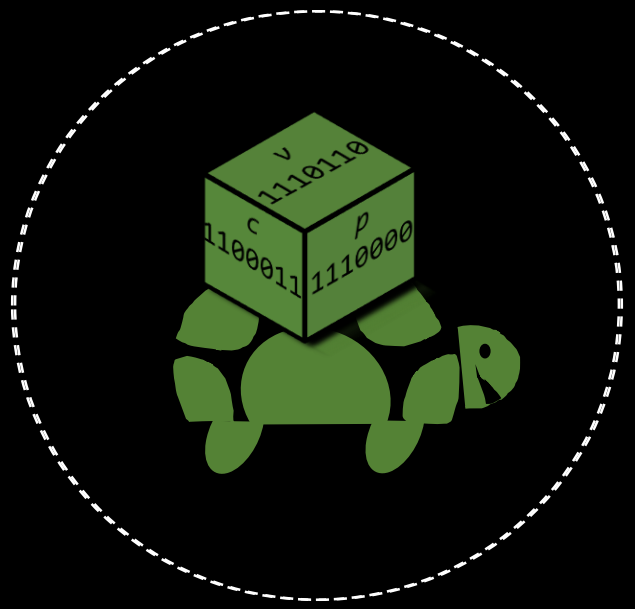
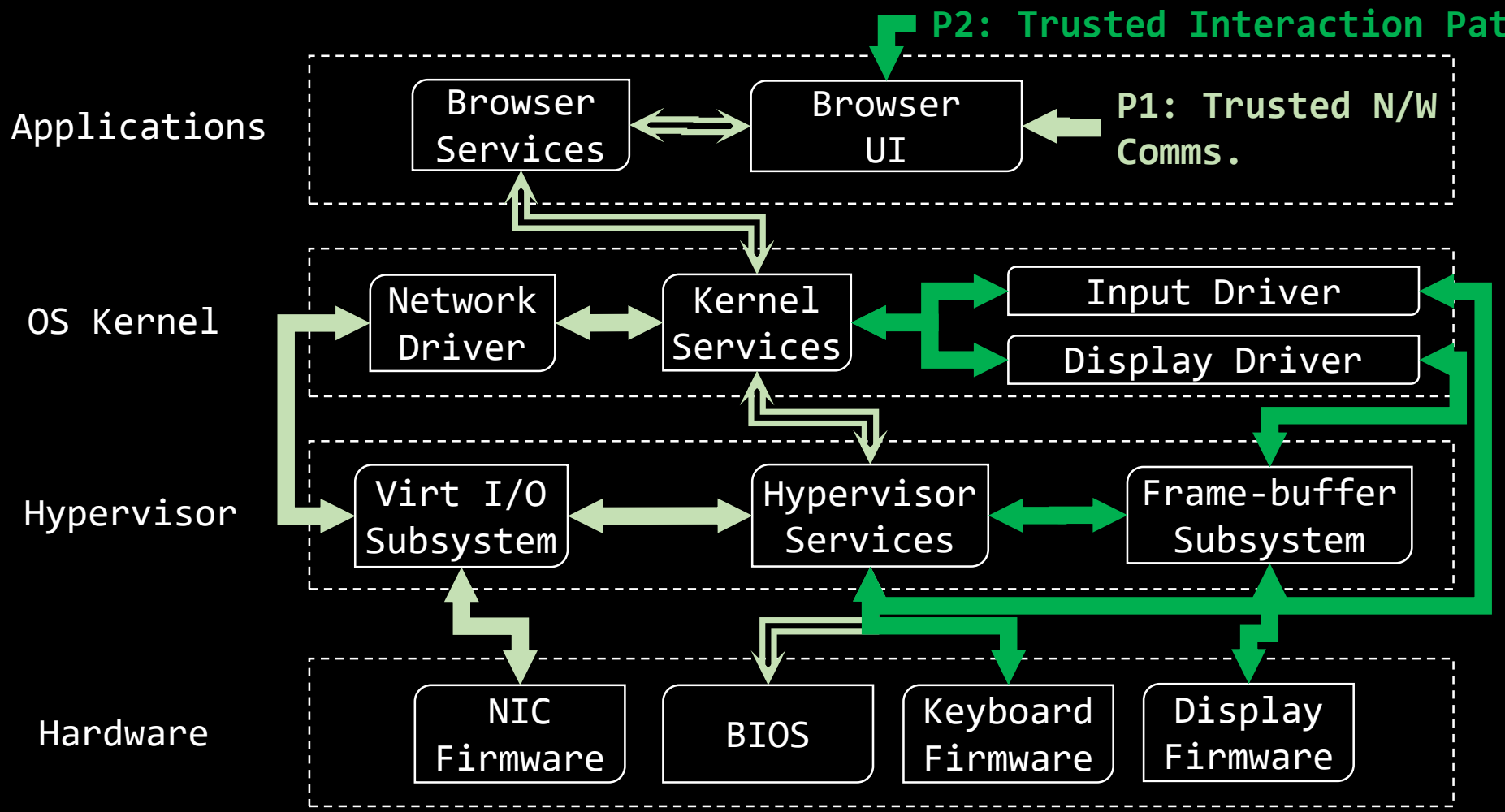


