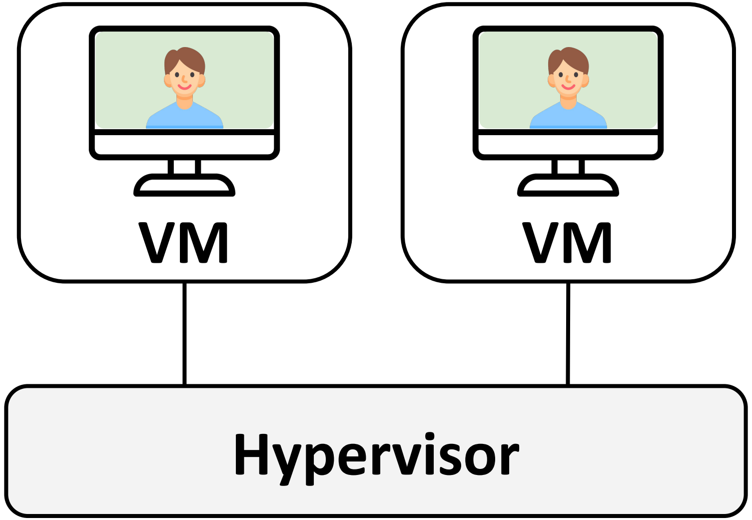
MundoFuzz: Hypervisor Fuzzing with Statistical Coverage Testing and Grammar Inference

Cheolwoo Myung[†], Gwangmu Lee[‡], and Byoungyoung Lee[†]

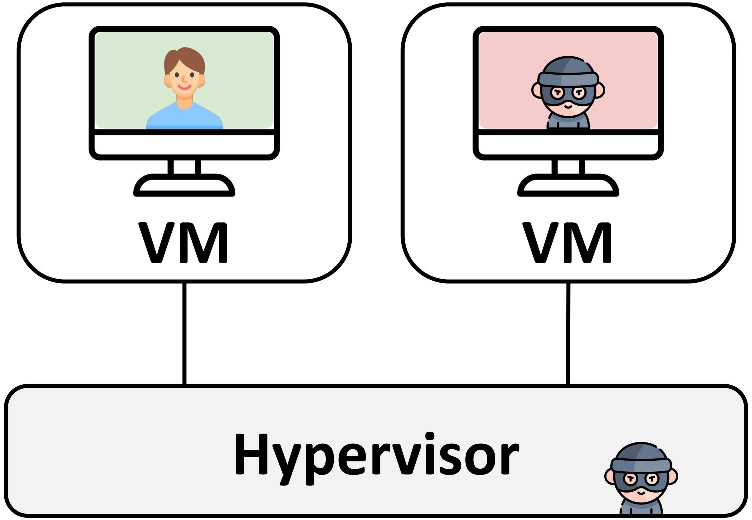
Seoul National University[†], EPFL[‡]

Hypervisor: Manager of Virtual Machine



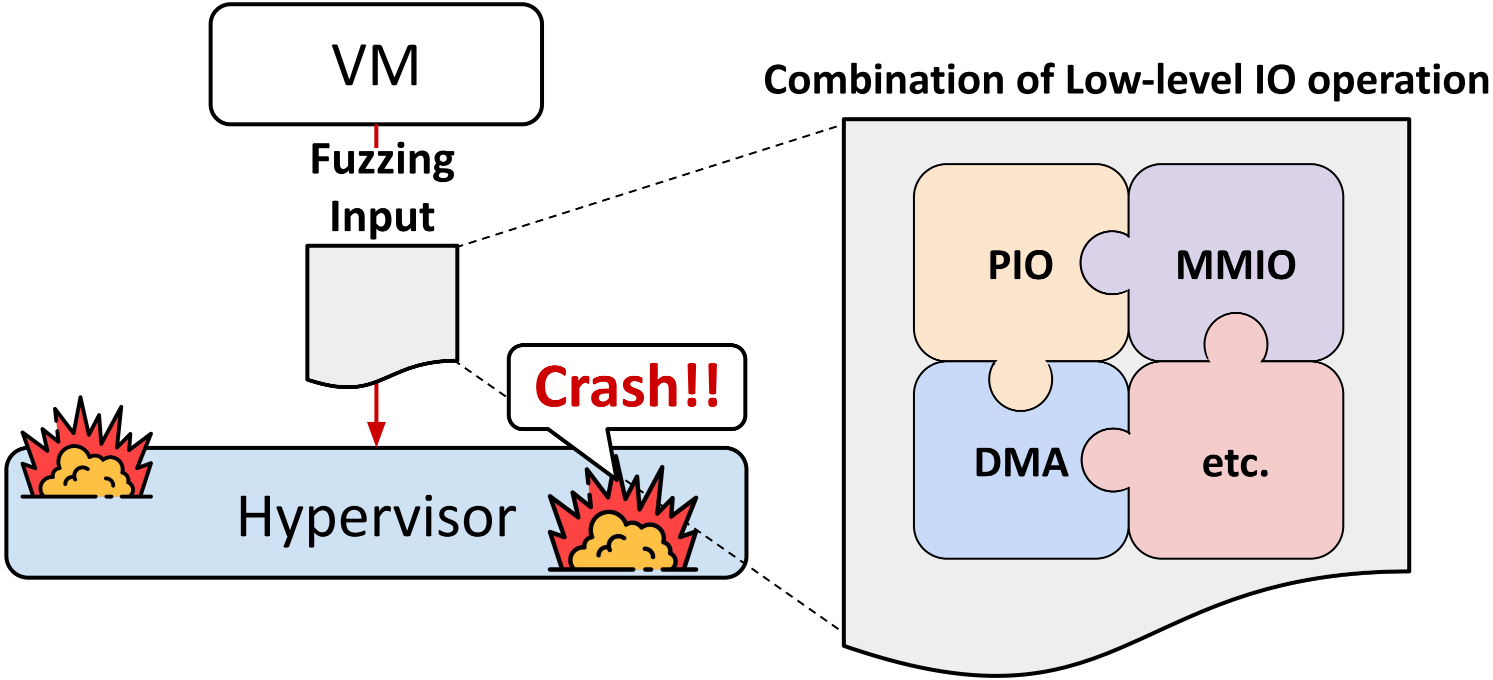
- Allow **remote users** to run guest VMs

Hypervisor can be attacked by Malicious VM



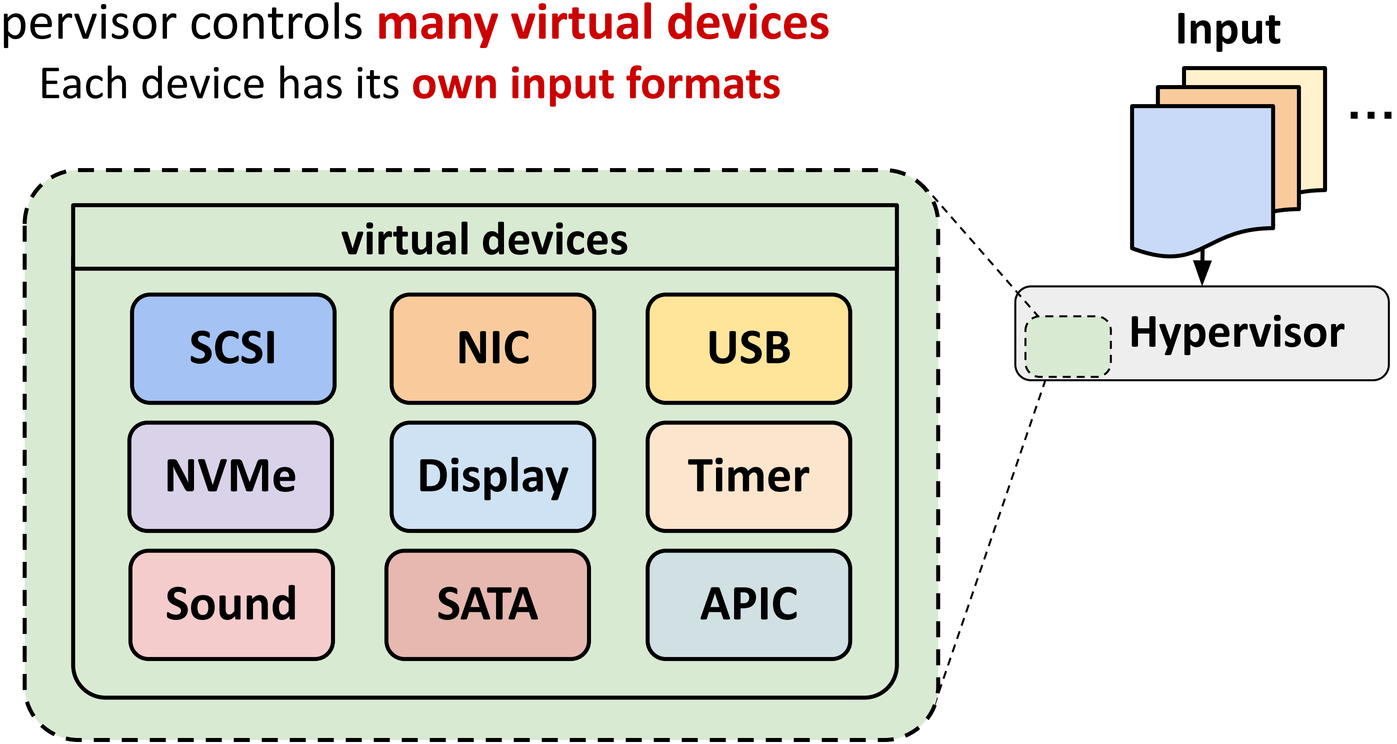
- One of guest VMs can be **malicious**

Fuzzing: Feed Random Inputs to Hypervisor



Motivation: Too many devices, too many formats

- Hypervisor controls **many virtual devices**
  - Each device has its **own input formats**



Limitations of Current Hypervisor Fuzzing

#1. Generating **random inputs** per device

Limitation ⇒ **Cannot explore deep states of the devices**

#2. Relying on **manual input grammars** per device

Limitation ⇒ **Require unacceptable manual work to specify grammar rules**

Let's fuzz hypervisor with grammar-awareness using automatic grammar inference!

Overview of MundoFuzz

- Augment hypervisor fuzzing capability with automatic grammar inference

- **Challenges** in inferring hypervisor grammars
  - #1. Hypervisor grammars have **hidden input semantics** per device
  - #2. Hardware features of hypervisor introduce **coverage noises**

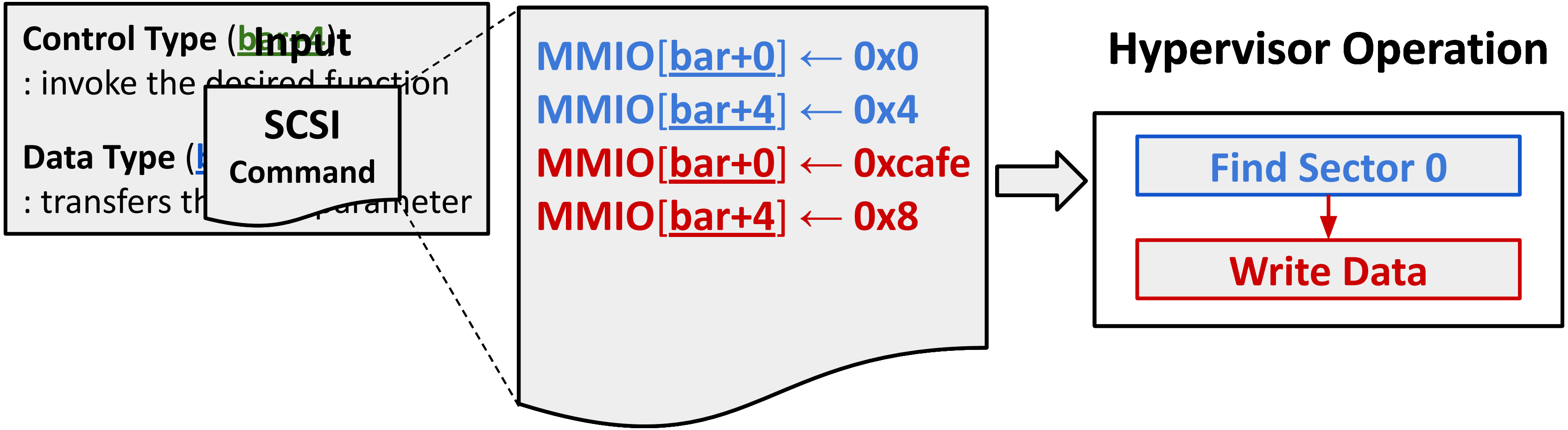
- **Our approach**
  - **Statistical and differential learning with coverage**

Challenge 1: Hidden Input Semantics

- Too difficult to infer **hidden input semantics** behind the hypervisor input
  - IO address semantics: correct semantic
  - **IO**

Invoke the "Write Data" func. (0x8)

"Find Sector" should be performed before "Write Data"



