Compositional security for higher-order systems

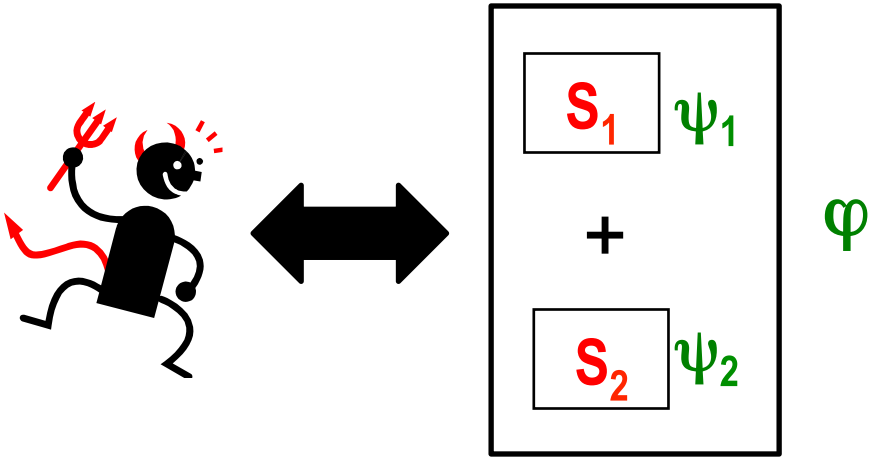
Limin Jia, ECE&INI CMU

Deepak Garg, MPI

Anupam Datta, CSD&ECE&CyLab CMU

Science of Security NSA Lablet
Sept. 27, 2013

Goal: Compositional security



S_1 ψ_1

+

S_2 ψ_2

φ

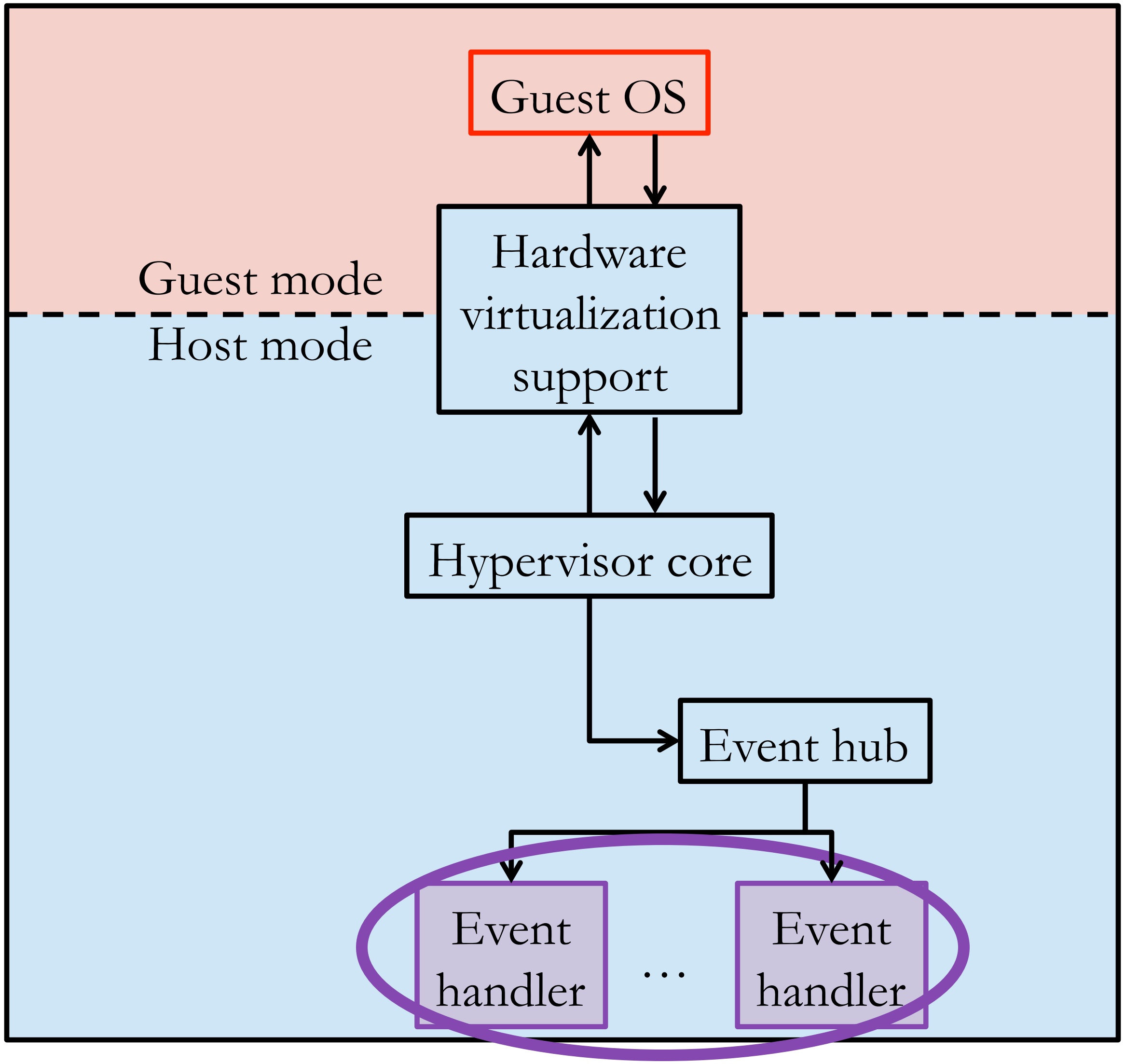
- Do S_1+ S_2 satisfy a global security property φ based on local properties ψ_1 of S_1 and ψ_2 of S_2 that are checkable separately?