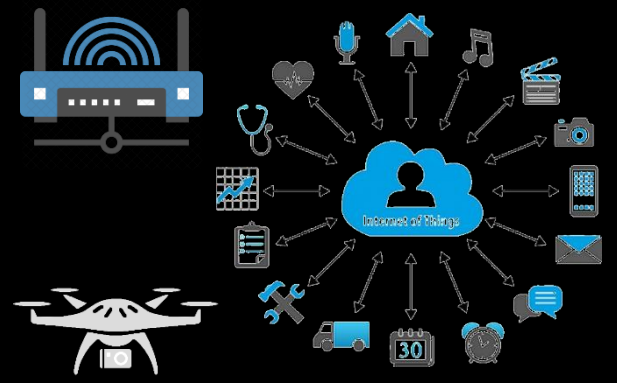
uberSpark:
Towards Piecemeal, Automated, and Composable Verification of Commodity System Software (CoSS) Stacks

Amit Vasudevan (SEI/CMU)

http://uberspark.org | http://uberxmhf.org | http://hypcode.org

Today's CoSS Stack

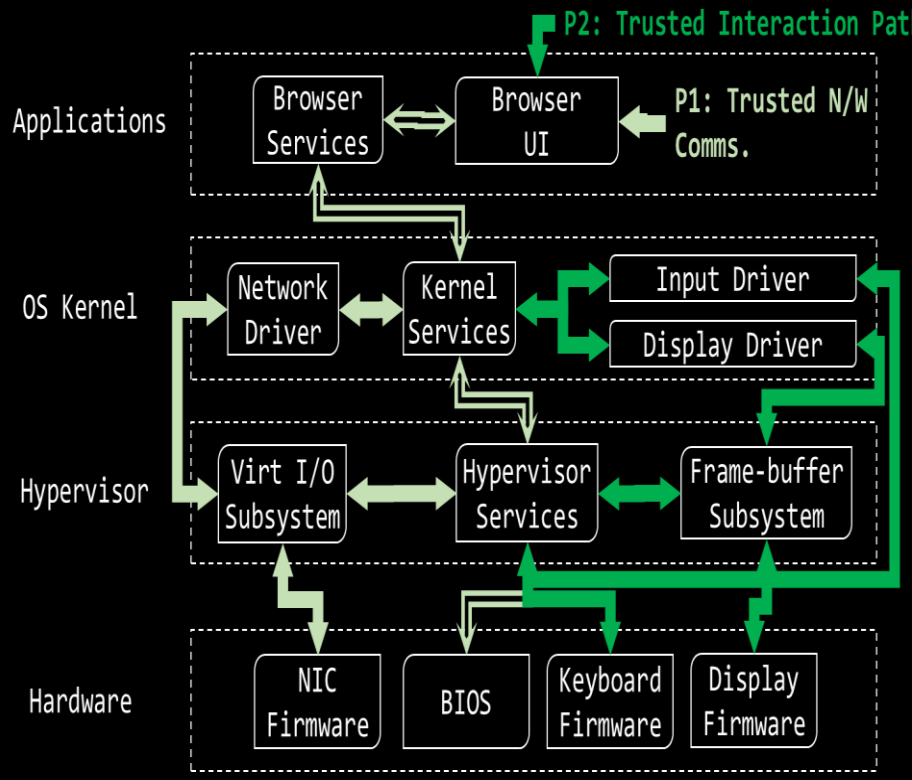




NOT Science fiction!

EXCEPT Trust through Verification!

CoSS Stack: Challenges/Goals



"Neither impossible, nor impassable!"
-- Optimus Prime, TF

Challenge: CoSS stack is not amenable to from-scratch redesign towards verification
Goal: Piecemeal Verification and Re-integration

Challenge: CoSS stack size, number of separate components, number of configurations, revisions
Goal: Compositional verification

Challenge: CoSS developers leverage device and platform hardware features, strive for performance
Goal: Commodity compatibility

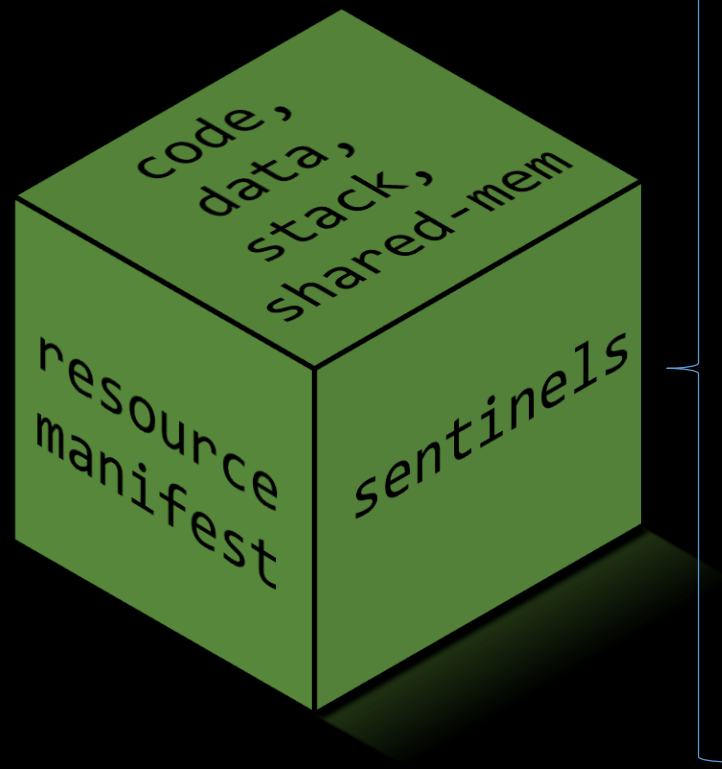
Challenge: CoSS developer adoption hinges on lessening verification burden
Goal: Developer Friendly Verification