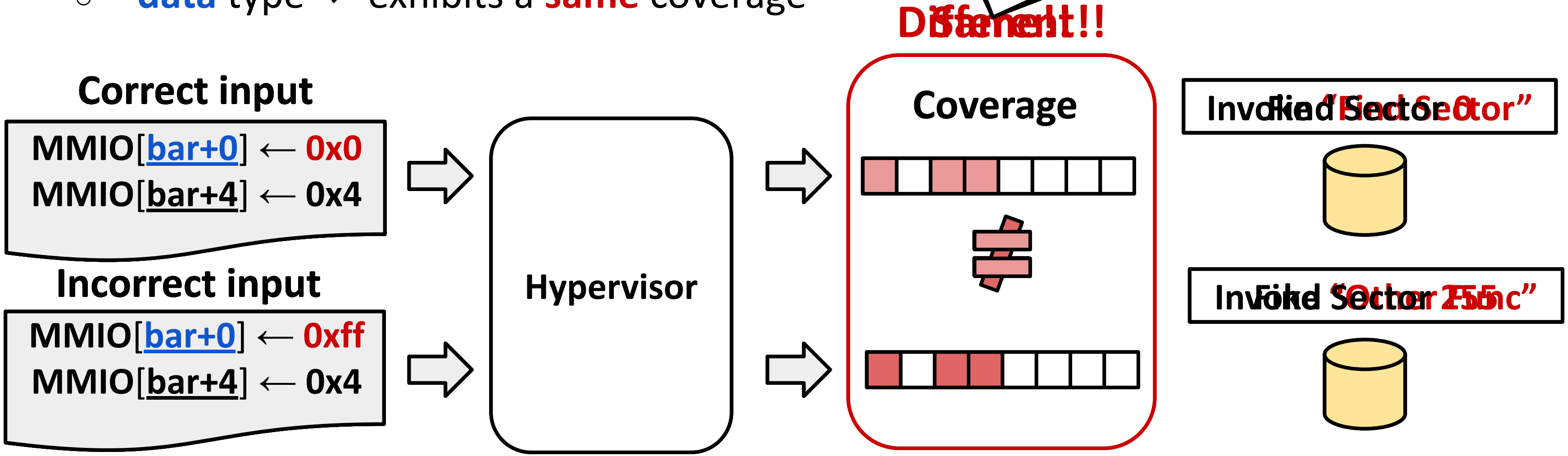
Example: SCSI command input

Solution 1: Differential Learning on Input Semantics

#1. IO address semantics

- **Different IO address types react to IO addr tly**
  - control type ⇒ exhibits a **different** coverag
  - data type⇒ exhibits a **same** coverage

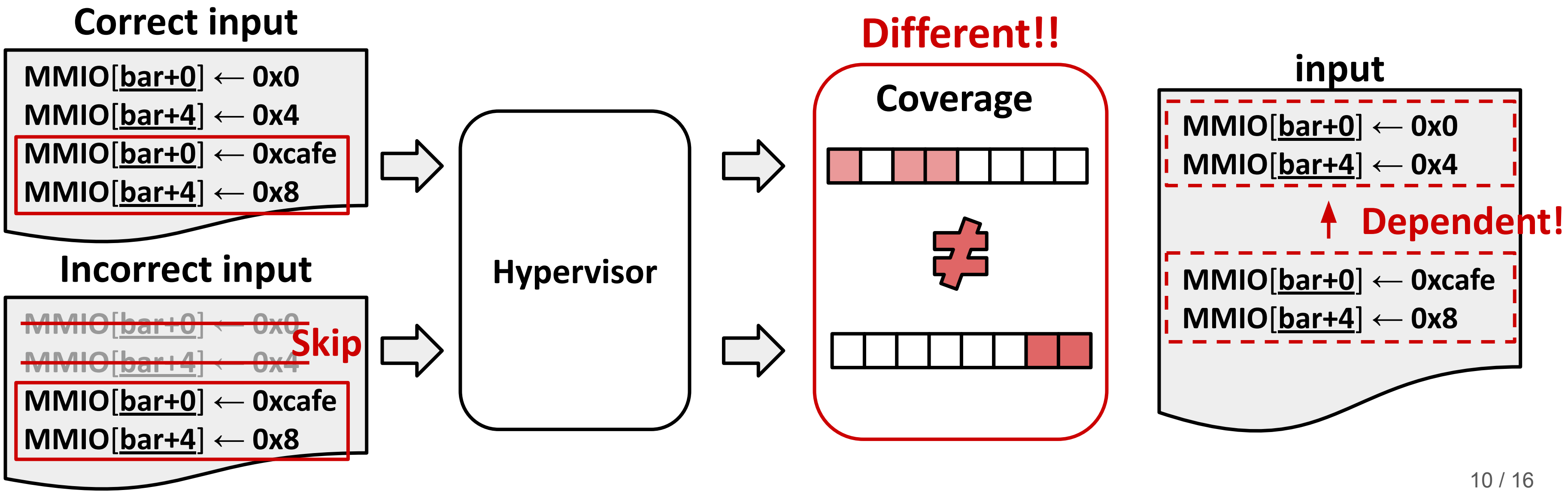
Data Type!



Solution 1: Differential Learning on Input Semantics

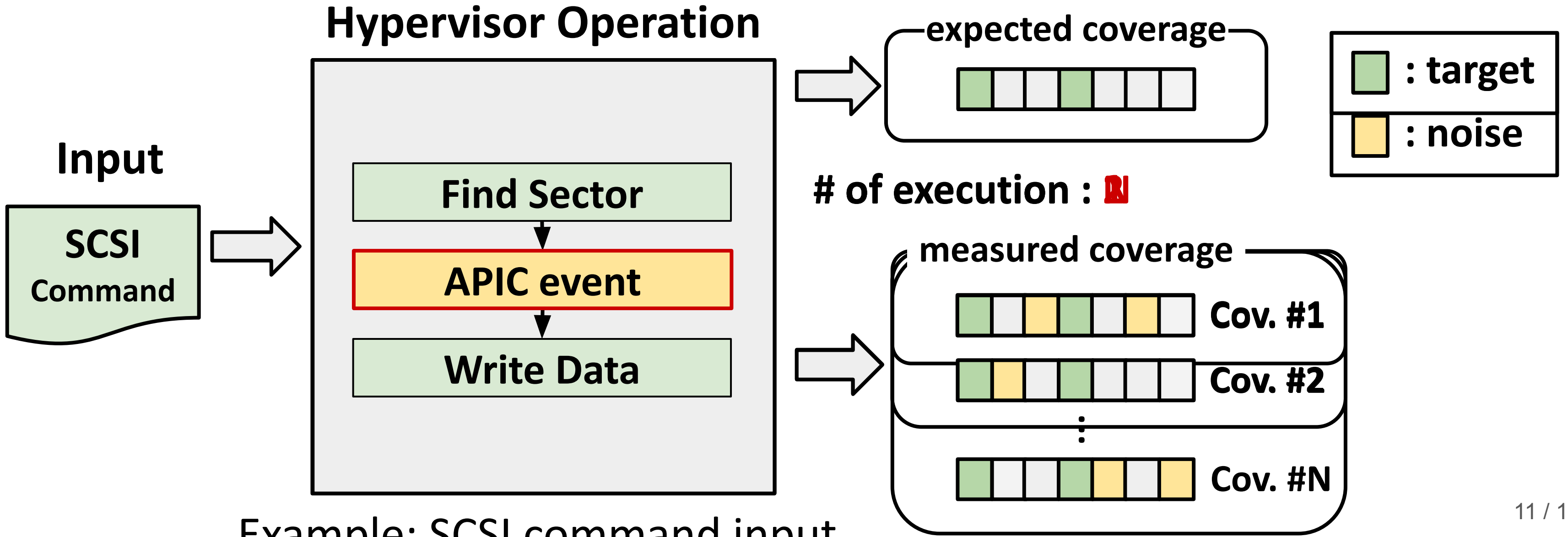
#2. IO order semantics

- **IO operations wouldn't work correctly without prerequisite IO operations**
  - **absence of IO operations ⇒** may distort **some following coverage**



Challenge 2: Coverage Noises

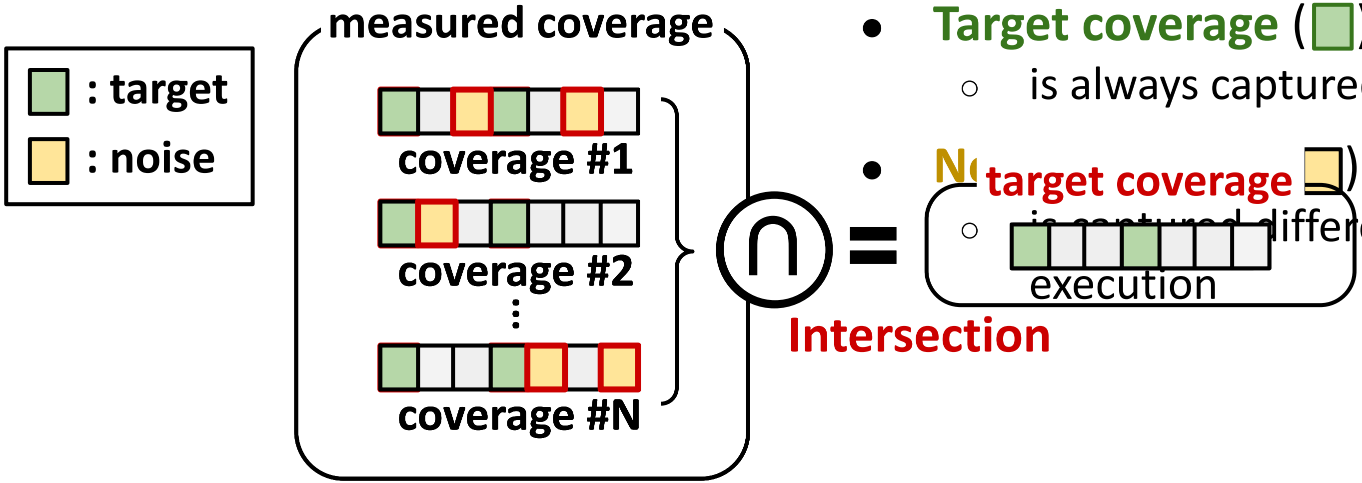
- The measured input coverage includes **unwanted coverage**
  - due to the asynchronous event handling (e.g., timer, interrupt event)
  - asynchronous event introduces **non-deterministic (noise) coverage**



Example: SCSI command input

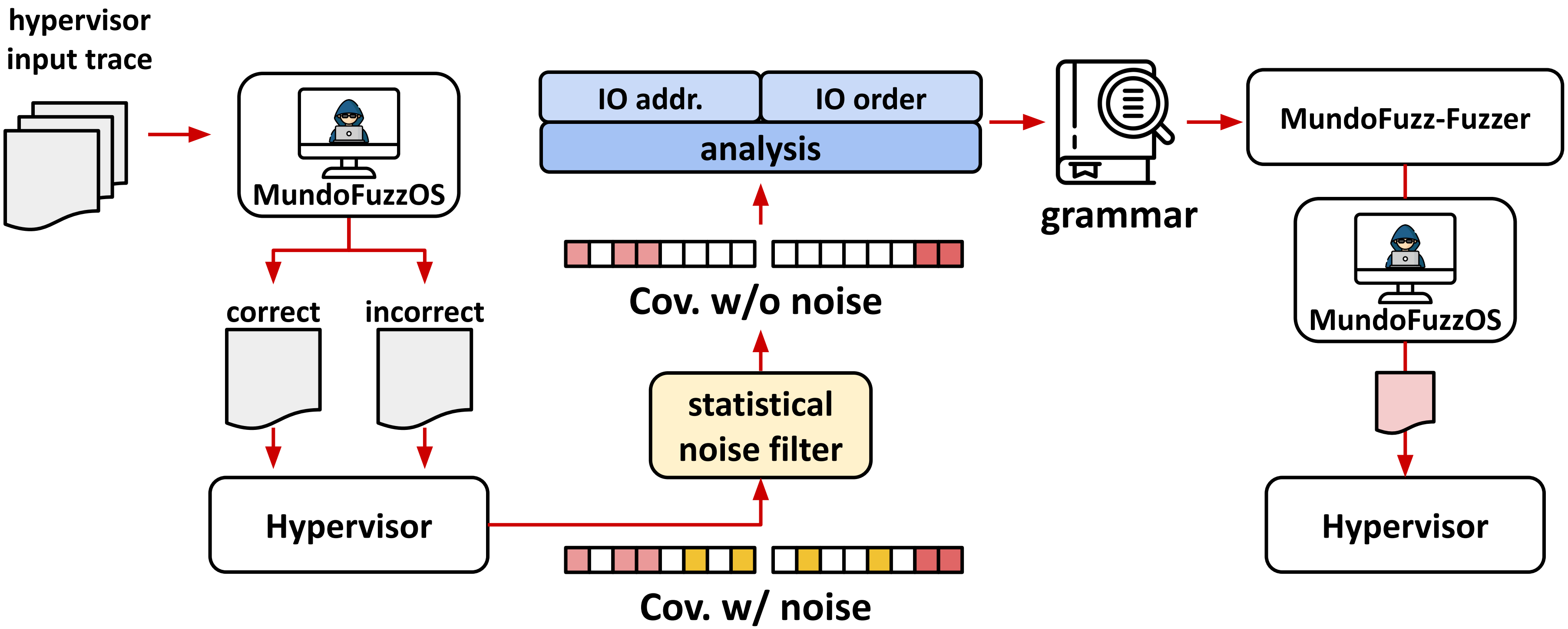
Solution 2: Statistical Differential Coverage Measurement

- Remove noise coverage **by intersecting all measured coverages**
  - the result only contains **target coverage**



- **Target coverage** ( )
  - is always captured for all execution
- **Non-target coverage** ( )
  - is captured differently for each execution

Architecture of MundoFuzz



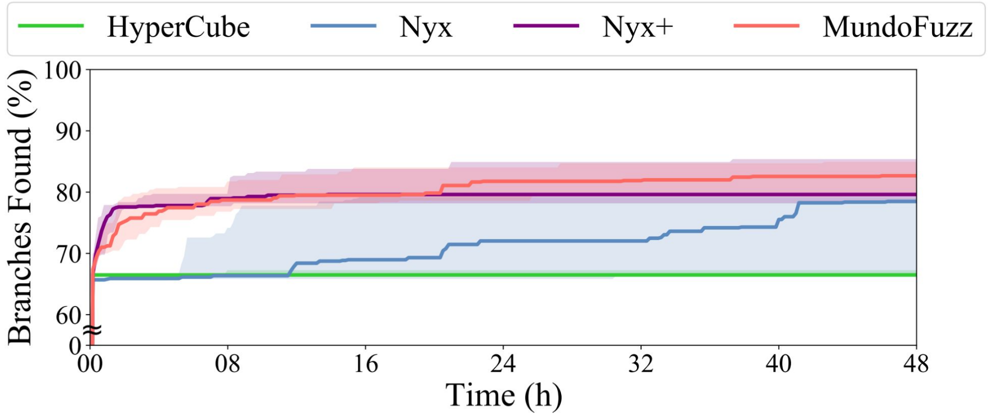
What MundoFuzz Found?

