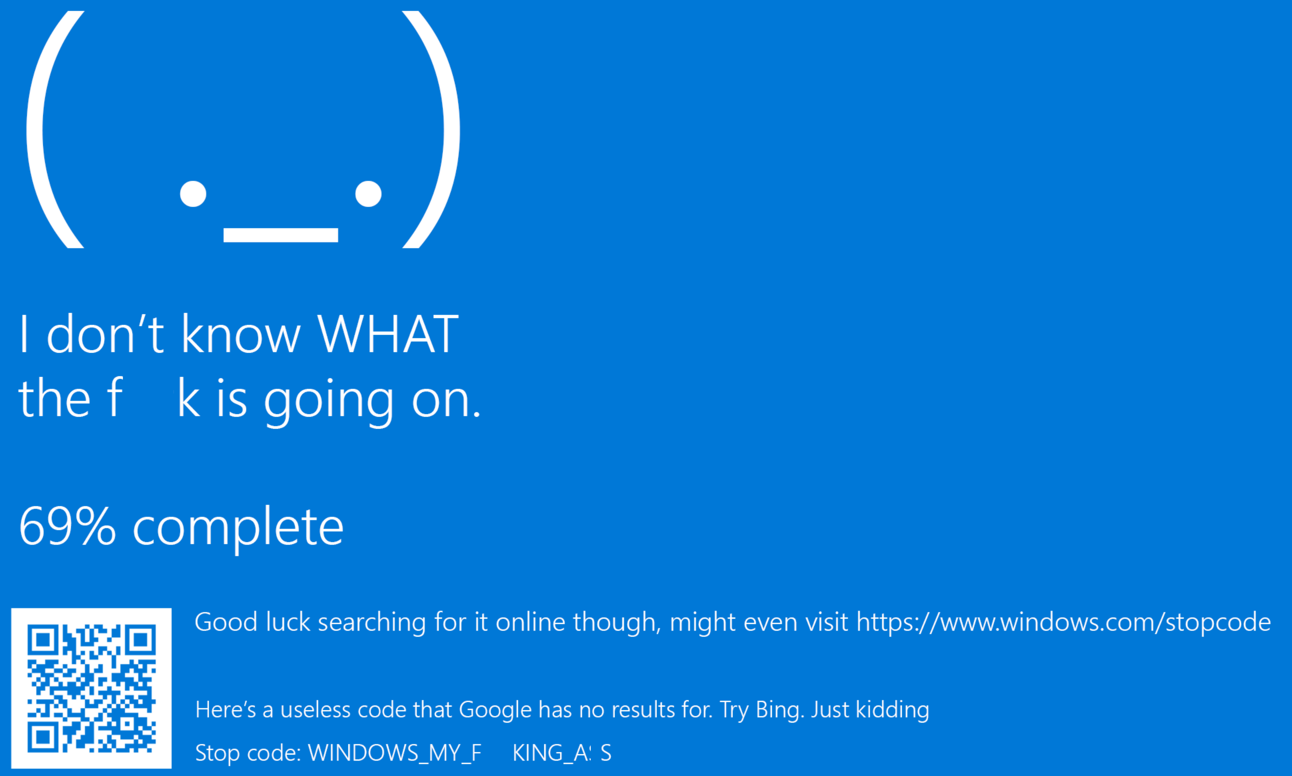
seL4: Formal Verification of an OS Kernel

Xilin Tang

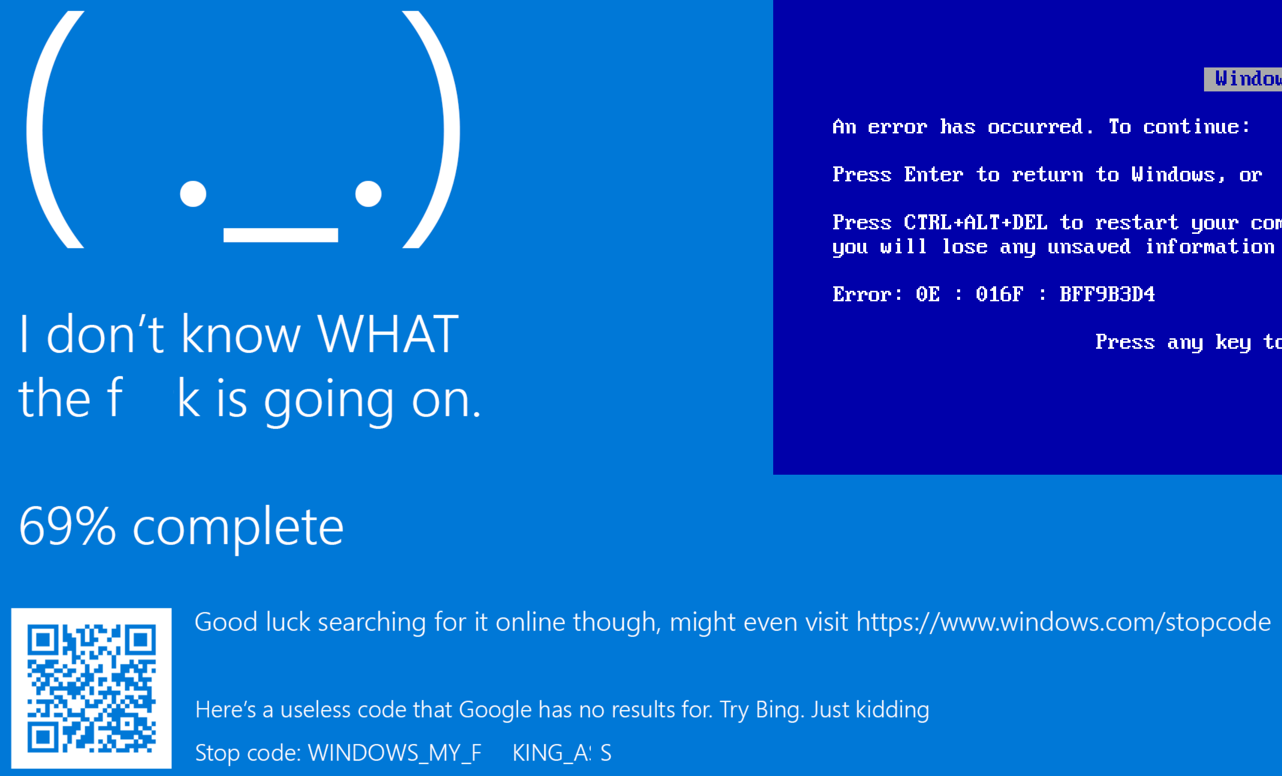
October 1, 2024

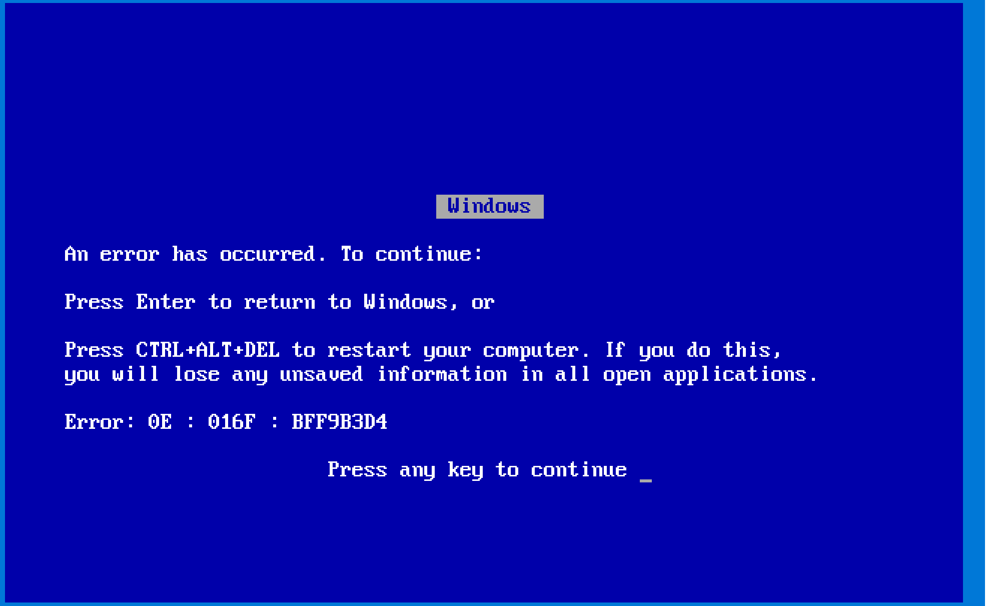
Outline

- OS Verification
- The paper



From Reddit
https://www.reddit.com/r/pcmasterrace/comments/1086q0j/cursed_bsod_one_of_the_coworkers_screen_saver/





