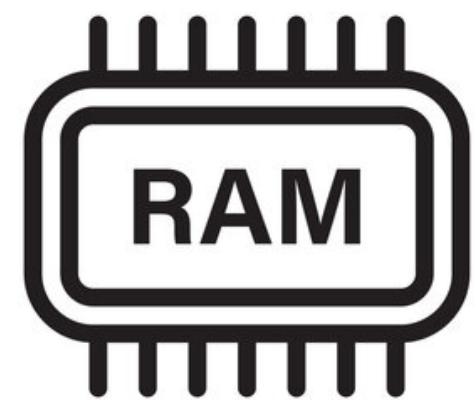


Invariants in the seL4 proof

The authors prove 4 types of invariants:

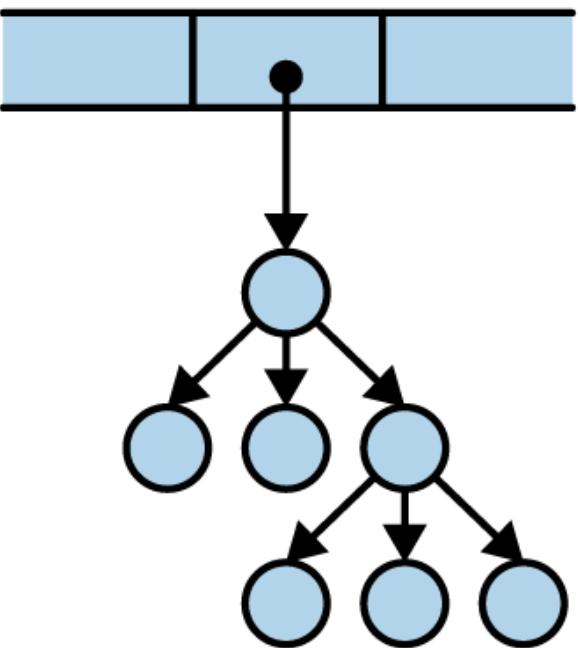
Memory invariants

no object is at address 0,
kernel objects don't
overlap in memory



Typing invariants

references point to objects of
the right type whose values
are in a specified range



Data structure invariants

(e.g. correct back-links in
doubly-linked lists)



(seL4-specific) Algorithmic invariants

(e.g. only the idle thread is in state `idle`,
and it is always in this state)

What their proof implies

The behavior of the C implementation is *always* defined:

- All kernel API calls terminate & return to user-level
 - i.e. the kernel never enters an infinite loop
- No null pointer dereferences, buffer overflows, memory leaks, etc.

⇒ **the kernel can never crash (as long as our assumptions hold)**

What their proof implies

The behavior of the C implementation is *always* defined:

- All kernel API calls terminate & return to user-level
 - i.e. the kernel never enters an infinite loop
- No null pointer dereferences, buffer overflows, memory leaks, etc.

⇒ **the kernel can never crash (as long as our assumptions hold)**

What their proof *doesn't* imply

- The spec describes the behavior that the end-user expects
 - (the spec could have bugs!)
- seL4 is “secure”
 - (“security” needs to be formally defined, this was done in follow-up work)

Bugs found + verification effort

during testing: 16

during verification:

- in C: 160
- in design: ~150
- in spec: ~150

460 bugs

```
void
schedule(void) {
    switch ((word_t)ksSchedulerAction) {
        case ksSchedulerAction_PassiveCurren...
```

Effort

Haskell design	2 py
First C impl.	2 weeks
Debugging/Testing	2 months
Kernel verification	12 py
Formal frameworks	10 py
Total	25 py

Cost

Common Criteria EAL6:	\$87M
L4.verified:	\$6M

```
    tcbSchedDequeue(thread);
}
else {
    switchToThread(thread);
    return;
}
}
```

