MultiNyx: A Multi-Level Abstraction Framework for Systematic Analysis of Hypervisors

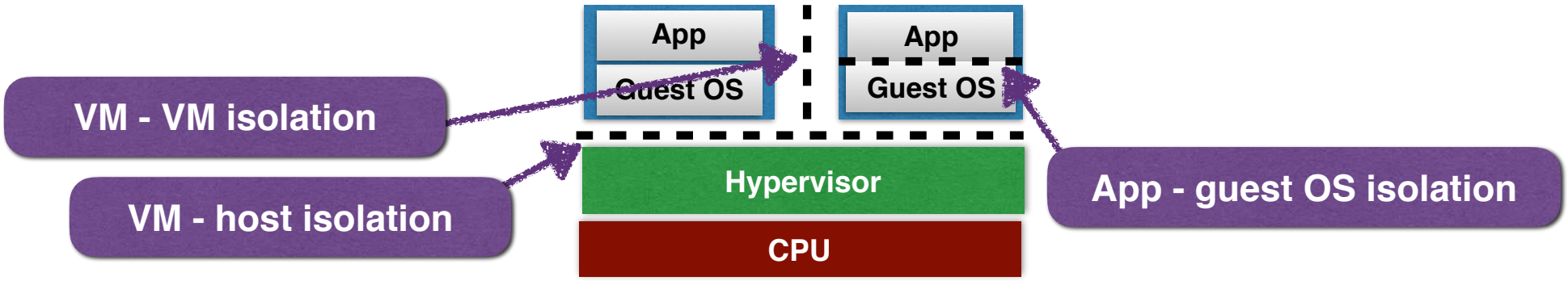
Pedro Fonseca, Xi Wang, Arvind Krishnamurthy

UNIVERSITY of WASHINGTON

PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

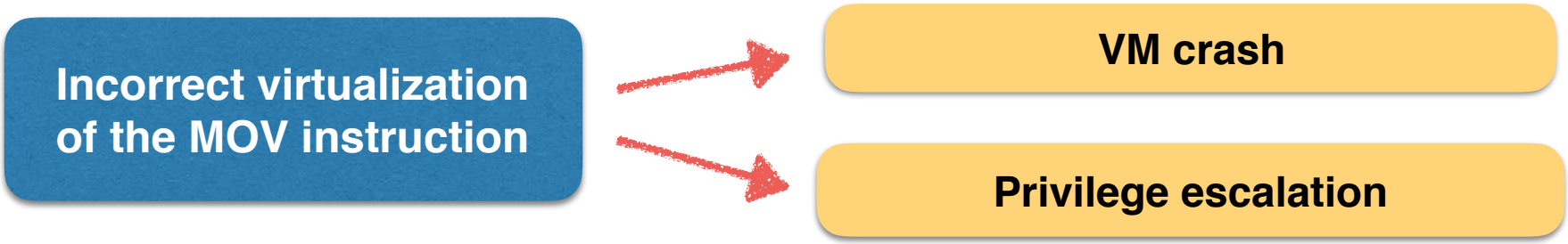
Hypervisor correctness is critical

- Hypervisors need to virtualize correctly all the architecture details
- Hypervisor bugs cause applications to crash, information leakage, etc.



Modern hypervisors rely on **CPU virtualization extensions**

KVM bug (CVE-2017-2583)



- Several conditions are required to trigger the bug:
  - MOV has to be emulated by the VMM
  - MOV has to load a NULL stack segment
  - Had to be executed in long mode and with CPL=3
  - Other privilege related fields had to have specific values (SS.RPL=3, SS.DPL=3)

Hard for fuzzing techniques to find such corner cases

How to effectively test hypervisors?