- 00: Division fault
- 01: Startup Error
- 02: Non-Maskable Interrupt
- 03: Shutdown Error
- 04: Overflow Trap
- 05: Bounds Check Fault
- 06: Invalid Opcode Fault
- 07: "Coprocessor Not Available" Fault
- 08: Double Fault
- 09: Coprocessor Segment Overrun
- 0A: Invalid Task State Segment Fault
- 0B: Not Present Fault
- 0C: Stack Fault
- 0D: General Protection Fault
- 0E: Page Fault
- 0F: Error Message Limit Exceed
- 10: Coprocessor Error Fault
- 11: Alignment Check Fault

From wiki
https://en.wikipedia.org/wiki/Blue_screen_of_death

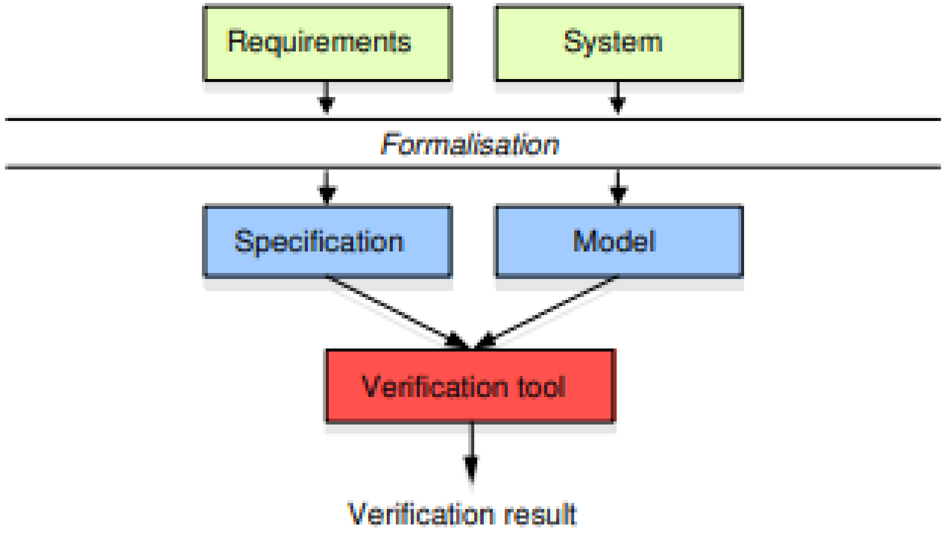
From Reddit
https://www.reddit.com/r/pcmasterrace/comments/1086q0j/cursed_bsod_one_of_the_coworkers_screen_saver/

Discussion

- Why do we need OS verification?
- What should OS verification support?
- If it comes with certain cost, is that acceptable?

OS verification?

- The huge risk exposure of bugs in OS
- Complex and repetitive tasks
- Semantic gap between user application and formal verification



From H. Tuch, G. Klein, and G. Heiser, “OS verification: now!,” in *Proceedings of the 10th conference on Hot Topics in Operating Systems - Volume 10*, in HOTOS’05. USA: USENIX Association, 2005, p. 2.

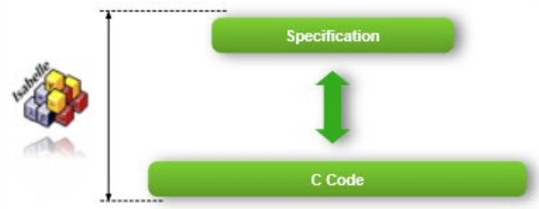
Implications of success

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

Not implied:

- “secure” (define secure)
- zero bugs from expectation to physical world
- covert channel analysis



- 00: Division fault
- 01: Startup Error
- 02: Non-Maskable Interrupt
- 03: Shutdown Error
- 04: Overflow Trap
- 05: Bounds Check Fault
- 06: Invalid Opcode Fault
- 07: "Coprocessor Not Available" Fault
- 08: Double Fault
- 09: Coprocessor Segment Overrun
- 0A: Invalid Task State Segment Fault
- 0B: Not Present Fault
- 0C: Stack Fault
- 0D: General Protection Fault
- 0E: Page Fault
- 0F: Error Message Limit Exceed
- 10: Coprocessor Error Fault
- 11: Alignment Check Fault

From wiki
https://en.wikipedia.org/wiki/Blue_screen_of_death

From SOSP’09 Presentation of seL4

Solution preview

- Founctional Correctness
  - for each input it produces an output satisfying the specification.
- Combination of logical proof and functional programming
- Design & Implementation
  - Refinement
  - Confinement

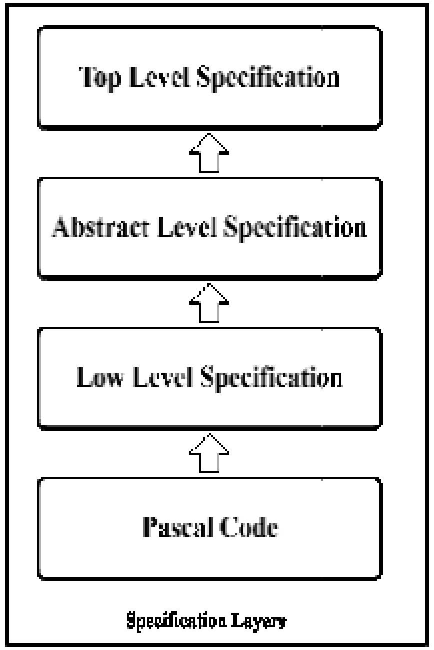
Basic terminologies

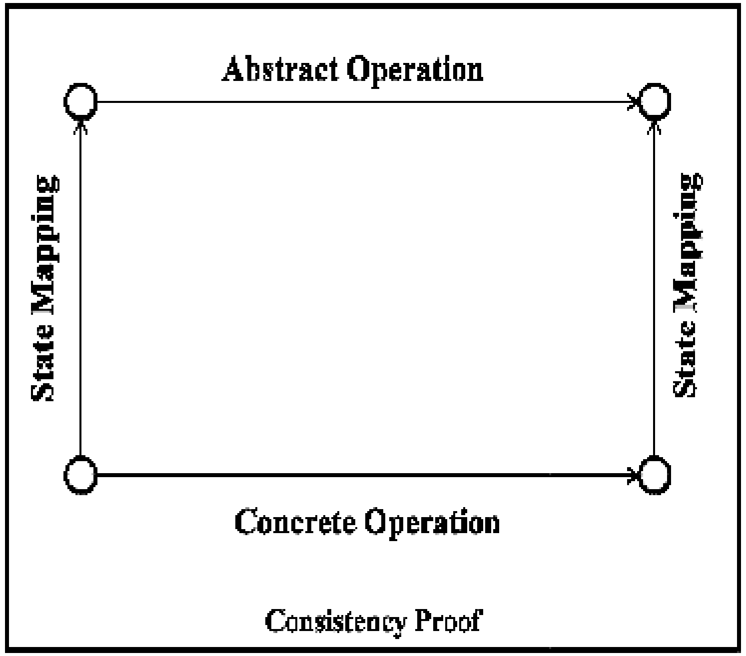
- Model Checking
  - Finite state concurrent system
- Proof-carrying code
  - A theorem prover inside kernel
- Static source-code checking
- Functional correctness
  - Implementation always strictly follows high-level abstract specification of kernel behaviour
  - Feasible to prove (not to imply) security properties at the code level