- MundoFuzz found new 40 bugs in QEMU and Bhyve
  - 23 bugs in QEMU
  - 17 bugs in Bhyve
  - 9 of these were acknowledged as CVEs

Hypervisor	Bug Types	Numbers
QEMU	Use-after-free	3
	Heap Overflow	2
	Segmentation Fault	3
	Infinite Loop	3
	Stack Overflow	1
	Assertion	11
Bhyve	Segmentation Fault	4
	Floating Point Exception	1
	Assertion	12

Our result

- Overall coverage: MundoFuzz outperforms state-of-art hypervisor fuzzer
  - HyperCube: **+4.91%**
  - Nyx: **+6.60%**
- MundoFuzz shows higher coverage than Nyx+ (with manual grammar rule)
  - for **USB-XHCI device** (48 hours)



Conclusion

- Proposed MundoFuzz, a hypervisor fuzzing technique
  - statistically removes noise coverage in raw coverage
  - automatically learns the grammar using two hidden semantics
- MundoFuzz discovered 40 new bugs (including 9 CVEs)
- MundoFuzz presented better coverage, compared to state of the arts.

Thank you!

Q & A

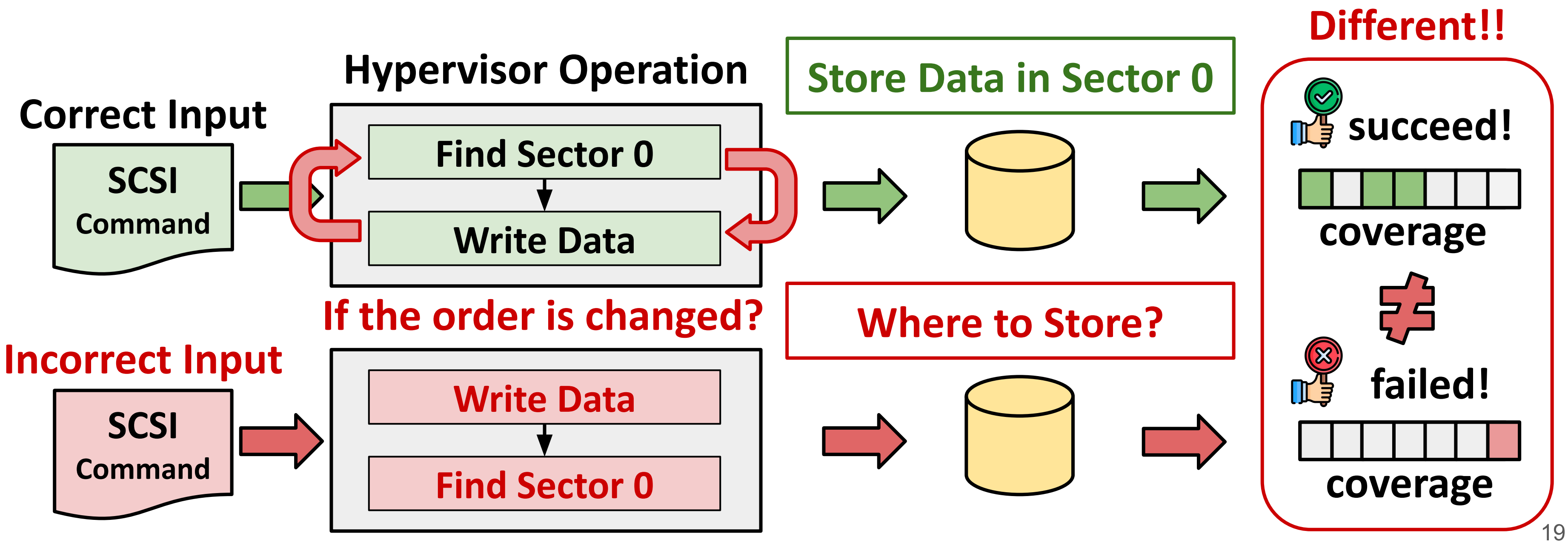
Contact Cheolwoo Myung
Ph.D. Student at Seoul National University (SNU)
cwmyung@snu.ac.kr

Our approach: Infer the grammar with semantic constraints

- **MundoFuzz infers the semantic constraints by the input coverage**
  - **Register types**
    - to synthesize the IO operations correctly
  - **Order dependency**
    - to place the IO operations in correct order

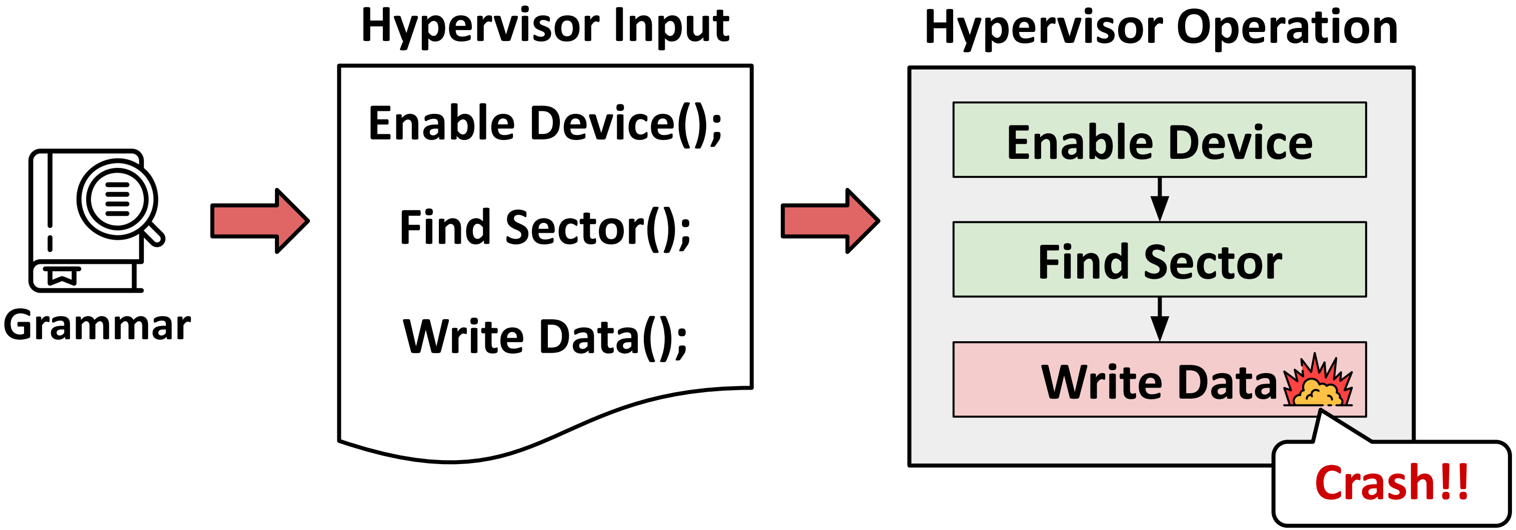
Idea: Inferring the grammar through input coverage

- Hypervisor behaves **differently** depending on the **input grammar correctness**



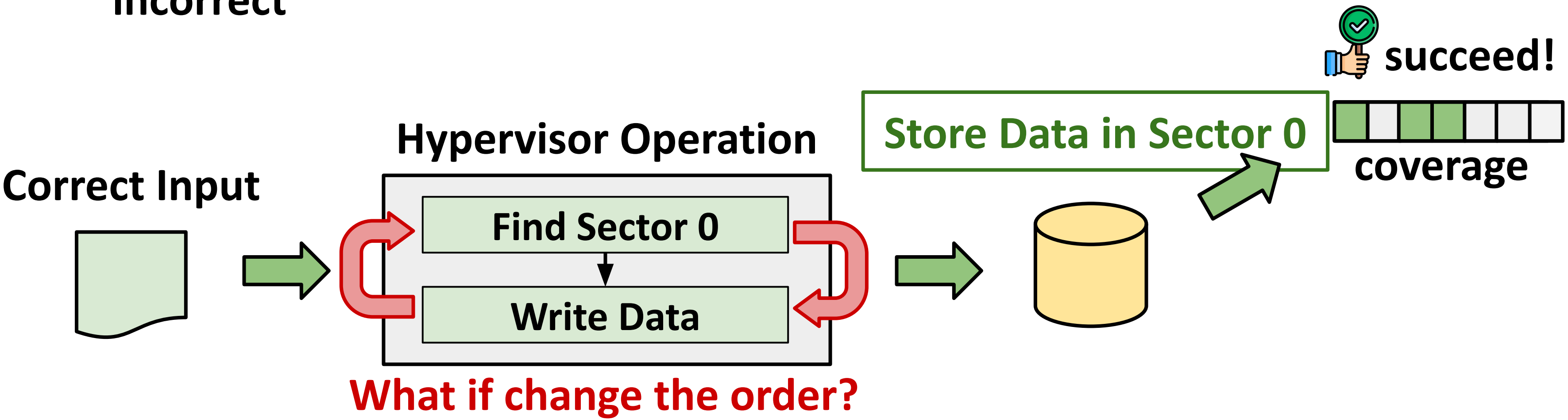
Let's fuzz the hypervisor with grammar-awareness!

- Synthesizes **correct input semantics** with **correct order**



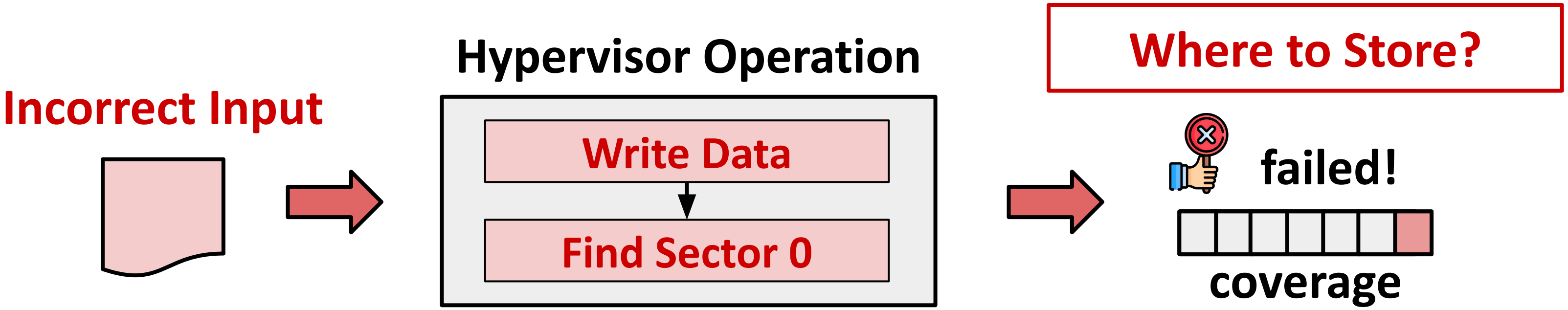
Our approach: Inferring the grammar through coverage

- Hypervisor behaves **differently** if the input is given **grammatically correct or incorrect**



