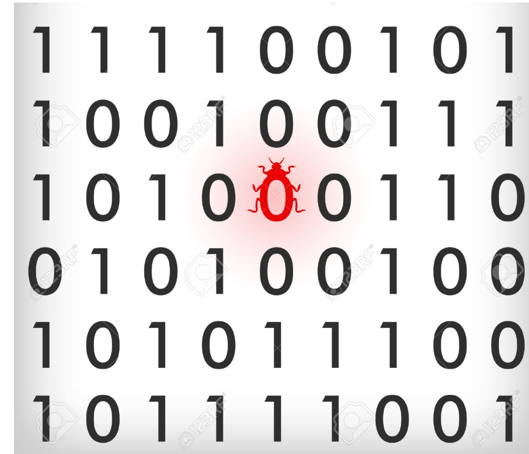
Welcome to the 2019 DeepSpec Workshop!

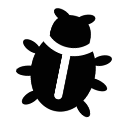
The Science of
Deep Specification

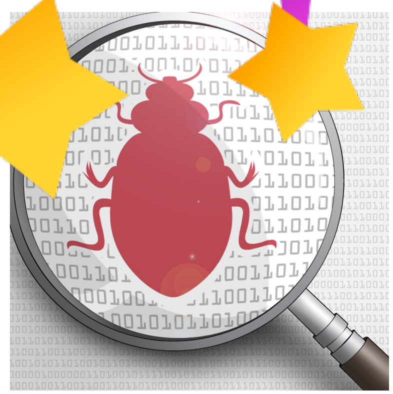
Benjamin C. Pierce
University of Pennsylvania

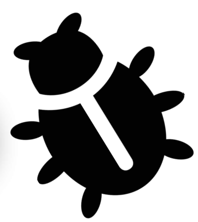
DeepSpec Workshop @ PLDI
June, 2019

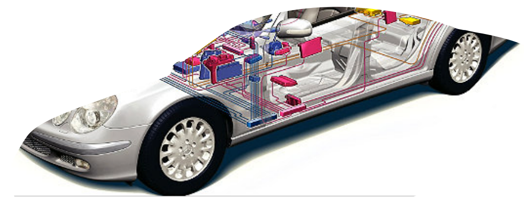


“We can’t build software that works!”

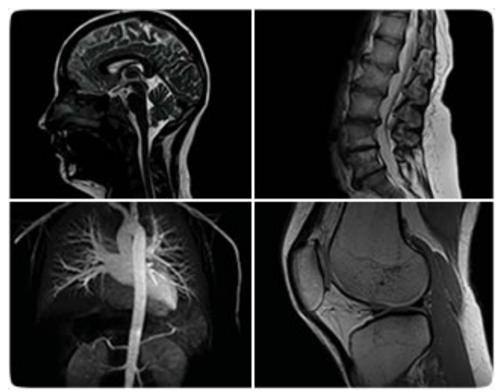








Or...?

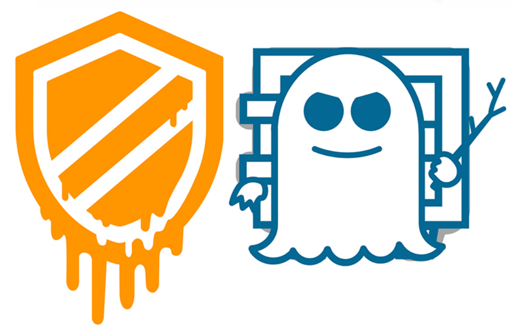


How did that happen?

- Better programming languages
  - Powerful mechanisms for *abstraction* and *modularity*
- Better software development methodology
  - Agile workflows, unit testing, …
- Stable platforms and frameworks
  - Posix, Win32, Android, iOS, apache, DOM/JS, …

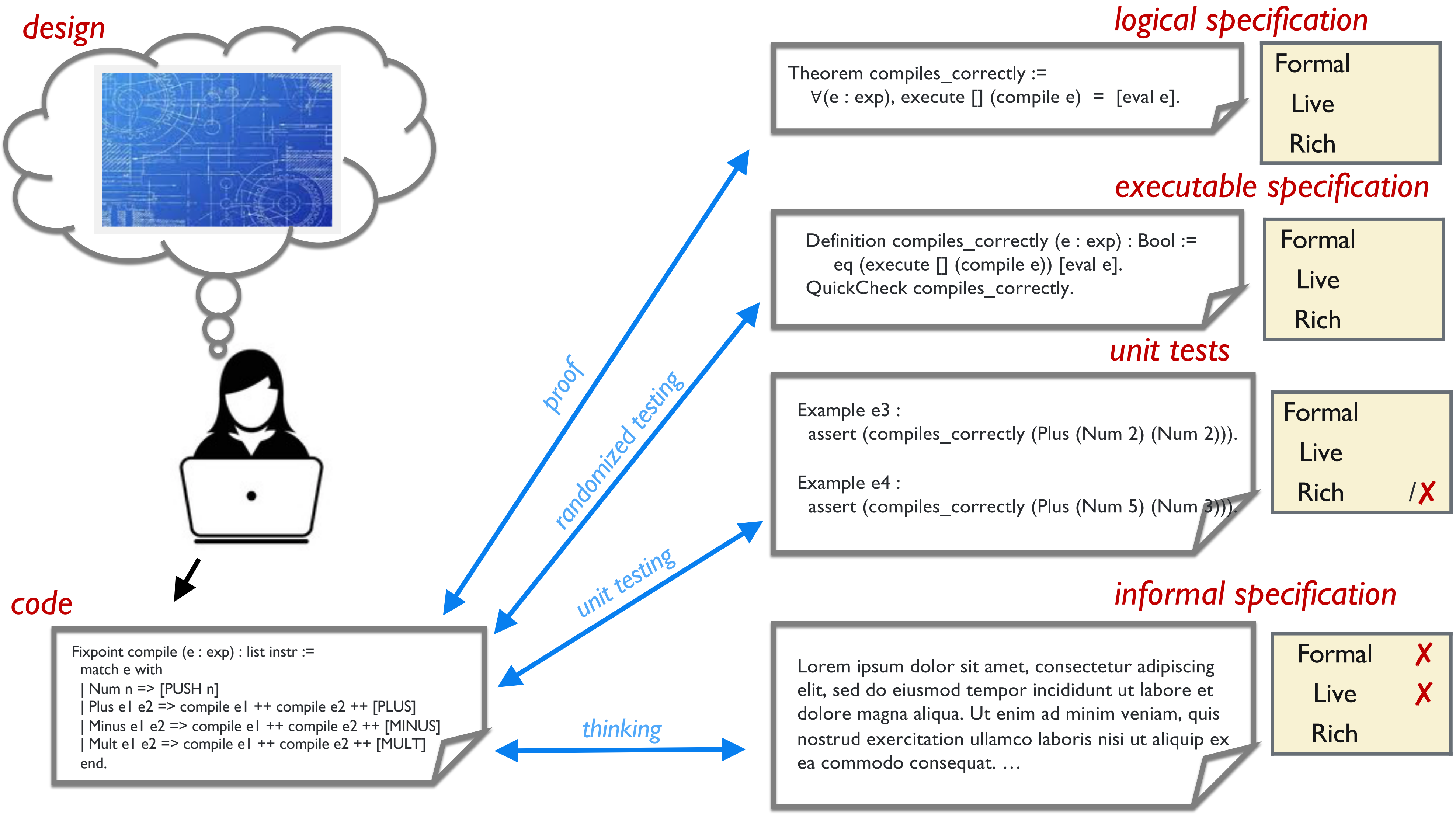

Are we done?
Nope

What about secure software?



Grounds for hope…

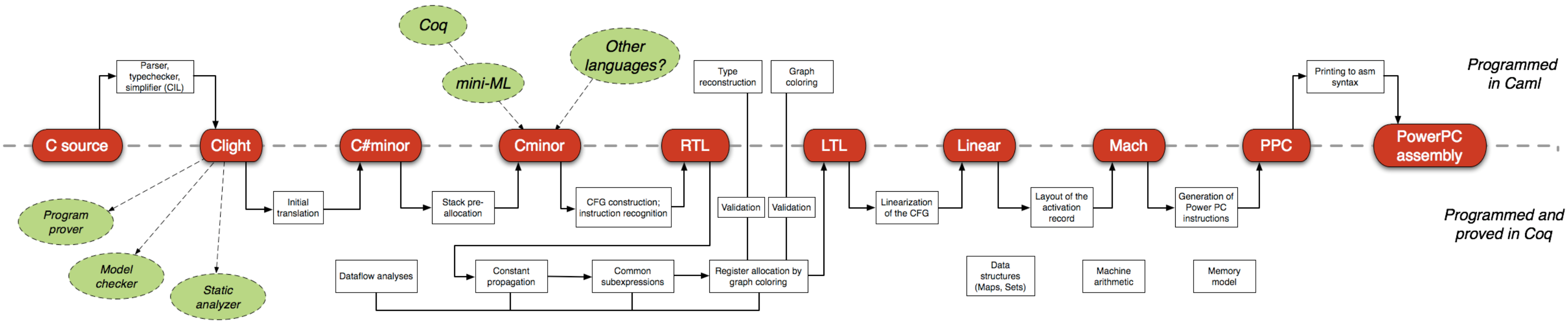
- Better programming languages
  - Basic *safety guarantees* built in
- Better understanding of risks and vulnerabilities
- Better system architectures for security
  - Separation kernels, hypervisors, sandboxing, TPMs, …
- Success stories of formal specification and machine-checked verification of critical software at scale
  - Not a panacea (side channels, etc.)
  - But a big step in the right direction!



design
code
Fixpoint compile (e : exp) : list instr :=
match e with
| Num n => [PUSH n]
| Plus e1 e2 => compile e1 ++ compile e2 ++ [PLUS]
| Minus e1 e2 => compile e1 ++ compile e2 ++ [MINUS]
| Mult e1 e2 => compile e1 ++ compile e2 ++ [MULT]
end.
proof
randomized testing
unit testing
thinking
logical specification
Theorem compiles_correctly :=
∀(e : exp), execute [] (compile e) = [eval e].
Formal
Live
Rich
executable specification
Definition compiles_correctly (e : exp) : Bool :=
eq (execute [] (compile e)) [eval e].
QuickCheck compiles_correctly.
Formal
Live
Rich
unit tests
Example e3 :
assert (compiles_correctly (Plus (Num 2) (Num 2))).
Example e4 :
assert (compiles_correctly (Plus (Num 5) (Num 3))).
Formal
Live
Rich /✗
informal specification
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. …
Formal ✗
Live ✗
Rich

Are logical specifications practical?