Challenges

- **How to model and reason about the properties of the system in the presence of adversaries**
    - Trusted components execute adversary supplied code
        - Examples:
            - Dynamically downloaded script,
            - Trusted component's code region may be modified by the adversary

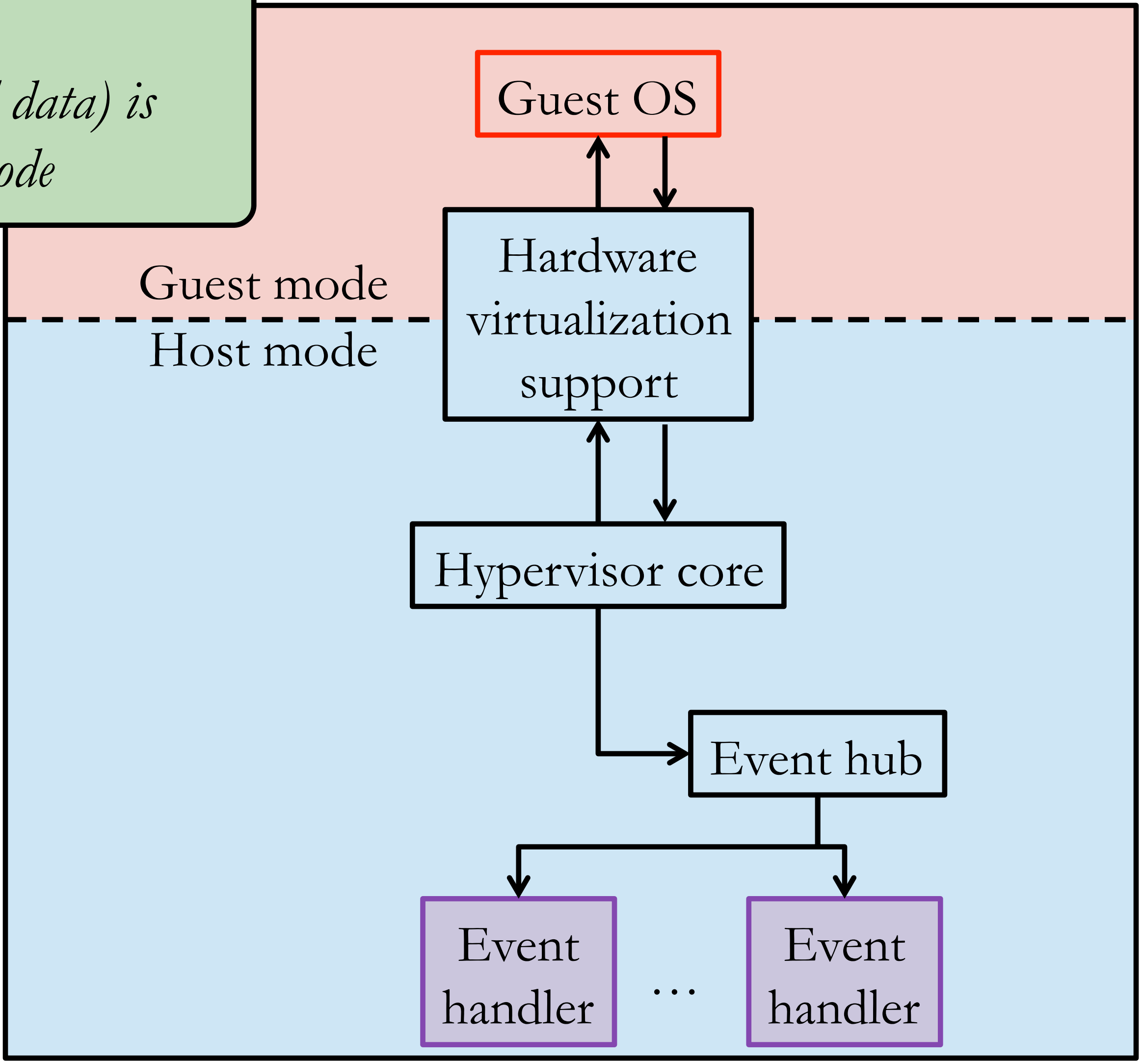
- **Key ideas:**
    - Interface-confined adversary (higher-order)
    - Leverage code-integrity property

