The CertiKOS Project

Zhong Shao

Yale University

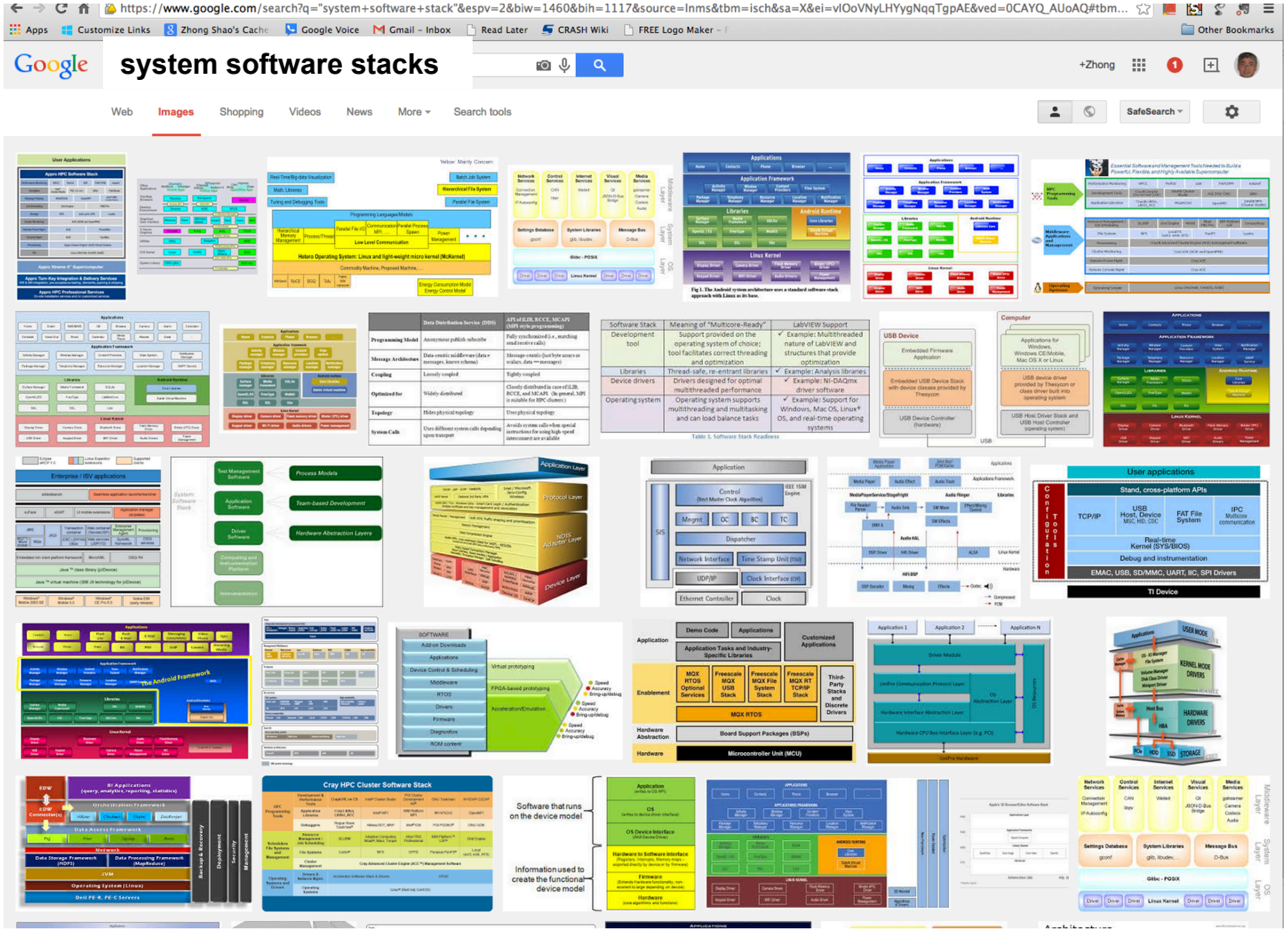
June 6, 2016

http://flint.cs.yale.edu

Acknowledgement: Ronghui Gu, David Costanzo, Jeremie Koenig, Tahina Ramananandro, Newman Wu, Hao Chen, Jieung Kim, Vilhelm Sjoberg, Hernan Vanzetto, Mengqi Liu, Shu-Chun Weng, Quentin Carbonneaux, Zefeng Zeng, Zhencao Zhang, Liang Gu, Jan Hoffmann, Haozhong Zhang, Yu Guo, Joshua Lockerman, and Bryan Ford. This research is supported in part by DARPA CRASH and HACMS programs and NSF SaTC and **Expeditions in Computing** programs.

Motivation

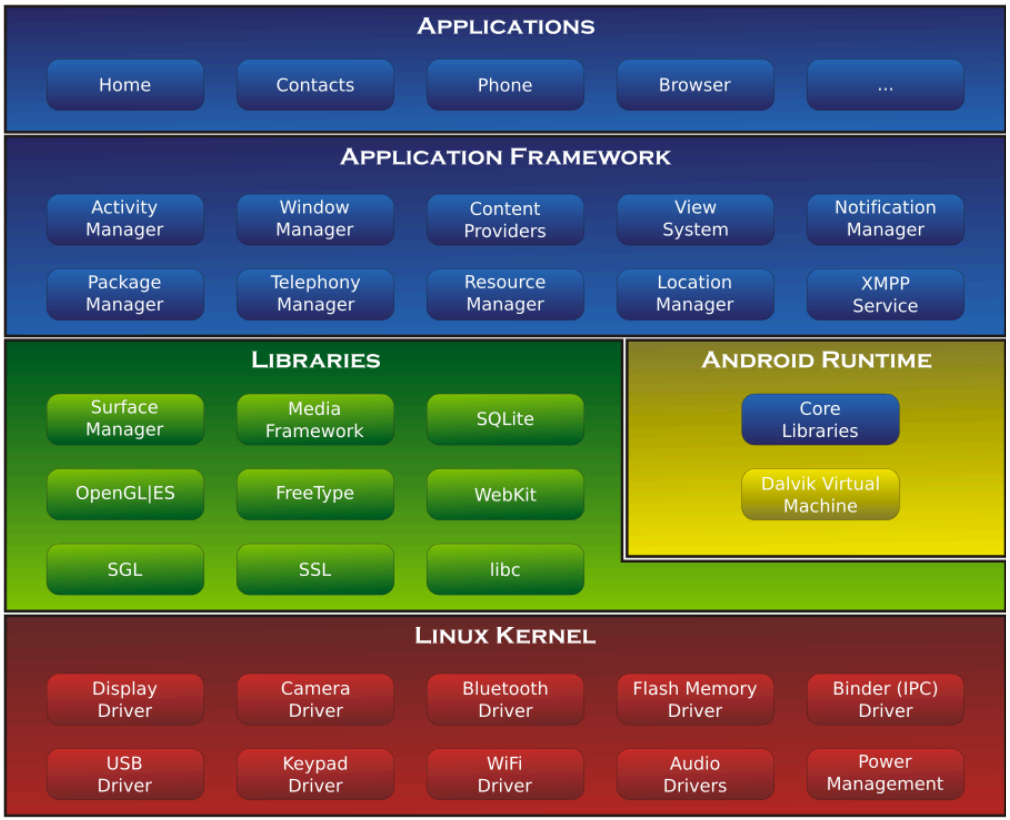
*How to build reliable & secure **system software stacks**?*

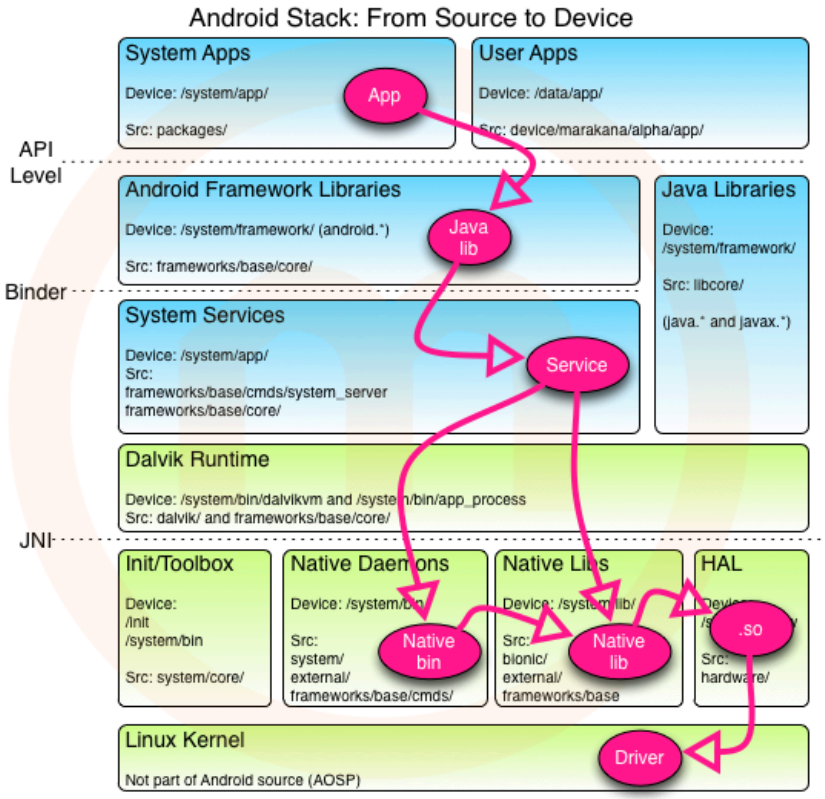


Motivation

Android architecture & system stack

From https://thenewcircle.com/s/post/1031/android_stack_source_to_device &
http://en.wikipedia.org/wiki/Android_(operating_system)