COMPCERT



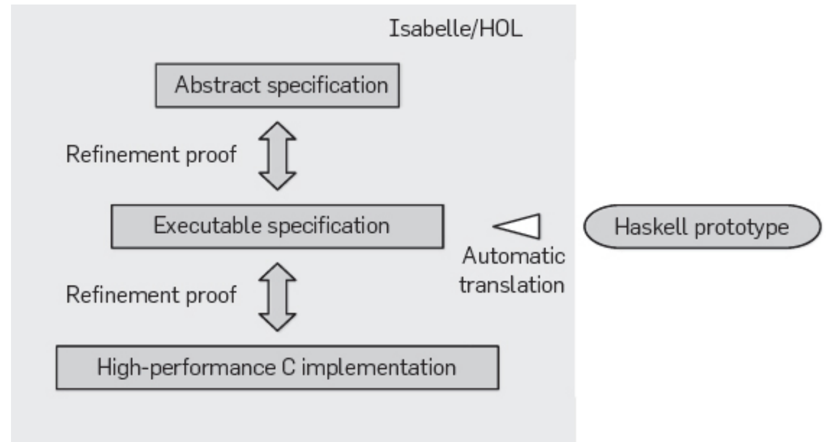
- Accepts most of ISO C 99
- Produces machine code for PowerPC, ARM, x86 (32-bit), and RISC-V architectures
- 90% of the performance of GCC (v4, opt. level 1)

Verification really works!

Regehr's Csmith project used random testing to assess all popular C compilers, and reported:

"The striking thing about our CompCert results is that the middle-end bugs we found in all other compilers are absent. As of early 2011, the under-development version of CompCert is the only compiler we have tested for which Csmith cannot find wrong-code errors. This is not for lack of trying: we have devoted about six CPU-years to the task. *The apparent unbreakability of CompCert supports a strong argument that developing compiler optimizations within a proof framework, where safety checks are explicit and machine-checked, has tangible benefits for compiler users."*

John Regehr
Univ. of Utah



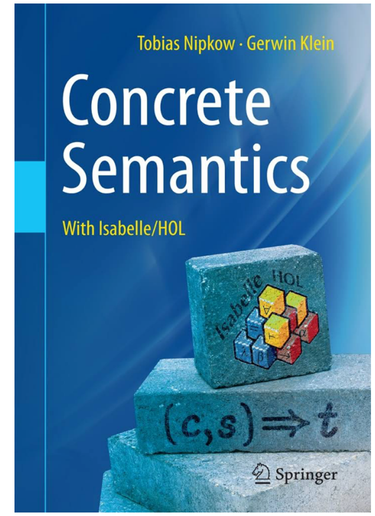
- Real-world operating-system kernel
- With an end-to-end proof of implementation correctness and security enforcement
- Verified down to machine code

And many, many more!

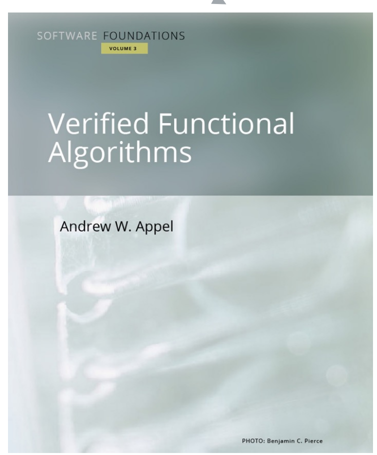
- Bedrock system
- Ur/Web compiler
- CompCert TSO compiler
- CompCert static analysis tools
- Jitk and Data6 verified filesystems
- Fscq file system from MIT
- Verdi distributed system framework
- Testable formal spec for AutoSAR
- CakeML compiler
- Vellvm: Verified LLVM optimizations
- IronClad Apps
- Full-scale formal specifications of critical system interfaces
  - X86 instruction set
  - TCP protocol suite
  - Posix file system interface
  - Weak memory consistency models for x86, ARM, PowerPC
  - ISO C / C++ concurrency
  - Elf loader format
  - C language (Cerberus – also see Krebbers, K semantics, …)

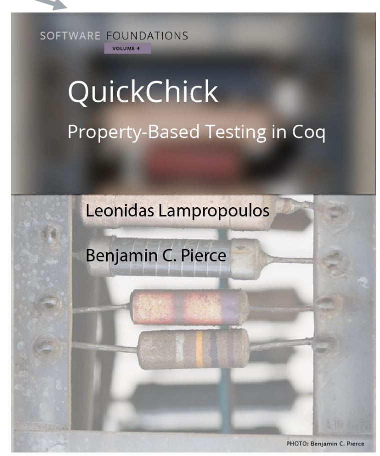
Verified Textbooks!

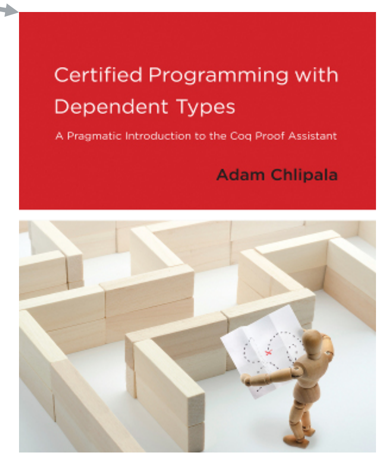
Isabelle



Coq







SoftwareFoundations.org

… and several others!

Why now?

Urgent need for increased confidence

+

Diminishing value of “paper proofs”

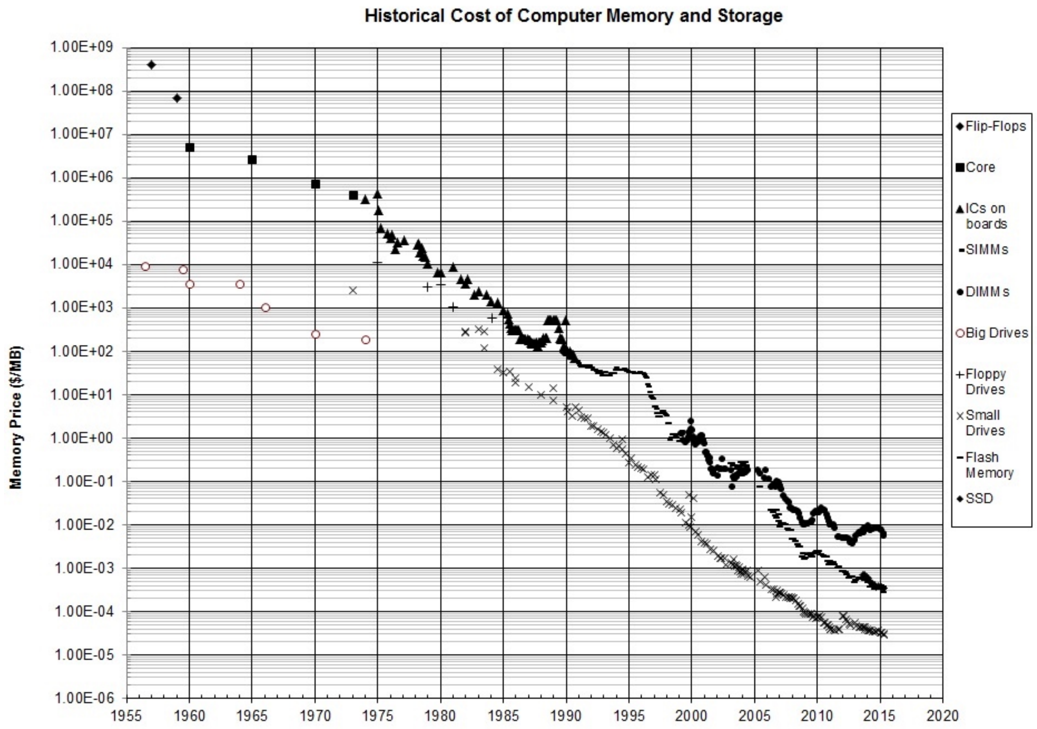
+

Progress on enabling technologies

Enabling Technologies

- Logics
  - Concurrent separation logic, …
- Proof assistants
  - Coq, Isabelle, ACL2, Twelf, HOL-light, …
- Testing tools and methodologies
  - QuickCheck, QuickChick, …
- DSLs for writing specifications
  - OTT, Lem, Redex, …
- Languages with integrated specifications
  - Dafny, Boogie, JML, F*, Liquid Types, Verilog PSL, Dependent Haskell, ...

