Reshaping Input Spaces to Fuzz Complex Targets

Alexander Bulekov
BU Seclab
ECE Department
PhD Thesis Defense
March 19, 2024

Thesis Committee:
Prof. Manuel Egele
Prof. Gianluca Stringhini
Prof. Orran Krieger
Prof. Mathias Payer

BOSTON UNIVERSITY

Software continues to have bugs

Software continues to have bugs

VULNERABILITIES

Firefox 116 Patches High-Severity Vulnerabilities

Firefox 116 was released with patches for 14 CVEs, including nine high-severity vulnerabilities, some of which can lead to remote code execution or sandbox escapes.

By Ionut Arghire
August 2, 2023

Software continues to have bugs

VULNERABILITIES

Firefox 116 Patches High-Severity Vulnerabilities

Firefox 116 was released with patches for 14 CVEs, including nine high-severity vulnerabilities, some of which can lead to remote code execution or sandbox escapes.

By Ionut Arghire
August 2, 2023

linux-cve-announce.vger.kernel.org archive mirror

CVE search help / color / mirror / Atom feed

Search results ordered by [date|relevance] view[summary|nested|Atom feed]
download mbox.gz: results only | full threads

1. CVE-2024-26626: ipmr: fix kernel panic when forwarding mcast packets
   - by Greg Kroah-Hartman @ 2024-03-06 6:46 UTC [4%]
2. CVE-2024-26628: drm/amdkfd: Fix lock dependency warning
   - by Greg Kroah-Hartman @ 2024-03-06 6:46 UTC [6%]
3. CVE-2024-26627: scsi: core: Move scsi_host_busy() out of host lock for waking up EH handler
   - by Greg Kroah-Hartman @ 2024-03-06 6:46 UTC [6%]
4. CVE-2023-52592: libbpf: Fix NULL pointer dereference in bpf_object__collect_prog_relos
   - by Greg Kroah-Hartman @ 2024-03-06 6:45 UTC [6%]
5. CVE-2023-52591: reiserfs: Avoid touching renamed directory if parent does not change
   - by Greg Kroah-Hartman @ 2024-03-06 6:45 UTC [6%]
6. CVE-2023-52590: ocfs2: Avoid touching renamed directory if parent does not change
   - by Greg Kroah-Hartman @ 2024-03-06 6:45 UTC [6%]

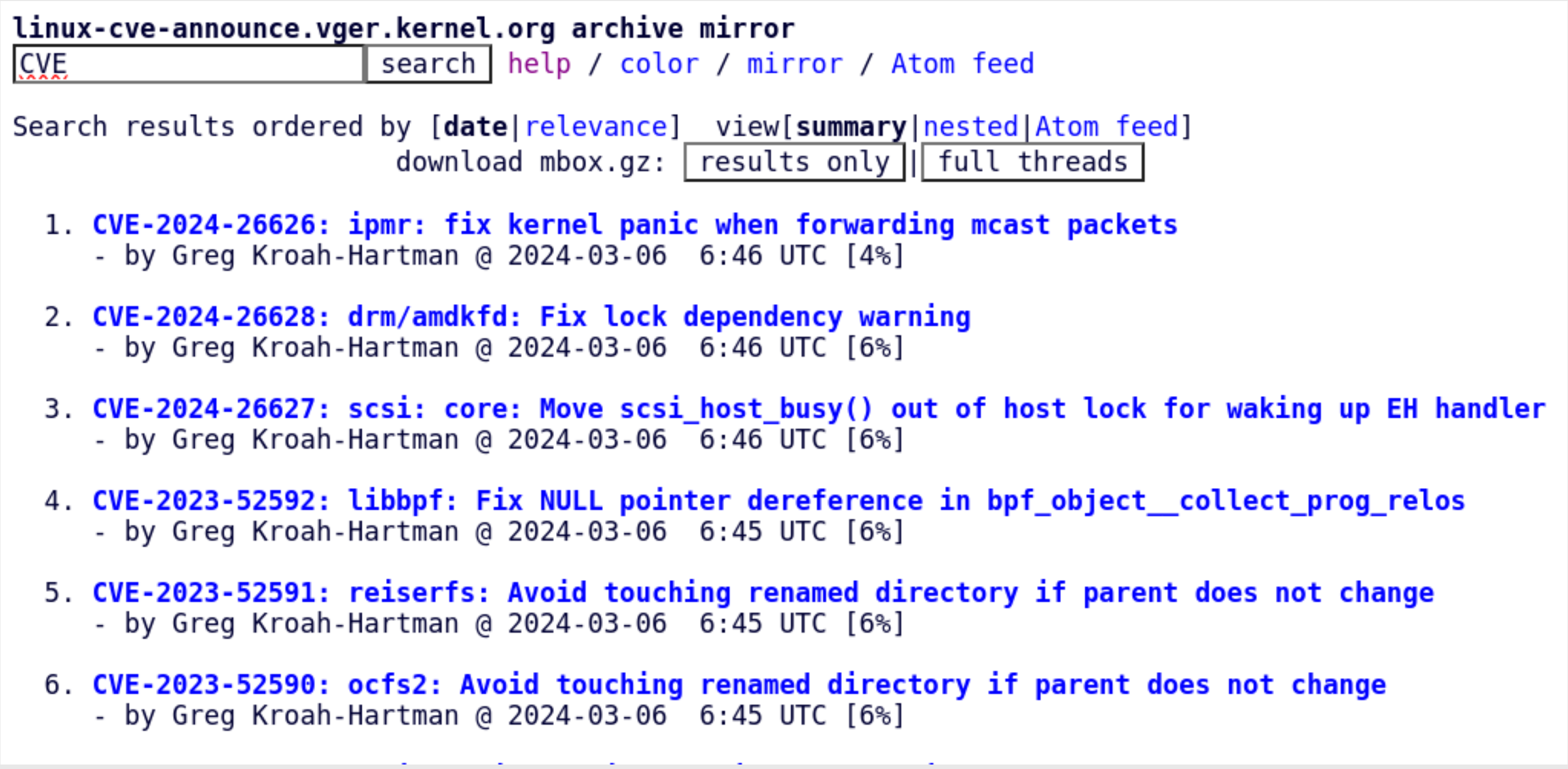
Software continues to have bugs

VULNERABILITIES

Firefox 116 Patches High-Severity Vulnerabilities

Firefox 116 was released with patches for 14 CVEs, including nine high-severity vulnerabilities, some of which can lead to remote code execution or sandbox escapes.

By Ionut Arghire
August 2, 2023



linux-cve-announce.vger.kernel.org archive mirror
CVE [search] help / color / mirror / Atom feed

Search results ordered by [date|relevance] view[summary|nested|Atom feed]
download mbox.gz: [results only]|[full threads]

1. CVE-2024-26626: ipmr: fix kernel panic when forwarding mcast packets
   - by Greg Kroah-Hartman @ 2024-03-06 6:46 UTC [4%]
2. CVE-2024-26628: drm/amdkfd: Fix lock dependency warning
   - by Greg Kroah-Hartman @ 2024-03-06 6:46 UTC [6%]
3. CVE-2024-26627: scsi: core: Move scsi_host_busy() out of host lock for waking up EH handler
   - by Greg Kroah-Hartman @ 2024-03-06 6:46 UTC [6%]
4. CVE-2023-52592: libbpf: Fix NULL pointer dereference in bpf_object__collect_prog_relos
   - by Greg Kroah-Hartman @ 2024-03-06 6:45 UTC [6%]
5. CVE-2023-52591: reiserfs: Avoid touching renamed directory if parent does not change
   - by Greg Kroah-Hartman @ 2024-03-06 6:45 UTC [6%]
6. CVE-2023-52590: ocfs2: Avoid touching renamed directory if parent does not change
   - by Greg Kroah-Hartman @ 2024-03-06 6:45 UTC [6%]

Bosch Smart Thermostat Feels the Heat From Firmware Bug

The vulnerability in a popular hospitality industry gadget allows attackers to take over the device, pivot into the user's network, or brick the device entirely, rendering HVAC unusable.

Nathan Eddy, Contributing Writer
January 16, 2024
3 Min Read

Software continues to have bugs

VULNERABILITIES

Firefox 116 Patches High-Severity Vulnerabilities

Firefox 116 was released with patches for 14 CVEs, including nine high-severity vulnerabilities, some of which can lead to remote code execution or sandbox escapes.

By Ionut Arghire
August 2, 2023

linux-cve-announce.vger.kernel.org archive mirror
CVE search help / color / mirror / Atom feed

Search results ordered by [date|relevance] view[summary|nested|Atom feed]
download mbox.gz: results only | full threads

1. CVE-2024-26626: ipmr: fix kernel panic when forwarding mcast packets
   - by Greg Kroah-Hartman @ 2024-03-06 6:46 UTC [4%]
2. CVE-2024-26628: drm/amdkfd: Fix lock dependency warning
   - by Greg Kroah-Hartman @ 2024-03-06 6:46 UTC [6%]
3. CVE-2024-26627: scsi: core: Move scsi_host_busy() out of host lock for waking up EH handler
   - by Greg Kroah-Hartman @ 2024-03-06 6:46 UTC [6%]
4. CVE-2023-52592: libbpf: Fix NULL pointer dereference in bpf_object__collect_prog_relos
   - by Greg Kroah-Hartman @ 2024-03-06 6:45 UTC [6%]
5. CVE-2023-52591: reiserfs: Avoid touching renamed directory if parent does not change
   - by Greg Kroah-Hartman @ 2024-03-06 6:45 UTC [6%]
6. CVE-2023-52590: ocfs2: Avoid touching renamed directory if parent does not change
   - by Greg Kroah-Hartman @ 2024-03-06 6:45 UTC [6%]

Bosch Smart Thermostat Feels the Heat From Firmware Bug

The vulnerability in a popular hospitality industry gadget allows attackers to take over the device, pivot into the user's network, or brick the device entirely, rendering HVAC unusable.

Nathan Eddy, Contributing Writer
January 16, 2024
3 Min Read

POTUS executive order aims to keep U.S. ports safe from cyberattacks

Steve Zurier February 21, 2024


port of seattle with downtown skyline early morning

Software continues to have bugs

VULNERABILITIES

Firefox 116 Patches High-Severity Vulnerabilities

Firefox 116 was released with patches for 14 CVEs, including nine high-severity vulnerabilities, some of which can lead to remote code execution or sandbox escapes.

By Ionut Arghire
August 2, 2023

linux-cve-announce.vger.kernel.org archive mirror
CVE search help / color / mirror / Atom feed

Search results ordered by [date|relevance] view[summary|nested|Atom feed]
download mbox.gz: results only | full threads

1. CVE-2024-26626: ipmr: fix kernel panic when forwarding mcast packets
   - by Greg Kroah-Hartman @ 2024-03-06 6:46 UTC [4%]
2. CVE-2024-26628: drm/amdkfd: Fix lock dependency warning
   - by Greg Kroah-Hartman @ 2024-03-06 6:46 UTC [6%]
3. CVE-2024-26627: scsi: core: Move scsi_host_busy() out of host lock for waking up EH handler
   - by Greg Kroah-Hartman @ 2024-03-06 6:46 UTC [6%]
4. CVE-2023-52592: libbpf: Fix NULL pointer dereference in bpf_object__collect_prog_relos
   - by Greg Kroah-Hartman @ 2024-03-06 6:45 UTC [6%]
5. CVE-2023-52591: reiserfs: Avoid touching renamed directory if parent does not change
   - by Greg Kroah-Hartman @ 2024-03-06 6:45 UTC [6%]
6. CVE-2023-52590: ocfs2: Avoid touching renamed directory if parent does not change
   - by Greg Kroah-Hartman @ 2024-03-06 6:45 UTC [6%]

Bosch Smart Thermostat Feels the Heat From Firmware Bug

The vulnerability in a popular hospitality industry gadget allows attackers to take over the device, pivot into the user's network, or brick the device entirely, rendering HVAC unusable.