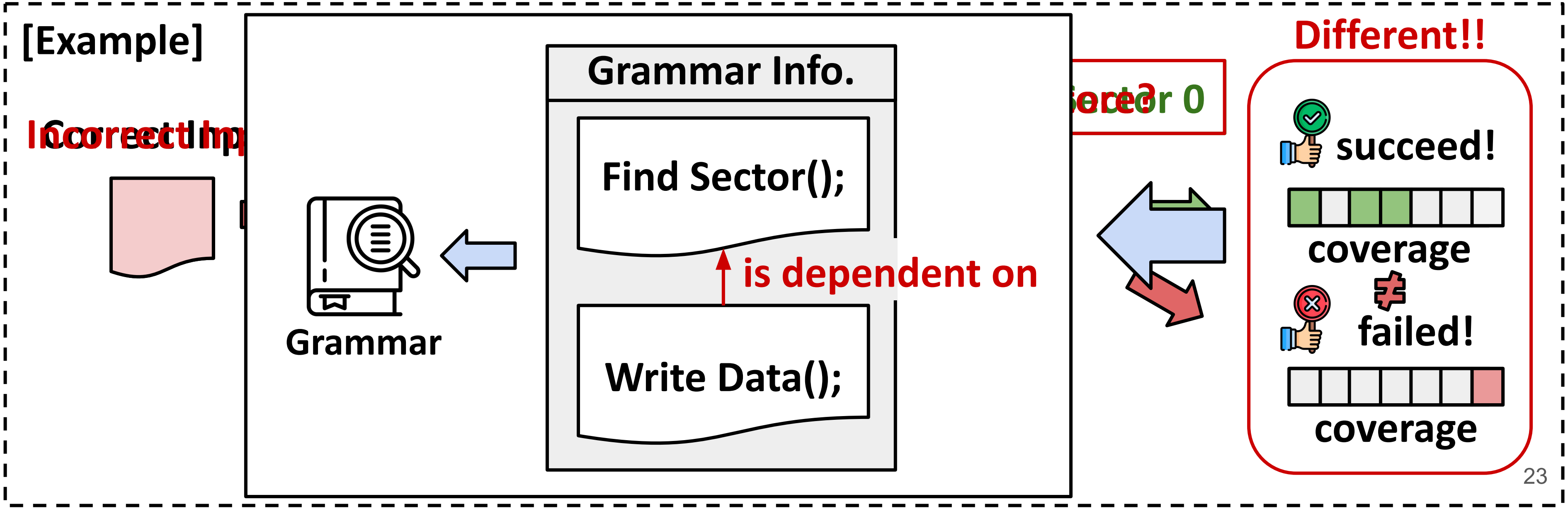
Our approach: Inferring the grammar through coverage

- Hypervisor behaves **differently** if the input is given **grammatically correct or incorrect**



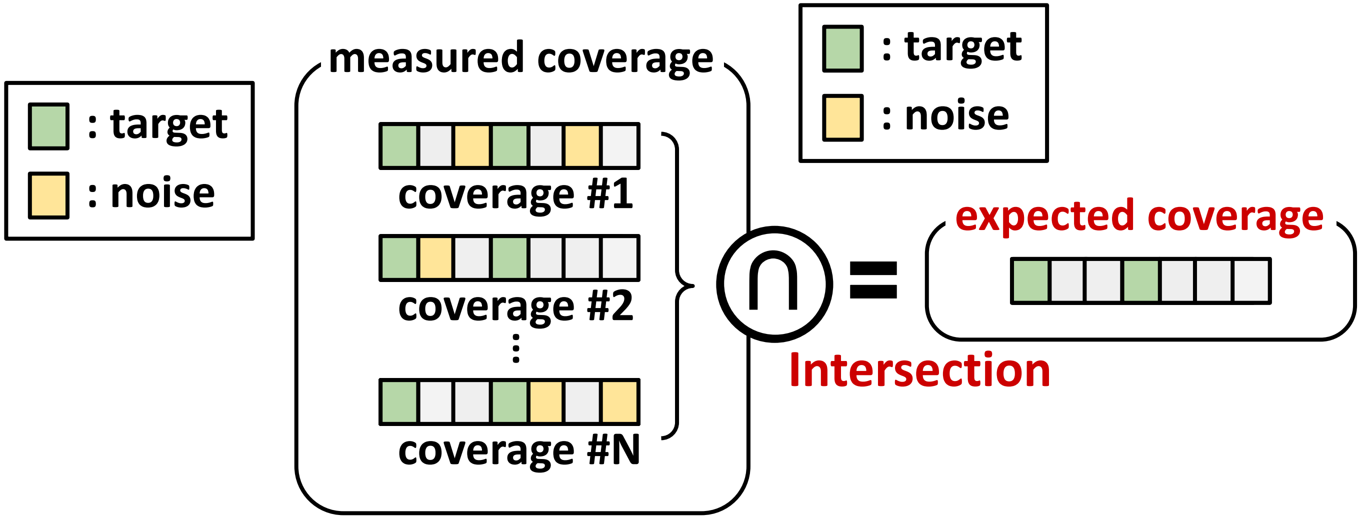
Our approach: Inferring the grammar through coverage

- Hypervisor behaves **differently** if the input is given **grammatically correct or incorrect**



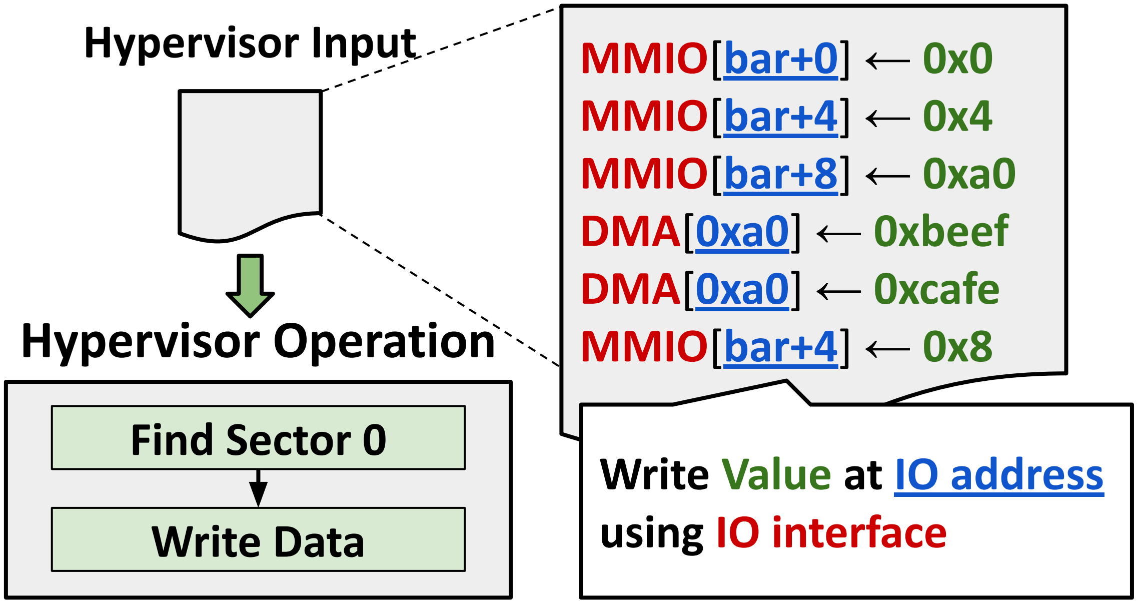
Idea 1: Noise coverage would appear in a different way

- Measure the coverage multiple times for same input
- Remove Noise coverage by intersecting them all



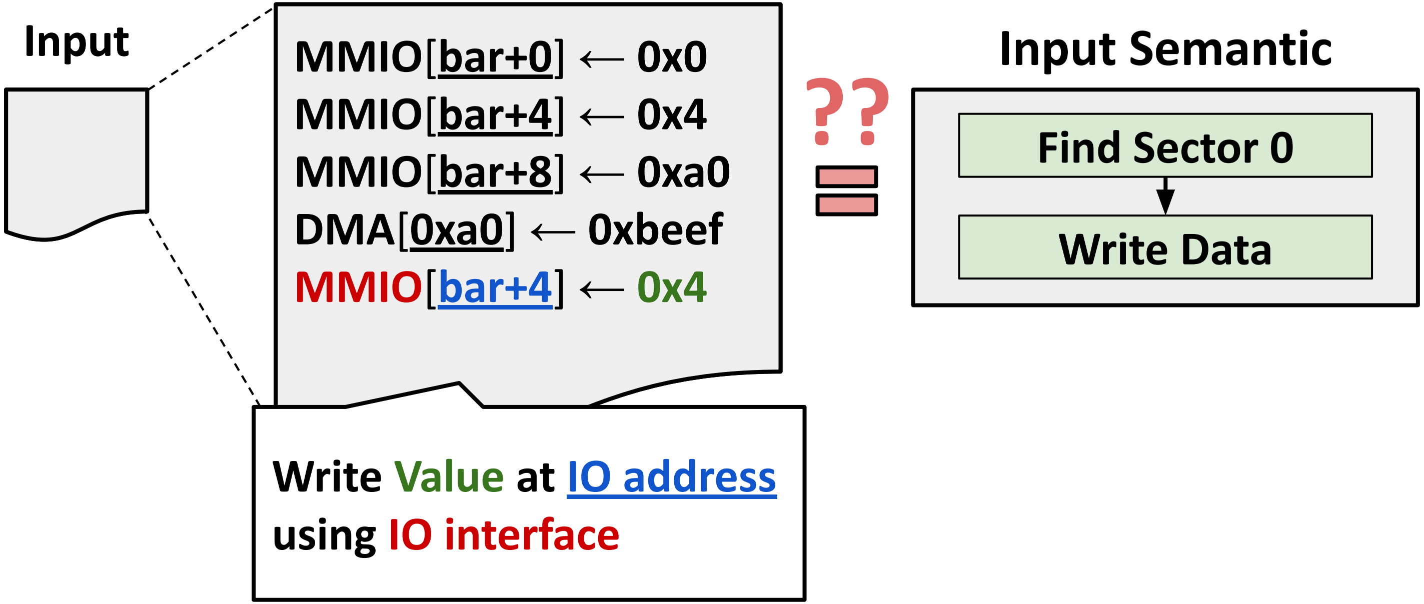
Problem 2: Uncertain Input Semantics

- Input semantics are presented in **a small sequence of IO interface inputs**
  - **hard to understand** by looking at individual IO interface input



Problem 2: Semantic meaning in Hypervisor Input

- Hypervisor input is presented in **a sequence of IO interface inputs**
  - hard to understand **its semantic meaning** by looking at individual IO interface input



Idea 2: Grammar Inference with Two Semantic Constraints

- The **Grammar** can be reconstructed by two semantic constraints

  1) The **register types** of IO address 2) **order dependency**

Register Types

Control register (bar+4)
: invoke the desired function

Data register (bar+0)
: transfers the data parameter

⇨

Find Sector 0

MMIO[bar+0] ← 0x0
MMIO[bar+4] ← 0x4

⇨

Hypervisor Operation

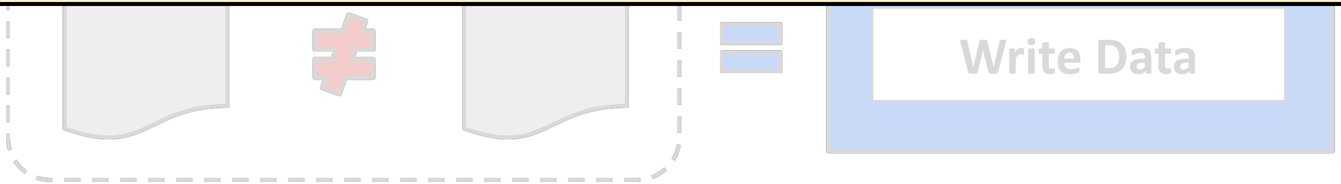
Find Sector 0

Invoke the “Find Sector” func.
with the parameter (“0”)

Example: too many devices, too many formats

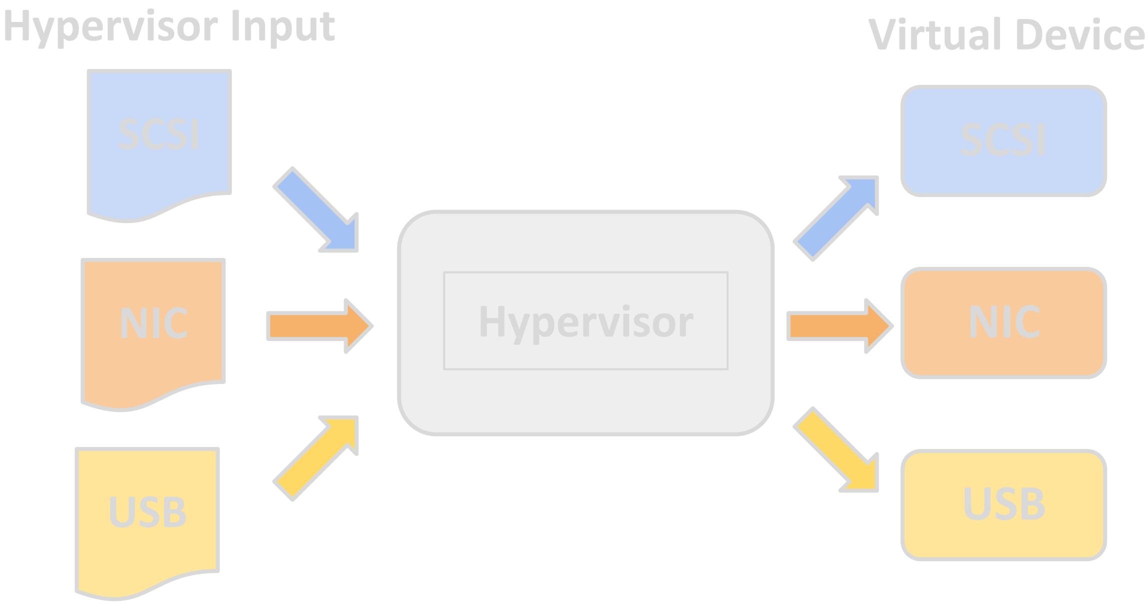
- Need different hypervisor inputs even if these behave same functionality