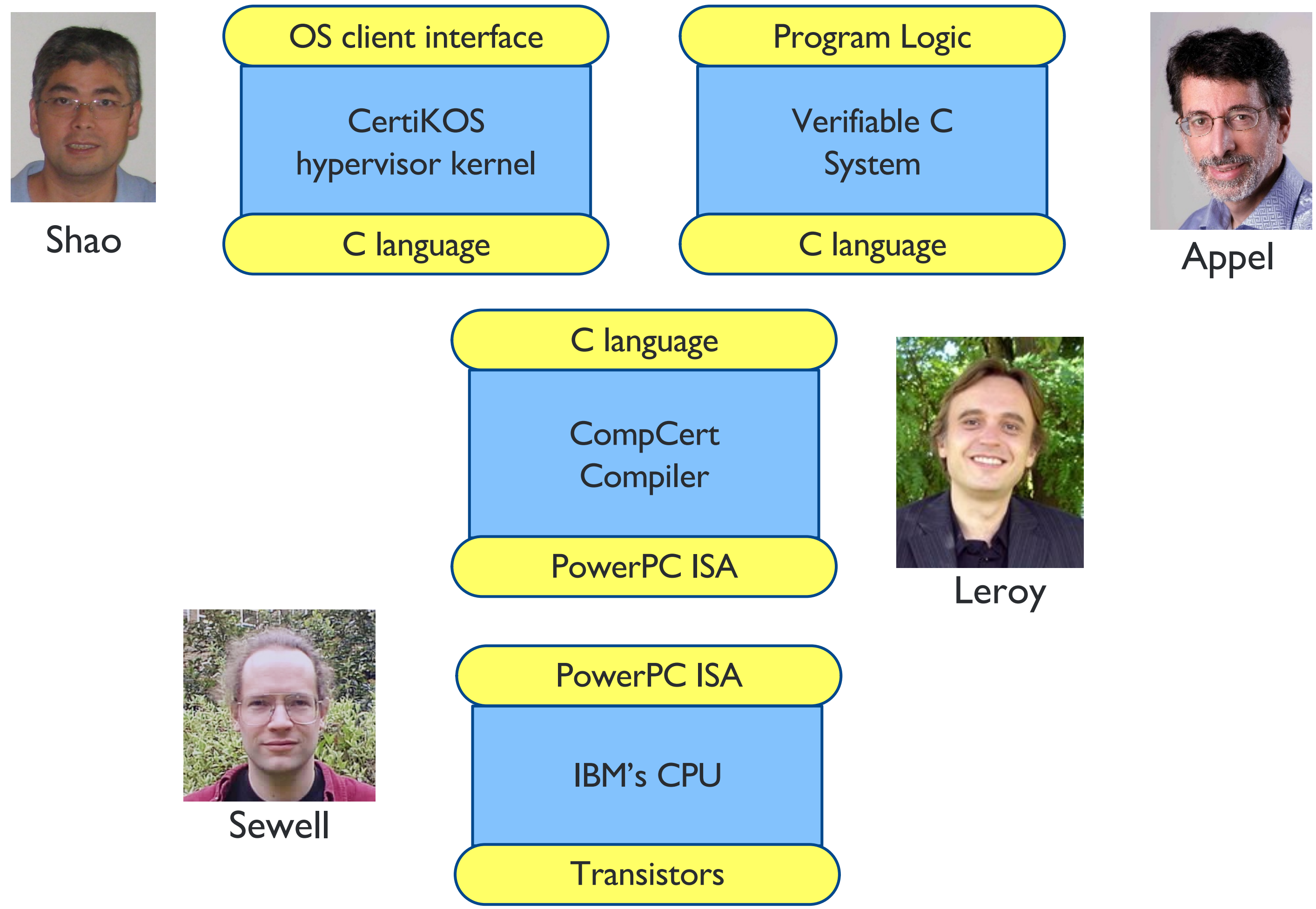
Enabling Technologies



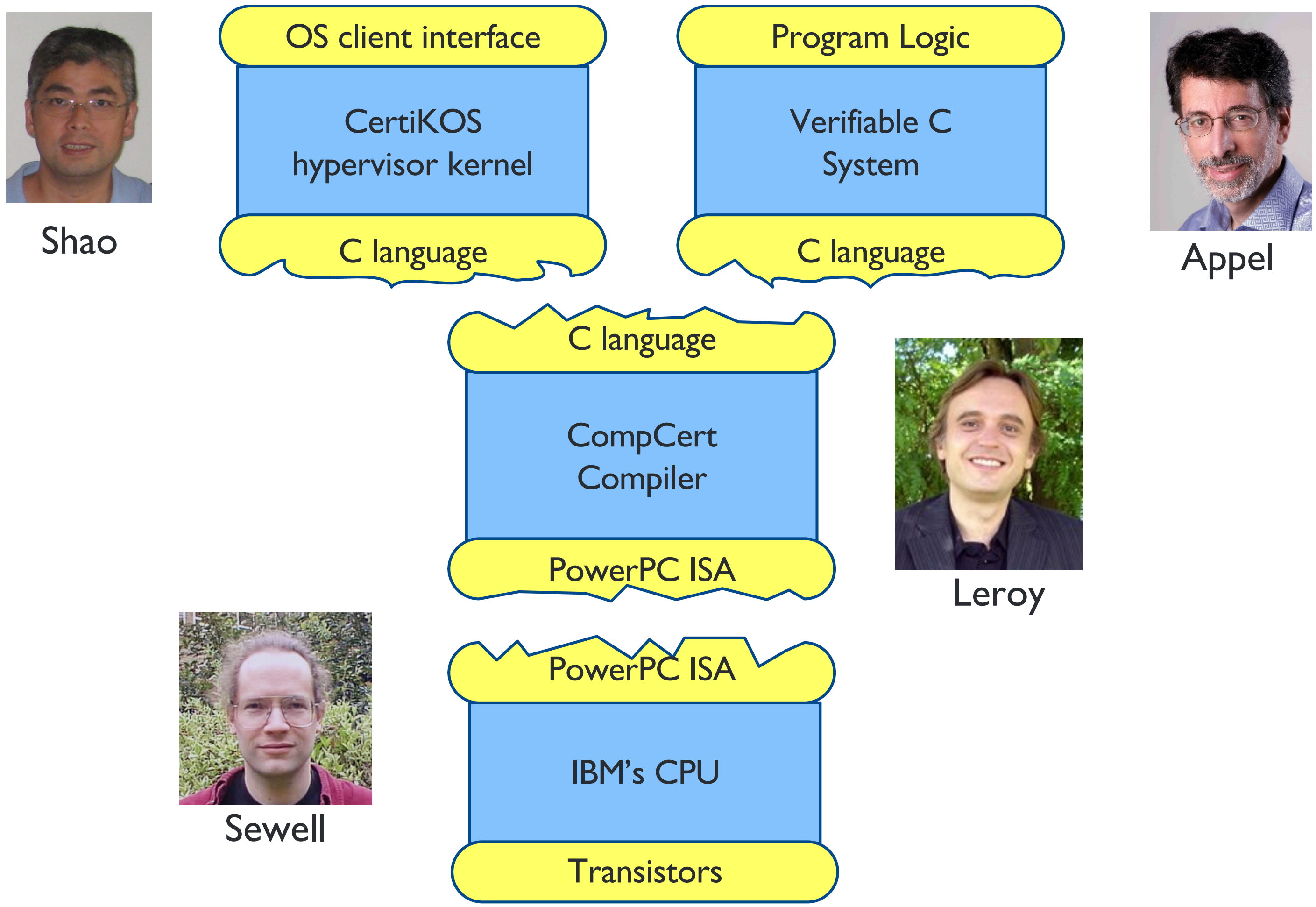
So are we done?

Nope.