Discussion Questions

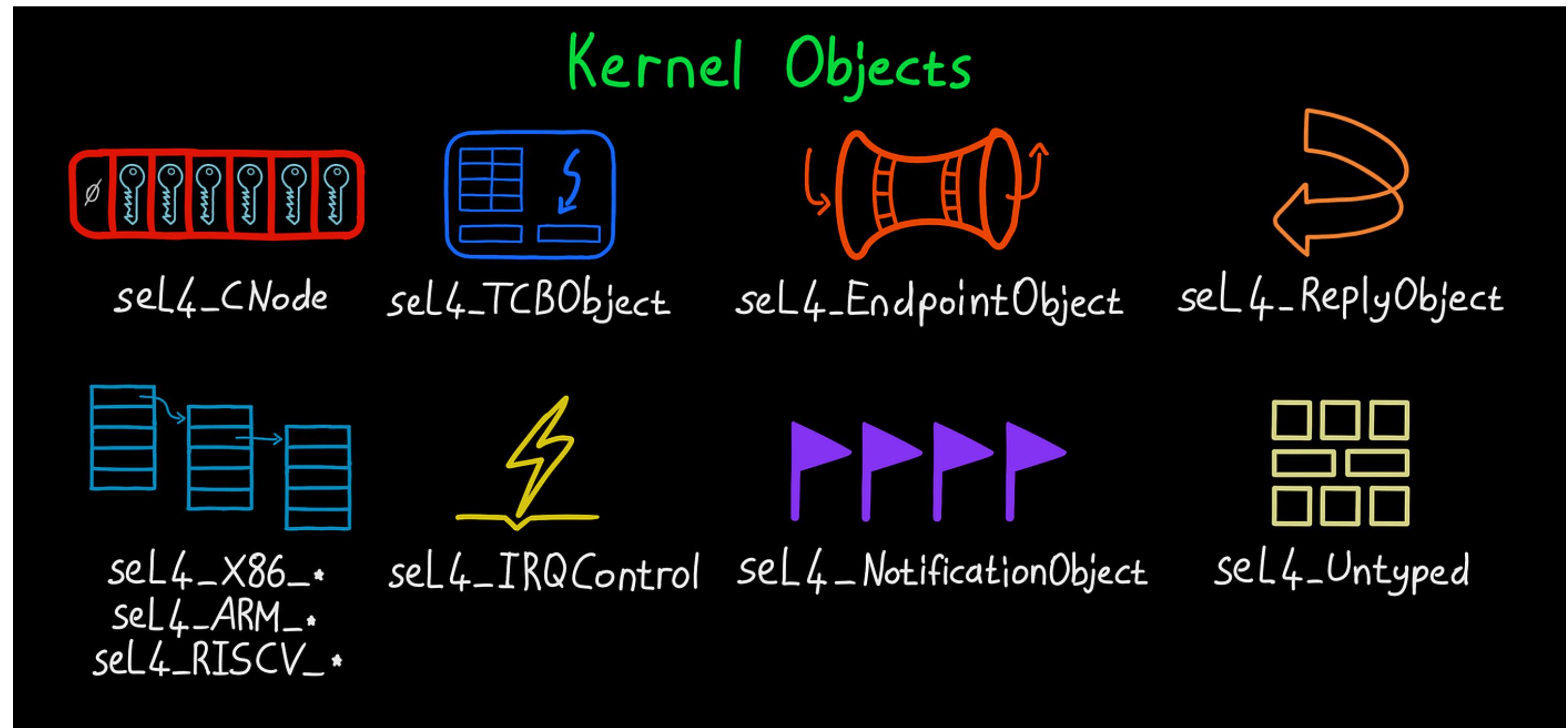
- Do you agree with the authors that their development process (implementing a prototype in Haskell before subsequently writing an optimized C version) is a net cost saver?
- Do you find the seL4 specification reasonable?
 - i.e. expressive enough to prevent bugs, but simple enough for developers
- Is the proof effort reasonable?
- Is the size of the trusted computing base appropriate?
 - (They assume correctness of the C compiler & underlying hardware)

seL4's Design

Kernel Objects

seL4 has an “object-oriented” API in which users manipulate “objects”:

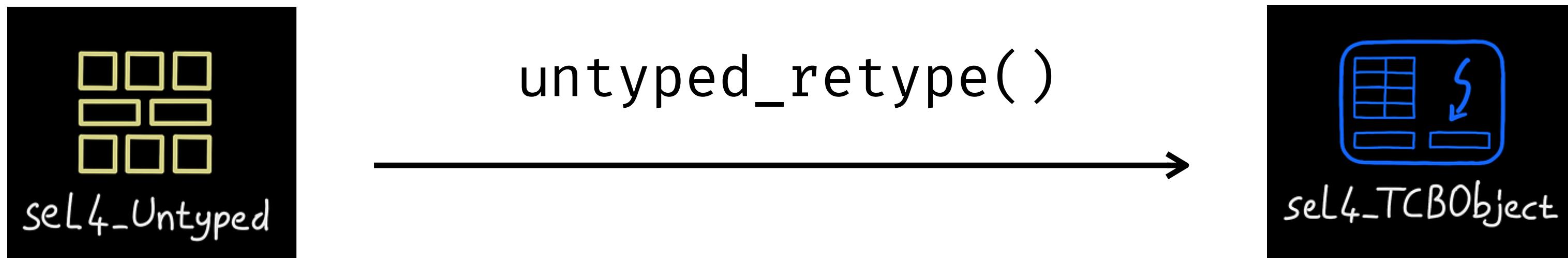
- Thread Control Blocks (representing threads)
- Memory objects (PageDirectory, PageTable, Frame) for building address spaces
- Endpoint & Notification objects for inter-process communication
- Untyped memory (discussed in the next slide)



Illustrations from Tuna Cici's slides on seL4: https://github.com/TunaCici/seL4_Architecture

Kernel Memory Management

- Memory is *typed*:
 - ***Untyped memory*** is unused and can be “casted” into other kernel objects via the `untyped_retype()` method, which allocates memory for the object
 - The rest have specific kernel object types (TCBs etc ...)



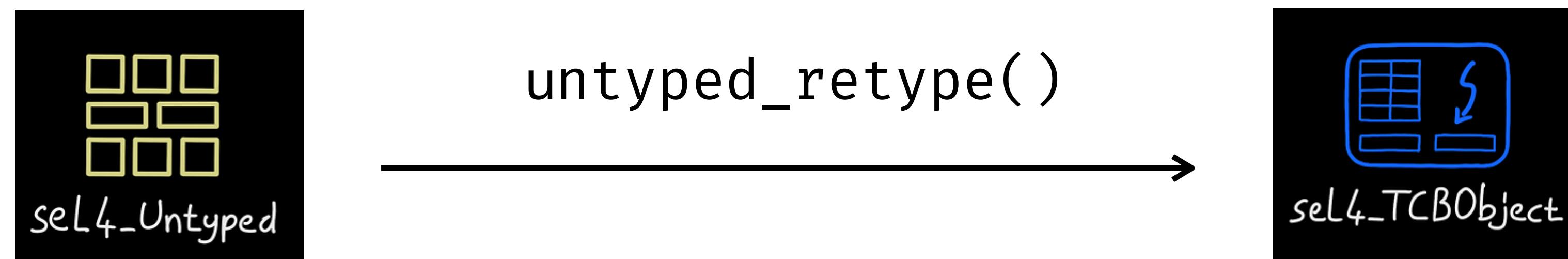
Kernel Memory Management

- **After startup, the kernel *never* allocates memory!**
- At boot-time, the memory required for the kernel is pre-allocated (including code/data/stack sections)
- All remaining memory is untyped and given to the initial user thread



Kernel Memory Management

- Kernel objects have to be explicitly created via `untyped_retype()`
 - Ensures strong resource separation
 - seL4 checks that new objects are wholly contained within regions of untyped memory that don't overlap with other objects
 - Eases verification (the memory allocation policy is pushed outside the kernel, so we only need to prove the correctness of the memory allocation mechanism, *not* the user-level policy)



Capability-Based Access Control

- A *capability* is a token that references a specific kernel object and carries access rights (e.g. read/write) that determine the methods that can be invoked
- Capabilities can be moved & copied (allowing delegation of authority)
- Capabilities live in capability space (CSpace)
- Each thread has a table mapping addresses in CSpace to kernel objects
 - If a thread doesn't have a capability to an object in its CSpace, it can't access it

