HyperPill

Fuzzing for Hypervisor-bugs by leveraging the Hardware Virtualization Interface

Alexander Bulekov
Qiang Liu
Manuel Egele
Mathias Payer

EPFL
Boston University
Zhejiang University
Amazon (all work completed prior to joining Amazon)

hexhive

BOSTON UNIVERSITY SeclaBU

Motivation

Applications:

Cloud

Personal Computing

Development

Security Research

Mobile

Automotive

Fuzzing Hypervisors: **Challenges**

Hypervisor inputs feature complex semantics

There are many hypervisors

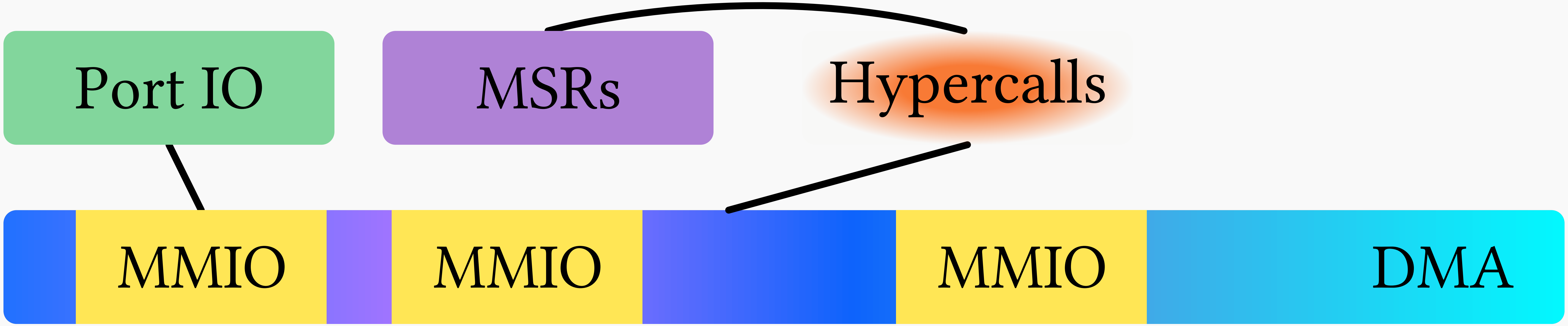
Implementations are diverse

Hypervisors are low-level systems software

All Hypervisors use an identical CPU virtualization interface

The Input Space

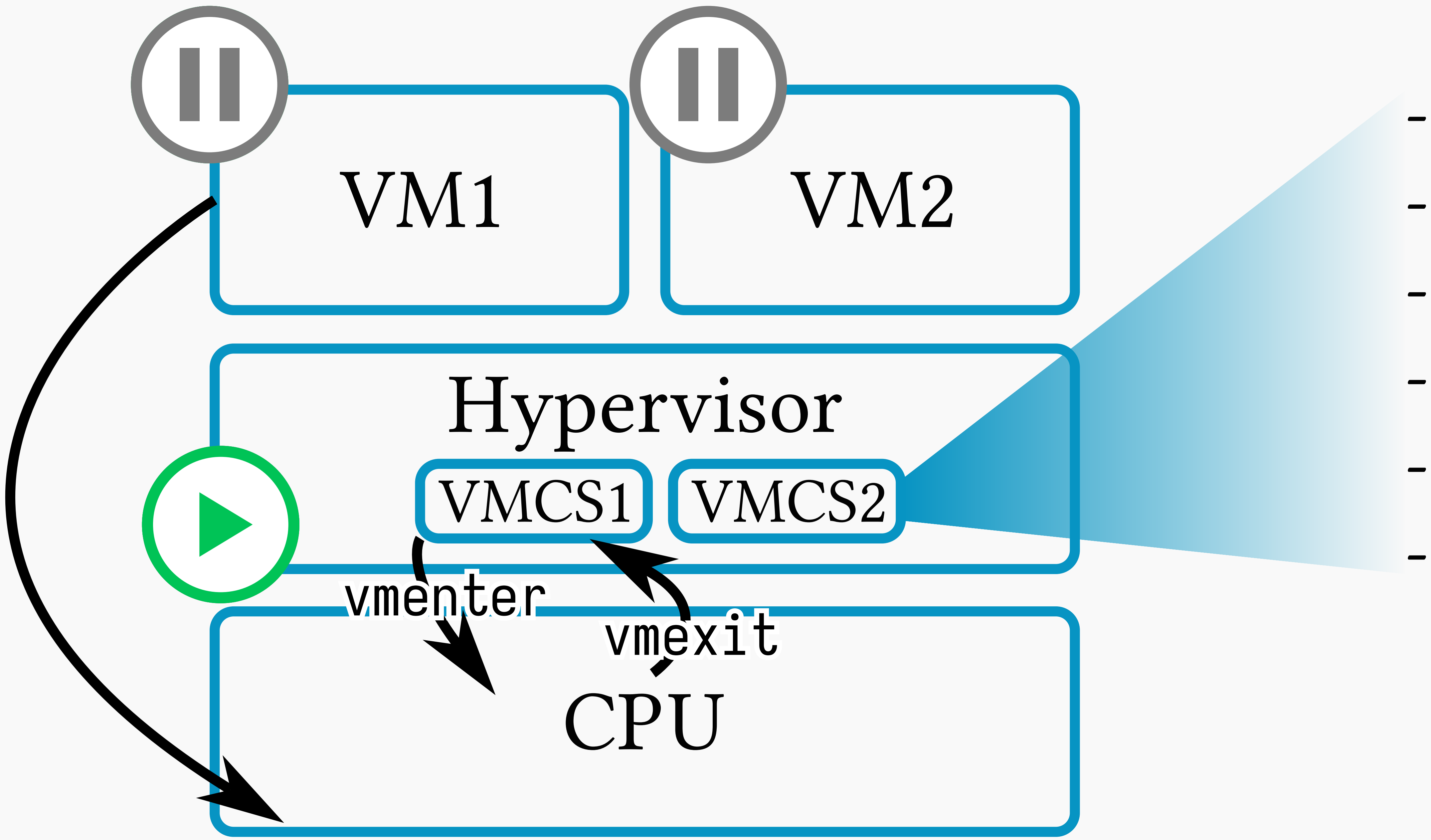
What interfaces do hypervisors expose to VMs?



