CertiKOS: Certified Kit Operating Systems

DeepSpec Summer School 2017

Zhong Shao

Yale University

July 24 - 28, 2017

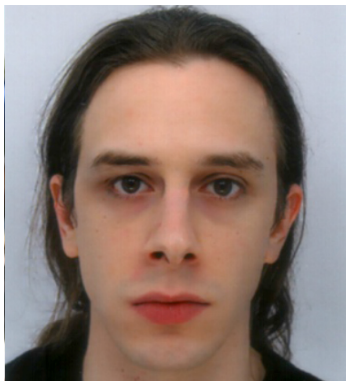
http://flint.cs.yale.edu

Acknowledgement: Ronghui Gu, David Costanzo, Jeremie Koenig, Tahina Ramananandro, Newman Wu, Hao Chen, Jieung Kim, Vilhelm Sjoberg, Hernan Vanzetto, Mengqi Liu, Lucas Paul, Wolf Honore, Pierre Wilke, Yuting Wang, Lionel Rieg, 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.

CertiKOS DSSS17 Lectures

- Day 1 (Monday 11-12:30)
  - CertiKOS Overview & Certified Abstraction Layers (CAL)
  - CAL Tutorial/Homework in Coq (by Jeremie Koenig)
- Day 2 (Tuesday 11-12:30)
  - CAL Tutorial/Homework in Coq & LayerLib (by Jeremie Koenig)
  - Certified Sequential OS Kernel (mCertiKOS)
- Day 3 (Thursday 4-5:30)
  - Observation Functions & Security-Preserving Simulation
  - End-to-End mCertiKOS Information Security Proofs
- Day 4 (Friday 11-12:30)
  - mCertiKOS with Interrupts & Certified Device Drivers
  - Multicore and Multithreaded Concurrency

CertiKOS Members as TAs

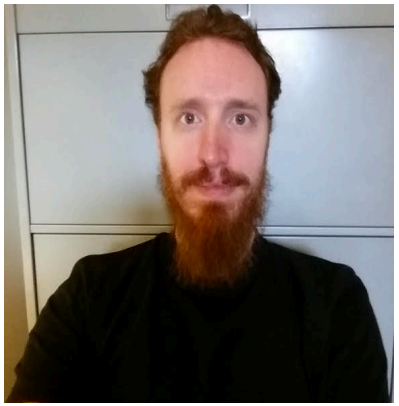


Jeremie Koenig

Wolf Honore

Jieung Kim

Vilhelm Sjoberg



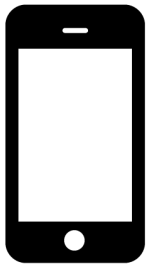
Lucas Paul

Yuting Wang

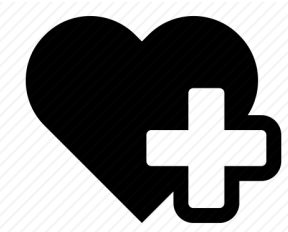
Do we really need high-assurance OS?

Desktop

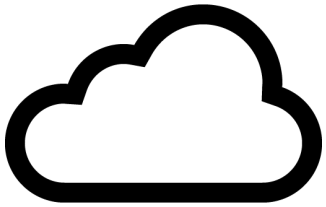
Transportation



Mobile



Health



cloud

Computer System

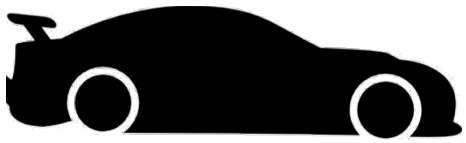
Aviation

Financial

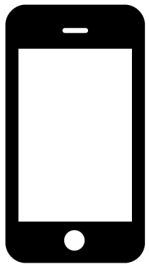
Environment

Do we really need high-assurance OS?

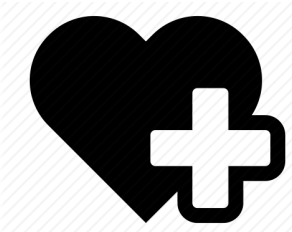
Desktop



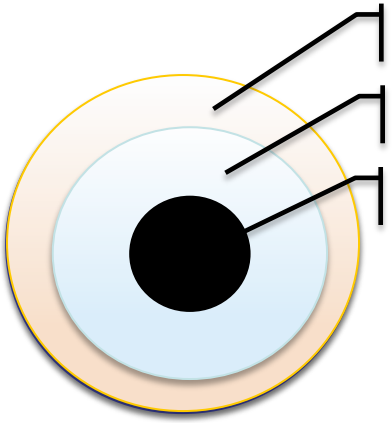
Transportation



Mobile



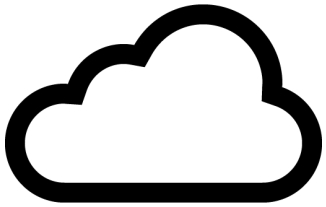
Health



Applications

OS

Hardware



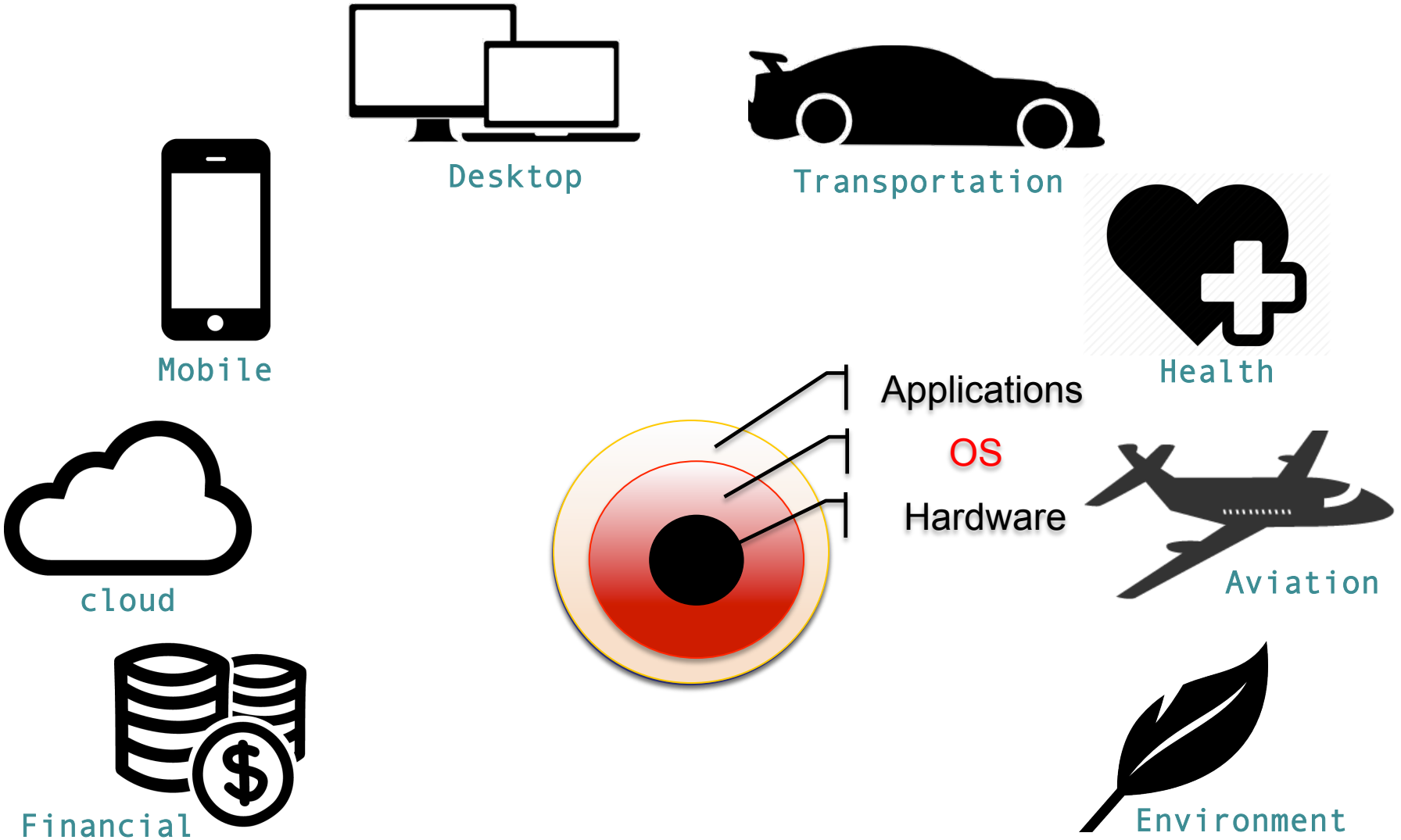
cloud

Aviation

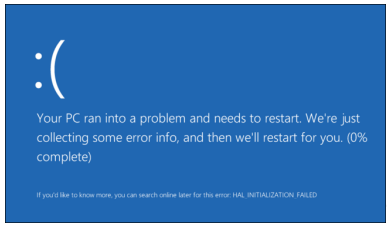
Financial

Environment

Do we really need high-assurance OS?



Do we really need high-assurance OS?

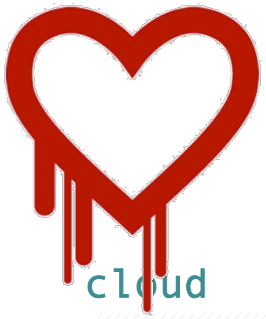


Crash

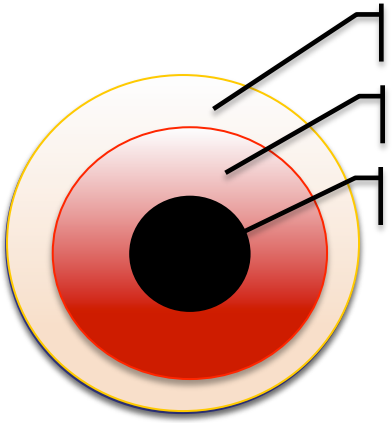
Accident

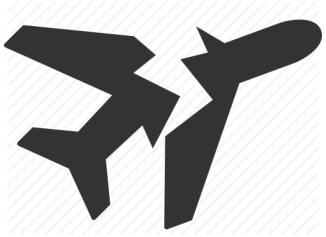
Mobile

Life



cloud





Loss

Financial

Environment

OS Landscape (July 2017)

Desktop: Linux, macOS, Windows, ChromeOS, freeBSD, …
Hypervisor/Cloud: Linux KVM & Docker, VMWare, Xen, …
Mobile: Android (Linux), iOS, …
Embedded: AGLinux, VxWorks, QNX, LynxOS, Integrity, …

- They are bloated and old, and contain many bugs
- Urgently need new OS for emerging platforms & apps

  (IoTs, Drones, Self-Driving Cars, Cloud, NetworkOS, Blockchains, …)

OS evolution has reached an inflection point:

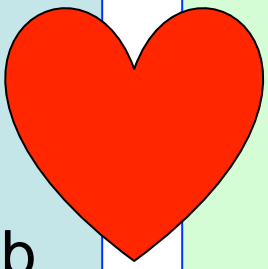
Need a certified "hacker-resistant" OS that provides security, extensibility, performance, and can work across multiple networked platforms & multipe scales.

Challenges & Problem Definition

- What is a certified OS kernel?
  - an OS kernel binary ***implements*** its specification?
  - what should its specification be like?
- What properties do we want to prove?
  - safety & partial correctness properties
  - total functional correctness
  - security properties (isolation, noninterference, confidentiality, integrity, availability, accountability)
  - resource usage properties (stack overflow, real time properties)
  - race-freedom, atomicity, and linearizability
  - liveness properties (wait-freedom, lock-freedom, obstruction freedom, deadlock-freedom, starvation freedom)
- How to cut down the cost of verification?

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

PL Meets OS: The Reality?

1967-2017

- Operational semantics is not compositional; denotational semantics does not scale
- Type-safe languages (Modula-3, Java, C#, Rust) only prove type-safety but make specs more complex
- PL theory occupied with higher-order functional languages, polymorphism, divergence, process calculi
- Hoare logic & separation logic do not scale; specs become complex very quickly

- OS still written in C, C++, and assembly
- Lack formal specification
- Unclear how to define various desirable correctness & security & liveness properties
- Lack clean device model; driver code still messy
- Preemption, interrupts, virtualization, fast IPC, concurrency, storage systems, distributed systems, clouds …

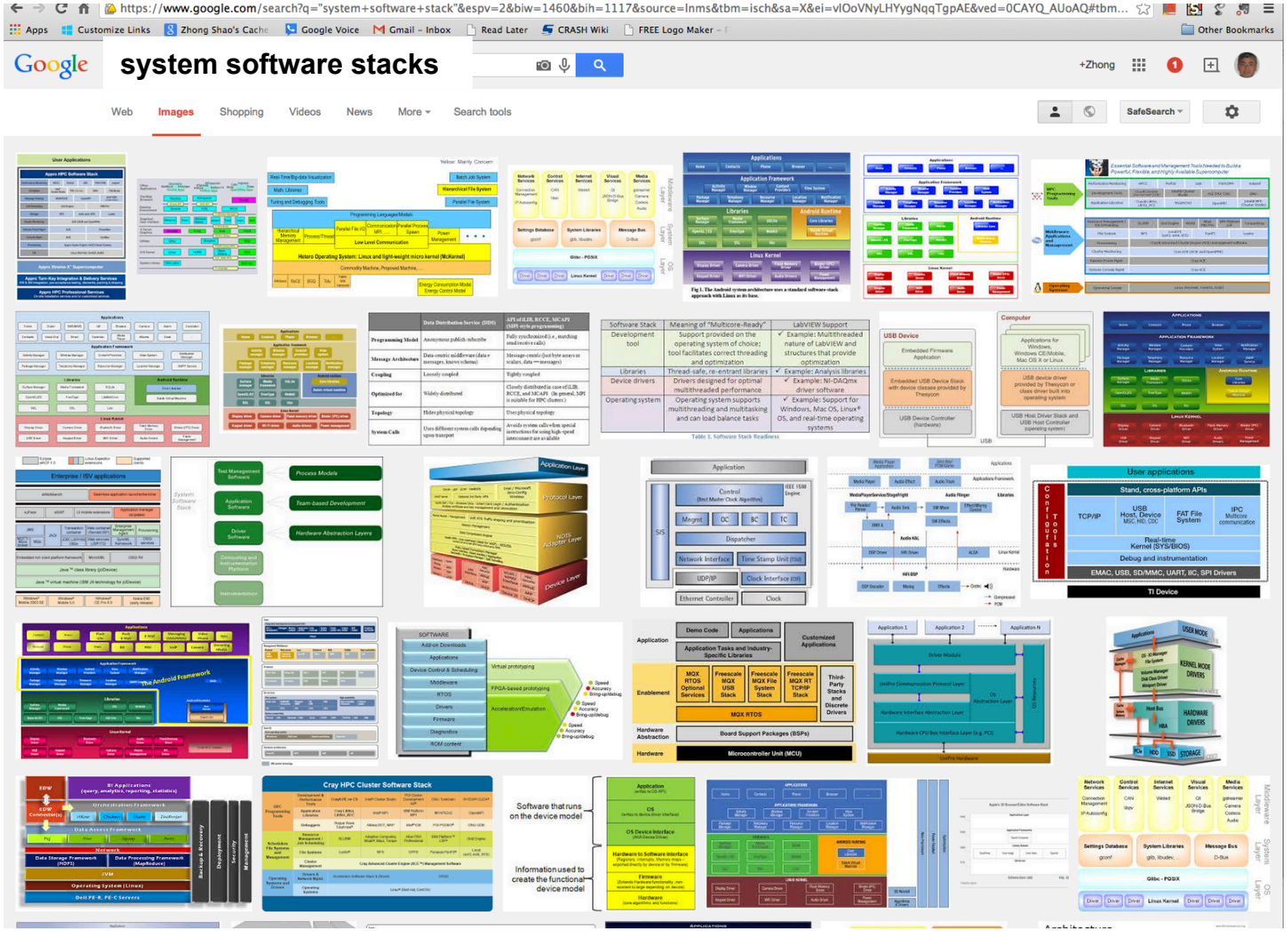
The Prequel to CertiKOS

- 1989-1995: Using SML/NJ with call/cc to model and implement OS components; FOX project, SPIN, …
- 1995-1998: Develop type systems for compiler intermediate languages; compiling ML module languages w. dependent types into FOmega++ (FLINT)
- 1998-2002: Need richer type systems (Fomega → CiC) for compiler intermediate lang. (TSCB) and assembly lang. (TAL); proof-carrying code

- 2002-2008: Foundational PCC in Coq; Certified Assembly Programming; CAP, SCAP, XCAP, OCAP, SAGL; separation logics for low-level assembly implemented in Coq; certified thread impl w. HW interrupts & certified GC; certified SMC
- 2008-2012: Concurrent objects with contextual refinement; certified thread libs; certified virtual memory manager
- 2012-2017: Certified abstraction layers & CertiKOS

Motivation

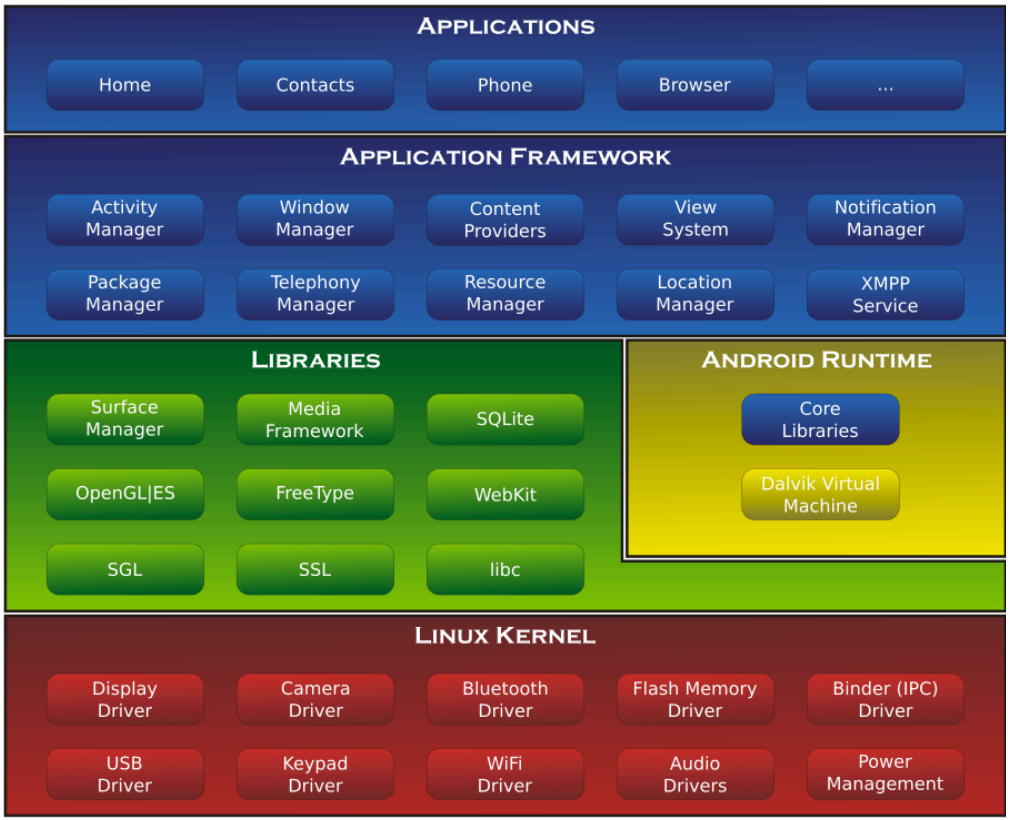
*How to build reliable & secure **system software stacks**?*

