Deep Specifications and Certified Abstraction Layers

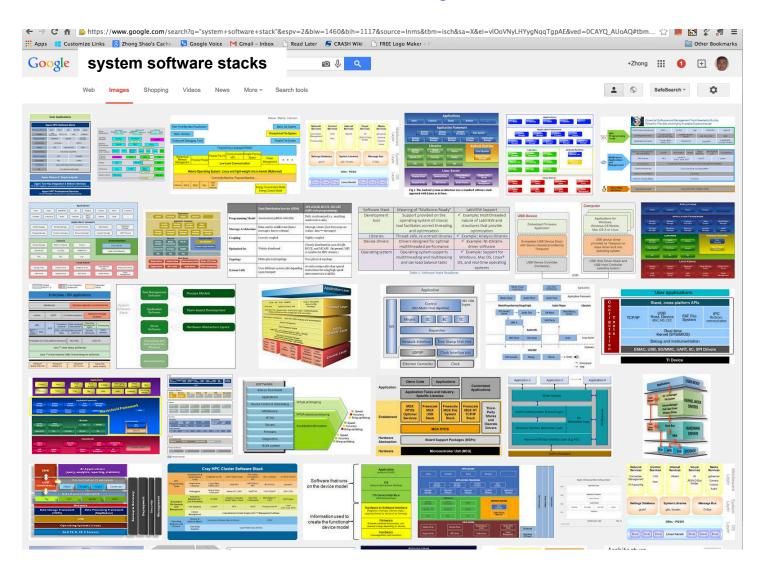
Ronghui Gu Jérémie Koenig Tahina Ramananandro Zhong Shao Newman Wu Shu-Chun Weng Haozhong Zhang¹ Yu Guo¹

Yale University ¹University of Science and Technology of China

January 17, 2015

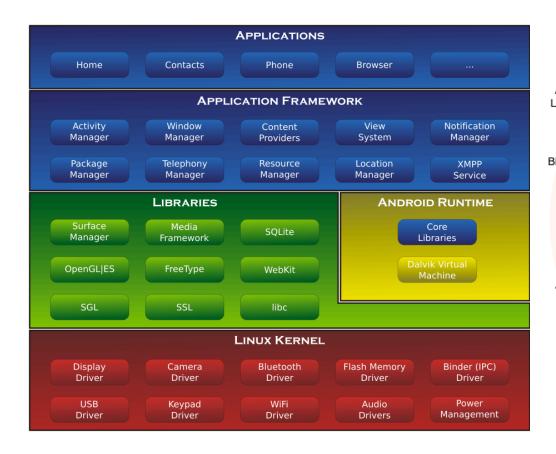
http://flint.cs.yale.edu

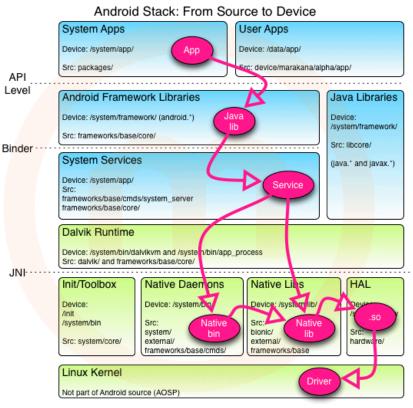
How to build reliable & secure **system software stacks**?



Android architecture & system stack

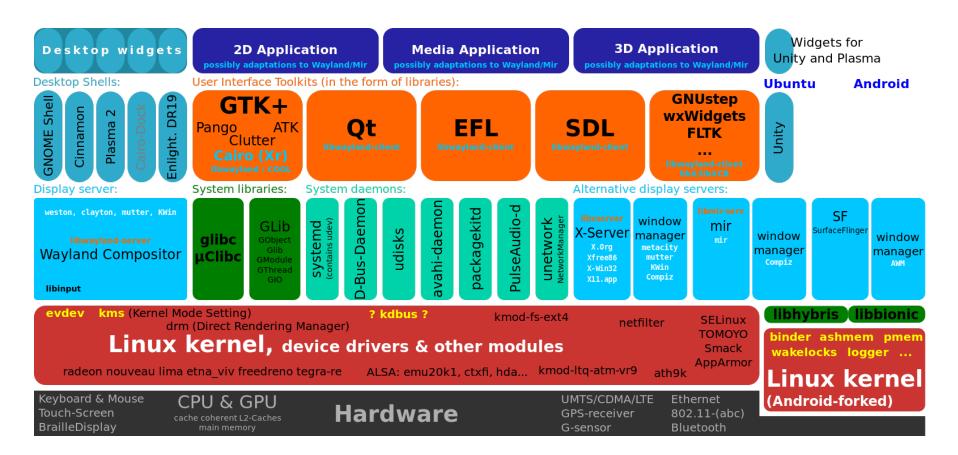
From https://thenewcircle.com/s/post/1031/android_stack_source_to_device & http://en.wikipedia.org/wiki/Android_(operating_system)





Visible software components of the Linux desktop stack

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



Software stack for HPC clusters

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

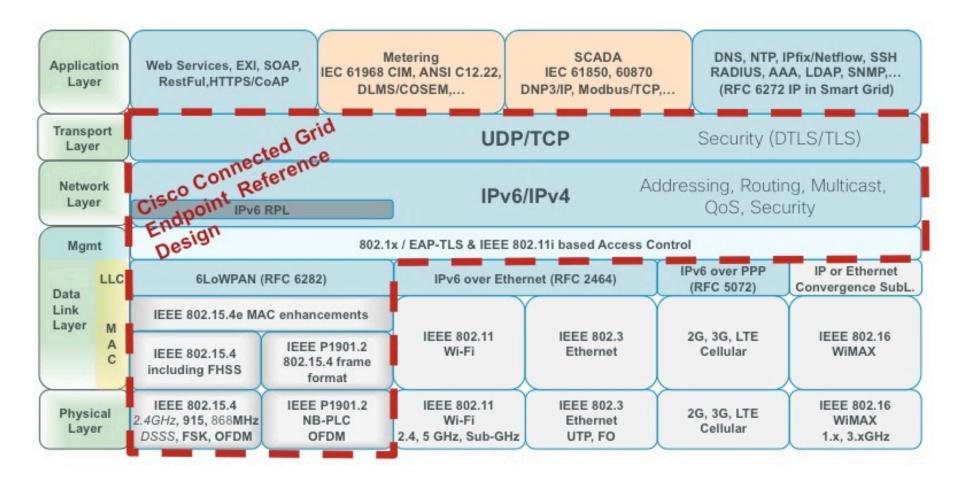


Essential Software and Management Tools Needed to Build a Powerful, Flexible, and Highly Available Supercomputer.