The combined hypervisor input space is enormous

Hypervisor semantics are complex and variable

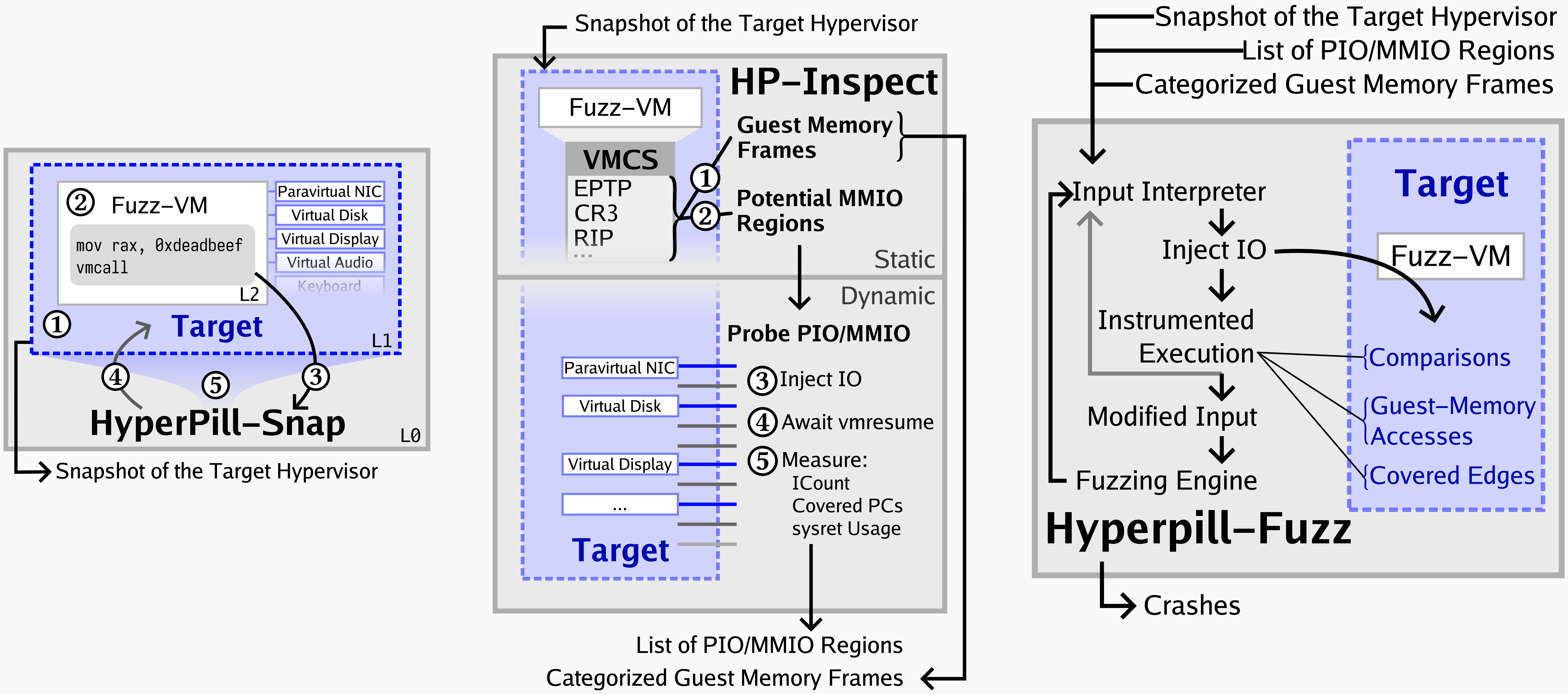
The Hardware Virtualization Interface



- Guest vCPU State
- Memory allocated to the Guest
- MMIO Regions
- PIO Regions
- Enabled MSRs
- VM Exit Reasons

Every Hypervisor must use an identical CPU virtualization interface.
HyperPill leverages this fact to fuzz any hypervisor - generically.