Universal Object Abstraction (uberObject)

- resource interface confinement
  - CPU, memory and devices
- behavior specifications

Enforces uberObject resource isolation

Enables separate verification and composition of properties of different uberObjects



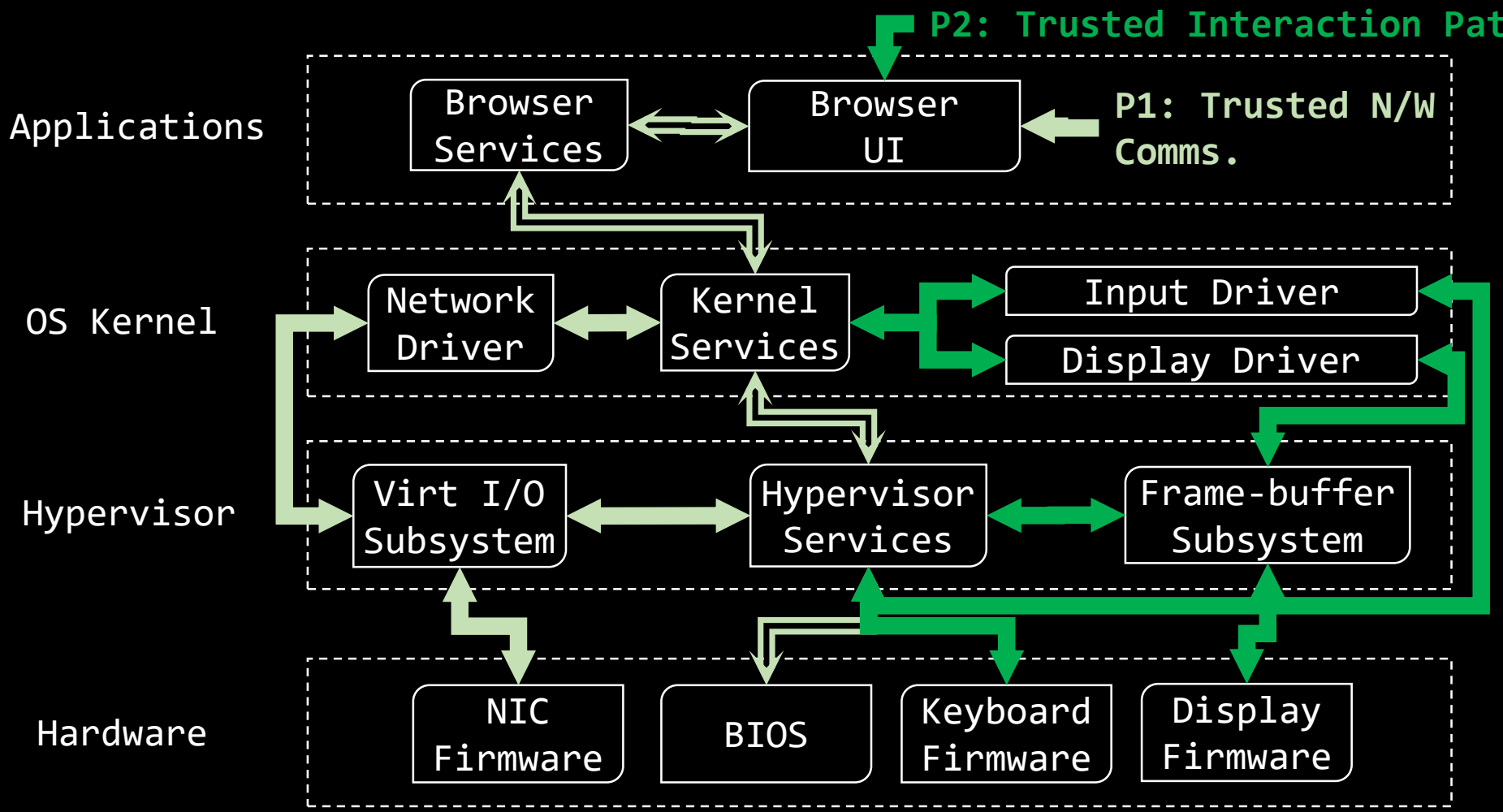
- call sentinel
- ret sentinel
- signal sentinels
- callee sentinels

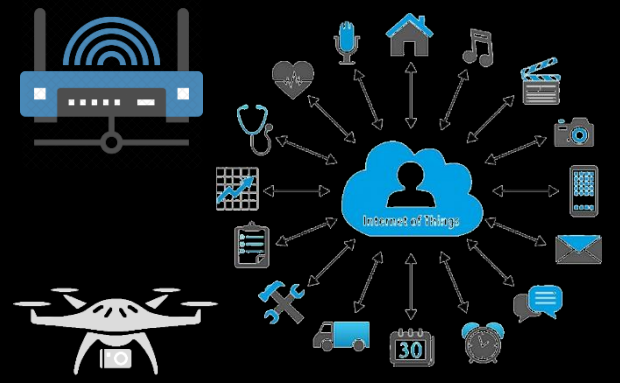
Enforces uberObject control flow integrity