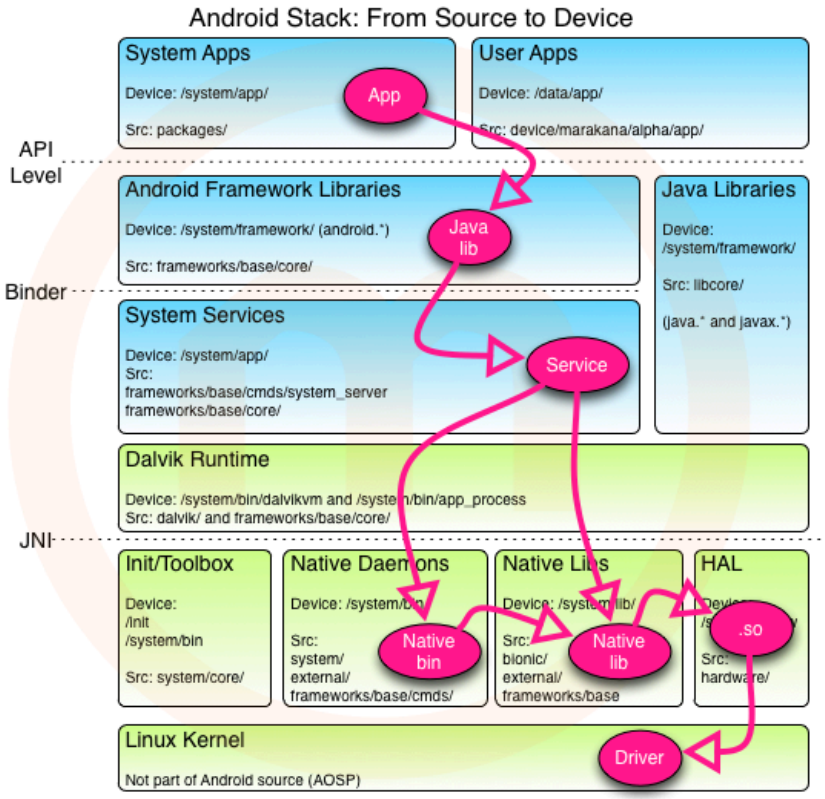


Motivation

Android architecture & system stack

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

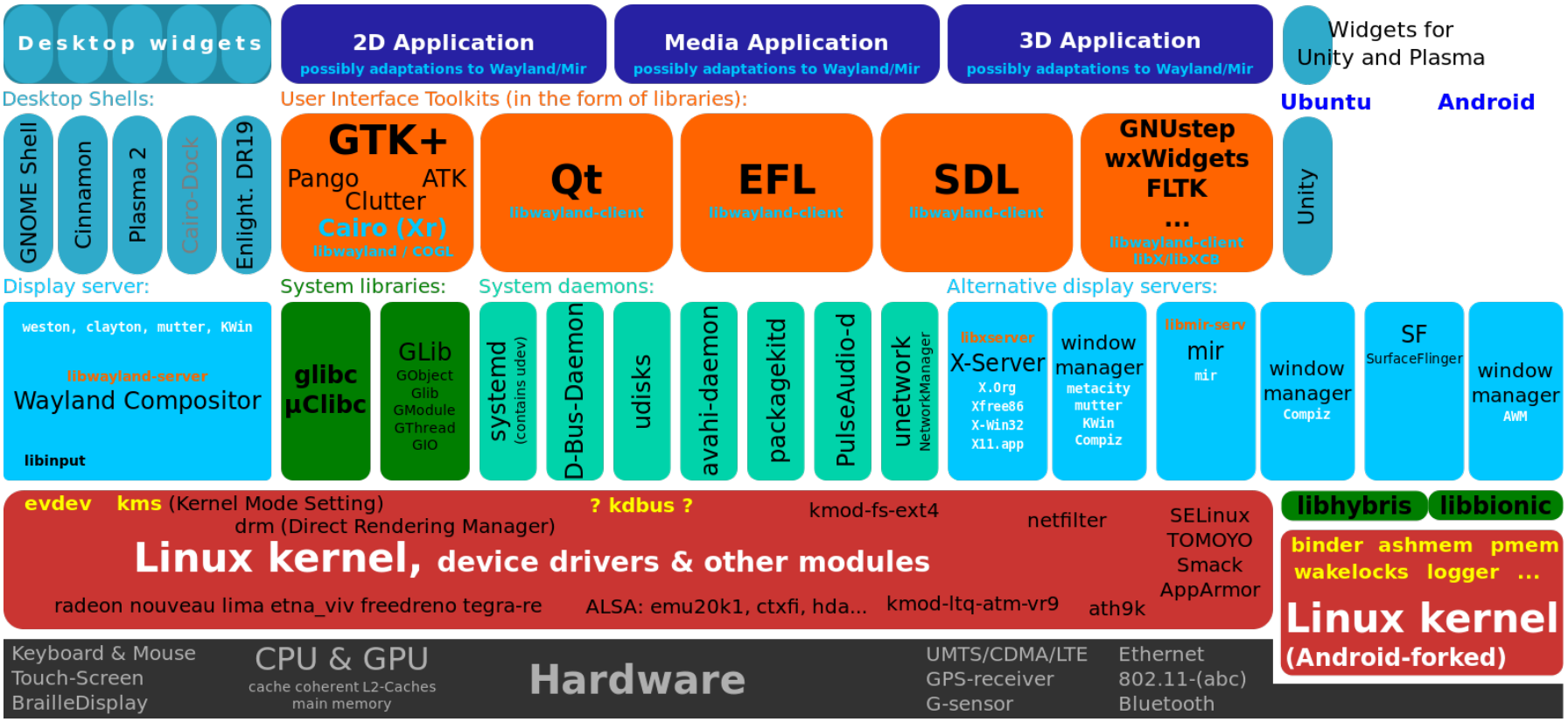




Motivation

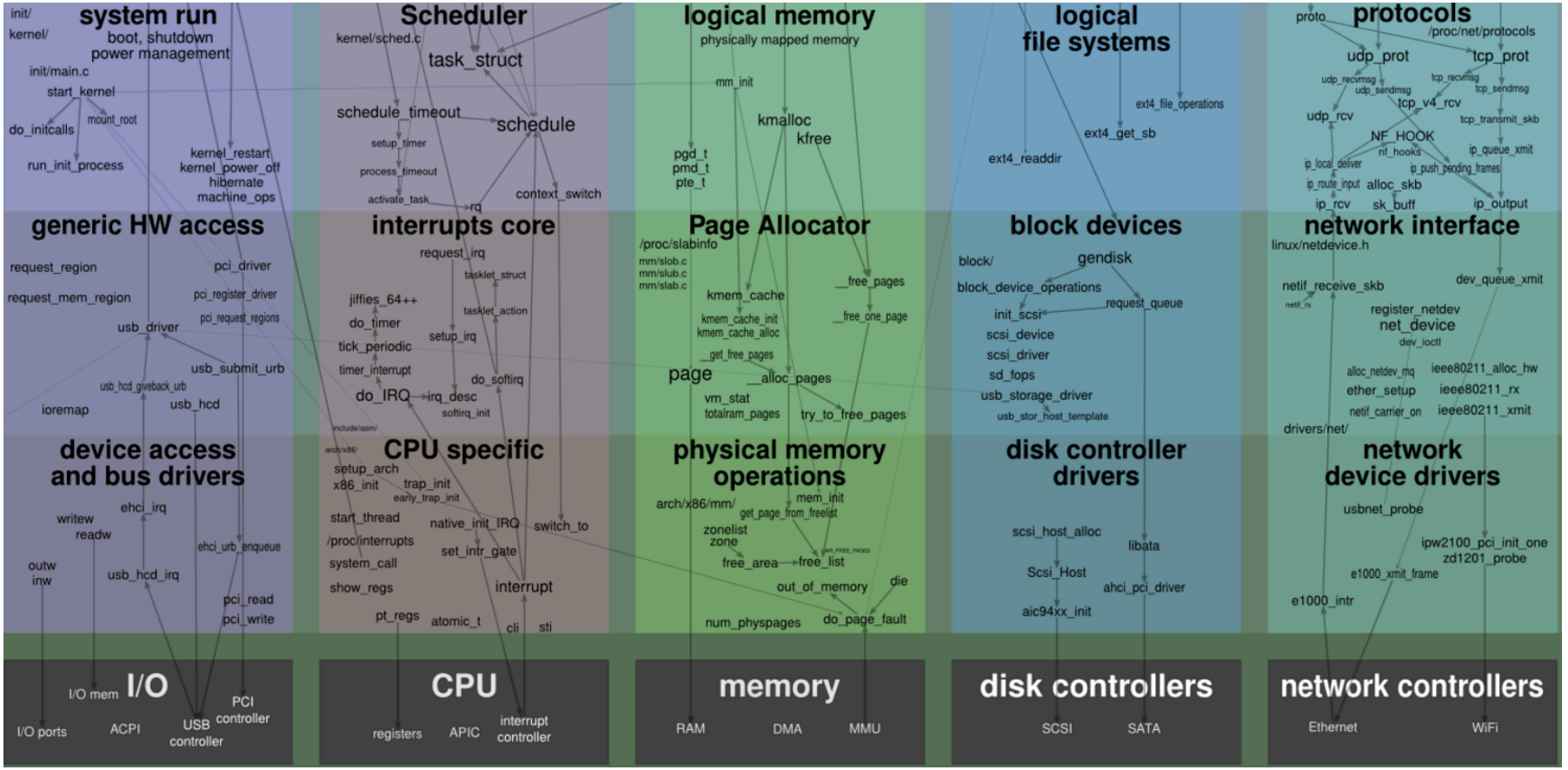
Visible software components of the Linux desktop stack

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/

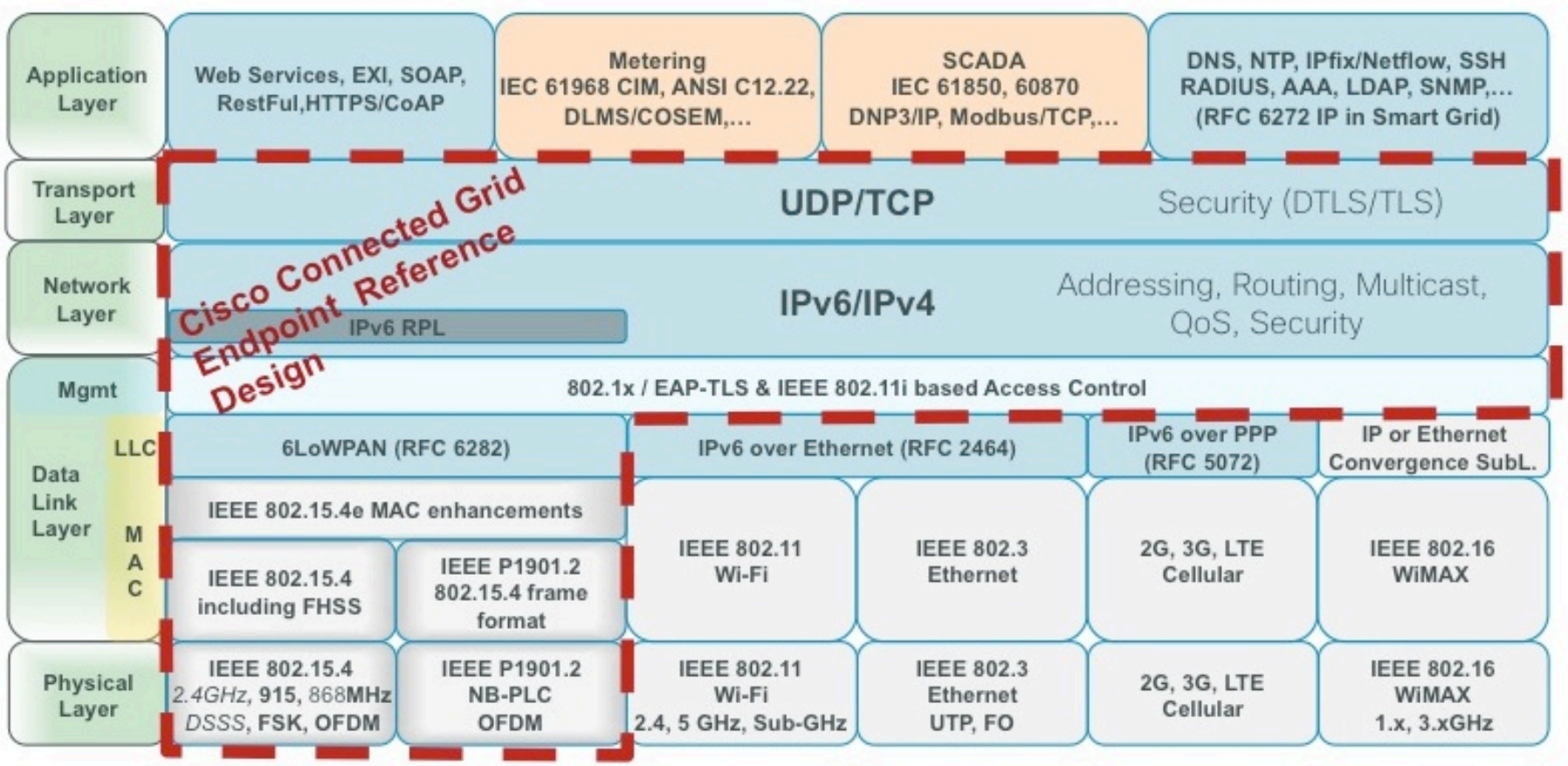
Essential Software and Management Tools Needed to Build a Powerful, Flexible, and Highly Available Supercomputer.

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



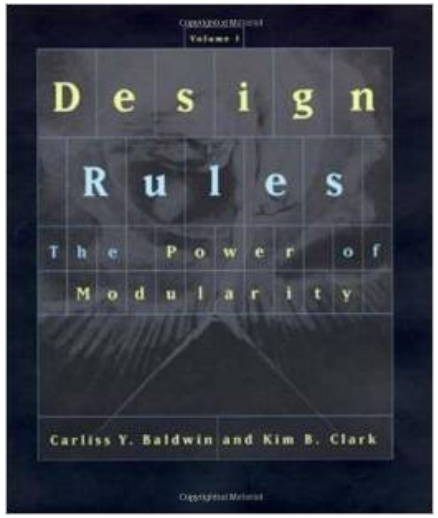
Motivation (cont'd)

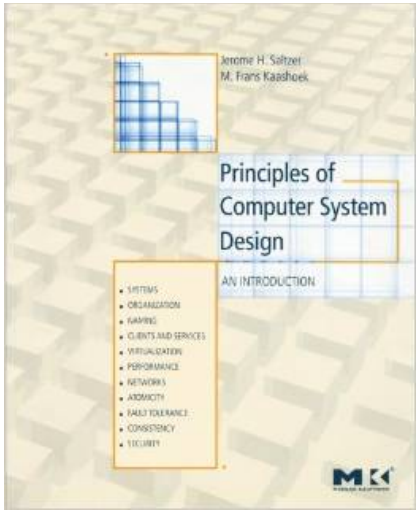
- Common themes: all system stacks are built based on abstraction, modularity, and layering

- 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)

Something magical going on … What is it?

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?

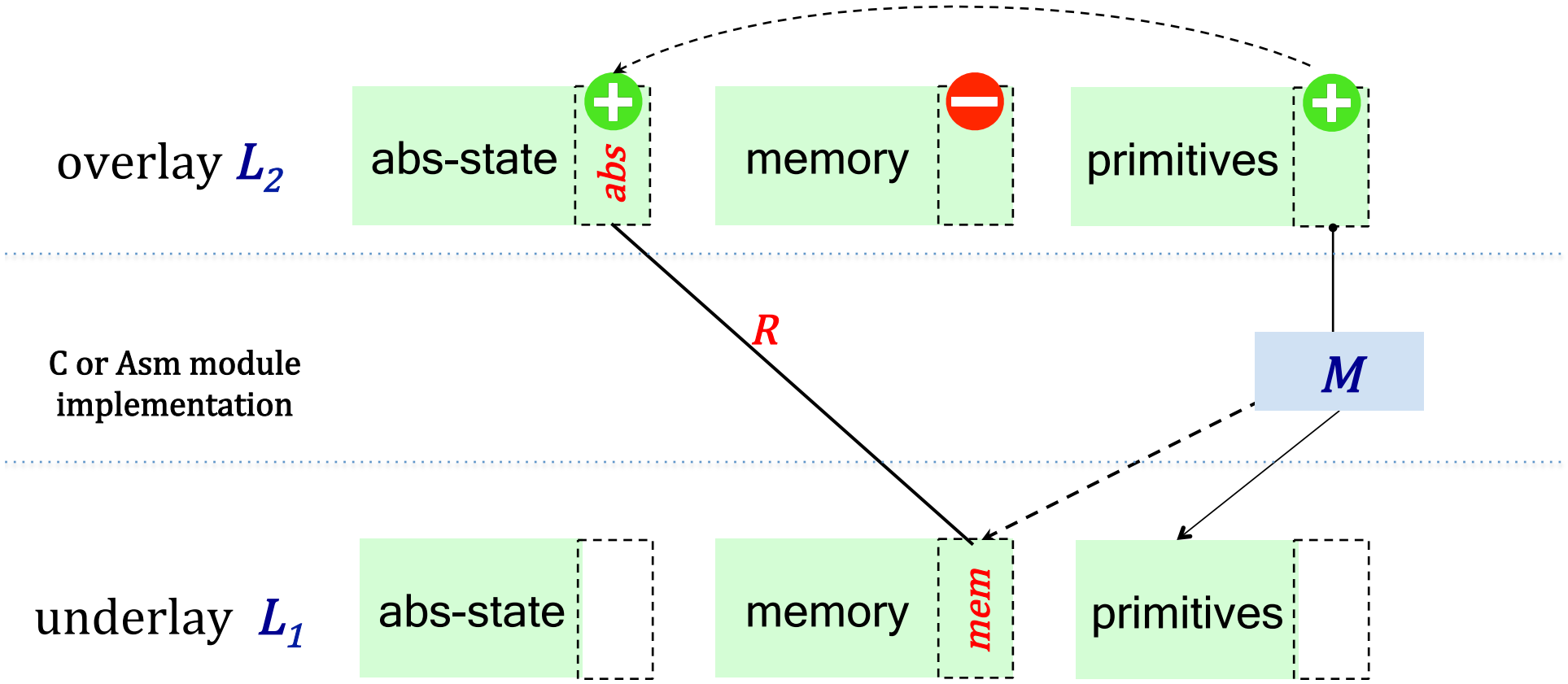
Our Contributions [POPL15]

- We introduce 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
  - mCertiKOS-hyper consists of 37 layers, took less than one-person-year to develop, and can boot Linux as a guest

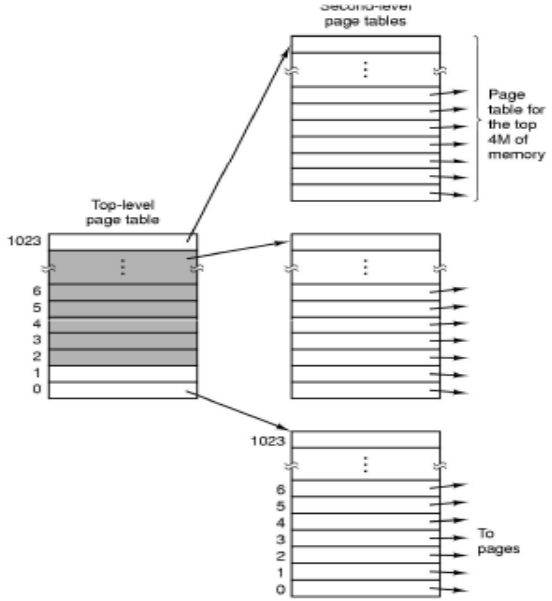
What is an Abstraction Layer?