Case study: An extensible hypervisor



Case study: An extensible hypervisor

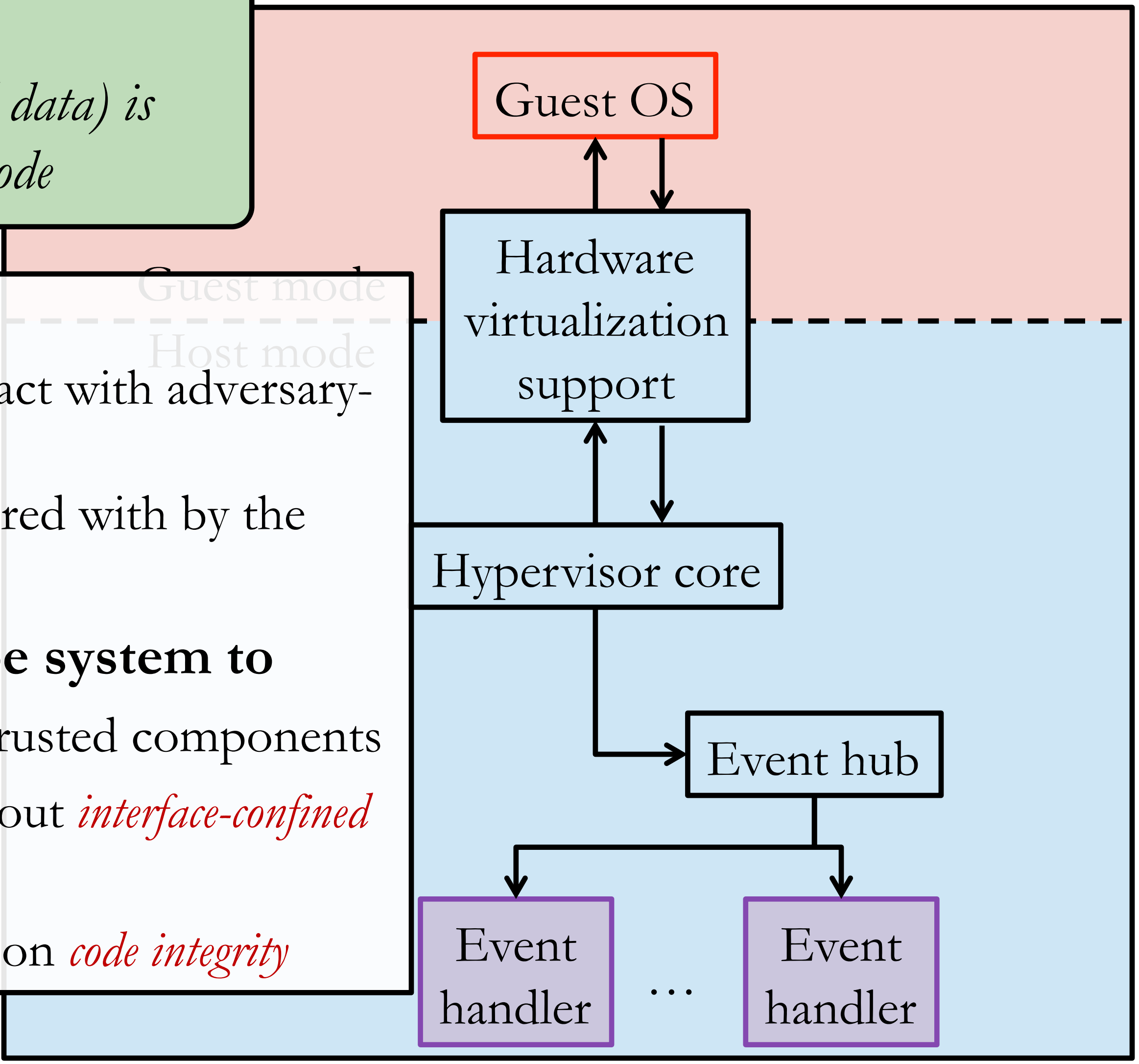
Memory Integrity:
Hypervisor's memory (code and data) is only written to by hypervisor's code



Case study: An extensible hypervisor

Memory Integrity:
Hypervisor's memory (code and data) is only written to by hypervisor's code