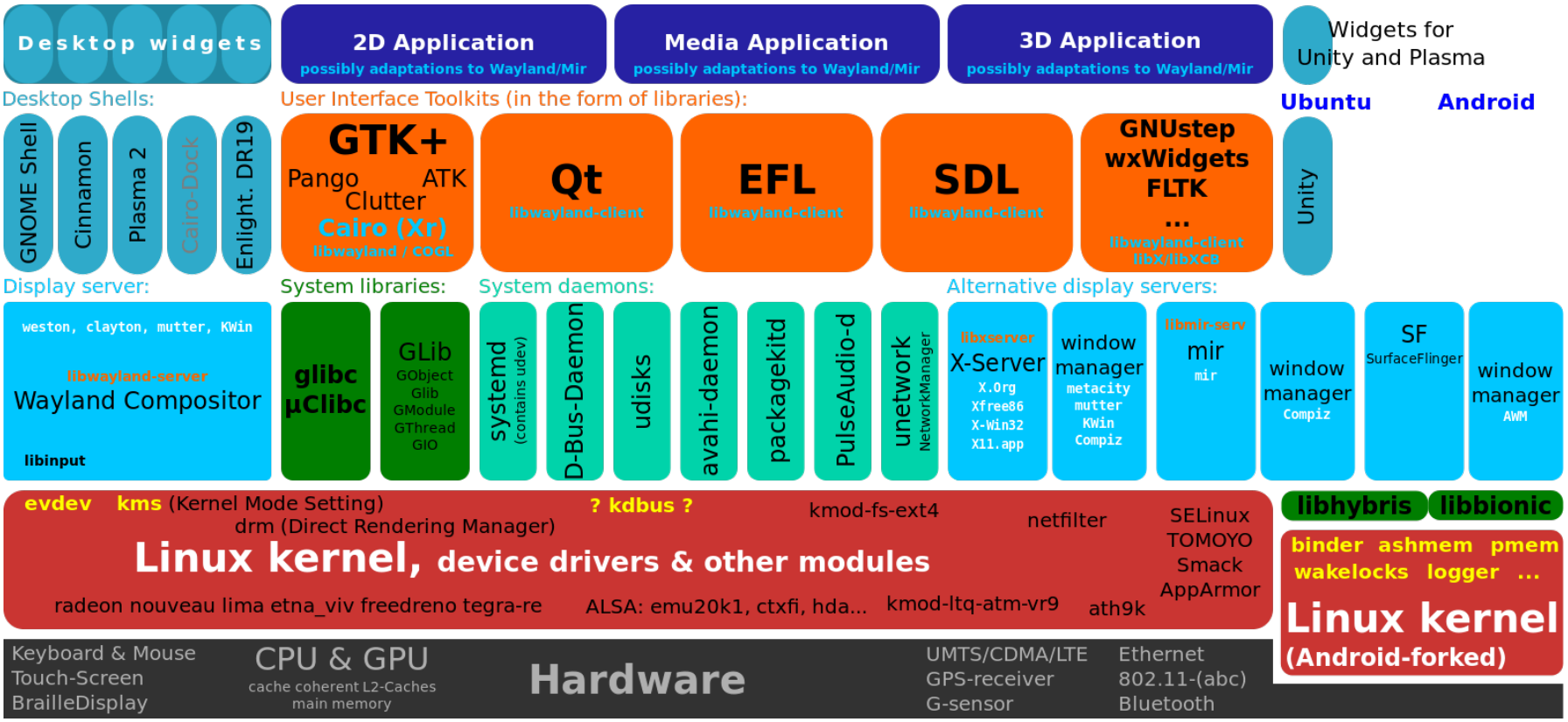




Motivation

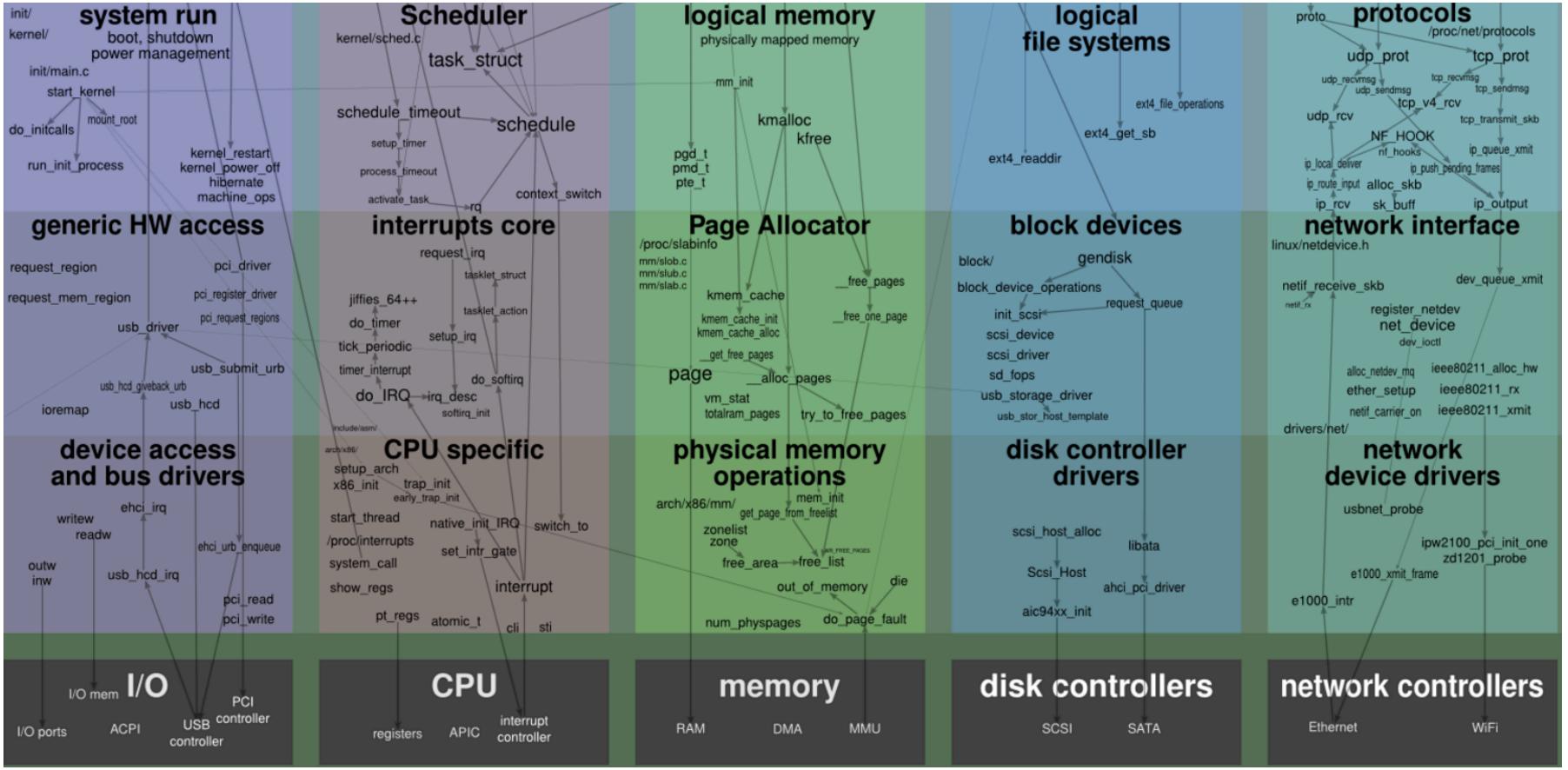
Visible software components of the Linux desktop stack

From http://en.wikipedia.org/wiki/Linux



Motivation

Linux Kernel Map: kernel components are sorted into different stacks of abstraction layers based on their underlying HW device.



Motivation

Software stack for HPC clusters

From http://www.hpcwire.com/2014/02/24/comprehensive-flexible-software-stack-hpc-clusters/

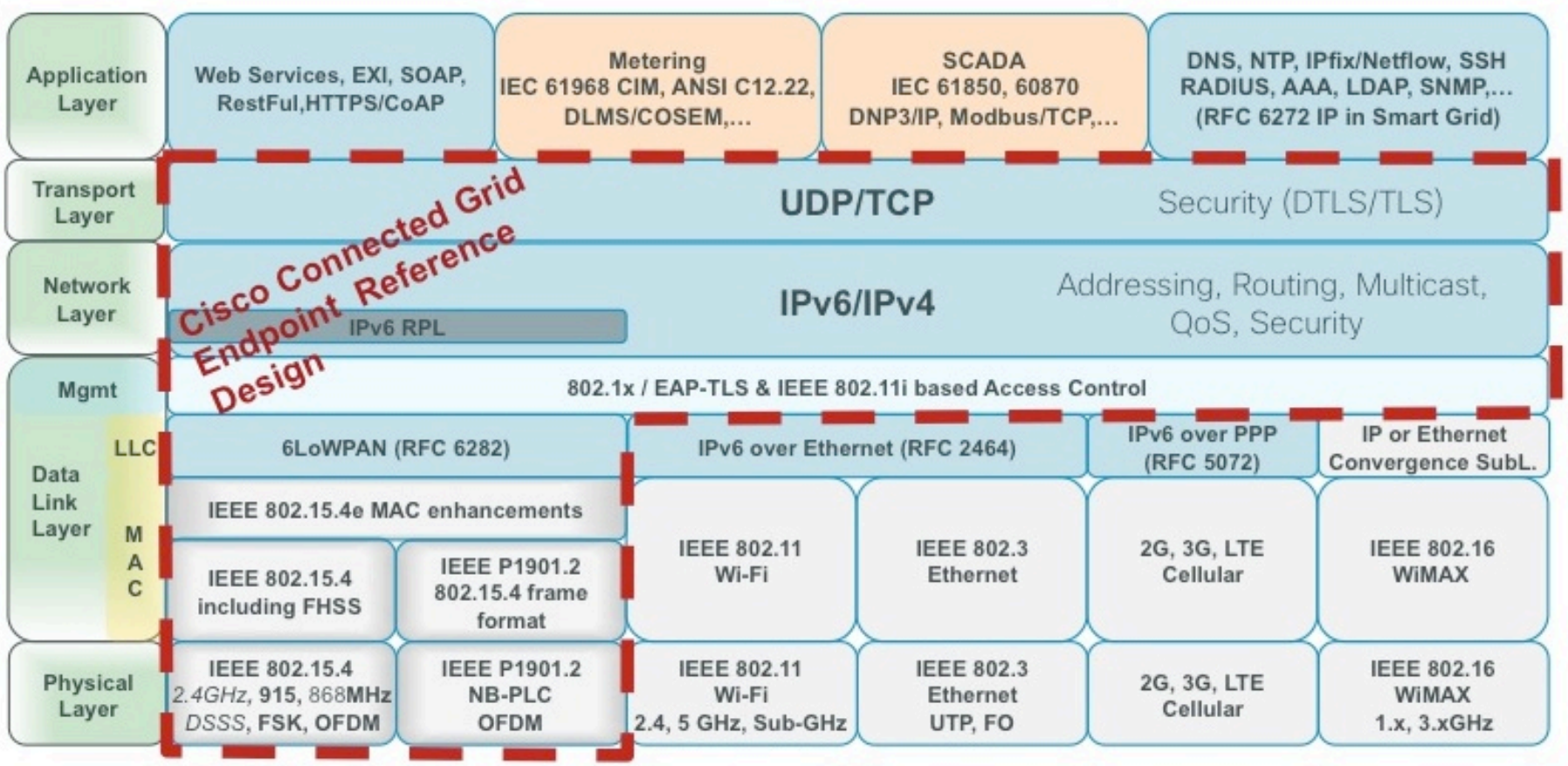
Essential Software and Management Tools Needed to Build a Powerful, Flexible, and Highly Available Supercomputer.

HPC Programming Tools	Performance Monitoring	HPCC	Perfctr	IOR	PAPI/IPM	netperf
	Development Tools	Cray® Compiler Environment (CCE)	Intel® Cluster Studio		PGI (PGI CDK)	GNU
	Application Libraries	Cray® LibSci, LibSci_ACC	MVAPICH2		OpenMPI	Intel® MPI- (Cluster Studio)
Middleware Applications and Management	Resource Management / Job Scheduling	SLURM	Grid Engine	MOAB	Altair PBS Pro	IBM Platform LSF	Torque/Maui
	File System	NFS	Local FS (ext3, ext4, XFS)		PanFS		Lustre
	Provisioning	Cray® Advanced Cluster Engine (ACE) management software					
	Cluster Monitoring	Cray ACE (iSCB and OpenIPMI)					
	Remote Power Mgmt	Cray ACE					
	Remote Console Mgmt	Cray ACE					
Operating Systems	Operating System	Linux (Red Hat, CentOS, SUSE)					

Motivation

Cisco's FAN (Field-Area-Network) protocol layering

From https://solutionpartner.cisco.com/web/cegd/overview



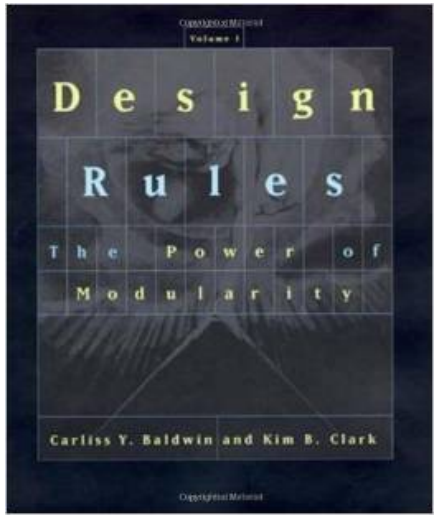
Motivation (cont'd)

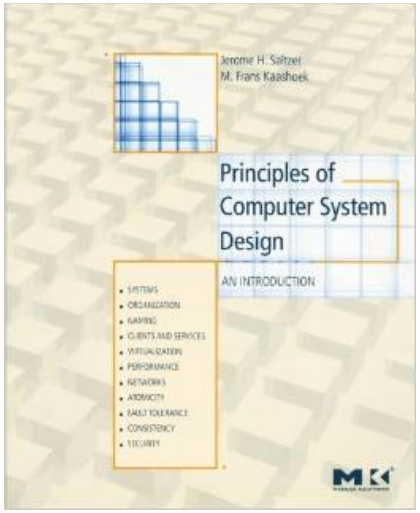
- Common themes: all system stacks are built based on abstraction, modularity, and layering

- Abstraction layers are ubiquitous!

Such use of abstraction, modularity, and layering is "**the key factor that drove the computer industry toward today's explosive levels of innovation and growth** because *complex products can be built from smaller subsystems that can be designed independently yet function together as a whole.*"

Baldwin & Clark " Design Rules: Volume 1, The Power of Modularity", MIT Press, 2000





Do We Understand Abstraction?

In the PL community:

(abstraction **in the small)**

- Mostly formal but tailored within a single programming language (ADT, objects, existential types)
- Specification only describes type or simple pre- & post condition
- Hide concrete data representation (we get the nice *repr. independence* property)
- Well-formed *typing* or *Hoare-style judgment* between the impl. & the spec.