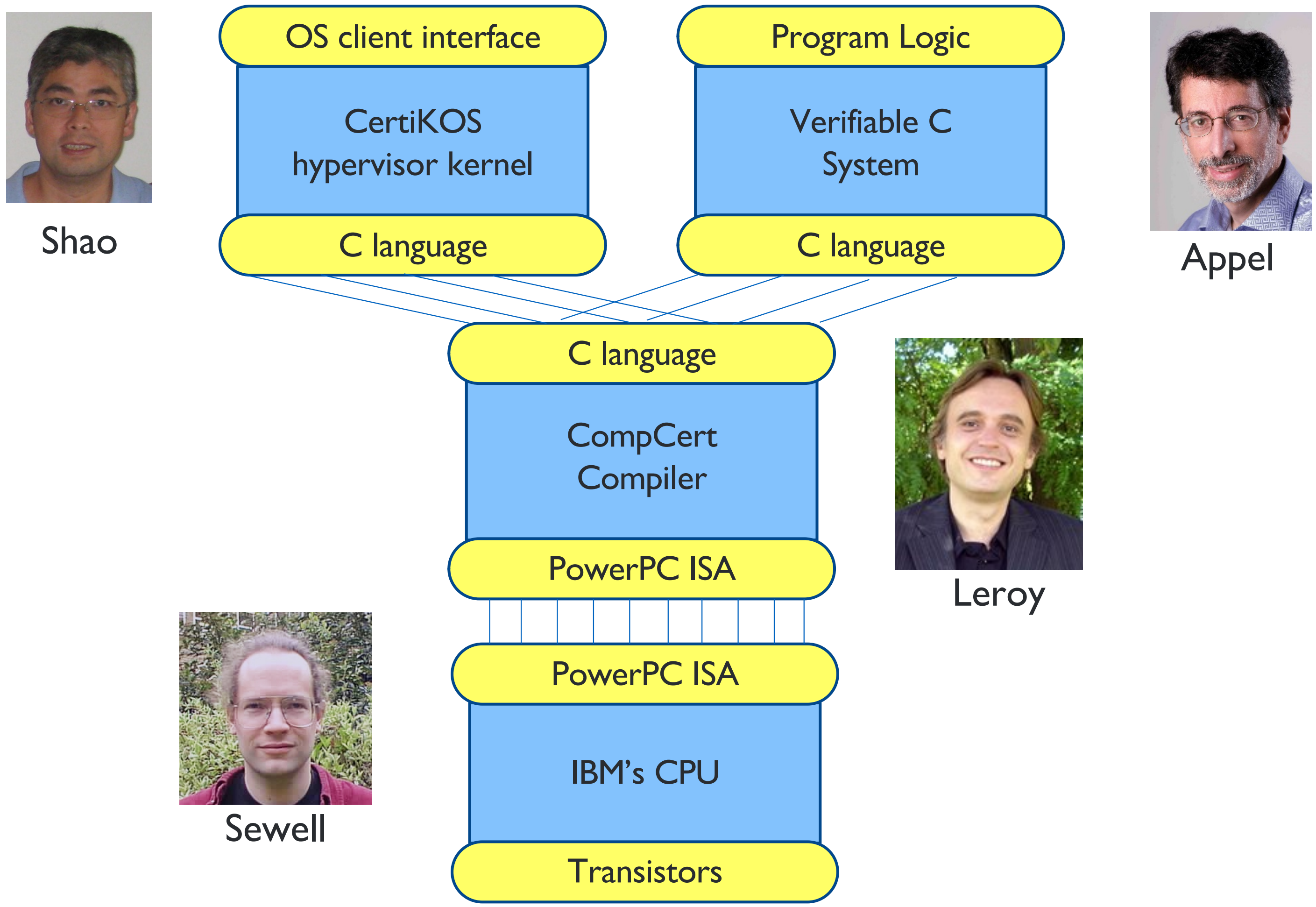
Lessons from CompCert



Lessons from CompCert



Lessons from CompCert

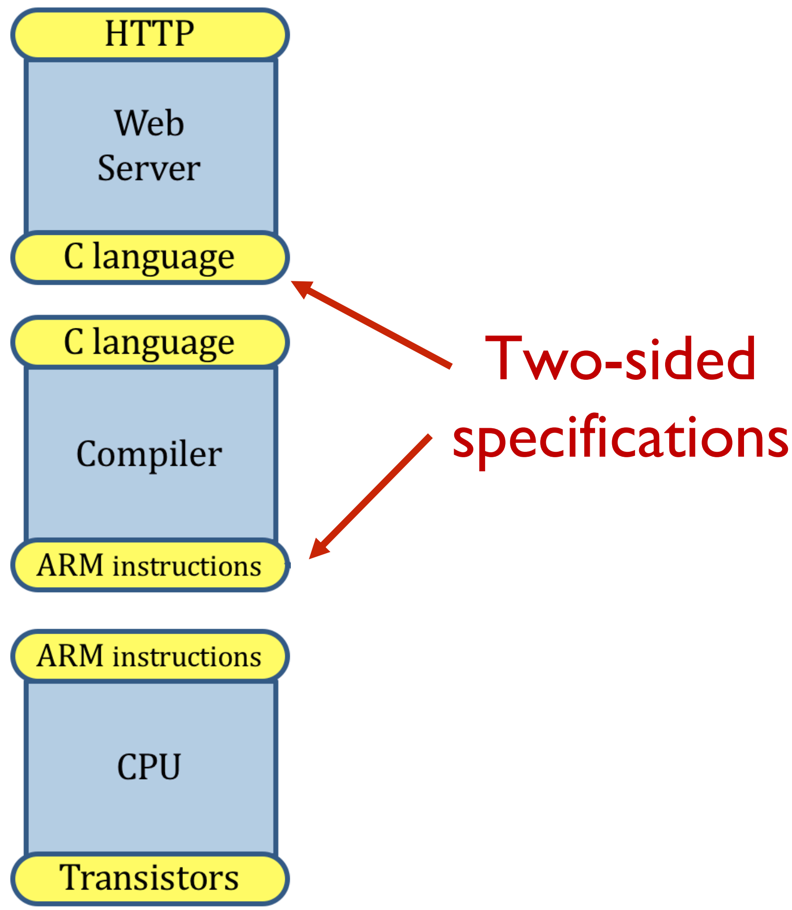


Lessons from seL4

- Original specification and correctness proof for seL4 kernel took **~20 person years**
- Later, the same team added a tool for setting up secure system configurations
  - where processes at different security levels were guaranteed not to interfere
- Proving correctness of this tool took ~4 person years, of which **1.5 years** were devoted to **upgrading the kernel specification** (and proof) to eliminate unwanted nondeterminism

Two-sided specifications

Verified components must <u>connect</u> at specification boundaries



“Deep” specifications:

Formal	mathematically rigorous
Rich	precisely expressing intended behavior of complex software
Live	automatically checked against actual code (not just a model)
Two-sided	exercised by both “implementors” and “clients”

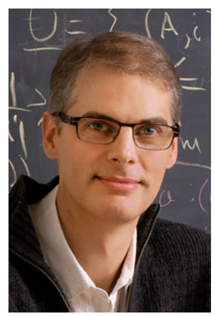
The Science of Deep Specification

Andrew Appel
Princeton

Steve Zdancewic
University of Pennsylvania

Lennart Beringer
Princeton

Adam Chlipala
MIT



Yours truly
University of Pennsylvania

Zhong Shao
Yale

Stephanie Weirich
University of Pennsylvania

And more importantly…

Andres Erbsen
Antal Spector-Zabusky
Antoine Voizard
Benjamin Sherman
Christine Rizkallah
David Costanzo
David Kaloper Meršinjak
Dmitri Garbuzov
Hernán Vanzetto
Jade Philipoom
Jason Gross
Ji-Yong Shin
Jieung Kim
Joachim Breitner
Joonwon Choi
Joshua Lockerman
Jérémie Koenig

Leonidas Lampropoulos
Li-yao Xia
Lionel Rieg
Lucas Paul
Matthew Weaver
Mengqi Liu
Mirai Ikebuchi
Murali Vijayaraghavan
Nick Giannarakis
Olivier Savary Belanger
Pedro Henrique Avezedo de Amorim
Paul He
Pierre Wilke
Qinxiang Cao
Quentin Carbonneaux

Richard Zhang
Ronghui Gu
Samuel Gruetter
Santiago Cuellar
Unsung Lee
Vilhelm Sjöberg
William Mansky
Wolf Honore
Xiongnan (Newman) Wu
Yao Li
Yishuai Li
Yuanfeng Peng
Yuting Wang
Zoe Paraskevopoulou

Goal:

Move from

point success stories

to

sustainable engineering practice
at industrially relevant scale

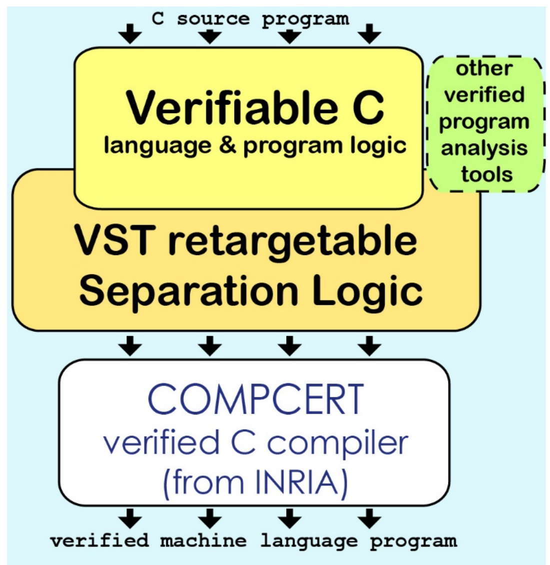
Verified Software Toolchain

Andrew Appel

Lennart Beringer

Program logic for proving correctness of (concurrent) C programs

Proof automation tools for applying the program logic



Demo projects:

crypto primitives,

"mailbox" communication system,

garbage collector for Certicoq

B^+ Trees DBMS

web server

Operating System

Zhong Shao

"Certified Abstraction Layers"

a new refinement-based methodology for software correctness proofs

. . . of programs with low-level concerns such as interrupts, virtual-memory mapping, scheduling, . . .