Example: Page Tables

concrete C types

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



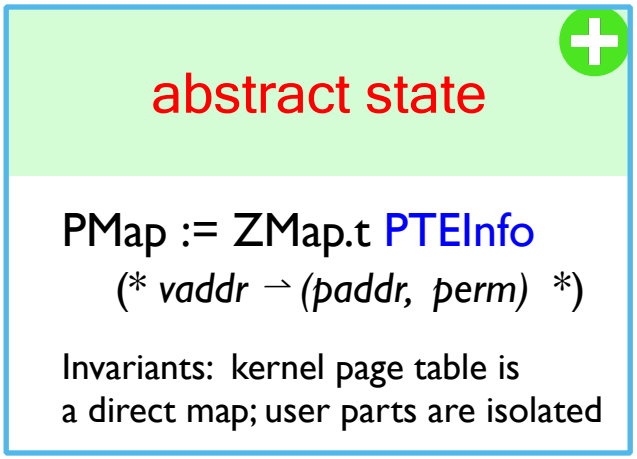
abstract Coq spec

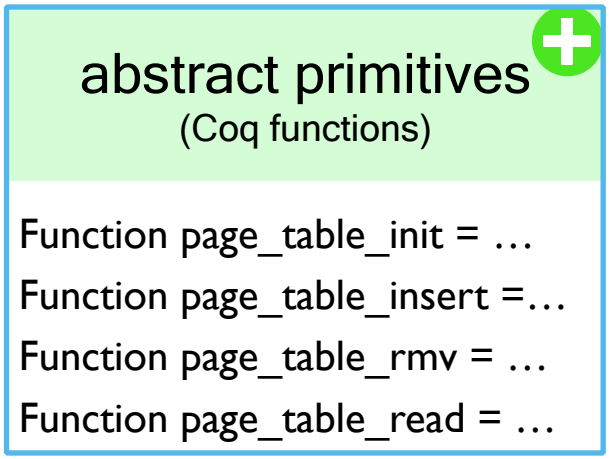
```
Inductive PTPerm:Type :=
    | PTP
    | PTU
    | PTK.

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

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

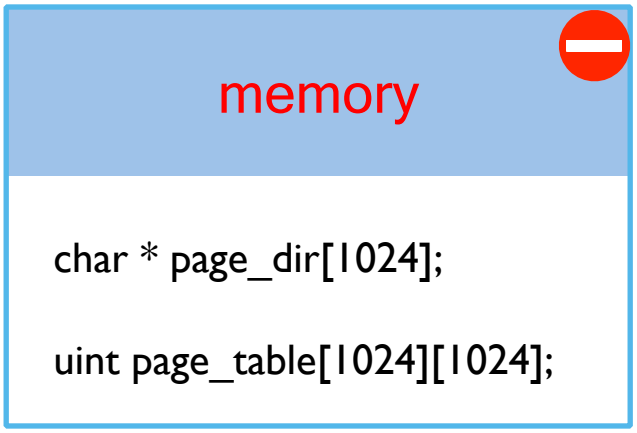
Example: Page Tables

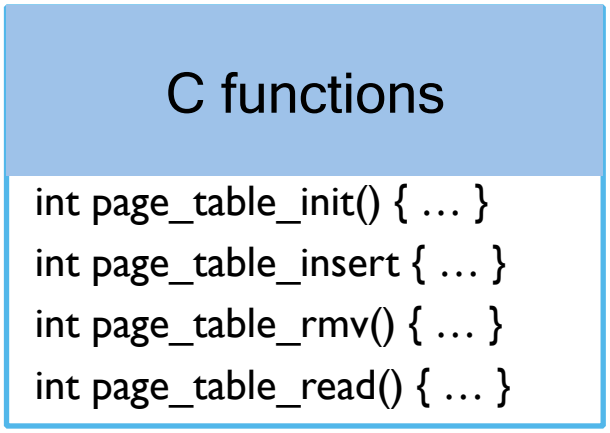
abstract layer spec

concrete C implementation



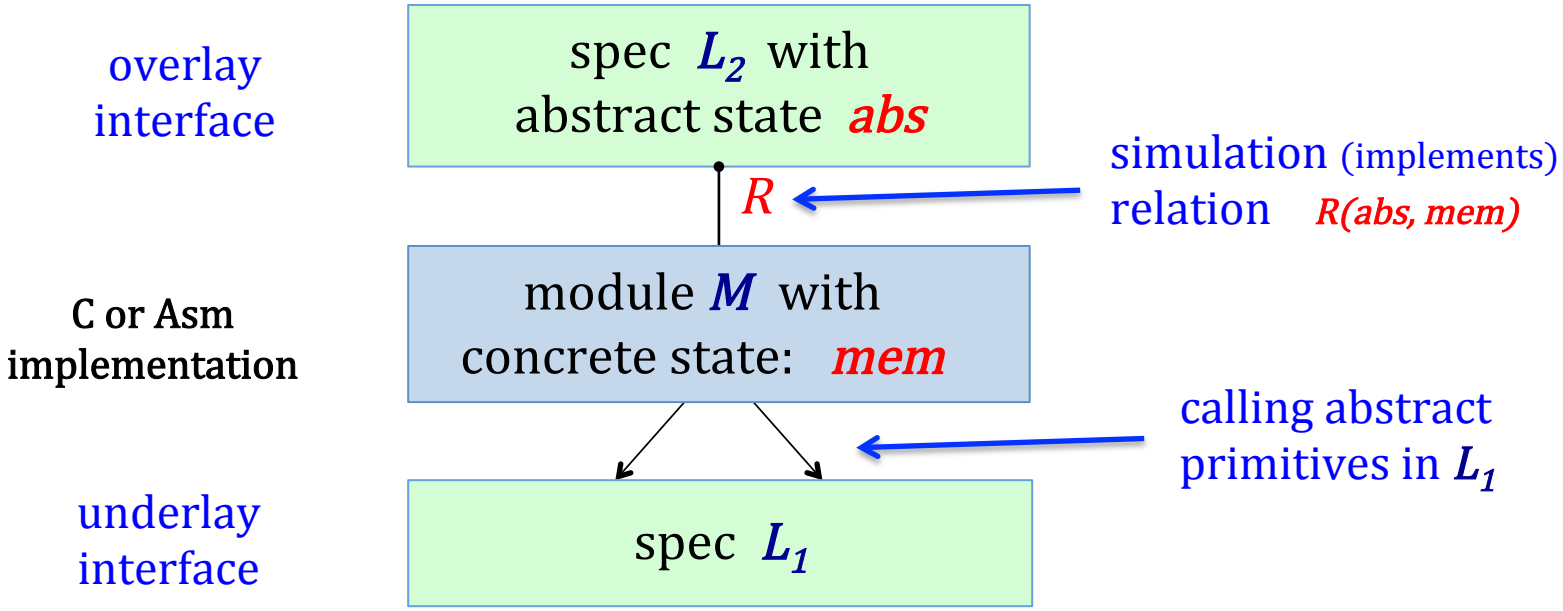






Formalizing Abstraction Layers

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	✓

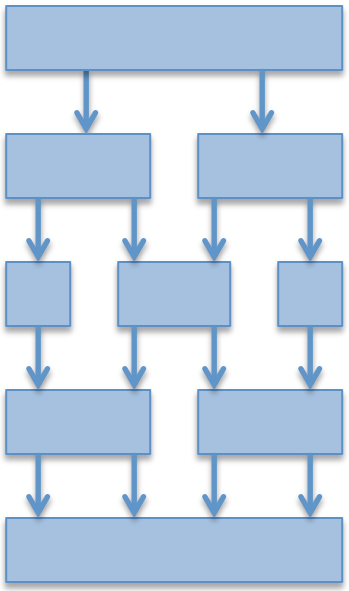
Problem w. “Rich” Specs


C or Asm module

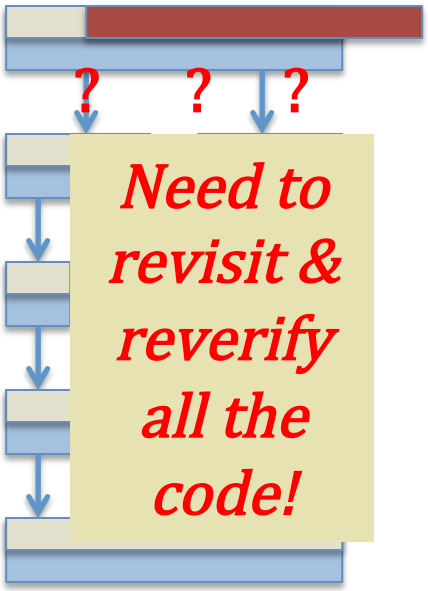

rich spec A


rich spec B

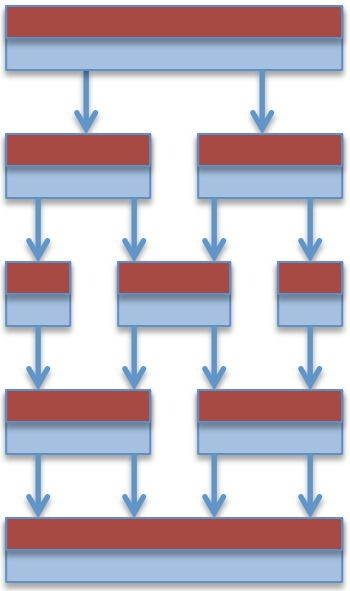
C & Asm Module Implementation



C & Asm Modules w. rich spec A



Want to prove another spec B ?

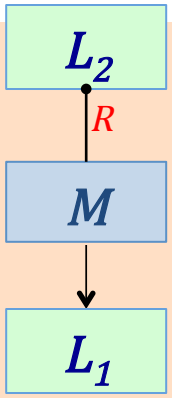


The Science of Deep Specs?

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

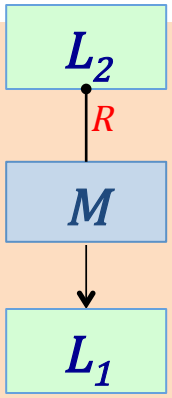
L_2 captures everything about running M over L_1

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*

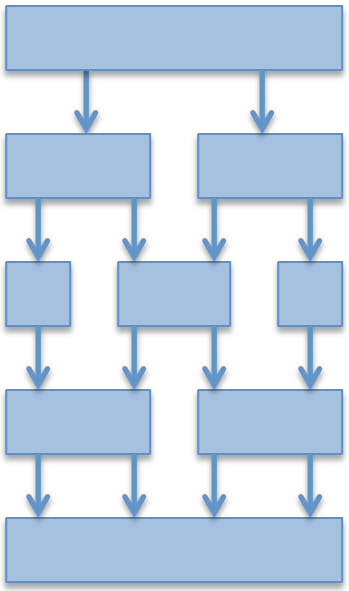
Shallow vs. Deep Specifications

C or Asm module

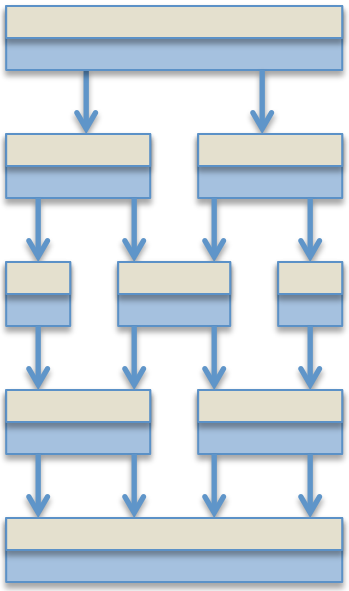
shallow spec

deep spec

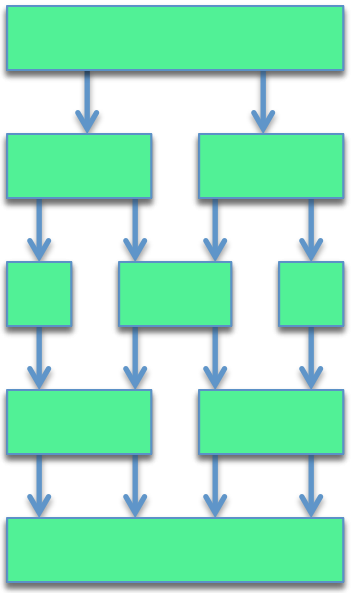
C & Asm Module Implementation



C & Asm Modules w. Shallow Specs



C & Asm Modules w. Deep Specs



Is Deep Spec Too Tight?

- Not really! It still *abstracts* away:
  - the *efficient* concrete data repr & impl. algorithms & strategies
- It can still be nondeterministic:
  - External nondeterminism (e.g., I/O or scheduler events) modeled as a set of deterministic traces relative to external events (*a la* CompCert)
  - Internal nondeterminism (e.g., sqrt, rand, resource-limit) is also OK, but any two implementations must still be *observationally equivalent*
- It *adds* new logical info to make it *easier-to-reason-about*:
  - auxiliary abstract states to define the full functionality & invariants
  - accurate precondition under which each primitive is valid

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

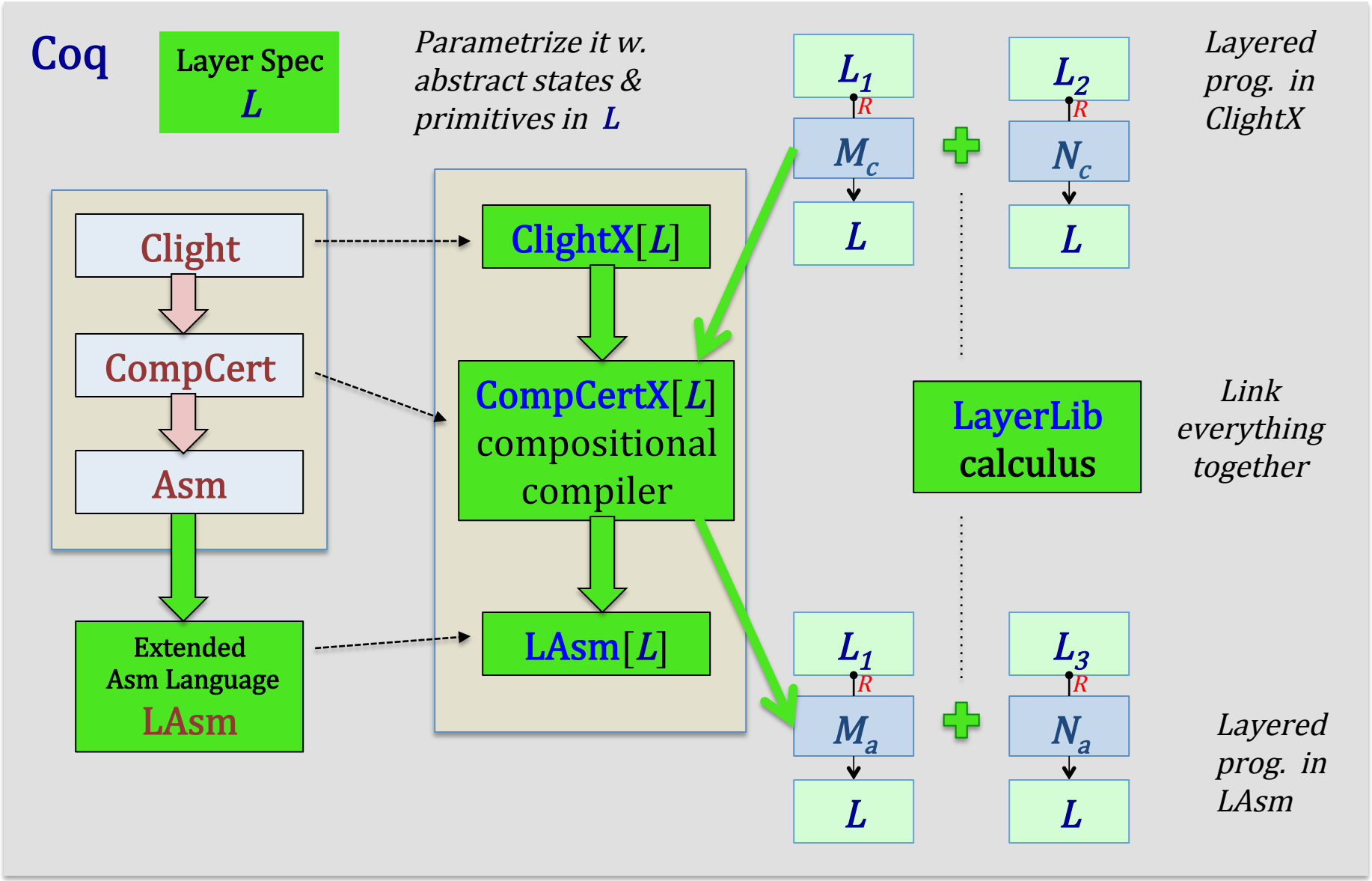
Our Contributions

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

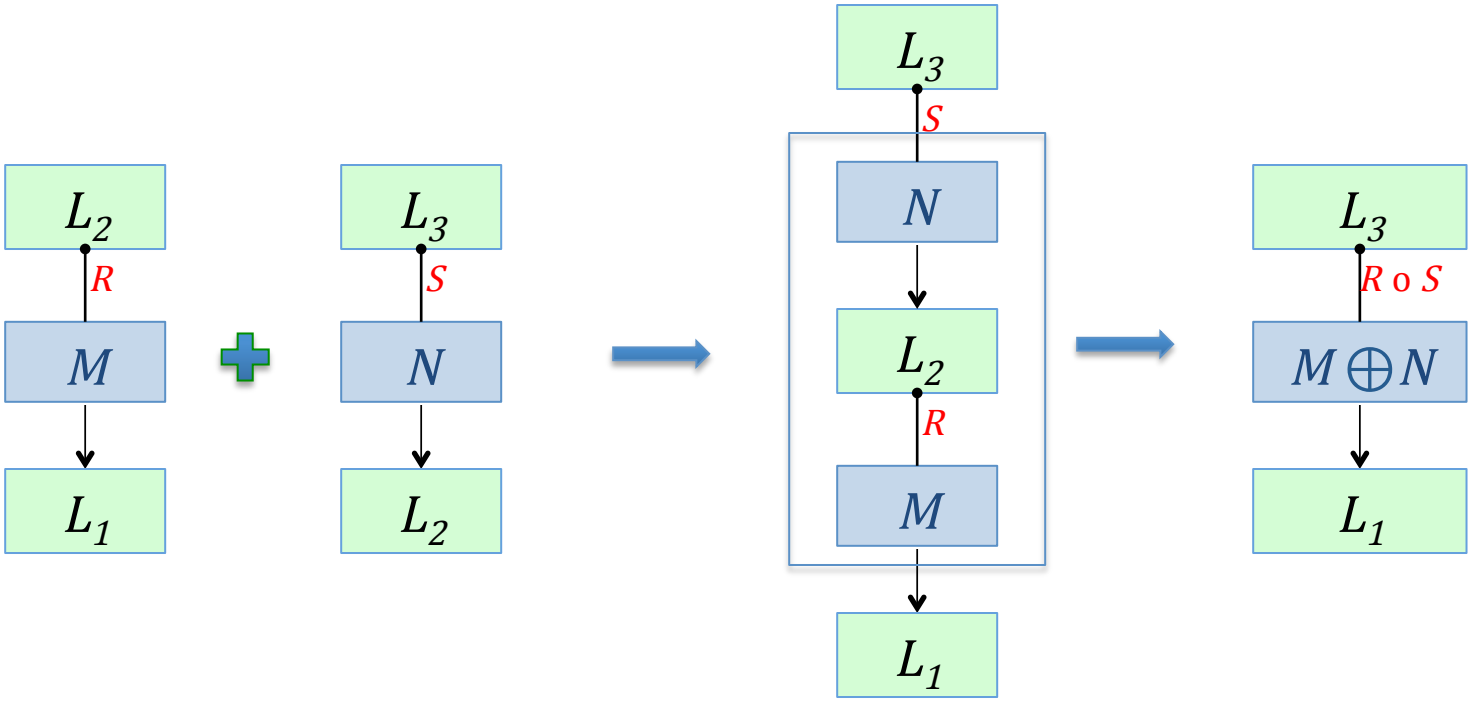
- We developed new languages & tools in Coq
  - A formal layer calculus for composing certified layers
  - ClightX for writing certified layers in a C-like language
  - LAsm for writing certified layers in assembly
  - CompCertX that compiles ClightX layers into LAsm layers

- We built multiple certified OS kernels in Coq
  - mCertiKOS-hyper consists of 37 layers, took less than one-person-year to develop, and can boot Linux as a guest

What We Have Done

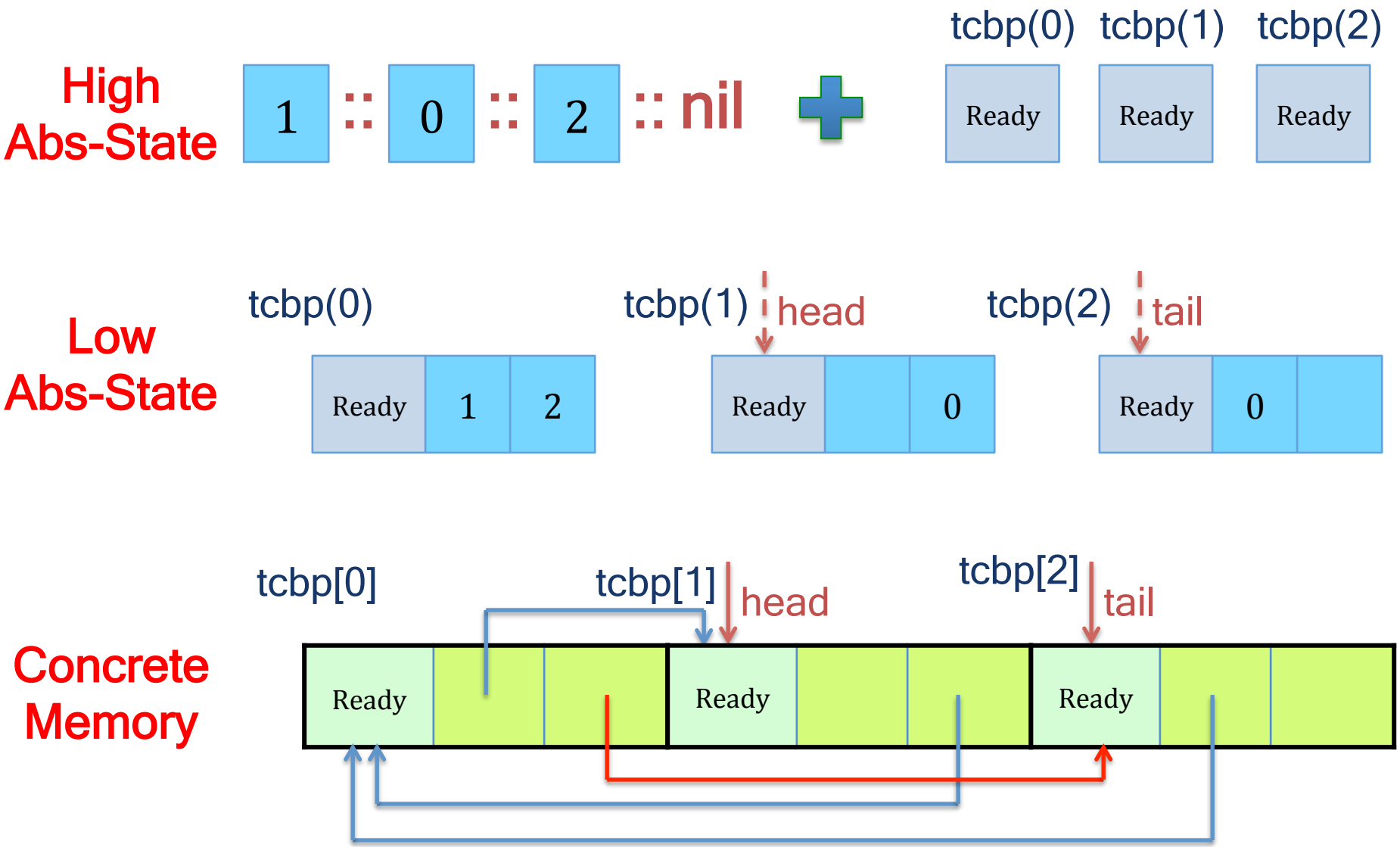


LayerLib: Vertical Composition



$$\frac{L_1 \vdash_R M : L_2 \qquad L_2 \vdash_S N : L_3}{L_1 \vdash_{R \circ S} M \oplus N : L_3}\ \textsc{Vcomp}$$

Example: Thread Queues



Example: Thread Queues

C Implementation