Nathan Eddy, Contributing Writer
January 16, 2024
3 Min Read

POTUS executive order aims to keep U.S. ports safe from cyberattacks

Steve Zurier February 21, 2024

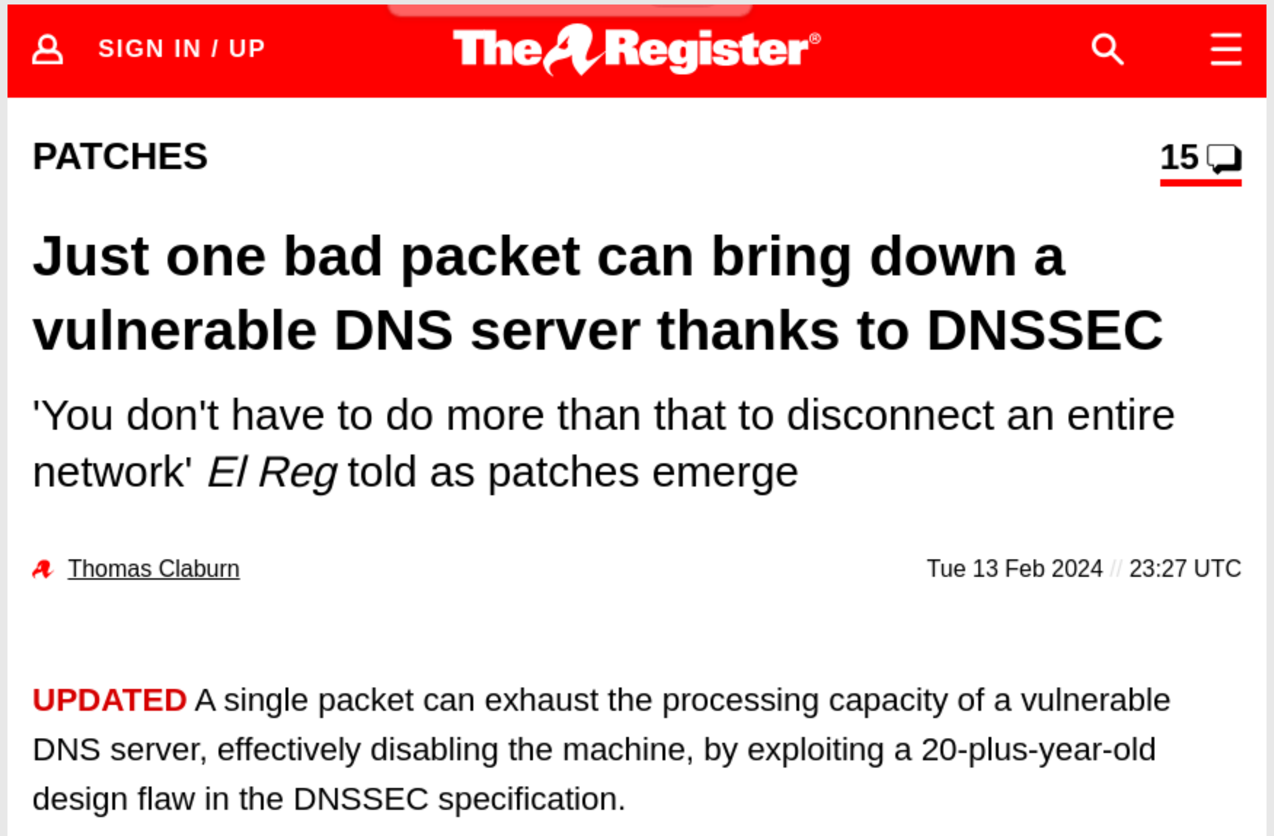
port of seattle with downtown skyline early morning

A software update bricked Rivian infotainment systems

Harri Weber @harriblogs / 12:24 PM EST • November 14, 2023
Comment

Weaponization of Bugs is a Growing Issue

Weaponization of Bugs is a Growing Issue



SIGN IN / UP The Register

PATCHES 15

Just one bad packet can bring down a vulnerable DNS server thanks to DNSSEC

'You don't have to do more than that to disconnect an entire network' *El Reg* told as patches emerge

Thomas Claburn Tue 13 Feb 2024 // 23:27 UTC

UPDATED A single packet can exhaust the processing capacity of a vulnerable DNS server, effectively disabling the machine, by exploiting a 20-plus-year-old design flaw in the DNSSEC specification.

Weaponization of Bugs is a Growing Issue

SIGN IN / UP The Register

PATCHES 15

Just one bad packet can bring down a vulnerable DNS server thanks to DNSSEC

'You don't have to do more than that to disconnect an entire network' *El Reg* told as patches emerge

Thomas Claburn Tue 13 Feb 2024 // 23:27 UTC

UPDATED A single packet can exhaust the processing capacity of a vulnerable DNS server, effectively disabling the machine, by exploiting a 20-plus-year-old design flaw in the DNSSEC specification.

FORBES > INNOVATION > CYBERSECURITY

SolarWinds Is A Game Changer – You Cannot Sugarcoat Cybersecurity

Stewart Room Contributor

I write about Data Protection, Privacy and Cyber Security law

Follow

Weaponization of Bugs is a Growing Issue

SIGN IN / UP — The Register

PATCHES — 15

Just one bad packet can bring down a vulnerable DNS server thanks to DNSSEC

'You don't have to do more than that to disconnect an entire network' *El Reg* told as patches emerge

Thomas Claburn — Tue 13 Feb 2024 // 23:27 UTC

UPDATED A single packet can exhaust the processing capacity of a vulnerable DNS server, effectively disabling the machine, by exploiting a 20-plus-year-old design flaw in the DNSSEC specification.

FORBES > INNOVATION > CYBERSECURITY

SolarWinds Is A Game Changer – You Cannot Sugarcoat Cybersecurity

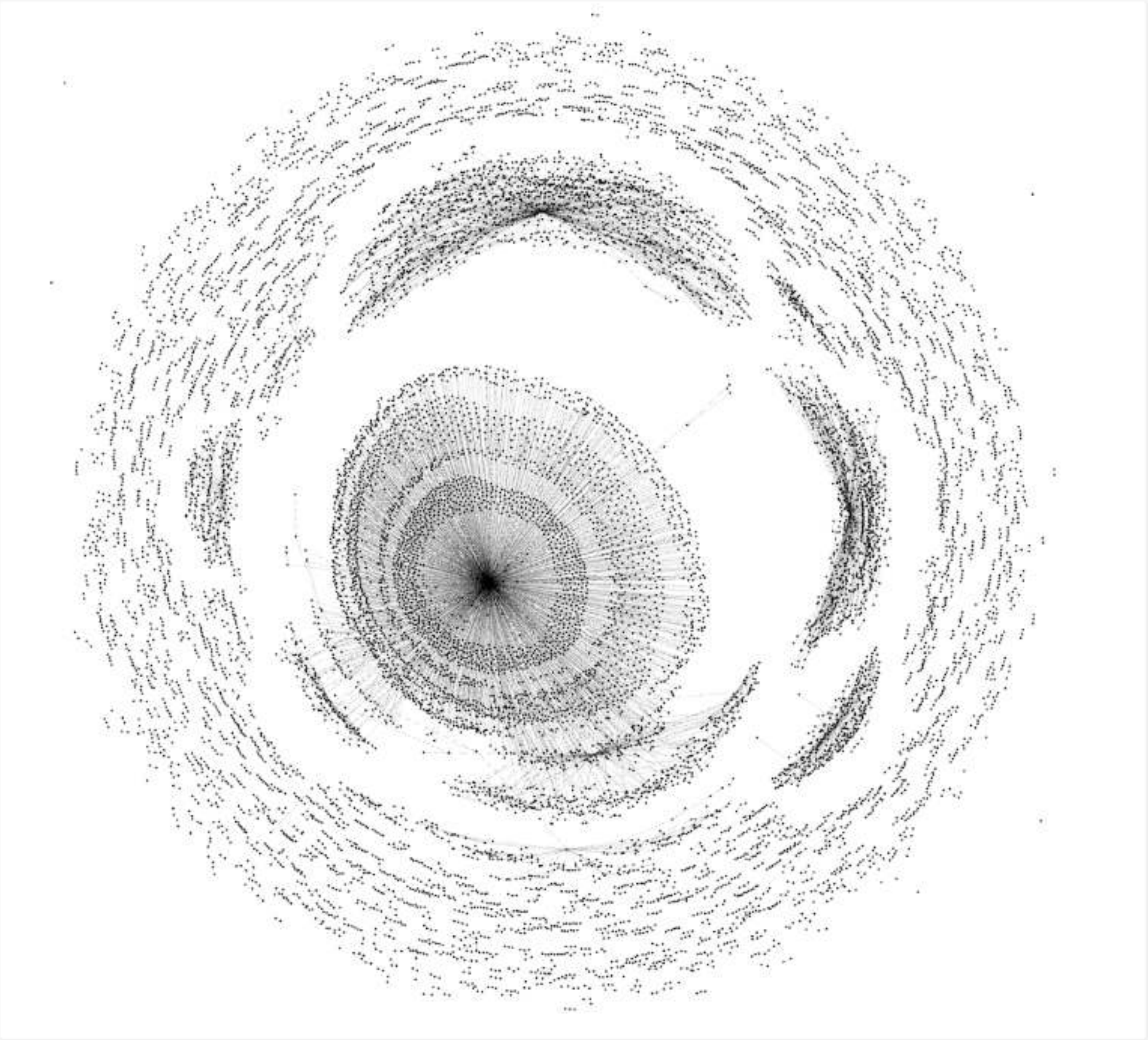
Stewart Room Contributor

I write about Data Protection, Privacy and Cyber Security law

Follow

An analysis of an in-the-wild iOS Safari WebContent to GPU Process exploit

By Ian Beer



A graph representation of the sandbox escape NSExpression payload

Weaponization of Bugs is a Growing Issue

We depend on tools to automatically find bugs

LLVM COMPILER INFRASTRUCTURE

LLVM Home | Documentation » Reference » libFuzzer – a library for coverage-guided fuzz

libFuzzer – a library for coverage-guided fuzz testing.

oss-fuzz oss-fuzz ▾ | New issue | All issues ▾ | Type=Bug label:clusterfuzz -status:Duplicate,WontFix | Sign in

1 - 100 of 38361 Next › | List | Grid | Chart

ID	Type	Component	Status	Proj	Reported	Owner	Summary + Labels
17	Bug	----	Verified	freetype2	----	----	Out-of-memory in freetype2_fuzzer ClusterFuzz Reproducible
47	Bug	----	Verified	sqlite3	----	----	Crash in sqlite3ExprCodeTemp ClusterFuzz Reproducible
52	Bug	----	Verified	freetype2	----	----	Crash in t1_builder_add_point ClusterFuzz Reproducible
62	Bug	----	Verified	libchewing	----	----	Heap-buffer-overflow in _Inner_InternalSpecialSymbol ClusterFuzz Reproducible
64	Bug	----	Verified	libchewing	----	----	Heap-buffer-overflow in ChewingIsChiAt ClusterFuzz Reproducible
65	Bug	----	Verified	libchewing	----	----	Crash in GetUint24 ClusterFuzz Reproducible
67	Bug	----	Verified	libchewing	----	----	Heap-buffer-overflow in ueStrNBytes ClusterFuzz Reproducible
68	Bug	----	Verified	libchewing	----	----	Negative-size-param in ChewingKillChar ClusterFuzz Reproducible
69	Bug	----	Verified	libchewing	----	----	Heap-use-after-free in GetUint16 ClusterFuzz Reproducible
70	Bug	----	Verified	libchewing	----	----	Floating-point-exception in OpenSymbolChoice ClusterFuzz Reproducible

syzbot Linux

Open [946] | Subsystems | Fixed [5111] | Invalid [12230] | Missing Backports [83]

Instances [tested repos]:

Name	Last active	Uptime	Corpus	Coverage	Crashes	Execs
ci-qemu-upstream	now	3h02m	31360	473977	10	769655
ci-qemu-upstream-386	now	2h53m	30998	481789	77	518064
ci-qemu2-arm32	now	3h11m	19480	28089	3	196968
ci-qemu2-arm64	now	2h36m	9853	15986	1	49283
ci-qemu2-arm64-compat	now	2h48m	9701	15525		47246
ci-qemu2-arm64-mte	now	3h02m	37199	50867		174128
ci-qemu2-riscv64	now	3h09m	381	36935	51	3317
ci-upstream-bpf-kasan-gce	now	14h32m	17911	139168	13	795393

Automated Bug Finding Techniques

Static

Static Data-flow Analysis

Formal Methods

Model Checking

Static Symbolic Execution

Compilers

Automated Bug Finding Techniques

Static

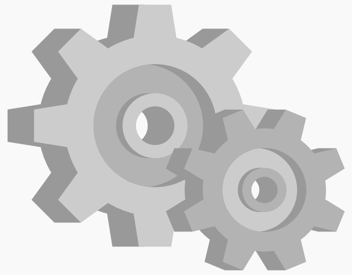
Static Data-flow Analysis

Formal Methods

Model Checking

Static Symbolic Execution

Compilers



Dynamic

Unit Testing

Memory Error Detection

Sanitization

Dynamic Symbolic Execution

Fuzz Testing

Fuzz Testing

Automatically providing unexpected data to a program

Fuzz Testing

Automatically providing unexpected data to a program

Normal Tests

```
bash -c "echo this is a test"
```

```
message="Hello World"; bash -c echo $message
```

Fuzz Testing

Automatically providing unexpected data to a program

Normal Tests

```
bash -c "echo this is a test"

message="Hello World"; bash -c echo $message
```

Fuzz Tests

```
A='() { 0 <<a <<b <<c <<d <<e <<f <<g <<h <<i <<j <<k <<l <<m; }' :

A='() { x() { _; }; x() { _; } <<a; }' bash -c :

A='() { _; } >_[$($())] { echo hi; id; }' bash -c :
```