In the System world:

(abstraction **in the large)**

- Mostly informal & language-neutral (APIs, sys call libraries)
- Specification describes full functionality (but in English)
- Implementation is a black box (*in theory*); an *abstraction layer* hides all things below
- *The "implements" relation* between the impl. & the spec

Problems

- What is an ***abstraction layer***?
- How to formally ***specify*** an abstraction layer?
- How to ***program***, ***verify***, and ***compile*** each layer?
- How to ***compose*** abstraction layers?
- How to apply ***certified abstraction layers*** to build *reliable* and *secure* system software?

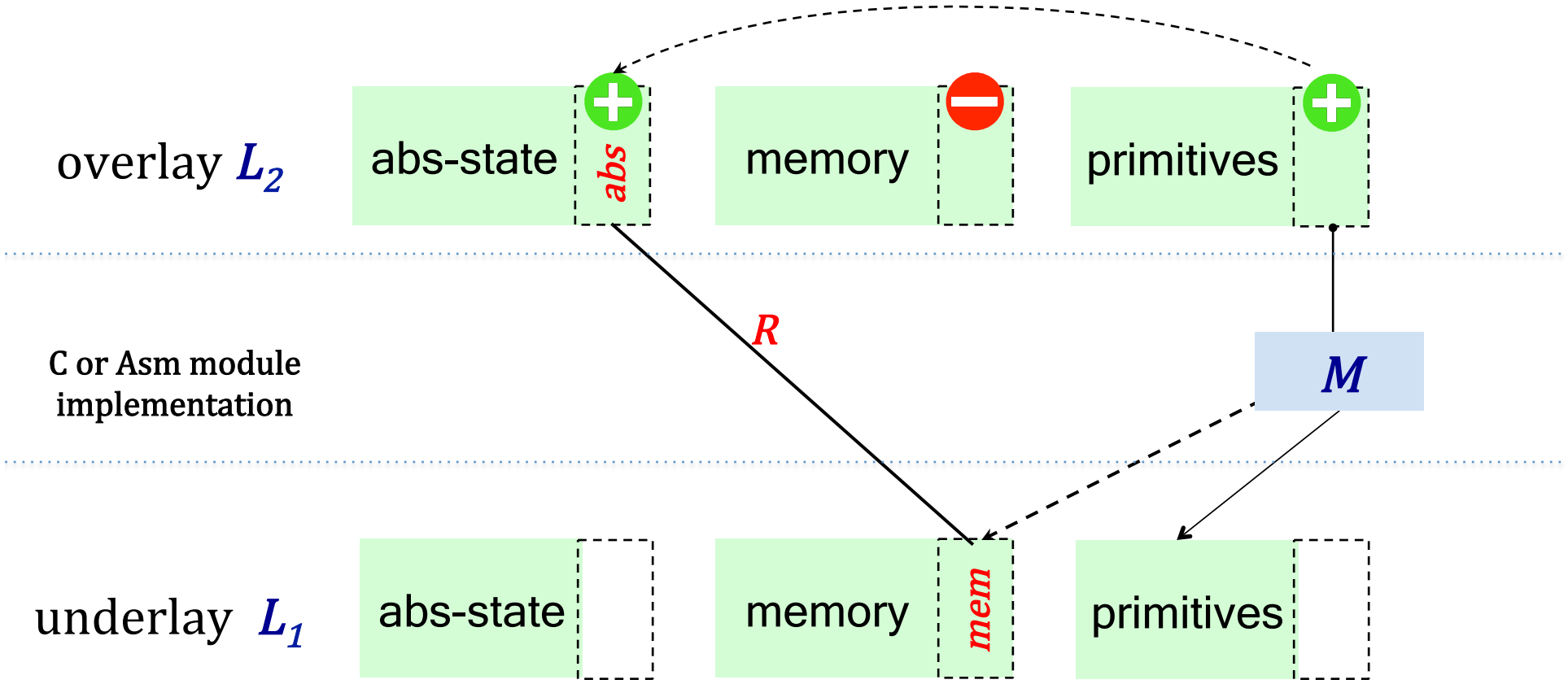
Our Contributions

- We define deep specification and present a language-based formalization of certified abstraction layer

- 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
  - Latest versions support interrupts and multicore concurrency

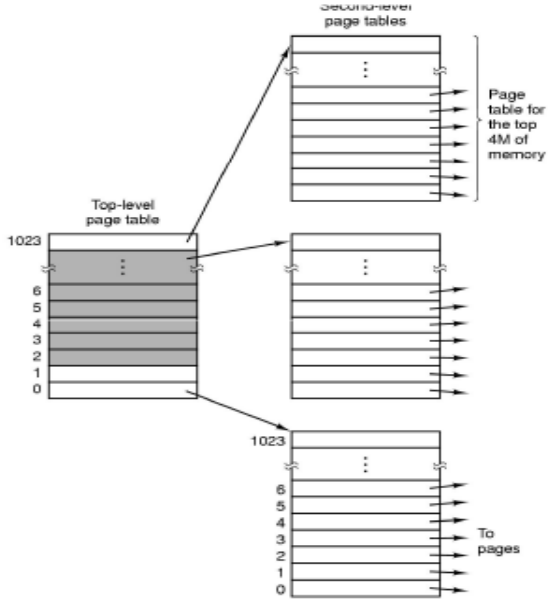
What is an Abstraction Layer?



Example: Page Tables

concrete C types

```
struct PMap {
    char * page_dir[1024];
    uint  page_table[1024][1024];
};
```



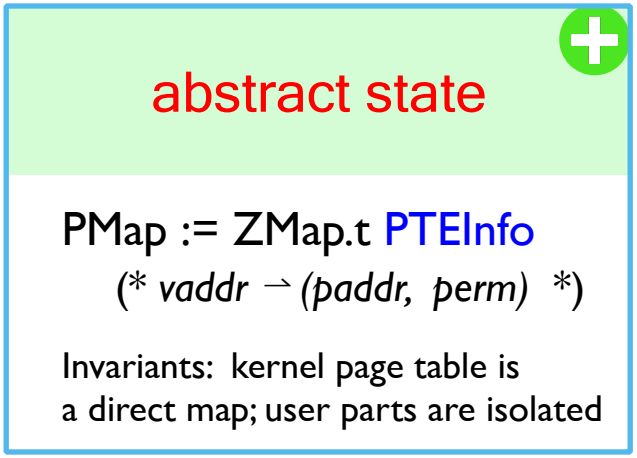
abstract Coq spec

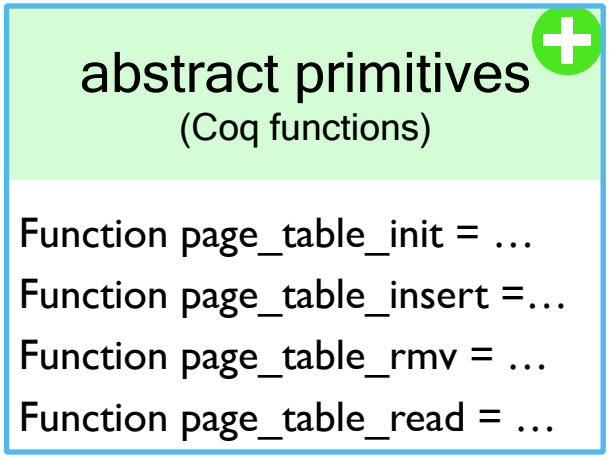
```
Inductive PTPerm:Type :=
    | PTP
    | PTU
    | PTK.

Inductive PTEInfo:=
    | PTEValid (v : Z) (p :  PTPerm)
    | PTEUnPresent.

Definition PMap := ZMap.t PTEInfo.
```

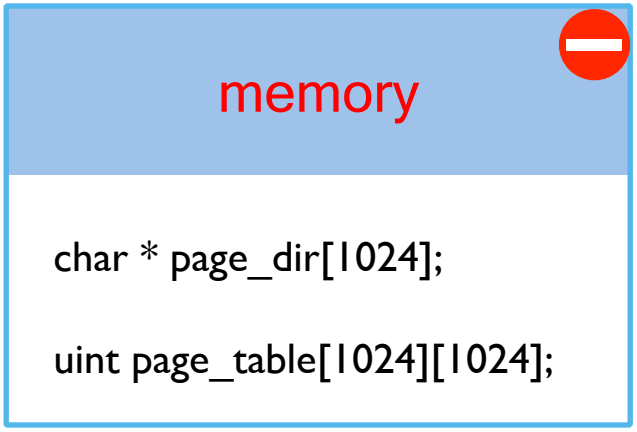
Example: Page Tables

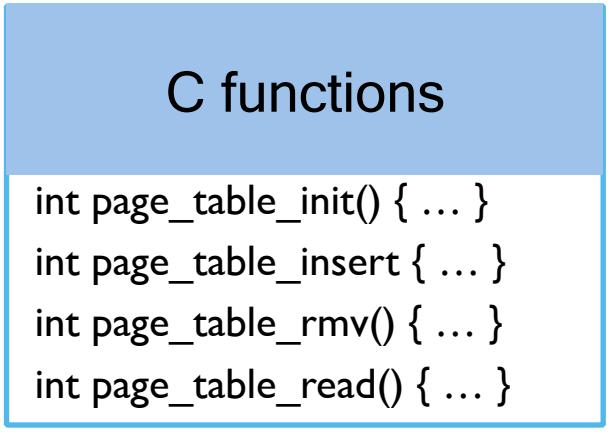
abstract layer spec

concrete C implementation



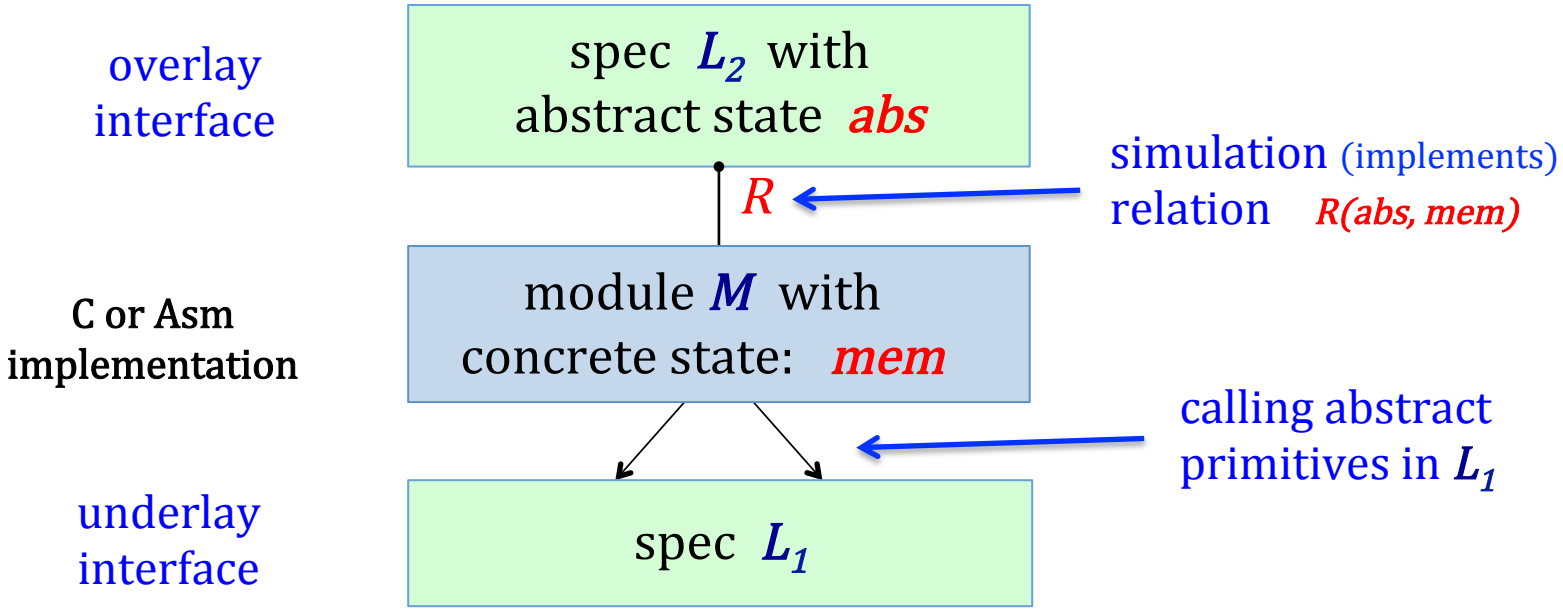






Formalizing Abstraction Layers

What is a ***certified*** abstraction layer (L_1, M, L_2) ?



Recorded as the *well-formed layer* judgment

$$L_1 \vdash_R M : L_2$$

Layer Interface vs. Deep Spec?