HyperPill



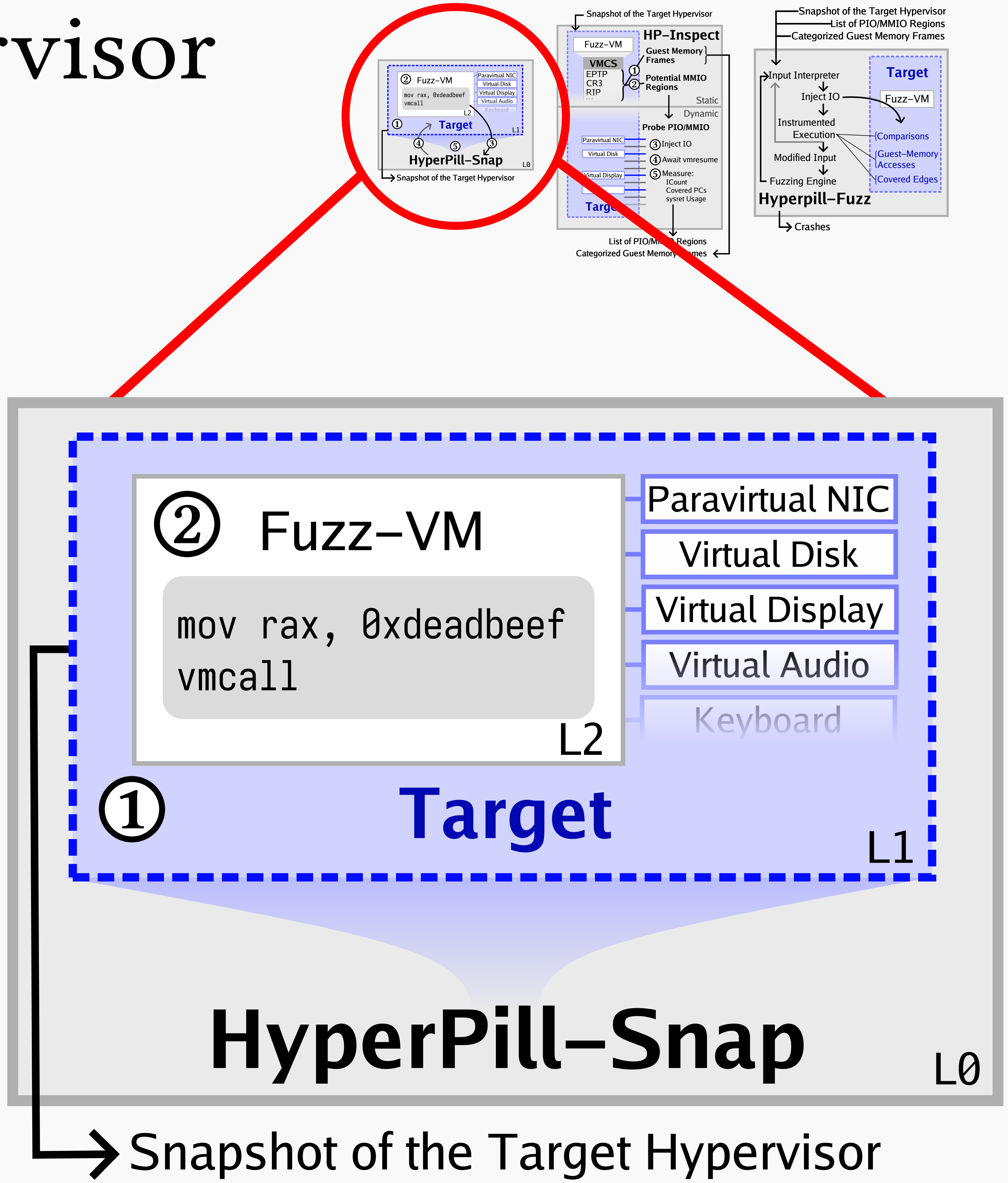
② Fuzz–VM
mov rax, 0xdeadbeef
vmcall
L2
Paravirtual NIC
Virtual Disk
Virtual Display
Virtual Audio
Keyboard
①
Target
L1
④
⑤
③
HyperPill–Snap
L0
Snapshot of the Target Hypervisor
Snapshot of the Target Hypervisor
HP–Inspect
Fuzz–VM
VMCS
EPTP
CR3
RIP
...
Guest Memory Frames
①
②
Potential MMIO Regions
Static
Dynamic
Probe PIO/MMIO
Paravirtual NIC
Virtual Disk
Virtual Display
...
③ Inject IO
④ Await vmresume
⑤ Measure:
ICount
Covered PCs
sysret Usage
Target
List of PIO/MMIO Regions
Categorized Guest Memory Frames
Snapshot of the Target Hypervisor
List of PIO/MMIO Regions
Categorized Guest Memory Frames
Input Interpreter
Inject IO
Instrumented Execution
Modified Input
Fuzzing Engine
Target
Fuzz–VM
Comparisons
Guest–Memory Accesses
Covered Edges
Hyperpill–Fuzz
Crashes