	HPC Programming Tools	Performance Monitoring	HPCC	Perfctr IOR		i.	PAPI/IPM	netperf
XX		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	моав	Altair PBS Pro	IBM Platforr LSF	ⁿ Torque/Maui
		File System	NFS	Local FS (ext3, ext4, XI	FS)	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	mt Cray ACE					
\bigcirc	Operating Systems	Operating System	Linux (Red Hat, CentOS, SUSE)					

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

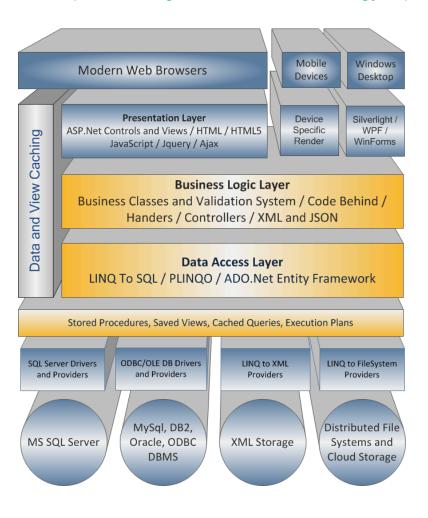
From https://solutionpartner.cisco.com/web/cegd/overview



Apollo Mobile Communication Stack http://www.layer2connections.com/apollo_clients.html

Web Application Development Stack From http://www.brightware.co.uk/Technology.aspx



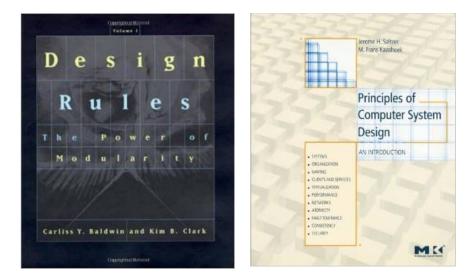


Motivation (cont'd)

- 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 *Hoarestyle judgment* between the impl. & the spec.

In the System world:

(abstraction in the large)

Mostly informal & language-

Something magical going on ... What is it?

between the impl. a the spec

Problems

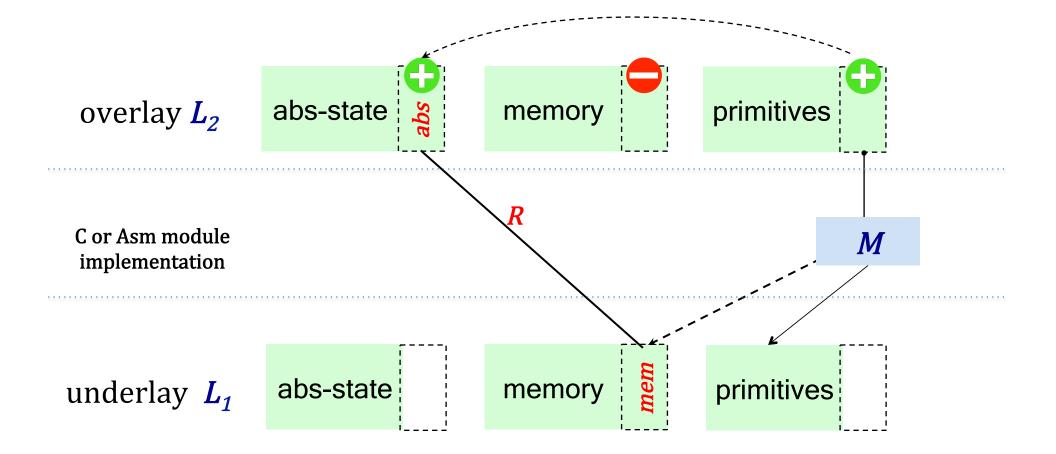
- What is an *abstraction layer*?
- How to formally **specify** an abstraction layer?
- How to *program*, *verify*, and *compile* each layer?
- How to **compose** abstraction layers?
- How to apply certified abstraction layers to build reliable and secure system software?