```
typedef enum {
  TD_READY, TD_RUN,
  TD_SLEEP, TD_DEAD
} td_state;

struct tcb {
  td_state tds;
  struct tcb  *prev,  *next;
};

struct tdq {
  struct tcb  *head,  *tail;
};

struct tcb tcbp[64];
struct tdq tdqp[64];

struct tcb * dequeue
     (struct tdq *q)  {
          ……  }
```

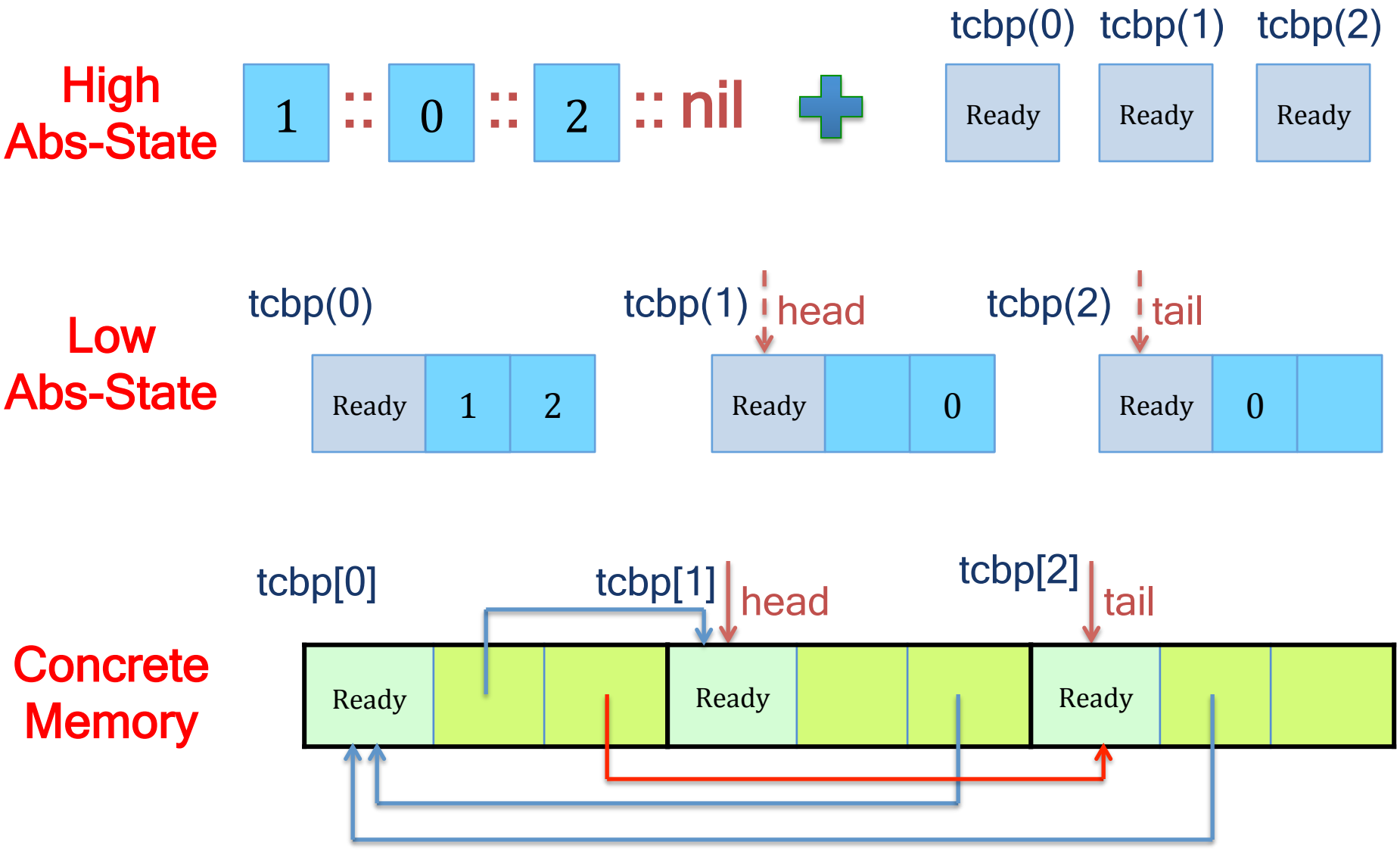
Low Layer Spec in Coq

```
Inductive td_state :=
| TD_READY | TD_RUN
| TD_SLEEP | TD_DEAD.

Inductive tcb :=
| TCBV (tds : td_state)
          (prev next : Z)

Inductive tdq :=
| TDQV (head tail: Z)

Record abs:={
          tcbp :  ZMap.t tcb;
          tdqp : ZMap.t tdq }

Function dequeue
       (d : abs) (i : Z) :=
……..
```

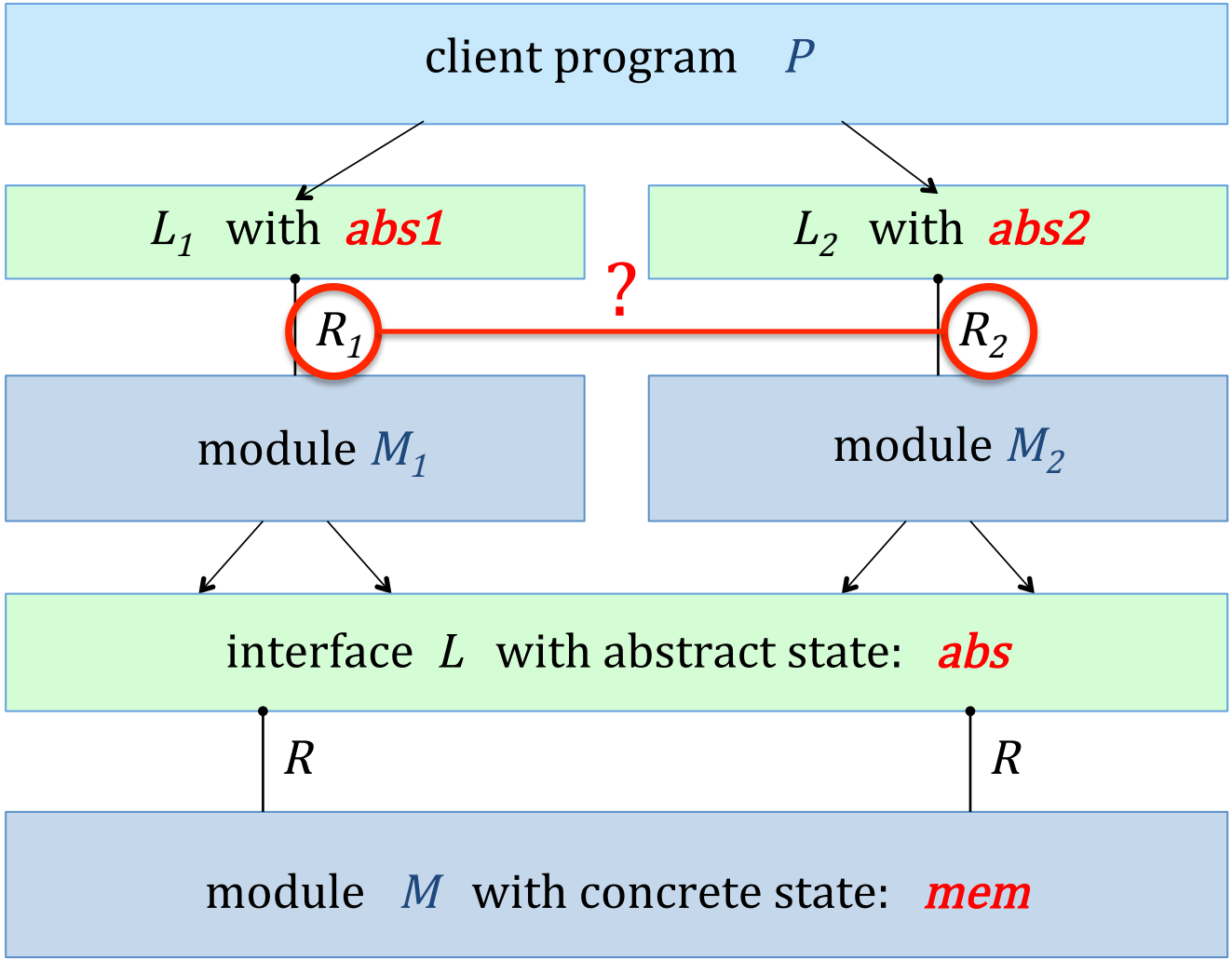
High Layer Spec in Coq

```
Inductive td_state :=
| TD_READY | TD_RUN
| TD_SLEEP | TD_DEAD.

Definition tcb := td_state.

Definition tdq := List Z.