- **Challenges**
  - Trusted components interact with adversary-supplied code
  - Core's code may be tampered with by the adversary
- **Our approach: use a type system to**
  - Analyze the programs of trusted components
  - Abstraction and reason about *interface-confined adversary*
  - Deriving properties based on *code integrity*



Outline

- Background
- Model
- Type system
- Case study

System model

- Configuration
  - $C ::= \sigma \triangleright T_1 \mid T_2 \mid \dots \mid T_n$

Shared state

Some malicious

- System transition
  - $C_1 \Rightarrow(u)\ C_2$ iff exists i, T_i transition to T_i' at time u
- Trace $\mathcal{T}$ is a sequence of transitions
  - $(u_0)\ C_1 \Rightarrow(u_1)\ C_2 \Rightarrow(u_2) \dots \Rightarrow(u_n)\ C_n$

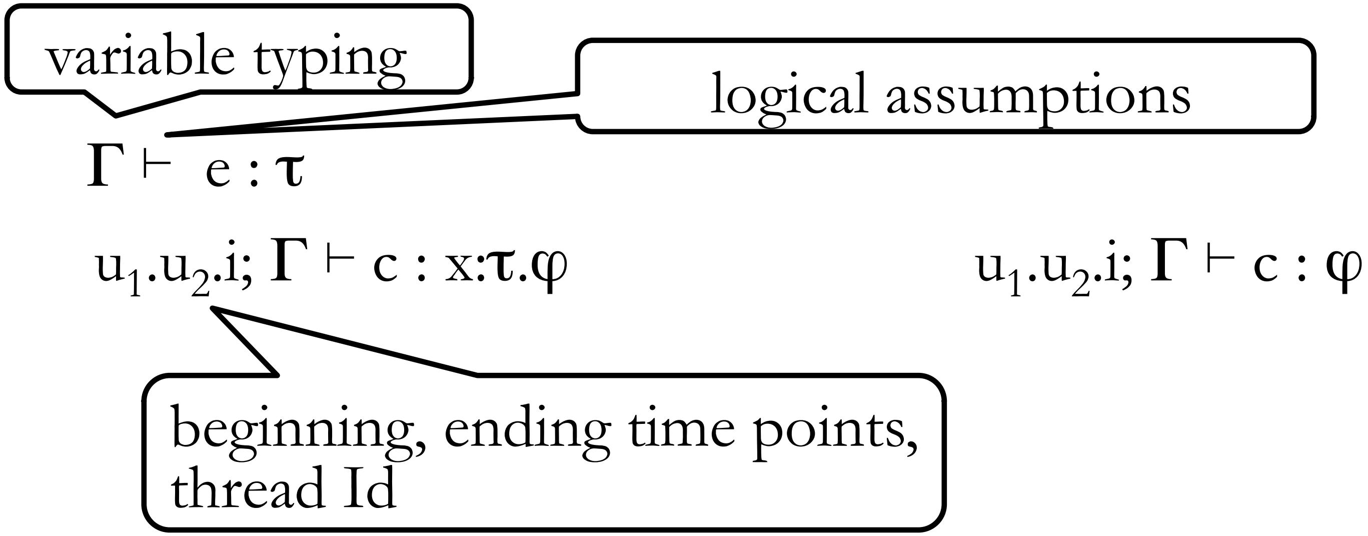
$$\underset{(t_b)}{\sigma}\ \boxed{\begin{array}{l} T_1 \\ \mid T_2 \\ \dots \\ \mid T_i \\ \dots \\ \mid T_n \end{array}} \Rightarrow(t_1)\ \sigma_1\ \boxed{\begin{array}{l} T_1 \\ \mid T'_2 \\ \dots \\ \mid T_i \\ \dots \\ \mid T_n \end{array}}$$

Types specify properties of components

- **Assertions**
  - Properties about execution traces
  - E.g. $\varphi = (\text{Read } i\ r\ x\ u_1) \wedge (\text{Read } i\ x\ y\ u_2) \wedge (u_1 < u_2)$
- **Computation types**
  - comp(τ, φ): partial correctness type
    - If e finishes, then it returns a value of type τ and the trace $\mathcal{T}$ containing the execution of e satisfies φ
      - $\mathcal{T}$ also contains other threads runs concurrently with e
  - comp(φ): invariant type
    - While e is running, the trace $\mathcal{T}$ containing the execution of e satisfies φ
      - $\mathcal{T}$ also contains other threads runs concurrently with e