MultiNyx: Systematic testing of modern hypervisors

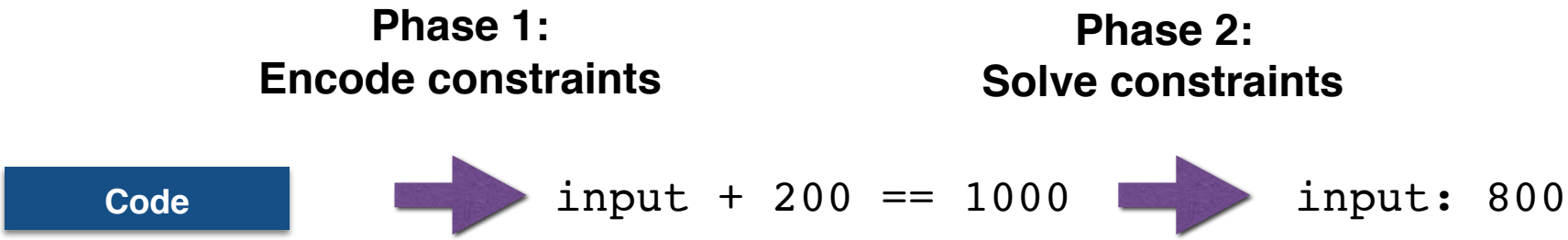
1. How to systematically generate test cases?

2. How to analyze the test case results?

Background: symbolic execution

Goal: find inputs that explore different paths

```
function start(input){
  x = input + 200;
  if(x == 1000){
    assert(0); // Crash
  }
  return;
}
```



Symbolic execution of hypervisors

Hypervisors use complex, system-level instructions

```
function hypervisor(input){
  x = VMENTER(input);
  if(x == 1000){
    assert(0); // Crash
  }
  return;
}
```

Virtualization instruction

How to encode complex instructions?