Enables sound application of sequential source code verification to verify properties over sequential uoberObject invocations

Enforcement through a combination of hardware and/or software verification techniques

Today's CoSS Stack

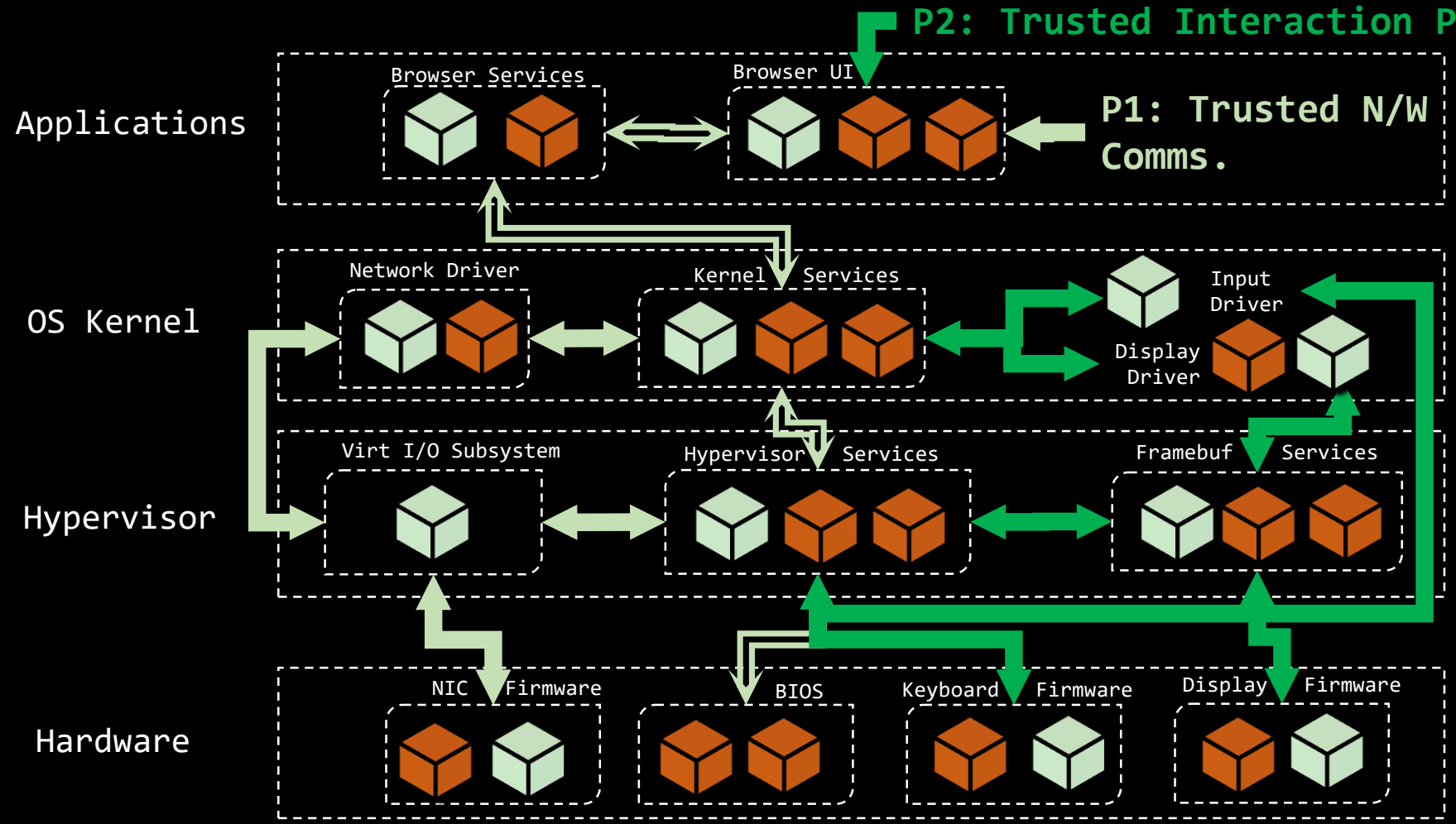


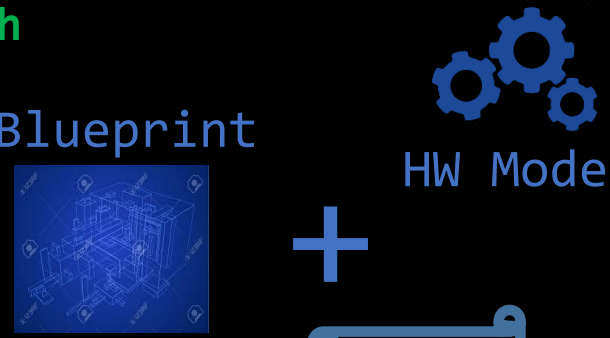


NOT Science fiction!

EXCEPT Trust through Verification!