Our Contributions



- We introduce deep specification and present a languagebased 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 oneperson-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];
};
```

Top-level page tables

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

C

abstract layer spec

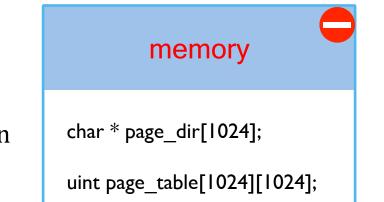
abstract state

PMap := ZMap.t PTEInfo(* vaddr \rightarrow (paddr, perm) *)

Invariants: kernel page table is a direct map; user parts are isolated

abstract primitives (Coq functions)

Function page_table_init = ... Function page_table_insert =... Function page_table_rmv = ... Function page_table_read = ...



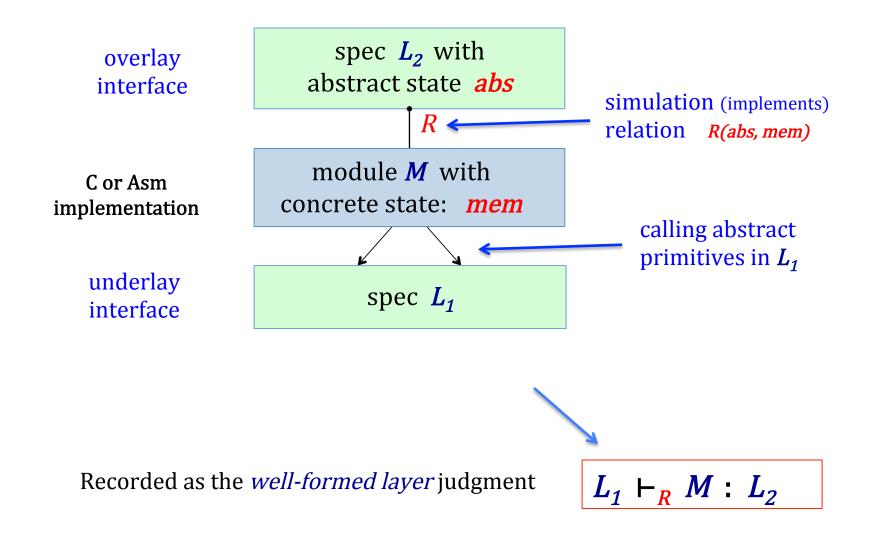
C functions

int page_table_init() { ... }
int page_table_insert { ... }
int page_table_rmv() { ... }
int page_table_read() { ... }

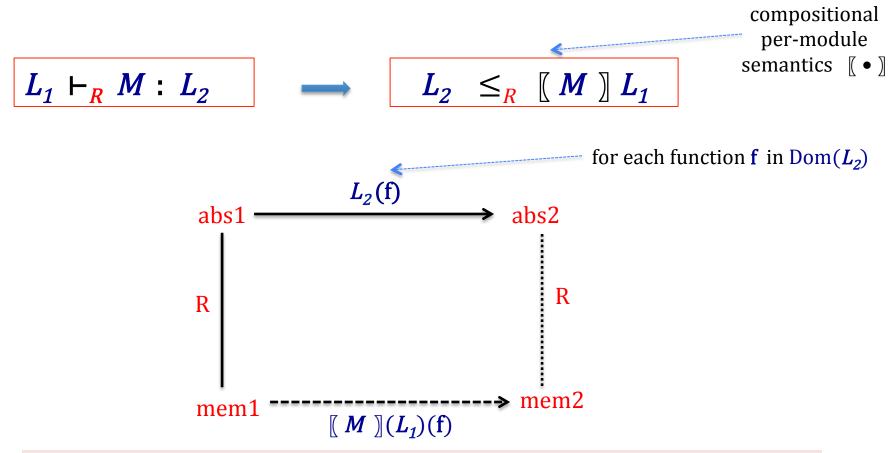
concrete C implementation

Formalizing Abstraction Layers

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

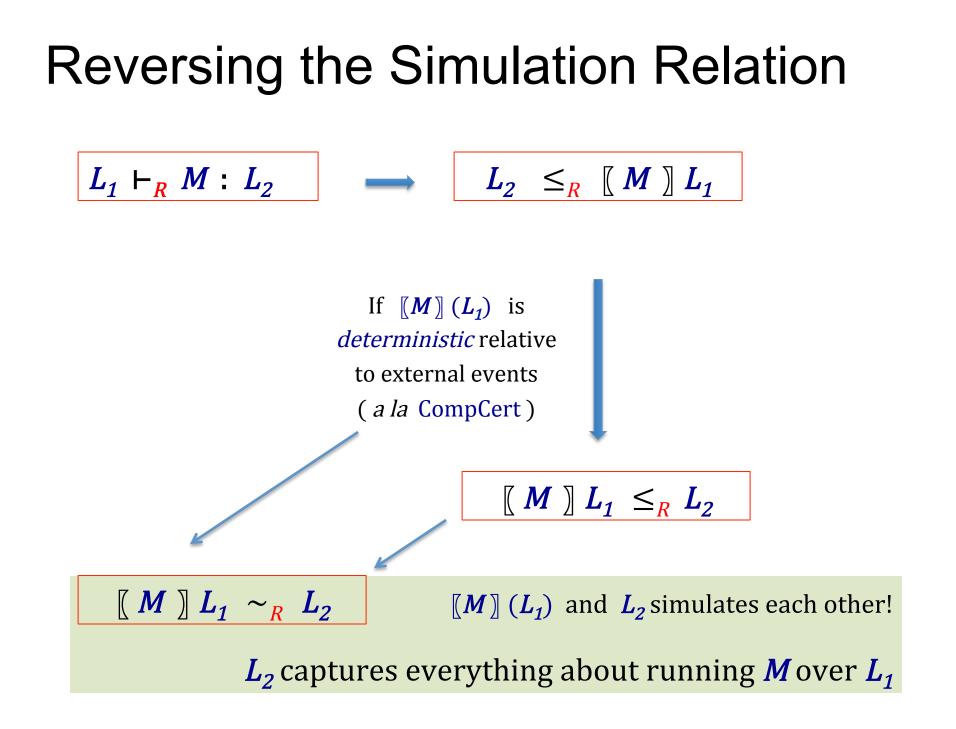


The Simulation Relation



Forward Simulation:

- Whenever $L_2(f)$ takes abs1 to abs2 in one step, and R(abs1, mem1) holds,
- then there exists mem2 such that [[M](L₁)(f) takes mem1 to mem2 in zero or more steps , and R(abs2, mem2) also holds.



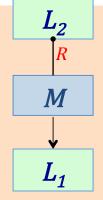
Deep Specification

 $\llbracket M \rrbracket L_1 \sim_R L_2$ $\llbracket M \rrbracket (L_1)$ 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 , $\left[P \bigoplus M\right] (L_1)$ and $\left[P\right] (L_2)$ 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 , $[\![P \bigoplus M]\!](L_1)$ 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!

 L_2

R

M

 L_1

• Any property about *M* can be proved using *L* alone.

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

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

Problem w. Shallow Specs

C or Asm module

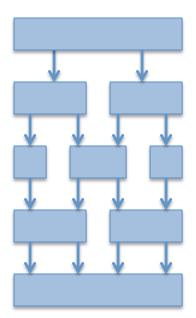


shallow spec A

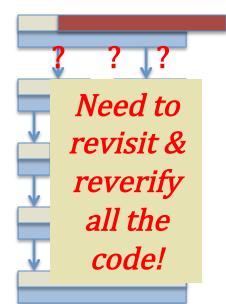


shallow spec B

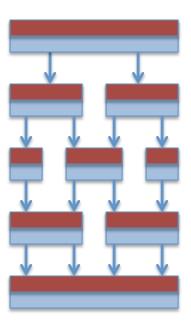
C & Asm Module Implementation



C & Asm Modules w. Shallow Spec A



Want to prove another spec B ?



Shallow vs. Deep Specifications

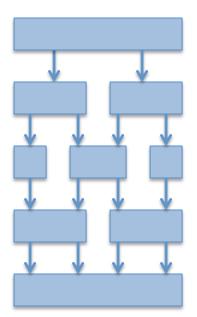
C or Asm module



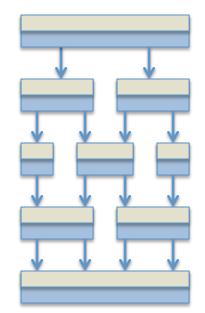
shallow spec



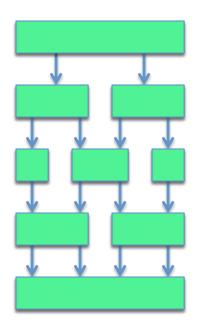
C & Asm Module Implementation



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:

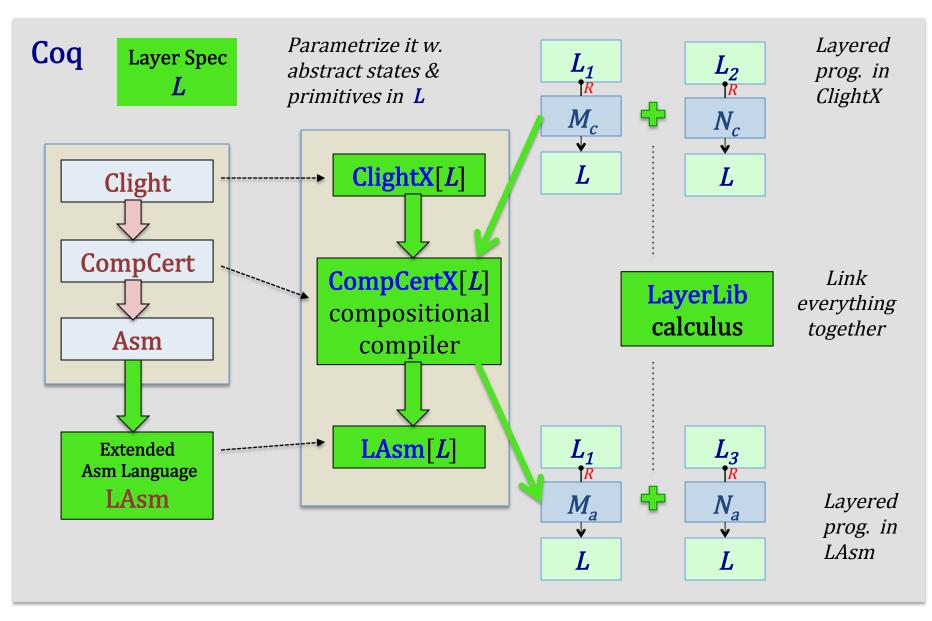
- Implementation done in C or assembly or ...
- Specification done in richer logic (e.g., Coq)
- Need to mix both and also simulation proofs
- Need to compile C layers into assembly layers
- Need to compose different layers

