CertiKOS Overview & Status Update

Zhong Shao

Yale University

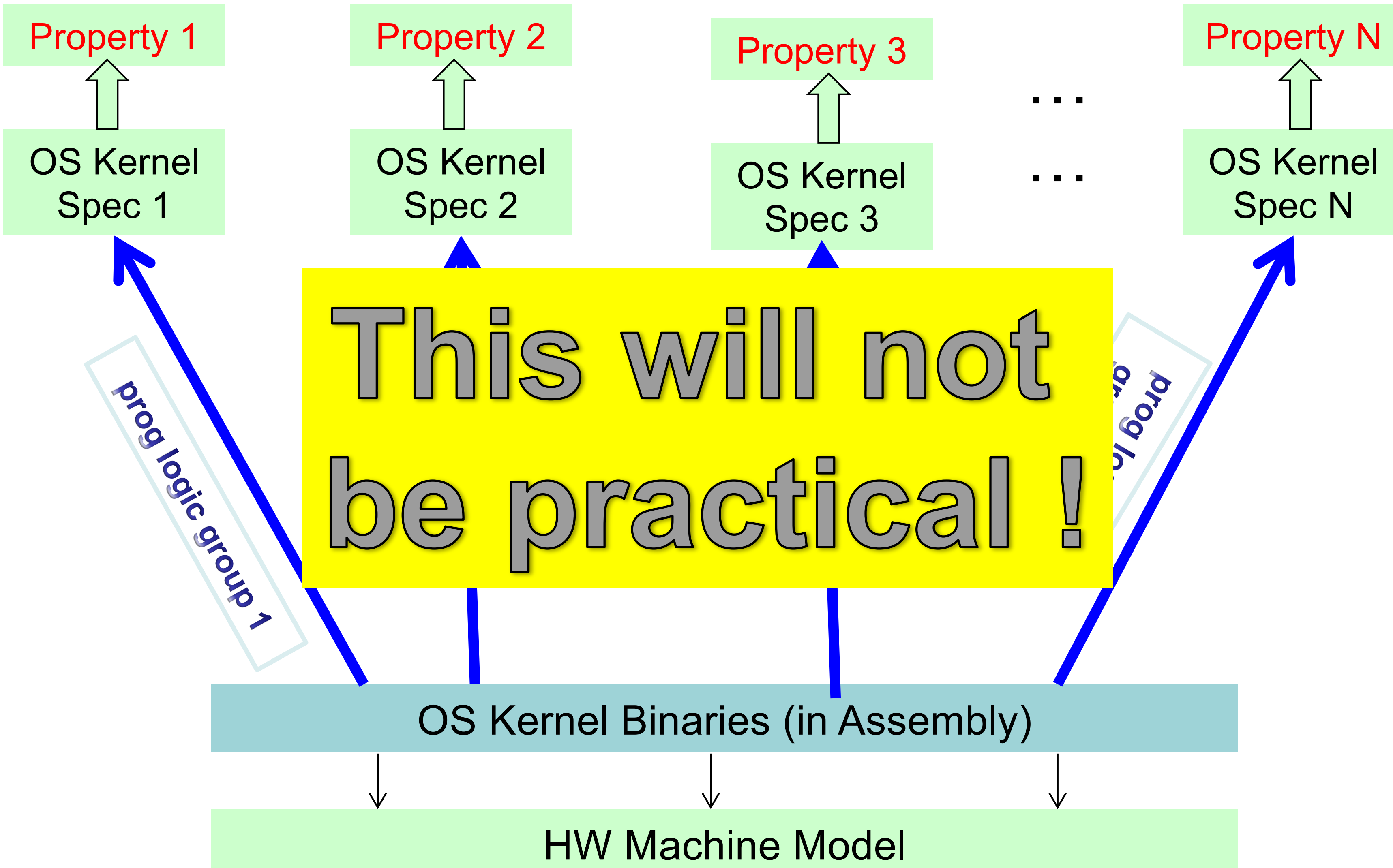
August 3, 2022

http://flint.cs.yale.edu

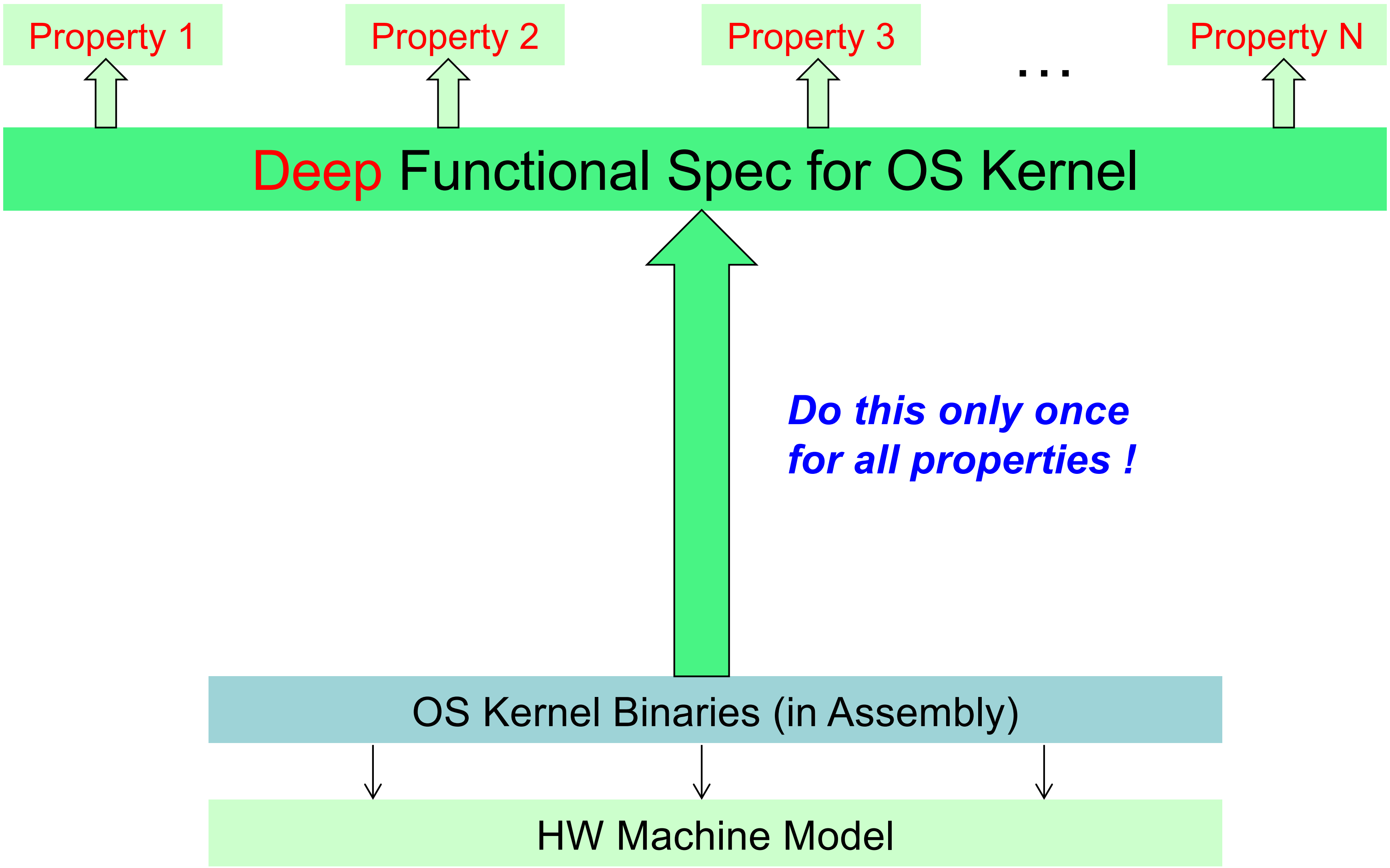
Problem Definition

- What is a certified OS kernel / hypervisor / security monitor?
  - a system binary ***implements*** its specification running over a HW machine model (w. devices & interrupts)?
  - what should the specification & the machine model be like?
- What properties do we want to prove?
  - safety & partial correctness properties
  - total functional correctness
  - security properties (isolation, confidentiality, integrity, availability)
  - resource usage properties (stack overflow, real time properties)
  - race-freedom, atomicity, and linearizability
  - liveness properties (deadlock-freedom, starvation freedom)
- How to cut down the cost of verification?