CoSS Stack as uberObject Collections





Abstraction: Sequential non-deterministic execution of uberObjects

uberObject: Coding

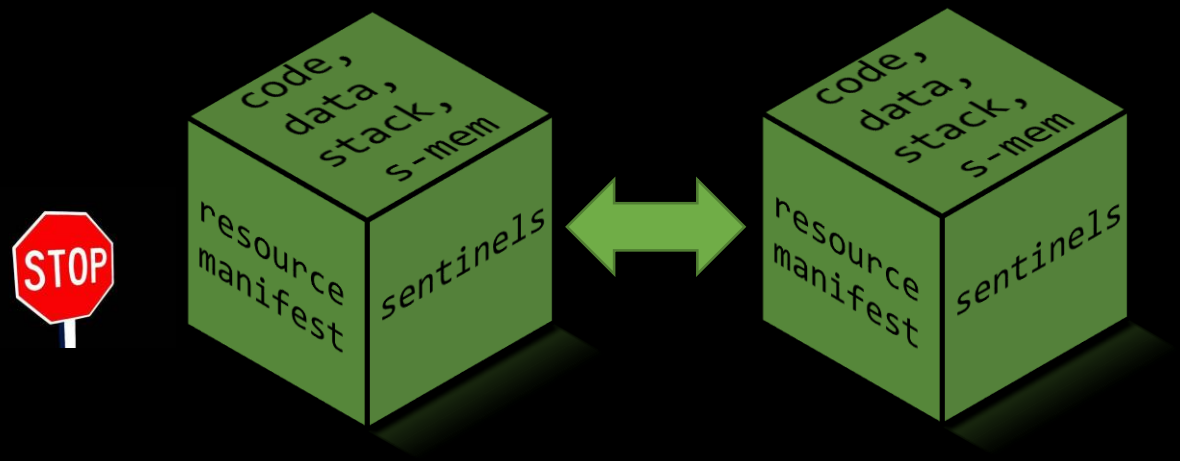
- C99 + CASM (principled Assembly)
- CASM Functions
  - C functions composed solely of Assembly instructions as macro
- HW model specifies semantics
  - Inline C99 semantics to verify
  - Inline Assembly to compile down

```
void gp_setup_vhmempgtbl(void){
  u32 i, spatype, slabid=XMHF_SLAB_PRIME;
  u64 flags; ...

  void casm_writecr3(u32 value){
    ci_movl_mesp_eax(0x4);
    ci_movl_eax_cr3();
    ci_ret();
  }                                  CASM Instructions

  vhpgtbl1t[i] = pae_make_pte((i*SZB_4K),flags);
  } ...
  casm_writecr3(vhsmpgtbl4t[0]);     CASM Function
}
```

uberObject: Verified Properties

- Base invariants
  - memory safety and control flow integrity
  - Automated w/o developer assistance
- uobject specific invariants
- System invariants via special uberObjects
  - prime, MMU, DMA and SMP (multi-processor)

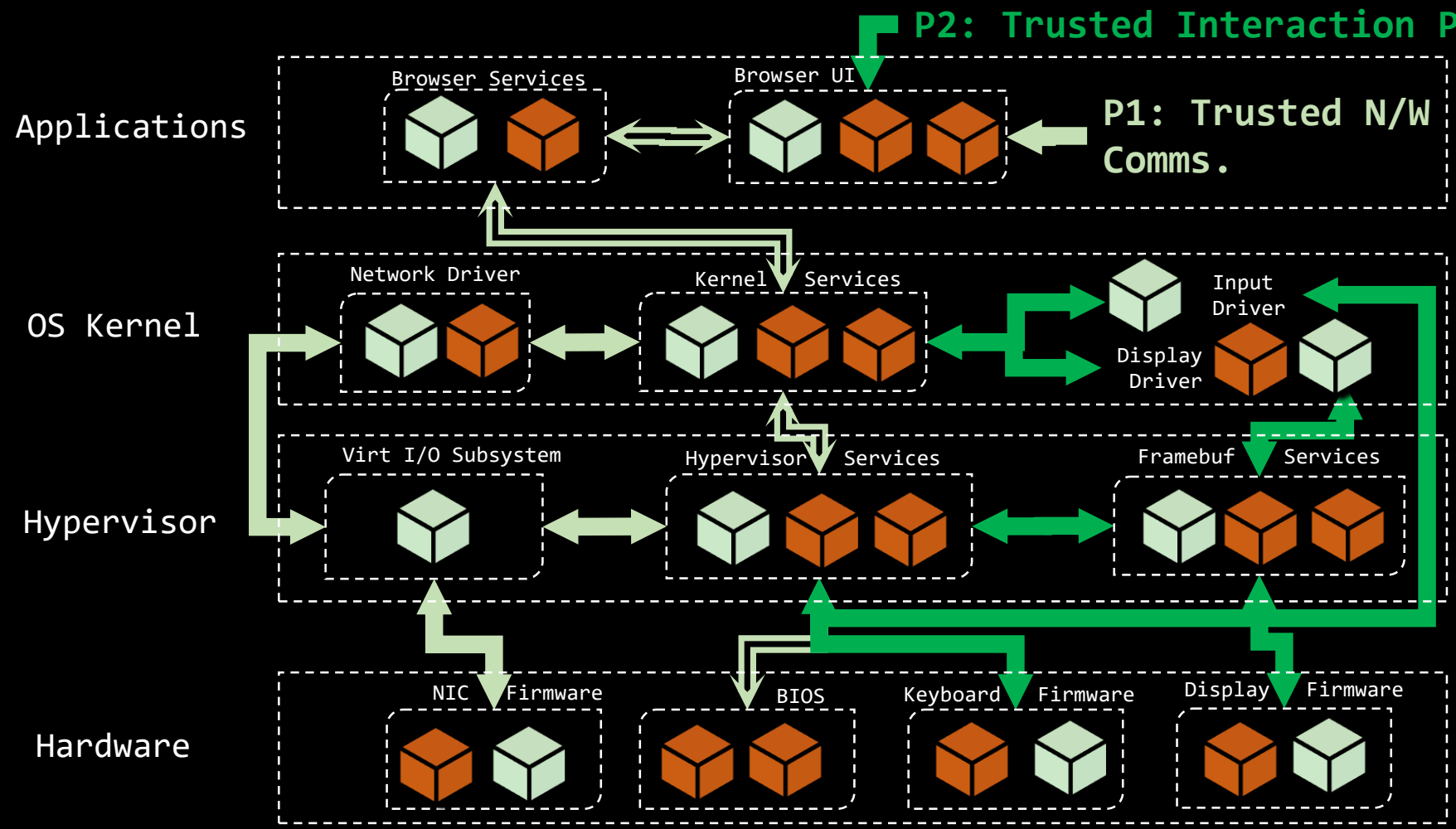
uberObject: Behavior Restriction and Composition