Demo project:

Certified Kit Operating System (CertiKOS)

Configurable as supervisor or hypervisor

Runs on Intel, AMD, ARM platforms

. . . multicore

Hosts Linux (or other client/guest software)

LLVM
compiler intermediate language

Vellvm
verified
LLVM

Steve Zdancewic

Vellvm:
Formal specification of LLVM; proofs of correctness of LLVM compiler phases

Demo project:

Use as basis for testing correctness of GHC, using QuickChick

LLVM
COMPILER INFRASTRUCTURE

widely used compiler framework based on Static Single-Assignment (SSA)

Core Spec

Haskell Language Specification

Stephanie Weirich

Haskell: widely used pure functional programming language with lazy evaluation

Haskell Core: near-source-level intermediate language inside GHC compiler

Haskell Core Spec:

Formal specification of semantics of the Haskell core language

Demo projects:

Prove correctness of some GHC phases using hs-to-coq

Use as basis for testing correctness of GHC, using QuickChick

Verified Compiler for Coq programs


Andrew Appel


Greg Morrisett

Gallina:
functional programming language inside Coq

"Extraction:"
Translate Gallina to ML, compile with Ocaml compiler

Extraction is quite good, but it's not verified correct

CertiCoq:

A verified compiler for Gallina

Write your software as a pure functional program in Coq, prove its correctness using Coq, use CertiCoq to compile to efficient machine code

Demo projects:

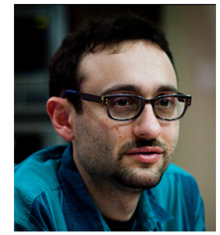
Resolution theorem prover for Separation Logic

(?) CompCert

(?) database query optimization

(?) parts of web server

Verified processor design



Adam Chlipala

Old way:
Write reference manual for ISA

Write RTL program in VHDL

Compile VHDL into transistors } Decent formal tools exist for verifying this part

New way:
Formal specification of ISA

} Prove correctness of Bluespec program

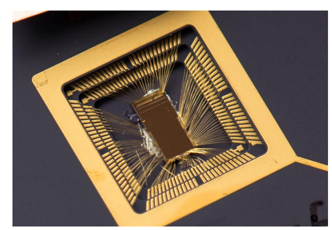
Write RTL program in Bluespec

} Prove correctness of Bluespec *compiler*

Compile Bluespec into VHDL

Compile VHDL into transistors } Use existing tools to verify correctness

Demo project:
specification / verification of
RISC-V processor implementation



Specification-based random testing

Benjamin Pierce

Old way:
Fuzz testing

Recent ways:
Semantic fuzz testing

Tool: QuickCheck, for Haskell and Erlang; fuzzes over (tree) data structures, automatically reduces bugs found into minimal input cases

QuickChick:
Semantic fuzz testing based on conformance to formal specification in Coq

Demo projects:

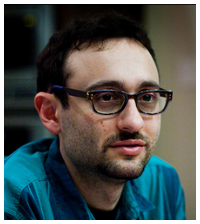
Apache web server
DeepSpec web server

Haskell compiler

deep spec crypto

Verification of cryptographic primitives

Lennart Beringer



Adam Chlipala

High-level cryptographic specs ("pseudorandom function, cryptographic advantage"), Message authentication, Random number generation

High-level functional specs (elliptic curves in finite fields)

Low-level functional specs (multibit carry)

Efficient imperative implementations

Demo projects: these crypto applications serve as demo projects for several of our other tools:

Fiat

Bedrock

Verified Software Toolchain

Vellvm verified LLVM

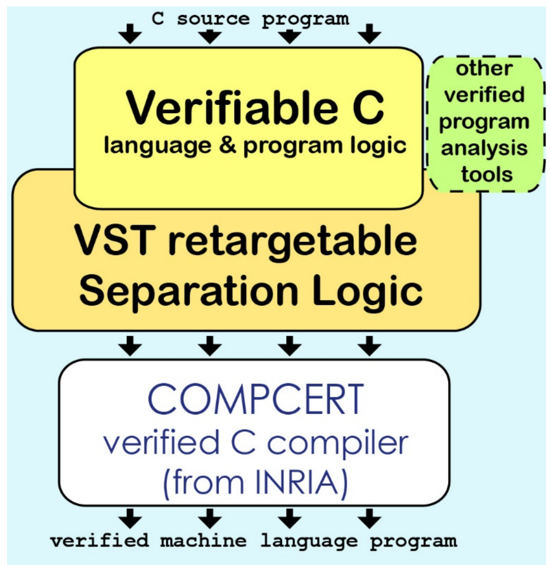
Verified Software Toolchain

Andrew Appel

Lennart Beringer

Program logic for proving correctness of (concurrent) C programs

Proof automation tools for applying the program logic



Demo projects:

crypto primitives,

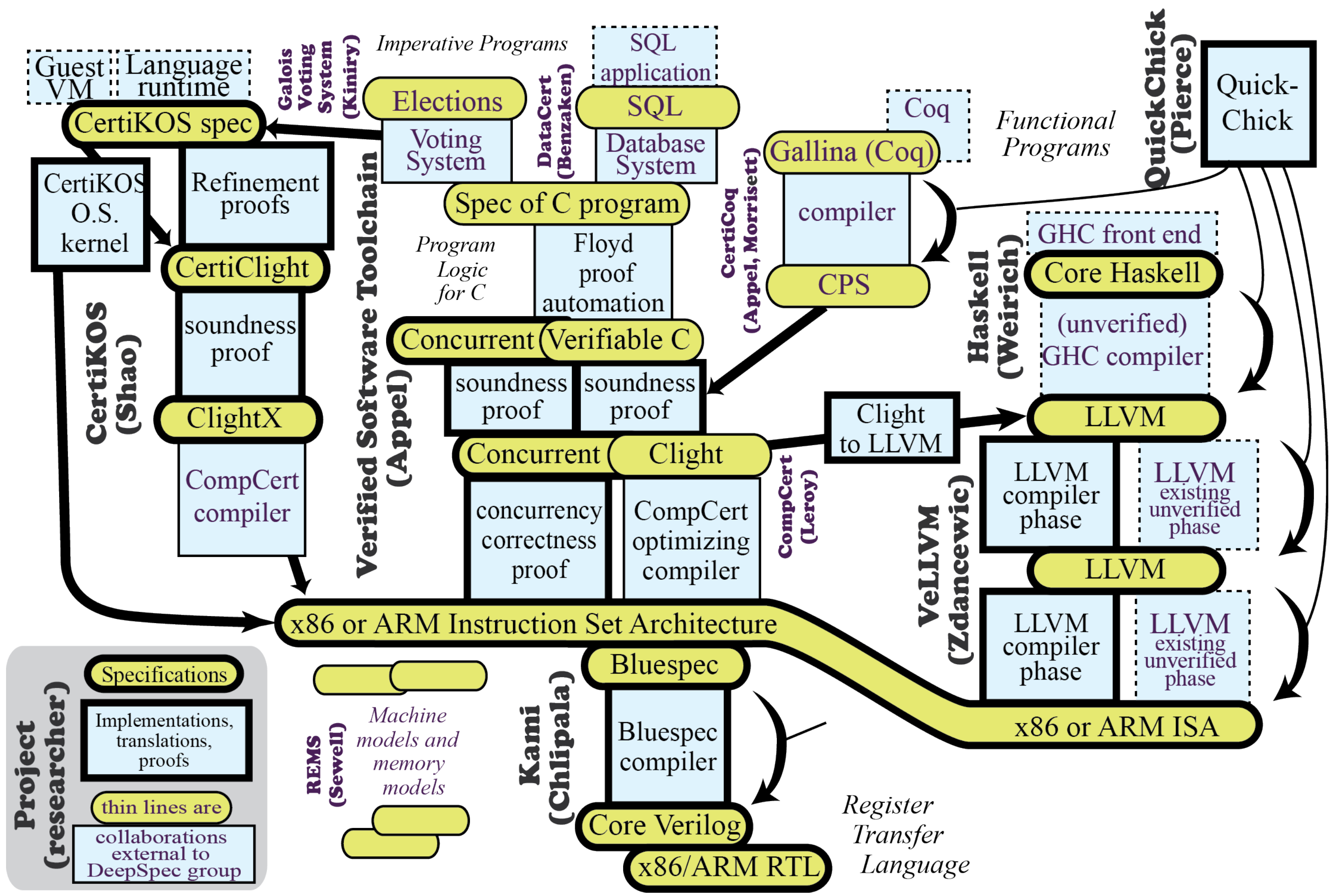
“mailbox”
communication system

garbage collector
for Certicoq

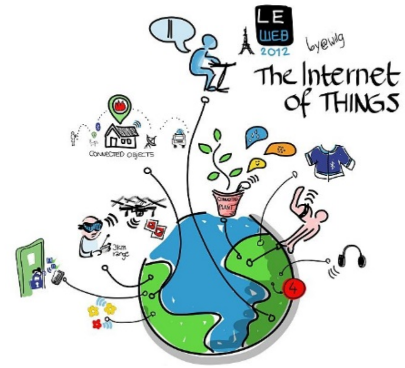
B^+ Trees DBMS

web server

Goal: Rich, formal, live, 2-sided specs



Application demo?