Fuzz Testing

Automatically providing unexpected data to a program

Normal Tests

```
bash -c "echo this is a test"
```

```
message="Hello World"; bash -c echo $message
```

Fuzz Tests

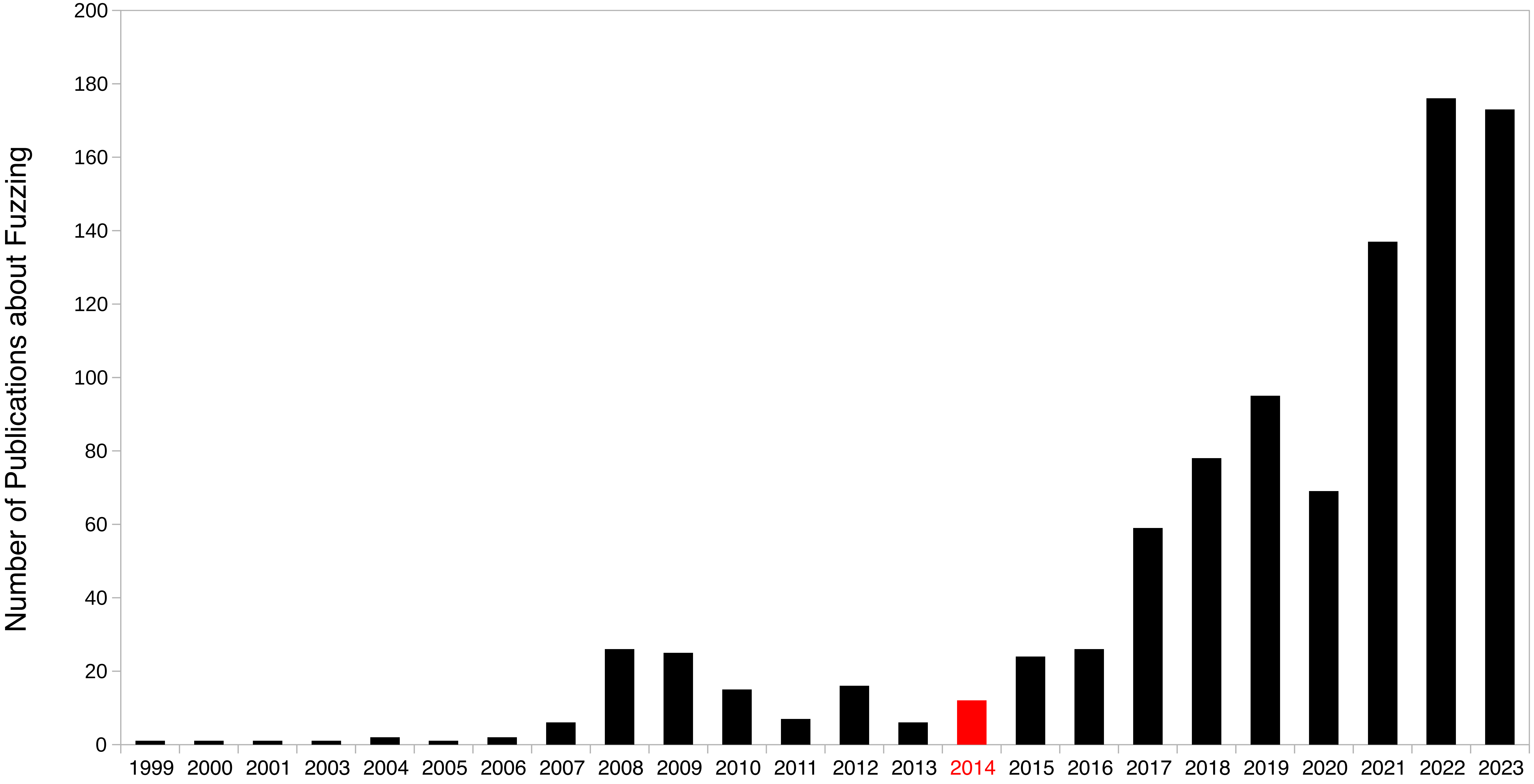
```
A='() { 0 <<a <<b <<c <<d <<e <<f <<g <<h <<i <<j <<k <<l <<m; }' :
```

```
A='() { x() { _; }; x() { _; } <<a; }' bash -c :
```

```
A='() { _; } >_[$($())] { echo hi; id; }' bash -c :
```

Shellshock
2014

Fuzz Testing

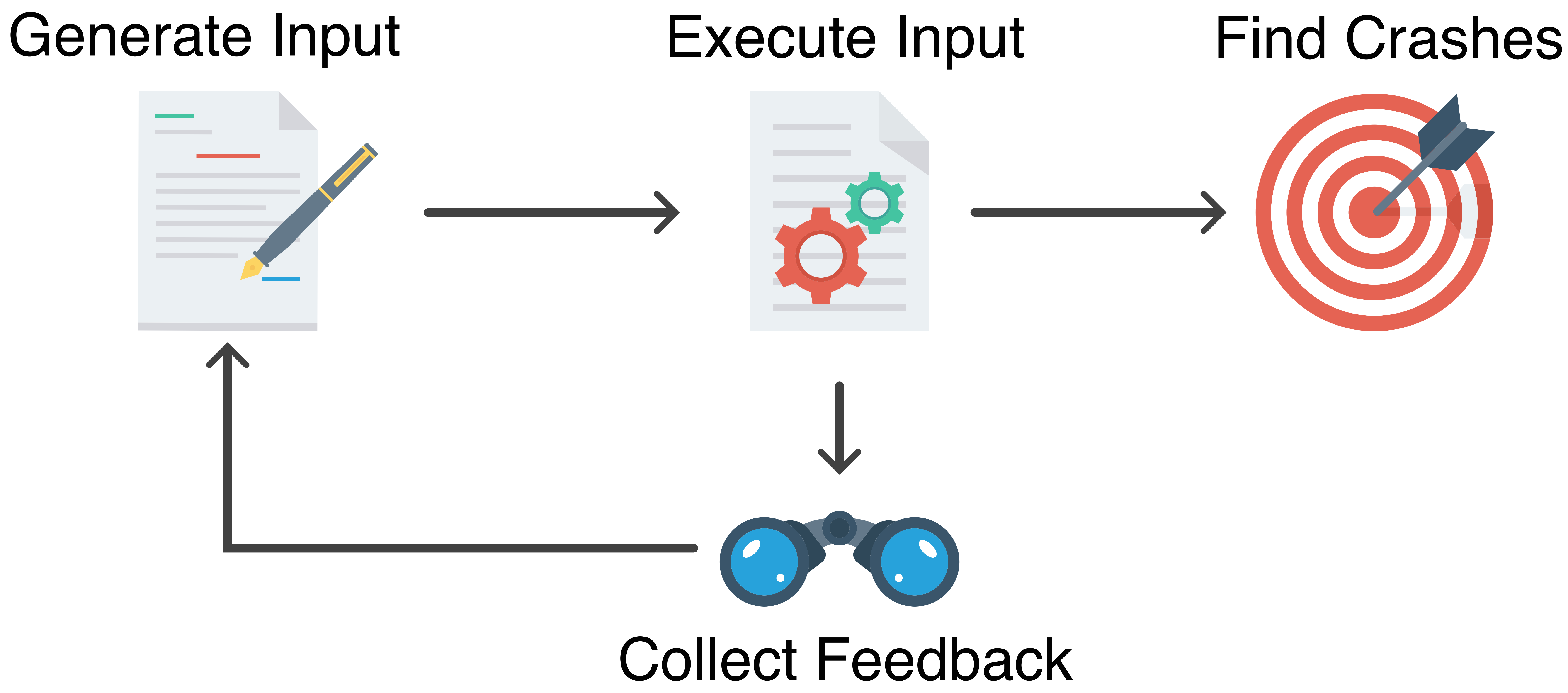


Shellshock
2014

How does Fuzzing Work?

Generate Input → Execute Input → Find Crashes

How does Fuzzing Work?



Fuzzing large systems is difficult

Fuzzing large systems is difficult

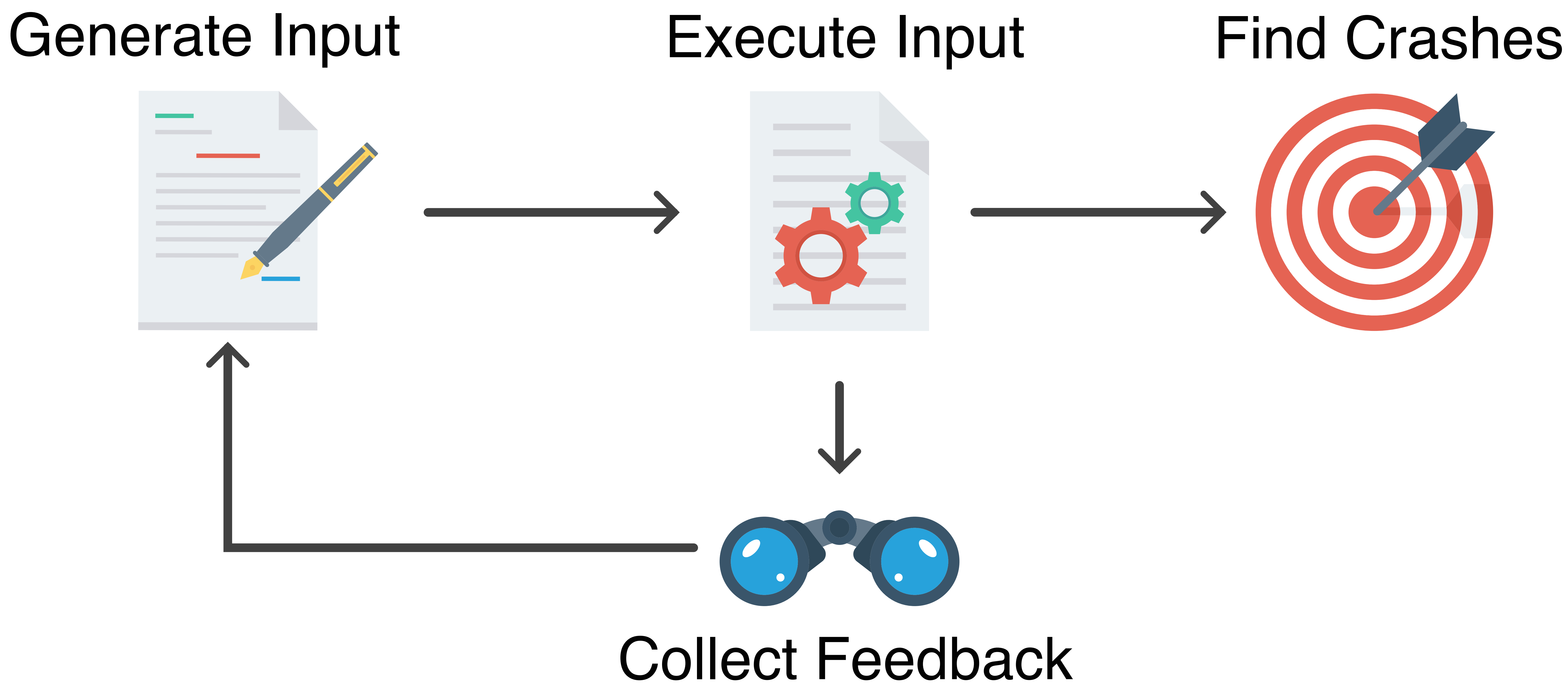
syzkaller - kernel fuzzer

ci passing | oss-fuzz fuzzing | go report A+ | codecov 63% | GO reference | License Apache 2.0