- Behavior Restriction
  - Syntactically verify code for allowed C99 features (e.g., no function pointers)
  - Specific CASM instructions (e.g. no MOV to CR3)

- Composition
  - Wrap a reference monitor around (shared) resource
    - MMU, IOMMU, CRs, MSRs, Devices
  - Client object manifests how it will use a resource
    - Verified on client via assertions
  - During integration
    - Use manifests combined into one formula

So, what do we have here?



Piecemeal Verification

Composable Verification

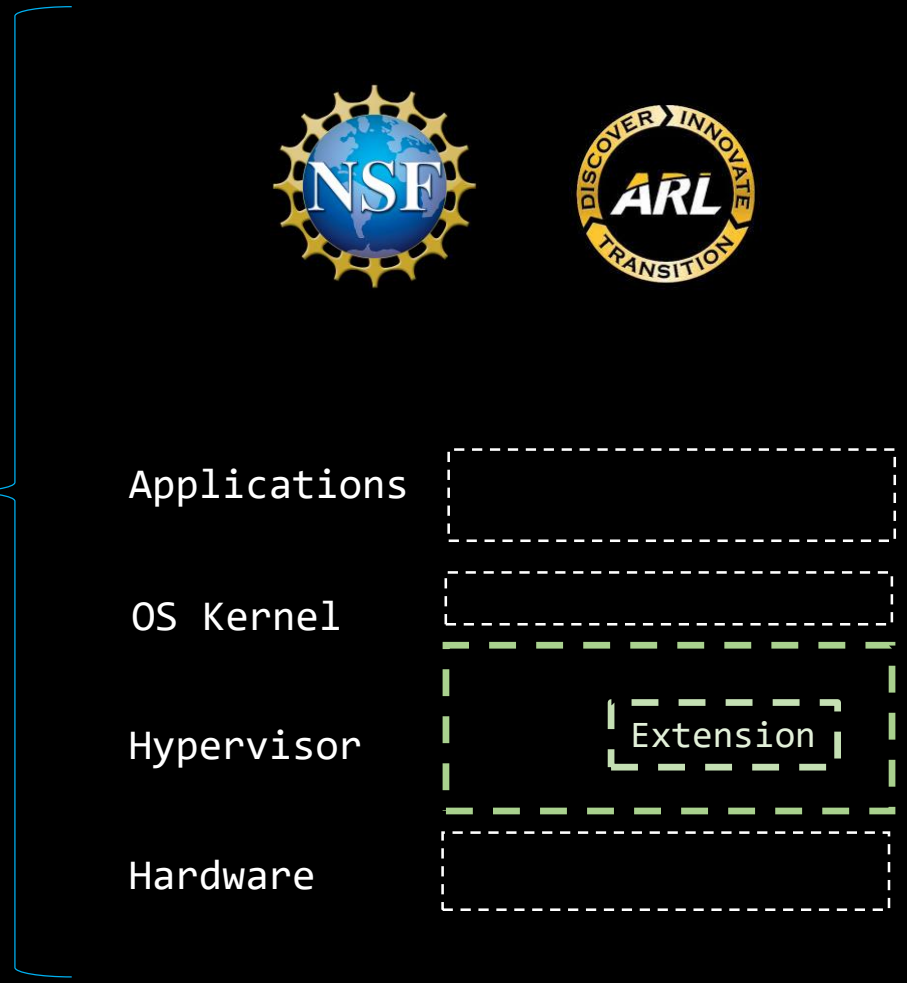
Commodity Compatibility

Developer Friendly Verification

"Neither impossible, nor impassable!"
-- Optimus Prime, TF

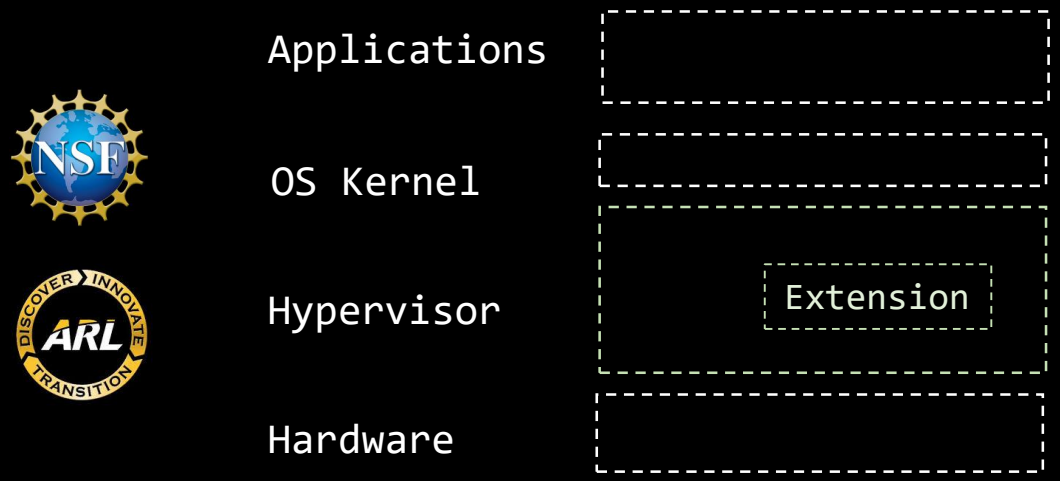
eXtensible Micro-Hypervisor Framework (XMHF)

- XMHF [2010-2015] (http://xmhf.org)