RICH	***What is the precise definition of rich? How rich should it be?***	
FORMAL	in notation with a clear semantics	✓
LIVE	machine-checked connection to implementations	✓
2-SIDED	connected to both implementations & clients	✓

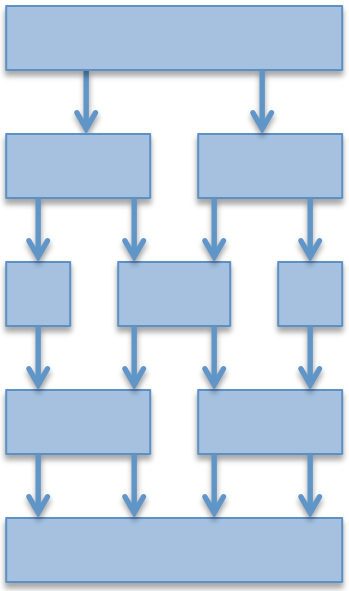
Problem w. “Rich” Specs


C or Asm module

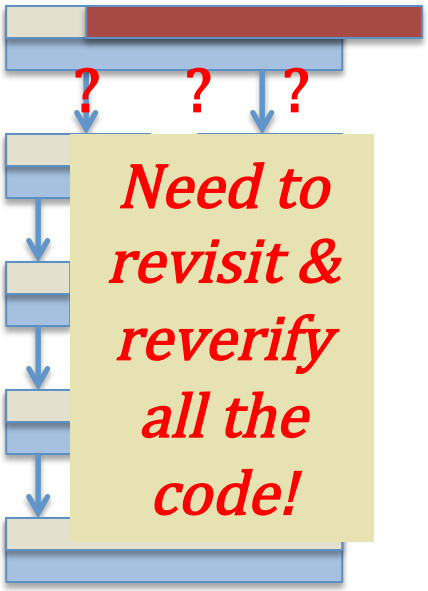

rich spec A


rich spec B

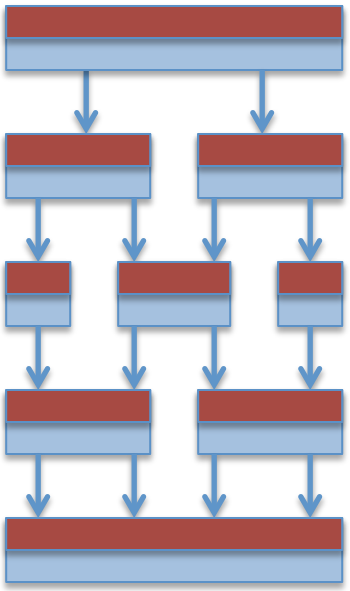
C & Asm Module Implementation



C & Asm Modules w. rich spec A



Want to prove another spec B ?

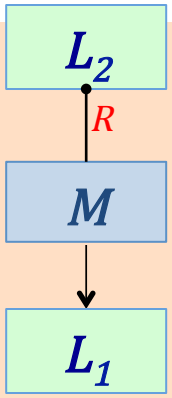


The Science of Deep Specs?

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

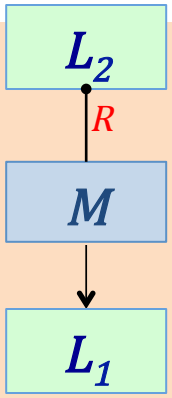
L_2 captures everything about running M over L_1

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

Why Deep Spec is Really Cool?



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

Deep spec L captures all we need to know about a layer M

- No need to ever look at M again!
- Any property about M can be proved using L alone.
- Provide direct support to concurrency

Impl. Independence : any two implementations of the same deep spec are *contextually equivalent*

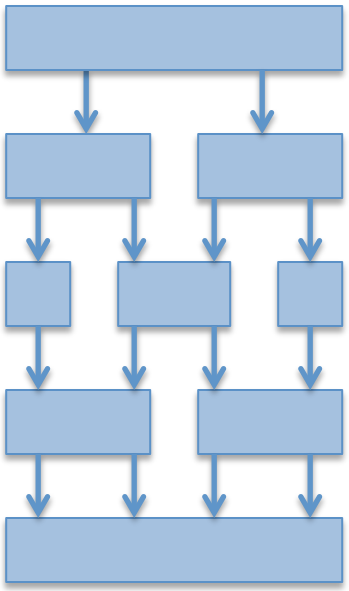
Shallow vs. Deep Specifications

C or Asm module

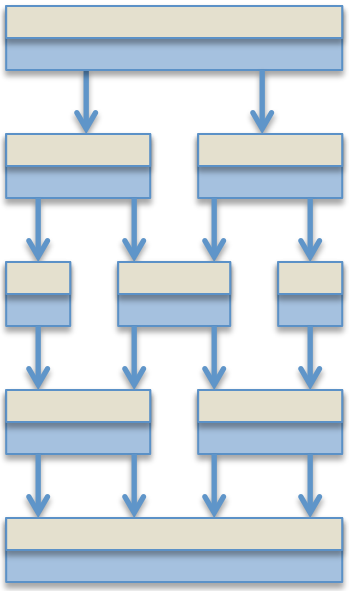
shallow spec

deep spec

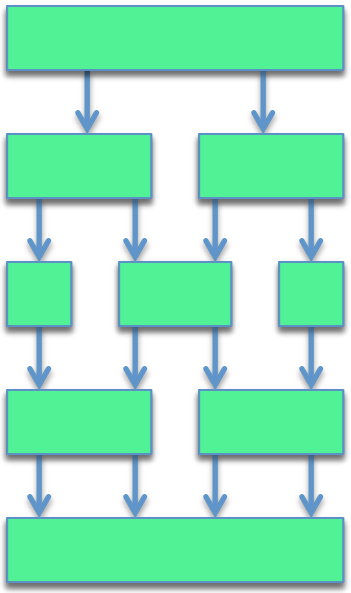
C & Asm Module Implementation



C & Asm Modules w. Shallow Specs



C & Asm Modules w. Deep Specs



How to Make Deep Spec Work?

No languages/tools today support deep spec & certified layered programming

Challenges:

- Implementation done in C or assembly or …
- Specification done in richer logic (e.g., Coq)
- Need to mix both and also simulation proofs
- Need to compile C layers into assembly layers
- Need to compose different layers

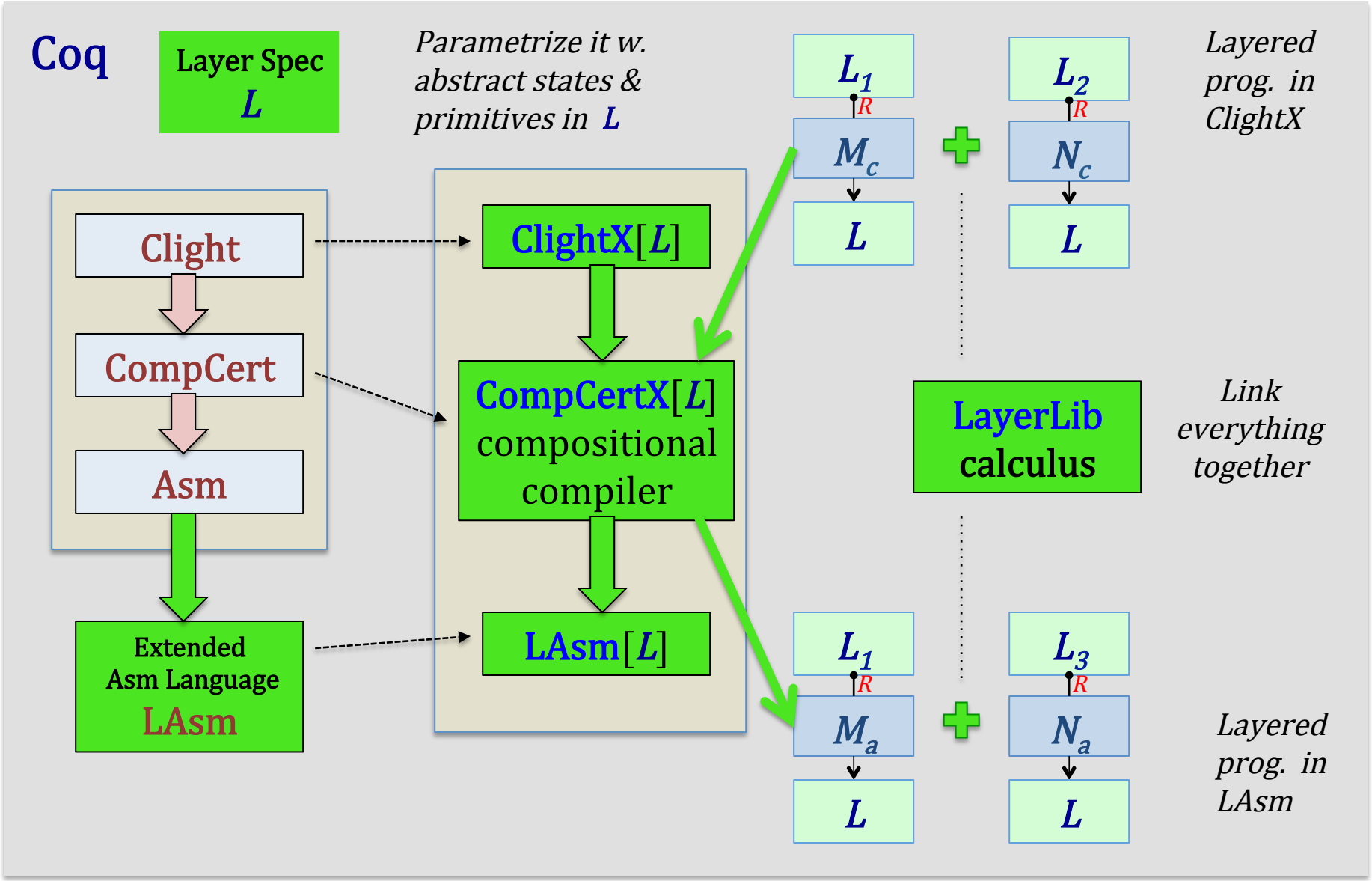
Our Contributions

- We define deep specification and present a language-based formalization of certified abstraction layer

- 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
  - Latest versions support interrupts and multicore concurrency

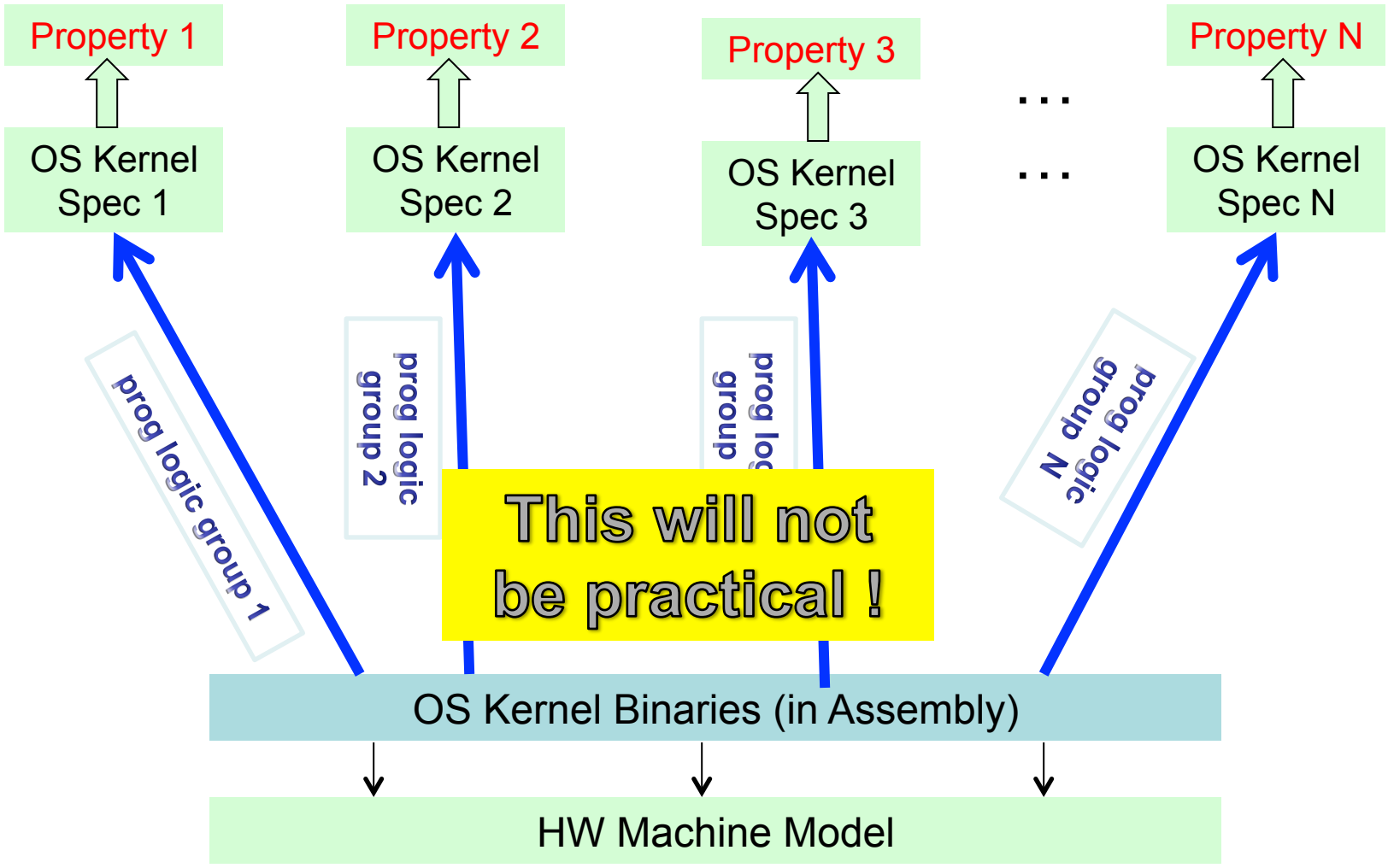
What We Have Done [POPL'15]