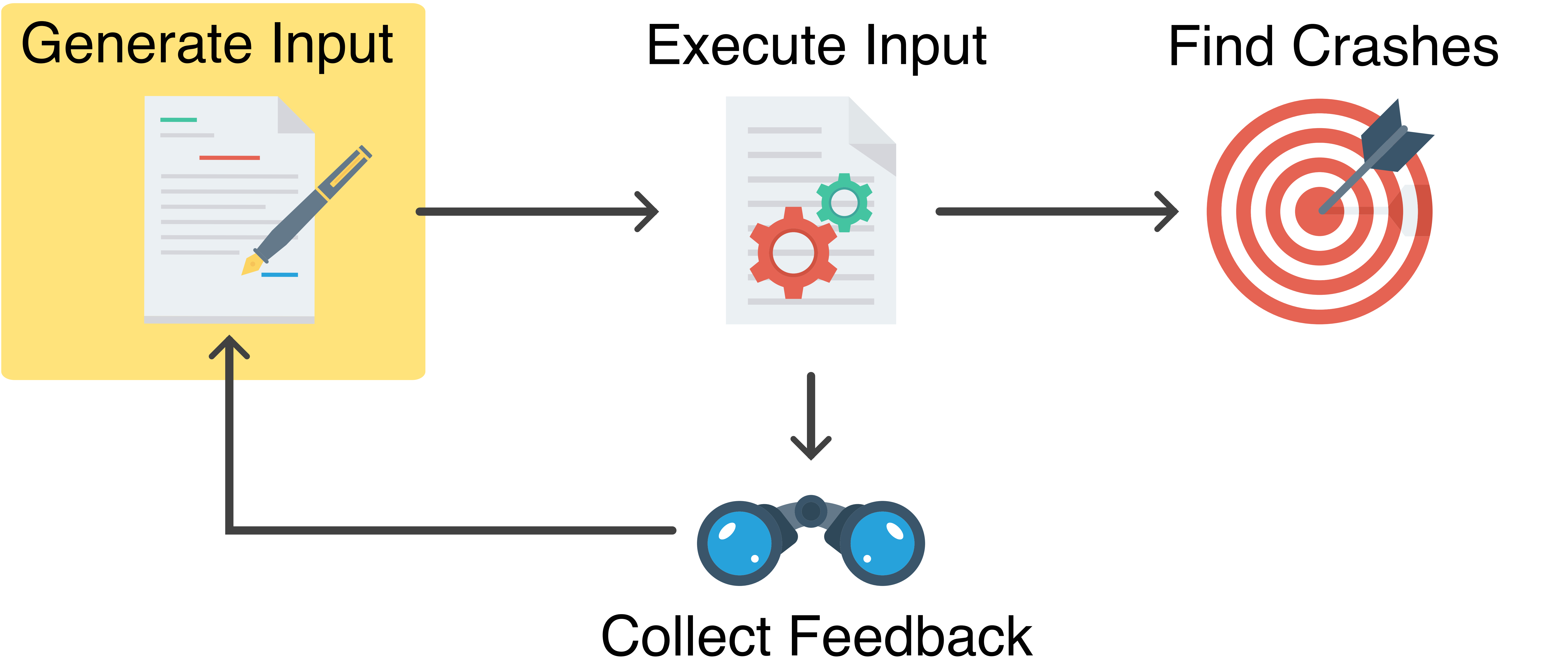
`syzkaller` (`[siːzˈkɔːlə]`) is an unsupervised coverage-guided kernel fuzzer.
Supported OSes: `Akaros`, `FreeBSD`, `Fuchsia`, `gVisor`, `Linux`, `NetBSD`, `OpenBSD`, `Windows`.

~600k Lines of Code to fuzz a single system

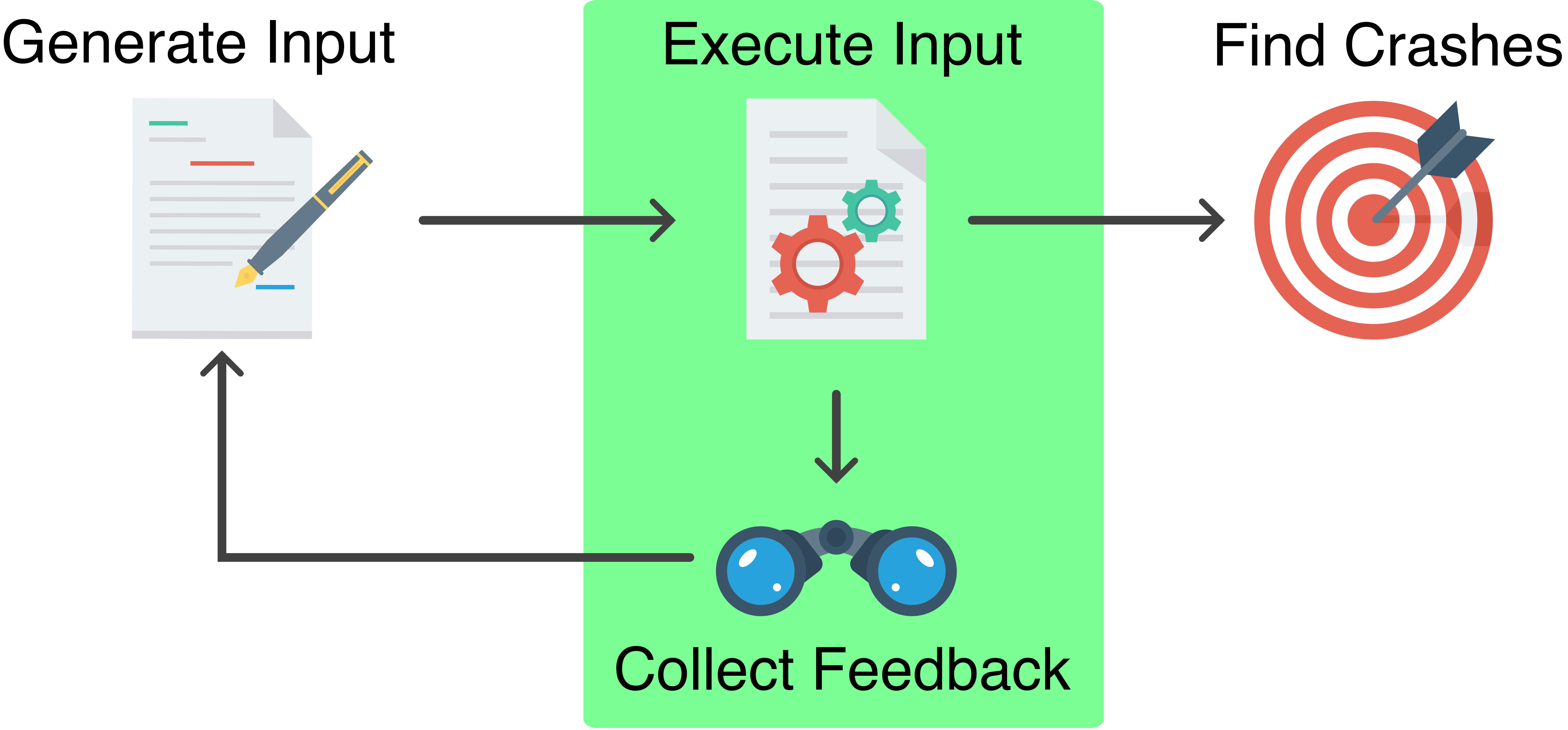
Generating Inputs to Large Systems is Difficult



Security Researchers spend effort generating better inputs

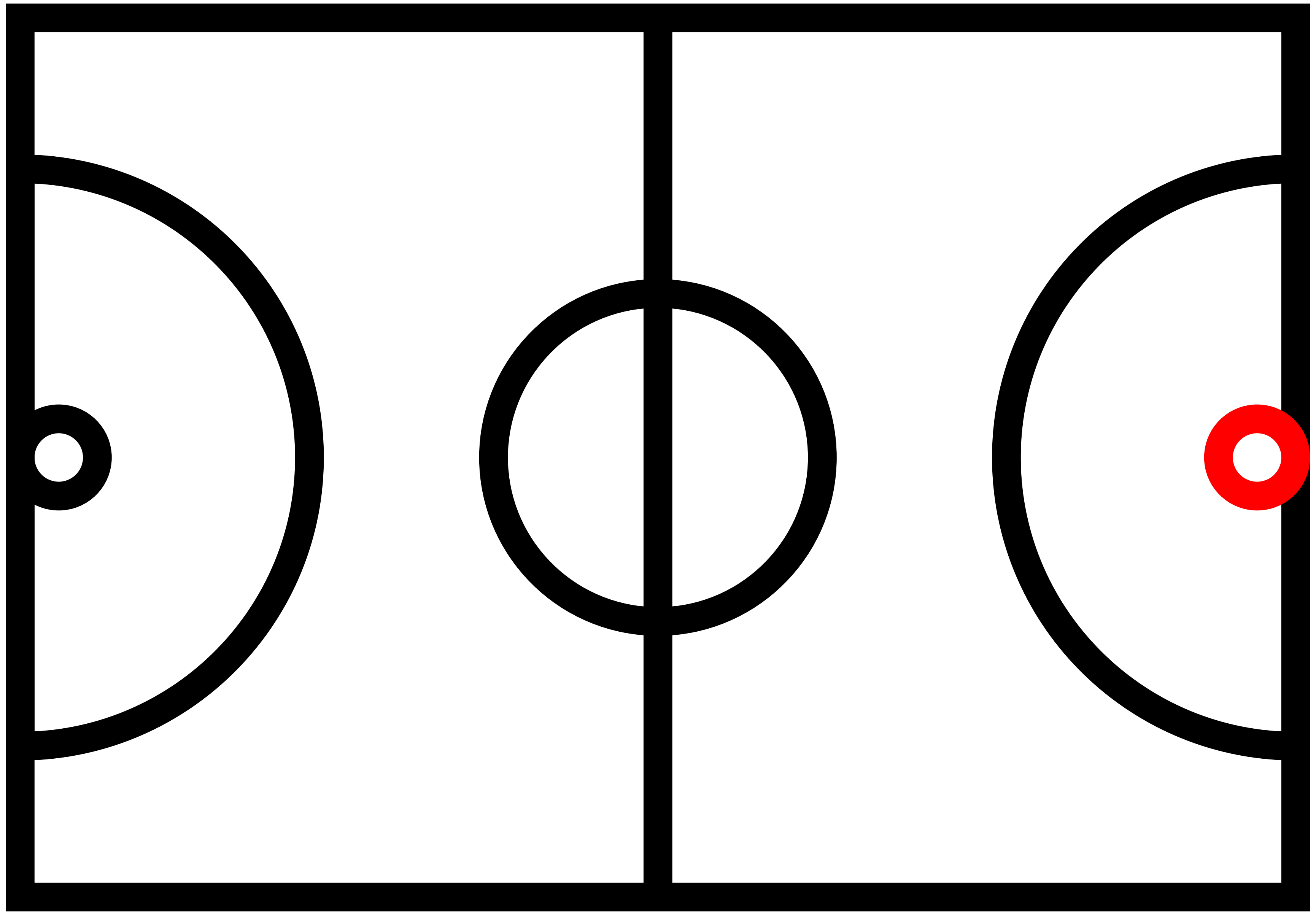


Reshaping makes large targets conducive to fuzzing with small modifications to the execution and feedback stages

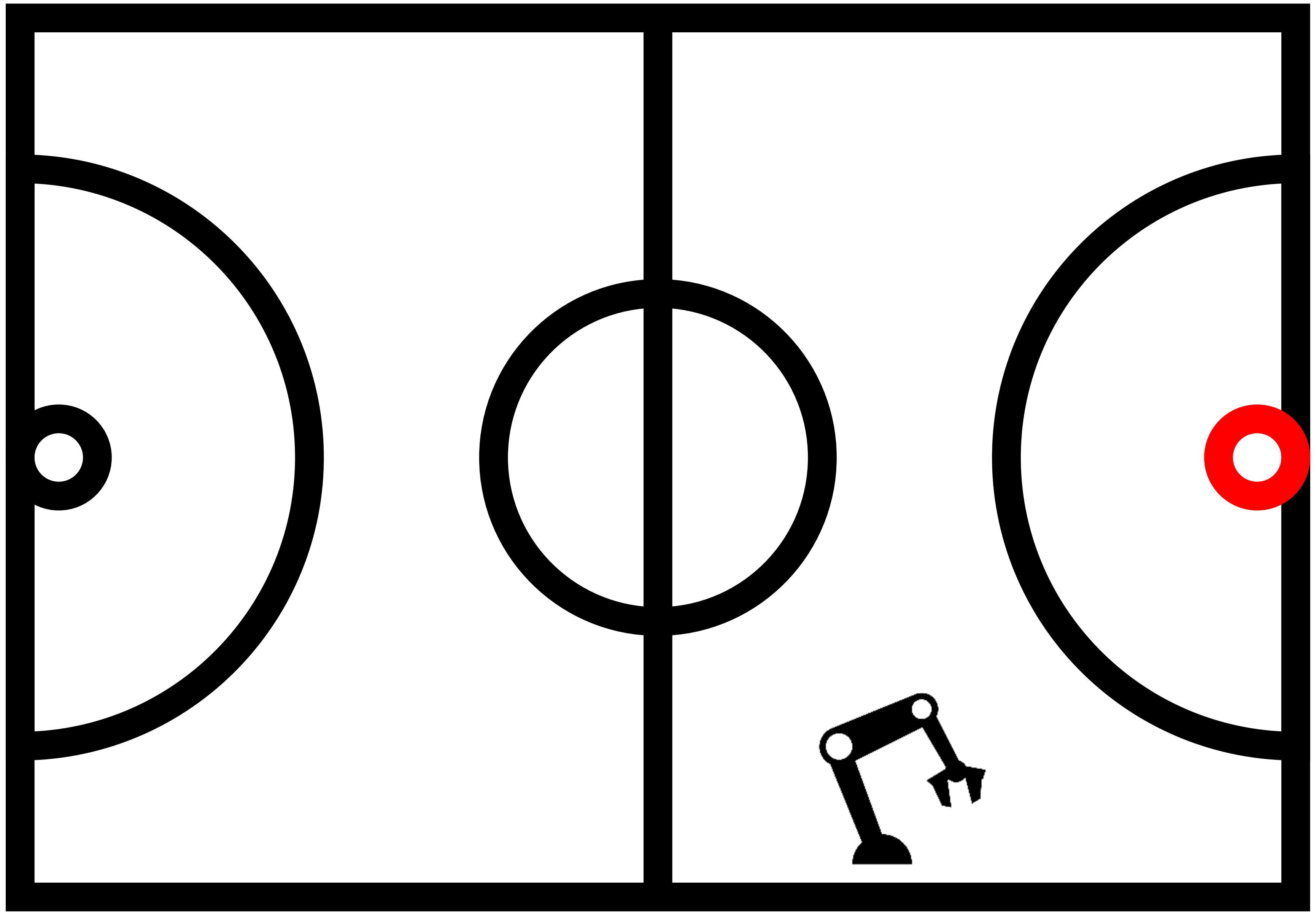


Reshaping: modifying a target to make it conducive to fuzzing without granular harnesses/grammars