Reasoning system

- **Typing judgments**



variable typing

logical assumptions

$\Gamma \vdash e : \tau$

$u_1.u_2.i; \Gamma \vdash c : x{:}\tau.\varphi$ $\qquad$ $u_1.u_2.i; \Gamma \vdash c : \varphi$

beginning, ending time points, thread Id

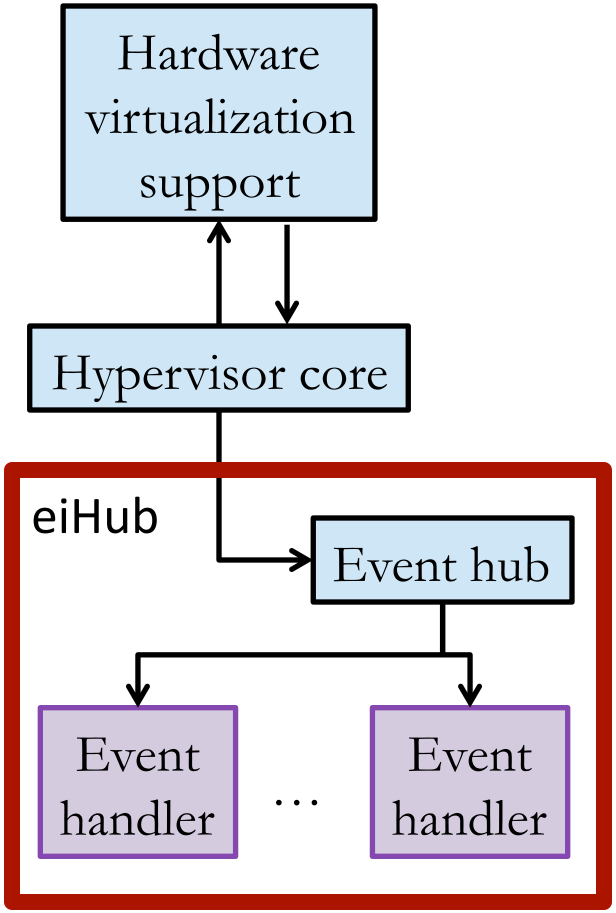
$\Gamma \vdash \varphi$

- **Typing rules construct valid typing derivations**
  - Leaves of the derivations are sound assertions about atomic actions
    - Have to be proven sound given the sematic model
  - Typing rules are sequential and parallel composition principles

Reasoning about the adversary

- **Adversary is interface-confined**
  - Can affect the system's state by calling interfaces
  - E.g., Event handlers can only access core's memory using safe read and write functions
  - To analyze its effect, we need to
    - Analyze the implementation of the interfaces

- **eiHub SafeRead SafeWrite**



Adversary typing – Typing rule

- **Conservatively approximate e's effects τ based on its simple type π and assertion φ**
  - stype(e, π): simple typing constraints, do not reason about effects
    - E.g., a Boolean is not used as a function
    - Can be achieved via cheap dynamic checking
  - Conf τ π φ: τ specifies expressions have effect φ

$$\frac{\Gamma \vdash \mathrm{stype}(e, \pi) \qquad \mathrm{Conf}\ \tau\ \ \pi\ \ \varphi \qquad \mathcal{C}(\varphi)}{\Gamma \vdash e : \tau}$$

Example

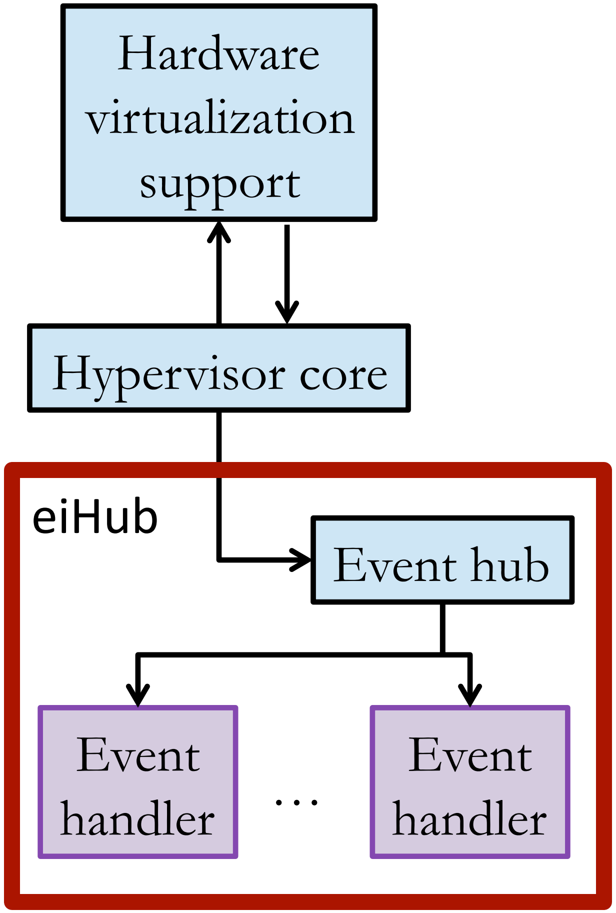
eiHub is a function

1. takes two functions as arguments
2. Only access memory through those two functions