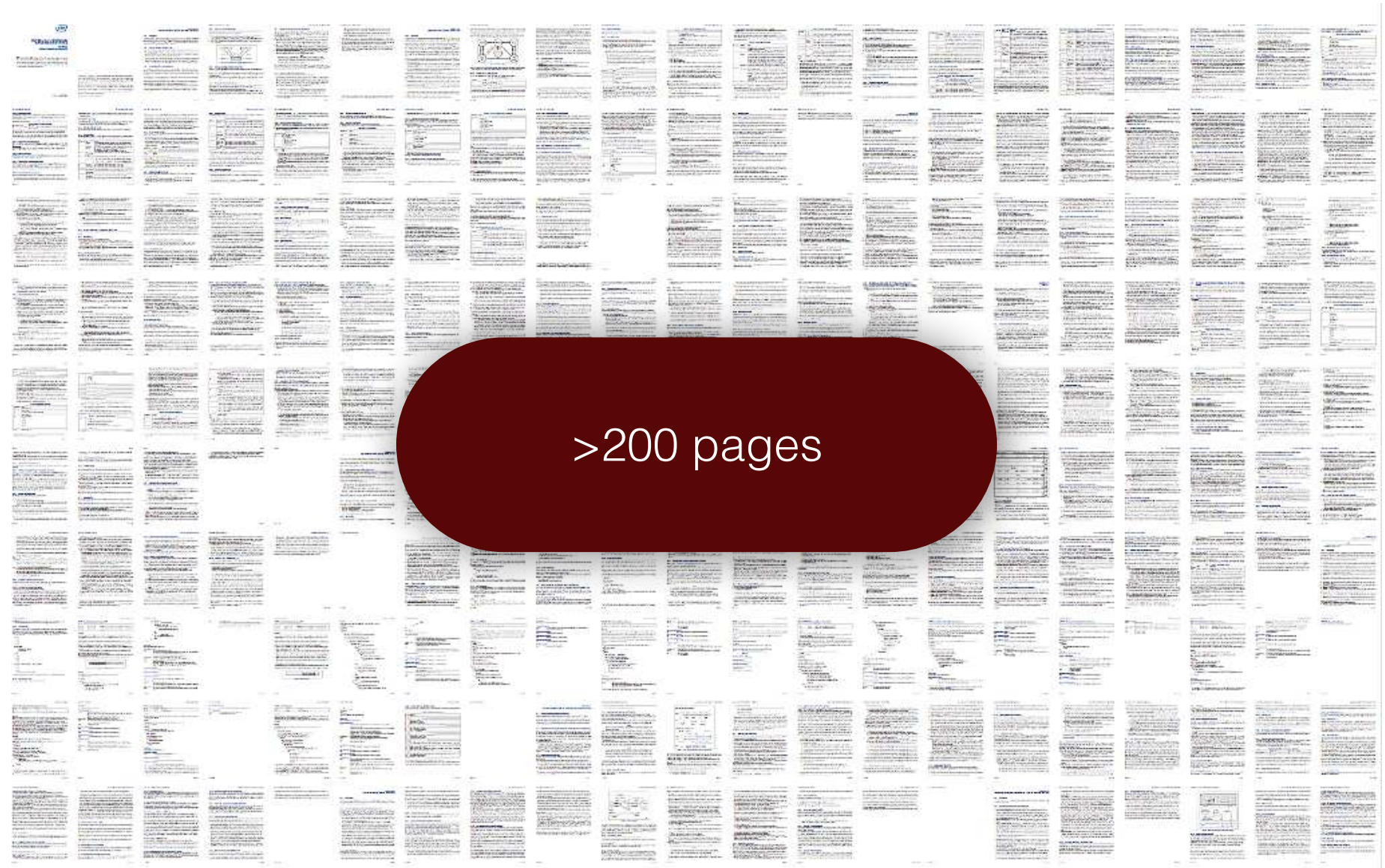
Phase 1:
Encode constraints

- Limit fields for CS, SS, DS, ES, FS, GS. If the guest will be virtual-8086, the field must be 0000FFFFH.
- Access-rights fields.
  - CS, SS, DS, ES, FS, GS.
    - If the guest will be virtual-8086, the field must be 000000F3H. This implies the following:
      - Bits 3:0 (Type) must be 3, indicating an expand-up read/write accessed data segment.
      - Bit 4 (S) must be 1.
      - Bits 6:5 (DPL) must be 3.
      - Bit 7 (P) must be 1.
      - Bits 11:8 (reserved), bit 12 (software available), bit 13 (reserved/L), bit 14 (D/B), bit 15 (G), bit 16 (unusable), and bits 31:17 (reserved) must all be 0.
    - If the guest will not be virtual-8086, the different sub-fields are considered separately:
      - Bits 3:0 (Type).
        - CS. The values allowed depend on the setting of the "unrestricted guest" VM-execution control:
          - If the control is 0, the Type must be 9, 11, 13, or 15 (accessed code segment).
          - If the control is 1, the Type must be either 3 (read/write accessed expand-up data segment) or one of 9, 11, 13, and 15 (accessed code segment).
        - SS. If SS is usable, the Type must be 3 or 7 (read/write, accessed data segment).
        - DS, ES, FS, GS. The following checks apply if the register is usable:
          - Bit 0 of the Type must be 1 (accessed).
          - If bit 3 of the Type is 1 (code segment), then bit 1 of the Type must be 1 (readable).
      - Bit 4 (S). If the register is CS or if the register is usable, S must be 1.
      - Bits 6:5 (DPL).
        - CS.
          - If the Type is 3 (read/write accessed expand-up data segment), the DPL must be 0. The Type can be 3 only if the "unrestricted guest" VM-execution control is 1.
          - If the Type is 9 or 11 (non-conforming code segment), the DPL must equal the DPL in the access-rights field for SS.
          - If the Type is 13 or 15 (conforming code segment), the DPL cannot be greater than the DPL in the access-rights field for SS.
        - SS.
          - If the "unrestricted guest" VM-execution control is 0, the DPL must equal the RPL from the selector field.
          - The DPL must be 0 either if the Type in the access-rights field for CS is 3 (read/write accessed expand-up data segment) or if bit 0 in the CR0 field (corresponding to CR0.PE) is 0.[1]
        - DS, ES, FS, GS. The DPL cannot be less than the RPL in the selector field if (1) the "unrestricted guest" VM-execution control is 0; (2) the register is usable; and (3) the Type in the access-rights field is in the range 0 – 11 (data segment or non-conforming code segment).
      - Bit 7 (P). If the register is CS or if the register is usable, P must be 1.