Our Contributions

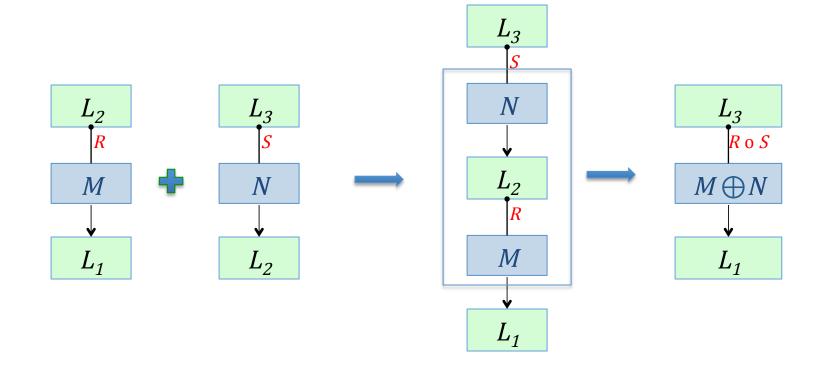


- We introduce deep specification and present a languagebased formalization of certified abstraction layer
- We developed new languages & tools in Coq
 - A formal layer calculus for composing certified layers
 - ClightX for writing certified layers in a C-like language
 - LAsm for writing certified layers 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 oneperson-year to develop, and can boot Linux as a guest

What We Have Done

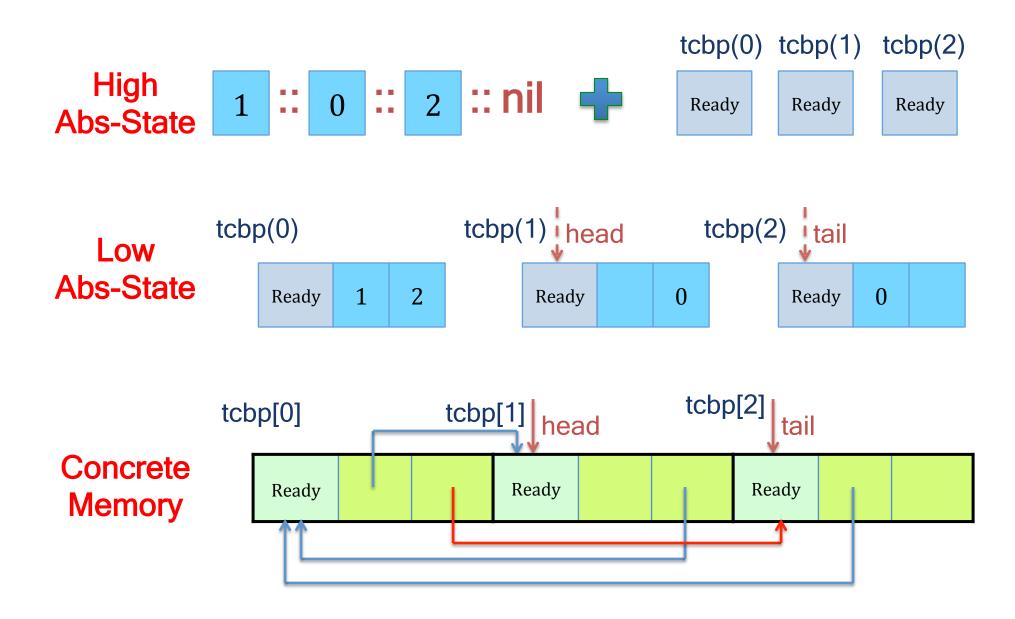


LayerLib: Vertical Composition



$$\begin{array}{cccc} L_1 \vdash_R M : L_2 & L_2 \vdash_S N : L_3 \\ \hline L_1 \vdash_{R \circ S} M \bigoplus N : L_3 \end{array} VCOMP$$

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)

Function **dequeue** (d:abs) (i:Z) :=

High Layer Spec in Coq

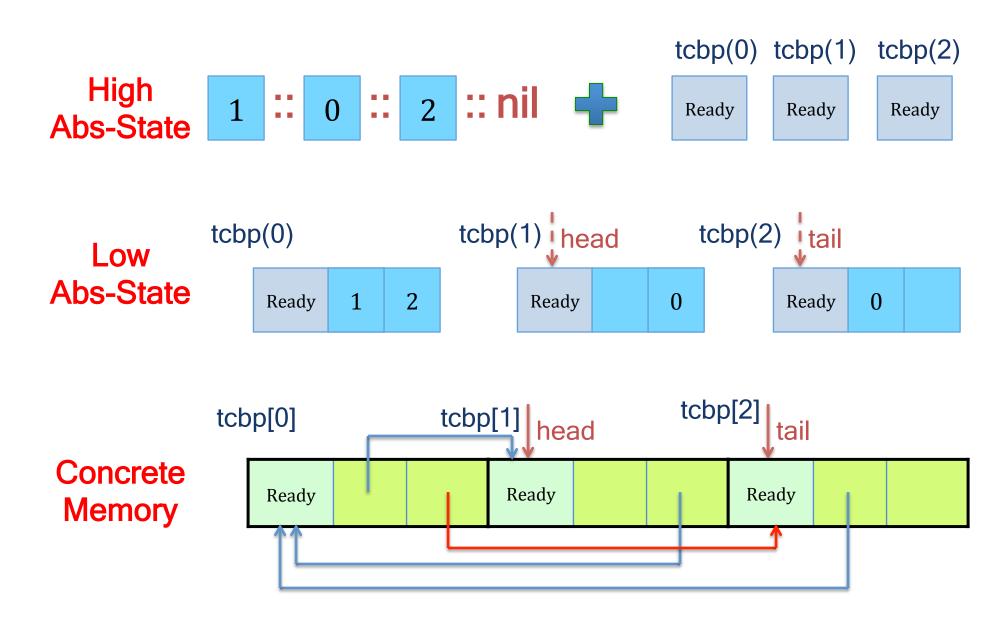
Inductive td_state := | TD_READY | TD_RUN | TD_SLEEP | TD_DEAD.

Definition tcb := td_state.

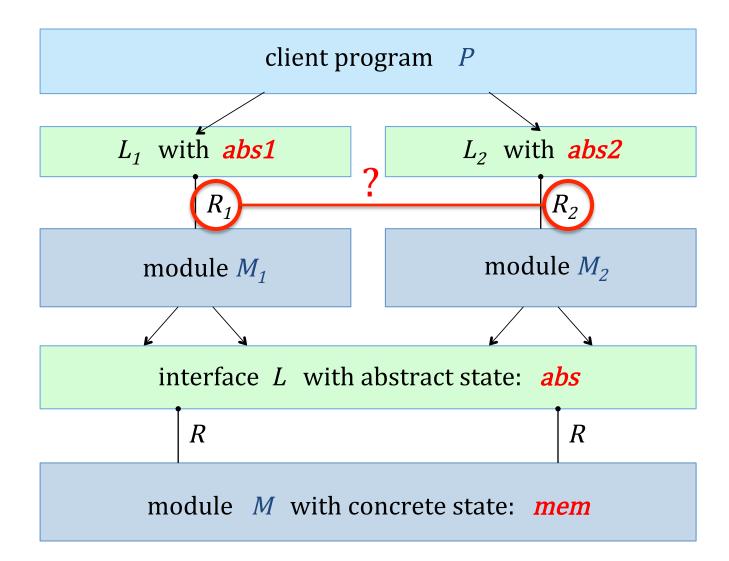
Definition tdq := List Z.

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

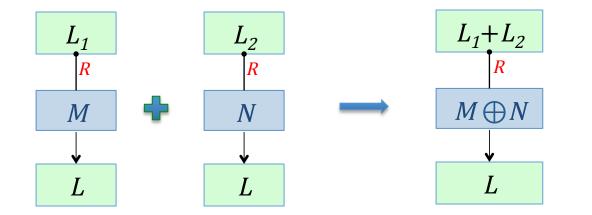
Example: Dequeue

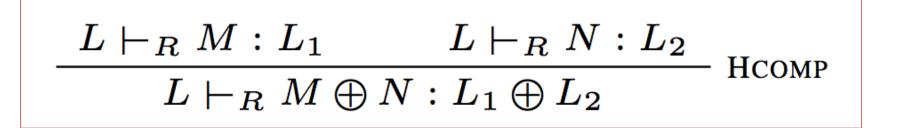


Conflicting Abstract States?



LayerLib: Horizontal Composition





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

Programming & Compiling Layers
ClightX
$$L \vdash_R M_c : L_1 \longrightarrow L_1 \leq_R [M_c]_{ClightX}(L)$$

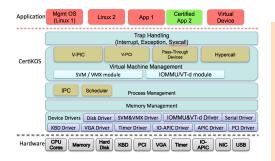
 I
CompCertX correctness theorem (where *minj* is a special kind of memory injection)
 $[M_c]_{ClightX}(L) \leq_{minj} [CompCertX[L](M_c)]_{LASM}(L)$
 I
 $L_1 \leq_{R \circ minj} [CompCertX[L](M_c)]_{LASM}(L)$
 I
 R must absorb such memory injection: $R \circ minj = R$ then we have:
 $L_1 \leq_R [CompCertX[L](M_c)]_{LASM}(L)$
 I
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 languagebased 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 oneperson-year to develop, and can boot Linux as a guest

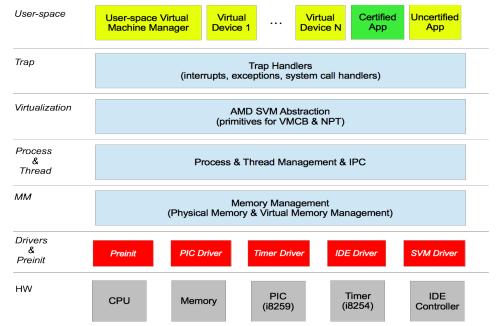
Case Study: mCertiKOS





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

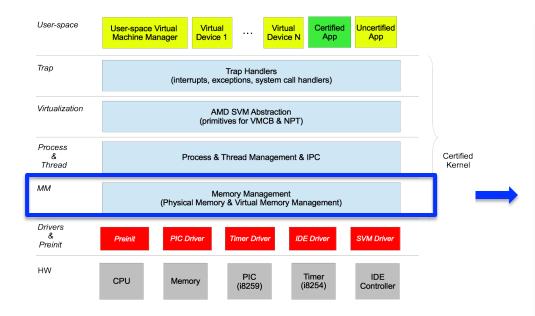
Aggressive use of abstraction over deep specs (37 layers in ClightX & LAsm)



(pe, il	xem, ihost, ipt, AT, PT, ptp, pbit, kctxp, Htcbp, Htqp, cid, chanp, uctxp, npt, hctx, vmst) thread_wakeup/kill/sleep/yield _pt_read _get/set_uctx _palloc/free _cid_get
	thread_wakeup/lill/sleep/yield pt_read get/set_uctx palloc/free cid_get sys_chcn_send/rec/wait/chck sys_yst_ext sys_gst_extic_senson sys_gst_ext_extic_senson sys_gst_ext_exti_sens
	vmcbinit pagefault_handler sys_reg_get/set sys_sync sys_run vm_exit
_	Û