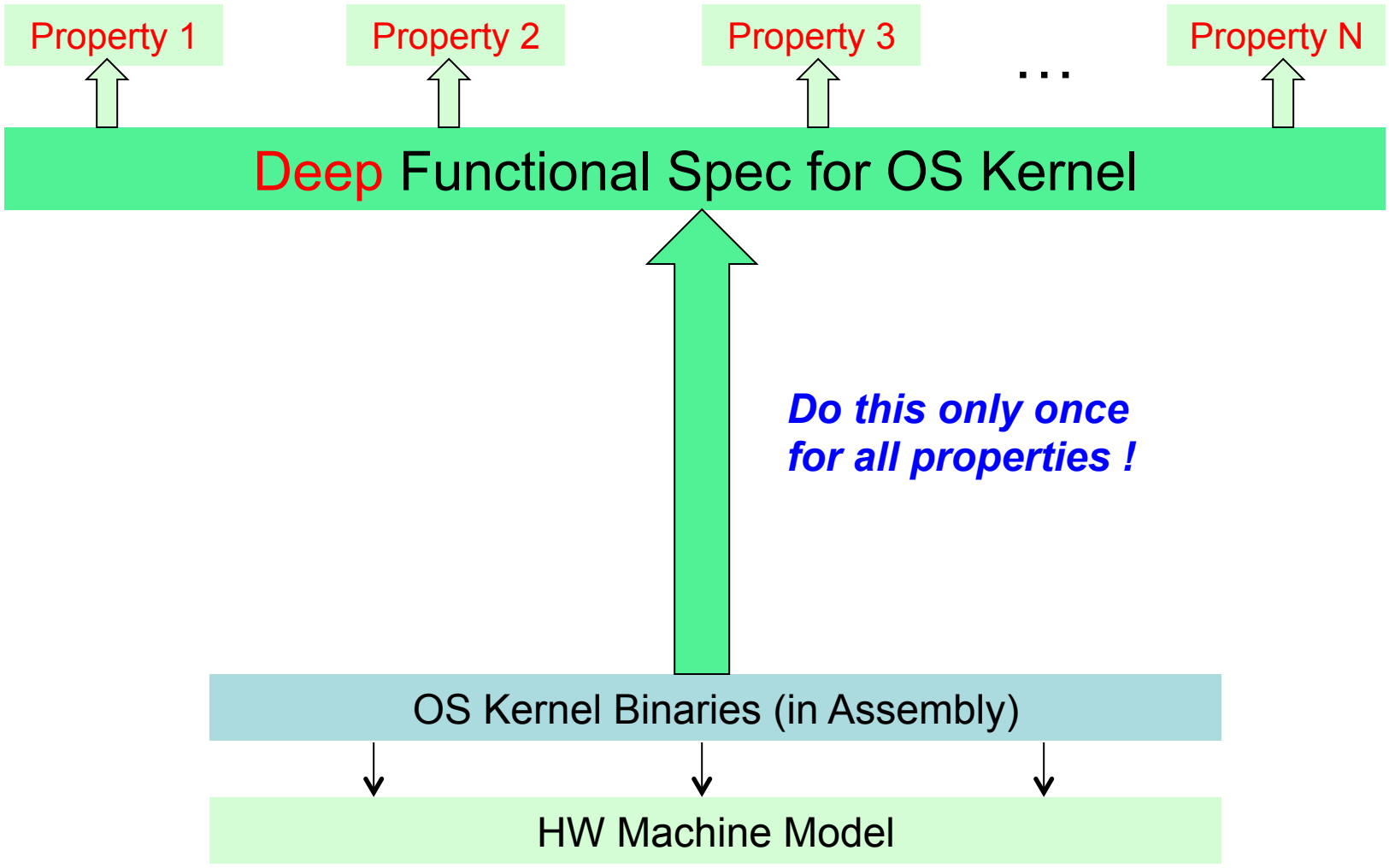
Our Contributions

- We define deep specification and present a language-based formalization of certified abstraction layer
- 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
  - Latest versions support interrupts and multicore concurrency