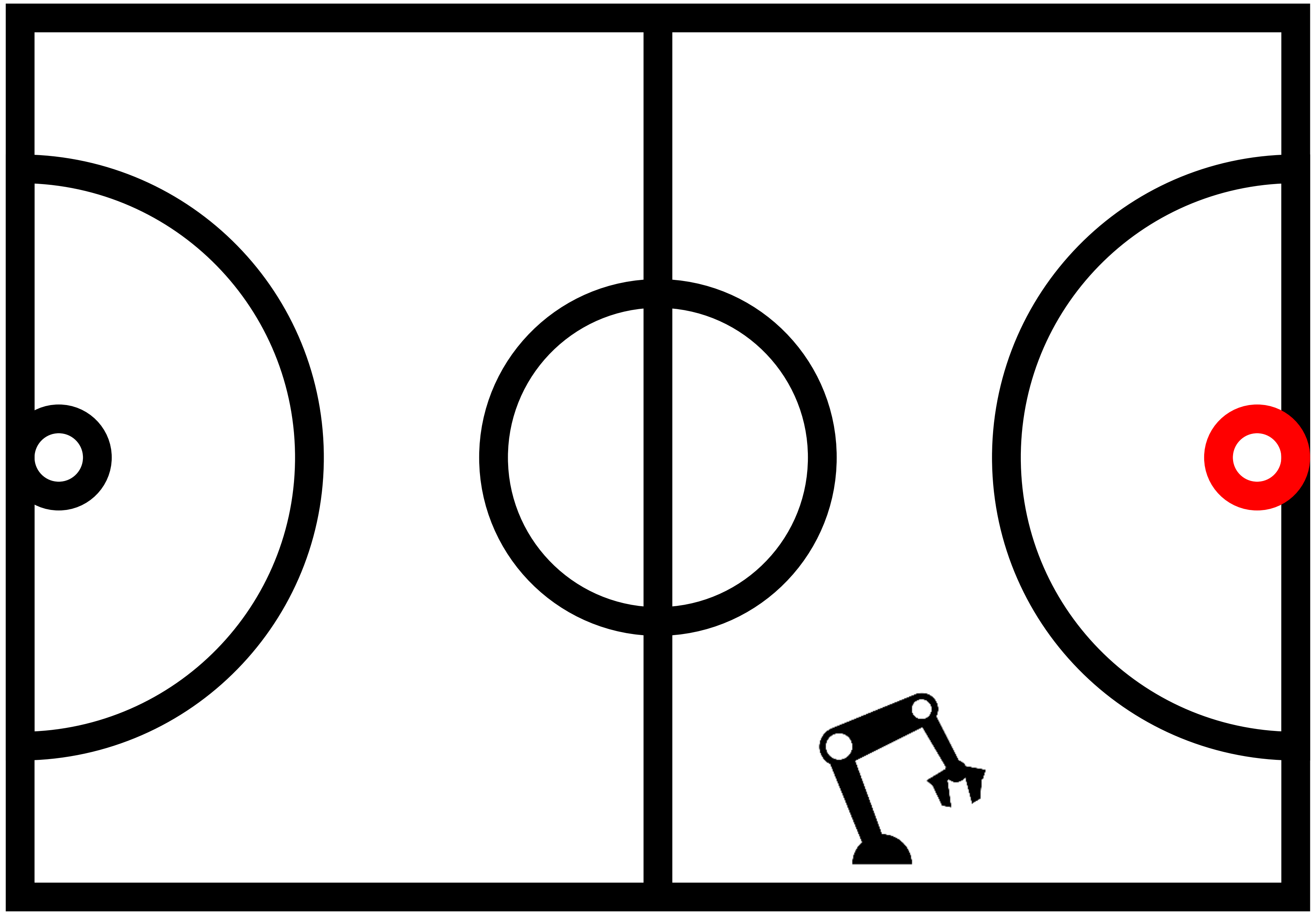
Reshaping: modifying a target to make it conducive to fuzzing without granular harnesses/grammars



Reshaping: modifying a target to make it conducive to fuzzing without granular harnesses/grammars

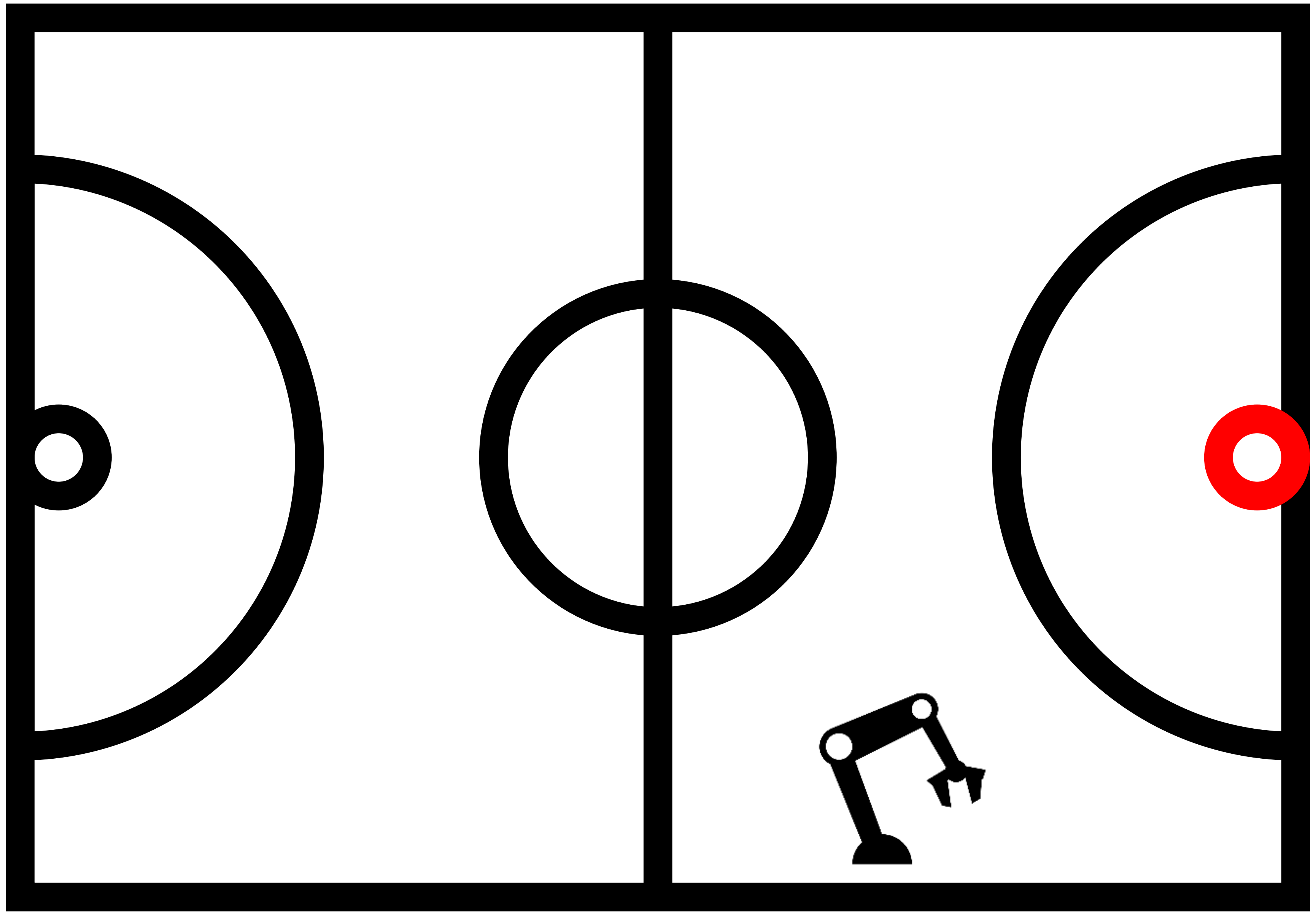


Reshaping: modifying a target to make it conducive to fuzzing without granular harnesses/grammars



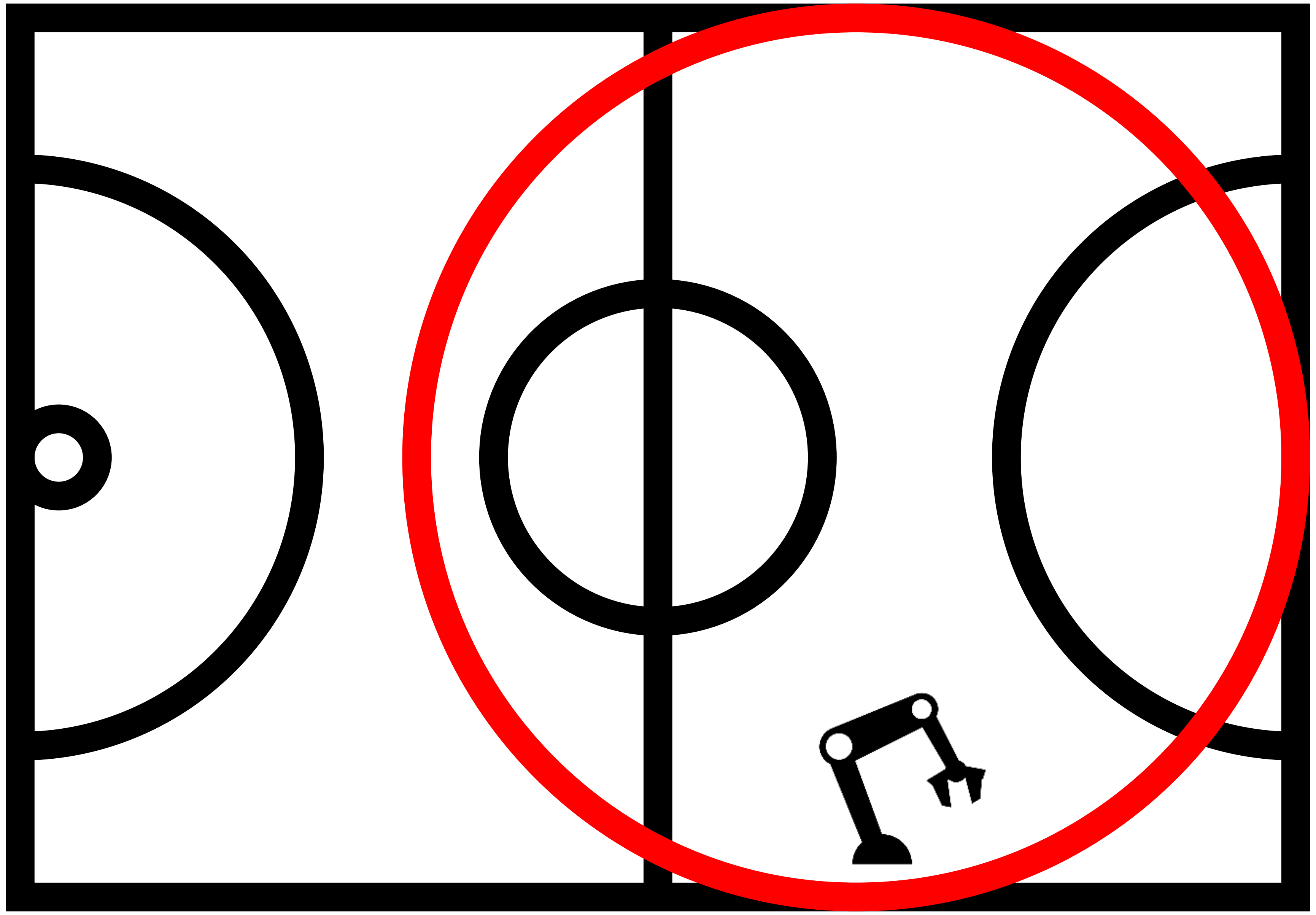
Grammar: Teaching the fuzzer to play Basketball

Reshaping: modifying a target to make it conducive to fuzzing without granular harnesses/grammars



Reshaping: Making Basketball Easier to play

Reshaping: modifying a target to make it conducive to fuzzing without granular harnesses/grammars



Reshaping: Making Basketball Easier to play

Reshaping: modifying a target to make it conducive to fuzzing without granular harnesses/grammars

Thesis:

Input-space reshaping is more effective than grammar-based harnessing approaches for fuzzing complex targets.