TSys (mm/p	sCall Layer roc/virt.abs) vmcbinit sys_run/ PageFault sys_yield sys_check/esit/syrc/ mm/ Handler sys_yield sys_check/esit/syrc/ proc_prim
_	
TTra (mm/p	pArg Layer roc/virt.abs) vmcbinit vm run get arg/ svm checkles/des/des/des/des/des/des/des/des/des/d
vsv	M Layer roc.abs, npt, hctx, vmst) vmcbinit swn.checkjesi(/ync/ vm_run_NPT inst proc.prim
VMC	BOp Layer roc.abs.npt, hcts.vmst) vmcbinit vmcb.cht%/day/ NPT aust from Goot proc.prim
vsvi	Wintro Layer roc.abs. npt. htts. vmst) vmcbinit "Vmetried NPT_insrt freed to mmo
VVM	CPInit Lawar
	Connectable, net being the second sec
	CBIntro Layer roc.abs, npt, hctx, vmcb) nptinit vmcb_read/write NPT_insrt switch, ta/ from puest proc.prim
(mm.al	MOp Layer proc.abs, npt, hctx) nptinit NPT_insrt switch_to/from_guest proc.prim
(mm.al	MSwitch Layer bs. proc.abs, npt, hctx) nptinit NPT_insrt restore/save_hctx proc_prim
	Tinit Layer mm/, ptinit NPT_inst proc.prim
	Tintro Lavor
PPro	bd, proc. abs, npt) procinit set. NPDE set. NPTE proc. prim
(mm.a	bs, proc.abs) procinit get/set_uctx sensitive proc_create/start/exit thread prim
PUC (mm.al	tx Layer s, thread.abs, uctxp) procinit get/set/save/ send/recv/check_chan thread.prim thread.prim
PIPC	Layer ss, kctxp, Htcbp, Htqp, cid, chanp) procinit send/recv/check_chan thread.prim
PIPC	Intro Layer 28, kctxp, Htcbp, Htqp, cid, chanp) schedinit get/set_chan_thread.prim_mm.prim
PThr	ead Laver
(mm.al	28, kctxp, Htcbp, Htqp, cid) schedinit cid_get spawn/kil/wakeup mm.prim
PSch (mm.al	ed Layer 26, kctxp, Htcbp, Htqp, cid) schedinit cid eet Htcb_set thread_sched/kill mm.prim
PCID (mm.al	Layer ss, kctxp, Htcbp, Htqp, cid) htdginit cid_et Htcb Hen/de/ kcts_switch mm.prim
_	Dueue Layer ss, kotxp, Htcpp, Htcpp, htdpinit Htcb_get/set m_source
PTD	Qinit Layer
	xs, kctxp, Ltcbp, Ltqp) tdqinit Ltcb_get/set
(mm.at	os, kctxp, Ltcbp, Ltqp) tdqinit Ltcb_get/set Ltdq_get/set kctx_switch mm.prim
PTC	Binit Layer abs, kctxp, Ltcbp) tcbinit Ltcb_get/set kctx_switch/new/free mm. prim
PTC	Bintro Layer
	bs, kctxp, Ltcbp) pmapinit Ltcb_get/set/init kctx_switch/new PT_free mm. prin
	AT, PT, ptp, pbit, kctxp) pmapinit kctx_switch kctx_new PT_free mm. prin
PKC	tx Layer AT, PT, ptp, pbit, kctxp) pmapinit kctx_switch PT_new/free mm.prim
MPT	New Laver
(iflags,	AT, PT, ptp, pbit) pmapinit PT_new/free [77, resw/ read/resw] palloc/free setPT iffags_se Bit Layer
	T, ptp, pbit, iflags) PT_init_get/set_bit_PT_insrt/ resultminv_palloc/free_setPT_iflags_set
(AT, P	Init Layer , ptp., pe, ikern, ipt) PT_init PT_insrt/read/rmv palloc/free setPT iflags_set
MPT (AT, P	Kern Layer T, ptp, illags) PTInitKern PT_insrt/read/rmv palloc/free setpe setPT illags_set
_	Comm Layer T, ptp, iflags) PTInitComm PT_insrt/read/rmv palloc/free setpe setPT iflags_set