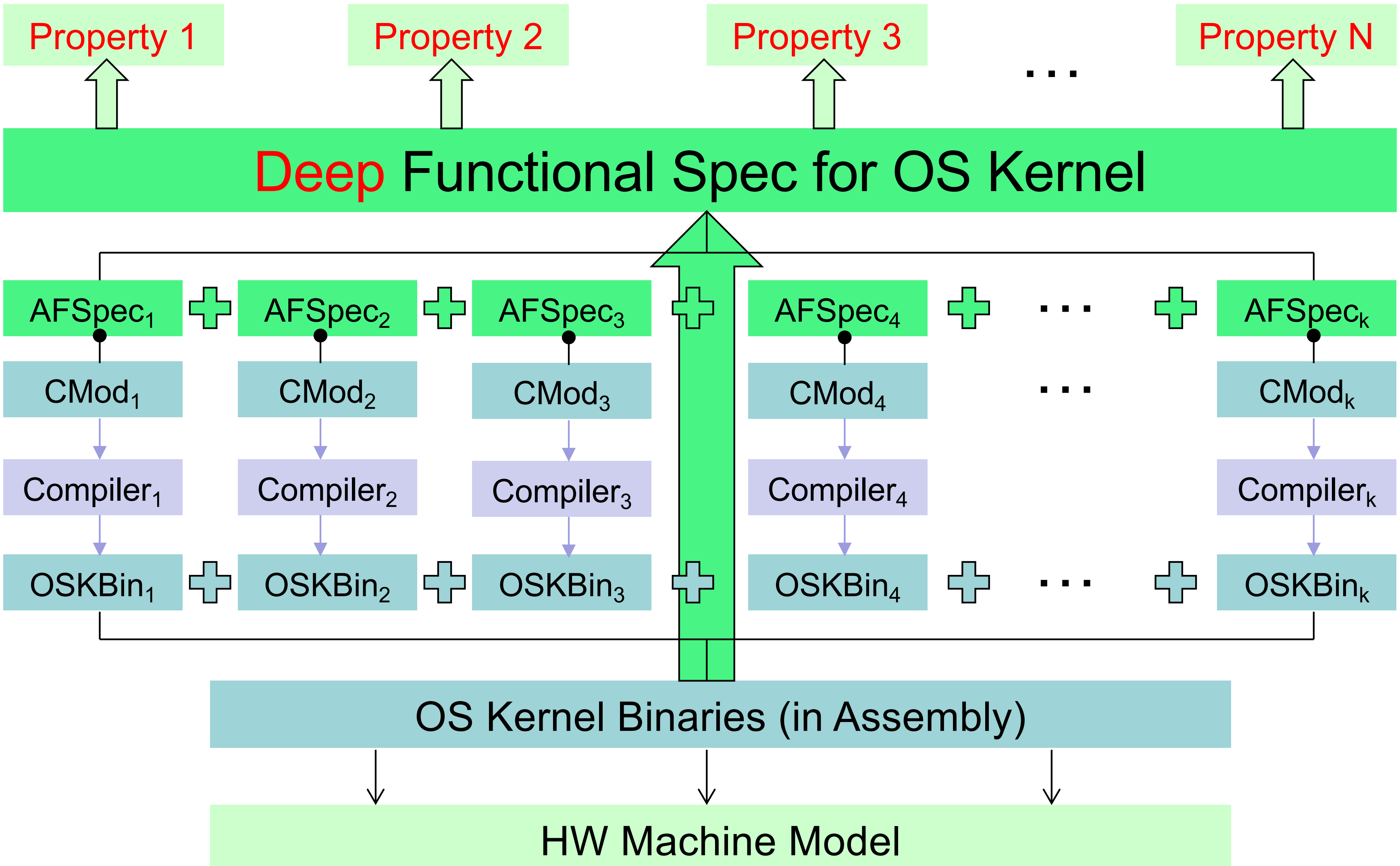
The Conventional Approach



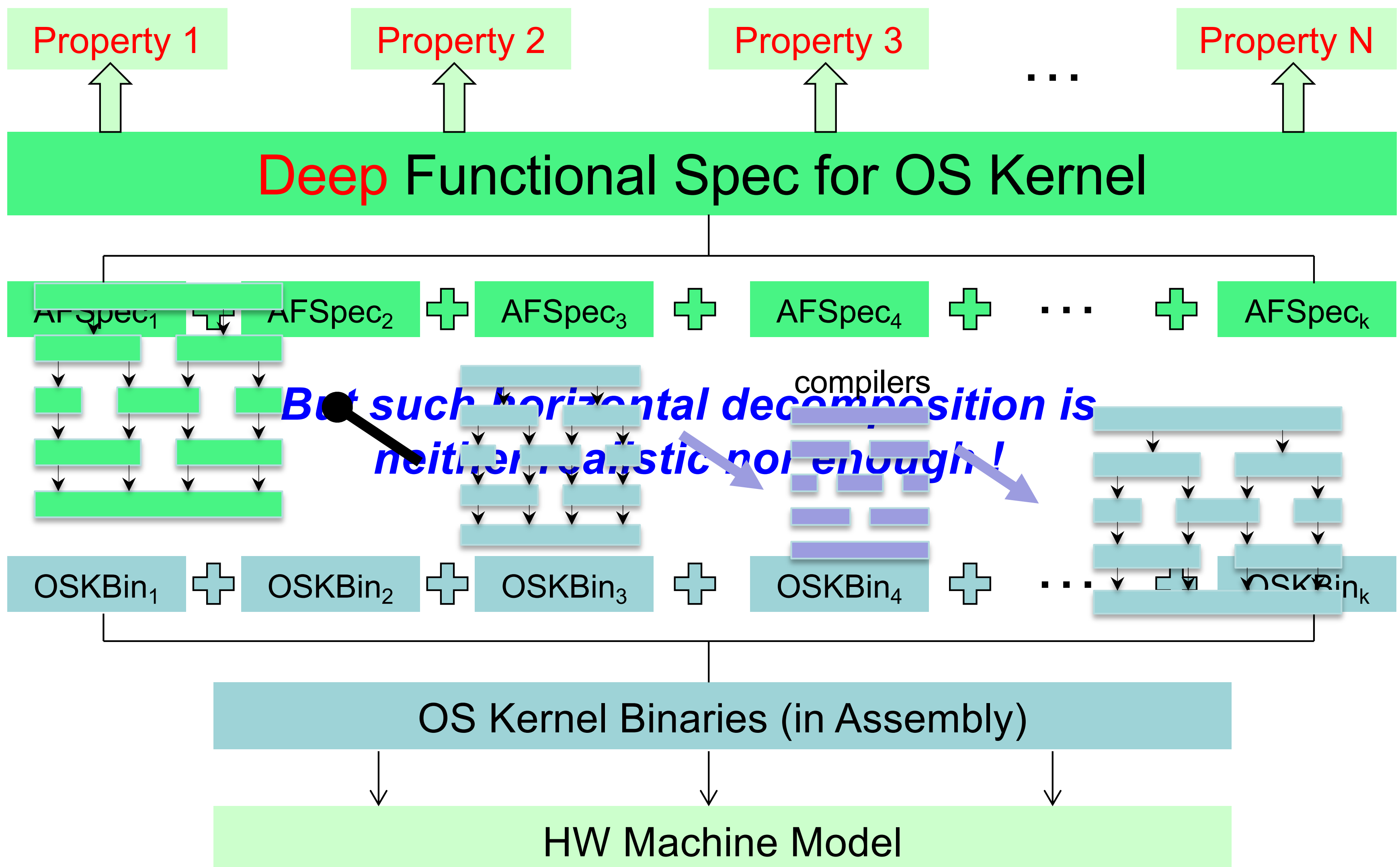
The CertiKOS Approach



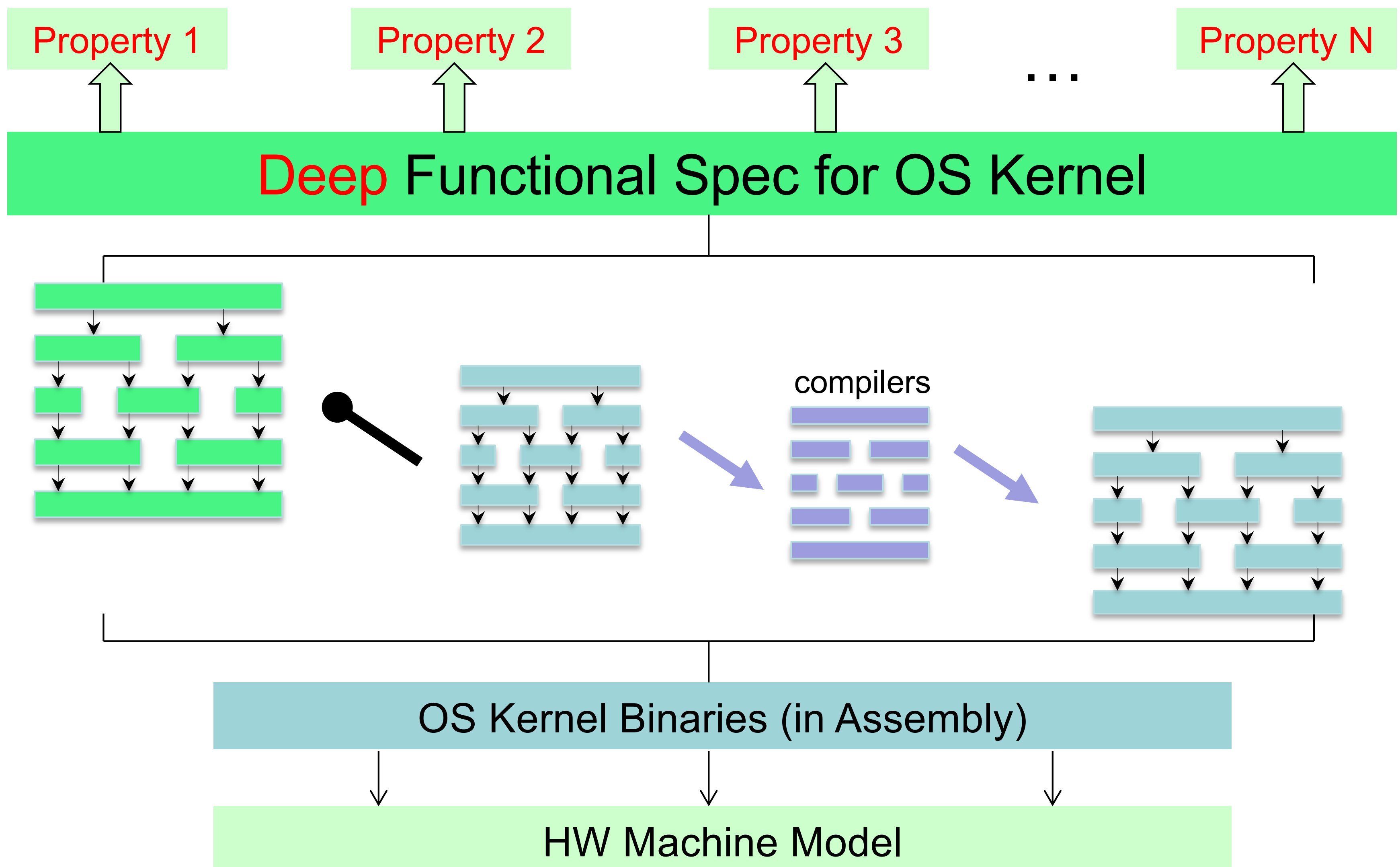
The CertiKOS Approach



The CertiKOS Approach



The CertiKOS Approach



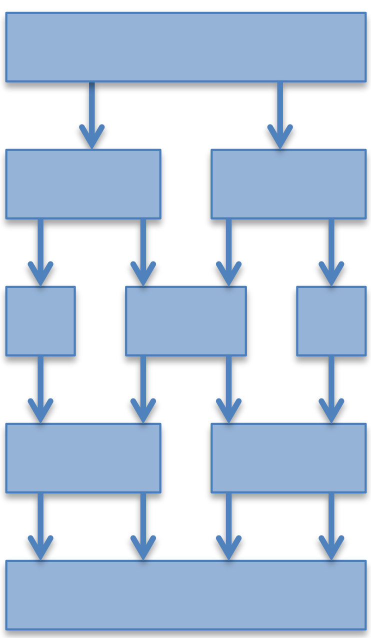
What is a Deep Spec?