Research Questions

Is reshaped fuzzing…

RQ1: effective at finding bugs?

Research Questions

Is reshaped fuzzing...

RQ1: effective at finding bugs?

RQ2: competitve with other approaches on coverage-achieved?

Research Questions

Is reshaped fuzzing…

RQ1: effective at finding bugs?

RQ2: competitve with other approaches on coverage-achieved?

RQ3: applicable to a diverse set of targets?

Research Questions

Is reshaped fuzzing…

RQ1: effective at finding bugs?

RQ2: competitve with other approaches on coverage-achieved?

RQ3: applicable to a diverse set of targets?

RQ4: beneficial even when grammars exist?

Research Questions

Is reshaped fuzzing...

RQ1: effective at finding bugs?

RQ2: competitve with other approaches on coverage-achieved?

RQ3: applicable to a diverse set of targets?

RQ4: beneficial even when grammars exist?

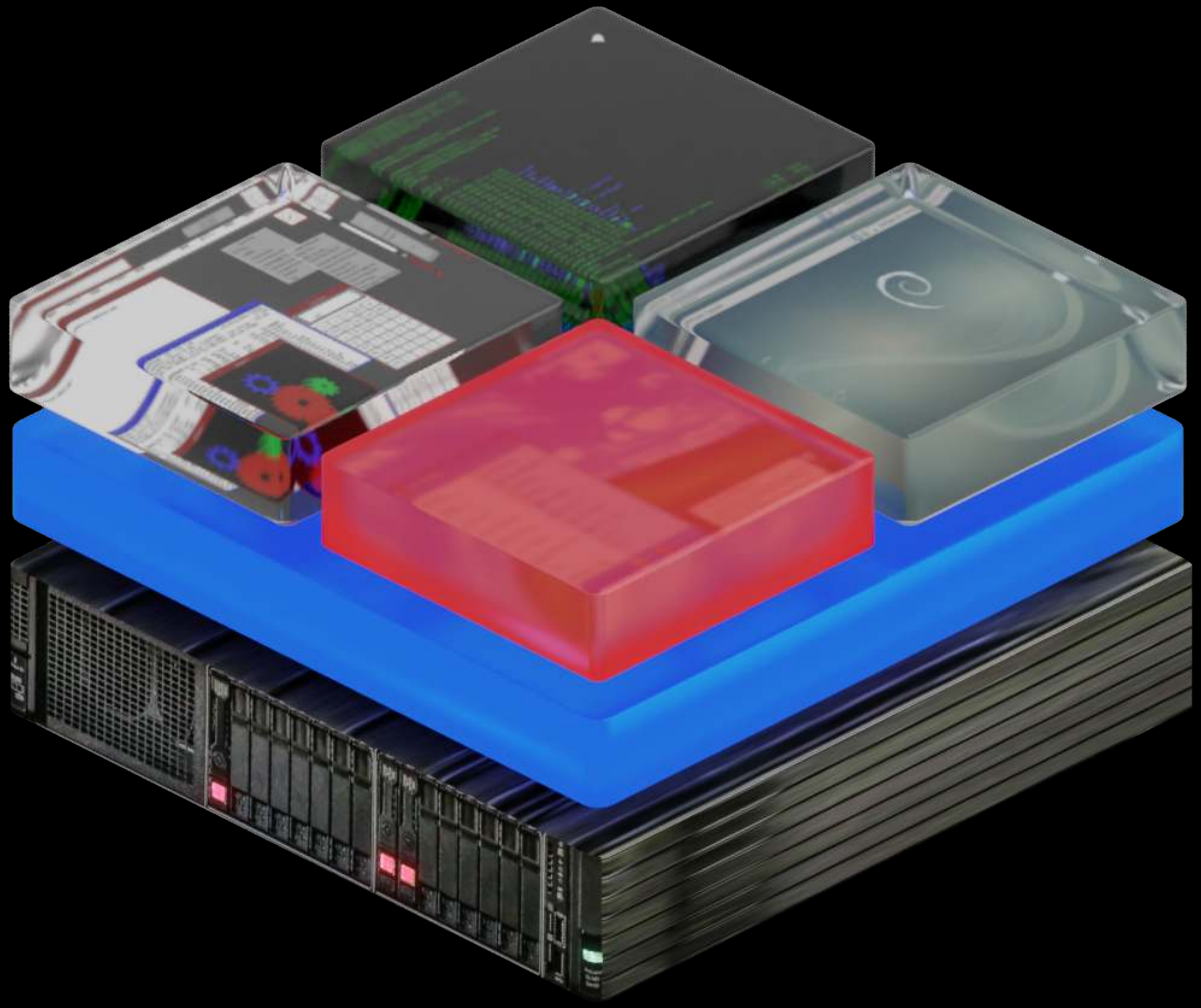
RQ5: compatible with other SoTA fuzzing techniques?

Outline

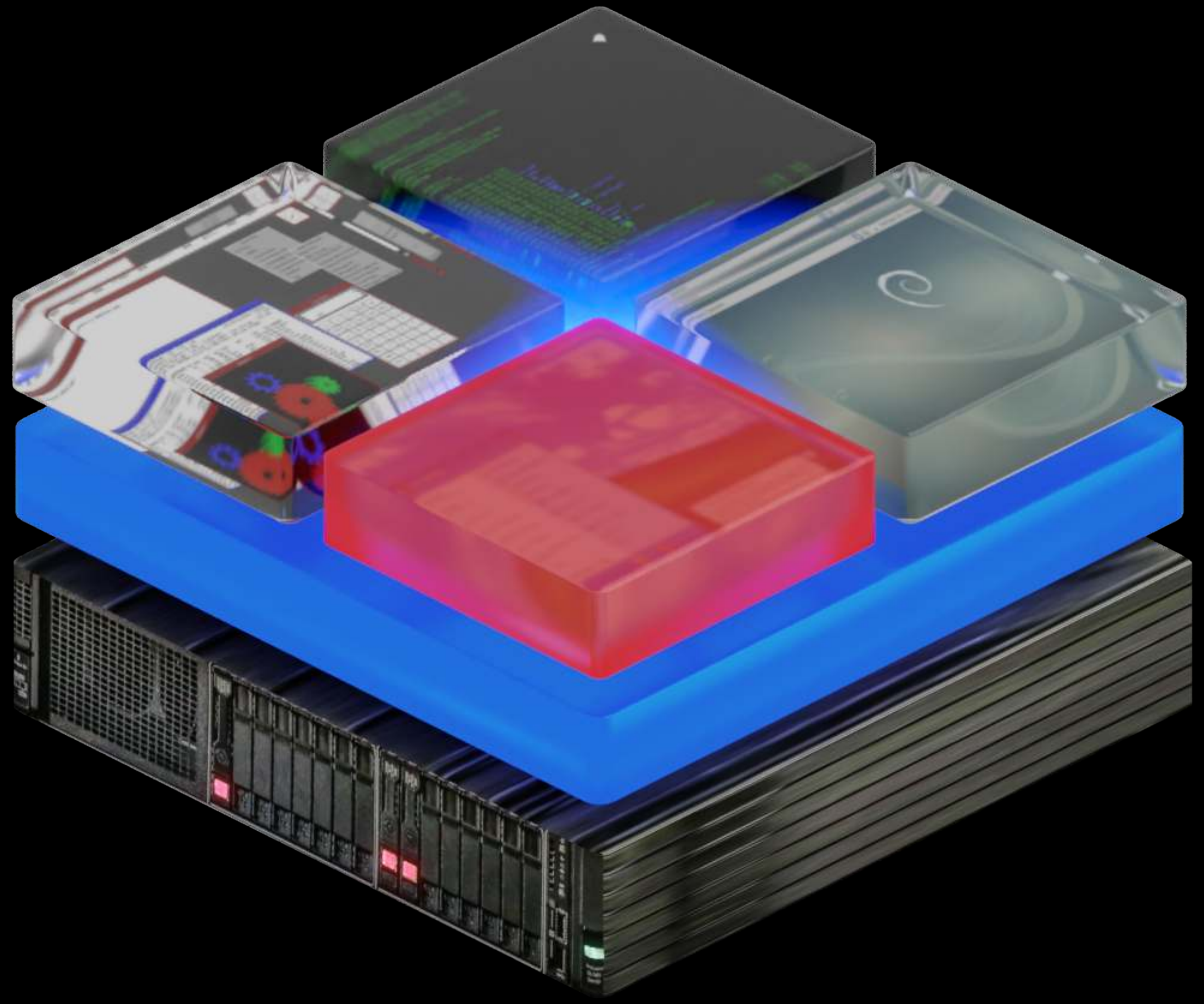
- **Introduction**
  - Motivation and Background
  - *Thesis Statement*
- **Source-based Hypervisor Fuzzing**
- **Binary Hypervisor Fuzzing**
- **Large-Scale Kernel Fuzzing**
- **Conclusion**
  - Summary of Contributions