Step 1: Snapshotting the Hypervisor

Problem: Make a snapshot of the hypervisor just as it is about to handle a VM Exit.

Solution:
Run the hypervisor nested in HyperPill-Snap.

Invoke a special hypercall from the VM to trigger a VM Exit and tell HyperPill-Snap to collect a snapshot.



Step 2: Inspecting the Snapshot

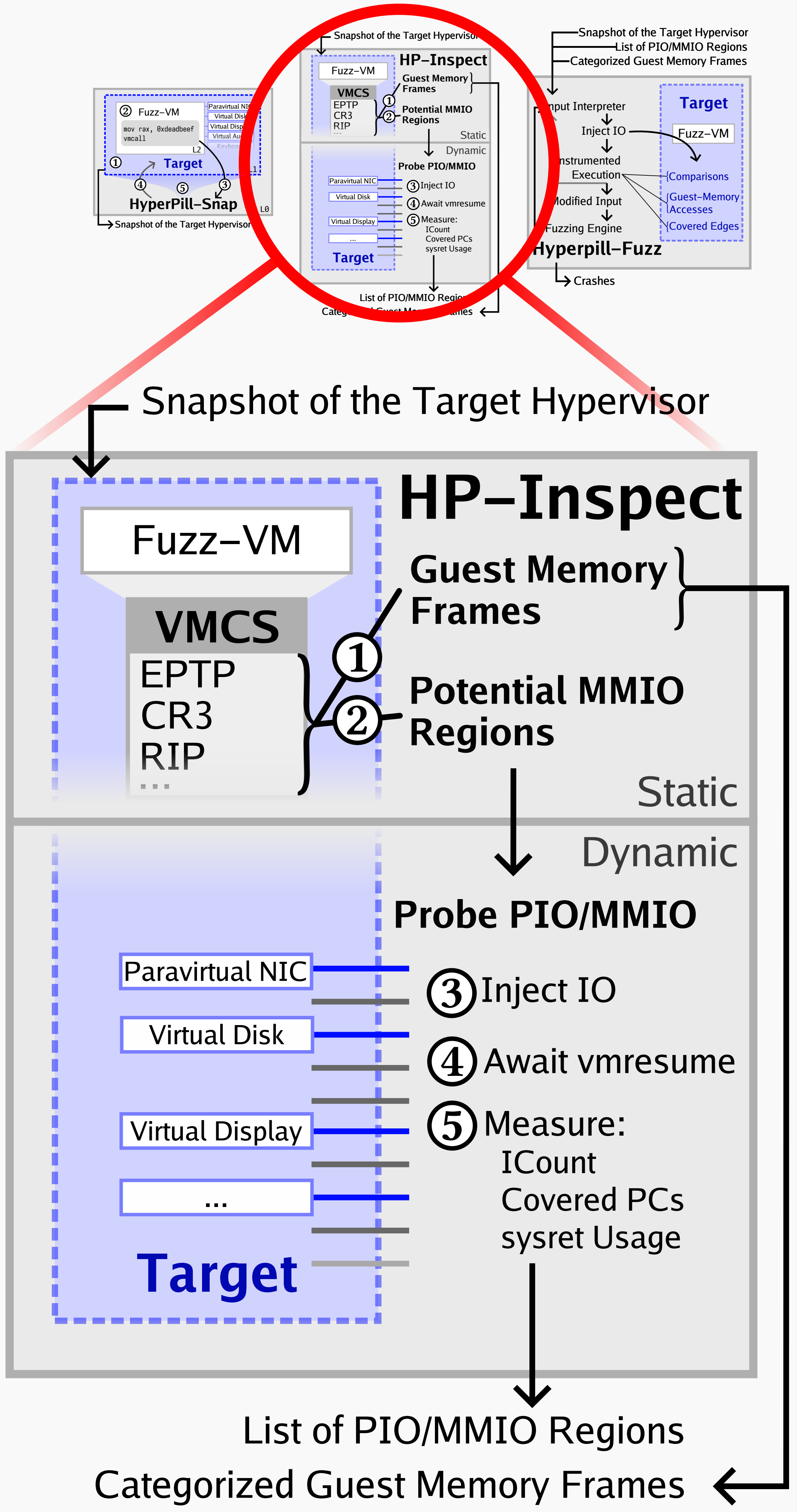
The snapshot contains the entire hypervisor state, including the VMCS the hypervisor sets up for the VM.