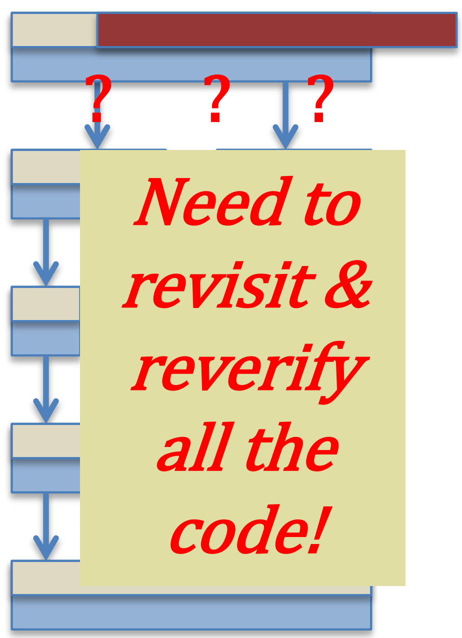
C or Asm module | rich spec A | rich spec B

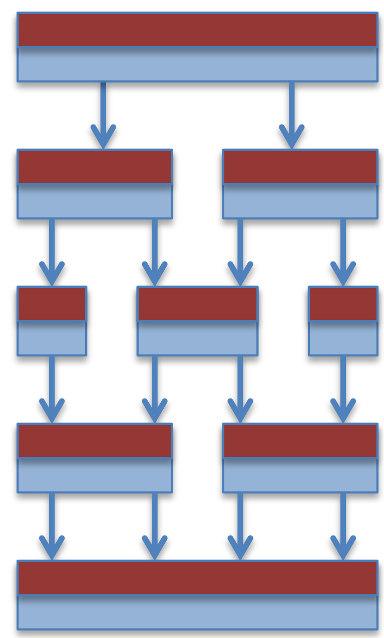
C & Asm Module Implementation

C & Asm Modules w. rich spec A

Want to prove another spec B ?







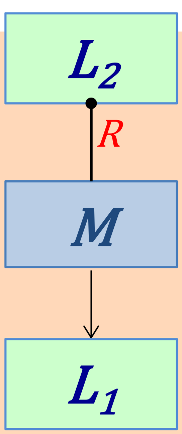
What is a Deep Spec?

$[\![M]\!] L_1 \sim_R L_2$

$[\![M]\!] (L_1)$ and L_2 simulates each other!

L_2 captures everything about running M over L_1

Making it "contextual" using
the whole-program semantics 【•】



L_2 is a **deep specification** of M over L_1
if under any valid program context P of L_2,
【 $P \oplus M$ 】 (L_1) and 【P】(L_2) are
observationally equivalent