Elections
- Operating system
- C programming
- Database
- Functional programming

Web Server
- Operating system
- C programming
- Database

IoT device
- Operating system
- C programming
- Hypervisor

Application demos!

deep spec server

Web Server

Operating system

C programming

File system

Crypto

Self-driving car

C programming

Hypervisor

DeepWeb

A web server built on DeepSpec

Many parts

CertiKOS

Verified Software Toolchain

Kami

deep spec crypto

Core Spec

Certi Coq

Vellvm verified LLVM

QuickChick

One whole

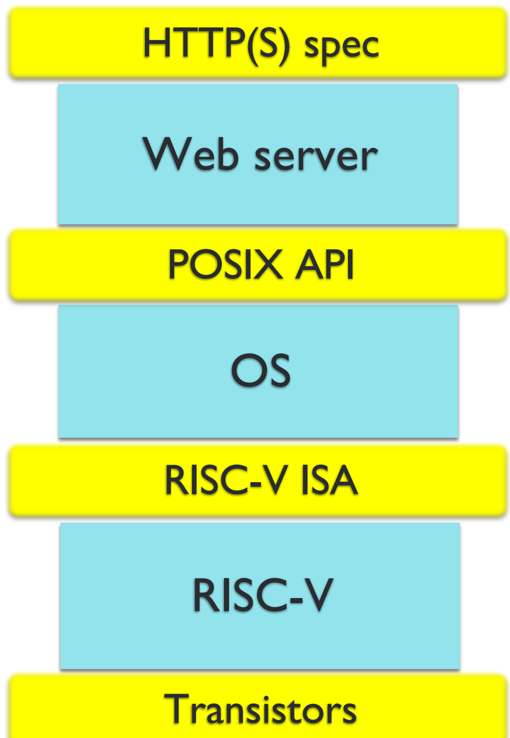
deep spec server

Challenges

- Extreme vertical integration
  - Make progress through a sequence of “integration experiments”
- Multiple levels and styles of specifications
  - Need a “lingua franca” for writing a variety of specs
  - → Interaction trees
- Combining testing and verification
- Reasoning about server behavior “modulo the network”

Challenge:
Vertical Integration

deep spec server =



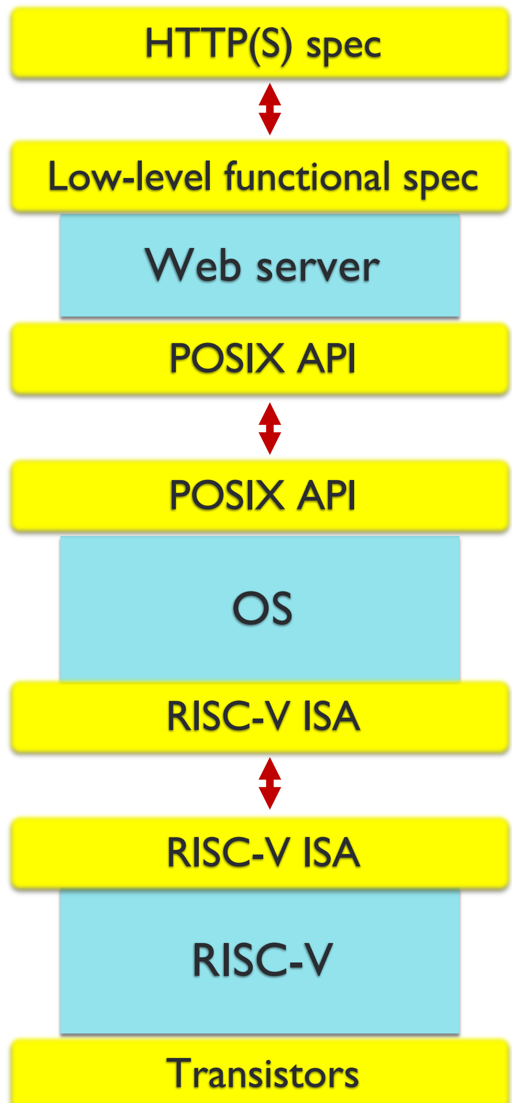
Executable high-level specification of HTTP(S) protocols and web services

System call interface specification

Instruction-set specification

RTL-level description of circuit behaviors

Goal: A "single QED" encompassing the whole stack



Executable high-level specification of HTTP(S) protocols and web services

Functional program with same observable behavior as C web server

System call interface specification (separation logic Hoare triples)

System call interface specification (CertiKOS "layer interface")

Instruction-set specification (assembly level, structured memory model)

Instruction-set specification (machine-code level, flat memory model)

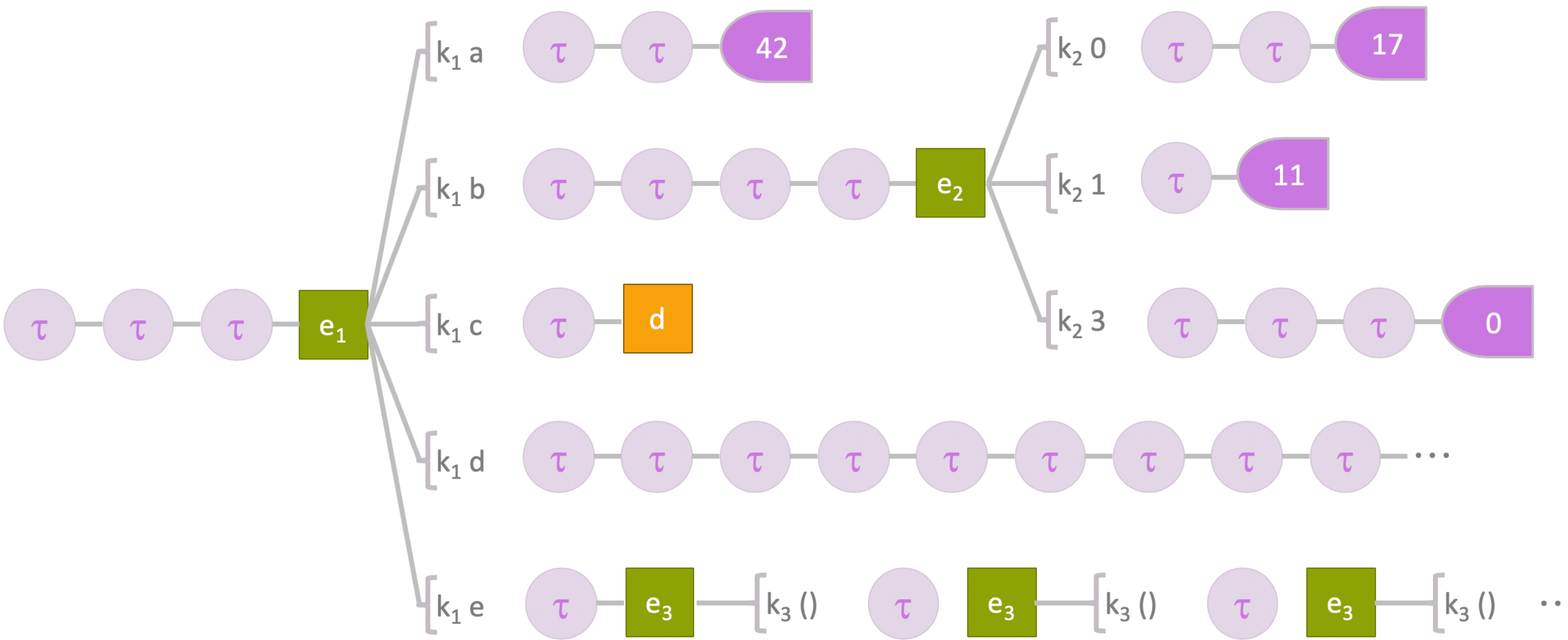
RTL-level description of circuit behaviors

Challenge:
Disparate Specification Styles

Too many metalanguages!