1 page of the Intel manual

tiny subset of the checks for the `VMENTER` instruction

Semantics of the virtualization instructions



MultiNyx approach: leverage a simulator

```
function hypervisor(input){
  x = VMENTER(input);
  if(x == 1000){
    assert(0); // Crash
  }
  return;
}
```

CPU Simulator

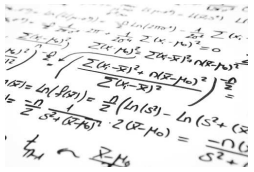
```
function VMENTER(input){
  x = input;
  x = x + 4
  …
  return x;
}
```

Phase 1:
Encode constraints

Phase 2:
Solve constraints

Hypervisor

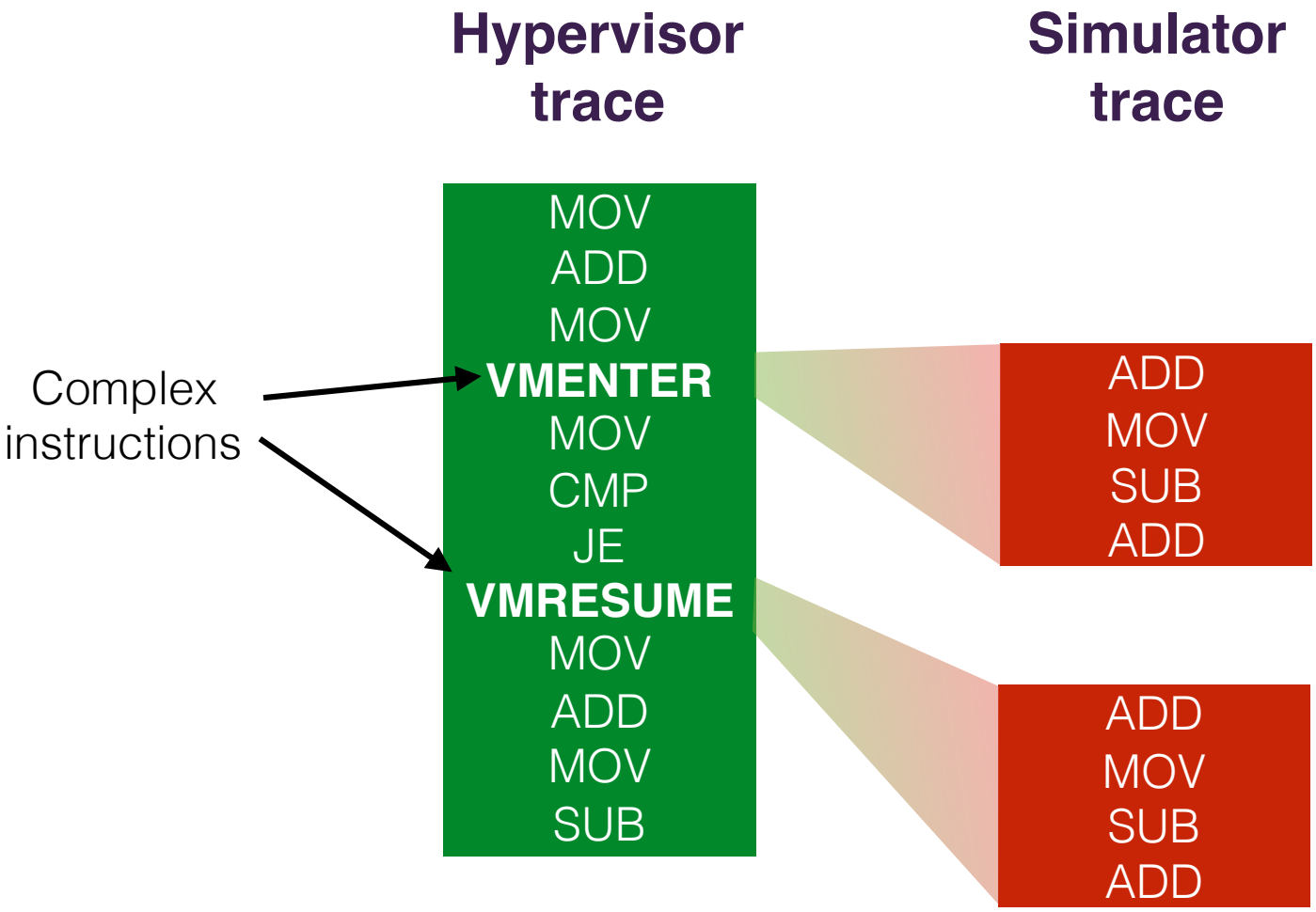
CPU simulator



Path constraints

Test cases

MultiNyx: multi-level symbolic execution