$\mathrm{stype}(\mathrm{eiHub}, \pi) \vdash \mathbf{e}\mathrm{iHub} : \tau$

eiHub is a function

1. takes two functions (r, w) as arguments
2. If r and w maintain a memory invariant φ, then the body of eiHub mains the invariant φ



Leveraging code integrity

- **let** $\color{red}{x}$ **= read** L_{ihub}
  in let y = $\color{red}{x}$ **SafeRead SafeWrite**
  in ret y

$\vdash x = \text{eiHub}$ $\qquad$ $\vdash \textbf{eiHub} : \boldsymbol{\tau}$

$\Uparrow$

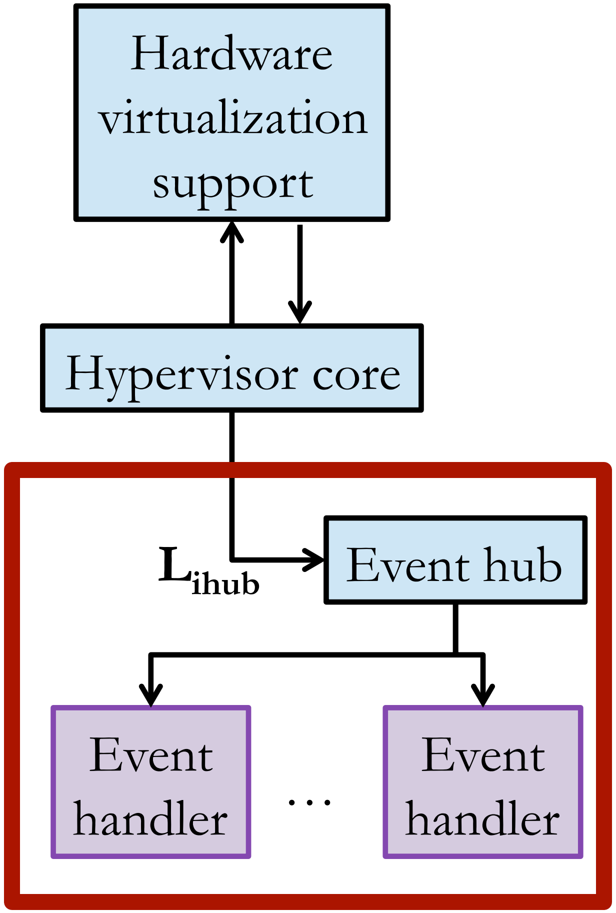
$\text{stype}(\text{eiHub}, \pi)$

$\forall v, \text{mem}\ (L_{ihub}, v)\ @\ u_2 \supset (v = \text{eiHub})$ $\qquad \vdash \quad x : ?\ \boldsymbol{\tau}$

$\forall v, \text{mem}\ (L_{ihub}, v)\ @\ u_2 \supset (x = v)$

Beta equivalence
(evaluate to the same term)

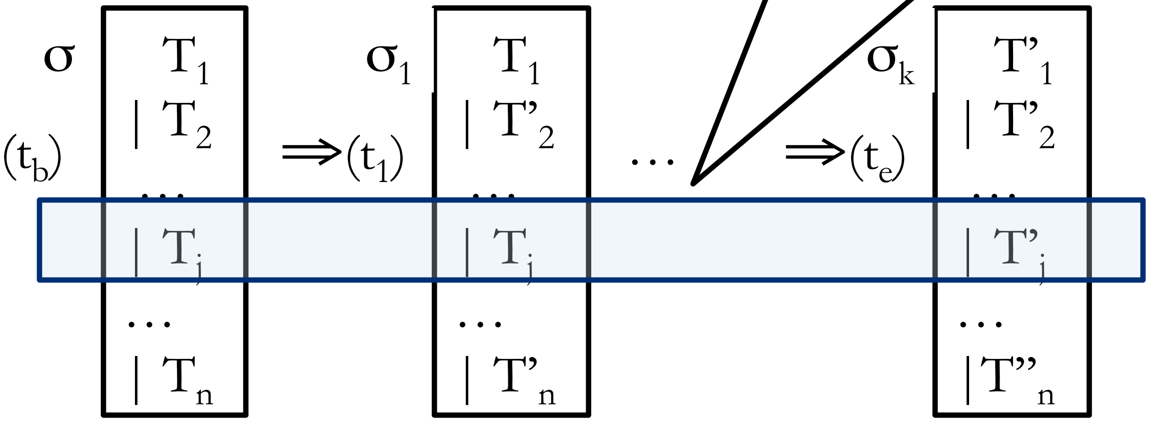
$$\frac{\Gamma \vdash e' : \tau \qquad \Gamma \vdash e' \equiv e}{\Gamma \vdash e : \tau}\ \text{Beta}$$