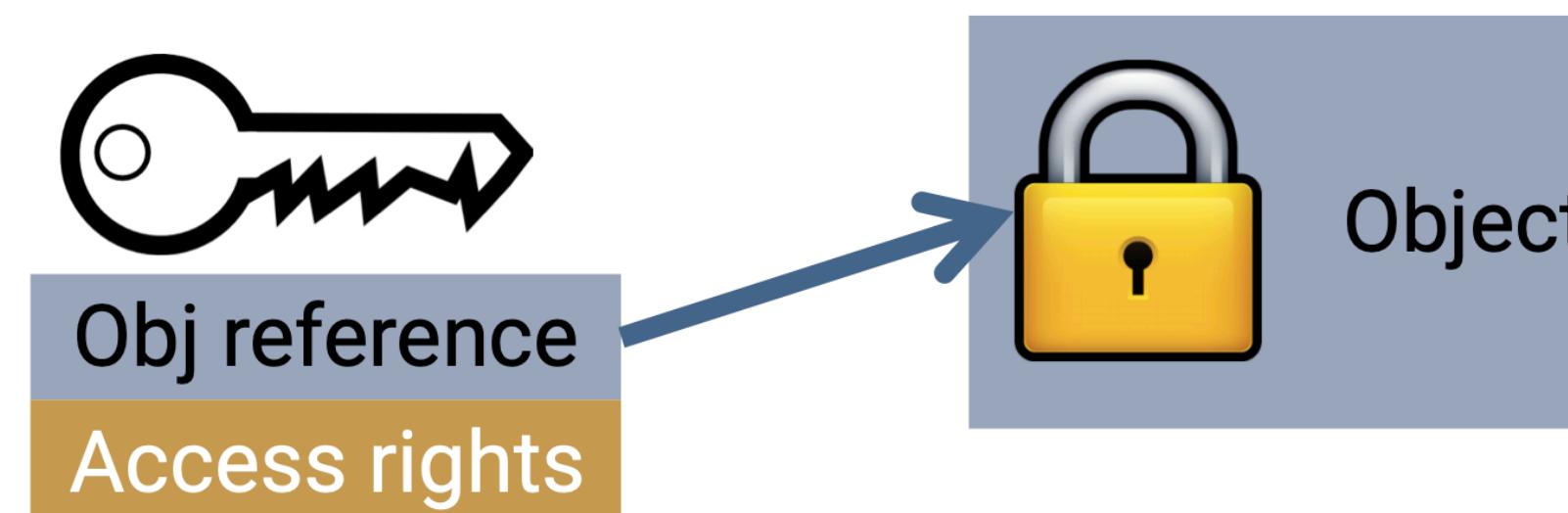


Illustration taken from the seL4 manual

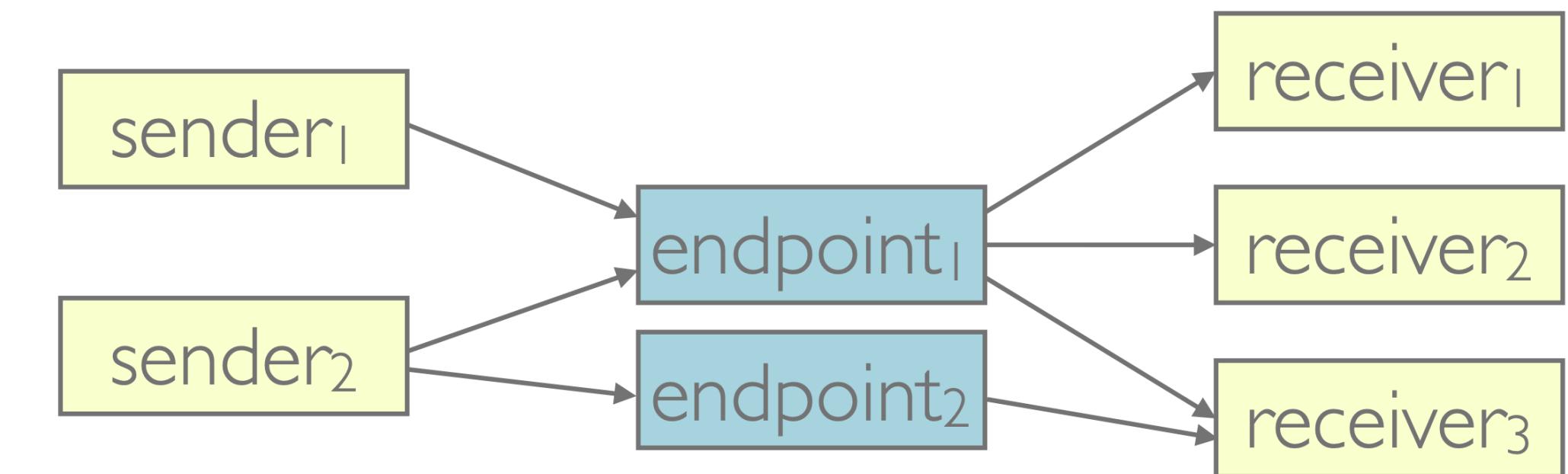
seL4 System Calls

Only 3 core syscalls!