VMCS Inspection/Probing to Identify:

MMIO Regions

PIO Regions

Memory allocated to the Guest



Step 3: Fuzzing

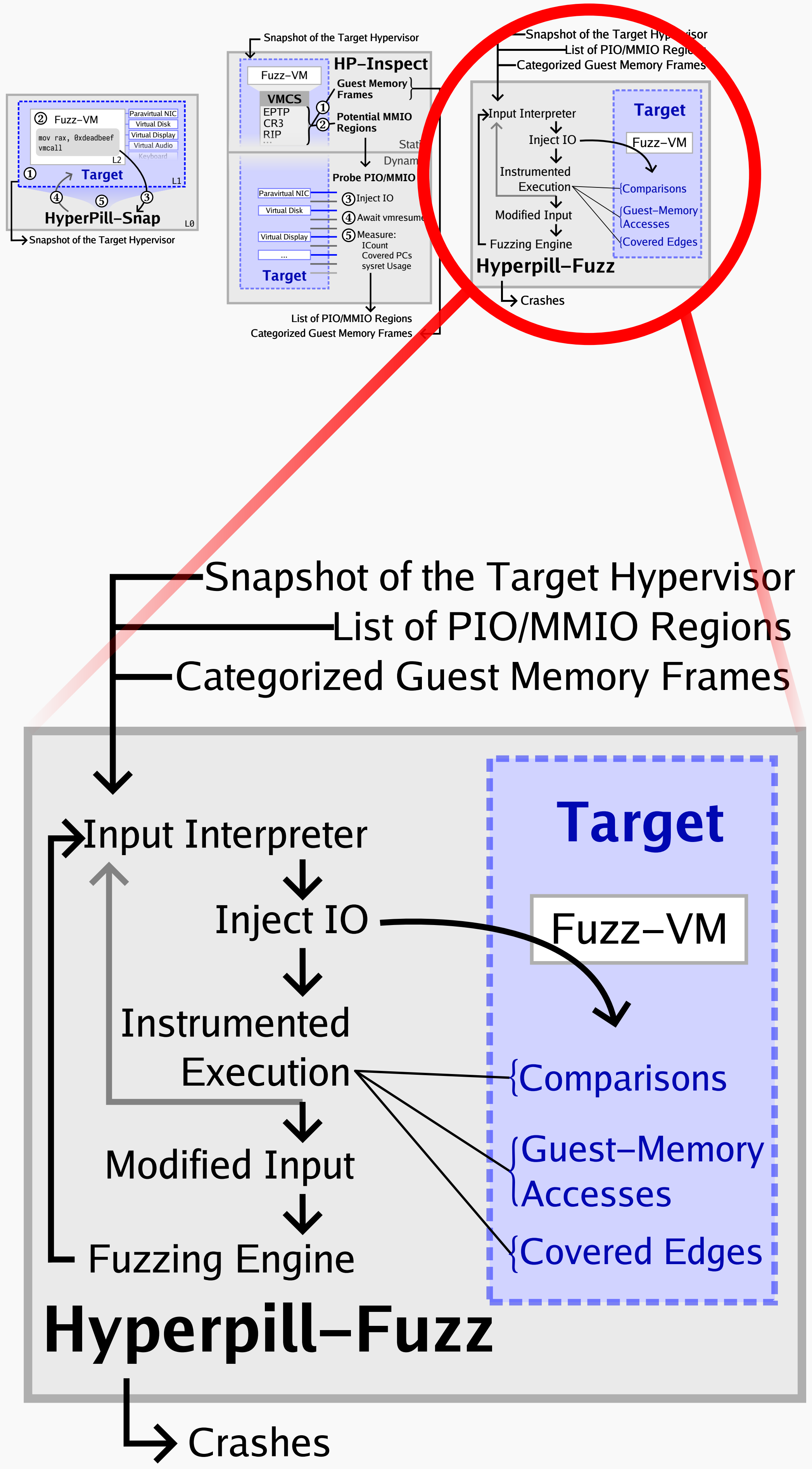
Fuzzer input is a sequence of IO Operations

MMIO | PIO | VMCALL | …

Modify VMCS to reflect VMExit for the IO Operation. Resume the Hypervisor Snapshot in an emulator.

When the Hypervisor resumes the VM, immediately inject the next VM-Exit

PIO ✓ MMIO ✓ MSRs ✓ HyperCalls ✓ DMA ?