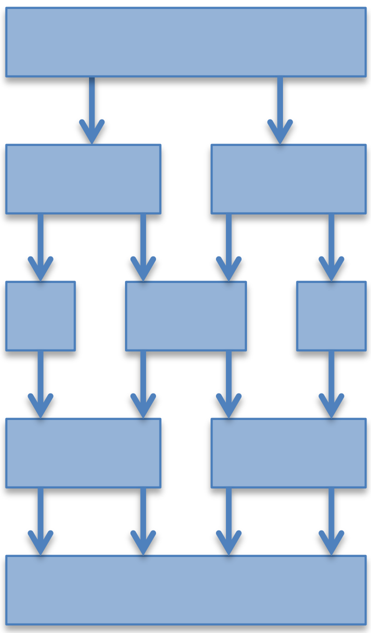
Shallow vs. Deep Specifications

C or Asm module

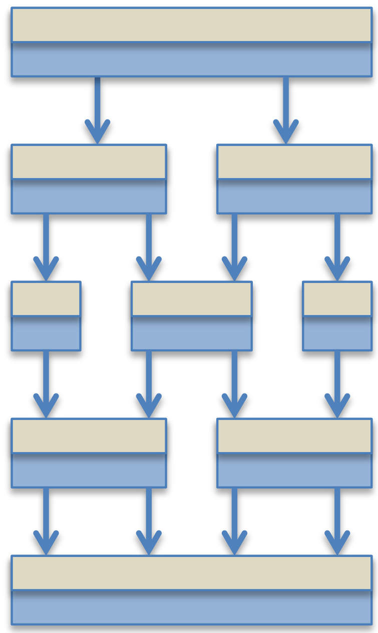
shallow spec

deep spec

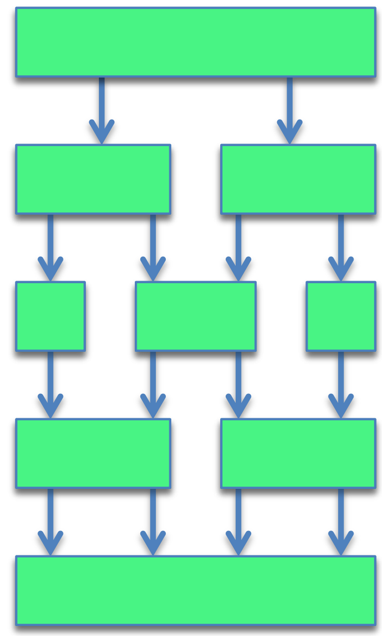
C & Asm Module Implementation



C & Asm Modules w. Shallow Specs



C & Asm Modules w. Deep Specs



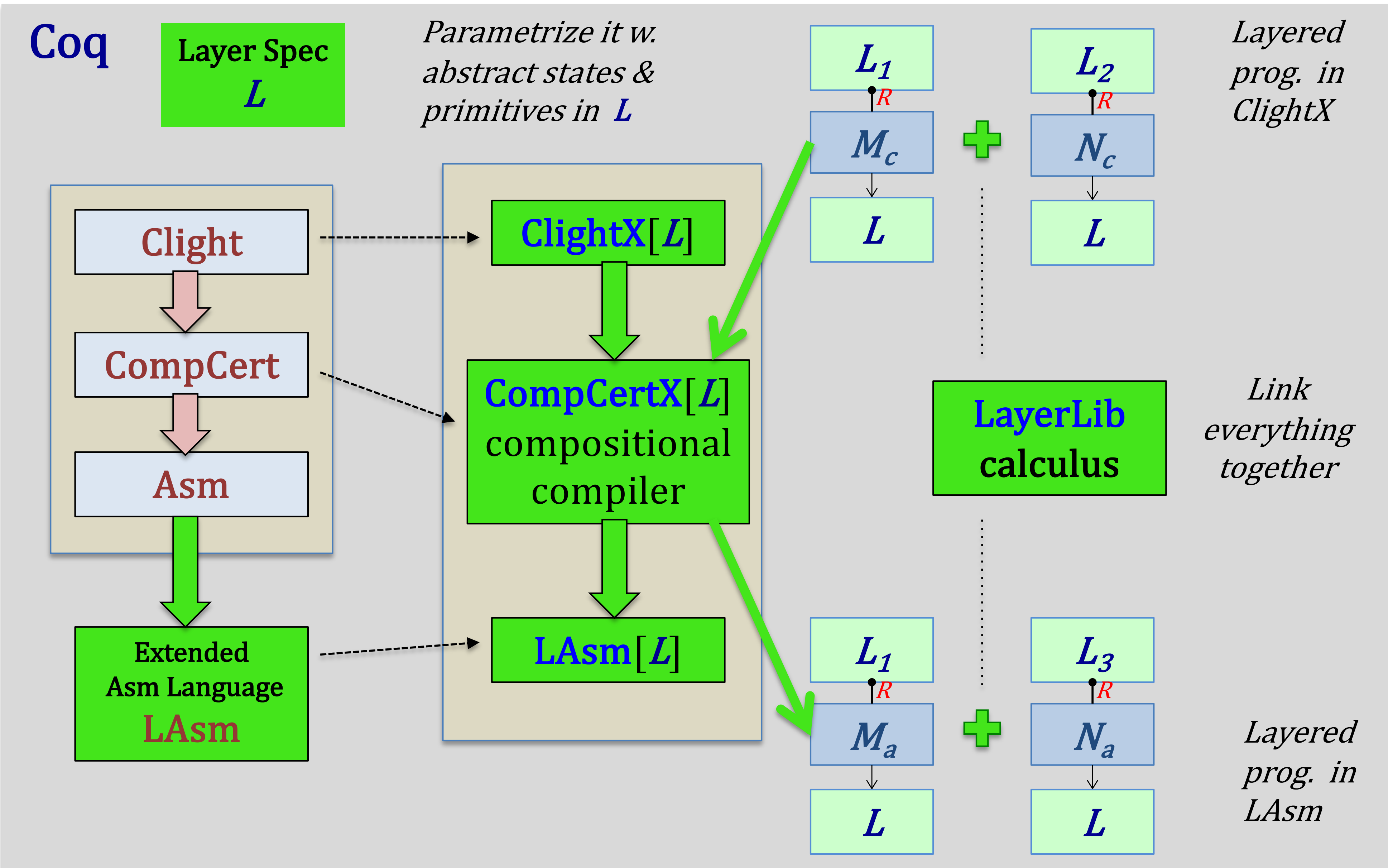
The CertiKOS Approach

- We developed a language-based formalization of certified abstraction layers with deep specifications