Record abs':={
          tcbp : ZMap.t tcb;
          tdqp : ZMap.t tdq }

Function dequeue
      (d : abs') (i : Z) :=
match (d.tdqp i) with
  | h :: q' =>
      Some(set_tdq d i q', h)
  | nil => None
end
```

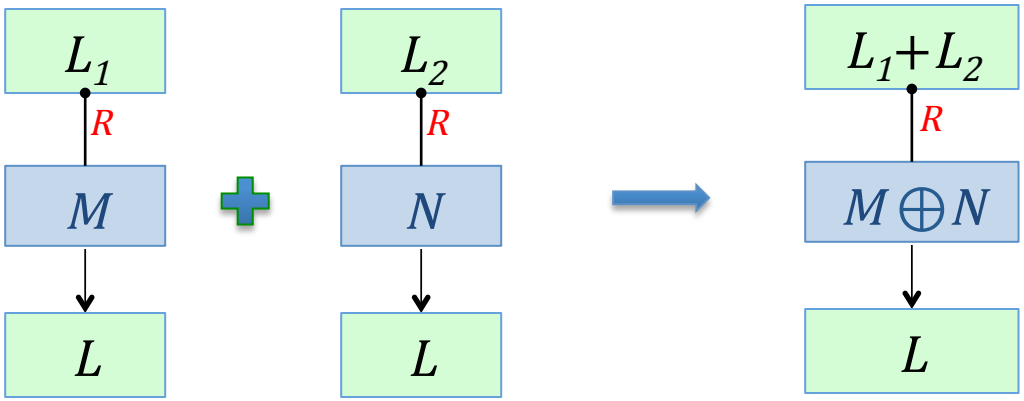
Example: Dequeue



Conflicting Abstract States?



LayerLib: Horizontal Composition



$$\frac{L \vdash_R M : L_1 \qquad L \vdash_R N : L_2}{L \vdash_R M \oplus N : L_1 \oplus L_2}\ \textsc{Hcomp}$$

- L_1 and L_2 must have the same abstract state
- both layers must follow the same simulation relation R

Programming & Compiling Layers

ClightX $L \vdash_R M_c : L_1$ ⟶ $L_1 \leq_R [\![M_c]\!]_{\text{ClightX}} (L)$

↓

CompCertX correctness theorem (where *minj* is a special kind of memory injection)

$$[\![M_c]\!]_{\text{ClightX}} (L) \leq_{minj} [\![\text{CompCertX[L]}(M_c)]\!]_{\text{LAsm}} (L)$$

↓

$$L_1 \leq_{R \circ minj} [\![\text{CompCertX[L]}(M_c)]\!]_{\text{LAsm}} (L)$$

↓

R must absorb such memory injection: $R \circ minj = R$ *then we have:*

$$L_1 \leq_R [\![\text{CompCertX[L]}(M_c)]\!]_{\text{LAsm}} (L)$$

↓

Let $M_a = \text{CompCertX[L]}(M_c)$ then $L \vdash_R M_a : L_1$ LAsm

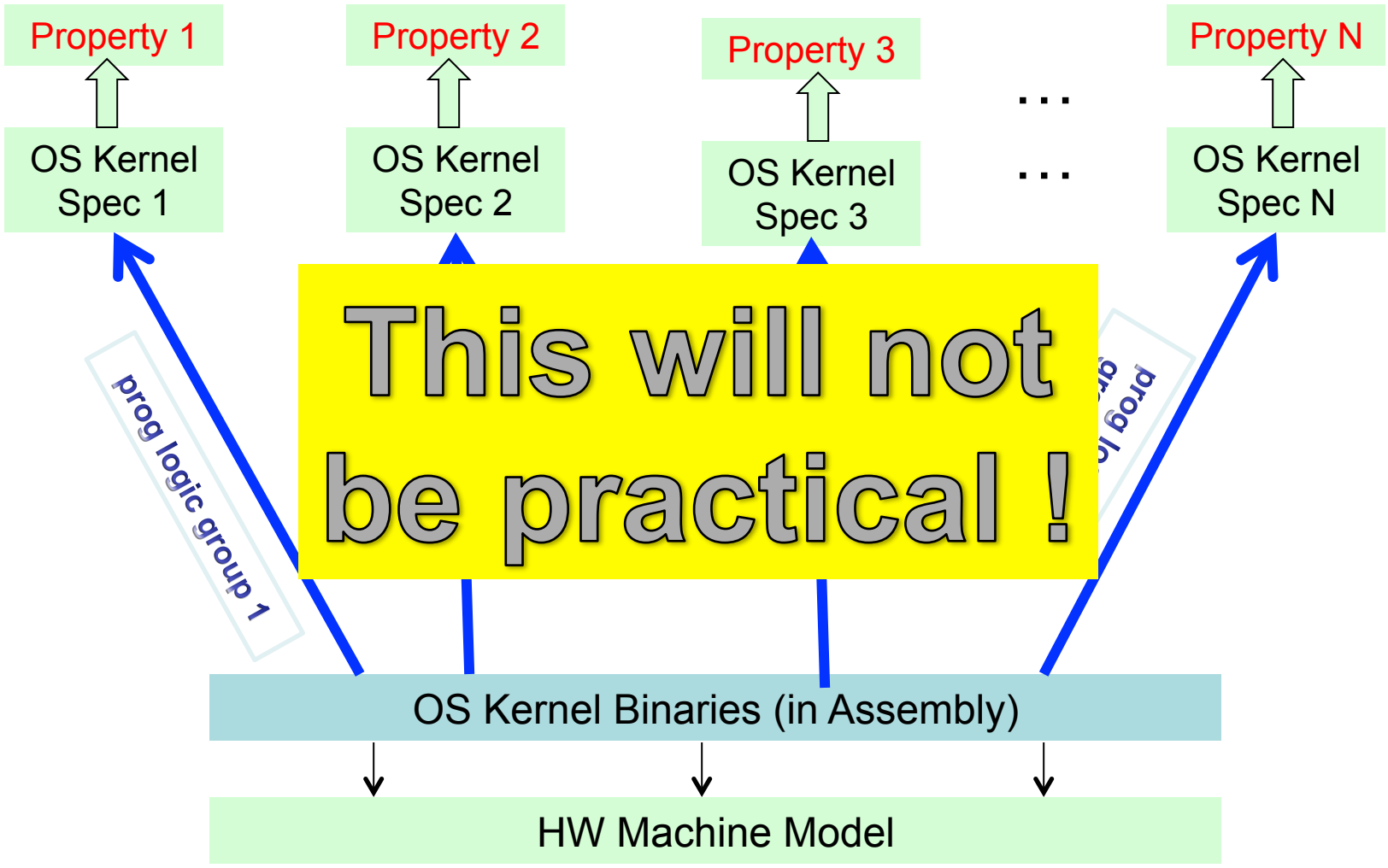
Our Contributions

- We introduce 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
  - mCertiKOS-hyper consists of 37 layers, took less than one-person-year to develop, and can boot Linux as a guest

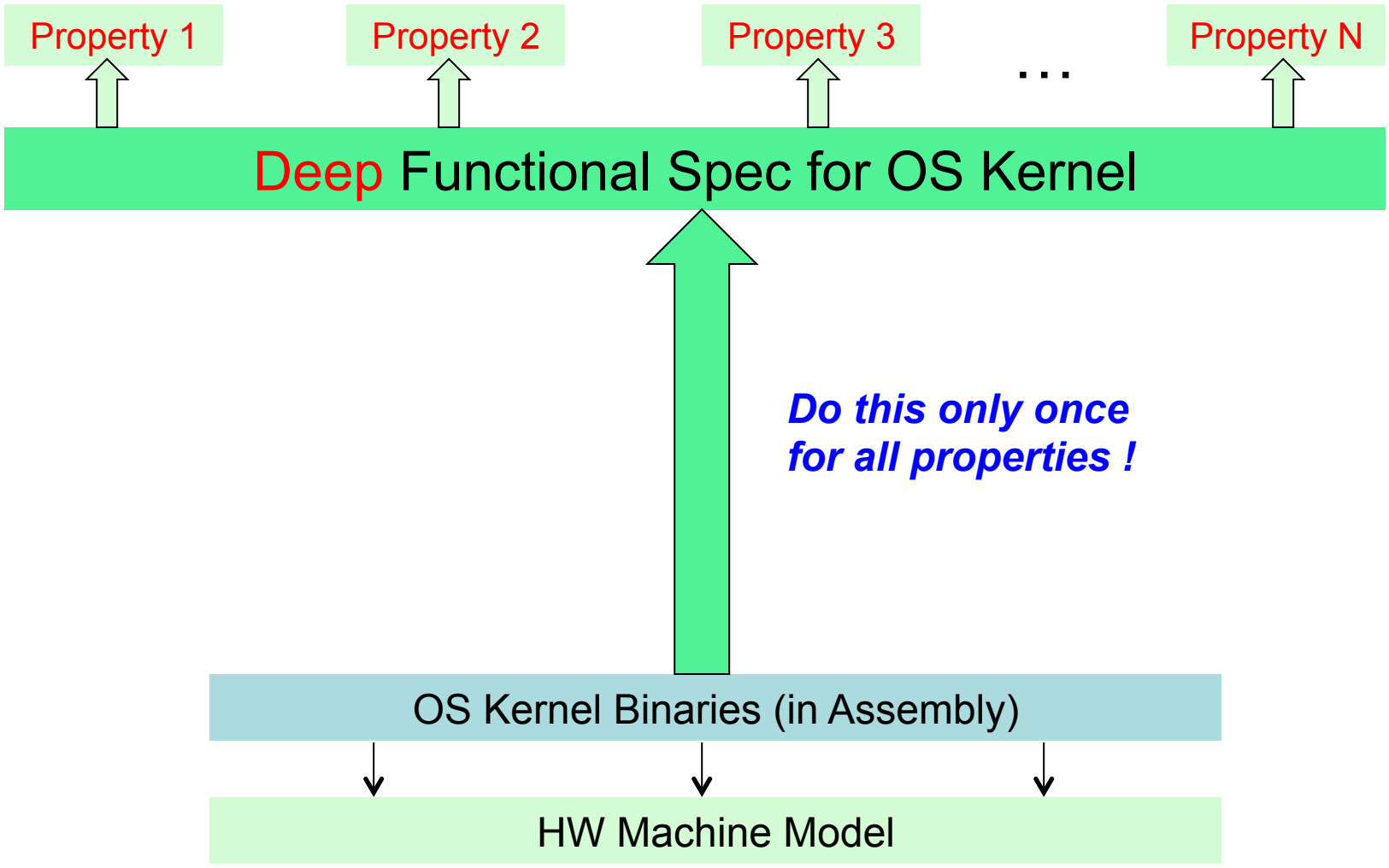
Problem Definition

- What is a certified OS kernel?
  - an OS kernel binary ***implements*** its specification?
  - what should its specification be like?
- What properties do we want to prove?
  - safety & partial correctness properties
  - total functional correctness
  - security properties (isolation, noninterference, confidentiality, integrity, availability, accountability)
  - resource usage properties (stack overflow, real time properties)
  - race-freedom, atomicity, and linearizability
  - liveness properties (wait-freedom, lock-freedom, obstruction freedom, deadlock-freedom, starvation freedom)
- How to cut down the cost of verification?

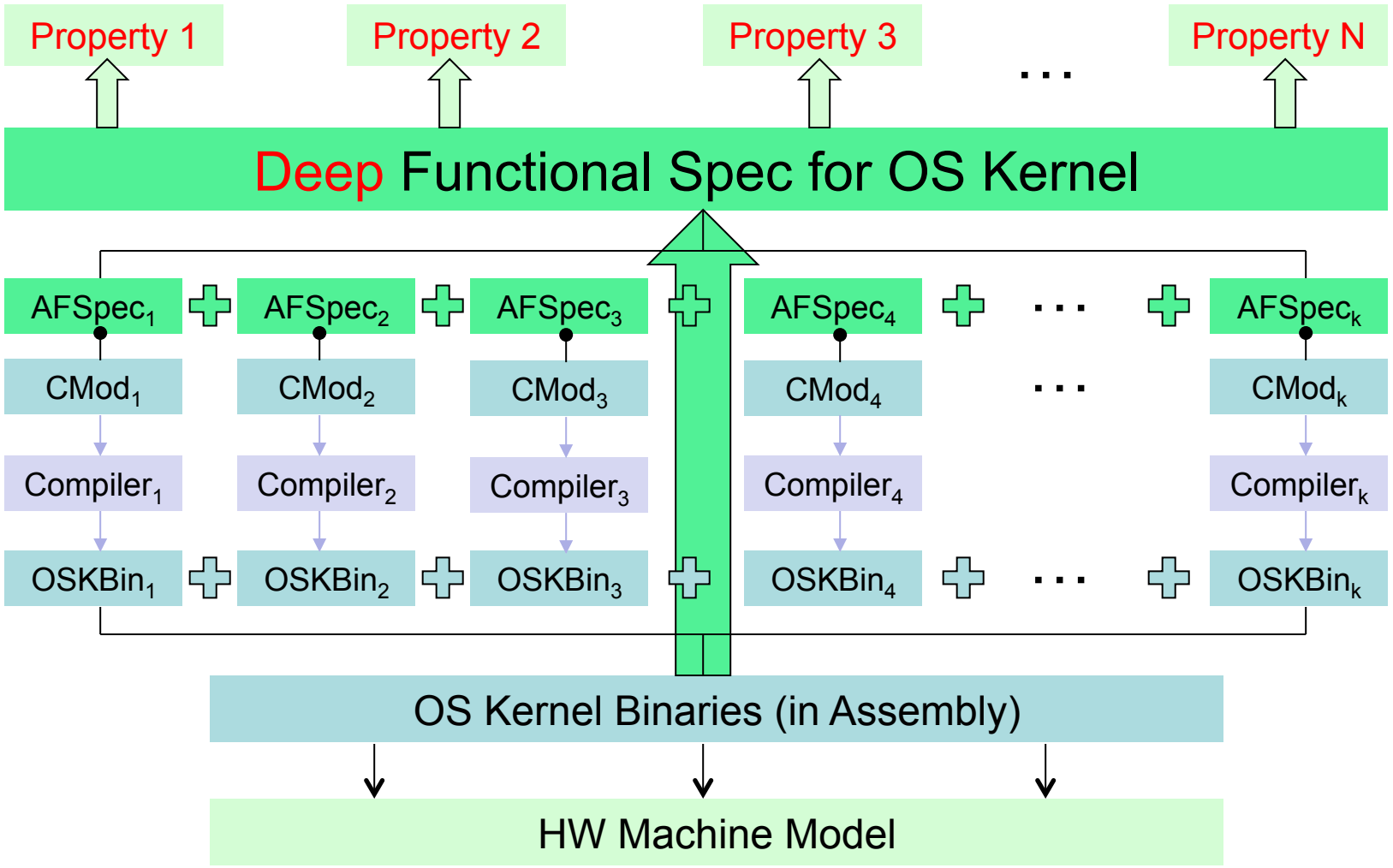
The Conventional Approach



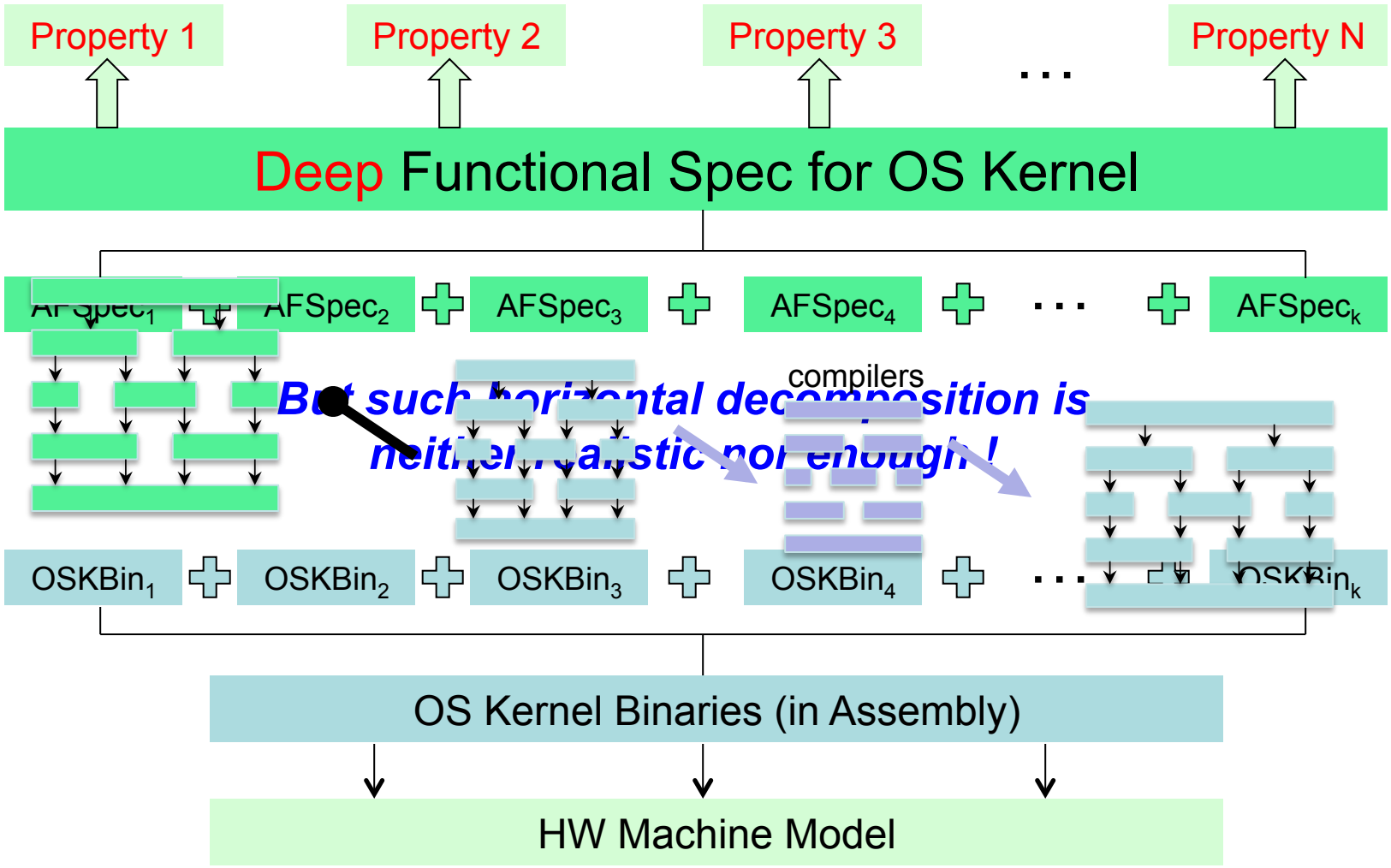
A Clean-Slate Approach?



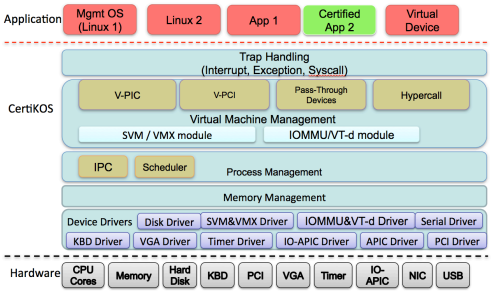
A Clean-Slate Approach?



A Clean-Slate Approach?

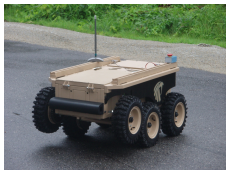


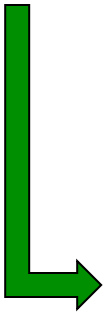
Case Study: mCertiKOS

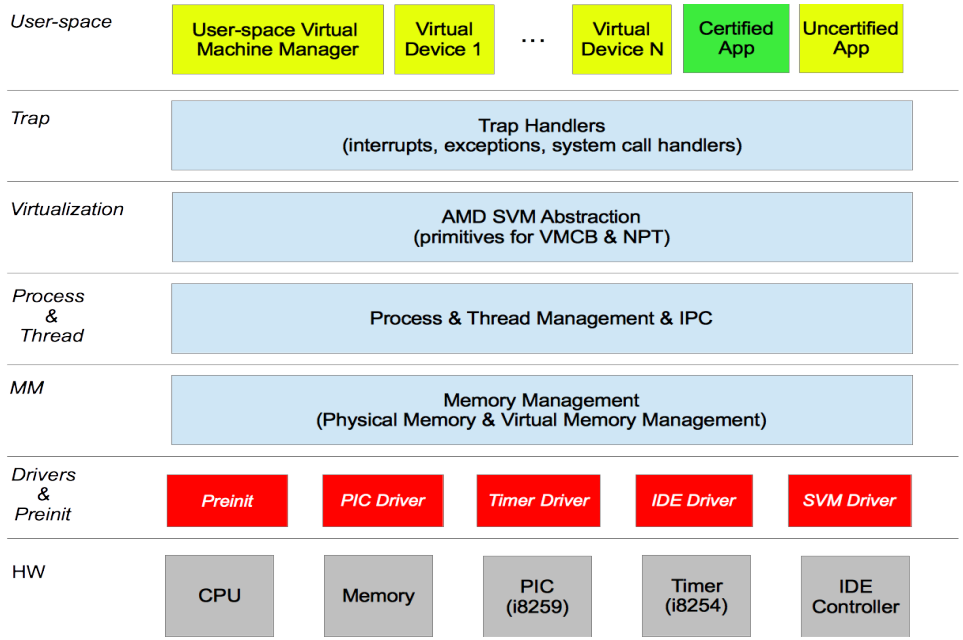


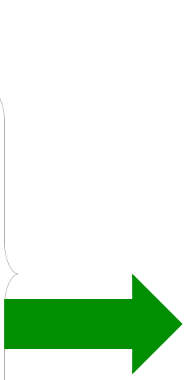
Single-core version of CertiKOS (developed under DARPA CRASH & HACMS programs), 3 kloc, can boot Linux

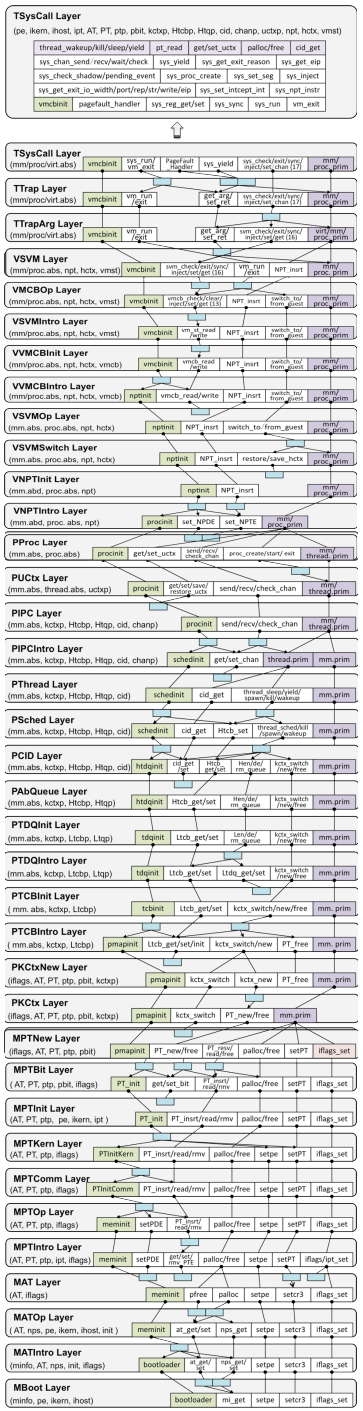
Aggressive use of abstraction over deep specs (37 layers in ClightX & LAsm)



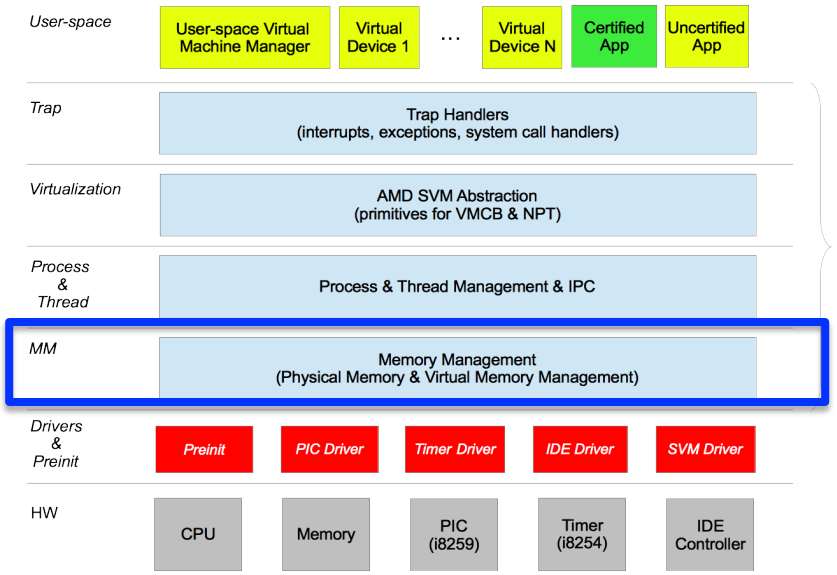


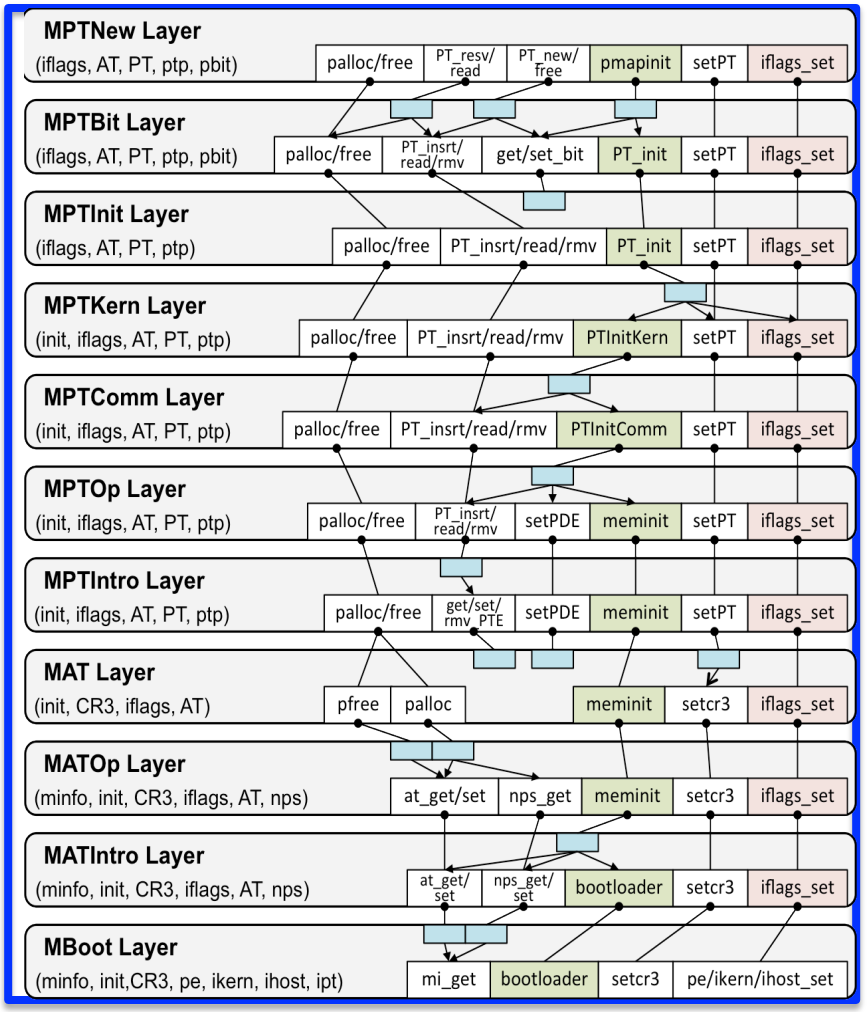






Decomposing mCertiKOS

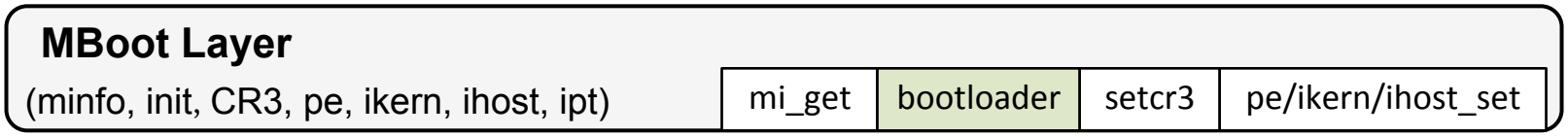




Physical Memory and Virtual Memory Management (11 Layers)

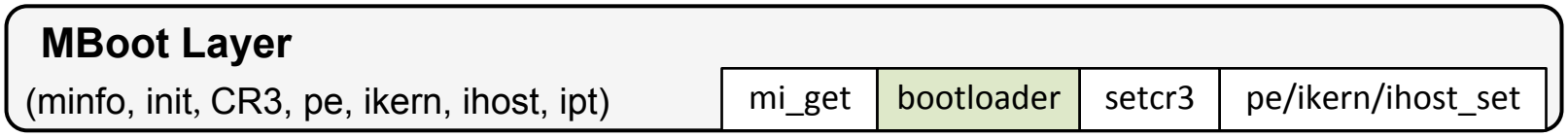
Based on the abstract machine provided by boot loader

1: MBoot Layer



- Abstract State (*abs*)
  - minfo: physical memory information table
  - init: (logical) whether bootloader/preinit has been called or not
  - CR3: abstract CR3 register (start address of page table)
  - pe: abstract CR0 register (paging is enabled or not)
  - ikern: whether it is in the kernel mode or not
  - ihost: whether it is in the host mode or not
  - ipt: (logical) whether it is using the kernel's page table

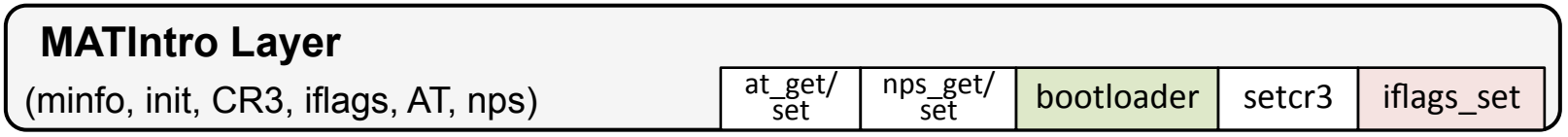
1: MBoot Layer



- Primitives
  - mi_get: read the minfo
  - setcr3: set the start address of the page table
  - pe/ikern/ihost_set: set the corresponding *abs-state bit*
  - bootloader: bootloader/preinit of mCertiKOS

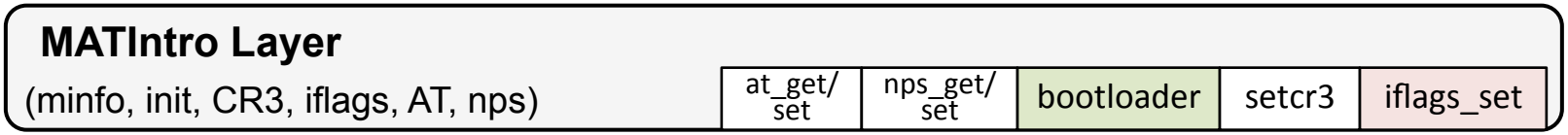
- Initialization function is marked by green

2: MATIntro Layer



- Introduce the page allocation table
- Abstract State
  - iflags: (pe, ikern, ihost, ipt)
  - AT: page allocation table
  - nps: number of physical pages
- Primitive
  - iflags_set: a set of primitives that set the value of iflags
  - at_get/set: getter and setter for AT
  - nps_get/set: getter and setter for nps

2: MATIntro Layer



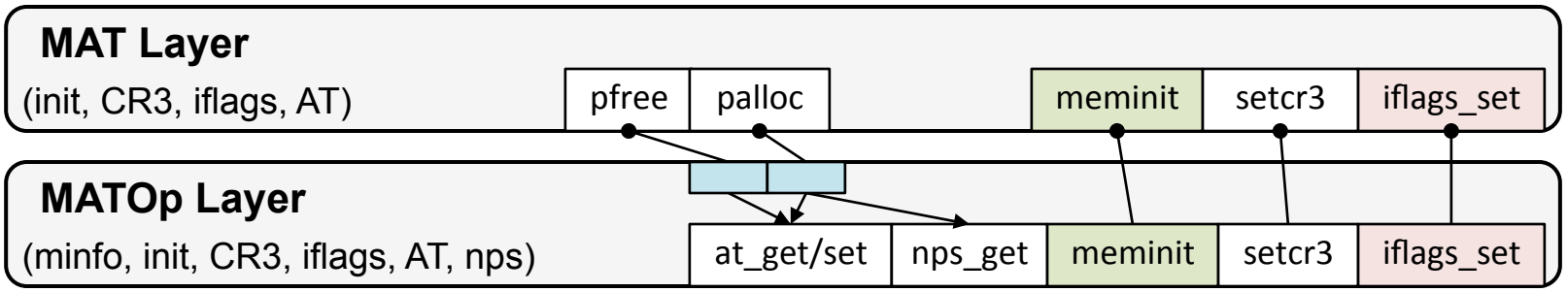
Concrete data structures in C