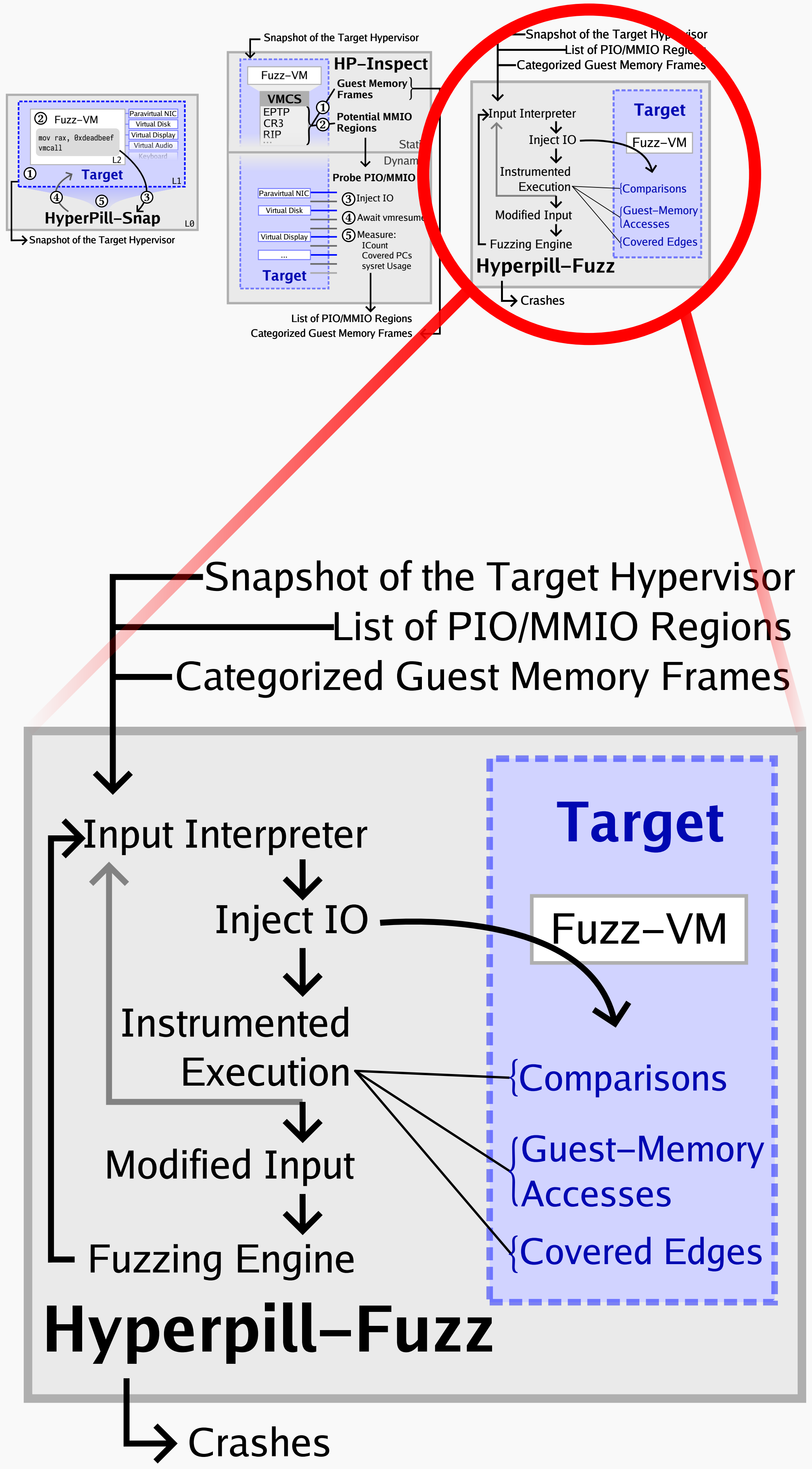


Step 3: Fuzzing DMA

DMA allows hypervisor to read from guest memory at any time while handling VMExits

In step 2, HyperPill identified all Guest Memory pages within the snapshot

During emulation, if any instruction reads from the Guest Memory, HyperPill fills the memory with fuzzer data



Step 3: Fuzzing DMA

DMA allows hypervisor to read from guest memory at any time while handling VMExits