- We developed new languages & tools in Coq
  - A formal layer calculus for composing certified layers
  - ClightX for writing certified layers in a C-like language
  - LAsm for writing certified layers in assembly
  - CompCertX that compiles ClightX layers into LAsm layers

- We built multiple certified OS kernels in Coq
  - The initial version has 37 layers and can boot Linux as a guest
  - The later versions support interrupts & multicore concurrency & security (spatial & temporal isolation w. real-time guarantee)

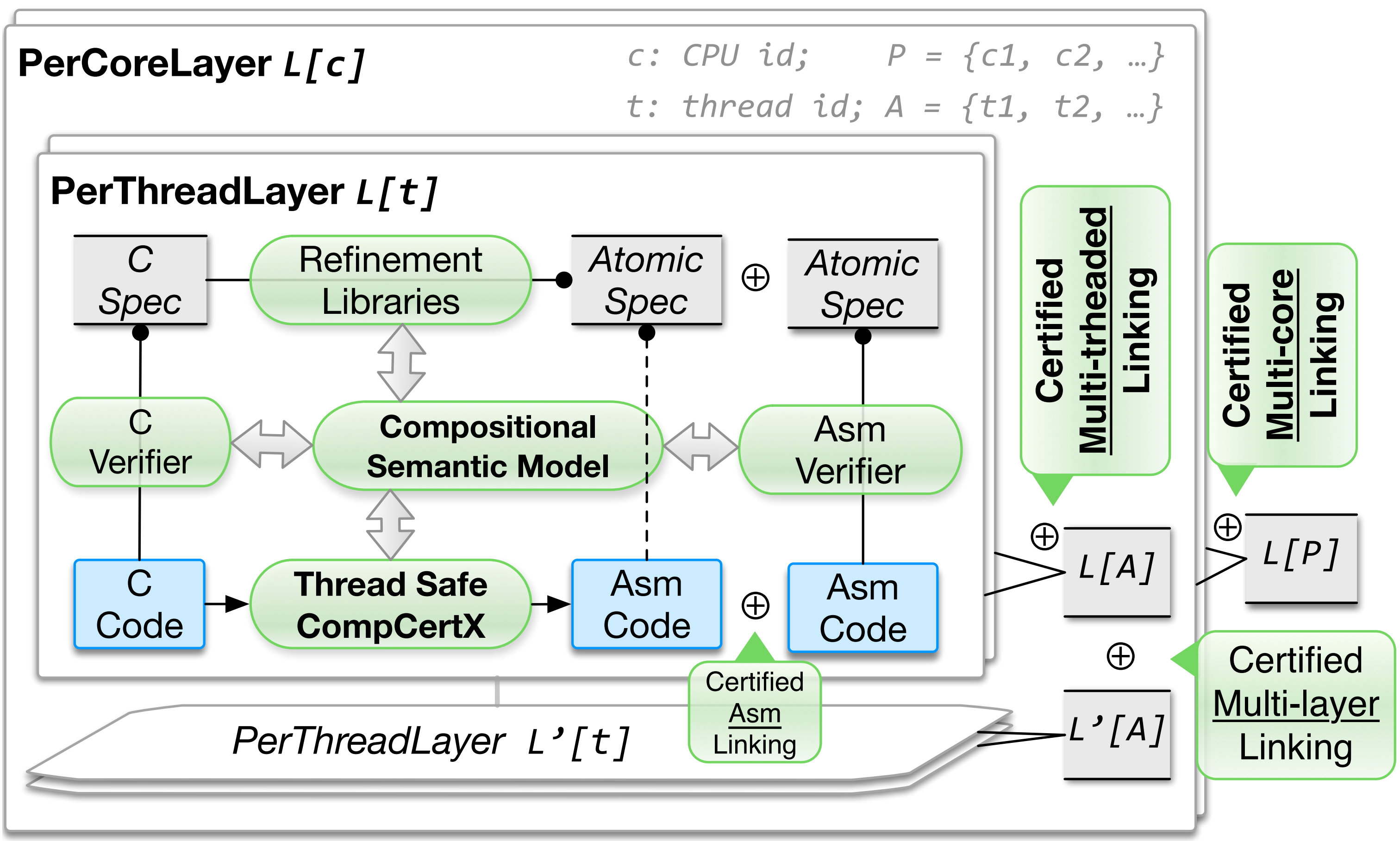
The CertiKOS Toolchain (CAL) [POPL'15]