Random fuzzing cannot develop the hypervisor input regarding the input format



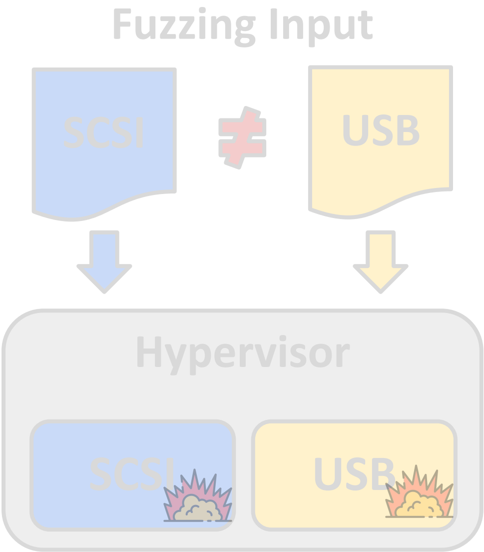
Example: too many devices, too many formats

- Need different hypervisor inputs to control each devices



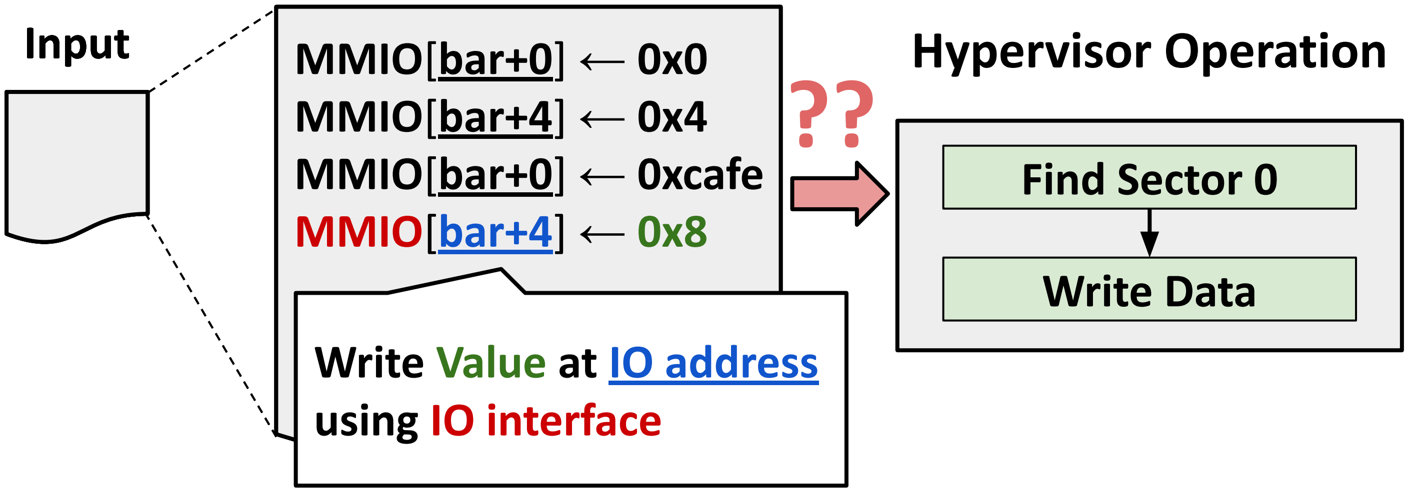
Example: too many devices, too many formats

- Need different input formats to fuzz each device in hypervisor



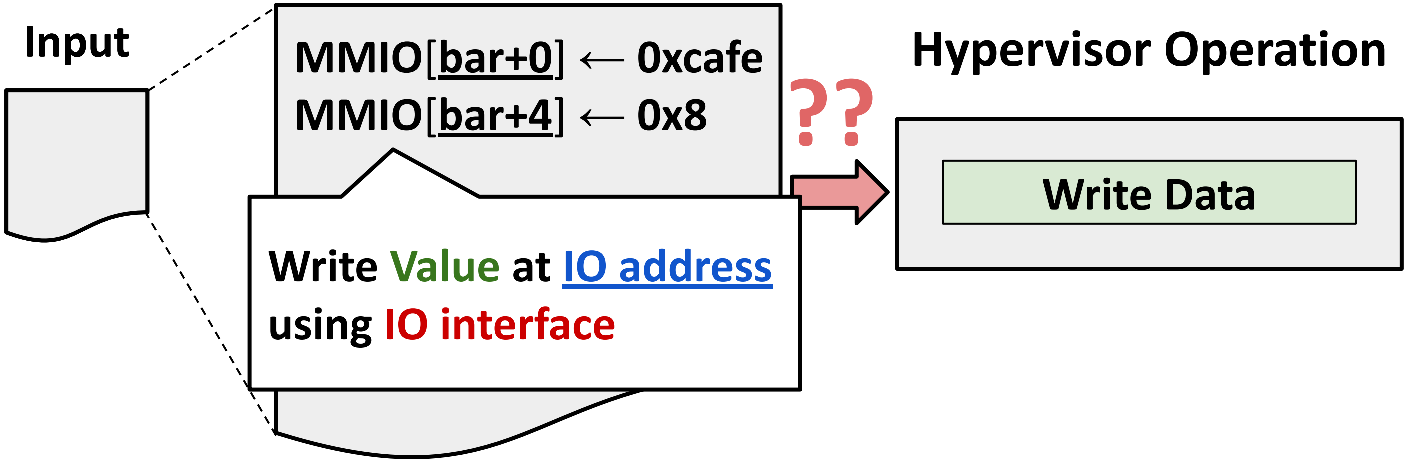
Challenge 2: Uncertain grammar information

- Hypervisor input is presented in **a sequence of IO interface inputs**
  - hard to understand **its grammar** by looking at individual IO interface input



Problem 2: Uncertain Input Semantics

- Hypervisor input is presented in **a sequence of IO interface inputs**
  - hard to infer **its semantic meaning** by looking at individual IO interface input

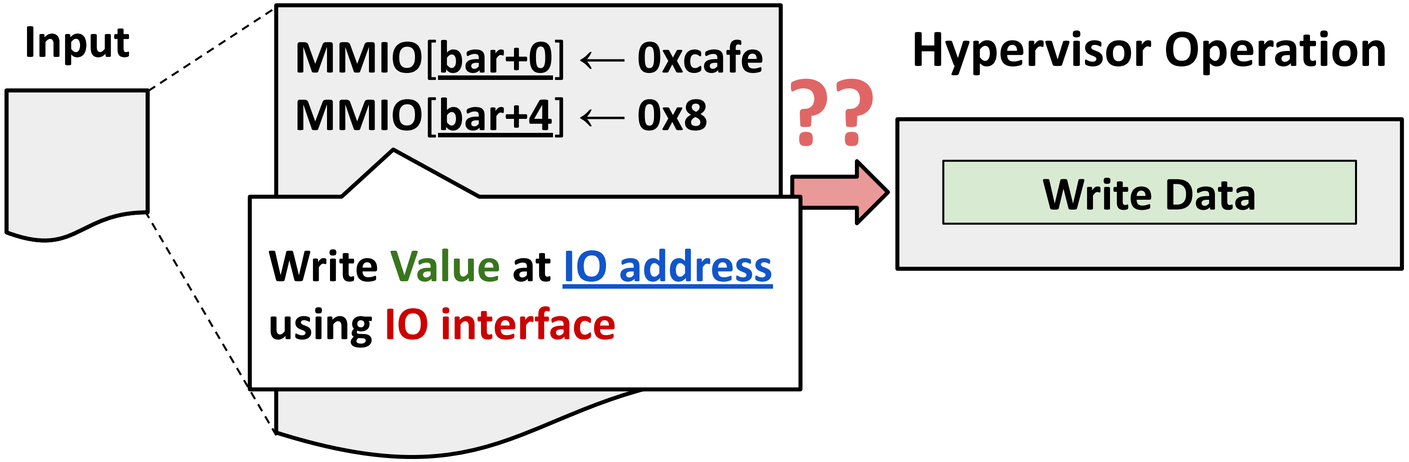


Idea 2: Subdivides IO Task using Completion Signal

- Input Semantics comprise a small sequence of IO interface inputs
    - **IO Task**: serves as high-level semantic unit

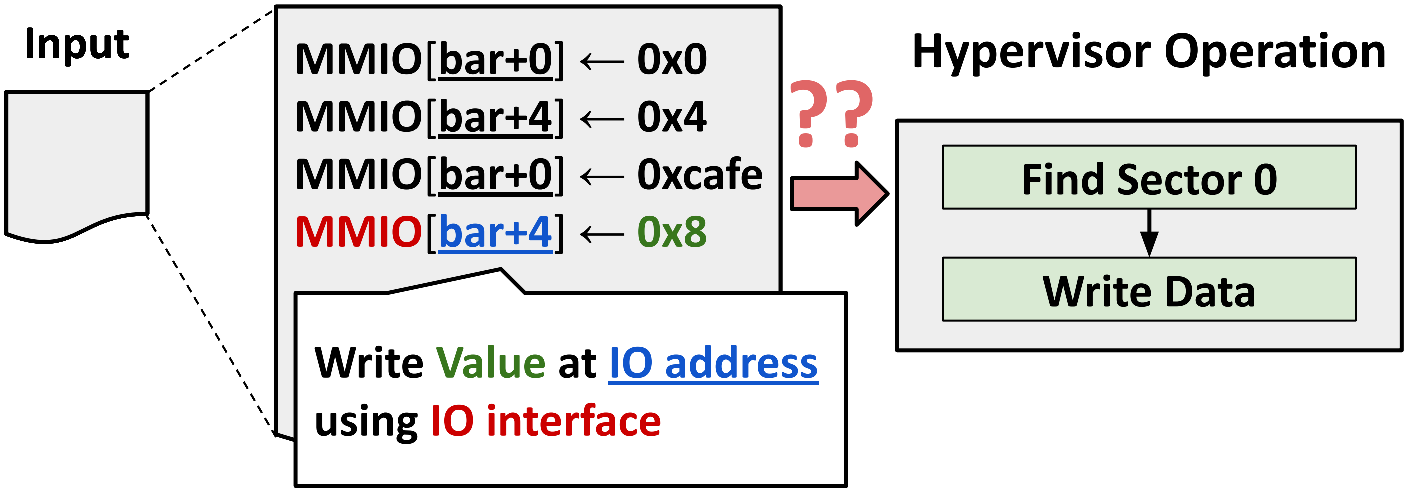
Problem 3: Uncertain Grammar Rules

- **With IO Task, still we have no grammar rules**
  - hard to infer **its grammar** by looking at individual IO interface input



Challenge 2: Uncertain grammar information

- Input semantics are presented in **a sequence of IO interface inputs**
  - hard to infer **its grammar** by looking at individual IO interface input



Idea 2: Grammar Inference with Constraints

#1. Types of register

- IO interface input serves its own semantic meaning depending on the types of register

Register Types

Control register (bar+4)
: invoke the desired function

Data register (bar+0)
: transfers the data parameter

Invoke the "Write Data" func. (0x8) with the parameter (0xcafe)

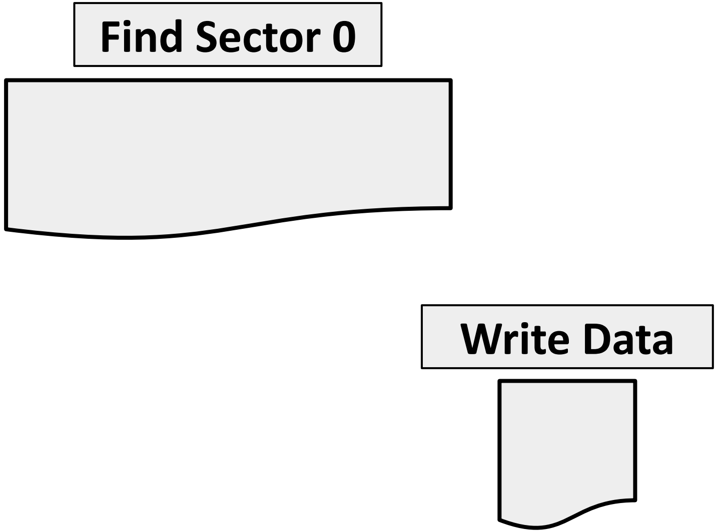
Write Data

MMIO[bar+0] ← 0xcafe
MMIO[bar+4] ← 0x8

Idea 2: Grammar Inference with Two Constraints

#2. Order dependency

- Each Input semantic functions correctly depending on order dependency

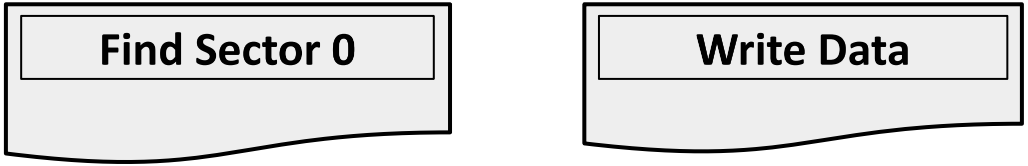


Idea 2: Grammar Inference with Two Constraints

- Grammar can be reconstructed by **two constraints**

1) The **register types** of IO address

2) **order dependency**



Idea 2: Grammar Inference with Two Constraints

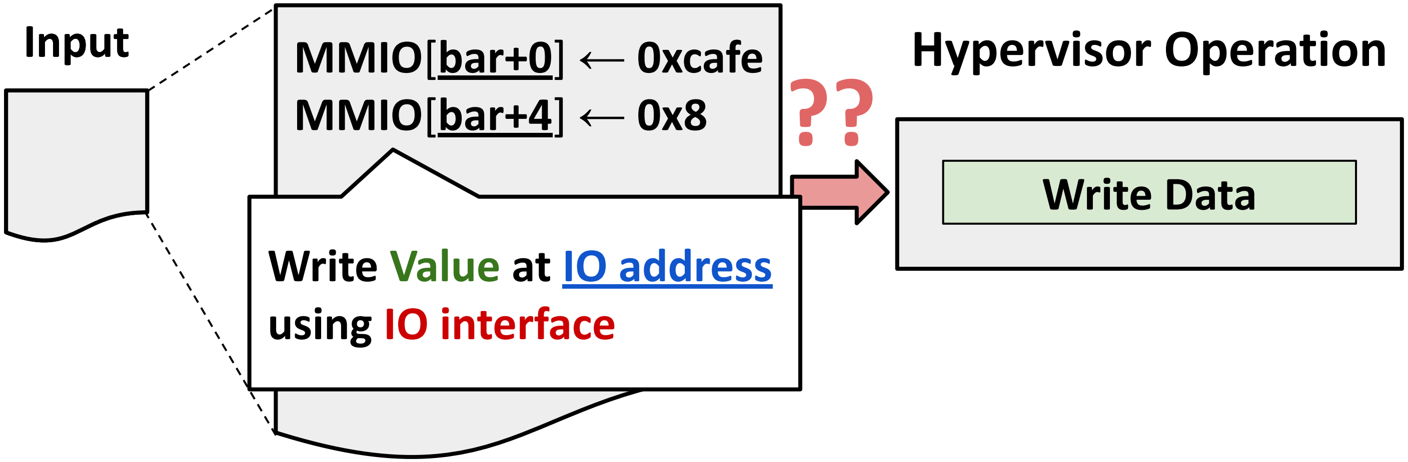
#1. Inferring Types of Register

- Infer the register type by giving correct and incorrect input
- Each register type
  - due to its operational characteristic