Formal properties of the type system

- **Soundness**
  - If $\Gamma \vdash \varphi$ then for all substitution δ for Γ, for all trace $\mathcal{T}$, $\mathcal{T} \vDash \text{AssumptionsIn}(\Gamma)\delta$ implies $\mathcal{T} \vDash \varphi\delta$
- **Composition (Robust safety)**
  - If u1; u2; i; $\vdash c : \varphi$ then $\mathcal{T} \vDash \varphi[\text{Ub}, \text{Ue}, j / \text{u1}, \text{u2}, i]$

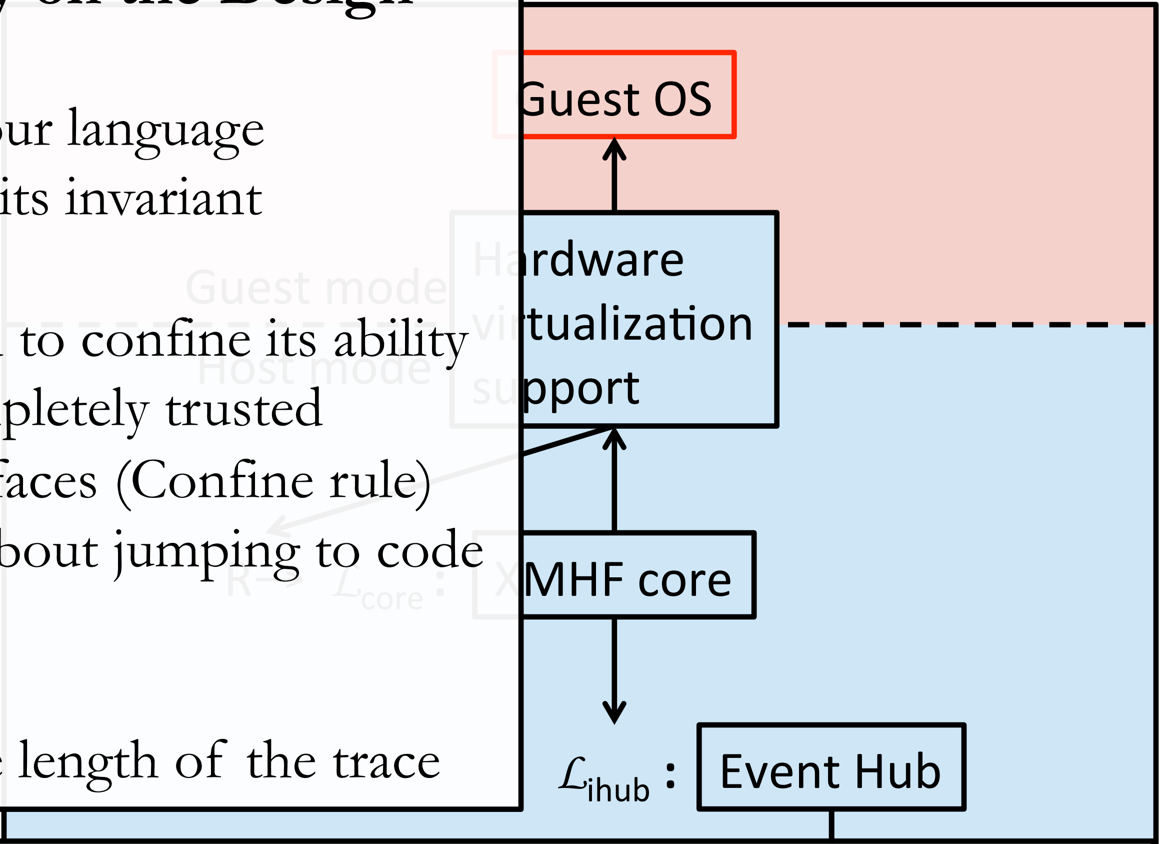
at time Ub, thread j is about to run c
at time Ue, c has not returned