Timeline of attempts

- 1978 UCLA
- 1980 PSOS
- 1989 KIT(Kernel for Isolated Task)
- 2000 EROS (Extremely Reliable Operating System)
- 2002 VFiasco
- 2009 seL4

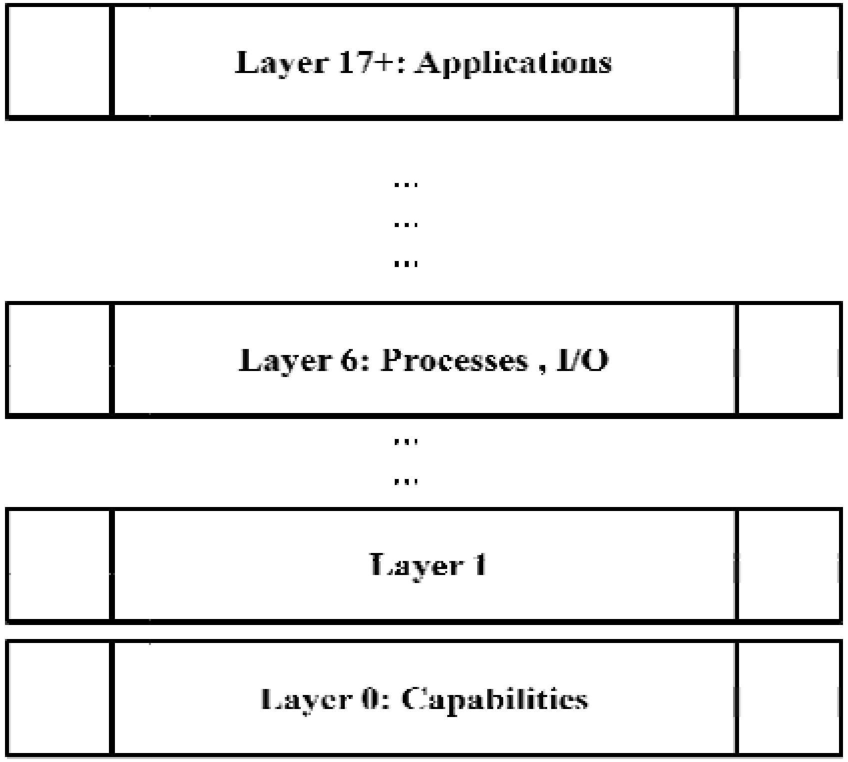
UCLA





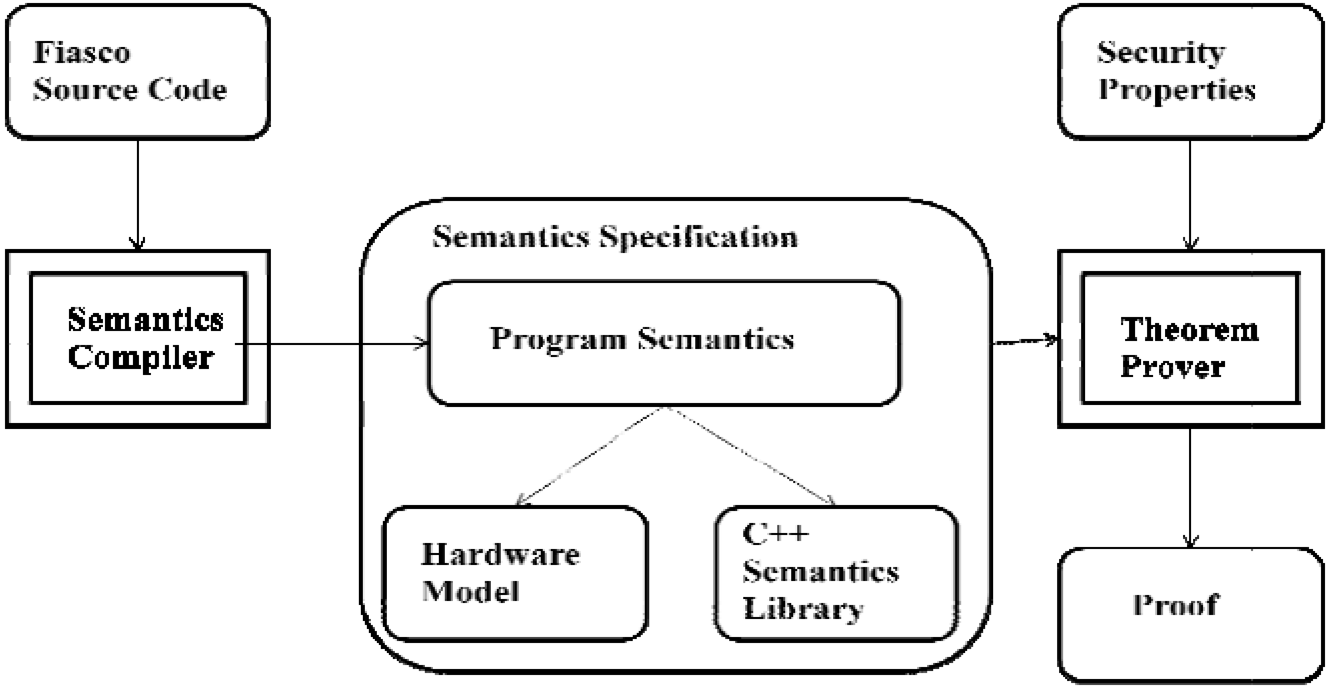
From K. Anjaria and A. Mishra, “OS Verification- A Survey as a Source of Future Challenges,” *IJCSES*, vol. 6, no. 4, pp. 1–20, Aug. 2015, doi: 10.5121/ijcses.2015.6401.

PSOS



From K. Anjaria and A. Mishra, “OS Verification- A Survey as a Source of Future Challenges,” *IJCSES*, vol. 6, no. 4, pp. 1–20, Aug. 2015, doi: 10.5121/ijcses.2015.6401.

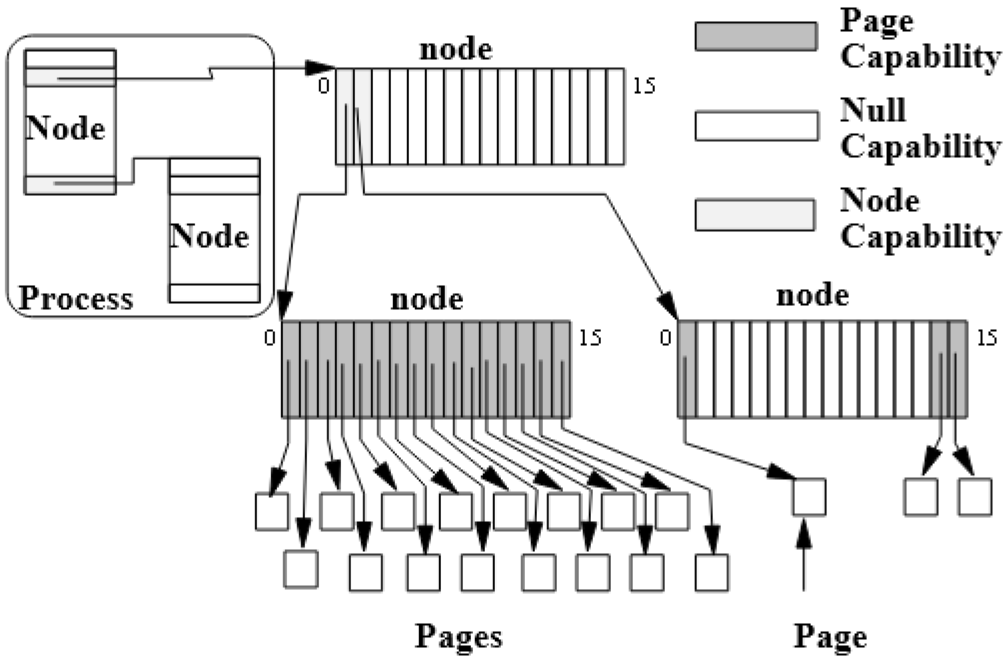
VFiasco



From K. Anjaria and A. Mishra, “OS Verification- A Survey as a Source of Future Challenges,” *IJCSES*, vol. 6, no. 4, pp. 1–20, Aug. 2015, doi: 10.5121/ijcses.2015.6401.