Hypervisor Security

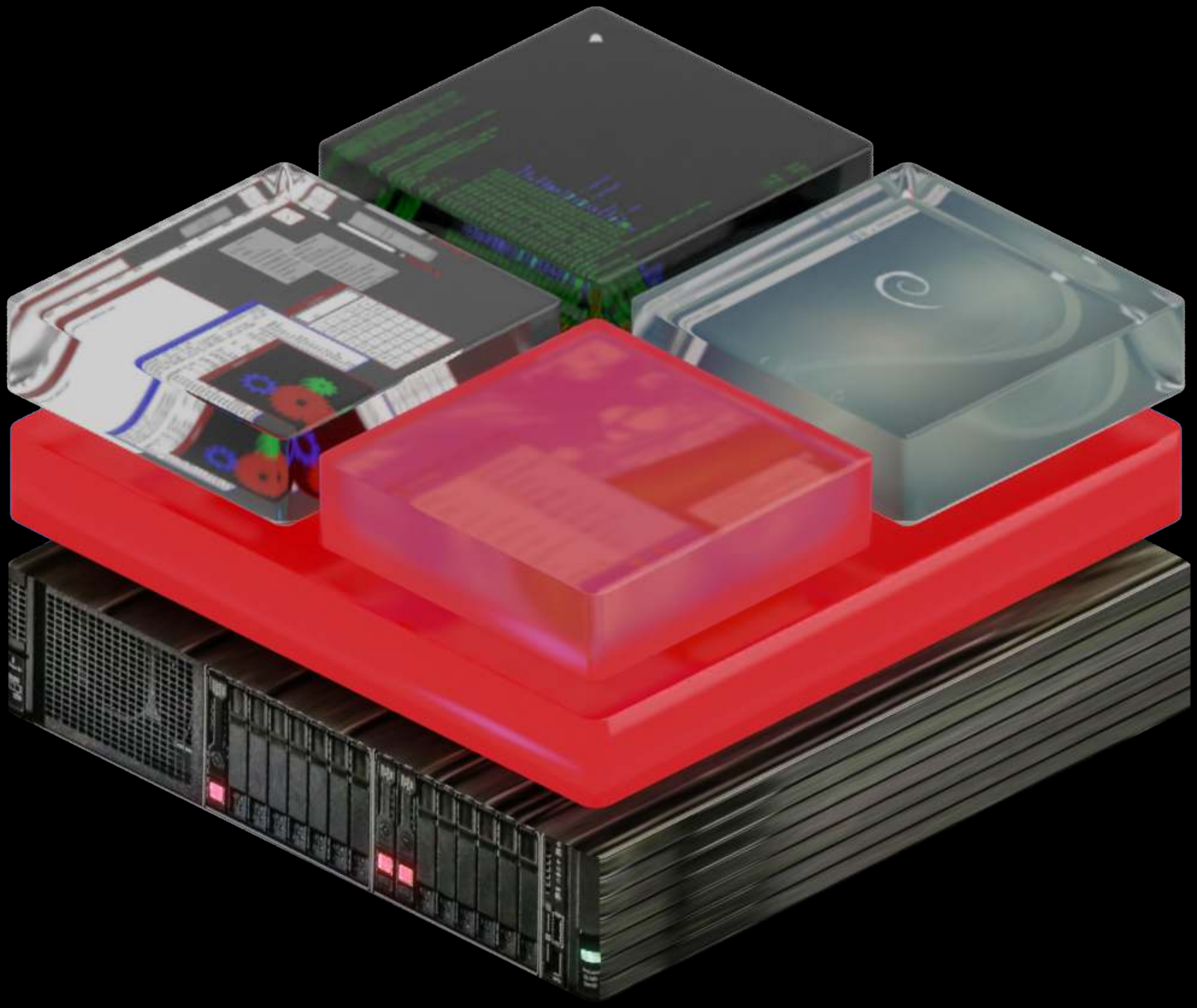
Hypervisor Security



Hypervisor Security



Hypervisor Security



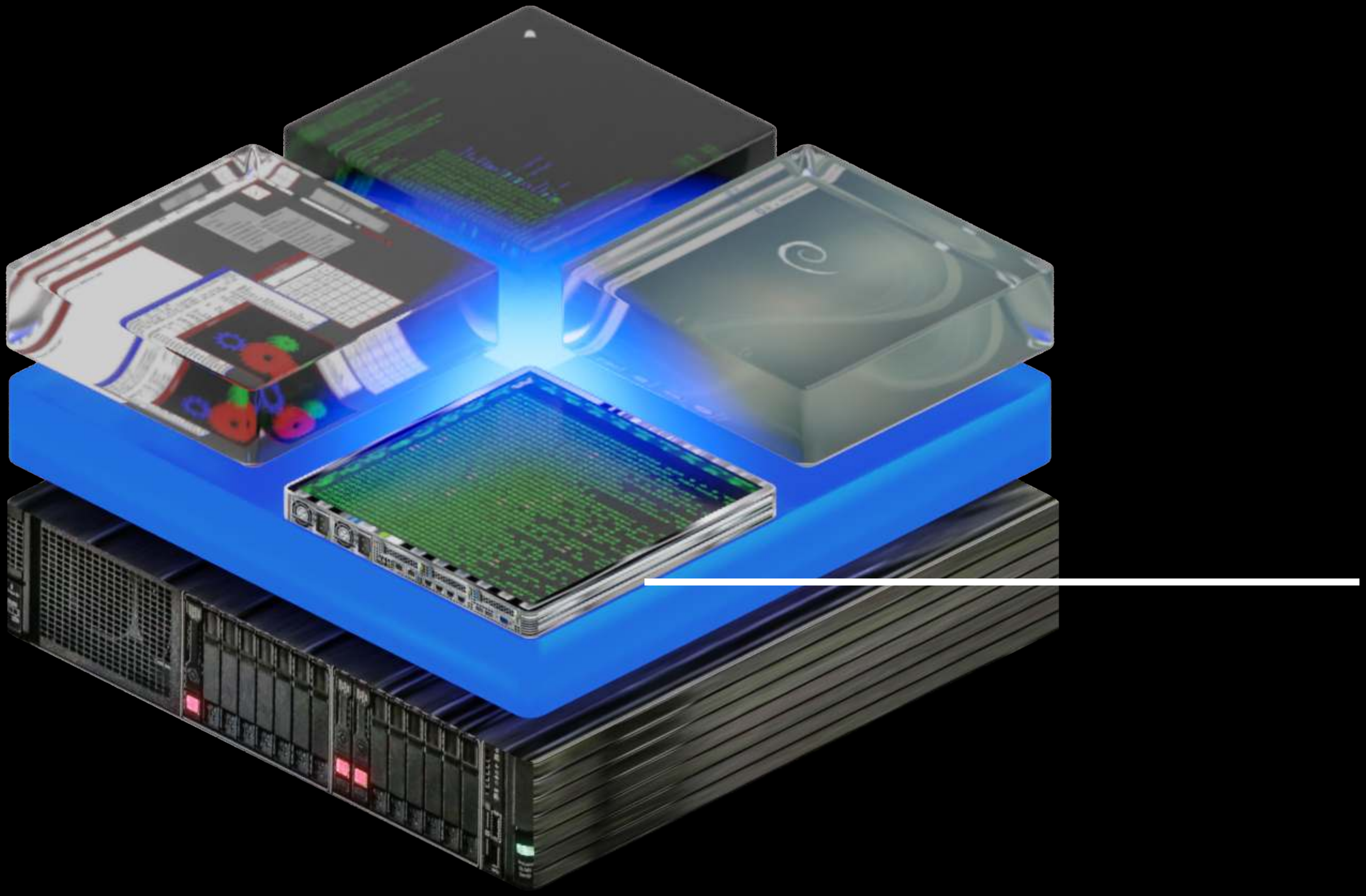
Hypervisor Security

Hypervisor Security

Hypervisor Security

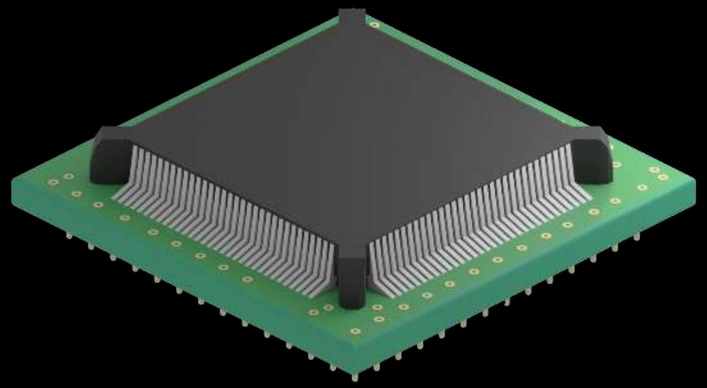
Virtual Devices

Hypervisor Security

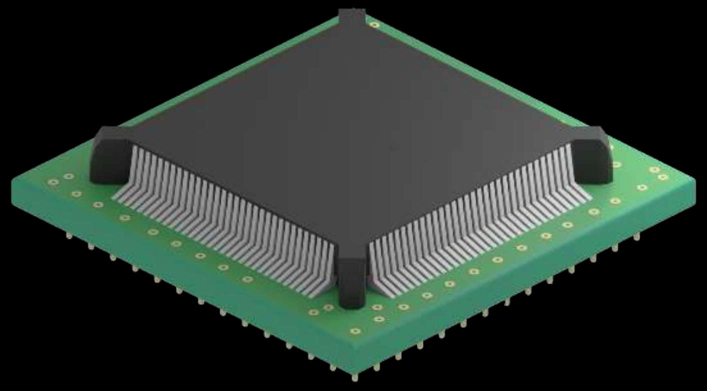


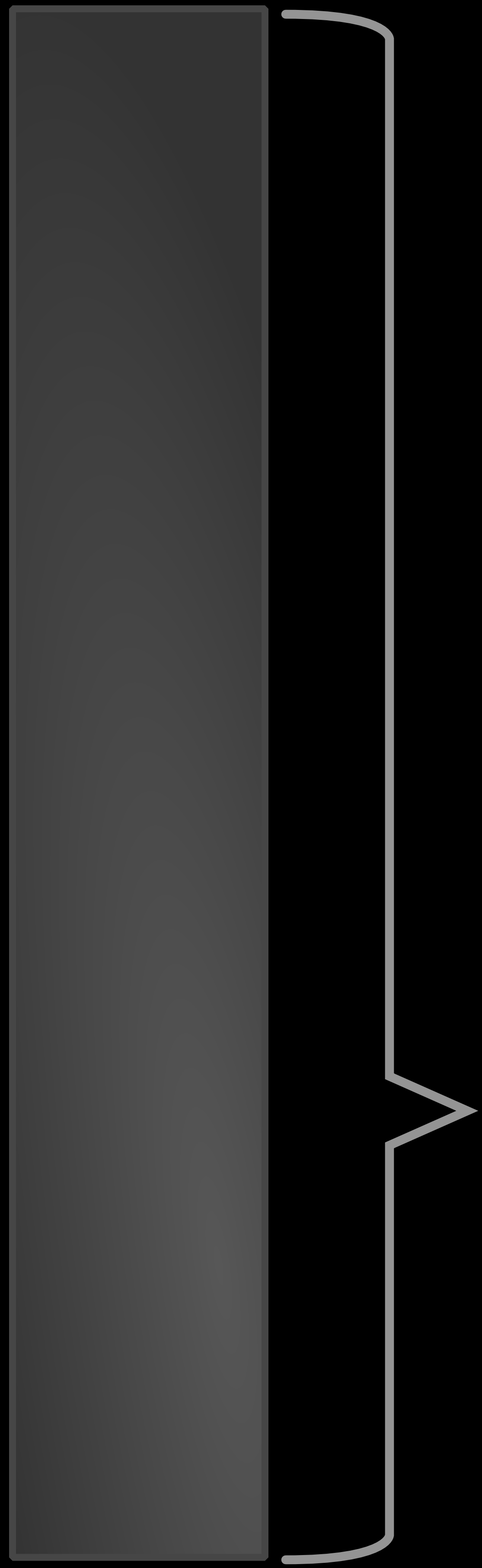
Virtual Devices

MMIO/PIO Example

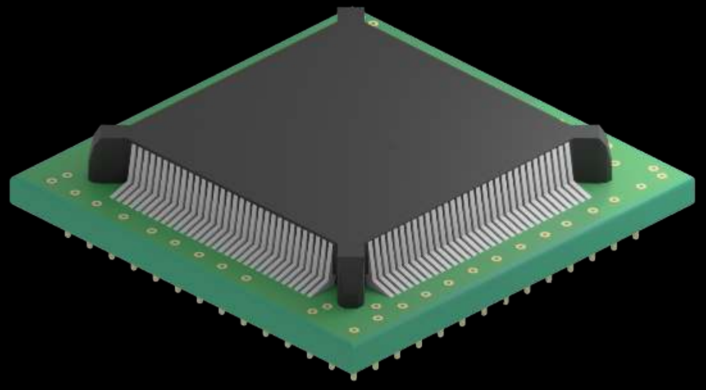


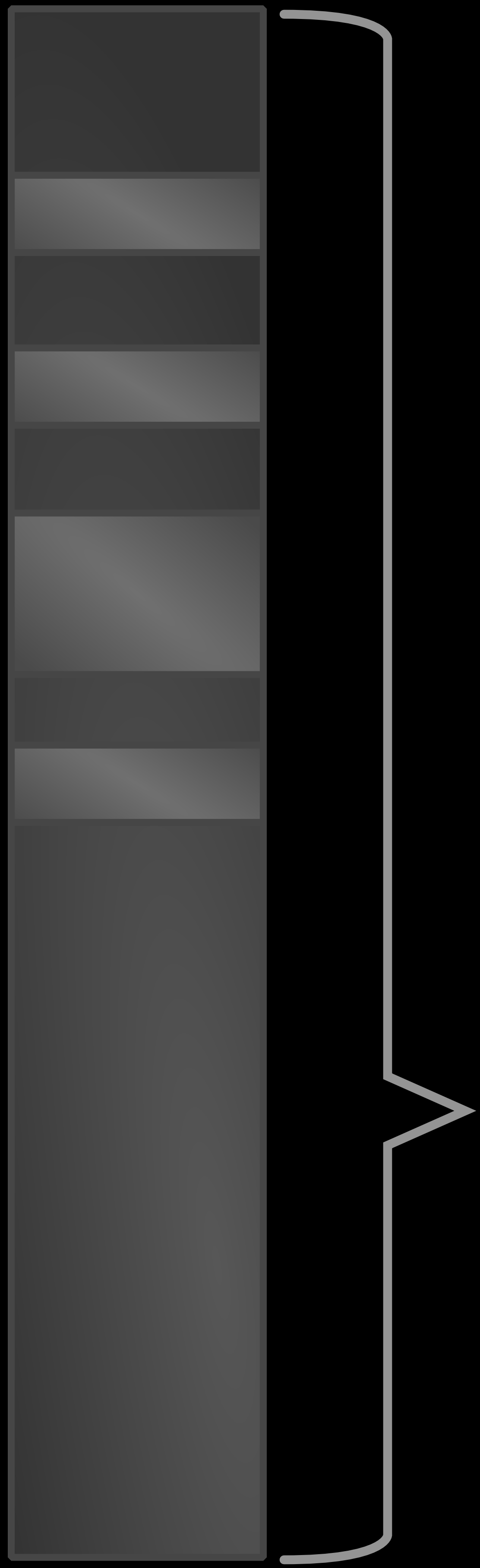