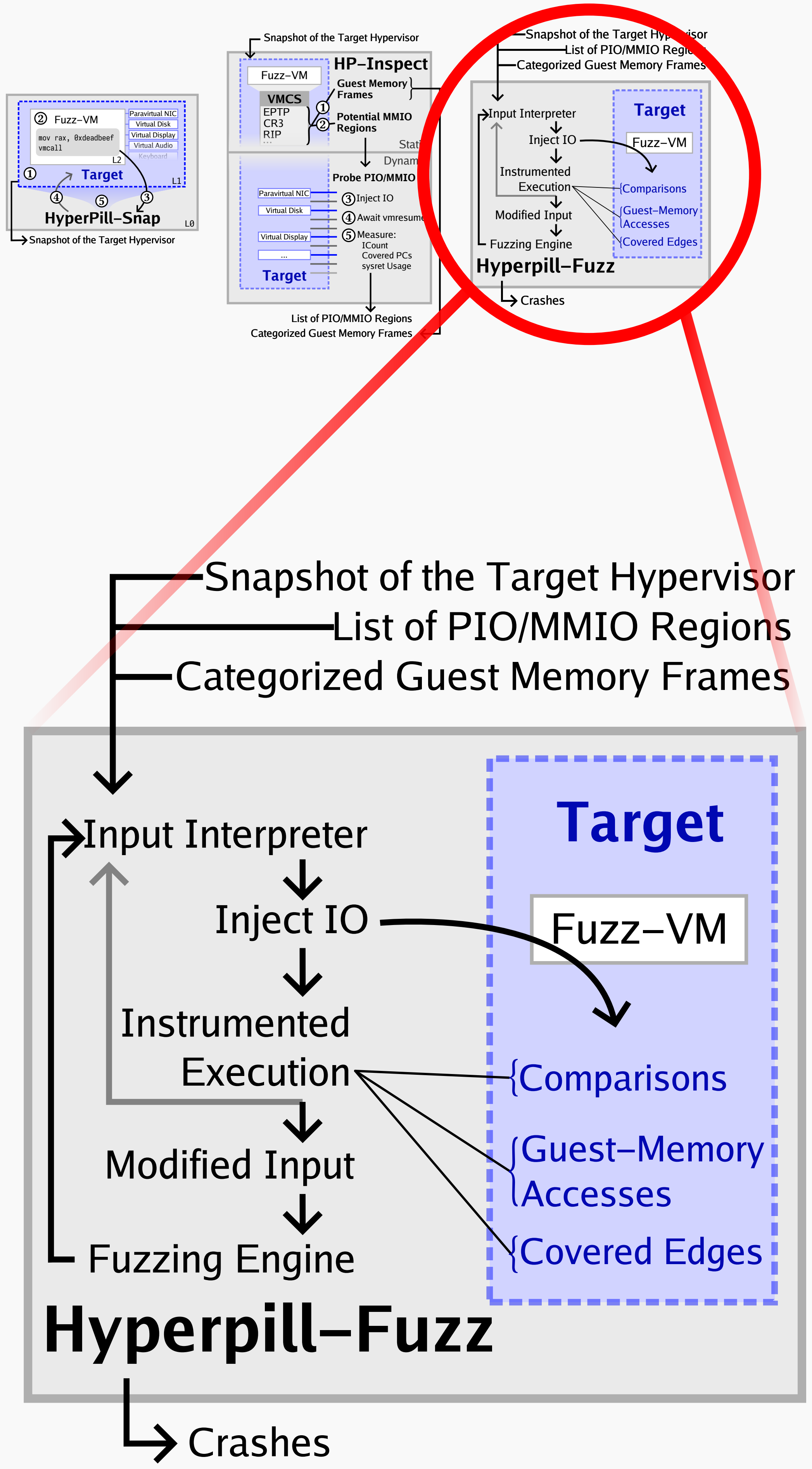
In step 2, HyperPill identified all Guest Memory pages within the snapshot

PIO ✓ MMIO ✓ MSRs ✓ HyperCalls ✓ DMA ✓

Guest Memory

During emulation, if any instruction reads from the Guest Memory, HyperPill fills the memory with fuzzer data

MMIO DMA PIO



Results

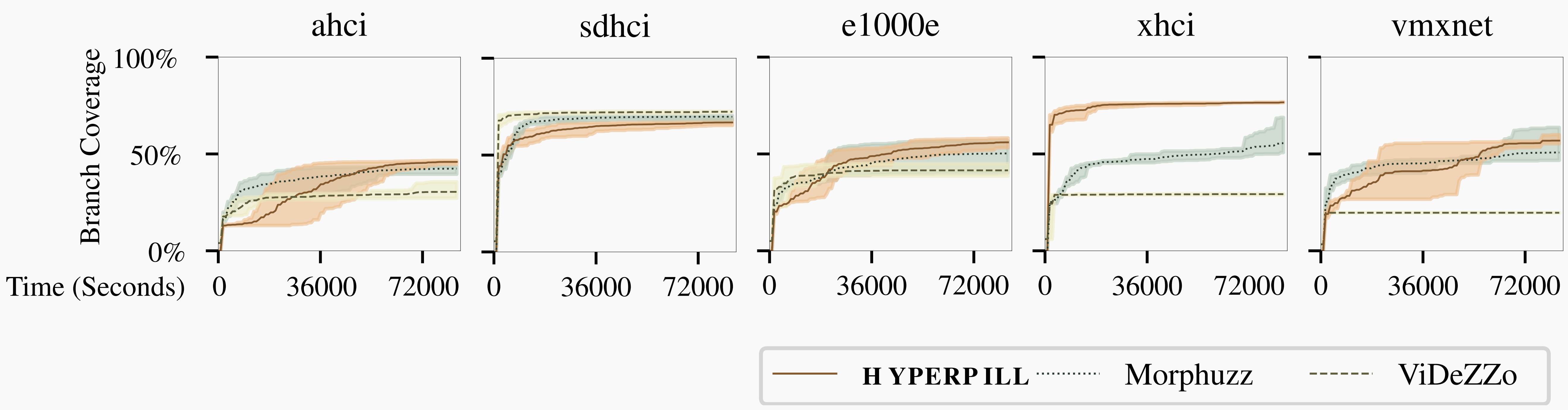
Three Hypervisors

HyperPill achieves higher coverage for 10/12 devices.