EROS

- Paper and pen
- Capability based Confinement Mechanism



From J. S. Shapiro and S. Weber, “Verifying the EROS confinement mechanism,” in *Proceeding 2000 IEEE Symposium on Security and Privacy. S&P 2000*, May 2000, pp. 166–176. doi: 10.1109/SECPRI.2000.848454.

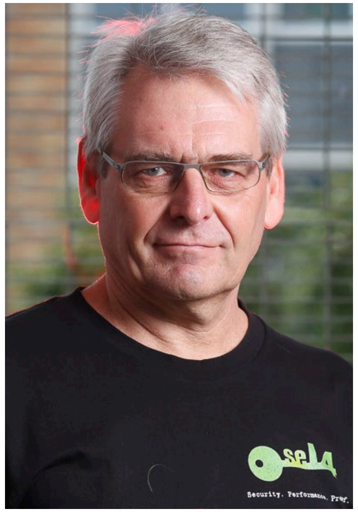
Comparison

Project	Highest level	Lowest level	Specs	Proofs	Prover	Approach	Year
UCLA	Security model	Pascal	90%	20%	XIVUS	Alphard	(?)-1980
KIT	Isolated task	Assembly	100%	100%	Boyer Moore	Interpreter equivalence	(?)-1987
PSOS	Applicatio n level	Secure code	17 layers	0%	SPECI AL	HDM	1973-1983
VFiasco	Doesn't crash	C++	70%	0%	PVS	Semantic compiler	2001-2008
EROS	Security model	BitC	Security model	0%	ACL2(?)	Language based	2004-(?)
L4 verified	Security model	C/assembly	100%	70%	Isabell e	Performance production code	2005-(2008)

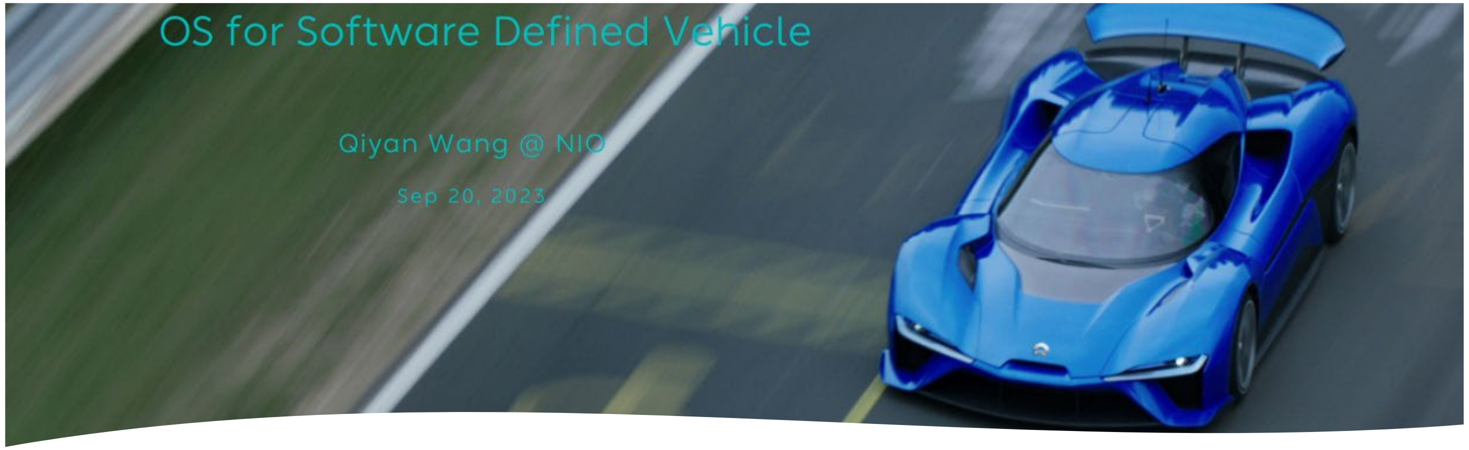
From K. Anjaria and A. Mishra, "OS Verification- A Survey as a Source of Future Challenges," *IJCSES*, vol. 6, no. 4, pp. 1–20, Aug. 2015, doi: 10.5121/ijcses.2015.6401.