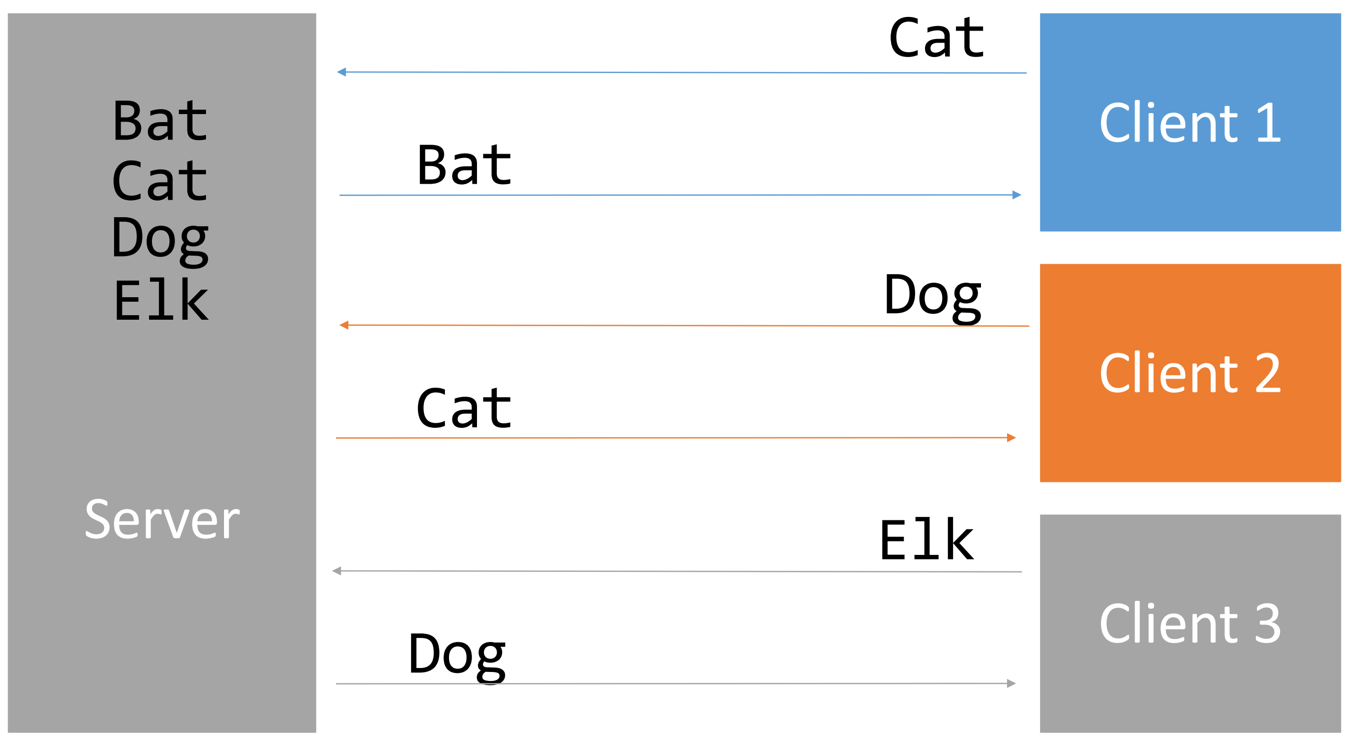
- Network-level HTTP spec
  - Nondeterministic "model implementation" (functional program)
  - Client-side acceptance tester (functional program)
- Web server implementation
  - CompCert "observation traces"
- VST C verification tool
  - Hoare triples in separation logic
- CertiKOS
  - "Layer interfaces"

Interaction trees

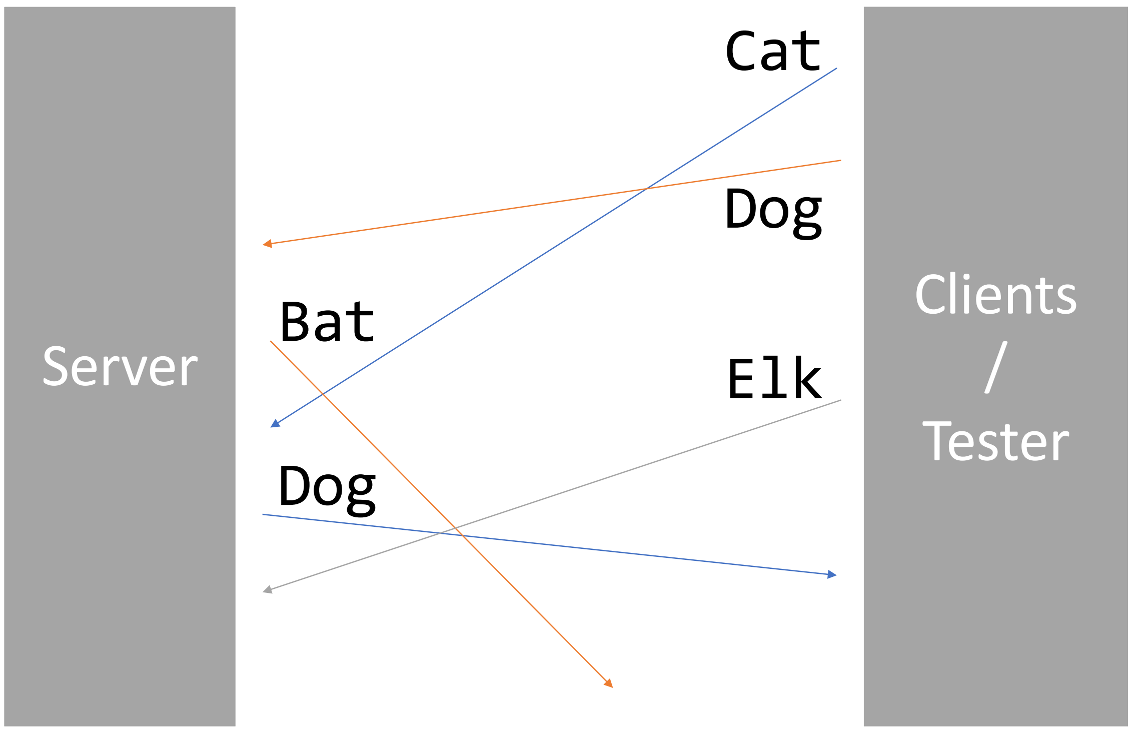


Reasoning
“Modulo the Network”