Outline

- Background
- Model
- Type system
- **Case study**

Case study: an extensible hypervisor

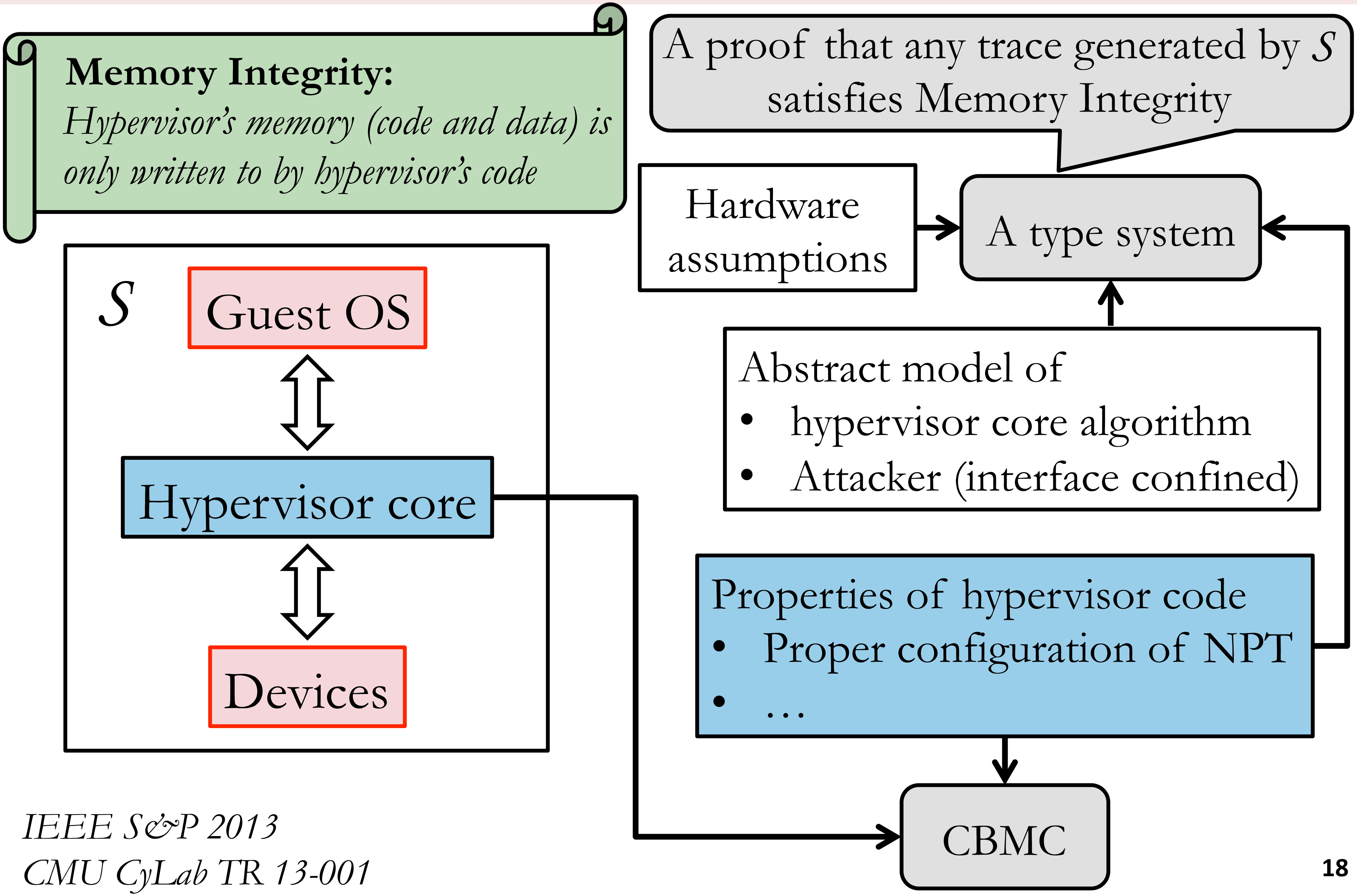


Verified Memory Integrity on the Design

- Core is trusted
  - Encode the algorithm in our language
  - Use type system to derive its invariant
- Guest OS is untrusted
  - Hardware axioms are used to confine its ability
- Event Handlers are not completely trusted
  - Confined to a set of interfaces (Confine rule)
- Beta rule is used to reason about jumping to code locations
  - L_{Core}, and L_{iHub}
- Inductive reasoning over the length of the trace

$$\begin{array}{l} \text{Axioms, start(tCore, eCore, } T_0\text{),} \\ \text{stype(eiHub, } \pi'\text{),} \\ \text{mem(} L_{Core}\text{, eCore, } T_0\text{),} \\ \text{mem(} L_{iHub}\text{, eiHub, } T_0\text{)} \end{array} \vdash \begin{array}{c} \forall l,v,u,\ u>T_0 \wedge \text{write}(i, l, v)@u \wedge \text{coreMem } l \\ \supset i = \text{tCore} \end{array}$$

Case study: a hypervisor core



Memory Integrity:
Hypervisor's memory (code and data) is only written to by hypervisor's code

A proof that any trace generated by *S* satisfies Memory Integrity

Hardware assumptions

A type system

S

Guest OS

Hypervisor core

Devices

Abstract model of
- hypervisor core algorithm
- Attacker (interface confined)

Properties of hypervisor code
- Proper configuration of NPT
- …

CBMC

IEEE S&P 2013
CMU CyLab TR 13-001

Summary

- **Designed a type system for reasoning about trace properties of systems that contain adversarial components**
  - Monad
  - Confine and beta rules
- **Defined trace semantics for types**
- **Proved soundness**
- **Verified the algorithm of an extensible hypervisor**