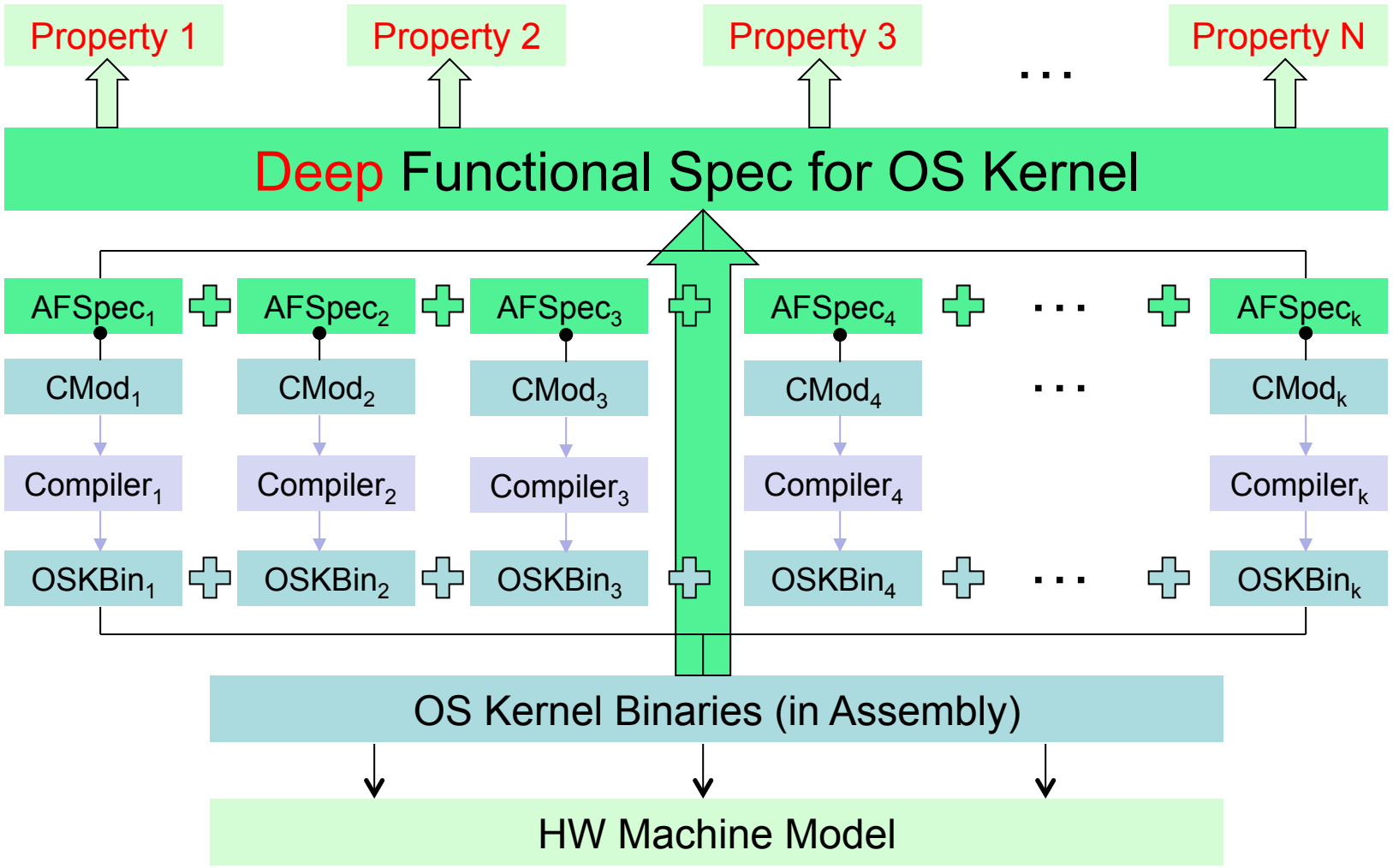
The Conventional Approach



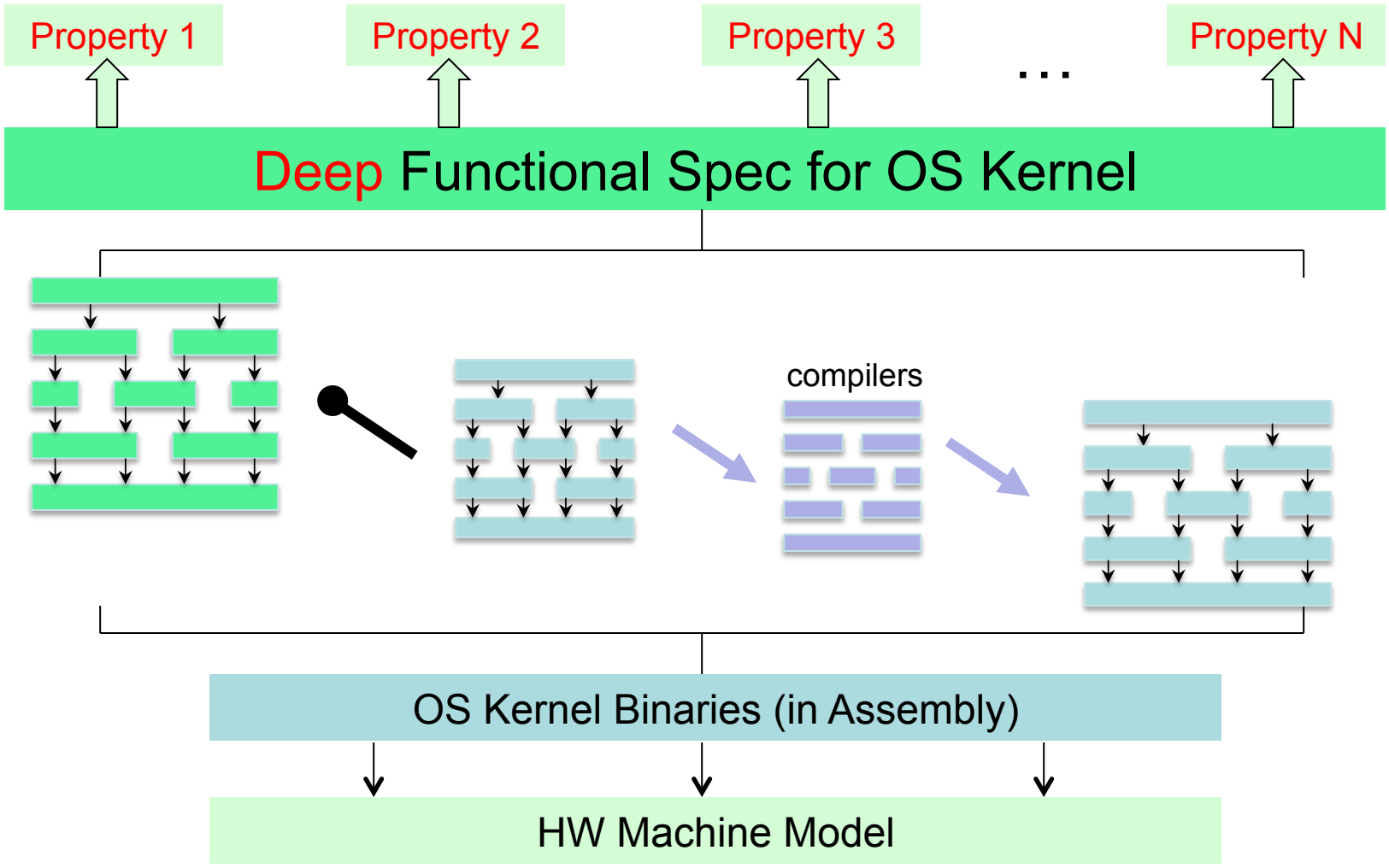
The DeepSpec Approach



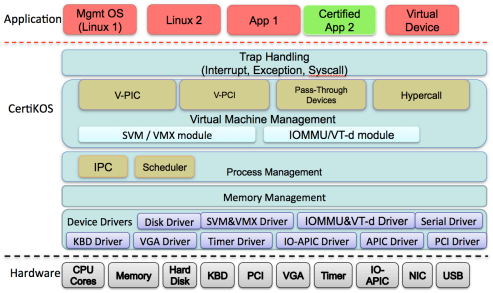
The DeepSpec Approach



The DeepSpec Approach

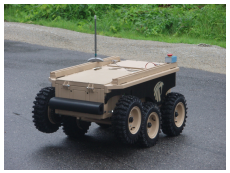


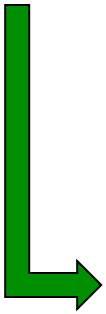
Case Study: mCertiKOS

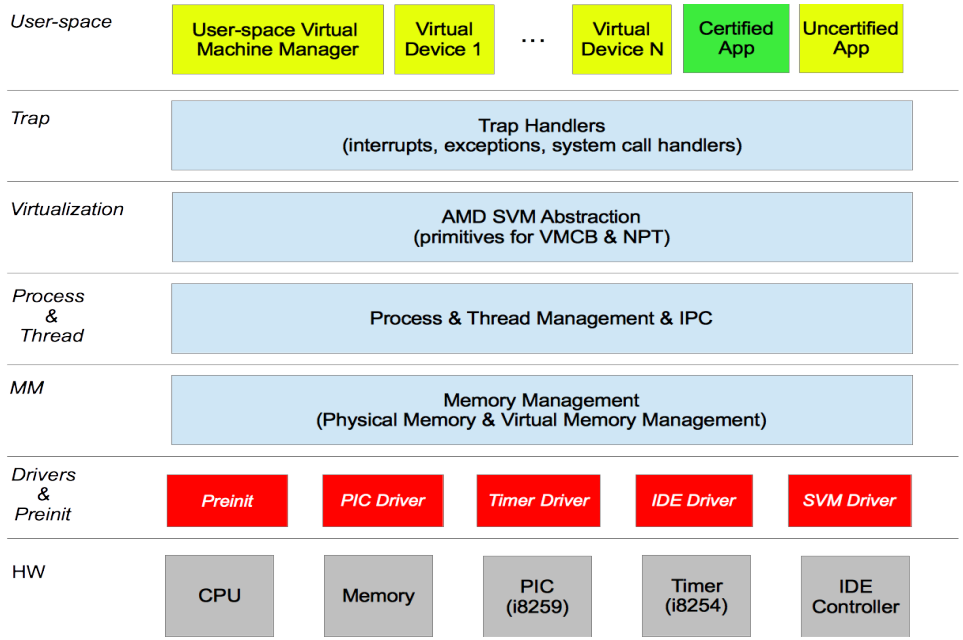


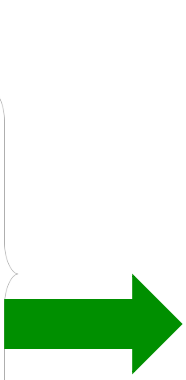
Single-core version of CertiKOS (developed under DARPA CRASH & HACMS programs), 3 kloc, can boot Linux

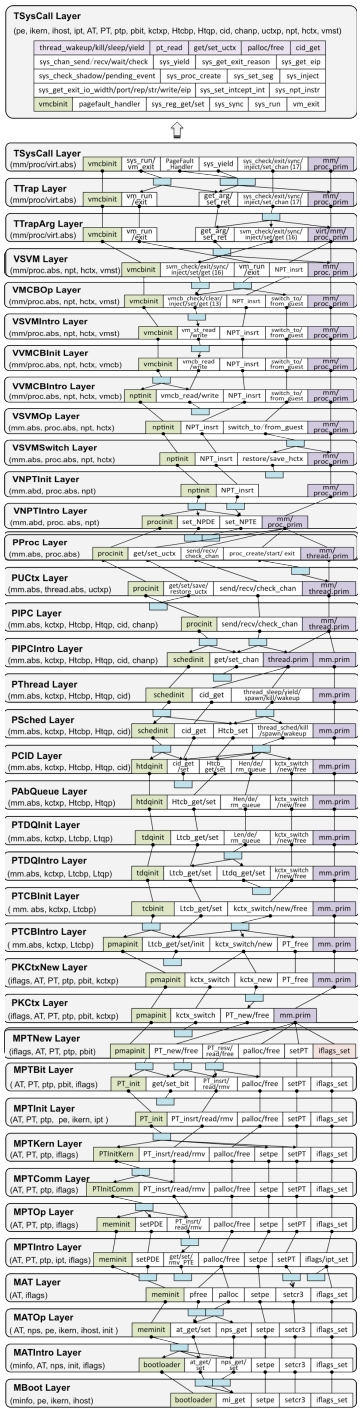
Aggressive use of abstraction over deep specs (37 layers in ClightX & LAsm)