  - Core hypervisor + single extension
  - Rich Guest
    - 32-bit SMP OS (Ubuntu 12.04) on Intel/AMD
  - Various extensions
    - tracing, attestation, app-level integrity, trusted path
- Verified for memory integrity [IEEE S&P 2013]
  - CBMC/model-checking
  - No hardware states, assembly language, loops

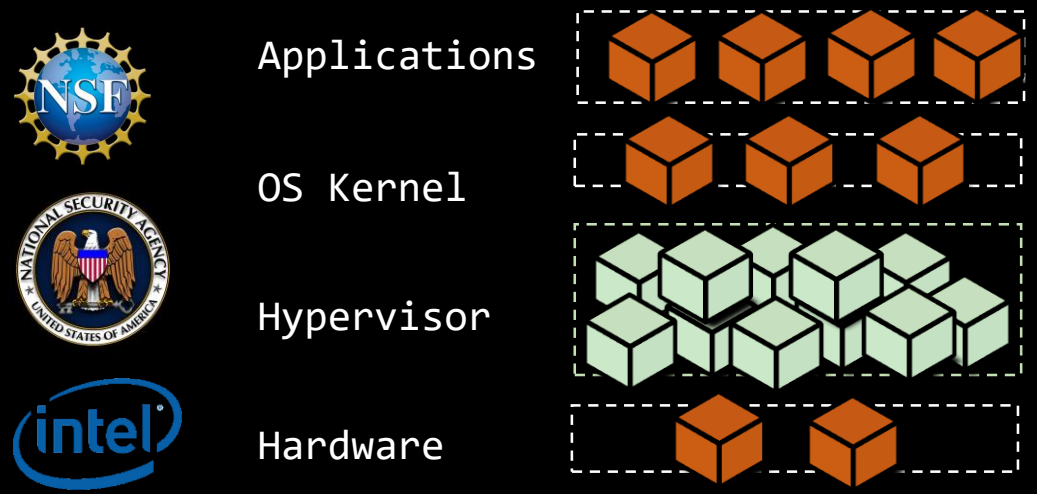


XMHF [2010-2015] → uberXMHF [2016 – current]

- XMHF [IEEE S&P 2013]
  - single extension
  - unverified hardware states
  - unverified loops
  - unverified assembly language



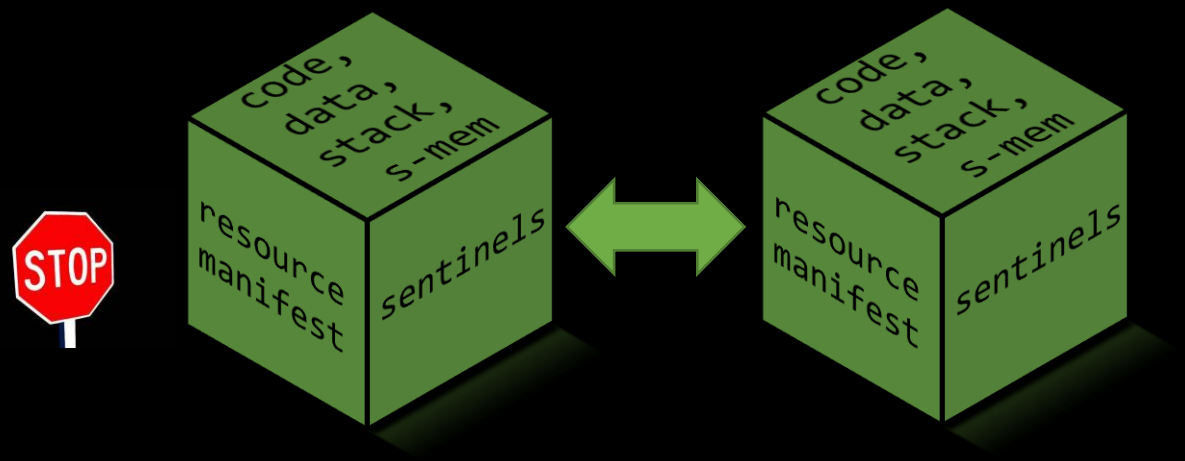
- uberXMHF [USENIX Sec 2016; IEEE EURO S&P 2018]
  - multiple extensions
  - verified hardware states, loops and assembly language (Frama-C)
  - 11 verified uobjs, 1 person year piecemeal