- Data register has **same** coverage
  - two inputs **only transfer a data** to device
- Control register has **different** coverage
  - two inputs **invoke different functions**

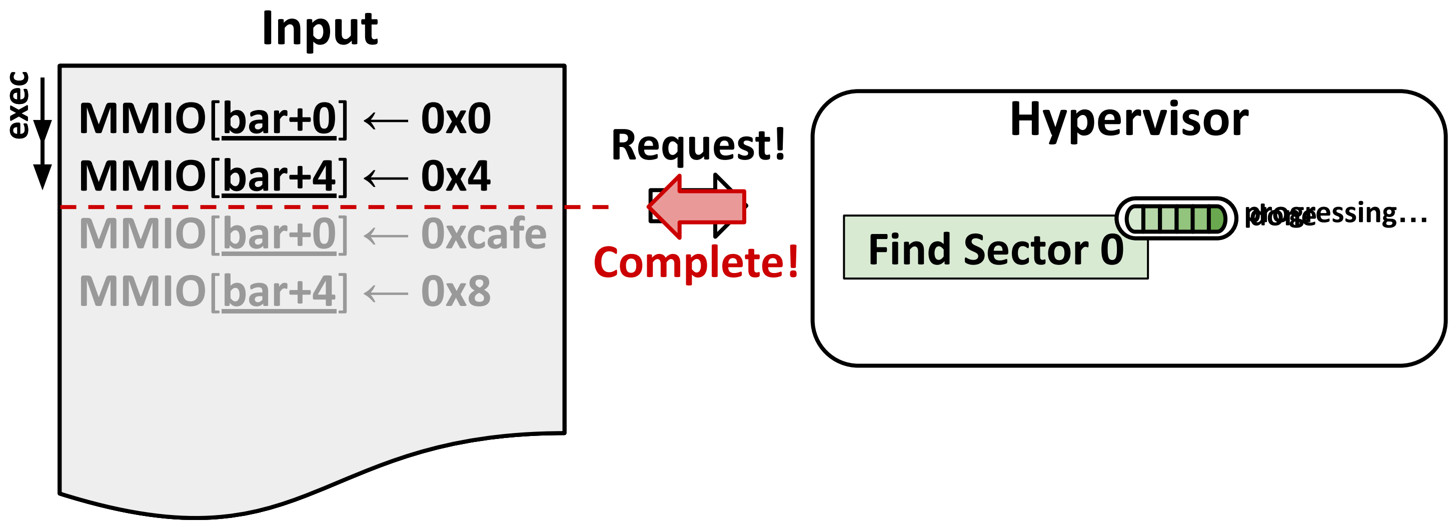
Problem 2: Uncertain Input Semantics

- Hypervisor input is presented in **a sequence of IO interface inputs**
  - hard to infer **its semantic meaning** by looking at individual IO interface input



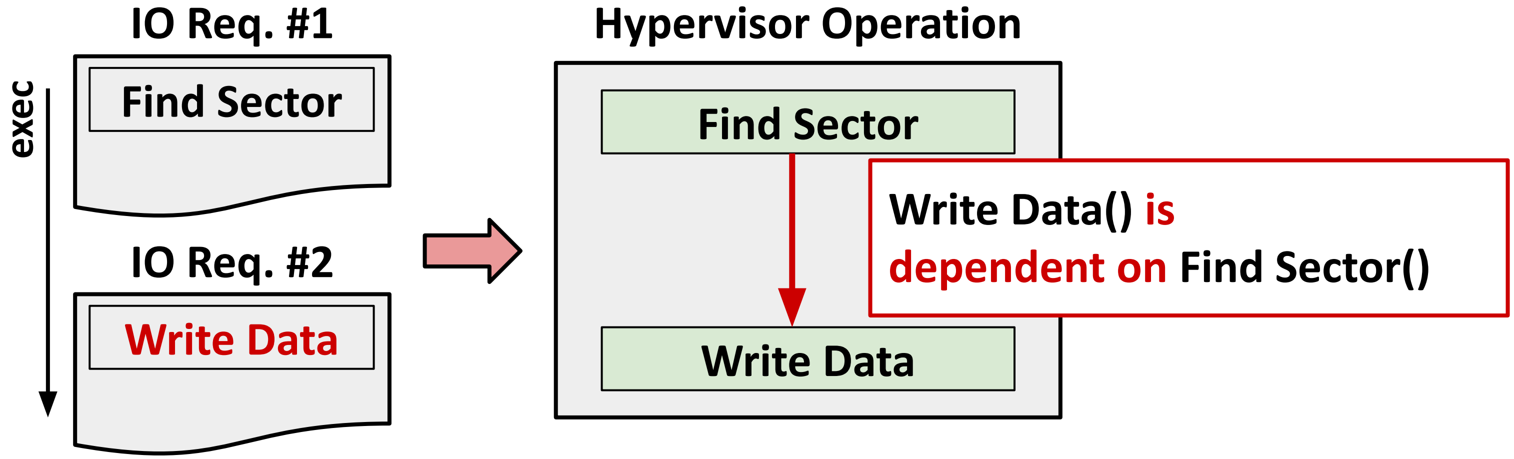
Solution 2: Subdivides the hypervisor input into IO Request

- Input semantics comprise **a small sequence of low-level IO operation**
  - **IO Request**: serves as high-level semantic task
  - hypervisor returns **completion signal** after the IO request accepts



Our approach: Inferring the grammar with semantic constraints

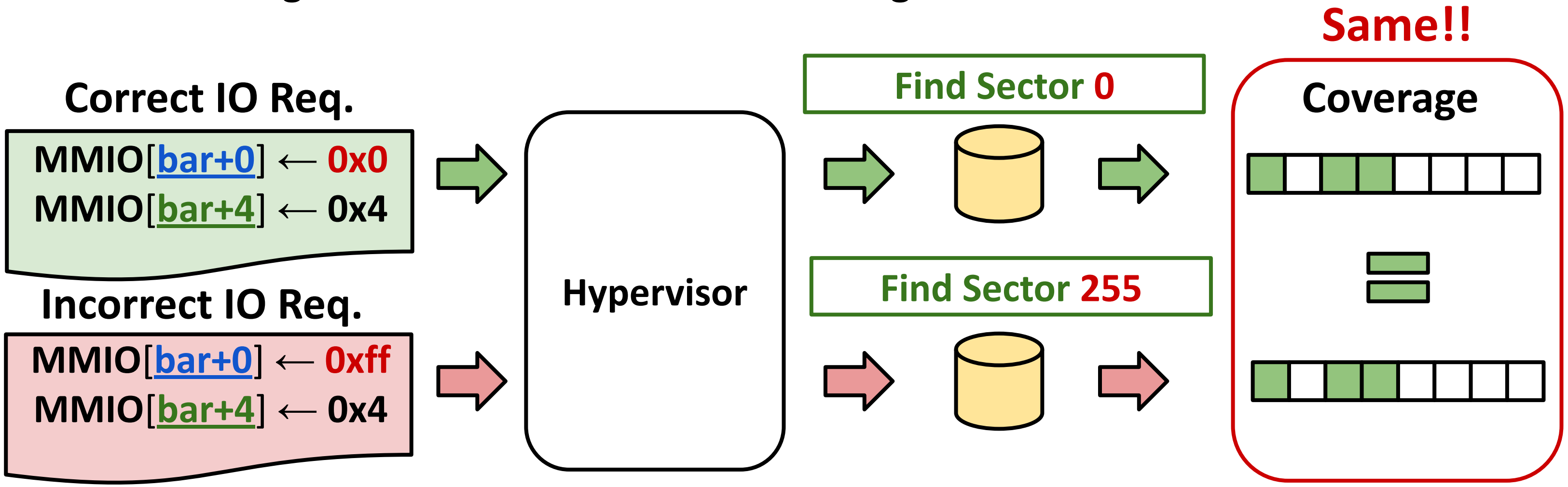
- **Giving correct/incorrect IO requests based on two constraints**
  - **Register types** (of low-level IO operation)
    - gives the information on **how to synthesize the IO request correctly**
  - **Order dependency**
    - gives the information on **how to place the IO request in a correct order**



Our approach: Inferring the grammar with semantic constraints

#1. Register Types

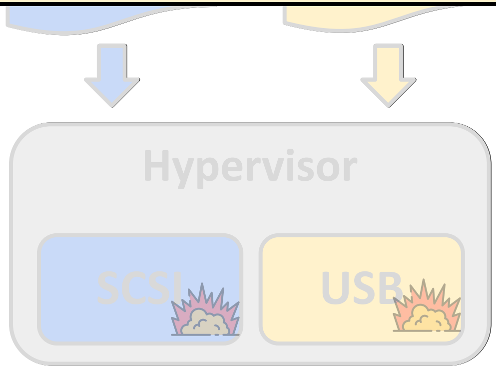
- **Inspect the input coverage by giving correct/incorrect values at IO address**
  - control register ⇒ exhibits a **different** coverage
  - data register ⇒ exhibits a **same** coverage



Example: too many devices, too many formats

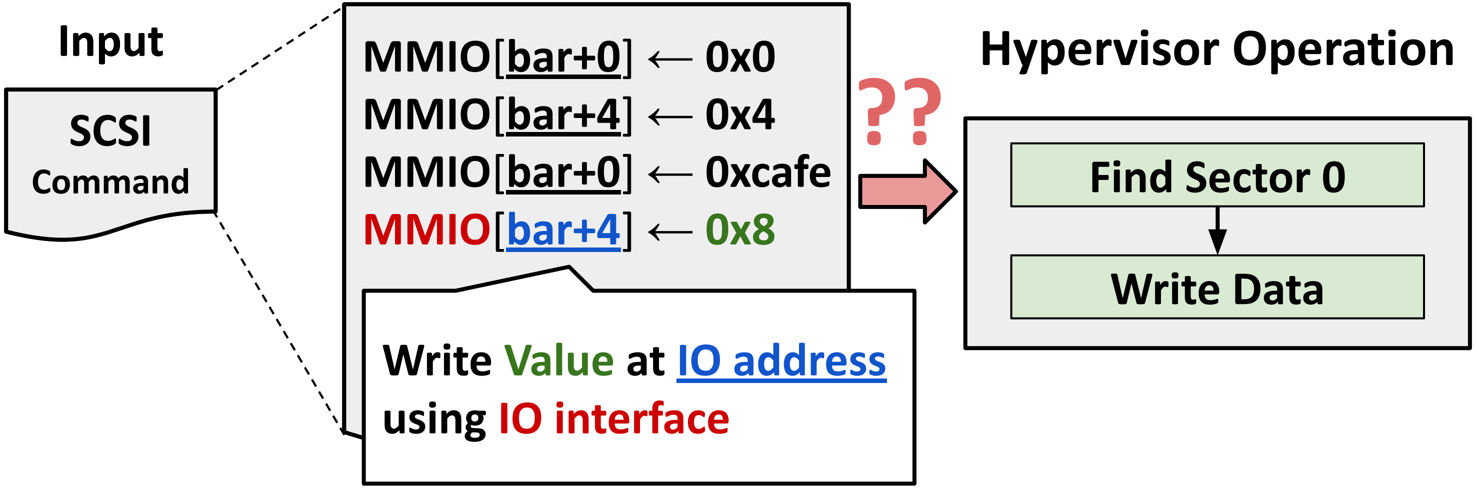
- Need different input formats to fuzz each device in hypervisor