```
struct A {
        unsigned int isnorm;
        unsigned int allocated;
};

struct  A  AT_LOC[1048576];
```

Abstract state defined in Coq

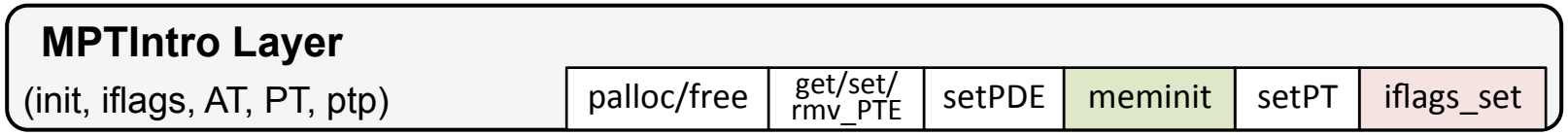
```
(** Allocation table*)
  Inductive ATType: Type :=
  | ATKern
  | ATResv
  | ATNorm.

  Inductive ATInfo :=
  | ATValid (b: bool) (t: ATType)
  | ATUndef.

Definition ATable := ZMap.t ATInfo
```

3: MATOp Layer – 4: MAT Layer



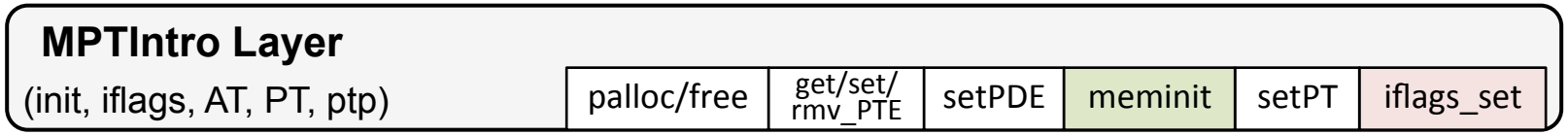
- Initialize the allocation table and provide primitives to manipulate the allocation table
- Abstract State
  - minfo and nps are hidden
- Primitive
  - meminit: initialize AT and nps from minfo
  - palloc: allocate a page in High Memory
  - pfree: free a page

5: MPTIntro Layer



- Introduce the two-level page table pool
- Abstract State
  - PT: the current page table index
  - ptp: page table pool (64 page tables)
- Primitive
  - setPDE: setter for the first level page table entry
  - get/set/rmv_PTE: getter and setter for the second level page table entry

5: MPTIntro Layer



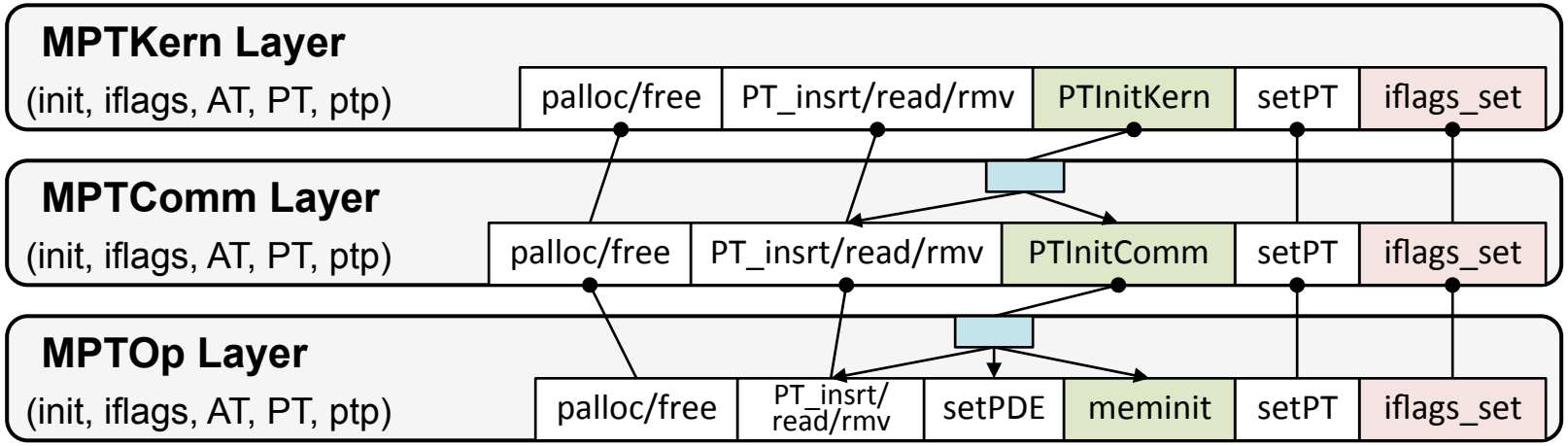
Concrete data structures in C

```
struct PTStruct {
   char * pdir[1024];
   unsigned int pt[1024][1024];
};

struct PTStruct PTPool[64];
```

Abstract state defined in Coq

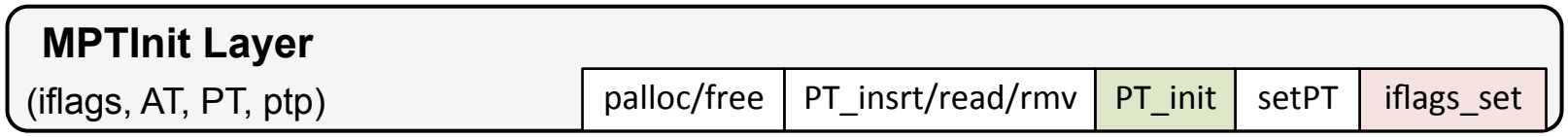
```
Inductive PTPerm: Type :=
      PTP | PTU | PTK (b: bool).

Inductive PTInfo:=
   PTValid (v: block) (p: PTPerm)
 | PTUnPresent | PTUndef.

Definition PTE := ZMap.t PTInfo.
Inductive PDTInfo :=
   PDTValid (pte: PTE) | PDTUndef.