Swap server specification

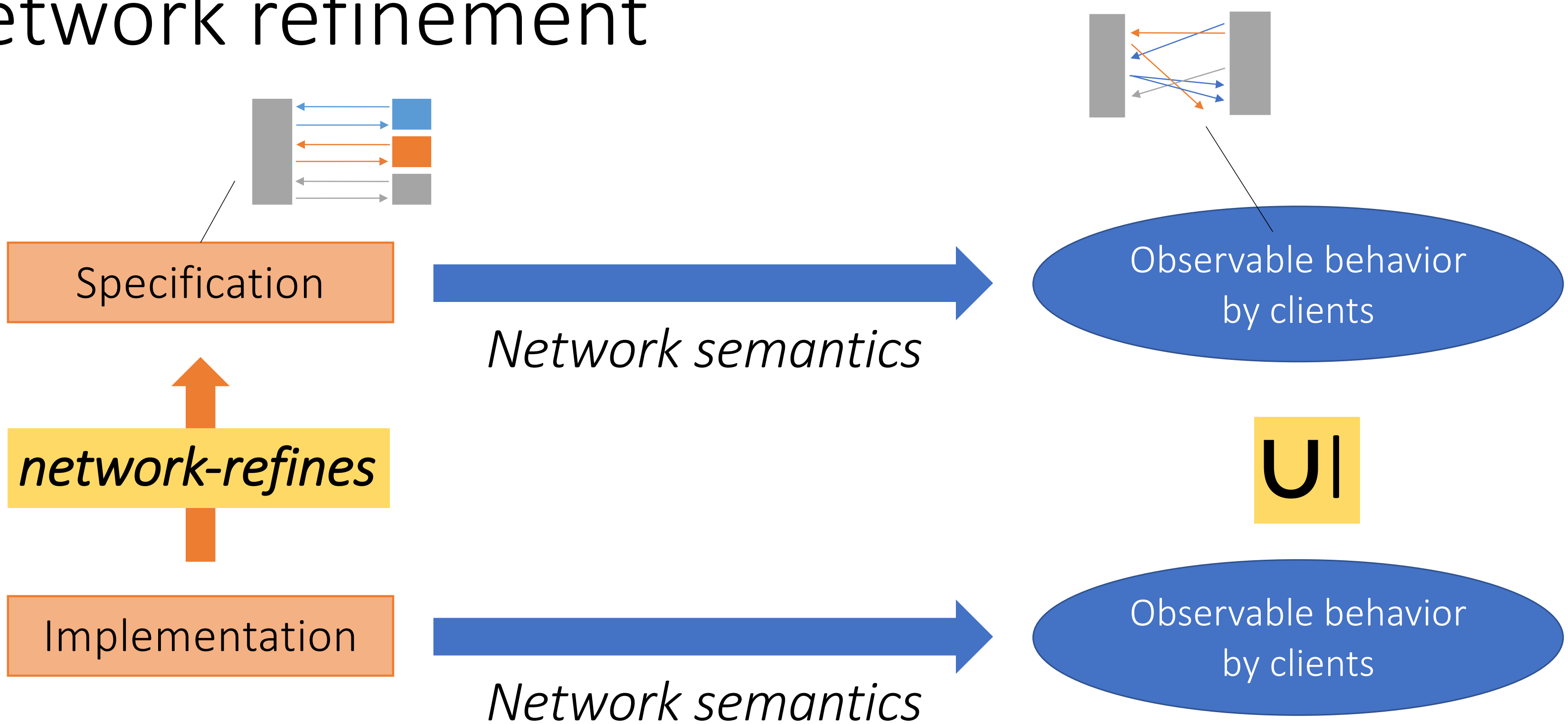


Swap server: in the real world



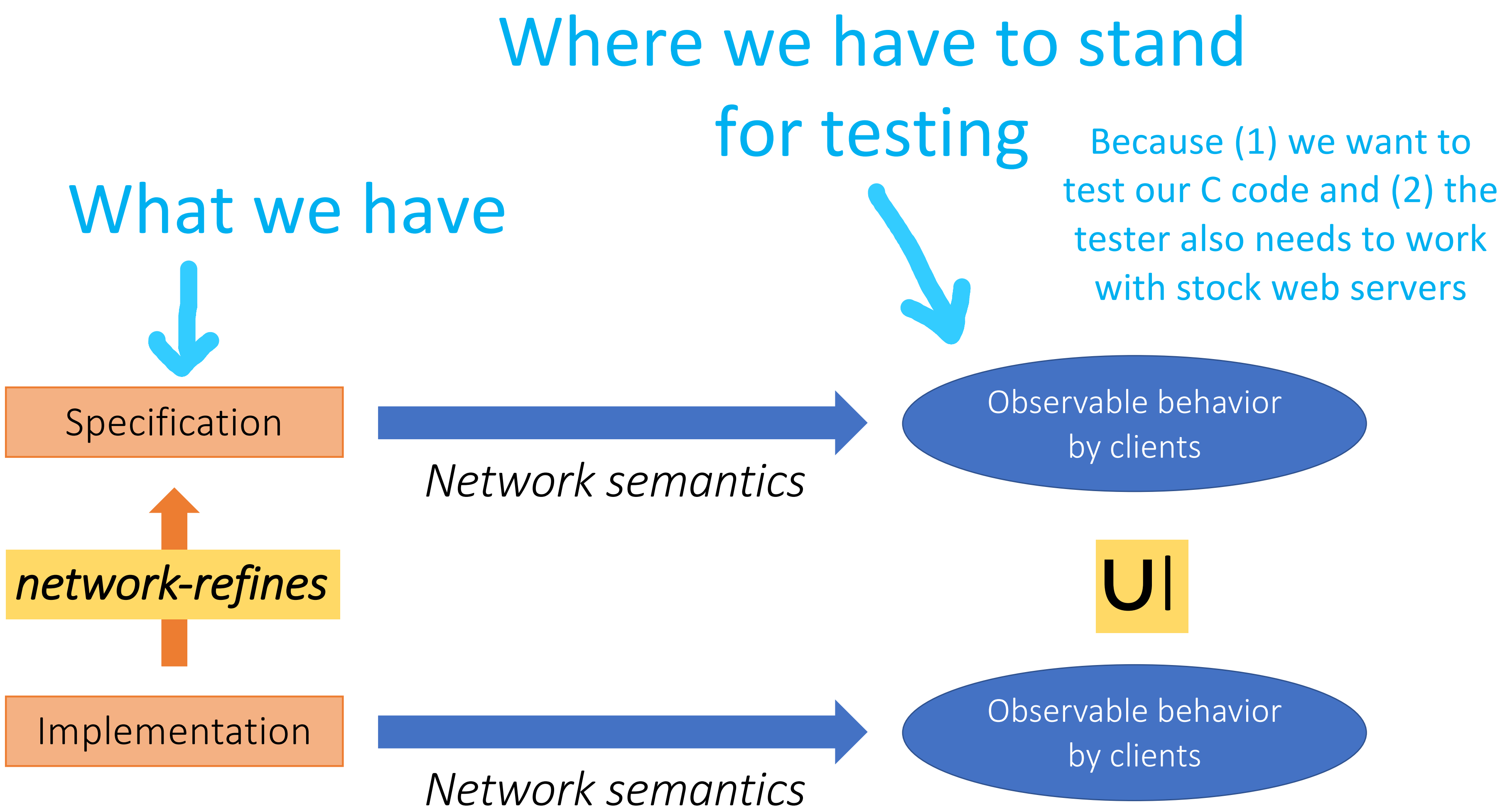
- Messages on different connections can be reordered
- Messages can be delayed indefinitely

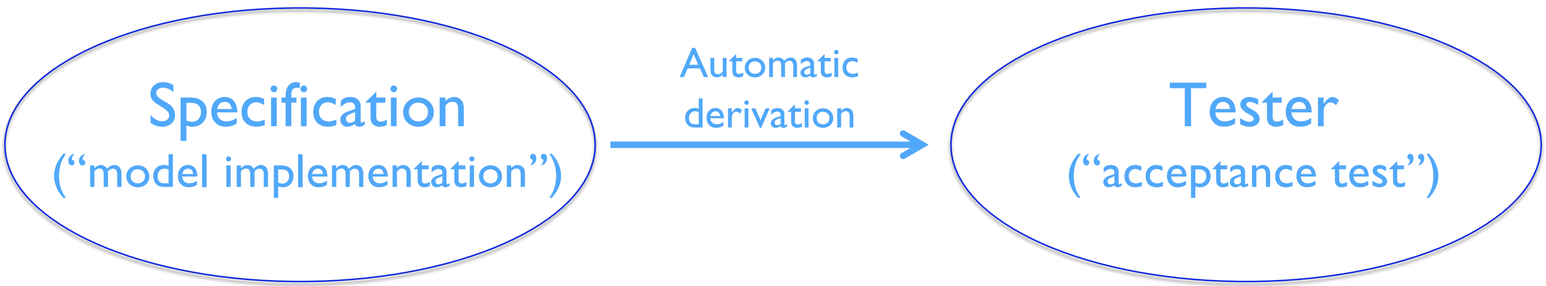
Network refinement



Adaptation of Observational refinement/Linearizability

Challenge:
Testable High-Level Specifications


Where we have to stand
for testing
Because (1) we want to test our C code and (2) the tester also needs to work with stock web servers
What we have
Specification
network-refines
Implementation
Network semantics
Network semantics
Observable behavior by clients
⊇
Observable behavior by clients



Main challenge: nondeterminism

- introduced by the network
- … or present in the original spec

Final theorem

- “If you put these bits (produced by compiling CertiKOS and the web server using CompCert) into a memory connected to this connection of transistors (produced by compiling a RISC-V implementation using Kami), the behaviors of the resulting system will network refine the behaviors describe ed by the model implementation.”

Progress

- Vertical integration
  - See CPP 2019 paper about testing and VST verification of a “swap server”
- Interaction trees
  - https://github.com/DeepSpec/InteractionTrees
  - See talks by Steve Zdancewic today and by Yann Régis-Giannis and Gil Hur tomorrow
- Connecting VST and CertiKOS
  - See talk by William Mansky today
- Connecting CertiKOS and Risc-V
  - Ongoing work at Yale and MIT on a “flat memory” semantics for CompCert

The demo is not the (only) scientific result!

deepspec server $\subsetneq$ deepspec

DeepSpec is not "build a verified stack"

DeepSpec is . . .

a coherent collection of tools and techniques . . .

CERTIKOS

Verified Software Toolchain

COMPCERT

Kami

Core Spec

QuickChick

deep spec crypto

Vellvm verified LLVM

CertiCoq

…that can be connected, combined, and configured to allow users to build and foundationally verify high assurance, functionally correct software and hardware.

DeeoSpec Workshop Overview

Welcome and overview

09:00 - 09:45 ☆ Overview of the DeepSpec Expedition and its Capstone Application
Talk — Benjamin C. Pierce University of Pennsylvania

09:45 - 10:30 ☆ Project Updates from Participating Sites
Talk — Andrew Appel Princeton, Adam Chlipala Massachusetts Institute of Technology, USA, Zhong Shao Yale University

Compiler Verification

11:00 - 11:30 ☆ Closure Conversion is Safe for Space
Talk — Zoe Paraskevopoulou Princeton University, Andrew Appel Princeton

11:30 - 12:00 ☆ Fast, Verified Partial Evaluation
Talk — Adam Chlipala Massachusetts Institute of Technology, USA

12:00 - 12:30 ☆ Stack-Aware CompCert
Talk — Yuting Wang Yale University

Modular Reasoning

14:00 - 14:30 ☆ Abstraction, Subsumption, and Linking in VST
Talk — Lennart Beringer Princeton University, Andrew Appel Princeton

14:30 - 15:00 ☆ Compositional Verification of Preemptive OS Kernels with Temporal and Spatial Isolation
Talk — Mengqi Liu Yale University

15:00 - 15:30 ☆ Modular Correctness Proofs at the Hardware-Software Interface
Talk — Joonwon Choi Massachusetts Institute of Technology, USA

Interaction Trees and Algebraic Effects I

16:00 - 16:20 ☆ Interaction Trees: Representing Recursive and Impure Programs in Coq
Talk — Steve Zdancewic University of Pennsylvania

16:20 - 16:45 ☆ Connecting Separation Logic with First-Order Reasoning on Memory
Talk — William Mansky University of Illinois at Chicago

16:45 - 17:30 ☆ Typed Programming with Algebraic Effects (in terms of ambient values, functions, and control)
Talk — Daan Leijen Microsoft Research, USA

Implementation and Verification of Modular Effectful Systems in Coq using FreeSpec
Yann Régis-Gianas IRIF, University Paris Diderot and CNRS, France / INRIA PI.R2

Names, Places, and Things: Generic Traversals over Generic Syntax with Binding
James McKinna University of Edinburgh

Interaction Trees and Algebraic Effects II

Development of the RISC-V ISA Formal Specification
Rishiyur Nikhil

Automated Formal Memory Consistency Verification of Hardware
Yatin Manerkar Princeton University

Hardware / Software Interface Specifications

Project Oak: Control Data in Distributed Systems, Verify All The Things
Ben Laurie Google Research

Refinement-Based Game Semantics for CompCert
Jérémie Koenig Yale University

Verifying all the things

Coinductive Reasoning about Interaction Trees
Chung-Kil Hur Seoul National University

Coverage Guided, Property Based Testing
Leonidas Lampropoulos University of Pennsylvania

Coinduction and testing

Join us!

Thank you!
(any (more) questions?)

Teaching materials

Technical workshops
(like this one :-)

Summer schools

visitors program

PhD and postdoc positions

Visit **deepspec.org** to see what's happening
and join our mailing list