Inherent performance penalty due to use of full-system snapshotting and emulation.

Byte-level DMA sensitivity allows better performance for DMA-heavy devices.

		Morphuzz		ViDeZZo		HYPERPILL	
		12 Cores 24 Hours					
Device	⊡	Branch Coverage (Executions/Second)					
Block							
ahci	✓	42.43%	(25.68)	30.42%	(562.24)	**45.90%**	(26.18)
nvme		29.12%	(23.82)			**36.44%**	(14.45)
sdhci	✓	69.81%	(22.98)	**72.37%**	(107.22)	66.85%	(32.34)
virtio-scsi	✓	27.96%	(23.83)	11.73%	(217.28)	**48.83%**	(51.68)
Display							
cirrus		88.10%	(19.06)	83.42%	(138.78)	**88.67%**	(32.18)
qxl	✓					**59.68%**	(26.96)
virtio-gpu	✓	24.37%	(26.21)	2.77%	(222.42)	**45.52%**	(36.53)
Networking							
e1000e	✓	50.27%	(24.83)	41.52%	(53.04)	**55.99%**	(42.22)
igb	✓	29.73%	(25.63)			**35.93%**	(60.85)
vmxnet	✓	50.75%	(27.01)	19.64%	(145.73)	**56.89%**	(48.14)
USB							
ehci	✓	73.76%	(24.58)	**74.38%**	(177.08)	73.32%	(10.46)
xhci	✓	55.54%	(28.83)	29.25%	(1061.36)	**76.64%**	(69.26)
Geo. Mean		45.20%	(24.65)	28.00%	(203.07)	55.45%	(33.20)



Results

Three Hypervisors

```c
/* map PRDT */
if (!(prdt = dma_memory_map(ad->hba->as, prdt_addr, &prdt_len,
                            DMA_DIRECTION_TO_DEVICE,
                            MEMTXATTRS_UNSPECIFIED)))){
    trace_ahci_populate_sglist_no_map(ad->hba, ad->port_no);
    return -1;
}
/* Get entries in the PRDT, init a qemu sglist accordingly */
if (prdtl > 0) {
    AHCI_SG *tbl = (AHCI_SG *)prdt;

    for (i = 0; i < prdtl; i++) {
        tbl_entry_size = prdt_tbl_entry_size(&tbl[i]);
        if (offset < (sum + tbl_entry_size)) {
            off_idx = i;
```

Source-Based DMA Hooking

```asm
leaq   (%r15,%r12), %r14
incq   %r14
movq   %r14, %rdi
movq   %rbx, %rsi
```

HyperPill DMA Hooking

QEMU

- Arbitrary memory-access in e1000e_start_xmit
- Heap-overflow in usb_mouse_poll
- Heap-overflow in virtqueue_alloc_element
- Heap-overflow in qxl_cookie_new
- Heap-overflow in igb_tx_pkt_switch
- Out-of-bounds memory access in nvme_process_sq
- Out-of-bounds memory access in nvme_io_mgmt_send
- DoS via arbitrary-sized allocation in qxl
- DoS via arbitrary-sized allocation in virtio_gpu
- DoS in process_ncq_command
- DoS in icmp_input

Bugs

Hyper-V

- Heap-corruption in EthernetCard::HandleTransmitSetupFrame
- Abort in EthernetCard::PollForTransmitDataTimer
- Abort after IdeChannel::EnlightenedHddCommand
- EthernetCard::SetupEthernetCardModeFromRegisters
- Out-of-bounds write in GuestStateAccess::SetDeviceInfo
- Abort after PitDevice::NotifyIoPortRead
- Abort in I8042Device::HandleCommand
- Abort after HvCallDetachDevice
- Abort after HvCallGetGpaPagesAccessState

macOS Virtualization Framework

- Memory-privilege violation in xHCI
- Out-of-bounds write in virtio-gpu
- Out-of-bounds write in virtio-audio
- Out-of-bounds access in virtio-block
- Out-of-bounds access in virtio-console
- Out-of-bounds access in virtio-net

Reshape hypervisors by modifying the CPU virtualization interface

No modification to hypervisors code needed!

Fuzz *any* hypervisor across its PIO, MMIO, DMA, and Hypercall interfaces

More precise than source-level reshaping

HyperPill

https://github.com/HexHive/HyperPill