The CertiKOS Toolchain (CCAL) [PLDI'18]

New programming toolkit w. certified multicore & multithreaded linking:

Composition = parallel composition + hiding (abstraction)



Other CCAL Use Cases

Formal Verification of a Multiprocessor Hypervisor on Arm Relaxed Memory Hardware

FUNCTIONAL REPRODUCED

Design and Verification of the Arm Confidential Compute Architecture

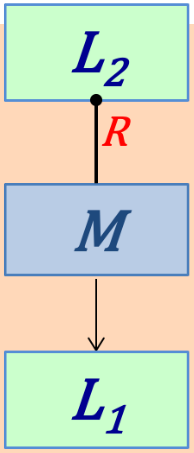
Xupeng Li, *Columbia University*
Xuheng Li, *Columbia University*
Christoffer Dall, *Arm Ltd*
Ronghui Gu, *Columbia University*
Jason Nieh, *Columbia University*
Yousuf Sait, *Arm Ltd*
Gareth Stockwell, *Arm Ltd*

Abstract

The increasing use of sensitive private data in computing is matched by a growing concern regarding data privacy. System software such as hypervisors and operating systems are supposed to protect and isolate applications and their private data, but their large codebases contain many vulnerabilities that can risk data confidentiality and integrity. We introduce Realms, a new abstraction for confidential computing to protect the data confidentiality and integrity of virtual machines. Hardware creates and enforces Realm world, a new physical address space for Realms. Firmware controls the hardware to secure

To address this problem, we introduce the *Arm Confidential Compute Architecture (Arm CCA)*. CCA provides *Realms*, secure execution environments that are completely opaque to privileged, untrusted system software such as OSes and hypervisors. CCA retains the ability of existing system software to manage hardware resources for Realms while preventing it from violating Realm confidentiality and integrity. For example, a hypervisor should retain its ability to dynamically allocate memory to or free memory from a Realm VM, but must never be allowed to access the protected memory contents of a Realm VM. CCA guarantees the confidentiality and integrity of Realm code and data in use, that is data in CPU

Limitation #1: Closed Context



L_2 is a **deep specification** of M over L_1 if under any valid program context P of L_2, 【$P \oplus M$】(L_1) and 【P】(L_2) are observationally equivalent

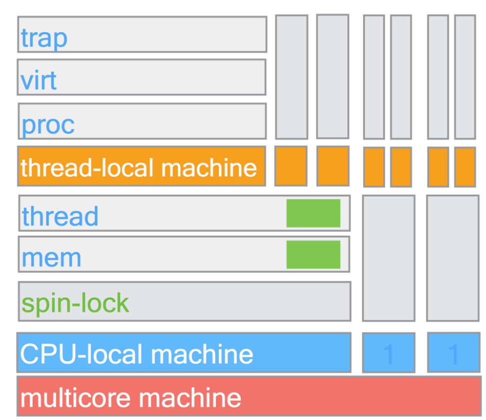
- What should be a valid program context P ?
  - Currently limited by a specific language or programming model
- Should support "open" components & "open" contexts
- New progress: layered & object-based game semantics [POPL'22]; refinement-based game semantics [LICS'20]

Limitation #2: CompCert / CompCertX

- CompCert is not compositional
  - Do not support compiling “open” modules
  - CompCertX is too ad hoc
  - New progress: CompCertO [PLDI’21]
- CompCert does not support finite & multi-threaded stacks
  - New progress: Nominal Stack-Aware CompCert [POPL’22, POPL’19]
- CompCert does not generate verified machine code
  - New progress: CompCertELF [OOPSLA’20]

Limitation #3: Concurrency Model

- CCAL does not support weak-memory models
- CCAL threading model is rather restrictive
  - How to support general user-level & kernel-level thread libraries?
- CCAL support for blocking concurrency is poor
  - How to model yield, sleep, block primitives cleanly?
- CCAL is biased toward atomic objects
- CCAL is still based on ”closed” contexts
- New progress: layered game semantics [POPL’22]



Lessons Learned

- In building certified system software, a closed notion of program contexts is not sufficient
- General concurrency is not compositional, much research is needed to develop truly compositional concurrency models
  - Multicore/multithreaded, interrupts, I/O, system-level concurrency
- Begin with the end in mind
  - What whole-system properties do you want to verify?
  - The HW machine model is evolving rapidly
  - The CCAL+DeepSpec approach decouples the code verification from the verification of specific system properties

Future Plans

- Develop a compositional game semantics for general concurrent components
- Develop the new CCAL based on the new game model
- Develop Nominal CompCertO that can compile concurrent layers into ELF binary layers
- Develop DeepSEA to support certified specification, programming, composition of layered components
- Apply DeepSEA/CCAL to build more powerful certified heterogeneous systems