MPT	Op Layer T, ptp. (flags) meminit setPDE PT set/ rest/meminit setPDE intervention pallor/free setpe setPT interventions
MPT	Intro Lavor
MAT	Laver
(AT, if)	Op Laver
(AT, n	ps, pe, ikem, ihost, init) meminit at_get/set nps_get setpe setcr3 iflags_set
(minfo	Intro Layer AT, nps, init, iflags) bootloader at set of a
MBo (minfo	ot Layer , pe, ikern, ihost) bootloader mi_get_setpe_setcr3_iflags_set

TSysCall Layer

Decomposing mCertiKOS

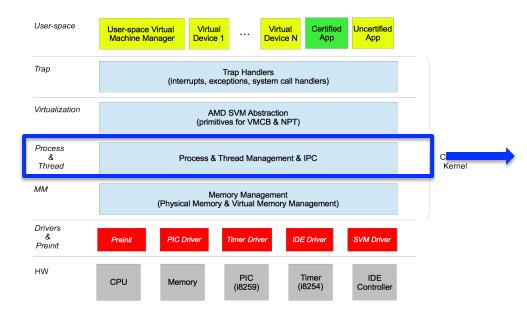


Physical Memory and Virtual Memory Management (11 Layers)

Based on the abstract machine provided by boot loader

MPTNew Layer
(iflage AT DT ptp phil) polloc/frog PT resv/ PT new/ proprint sotDT iflage sot
(illags, AI, PI, ptp, pbit) pailoc/liee read free pillapinit set i illags_set
MPTBit Layer
(iflags, AT, PT, ptp, pbit) palloc/free PT_insrt/ get/set_bit PT_init setPT iflags_set
MPTInit Layer
(iflags, AT, PT, ptp) palloc/free PT_insrt/read/rmv PT_init setPT iflags_set
MPTKern Layer
(init, iflags, AT, PT, ptp) palloc/free PT_insrt/read/rmv PTInitKern setPT iflags_set
(init, inags, AT, PT, ptp) panot/recent initiation and initiatinatination and ini
MPTComm Layer
(init, iflags, AT, PT, ptp) palloc/free PT_insrt/read/rmv PTInitComm setPT iflags_set
MPTOp Layer
(init, iflags, AT, PT, ptp) palloc/free PT_insrt/ read/rmv setPDE meminit setPT iflags_set
(init, iflags, AT, PT, ptp) palloc/free get/set/ palloc/free get/set/ palloc/free get/set/ setPDE meminit setPT iflags set
(init, iflags, AT, PT, ptp) palloc/free get/set/ rmv_PTE setPDE meminit setPT iflags_set
MAT Layer
(init, CR3, iflags, AT) pfree palloc meminit setcr3 iflags_set
MATOp Layer
(minfo, init, CR3, iflags, AT, nps) at_get/set nps_get meminit setcr3 iflags_set
(minfo_init_CP2_iffage_AT_npc)
(minfo, init, CR3, iflags, AT, nps) at_get/ set set/ set/ bootloader setcr3 iflags_set iflags_set
MBoot Layer
(minfo, init,CR3, pe, ikern, ihost, ipt) mi_get bootloader setcr3 pe/ikern/ihost_set

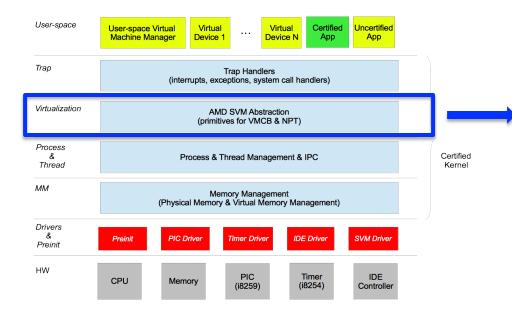
Decomposing mCertiKOS (cont'd)



Thread and Process Management (14 Layers)

PProc Layer
(mm.abs, proc.abs) get/set_uctx procinit send/recv/ proc_create/start/ exit mm/ thread.prim
PUCtx Layer (mm.abs, thread.abs, uctxp) get/set/save/ procinit send/recv/check_chan thread.prim
(mm.abs, thread.abs, uctxp) restore uctx procinit send/recv/theck_than thread.prim
PIPC Layer
(mm.abs, kctxp, Htcbp, Htqp, cid, chanp) procinit send/recv/check_chan mm/
PIPCIntro Layer (mm.abs, kctxp, Htcbp, Htqp, cid, chanp) schedinit get/set_chan thread.prim mm.prim
PThread Layer
(mm.abs, kctxp, Htcbp, Htqp, cid) schedinit cid_get thread_sleep/yield/ mm.prim
PSched Layer
(mm.abs, kctxp, Htcbp, Htqp, cid) schedinit cid_get Htcb_set thread_sched/kill mm.prim