MultiNyx: multi-level symbolic execution

Multi-level trace

Only contains simple instructions

Challenge: Different abstractions have different state representations

MultiNyx converts between different state representations on each transition

Scaling symbolic execution to hypervisors

- Traditional tests execute millions of VM instructions
- Key observation: Hypervisor interface allows externally setting the initial VM state

MultiNyx: each test executes a **single VM instruction**

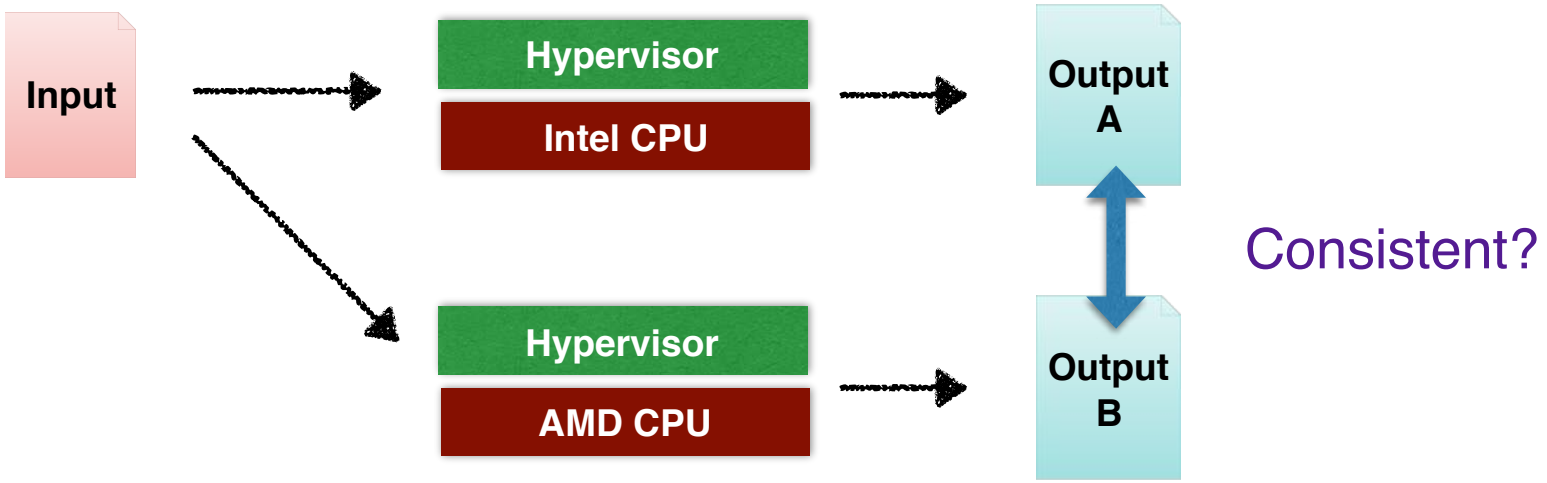
1. Set the initial VM state
2. Run a single VM instruction
3. Get the final VM state