- `Send()`, `Receive()`: requires capabilities
- `Yield()`: invokes scheduler (doesn't require capability)
- All other syscalls are combinations thereof and invoked via message-passing (i.e. calling `Send()` + `Receive()` on a capability)

Inter-Process Communication (IPC)

- IPC between threads is done via (capabilities to) an **endpoint**
 - Endpoint = queue of threads waiting to Send() or Receive()
 - A thread can't be waiting to Send() & Receive() at the same time
- Each thread has a region in its address space designated as its “IPC buffer”



Courtesy of Mark P. Jones, Portland State University

Performance

“seL4 performance is in the vicinity of the fastest L4 kernels” - §5.1

They measure IPC performance, since in a microkernel, “all interactions occur via IPC”

	L4	seL4
Best-case no. of cycles consumed in kernel mode to deliver a zero length message	151 cycles (Uses a hand-crafted assembly fastpath)	224 cycles

Discussion Questions

- Unlike most of the papers we've read, there are very few performance metrics provided in the seL4 paper. To what extent is this a weakness?
- Would reimplementing seL4 in a memory-safe language like Rust reduce the verification burden?
- How does seL4 guarantee the correctness of virtual memory operations even though the kernel itself runs under virtual memory? For example, how do they guarantee that a dereference of a pointer in the kernel does not cause a fault?

Limitations

Gernot Heiser gave a keynote at ASPLOS / EuroSys '25 about seL4's limitations

11:00 AM CEST – 12:00 PM CEST: ASPLOS + EuroSys 2025 Joint Keynote 2 by Gernot Heiser (Univ. of New South Wales)

Will we ever have truly secure operating systems?

+

LOCATION: ROTTERDAM HALL 1

Session Chair: Haibo Chen (Shanghai Jiao Tong University)

Abstract

Half a century after PSOS, the first attempts to prove an operating system (OS) secure, OS faults remain a major threat to computer systems security. A major step forward was the verification of the seL4 microkernel, the first proof of implementation correctness of an OS kernel. Over the next 4 years this proof was extended to the binary code, proofs of security enforcement, and sound and complete worst-case execution-time analysis. The proofs now cover 4 ISAs.

Yet, 15 years later, there is still no provably secure OS. While seL4 has been successfully deployed in defence and civilian security- and safety-critical systems, it is a microkernel that mostly guarantees process isolation without providing the application-oriented services expected from an OS. This not only makes seL4 difficult to deploy, but means that there is limited assurance that a system built on top is secure in any real sense.

Why has seL4 not been leveraged into a secure OS? In this talk I will explore some of the reasons behind this disappointing state of affairs, and what can be done about it. Specifically I will discuss our current work on LionsOS, a new seL4-based OS targeting the embedded/cyberphysical domain, and designed to be verifiable. I will also discuss more speculative, early-stage work towards a provably secure, general-purpose OS.



Limitations of seL4 (after 10+ years)

“While seL4 has been successfully deployed [...], it is a microkernel that mostly guarantees process isolation without providing the application-oriented services expected from an OS. This not only makes seL4 difficult to deploy, but means that there is limited assurance that a system built on top is secure in any real sense.”

— Gernot Heiser’s EuroSys ’25 keynote

Limitations of seL4 (after 10+ years)

Also from Prof. Heiser's keynote / the seL4 manual:

- “It’s too hard to build things on seL4 [...] you need years and years of work”
- seL4’s “arcane build system didn’t help”
- “Much more is needed” re: device drivers, network protocol stacks, file systems

Other (practical) limitations

- seL4 doesn't load-balance across cores for you
 - you assign threads to cores (or migrate them manually)
- seL4 emphasizes static resource allocation
 - (i.e. if you run out of untyped memory, you can't create more kernel objects until they have been freed)
- Not everything is verified
 - Users need to consult the seL4 manual to see what is trusted / unverified
 - e.g. original proofs were for 32-bit ARM & x86, RISC-V proofs are in progress

Follow-up papers

- Translation validation for a verified OS kernel (PLDI '13)
- Time protection: The missing OS abstraction (EuroSys '19)
- seL4: From general purpose to a proof of information flow enforcement (IEEE S&P '13)
- Formally Verified System Initialisation (Formal Methods & Software Engineering 2013)

Thanks!