, ppumpuncup
PCID Layer
(mm.abs, kctxp, Htcbp, Htqp, cid) htdqinit cid_get /set Htcb Hen/de/ kctx_switch mm.prim
PAbQueue Layer
(mm.abs, kctxp, Htcbp, Htqp) htdginit Htcb_get/set Hen/de/ kctx_switch mm.prim //new/free mm.prim
PTDQInit Layer
(mm.abs, kctxp, Ltcbp, Ltqp) tdqinit Ltcb_get/set rm_queue /rew/free mm.prim
PTDQIntro Layer
(mm.abs, kctxp, Ltcbp, Ltqp) tdqinit Ltcb_get/set Ltdq_get/set kctx_switch /new/free mm.prim
PTCBInit Layer
PTCBIntro Layer
(mm.abs, kctxp, Ltcbp) pmapinit Ltcb_get/set/init kctx_switch/new PT_free mm. prim
PKCtxNew Layer
(iflags, AT, PT, ptp, pbit, kctxp) pmapinit kctx_switch kctx_new PT_free mm. prim
PKCtx Layer
(iflags, AT, PT, ptp, pbit, kctxp) pmapinit kctx_switch PT_new/free mm.prim
MPTNew Layer
(iflags, AT, PT, ptp, pbit) pmapinit PT_new/free PT_resv/ palloc/free setPT iflags_set

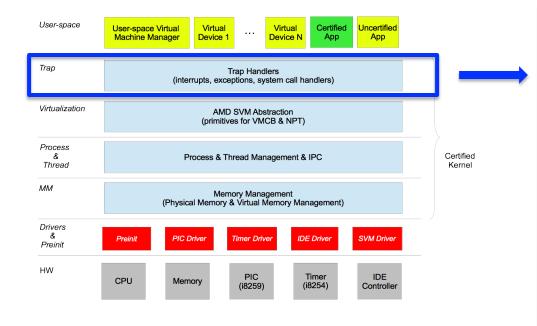
Decomposing mCertiKOS (cont'd)



Virtualization Support (9 Layers)

VSVM Layer	
(mm/proc.abs, npt, hctx, vmst) vmcbinit svm_check/exit/sync vm_run /exit NPT_insrt	mm/ proc. prim
VMCBOp Layer	
(mm/proc.abs, npt, hctx, vmst) vmcbinit vmcb check/clear/ inject/set/get (13) NPT_insrt switch_to/ from_guest	mm/ proc. prim
VSVMIntro Layer	 /
(mm/proc.abs, npt, hctx, vmst) vmcbinit vm st_read NPT_insrt switch_to/ /write NPT_insrt from guest	proc. prim
VVMCBInit Layer	
(mm/proc.abs, npt, hctx, vmcb) vmcbinit vmcb read NPT_insrt switch_to/ /write NPT_insrt from guest	mm/ proc. prim
	proceptim
VVMCBIntro Layer	
(any lange the next hat weat) patinit which read (write NDT insert switch to/	mm/
(mm/proc.abs, npt, nctx, vmcb) nptint vncb_read/write vr _ nst from guest	proc. prim
VSVMOp Layer	
(mm.abs,proc.abs, npt, hctx) nptinit NPT_insrt switch_to/from_guest	mm/
	proc. prim
VSVMSwitch Layer	
	mm/
(mm.abs,proc.abs, npt, hctx) nptinit NPT_insrt restore/save_hctx	proc. prim
VNPTInit Layer	
(mm.abd, proc.abs,npt) nptinit NPT_insrt	mm/
(mm.abd, proc.abs,npt)	proc. prim
VNPTIntro Layer	
(mm.abd, proc.abs, npt) procinit set_NPDE set_NPTE mm/ proc_prim	
PProc Layer	\searrow
(mm aba proc aba) procinit proc_create/ get/ send/recv/	mm/

Decomposing mCertiKOS (cont'd)



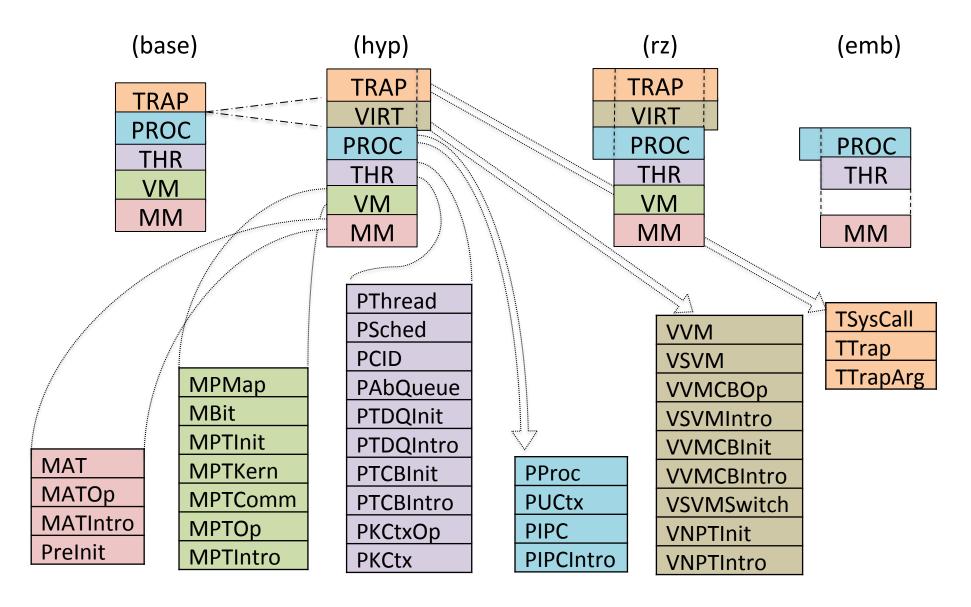
TSysCall Layer

(pe, ikern, ihost, ipt, AT, PT, ptp, pbit, kctxp, Htcbp, Htqp, cid, chanp, uctxp, npt, hctx, vmst)

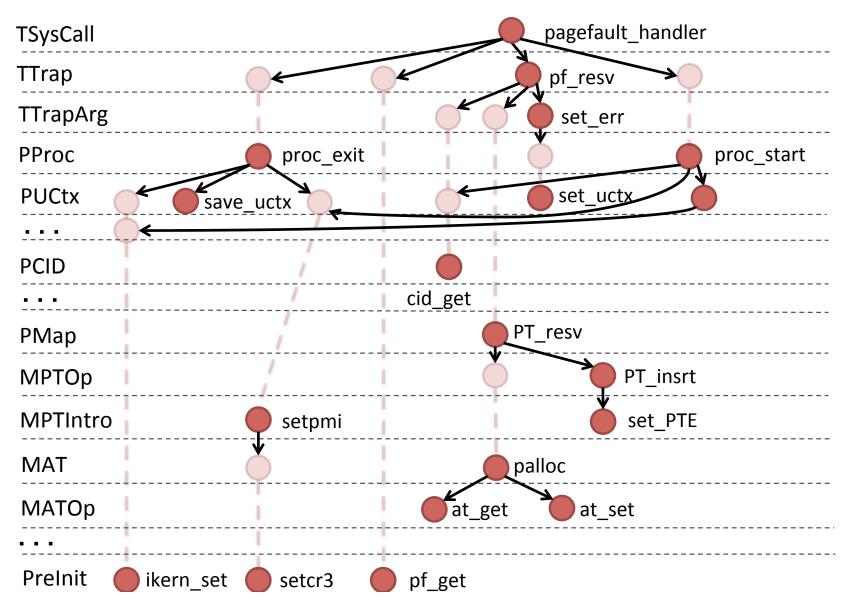
thread_v	vakeup/kill/sleep/yield	pt_read	get/set_u	ctx palloc/fi	ree cid_get		
sys_char	_send/recv/wait/checl	k sys_yie	d sys_get	sys_get_exit_reason			
sys_chec	k_shadow/pending_eve	ent sys_p	_proc_create sys_set_set		sys_inject		
sys_get_	sys_get_exit_io_width/port/rep/str/write/eip sys_set_intcept_int sys_npt_instr						