MultiNyx: Systematic testing of modern hypervisors

1. How to systematically generate test cases?

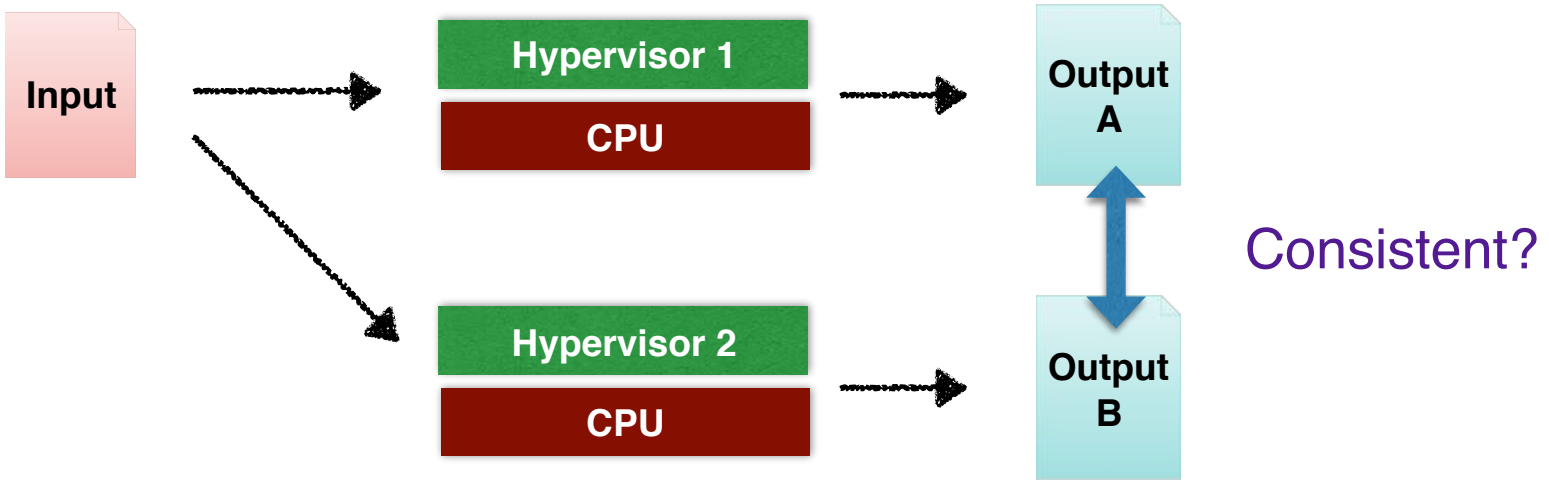
2. How to analyze the test case results?

How to analyze the test cases results?



Run the same test on different configurations

How to analyze the test cases results?



Run the same test on different configurations

MultiNyx implementation

- Symbolic execution engine: Triton + Z3
- Executable specification: Bochs simulator

Component	Language	LOCs
KVM driver	C	2,400
KVM annotations	C	1,400
Low-level trace recording	C++	600
High-level trace recording	C++	1,300
Multi-level analysis	C++ / Python	3,100
Diff. testing and diagnosis	Bash / Python	4,400

Testing with MultiNyx

- +200,000 tests automatically generated for KVM
- MultiNyx coverage is +8% higher than fuzzing
- MultiNyx tests revealed 739 mismatching tests

Example of KVM bug found by MultiNyx

- Incorrect update of `%SP` register (2 bytes instead of 4 bytes)
  - And incorrect update of the VM memory
- Instruction `PUSH %ES`
  - EPT option disabled
  - Segment registers initialized with specific values
  - Execution in real mode
- Bug we reported has been fixed in the latest KVM

MultiNyx:
Systematic testing of modern hypervisors

Modern hypervisor rely on complex instructions