Random cannot develop the hypervisor input regarding the input format



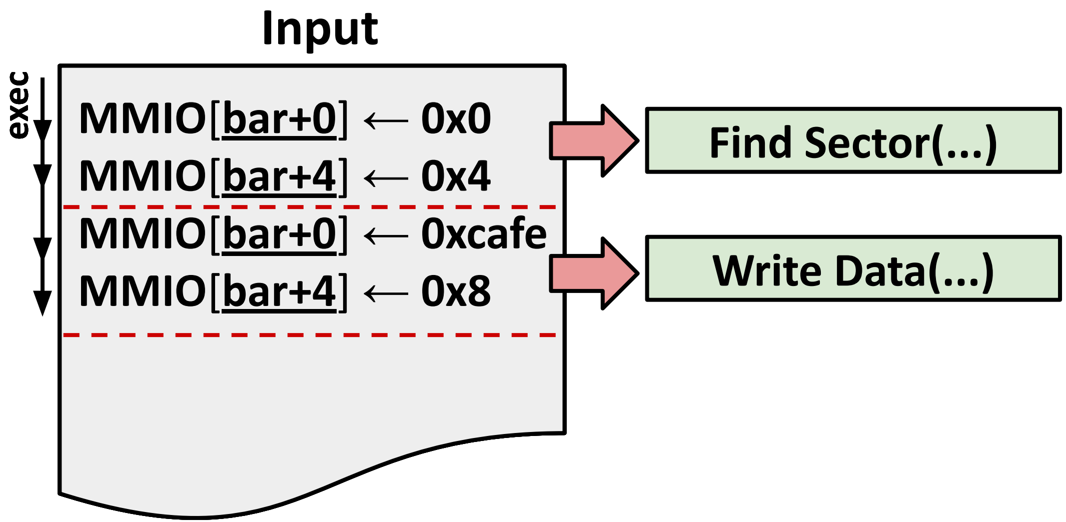
Challenge 2: Uncertain Input Semantics

- The hypervisor input is presented in **a sequence of low-level IO operations**
  - hard to infer **its semantic meaning** by looking at individual IO interface input



Solution 2: Capture the Semantic Unit with Completion Signal

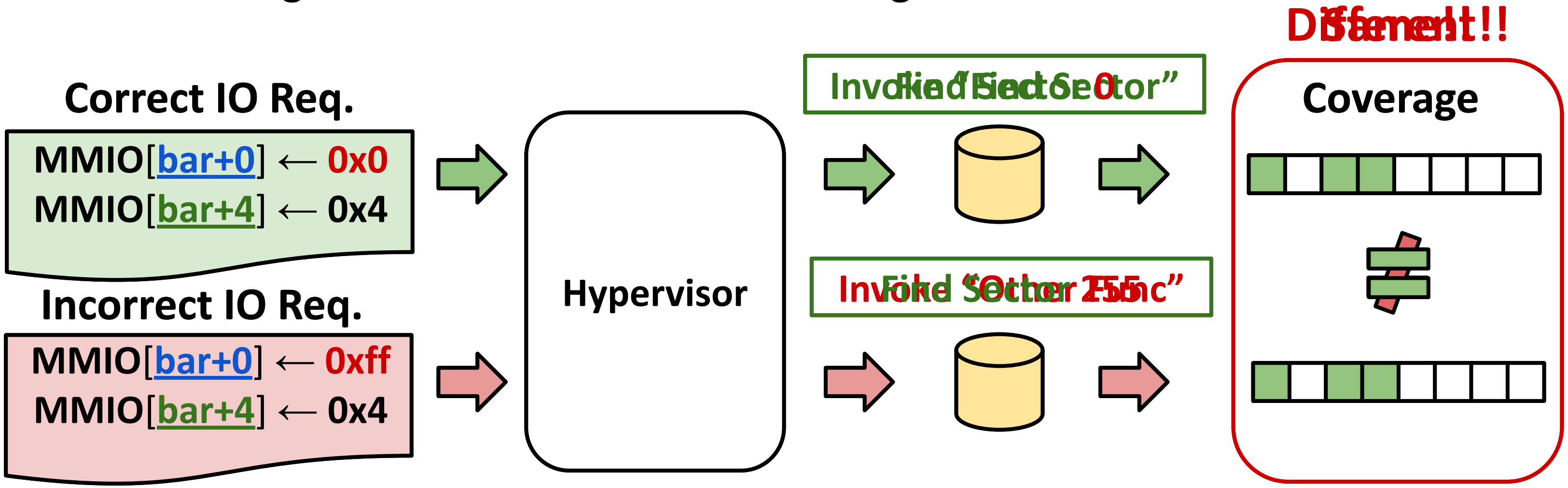
- Input semantics comprise **a small sequence of low-level IO operation**
  - **IO Request**: serves as high-level semantic task
  - hypervisor send **completion signal** after the IO request accepts



Infer the IO request with semantic constraints

#1. Register Types

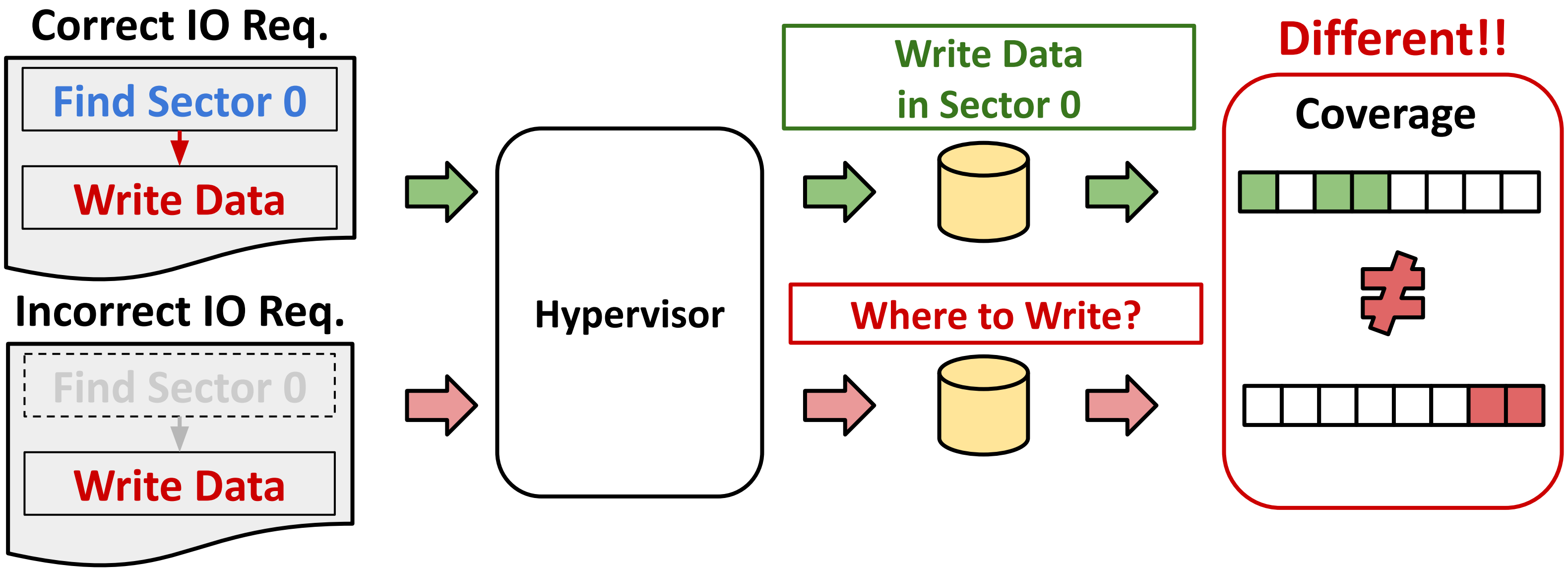
- **Inspect the input coverage by giving correct/incorrect values at IO address**
  - control register ⇒ exhibits a **different** coverage
  - data register ⇒ exhibits a **same** coverage



Infer the IO request with semantic constraints

#2. Order Dependency

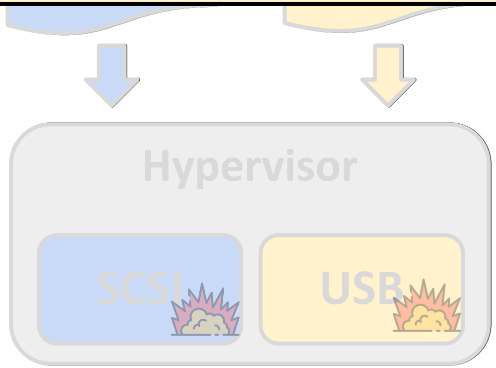
- **Inspect the input coverage by giving IO requests in correct/incorrect order**
  - **absence of one IO req. ⇒** may distort **the coverage of others**



Example: too many devices, too many formats

- Hypervisor accepts different inputs per device

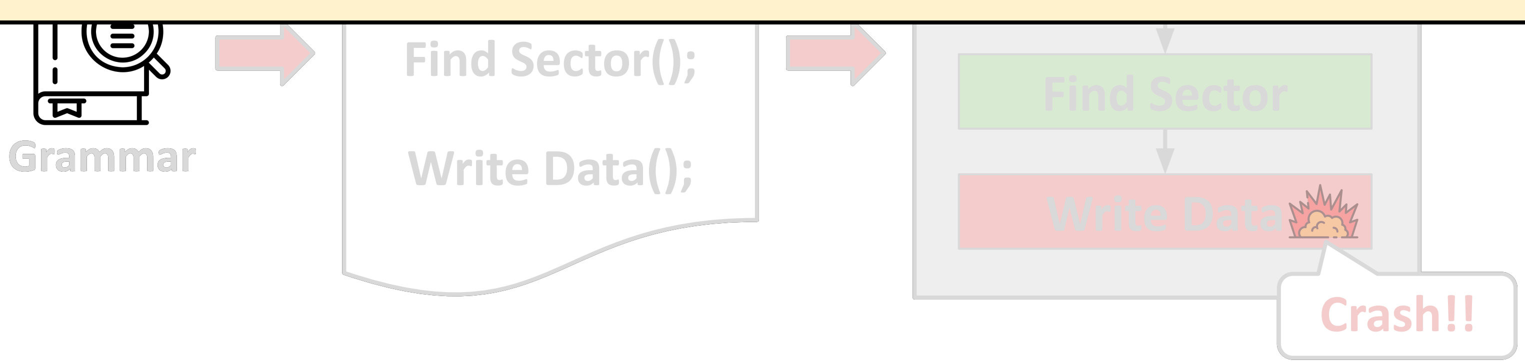
Random inputs cannot trigger interesting hypervisor behaviors



Let’s fuzz the hypervisor with grammar-awareness!

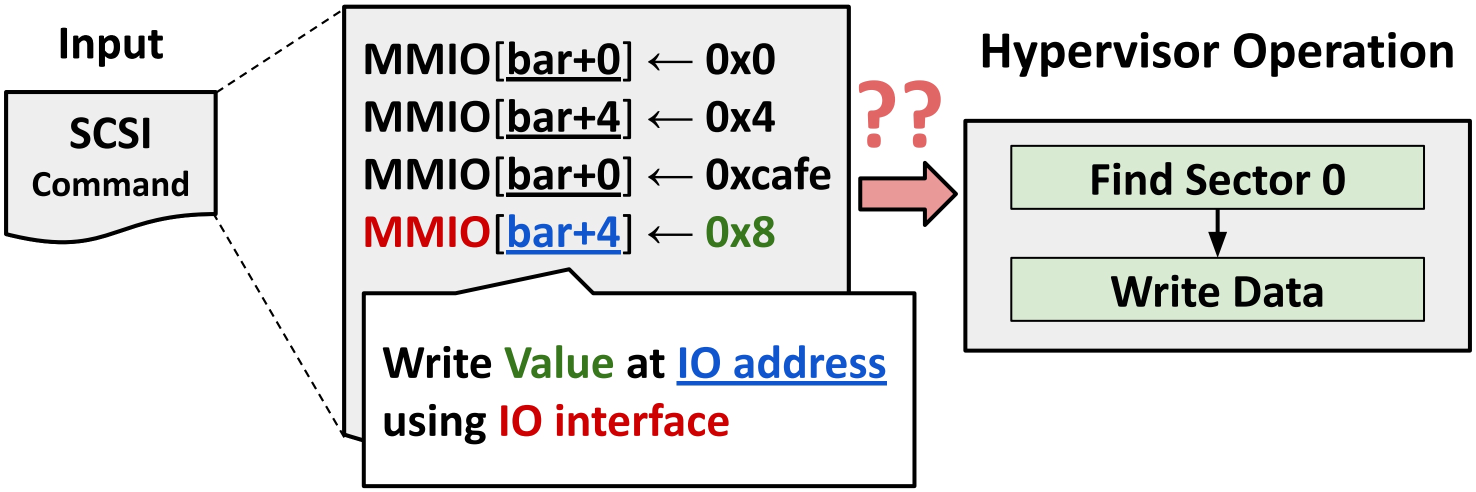
- Synthesizes **correct input semantics** with **correct order**

Grammar-aware fuzzing can explore deep state of the hypervisor!