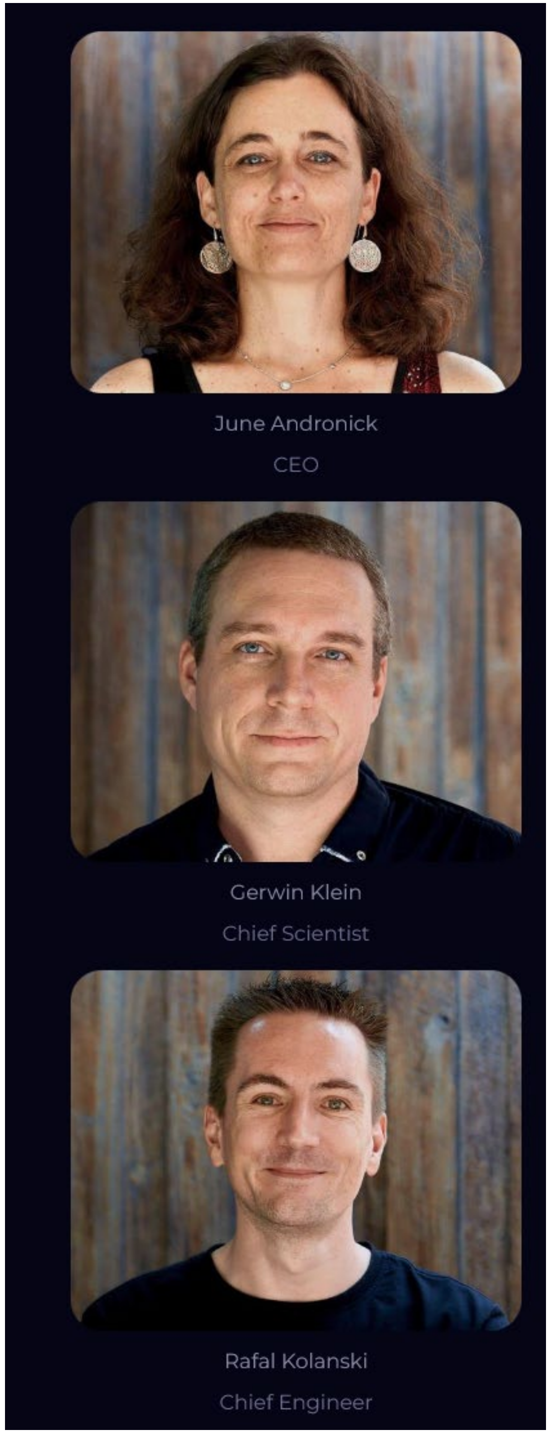
More Contexts

- Trustworthy System Lab
- Gernot Heiser

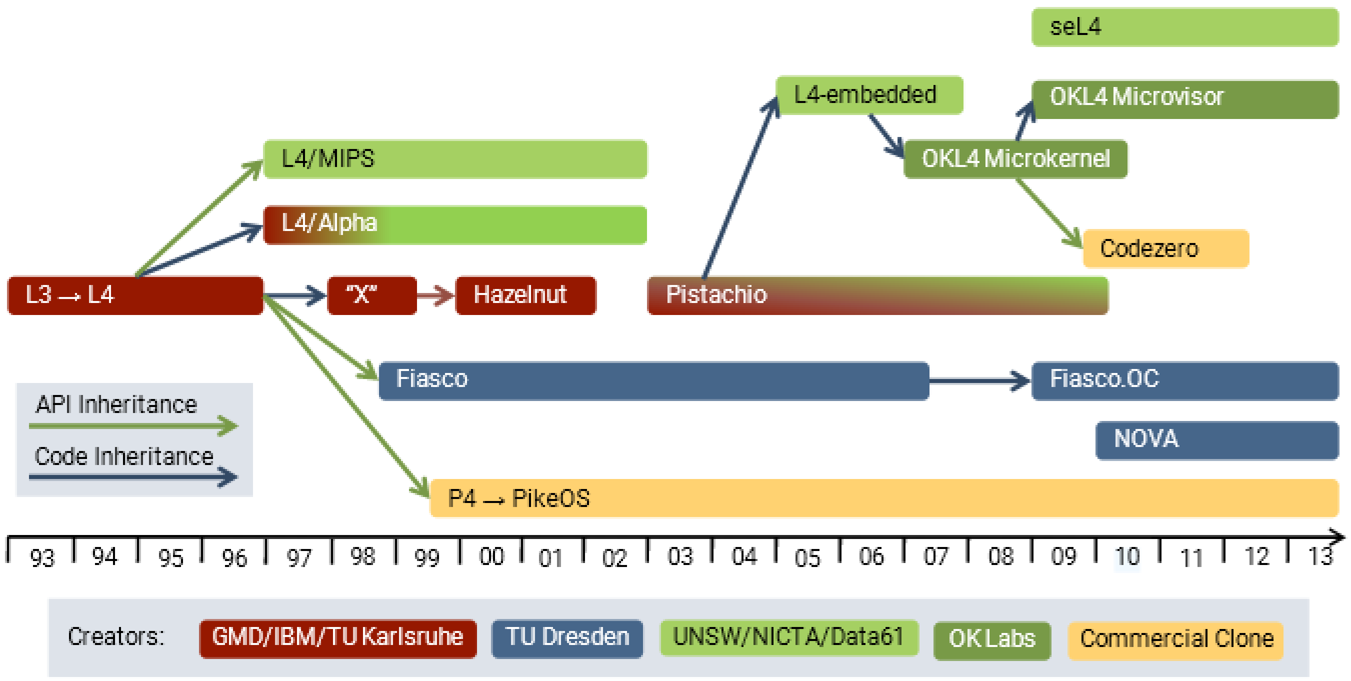


2006 OK Labs
2010 Open-Sourced
2014 Acquired by General Dynamics C4 Systems
2016 seL4 Foundation Established
2017 Adopted by HENSOLDT Cyber and Data61 (part of CSIRO)
2021 Proofcraft
2022 Dropped by Data61



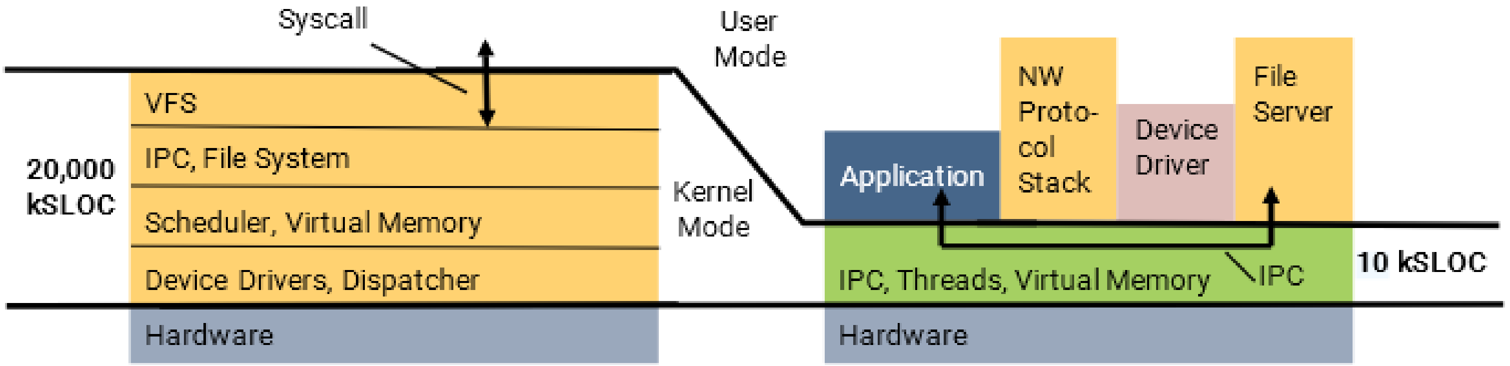


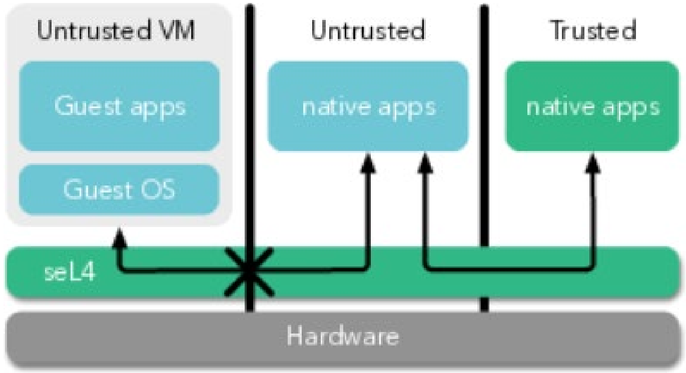
seL4 as a family member



From Gernot Heiser, "The seL4 Microkernel An Introduction, " May 2024

se+Microkernel





From Gernot Heiser, “The seL4 Microkernel An Introduction, ” May 2024

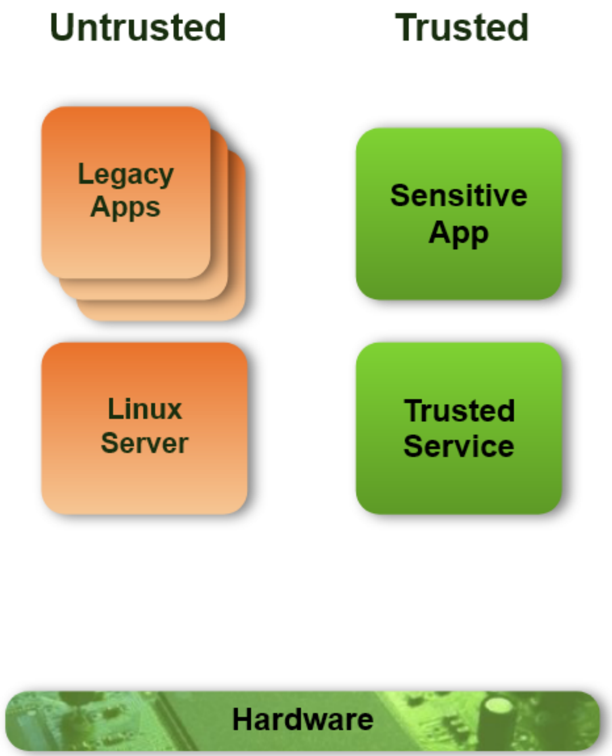
Overview

Small trustworthy foundation

- hypervisor, microkernel, nano-kernel, virtual machine, separation kernel, exokernel ...
- High assurance components in presence of other components

seL4 API:

- IPC
- Threads
- VM
- IRQ
- Capabilities



From SOSP'09 Presentation of seL4

Iterative design process

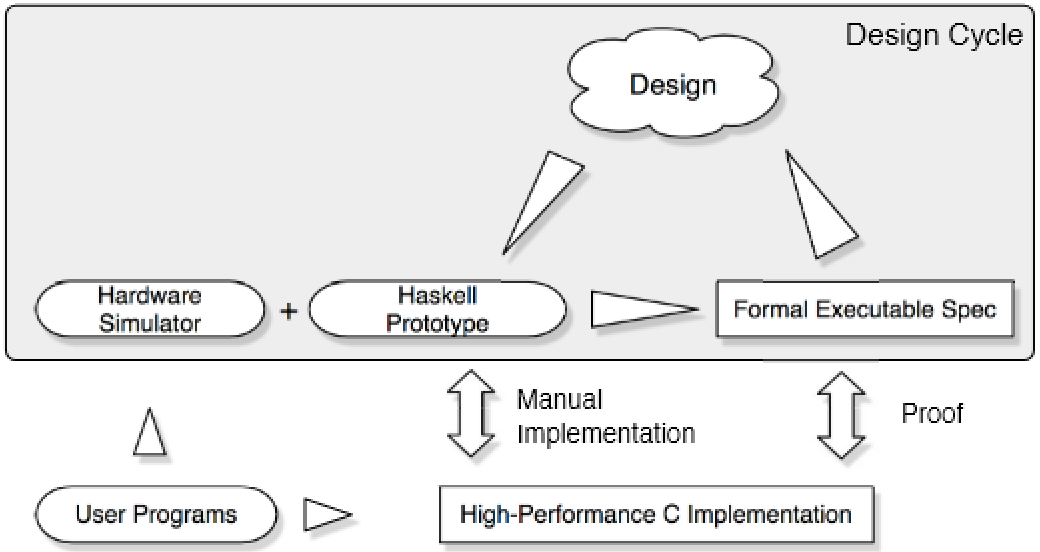
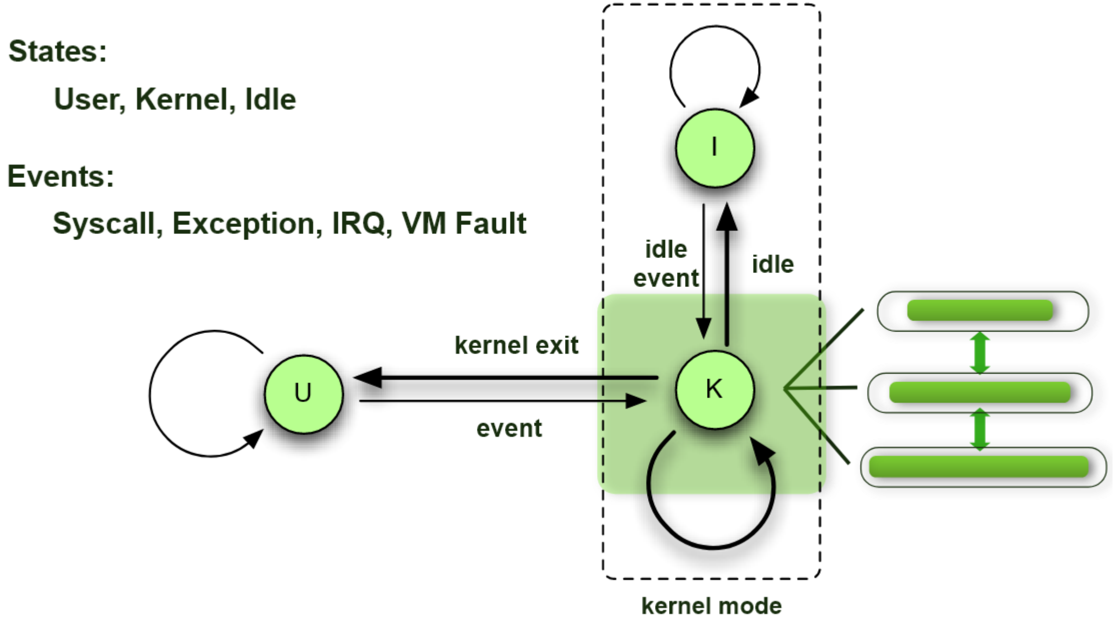


Figure 1: The seL4 design process

From G. Klein *et al.*, “seL4: formal verification of an OS kernel,” in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, Big Sky Montana USA: ACM, Oct. 2009, pp. 207–220. doi: 10.1145/1629575.1629596.

Architecture



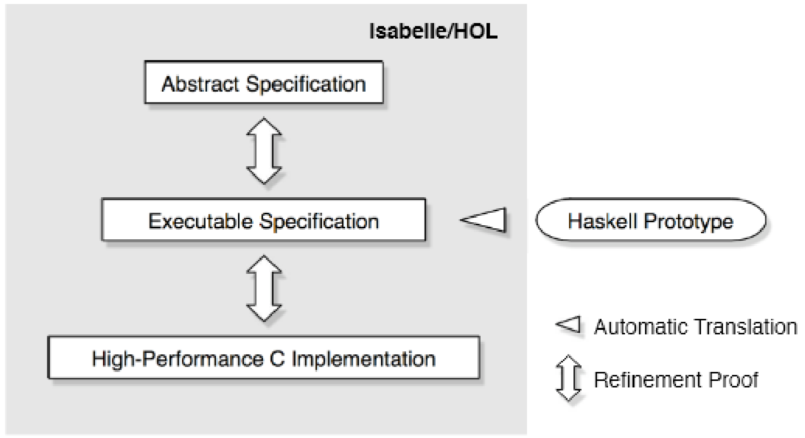


Figure 2: The refinement layers in the verification of seL4

From G. Klein *et al.*, "seL4: formal verification of an OS kernel," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, Big Sky Montana USA: ACM, Oct. 2009, pp. 207–220. doi: [10.1145/1629575.1629596](https://doi.org/10.1145/1629575.1629596).

Kernel objects

- Types of kernel objects include:
  - Untyped memory
  - TCB objects for representing threads
  - Endpoint and Notification objects for IPC
  - Memory objects (PageDirectory, PageTable, Frame) for building address spaces
  - CNode objects for building capability spaces
  - and more …
- Capabilities are used to manage user-level access to all of these different types of object

Capability

- Capability, supporting principle of least authority (POLA), is better than **ACLs** (access-control model of access-control lists).

From Gernot Heiser, “The seL4 Microkernel An Introduction, ” May 2024

System calls in seL4

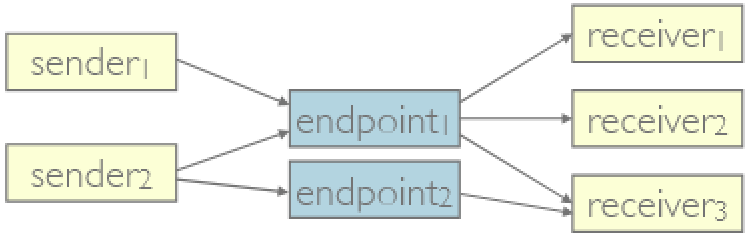
- Conceptually, seL4 has an "object-oriented" API with just three system calls
  - *Send* a message to an object (via a capability)
  - *Wait* for a message from an object (via a capability)
  - *Yield* (does not require an object/capability)
- For example:
  - send a message to an Endpoint object to communicate with another thread
  - send a message to a TCB object to configure the thread
- In practice, there are other variants of Send/Wait to support combined send and receive, RPC, and other patterns

Discussion

- How to support capability-based IPC?
- How can interprocess communication (IPC) be controlled and protected using capabilities?
- One option would be to use capabilities to TCB objects
  - These are useful for other purposes anyway (e.g., reading/modifying thread status, starting, suspending, …)
  - Could use send / receive permissions on TCB capabilities to determine which IPC actions are allowed
- But this is also inflexible:
  - Single thread to single thread communication is limiting
  - Lacks fine-grained control: if you can contact a thread for one purpose, you can contact it for any purpose

IPC via endpoints

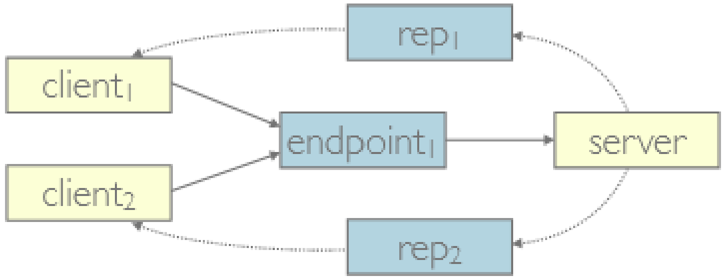
- Interprocess communication (IPC) in seL4 passes messages between threads using (capabilities to) an endpoint object:



- Allows flexible communication patterns
  - multiple senders and/or receivers on a single endpoint
  - multiple endpoints between communication partners
- Messages are transferred synchronously when both sender and receiver are ready ("rendez-vous")
- Multiple senders or receivers can be queued at each endpoint

A case study

- Practical systems often use a client-server architecture in which one "server" thread performs work for many "clients“

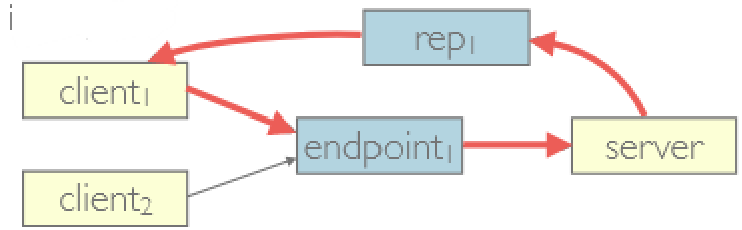


- What if the client needs a reply? How will the server know where to send it?
- The client could send a capability to a "reply" endpoint as part of its request. But this makes extra work for the client, and could be abused by a malicious (or buggy) server.