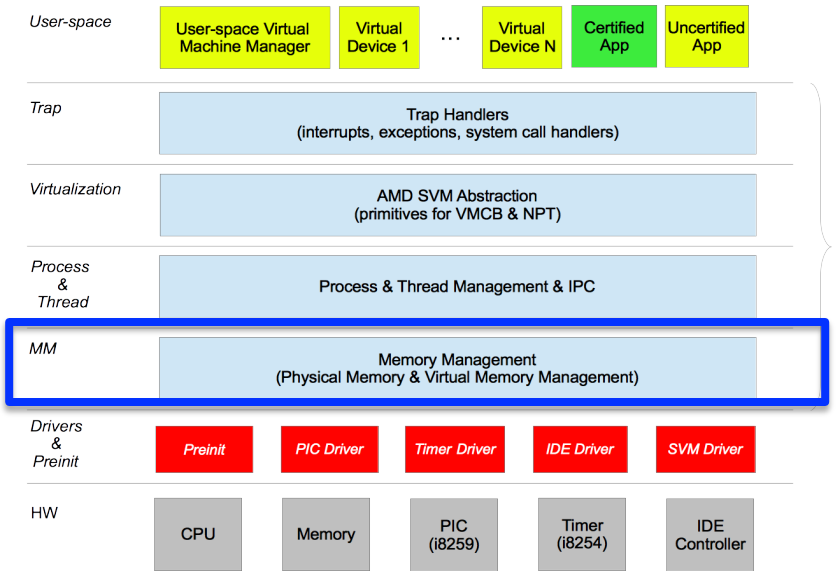


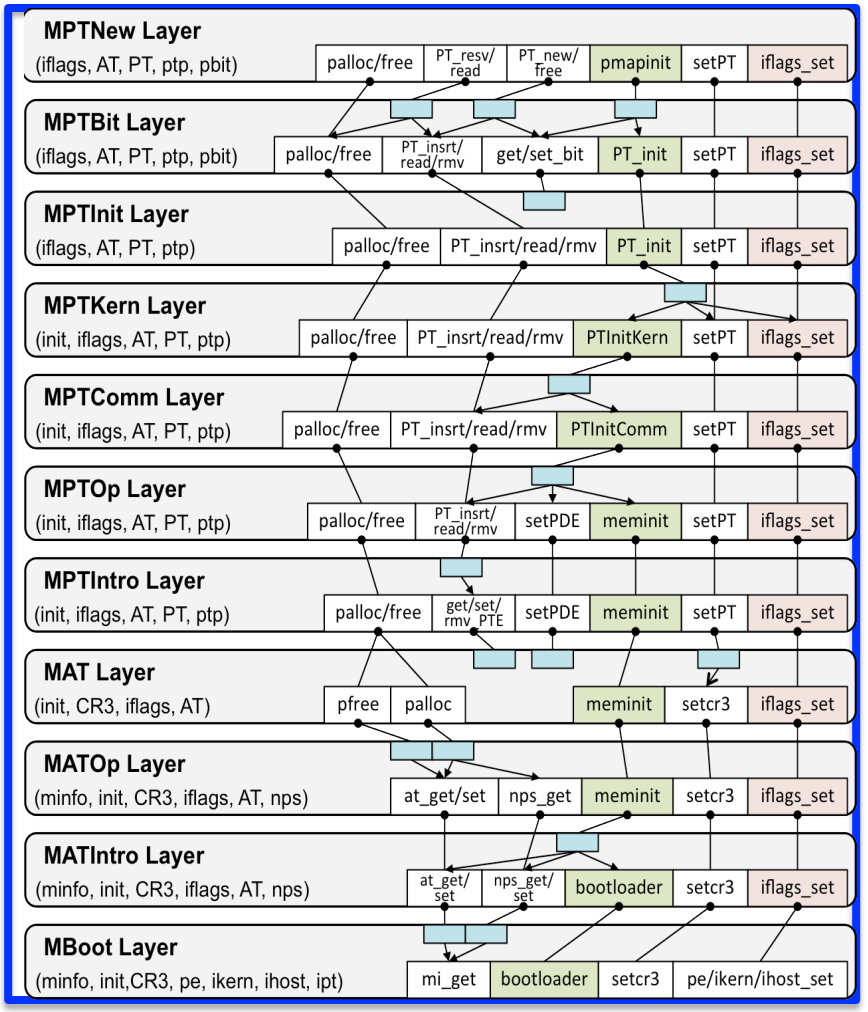






Decomposing mCertiKOS

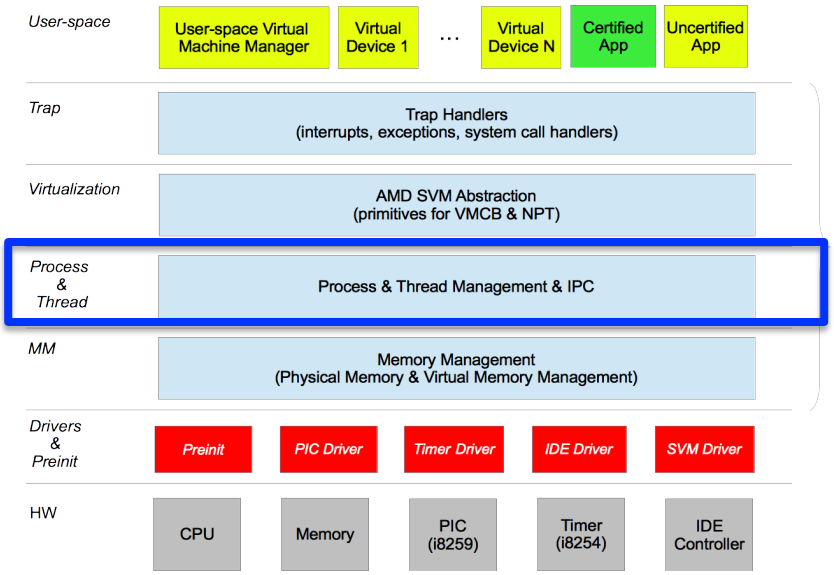




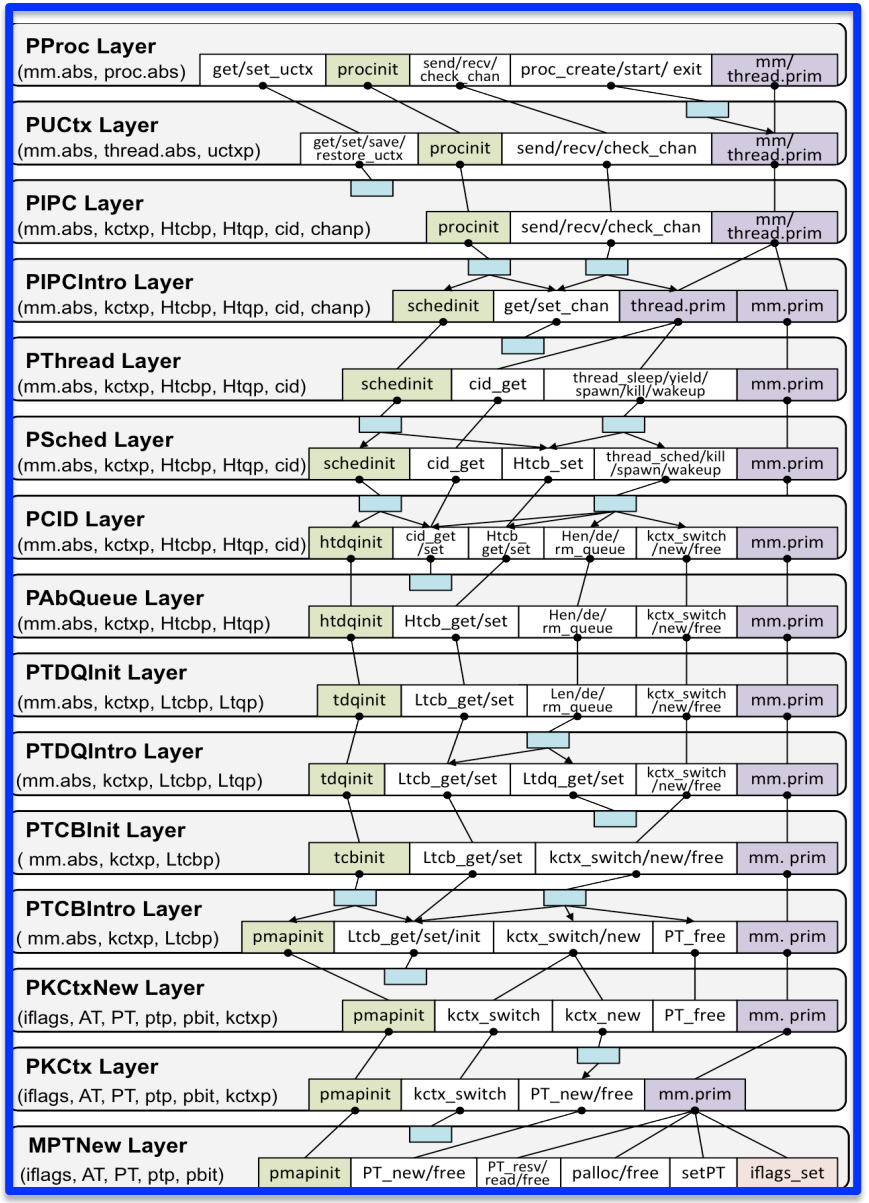
Physical Memory and Virtual Memory Management (11 Layers)

Based on the abstract machine provided by boot loader

Decomposing mCertiKOS (cont'd)

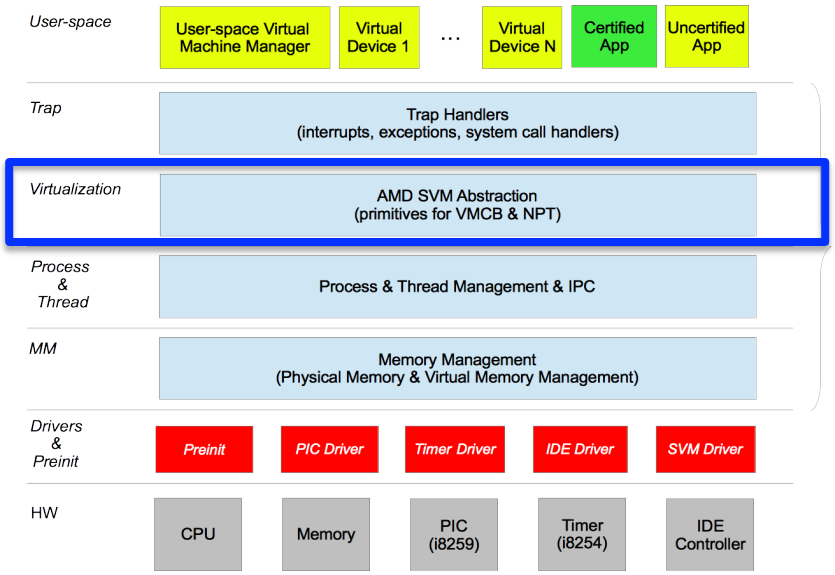


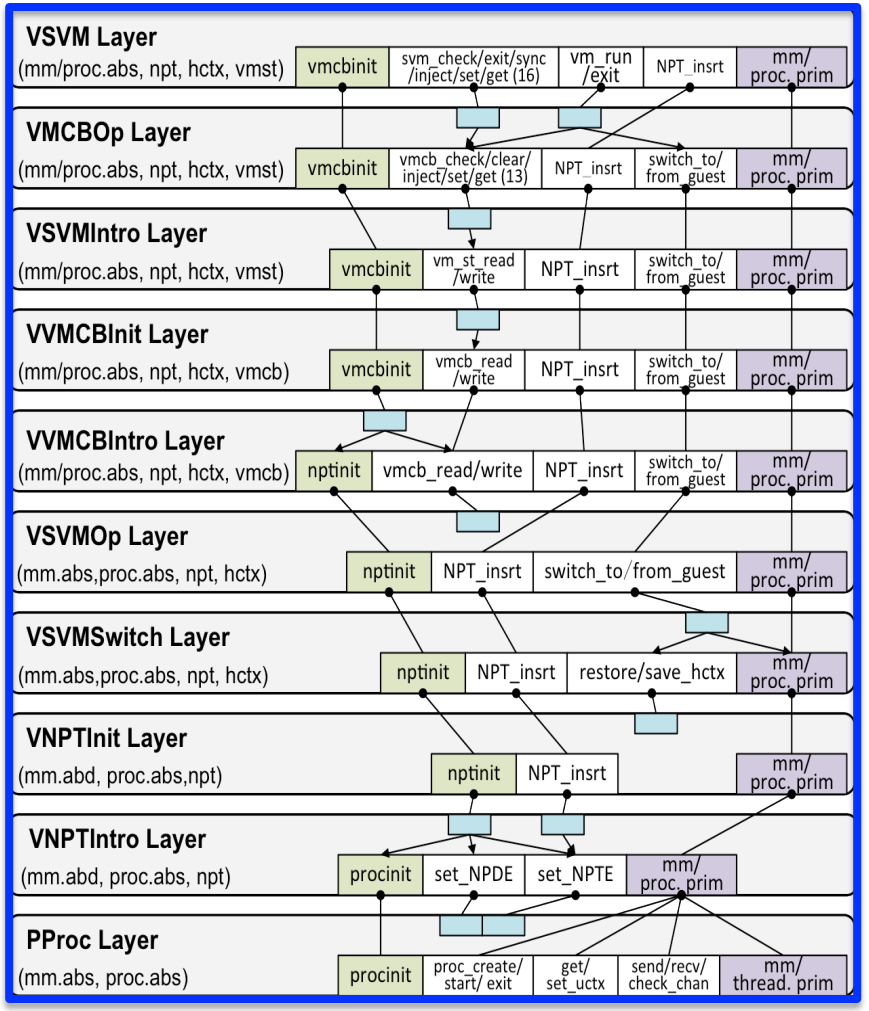
C Kernel



Thread and Process Management (14 Layers)

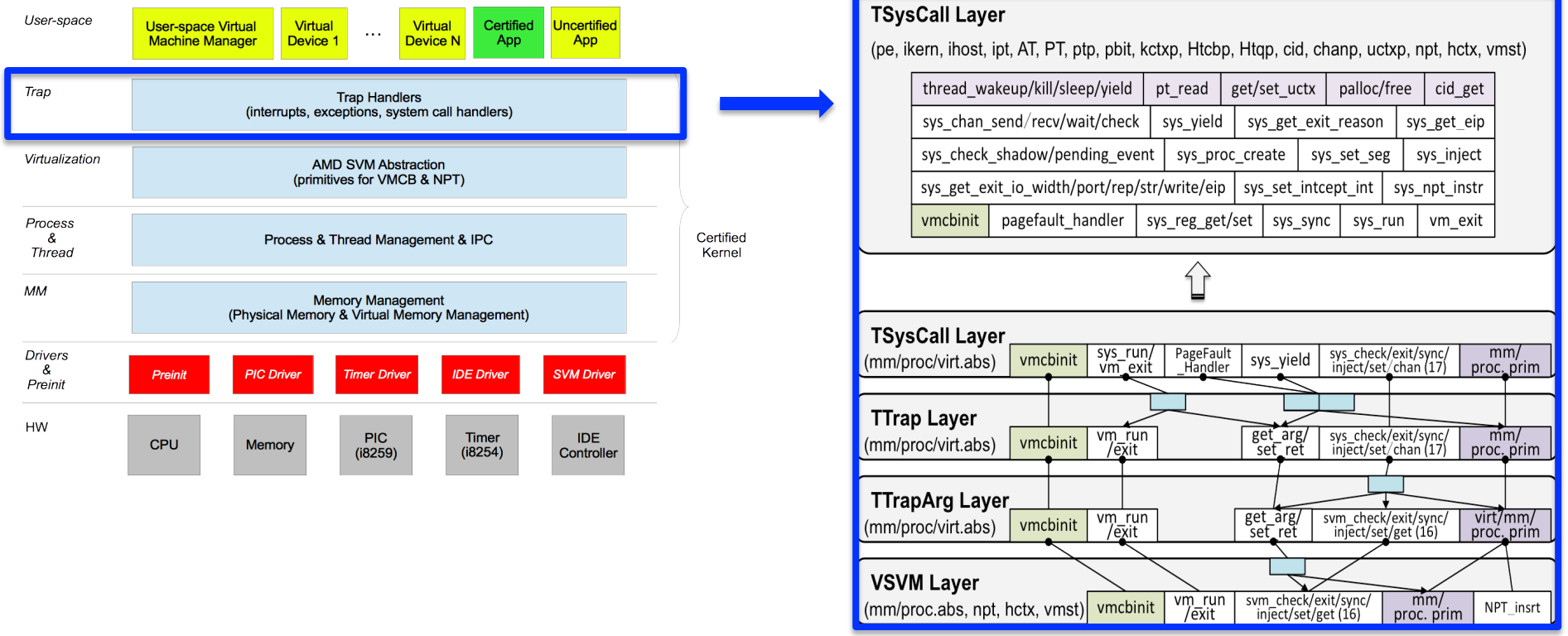
Decomposing mCertiKOS (cont'd)





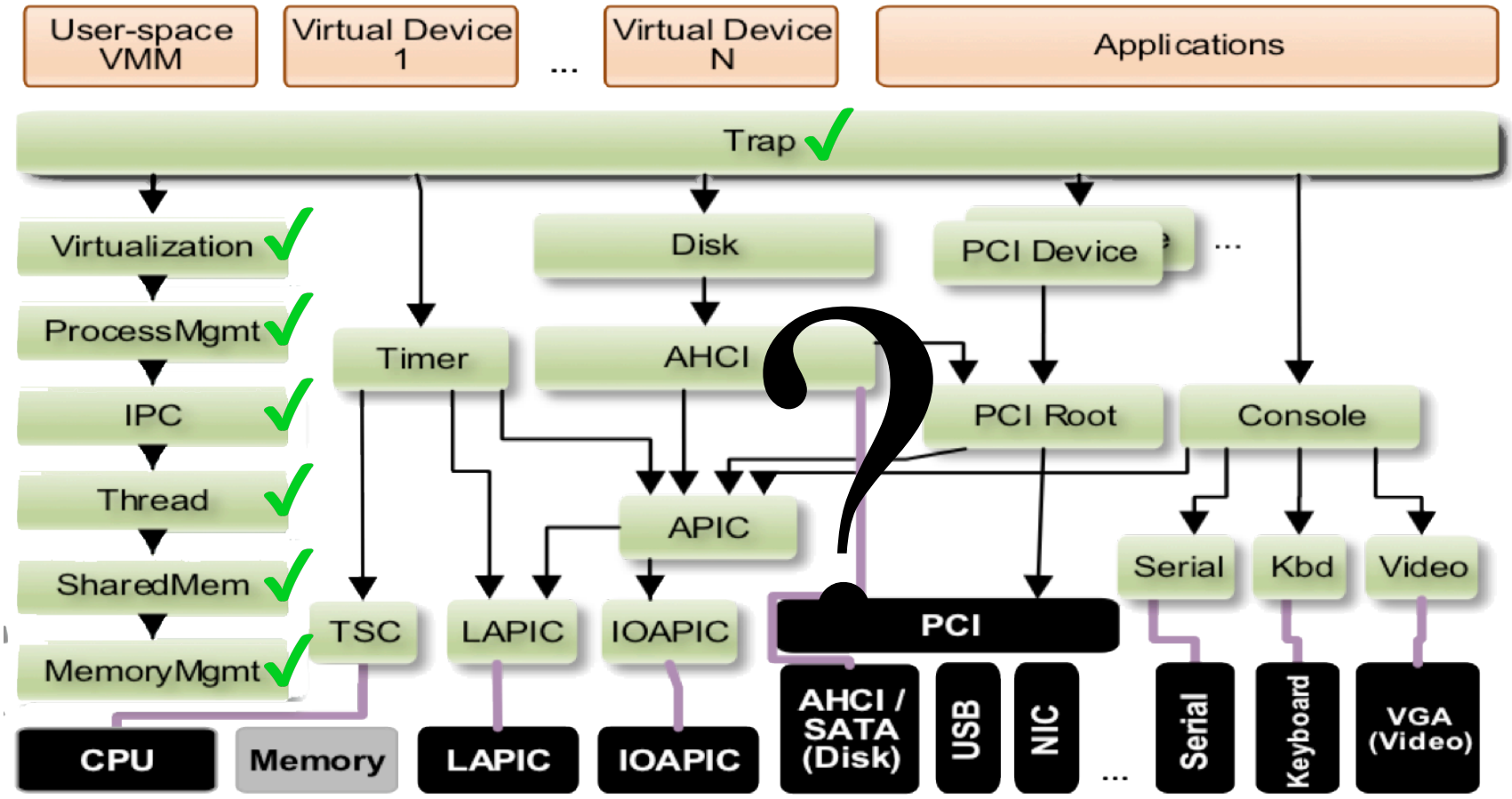
Virtualization Support (9 Layers)