vmcbinit	pagefault_handler	sys_reg_g	et/set sys	_sync sys_ru	ın vm_exit		
		ĩ					
TSvsCall Lav	TSysCall Layer						
(mm/proc/virt.abs) vmcbinit sys_run/ PageFault sys_yield sys_check/exit/sync/ mm/ ym_exit _Handler sys_yield sys_check/exit/sync/ mm/ inject/set/chan (17) proc_prim							
TTrap Layer	s) vmcbinit vm_ru		get_ar	g/ sys_check/	exit/sync/	mm/	
(mm/proc/virt.ab	(is) vincbinic /exit	t	set_re	et inject/set/	chan (17) pr	oc. prim	
TTrapArg Layer							
(mm/proc/virt.ab	- Vm ri	un t	get_arg set_ret	/ svm_check/ inject/set/		rt/mm/ oc. prim	
VSVM Layer							
(mm/proc.abs, n	pt, hctx, vmst) vmcbi	init vm_ru /exit	in svm_che inject/	ck/exit/sync/ set/get (16)	proc. prim	NPT_insrt	

Syscall and Trap Handlers (3 Layers)

Variants of mCertiKOS Kernels



Example: Page Fault Handler



Conclusions

- Great success w. today's system software ... but why?
- We identify, sharpen, & formalize two possible ingredients
 - abstraction over deep specs
 - a compositional layered methodology
- We build new lang. & tools to make layered programming rigorous & certified ---- this leads to huge benefits:
 - simplified design & spec; reduced proof effort; better extensibility
- They also help *verification in the small*
 - hiding implementation details as soon as possible
- Still need better PL and tool support (Coq / ClightX / LAsm)

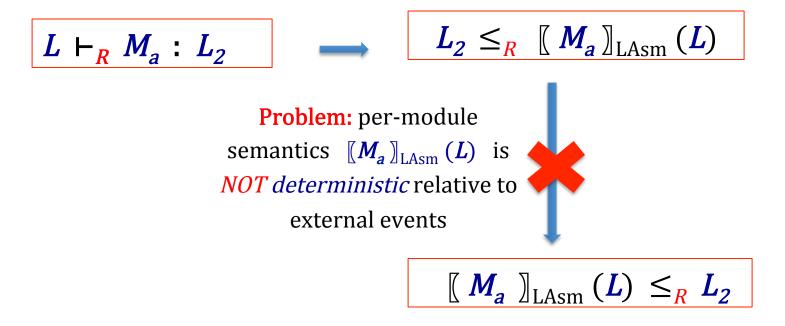
Thank You!

Interested in working on the CertiKOS project? we are hiring & recruiting at all levels:

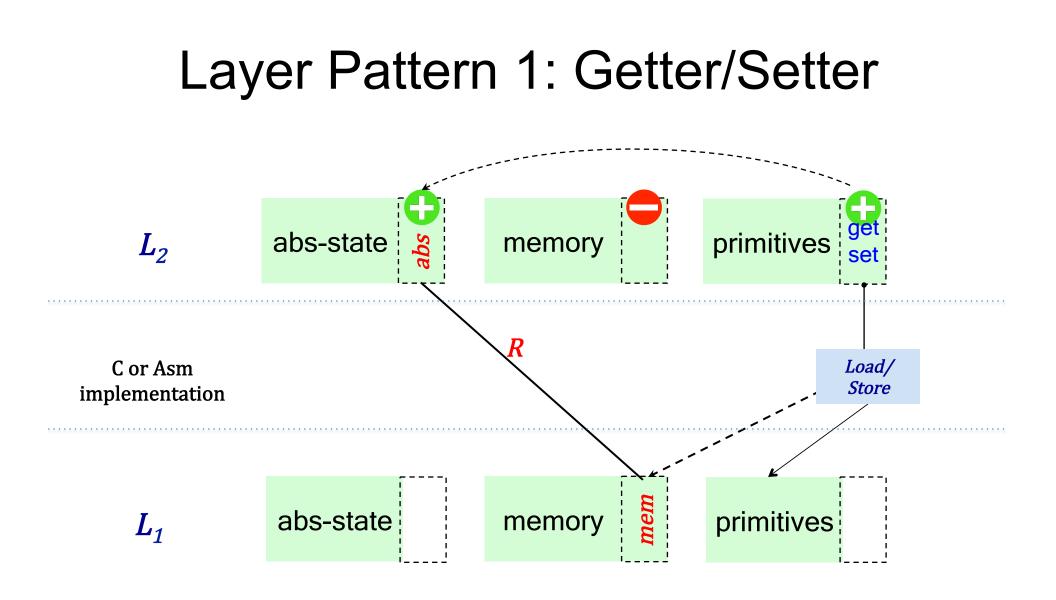
> postdocs, research scientists, PhD students, and visitors

A Subtlety for LAsm

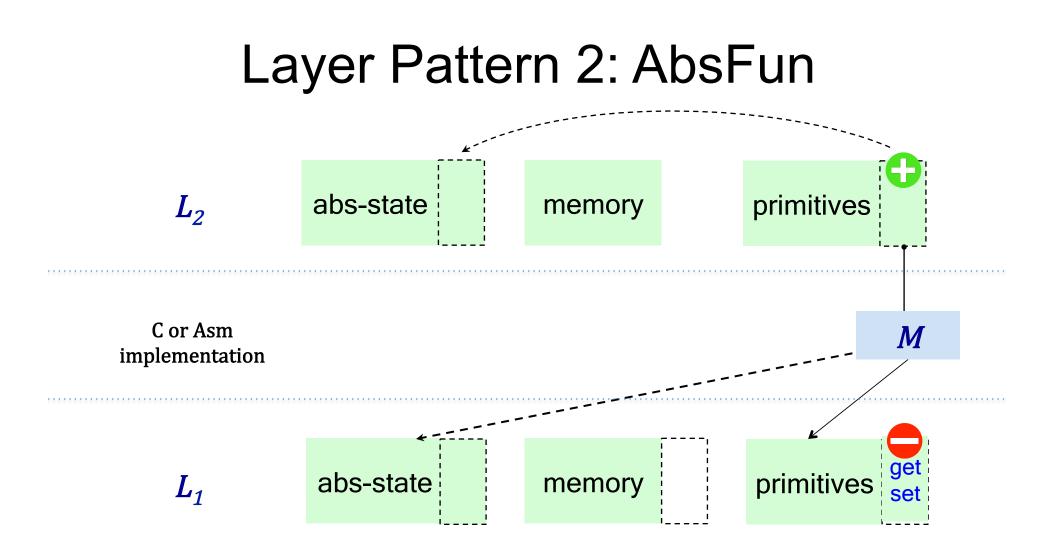
Some functions (e.g., kernel context switch) do not follow the C calling convention and must be programmed in LAsm[*L*].



Fortunately, whole-machine semantics $[\bullet]_{LAsm}(L)$ is deterministic relative to external events, so it can still be reversed: $\forall P. [P \oplus M_a]_{LAsm}(L) \sim_R [P]_{LAsm}(L_2)$



Hide concrete memory; replace it with Abstract State Only the getter and setter primitives can access memory



Memory does not change New implementation code does not access memory directly!

Development Cost

Development of Clig	10 pm	
Development of VCC	1.5 pm	
	Design: first 3 layers	0.5 pm
	Design: the rest 8	0.5 pm
Verification of mm	Refinement Proof: first 2	1.2 pm
5.7 pm	Refinement Proof: the rest	1 pm
	C verification	2.5 pm
	Design: 14 layers	1 pm
Verification of proc	Refinement Proof	0.5 pm
2.5 pm	C Verification	1 pm
	Design: 9 layers	0.6 pm
Verification of virt	Refinement Proof	0.4 pm
1.3 pm	C Verification	0.3 pm
	Design: 3 layers	0.2 pm
Verification of trap	Refinement Proof	0.1 pm
0.4 pm	C Verification	0.1 pm

Total: 9.9 *pm* + VCG *Dev:* 1.5 *pm*