uberObject: Verified Properties

- Base invariants
  - memory safety and control flow integrity
  - Automated w/o developer assistance
  - Frama-C: (Evolved) Value Analysis
    - uberSpark base invariant plugin (add assertions)
- uobject specific invariants
  - ANSI C Specification Language (ACSL) requires/assigns/ensures along with asserts
  - Hoare triple proven automatically via Frama-C wp + ensemble of SMT solvers
  - prime, MMU, DMA and SMP (multi-processor)

uberObject: Behavior Restriction and Composition



- Behavior Restriction
  - Syntactically verify code for allowed C99 features (e.g., no function pointers)
  - Specific CASM instructions (e.g. no MOV to CR3)
  - Frama-C AST analysis
- Composition
  - Wrap a reference monitor around (shared) resource
    - MMU, IOMMU, CRs, MSRs, Devices
  - Client object manifests how it will use a resource
    - Verified on client via assertions
  - During integration
    - Use manifests combined into one formula
    - Frama-C EVA/SMT solvers check composability

uberObject: Coding

- C99 + CASM (principled Assembly)
- CASM Functions
  - C functions composed solely of Assembly instructions as macro
- HW model specifies semantics
- Custom Frama-C verification plugins
  - Inline C99 semantics to verify
  - Inline Assembly to compile down

```
void gp_setup_vhmempgtbl(void){
  u32 i, spatype, slabid=XMHF_SLAB_PRIME;
  u64 flags; ...
  void casm_writecr3(u32 value){
    ci_movl_mesp_eax(0x4);
    ci_movl_eax_cr3();
    ci_ret();
  }                                  CASM Instructions
  vhpgtbl1t[i] = pae_make_pte((i*SZB_4K),flags);
  } ...
  casm_writecr3(vhsmpgtbl4t[0]);     CASM Function
}
```

uberXMHF Verification Results [USENIX Sec 2016]

- Verification Tools TCB
  - Frama-C, uberSpark Plugins (1021 SLoC), SMT Solvers (Z3, CVC3, Alt-ergo), HW Model (2079 SLoC)
- Security Invariants in Core Hypervisor and Extensions
  - memory-safety, control-flow integrity, no direct writes to hypervisor memory by guest, DEP, guest syscalls n/w logging
- Development and Verification Metrics
  - 11 uberObjects, 5544 SLoC total ACSL annotations
  - Annotation to code ratio 0.2:1 to 1.6:1
  - uberObject verification times from 48s to 23 min; cumulative ~1hr
  - Took 1 person year total, piecemeal

uberXMHF Performance Results [USENIX Sec 2016]

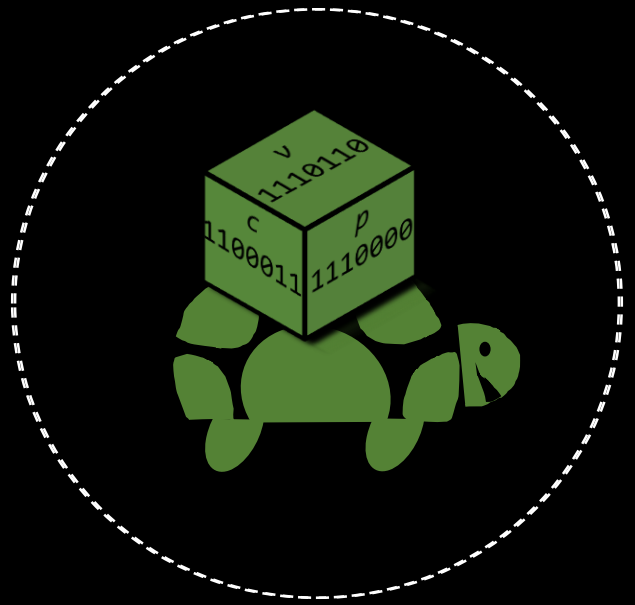
- Sentinel transfer cost

Verified-Verified	Verified-Unverified / Uverified-Verified			
	SEG	CR3	TSK	HVM
2x	37x	48x	70x	278x

- uberXMHF vs. vanilla XMHF
  - Verified extensions (2% avg. overhead)
  - Unverified extensions (10% avg. overhead)
  - I/O and normal Guest performance unaffected!

uberXMHF: Verification & Frama-C Experience

- ACSL requires/assigns/ensures
  - greatly aided behavior specifications (e.g., MMU and device page-tables, I/O tables)
- Loop invariants as proof assists
  - looping constructs over arrays (e.g., page-tables)
  - Wanna have: automatic loop invariant generation
- ACSL ghost variables and assertions
  - Hardware model and invariants over h/w state
  - Wanna have: ghost variables as part of external specifications
- ACSL predicates
  - behavior modeling/specification of sentinel stack
- AST-based analysis
  - behavior restrictions
- Other wanna haves ;)
  - better (incremental) project state save/restore
  - selectively turn off integer overflow checks (e.g., wp/sha-1)
  - memory model integration
  - multi-threaded verification



uberSpark:
Towards Piecemeal, Automated, and Composable Verification of Commodity System Software (CoSS) Stacks

Amit Vasudevan (SEI/CMU)

Questions?

http://uberspark.org | http://uberxmhf.org | http://hypcode.org