Definition PTable := ZMap.t PDTInfo.
Definition PTPool := ZMap.t PTable.
```

6: MPTOp Layer – 8: MPTKern Layer



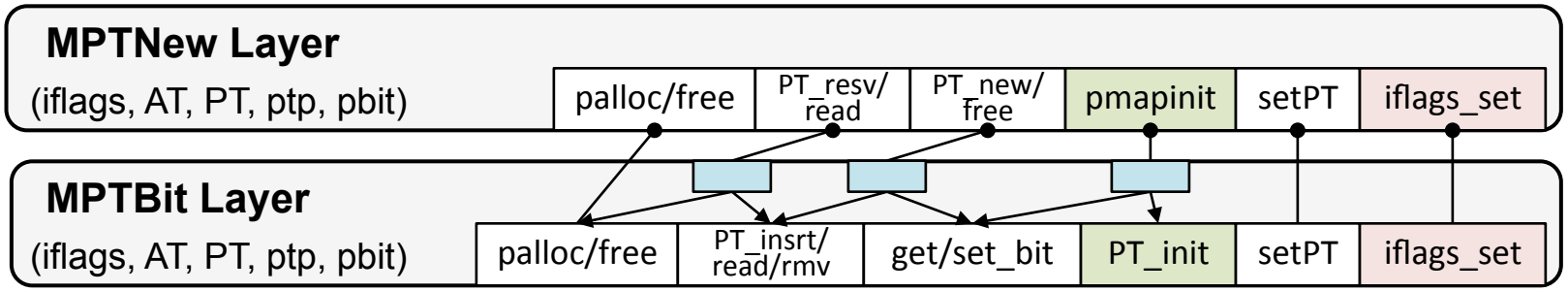
- Initialize the page table (pt) pool
- Primitive
  - PT_inst/read/rmv: insert/read/remove a map to/from a pt
  - PTinitComm: initialize the High Memory part of all the pts
  - PTinitKern: initialize the Low Memory part of the kernel's pt (with the index 0)

9: MPTInit Layer



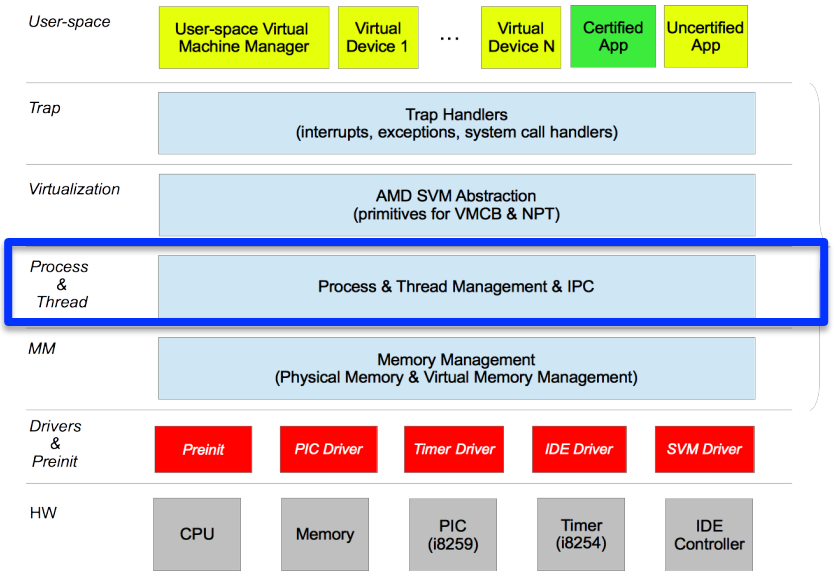
- Enable the paging mechanism
- Primitive
  - PT_init:
    - Initialize the kernel's pt (call PTInitKern)
    - Set the start address of kernel's pt to CR3 (call set_CR3)
    - Enable paging (call pe_set)

10: MPTBit Layer – 11: MPTNew Layer

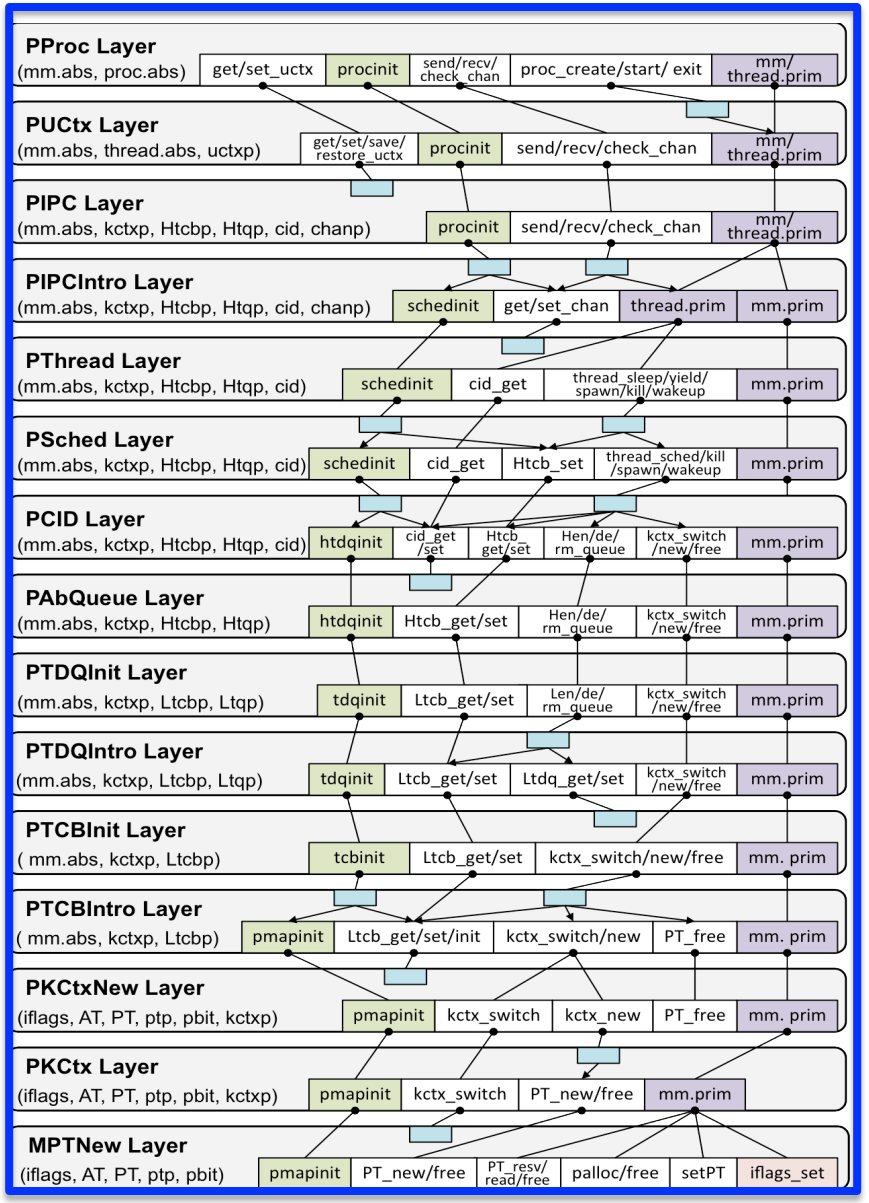


- Introduce the bit map for page table pool
- Abstract State
  - pbit: bit map for ptp
- Primitive
  - get/set_bit: getter and setter for the bit map
  - PT_new/free: allocate/free a pt from ptp
  - PT_resv: allocate a page and insert a map into pt
  - pmap_init: enable paging and reserve the 0-th bit in pbit

Decomposing mCertiKOS (cont'd)

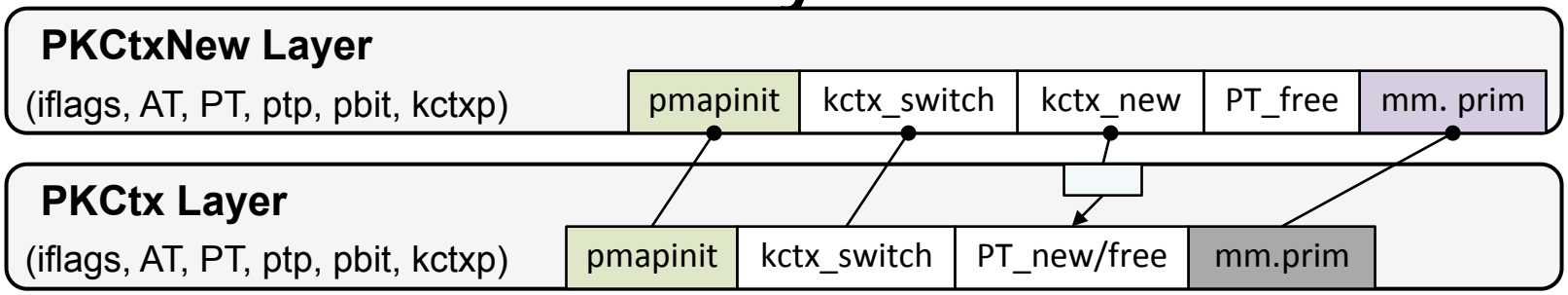


C Kernel



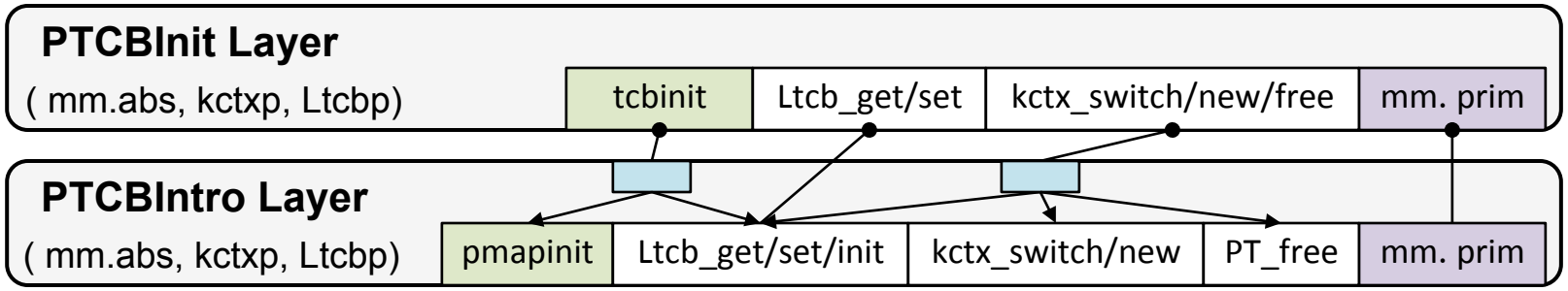
Thread and Process Management (14 Layers)

12: PKCtx Layer – 13: PKCtxNew Layer



- Introduce the kernel context pool
- Abstract State
  - kctxp: kernel context pool (using ptp as bit map)
- Primitive
  - kctx_switch: kernel context switch (written in assembly)
  - kctx_new: allocate a pt and kernel context (kctx) from ptp
  - mm.prim: primitives provided by memory

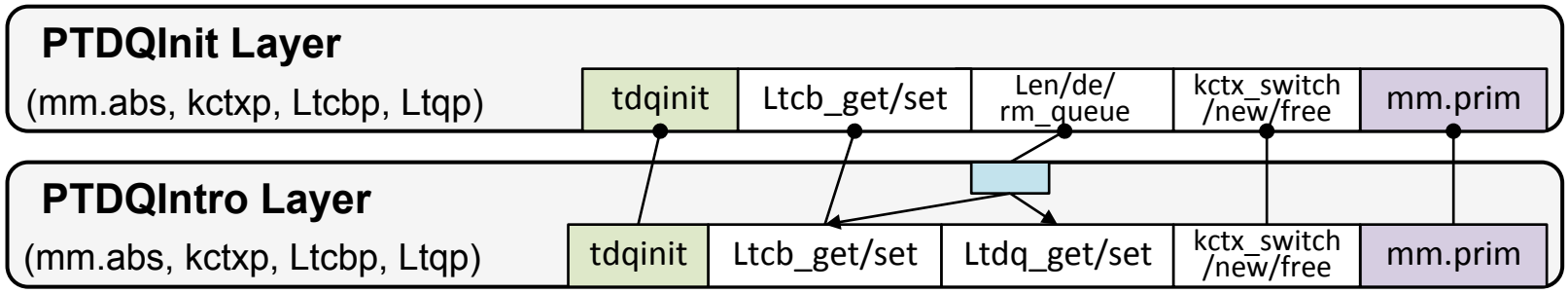
14: PTCBintro Layer – 15: PTCBInit Layer



- Introduce and initialize the thread control blocks (tcb) pool
- Abstract State
  - Ltcbp: low-level tcb pool (using ptp as bit map)
  - mm.abs: *abs* provided by memory management
- Primitive
  - Ltcb_get/set/init: getter and setter for Ltcbp
  - kctx_free: free a pt, kctx and tcb from ptp

    enable paging and initialize ptp and Ltcbp

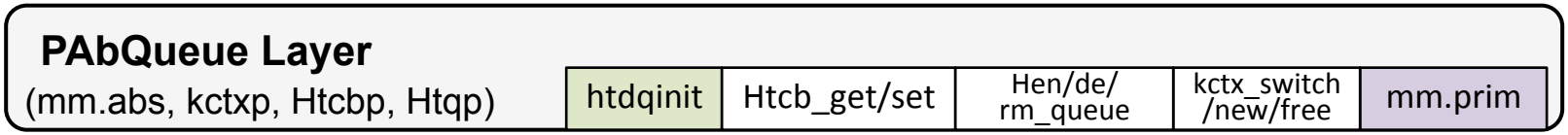
16: PTDQintro Layer – 17: PTDQInit Layer



- Introduce and initialize the thread queue (td) pool
- Abstract State
  - Ltqp: low-level td pool (using ptp as bit map)
- Primitive
  - Ltdq_get/set: getter and setter for Ltdqp
  - Len/de/rm_queue: enqueue, dequeue and remove a thread from the low-level thread queue

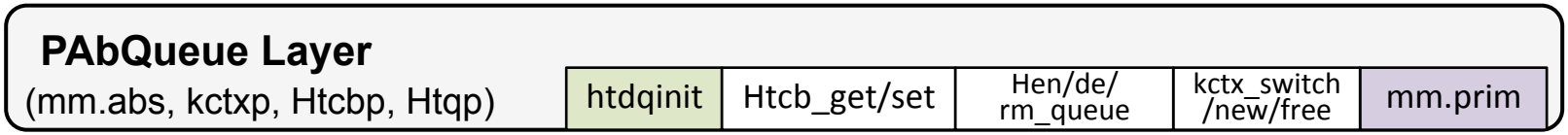
    enable paging and initialize ptp, Ltcbp and Ltdp

18: PAbQueue Layer



- Introduce the high-level (abstract) tcb and td
- Abstract State
  - Htcbp: high-level tcb (defined as Coq inductive type) pool
  - Htqp: high-level td (defined as Coq list) pool
- Primitive
  - Htcb_get/set: getter and setter for Htcb
  - Hen/de/rm_queue: enqueue, dequeue and remove a thread from the high-level thread queue
  - htdqinit: enable paging and initialize ptp, Htcbp and Htdp

18: PAbQueue Layer



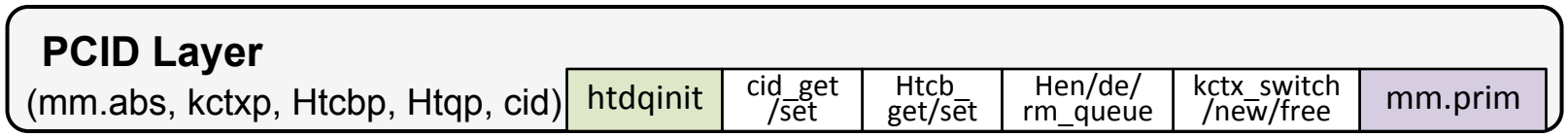
Low-level tcb and td defined in C

```
typedef enum {
  TD_READY, TD_RUN, TD_SLEEP, TD_DEAD
} td_state;

struct tcb {
  td_state tstate;
  struct tcb * prev, * next;
};

struct tdqueue {
  struct tcb *head, *tail;
};

static struct tcb tcb_pool[num_proc];
static struct tdqueue ready_queue;
static struct tdqueue sleep_queue [num_chan];
```

High-level tcb and td defined in Coq

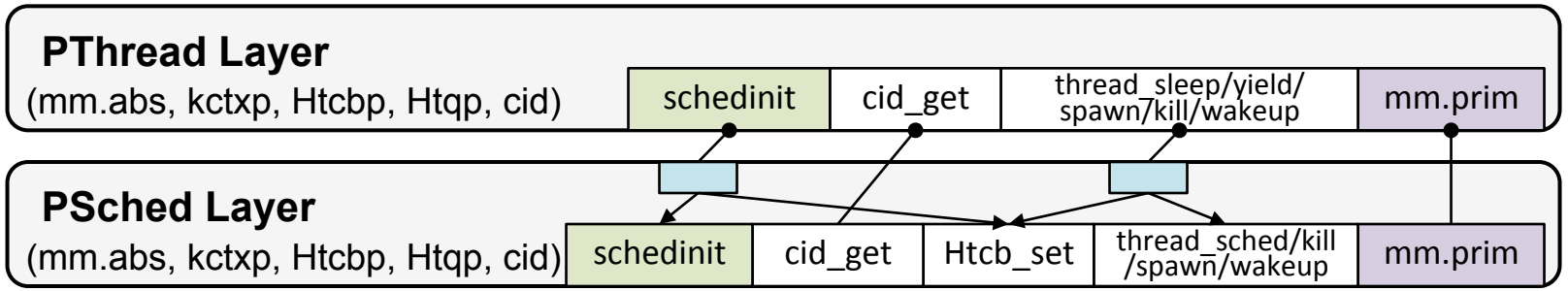
```
Inductive td_state :=
|TD_READY |TD_RUN |TD_SLEEP |TD_DEAD.