From CS 410/510 Fall 2018, Portland State University

Reply capabilities

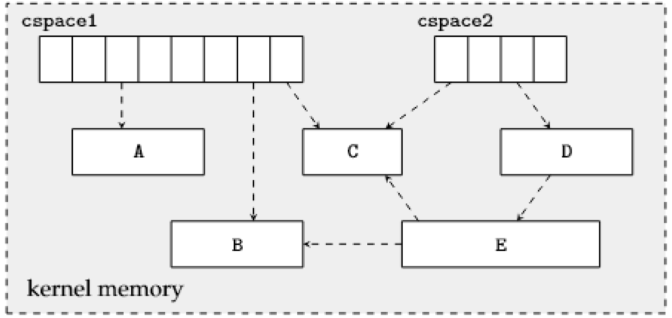
- seL4 tackles this problem by introducing a special "Reply" capability type:



- The Call system call combines a Send and a Wait
- The kernel gives a new "reply capability" to the receiver
- The receiver can move but not copy the reply capability
- The receiver can send a message to the reply capability
- The reply capability is deleted after its first (hence only) use

Capability spaces

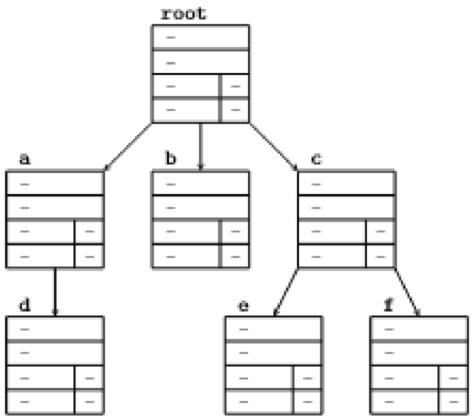
- Every thread has a "capability space", which is a table mapping capability indexes to kernel objects

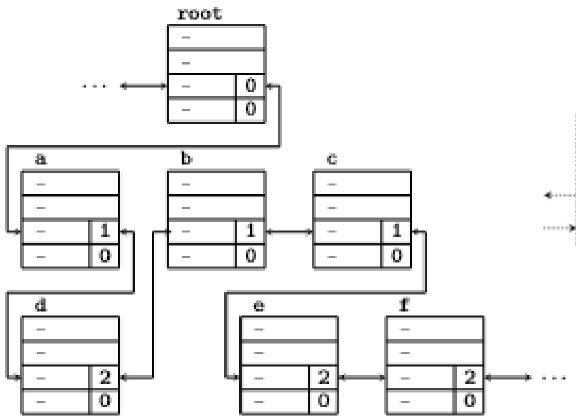


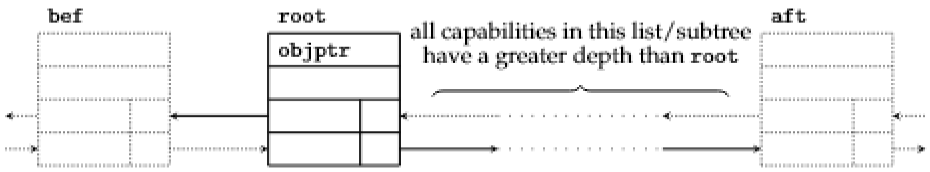
- If a thread doesn't have a capability to an object in its capability space, then it cannot directly access that object
- (cf. if there is no mapping to a particular physical address in a thread's address space, then it cannot access that location)

Derived Capabilities

- An implementation represents the tree as a doubly linked list with "depth" information at each node
- Fixed storage (two pointers + depth) per node
- (Limited) traversal of tree structure without recursion



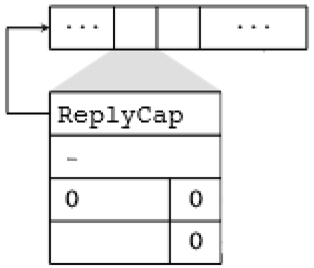




From CS 410/510 Fall 2018, Portland State University

A case study (continued)

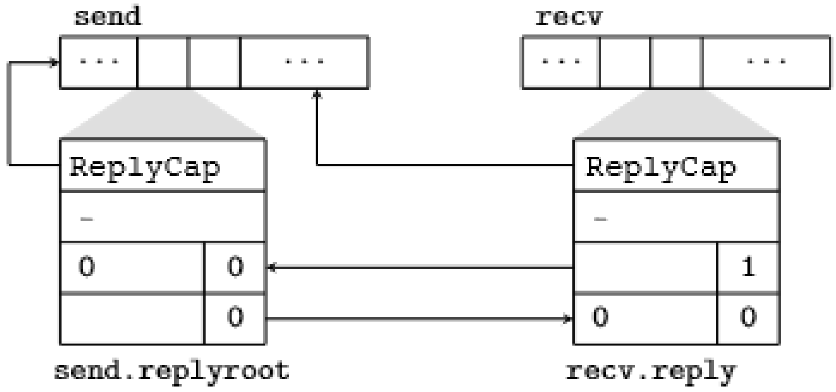
- Reply capabilities are a new capability type that store a pointer to the sending TCB
- Every TCB contains two capability slots:
  - a “replyroot” capability that holds a ReplyCap
  - a “reply” slot that is initially empty



From CS 410/510 Fall 2018, Portland State University

A case study (continued)

- If one thread makes a “Call” to another, the kernel will insert a child of the sender’s master capability in receiver’s reply slot \
- The receiver can use a “Reply” system call to send a message back to the sender, without knowing its identity
- The kernel can revoke the master reply capability, to remove the child, even if the receiver has moved it to a different slot