Decomposing mCertiKOS (cont'd)

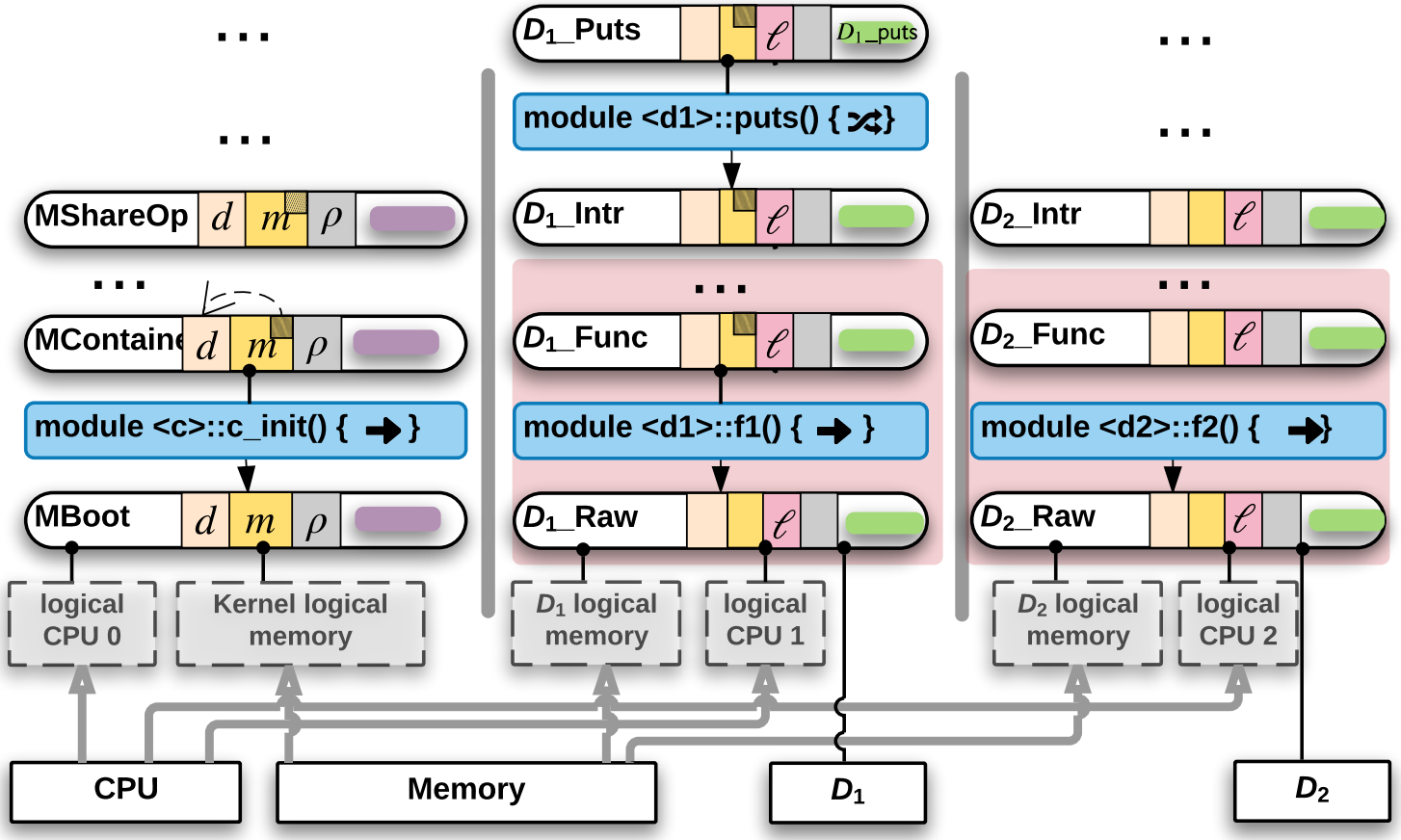


Syscall and Trap Handlers (3 Layers)

Interrupts & Devices?

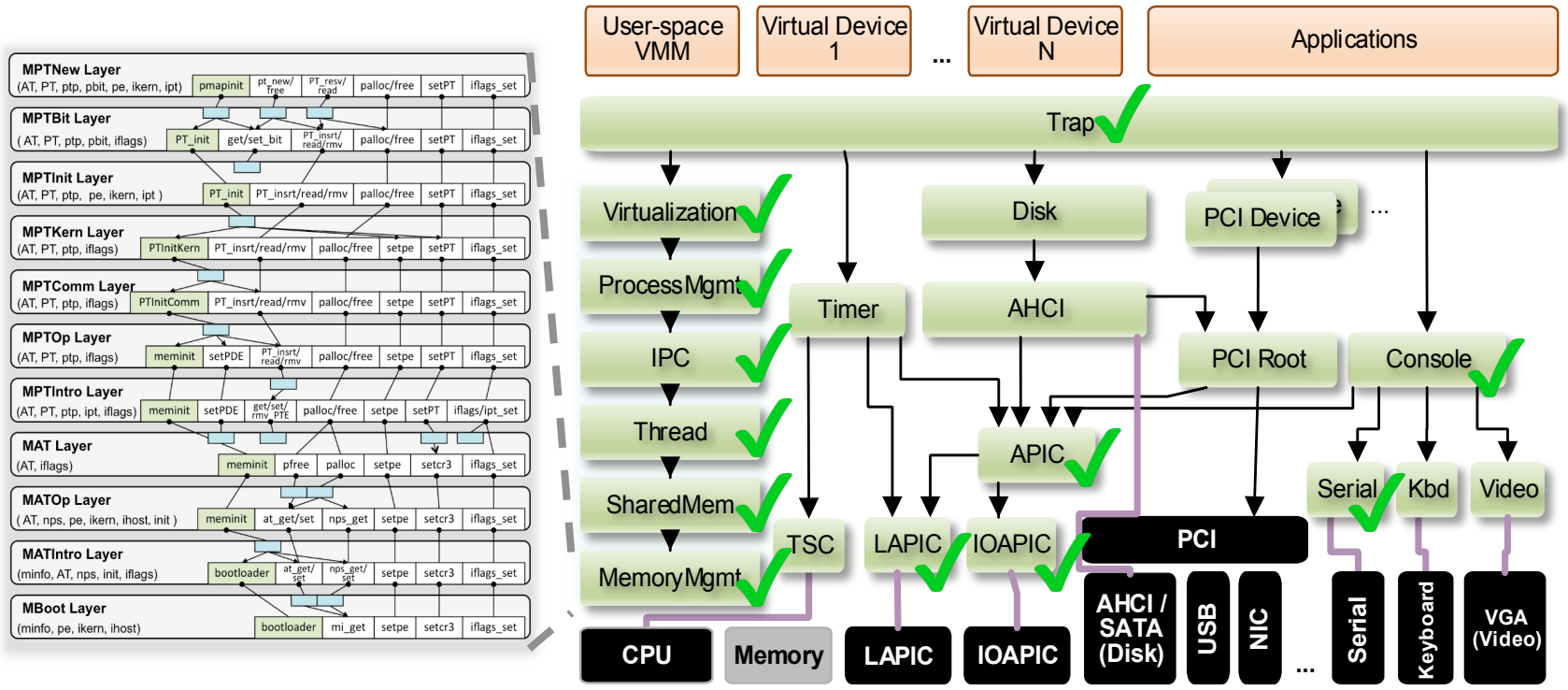


Certified Abstraction Layers with Multiple Logical CPUs



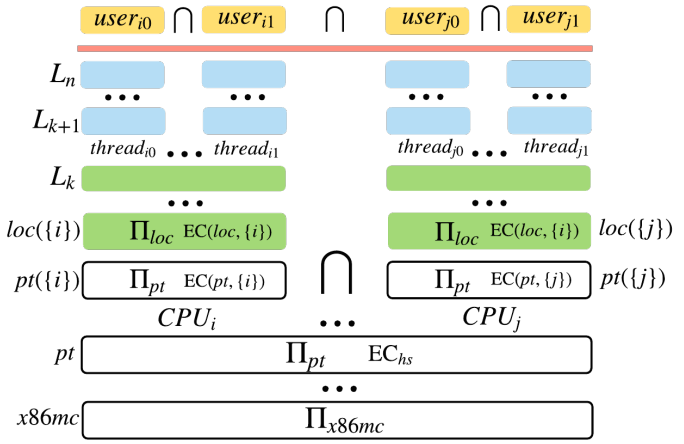
mCertiKOS w. Interrupts & Devices [PLDI'16]

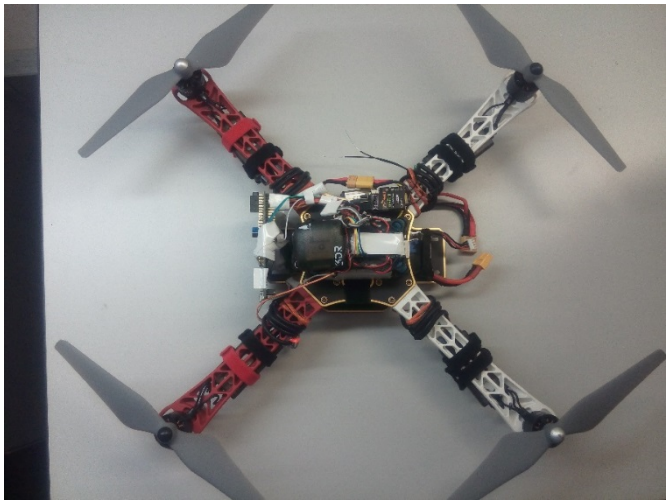
The first formally verified *interruptible* OS kernel with *device drivers (quick demo on the extracted kernel)*.



Other Recent Accomplishments

- Concurrent & Multicore CertiKOS
- End-to-End Verification of Information-Flow Security Properties [PLDI'16]
- CertiKOS on ARM --- Certified QuadCopter





PL Meets OS: A Marriage Made in Heaven?

- PL is about uncovering the laws of abstraction in the cyber world
- PL is to use abstraction to reduce complexities
- PL depends on the underlying OS for sys lib. & managing resources
- Many PL issues can be easily resolved in OS

- OS is about building layers of abstraction (e.g., VMs) for the cyber world
- OS is full of complexities
- OS is to manage, multiplex, and virtualize resources
- OS really needs PL help to provide safety and security guarantees

The CertiKOS / DeepSpec Project

Killer-app: high-assurance “cyber” systems (of systems)!

Conjecture: Today’s PL’s fail because they ignored OS, and today’s OS’es fail because they get little help from PLs

Opportunities (or our short-term deliverables):

- New certified system software stacks (CertiKOS ++)
- New certifying programming languages (DS vs. C & Asm)
- New certified programming tools
- New certified modeling & arch. description languages
- We verify all interesting properties (not just safety / partial correctness properties)