Definition tcb := td_state.
Definition tdqueue := List Z.

tcb_pool : Z -> option tcb.
ready_queue: tdqueue.
sleep_queues : Z-> option tdqueue.
```

19: PCID Layer



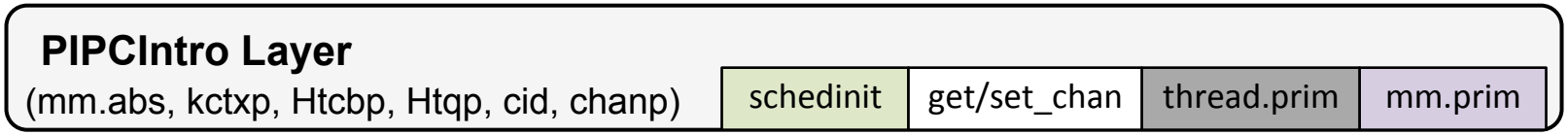
- Introduce the current thread id
- Abstract State
  - cid: current thread id
- Primitive
  - cid_get/set: getter and setter for cid

20: PSched Layer - 21: PThread Layer



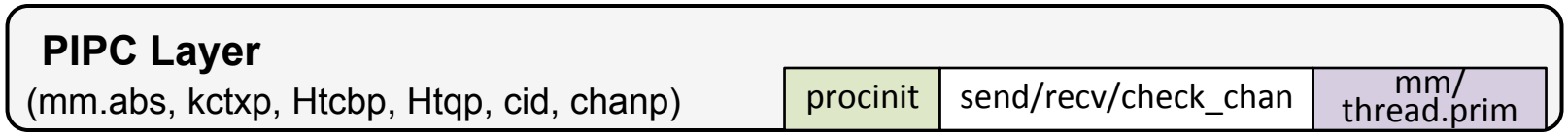
- Introduce the primitives for thread management
- Primitive
  - thread_sched: thread scheduler (non-preemptive)
  - thread_spawn/kill: spawn/kill a thread. Including allocate/free the corresponding pt and tcb, and modify the Htdp
  - thread_wakeup: wakeup a sleeping thread
  - thread_sleep: sleep for a resource (such as a channel)
  - thread_yield: yield to the first ready thread

22: PIPCIntro Layer



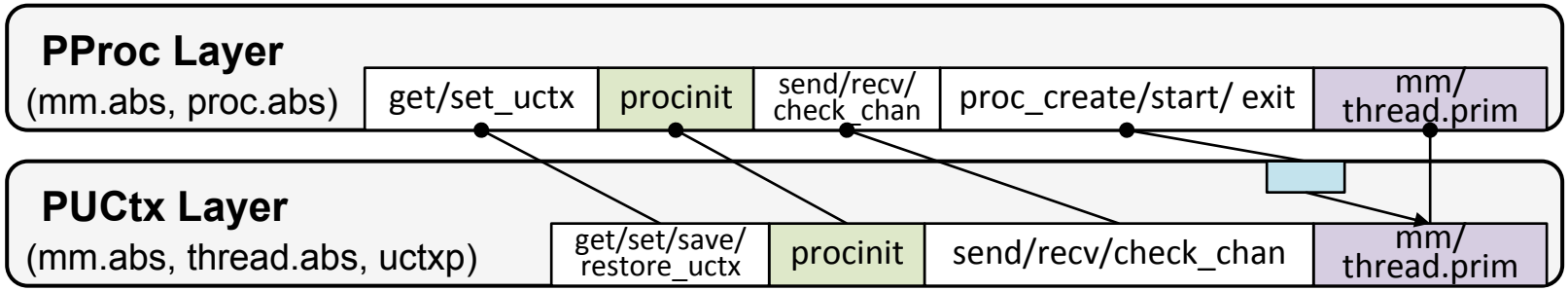
- Introduce the inter process communication channel pool
- Abstract State
  - chanp: channel pool for inter process communication
- Primitive
  - get/set_chan: getter and setter for chanp
  - thread.prim: primitives provided by thread management

23: PIPC Layer



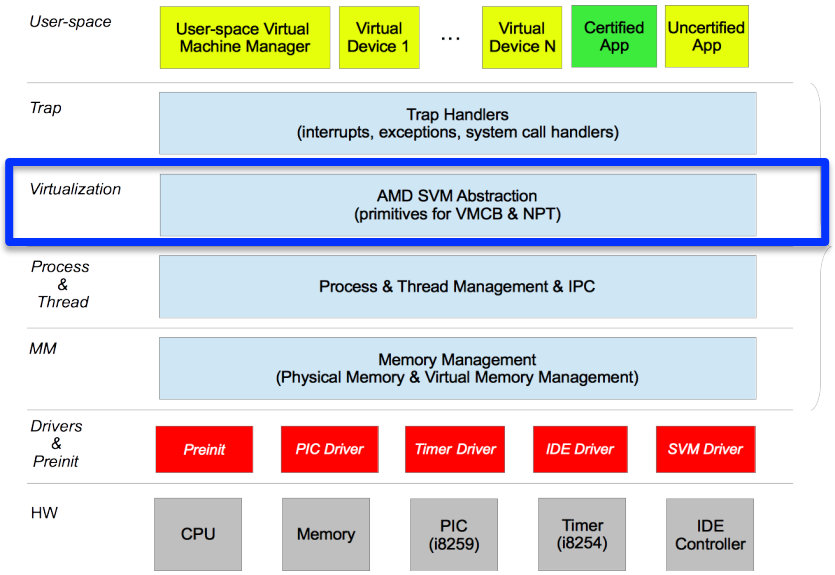
- Intitialize the inter process communication channel pool
- Primitive
  - send_chan: send the message to a channel
  - check_chan: check whether its channel is full or not
  - recv_chan: receive the message from its own channel, and wakeup the first thread sleeping on the channel
  - procinit: enable paging and initialize ptp, Ltcbp, Ltdp, cid and chanp

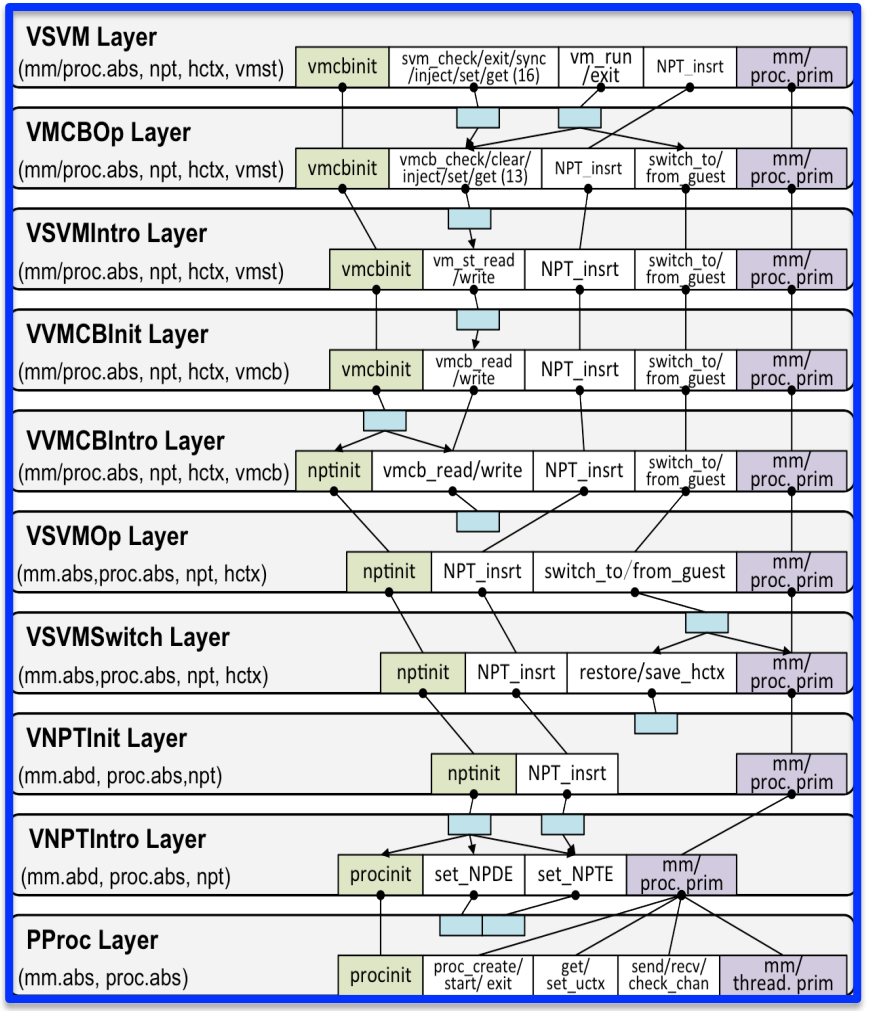
24: PUCtx Layer – 25: PProc Layer



- Introduce the user process context (uctx) pool
- Abstract State
  - uctxp: user process context pool (using ptp as bit map)
  - thread/proc.abs: *abs* provided by thread/proc management
- Primitive
  - get/set/save/restore_uctx: getter and setter for uctxp
  - proc_create: create a user process and initialize the uctx
  - proc_start/exit: start/exit a user process

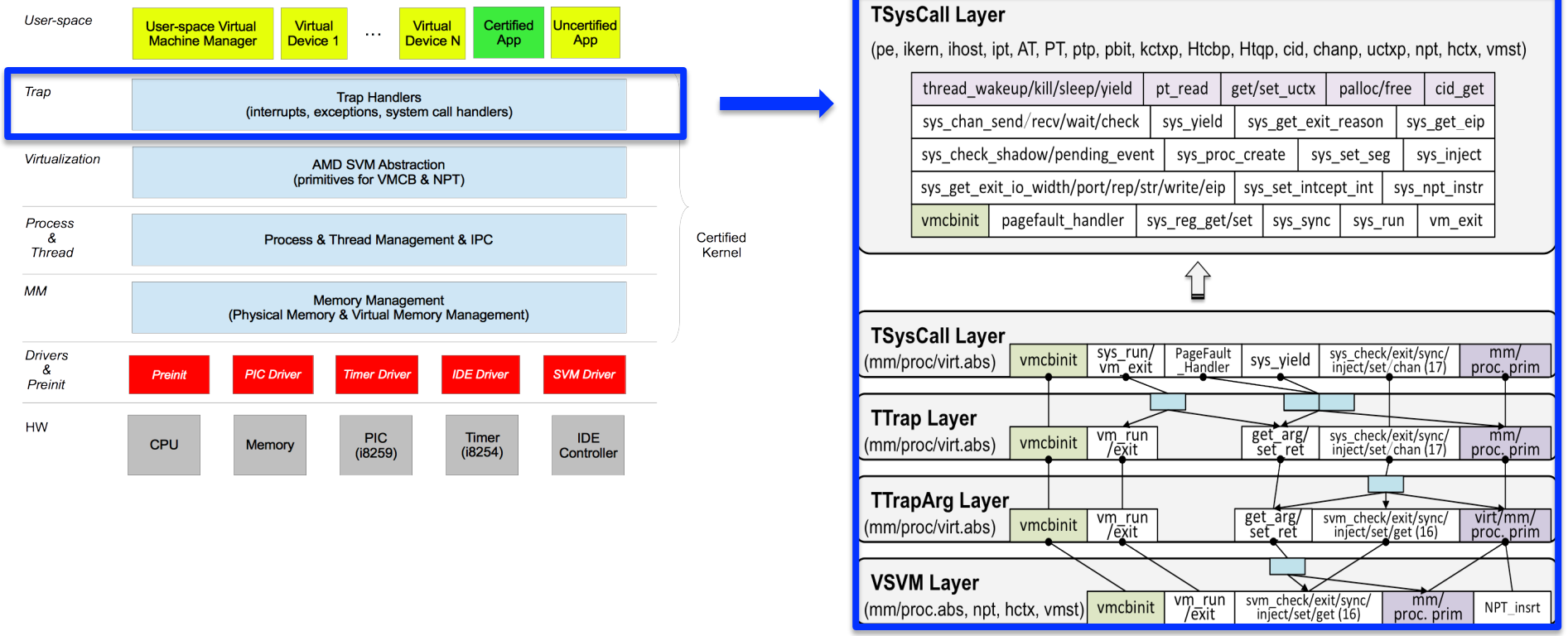
Decomposing mCertiKOS (cont'd)





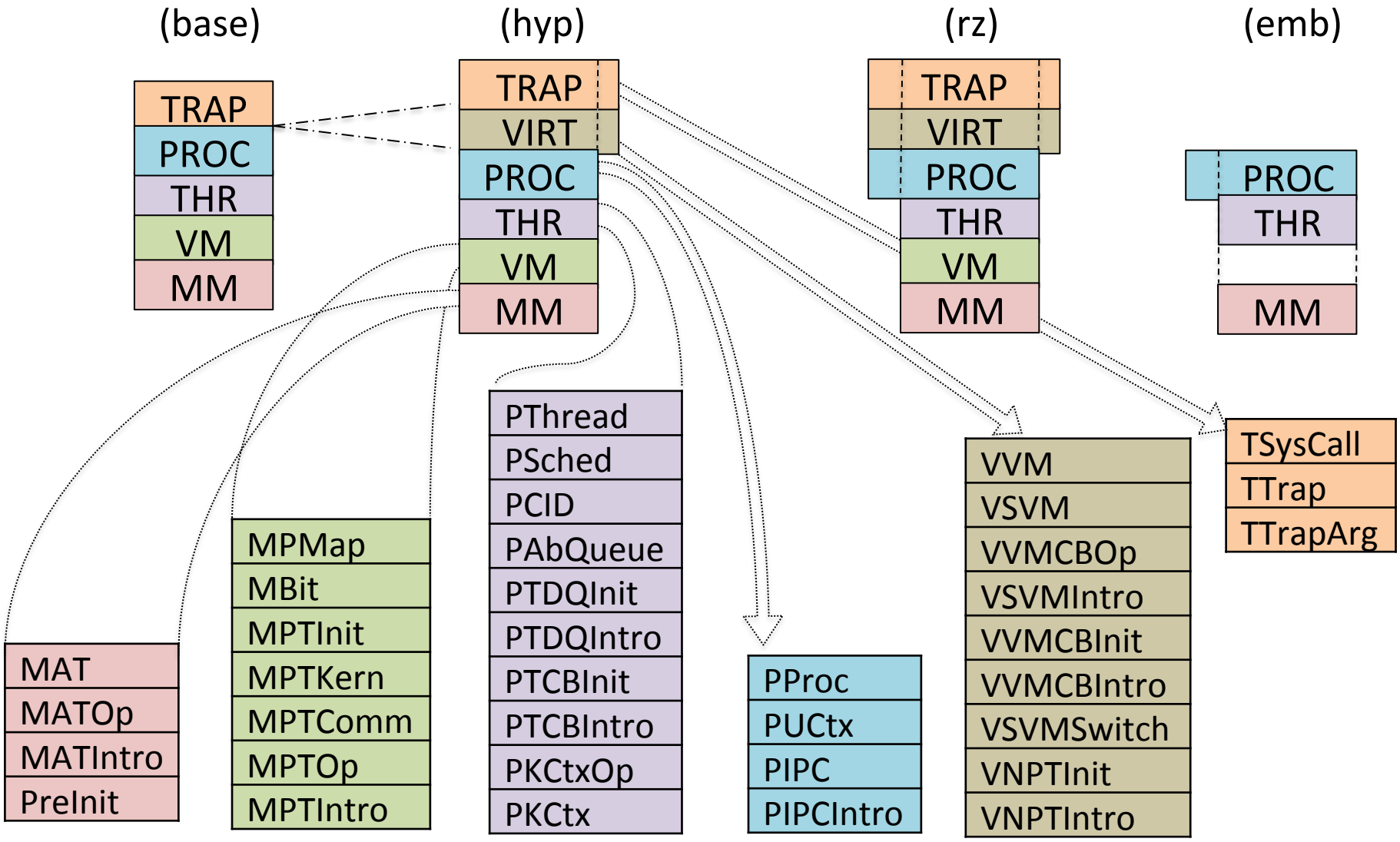
Virtualization Support (9 Layers)

Decomposing mCertiKOS (cont'd)

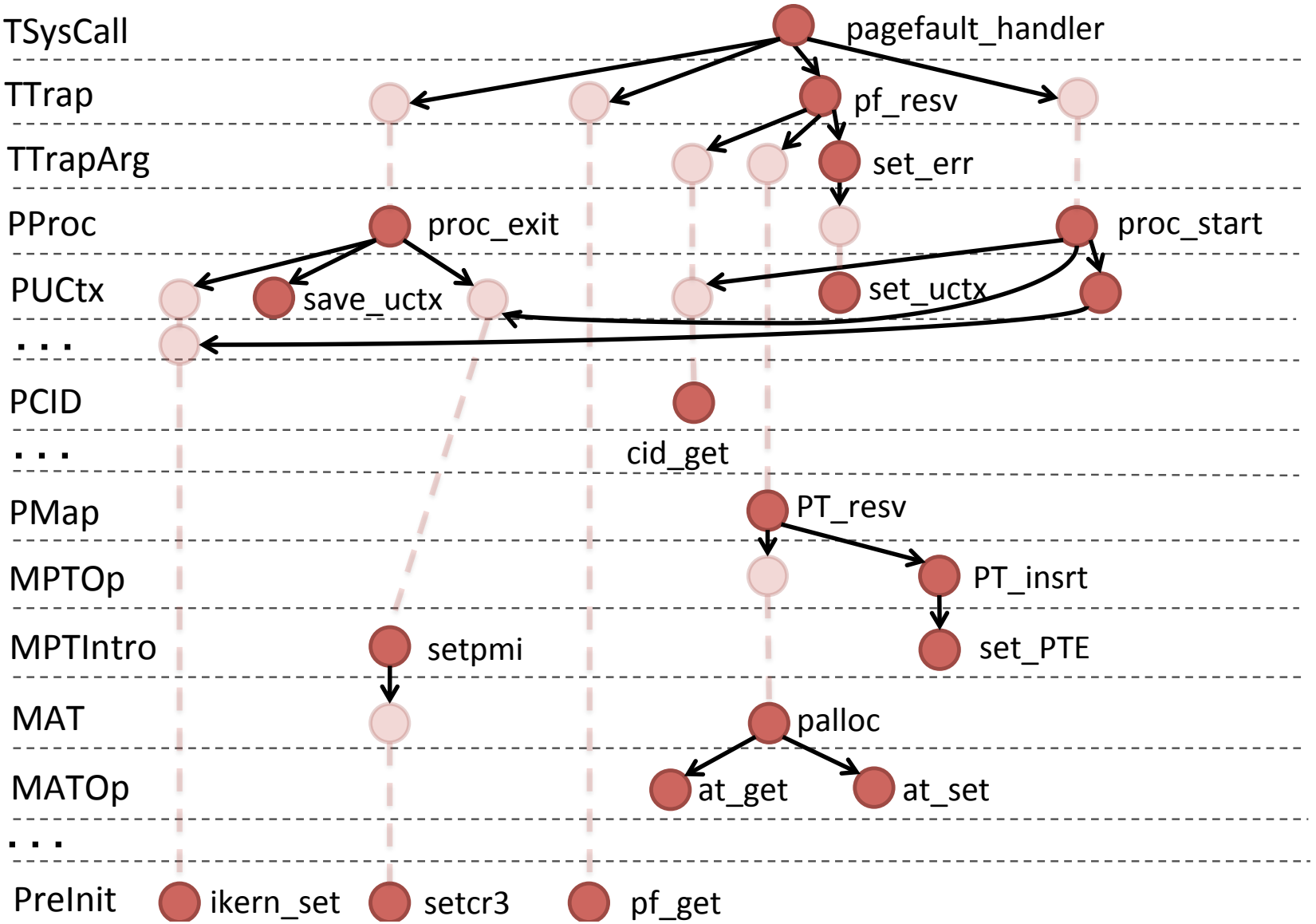


Syscall and Trap Handlers (3 Layers)

Variants of mCertiKOS Kernels



Example: Page Fault Handler



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 (DeepSpec vs. C & Asm)
- New certified programming tools
- New certified modeling & arch. description languages
- We verify all interesting properties (not just safety / partial correctness properties)

The CertiKOS Worldview

- Objects: coinductive values (e.g., a SW module, a thread, a concurrent object, a HW component)
- Equality on objects: ***simulation w. observable functions***

- Each object always has a declarative specification (a coinductively defined mathematical object)
  - Effects & interference are encapsulated within each object

- An abstraction layer: a collection of layers and objects
- The cyber world is always built as many layers of abstraction
- Resource management is built as libraries (no need for built-in GC requirement)

The CertiKOS Worldview (cont'd)

How to support formal reasoning about "layers of layers of abstraction?"

- A "clean-slate" certified world:
  - Everything here is formally specified and verified
  - We only use compositional features
  - We follow the refinement-based approach
  - No need to pay for any ugly legacy features
  - Some higher-order features are not compositional
- Legacy worlds ("layers") for backward compatibility:
  - Placed & encapsulated in various "virtual" environments (e.g., VMs, containers, processes)