Challenge 2: Hidden Input Semantics

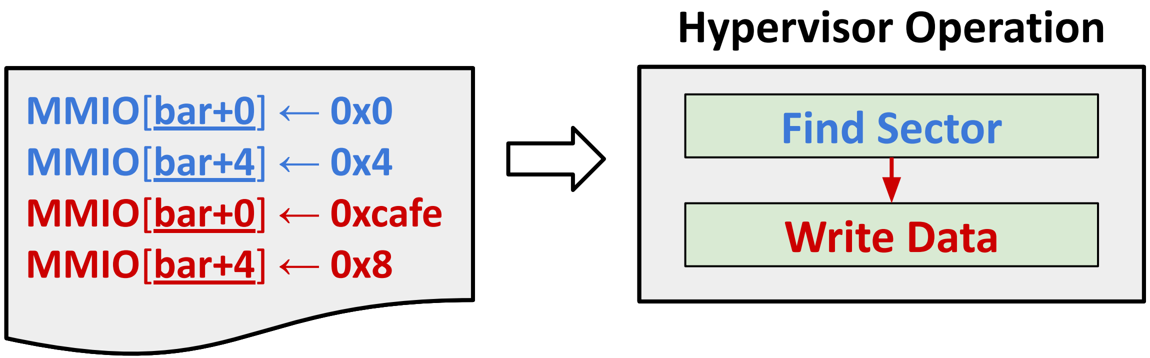
- The hypervisor input is presented in **a sequence of low-level IO operations**
  - difficult to infer **hidden semantics** behind individual IO operations



Challenge 2: Hidden Input Semantics

- **Two hidden semantics**
  - Register types (of low-level IO operation)
    - give a dedicated semantic meaning to IO operation
  - **Order dependency (between low-level IO operations)**
    - give a necessary order to correctly perform IO operation

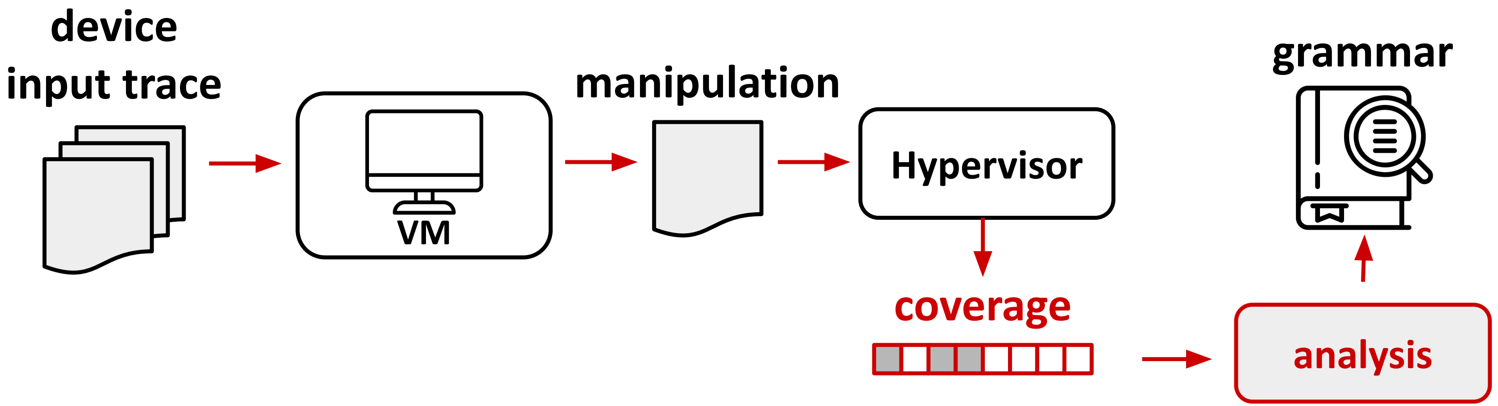
"Write Data" IO operations need "Find Sector" IO operations



MundoFuzz overview

- **Grammar-aware fuzzing with automatic grammar inference**

- **Idea**: **infer the hypervisor input grammar with input coverage**
  - #1. **measure a input coverage** by manipulating the input trace
  - #2. **analyze the difference input coverage** to make grammar



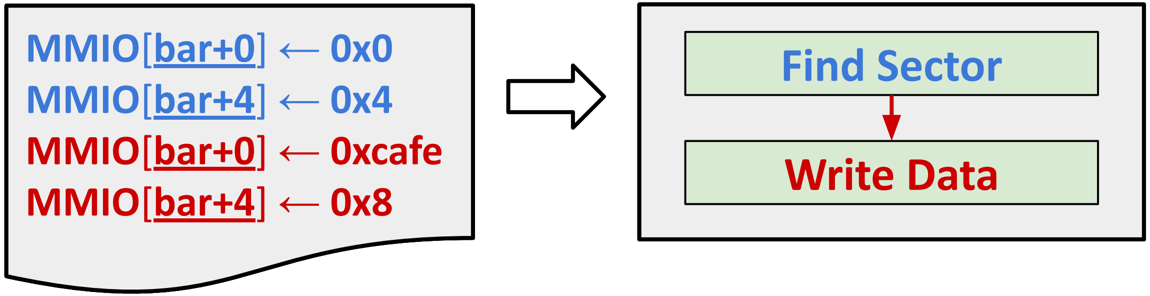
Challenge 2: Hidden Input Semantics

- IO address semantics
  - correct semantic should be given
- IO order semantics
  - give a necessary order to correctly perform IO operation

- Hypervisor input has its own semantic meaning
  - **Register types** (of low-level IO operation)
    - give a dedicated semantic meaning to IO operation
  - **Order dependency** (between low-level IO operations)
    - give a necessary order to correctly perform IO operation

"Write Data" IO operations need "Find Sector" IO operations

Hypervisor Operation



Challenge 2: Hidden Input Semantics

- Hidden input semantics
    - **IO address semantics:** correct semantic command should be given

Invoke the "Write Data" func. (0x8)
with the parameter (0xcafe)

Input

SCSI Command

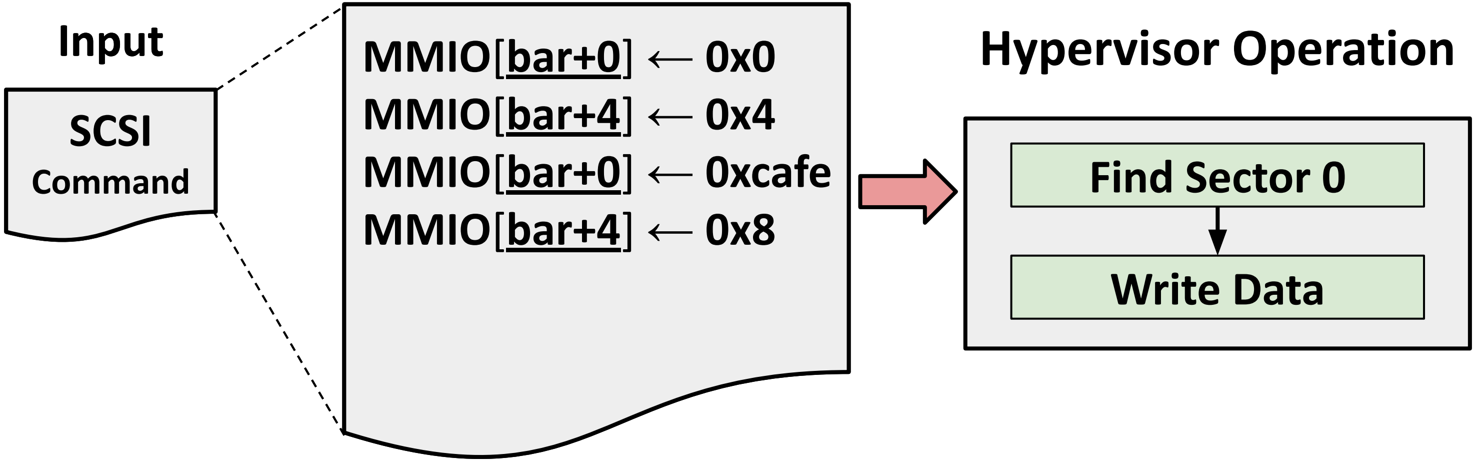
MMIO[bar+0] ← 0x0
MMIO[bar+4] ← 0x4
MMIO[bar+0] ← 0xcafe
MMIO[bar+4] ← 0x8

Control Type(bar+4)
: invoke the desired function

Data Type (bar+0)
: transfers the data parameter

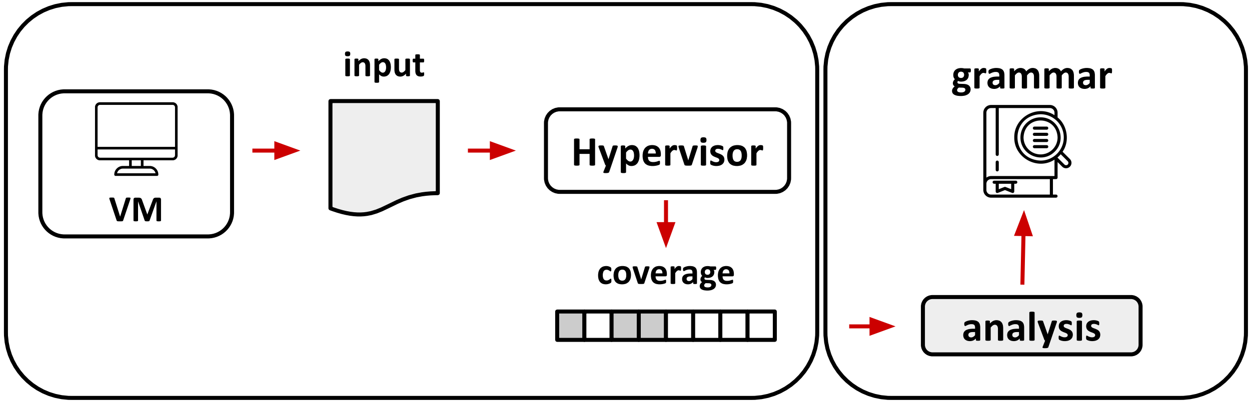
Challenge 2: Hidden Input Semantics

- **Hidden input semantics**
    - IO address semantics: correct semantic command should be given
    - **IO order semantics:** correct semantic order should be given



MundoFuzz overview

- **Find hypervisor bugs through automatic grammar inference**
- **Idea: Infer the grammar through hypervisor input coverage**
  - **#1. Measure the coverage by hypervisor input**
  - **#2. Infer the grammar by analyzing the input coverage**



MundoFuzz overview

- **Augment hypervisor fuzzing capability with automatic grammar inference**

- **Idea: Infer the grammar through hypervisor input coverage**
  - **#1. Measure the coverage by hypervisor input**

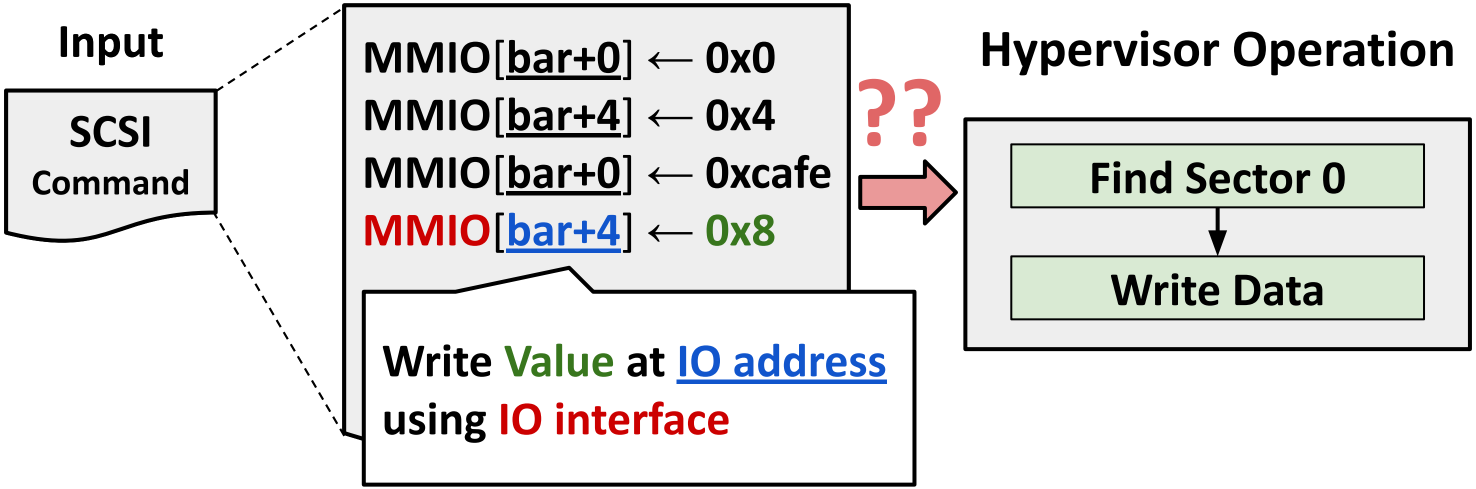
  - **#2. Infer the grammar by analyzing the input coverage**

How to teach hypervisor grammar awareness?

- We found two challenges in inferring hypervisor input grammars
  - **challenge #1. Coverage noises**
    : make different input coverage even same hypervisor input is given
  - **challenge #2. Hidden Input Semantics**
    : hard to infer the hidden semantics behind the hypervisor input
- **Our approach: Statistical and differential learning with coverage**

Challenge 2: Hidden Input Semantics

- Too difficult to infer input semantics behind the hypervisor input
  - hypervisor input is presented in **a sequence of IO operations**
  - difficult to infer **hidden semantics** behind individual IO operations



Challenge 1: Hidden Input Semantics

- Too difficult to infer input semantics behind the hypervisor input
  - hypervisor input is presented in **a sequence of IO operations**