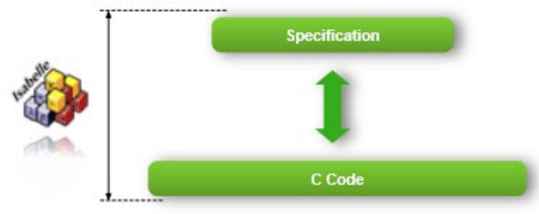


From CS 410/510 Fall 2018, Portland State University

Review

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

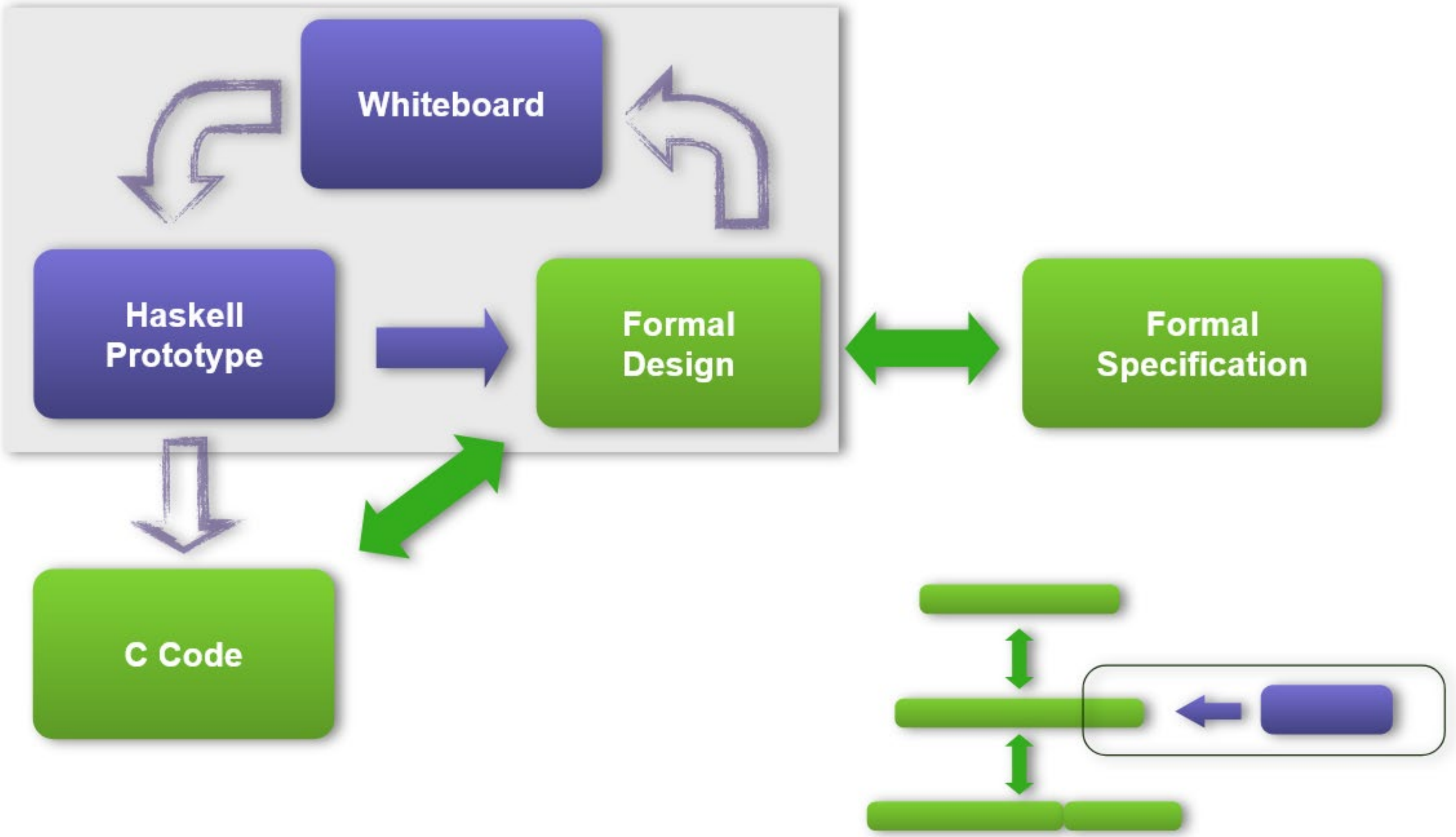


Not implied:

- “secure” (define secure)
- zero bugs from expectation to physical world
- covert channel analysis

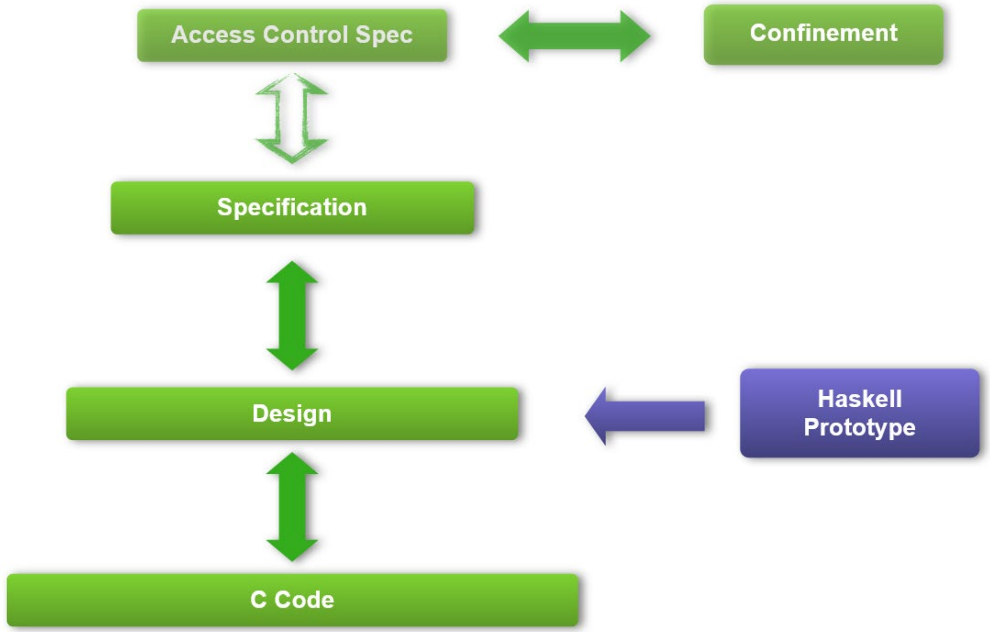
From SOSP’09 Presentation of seL4

Review



From SOSP'09 Presentation of seL4

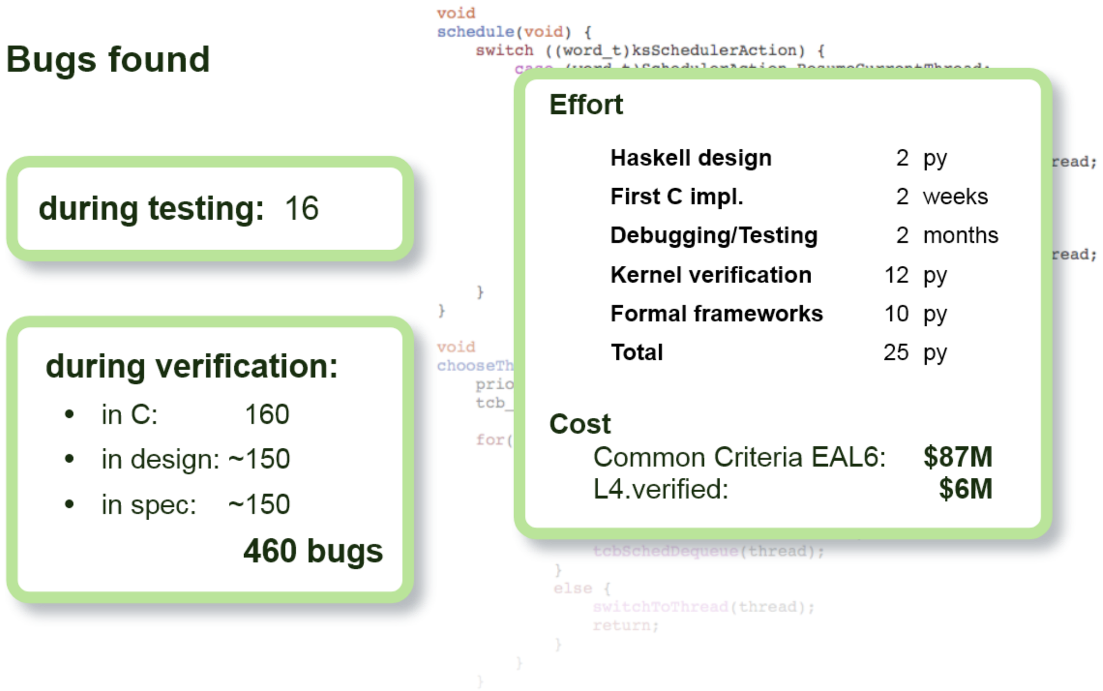
Review



From SOSP'09 Presentation of seL4

Data and development effort

- Initial Haskell kernel
  - Limited functionality (no IRQ, single address space)
- Abstract spec (4 pm) & first refinement (8 py)
  - Prototype (2 py) & C implementation (**2 pm**)
  - 300 changes
- Execution Spec
  - 200 changes
- Full functionality & second refinement (2 py)
  - Misreading, failing to update, typo



From SOSP’09 Presentation of seL4

Discussion and summary

- Can we consider seL4 as OS verification done right?
- What are the implications from its development experience?
- Future opportunities?