MMIO/PIO Example





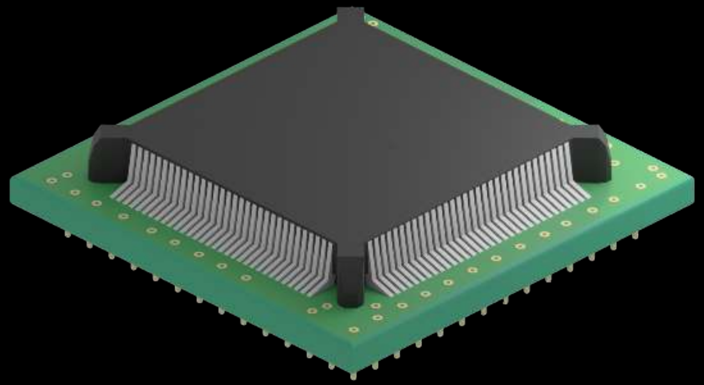
MMIO/PIO Example

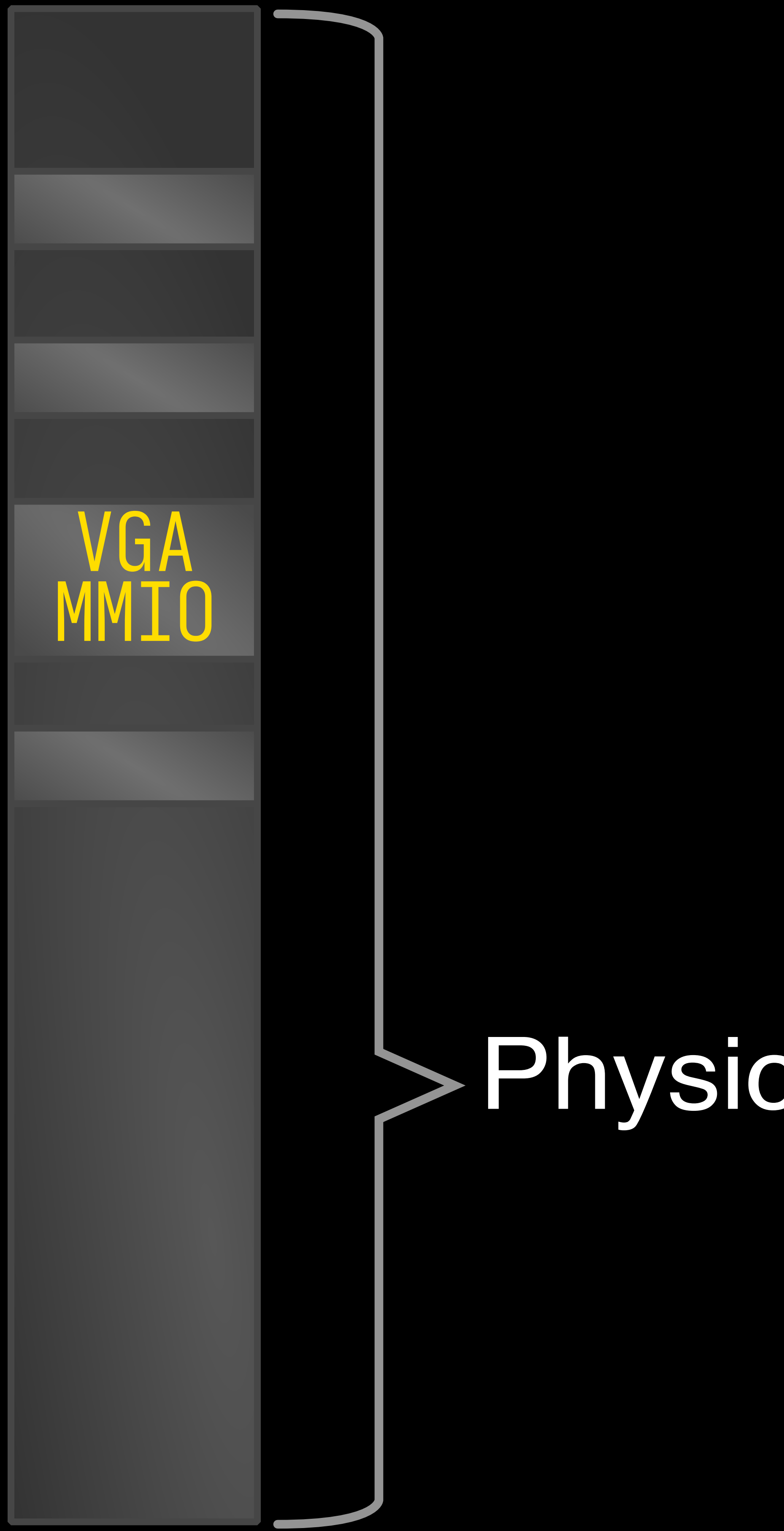




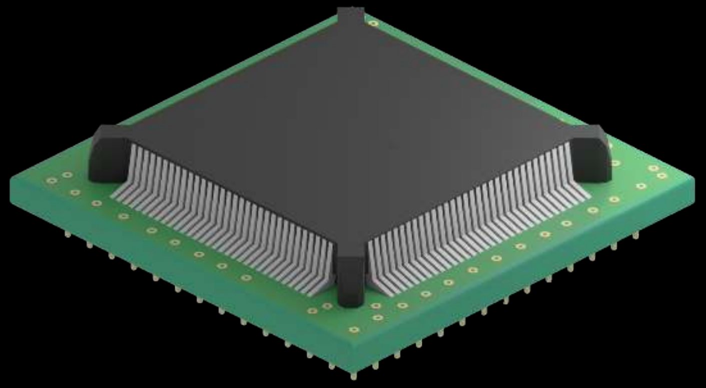
Physical Memory

MMIO/PIO Example



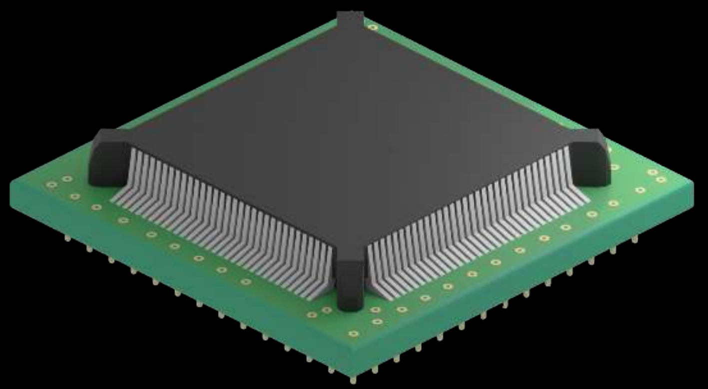


MMIO/PIO Example

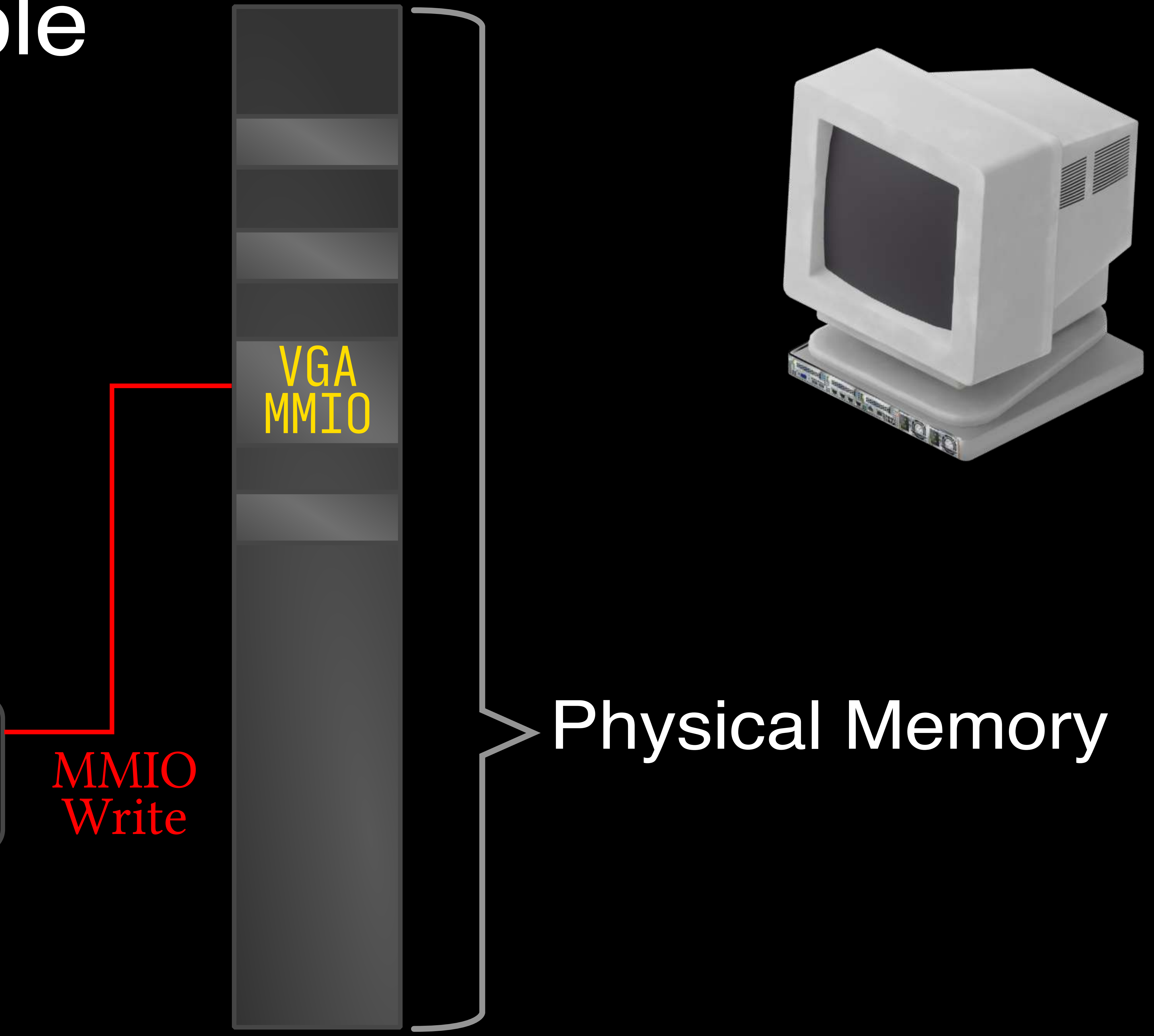


```
char* vga_mmio = 0xB8000;
vga_mmio[0] = 'H';
```

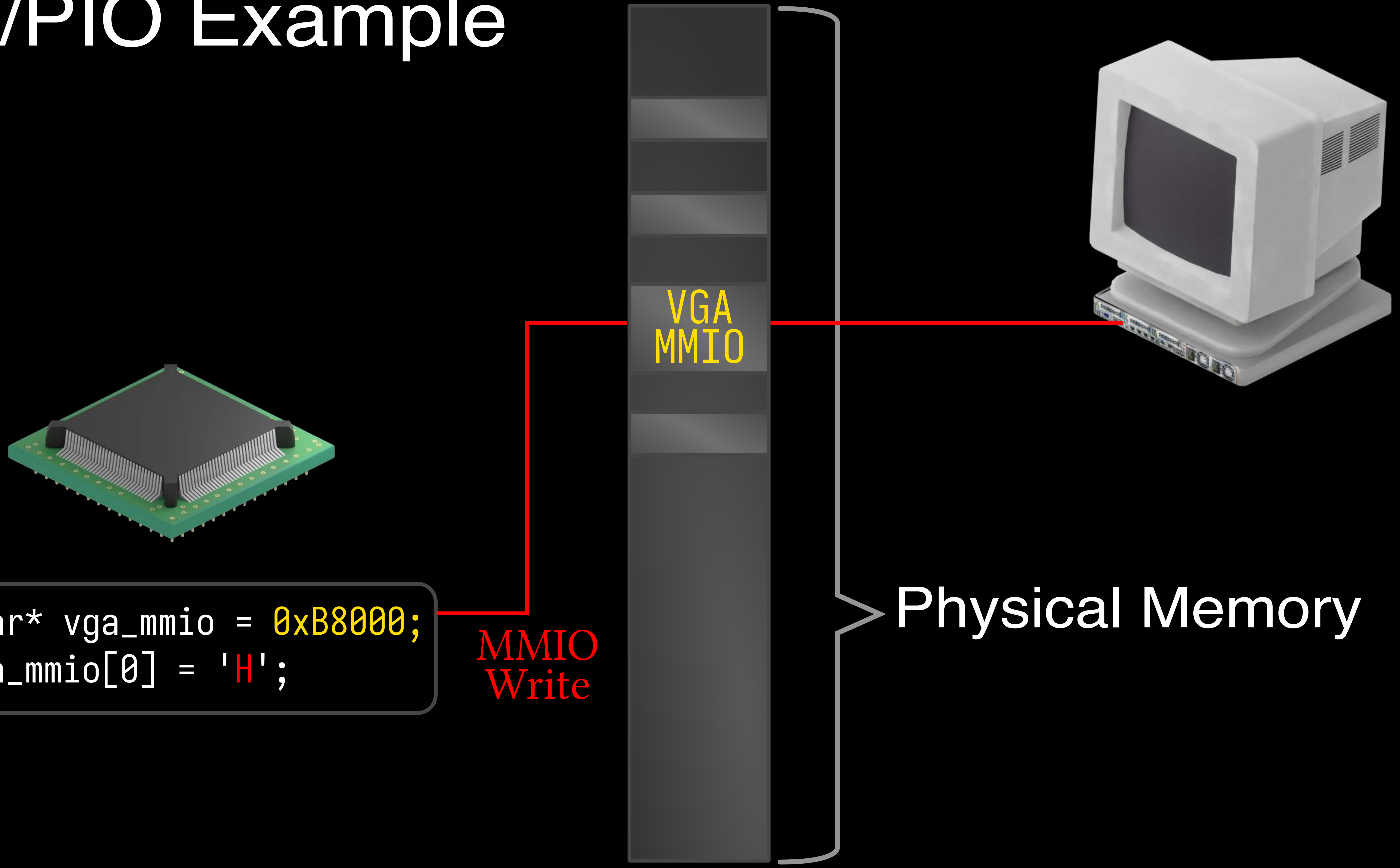
MMIO/PIO Example



```
char* vga_mmio = 0xB8000;
vga_mmio[0] = 'H';
```

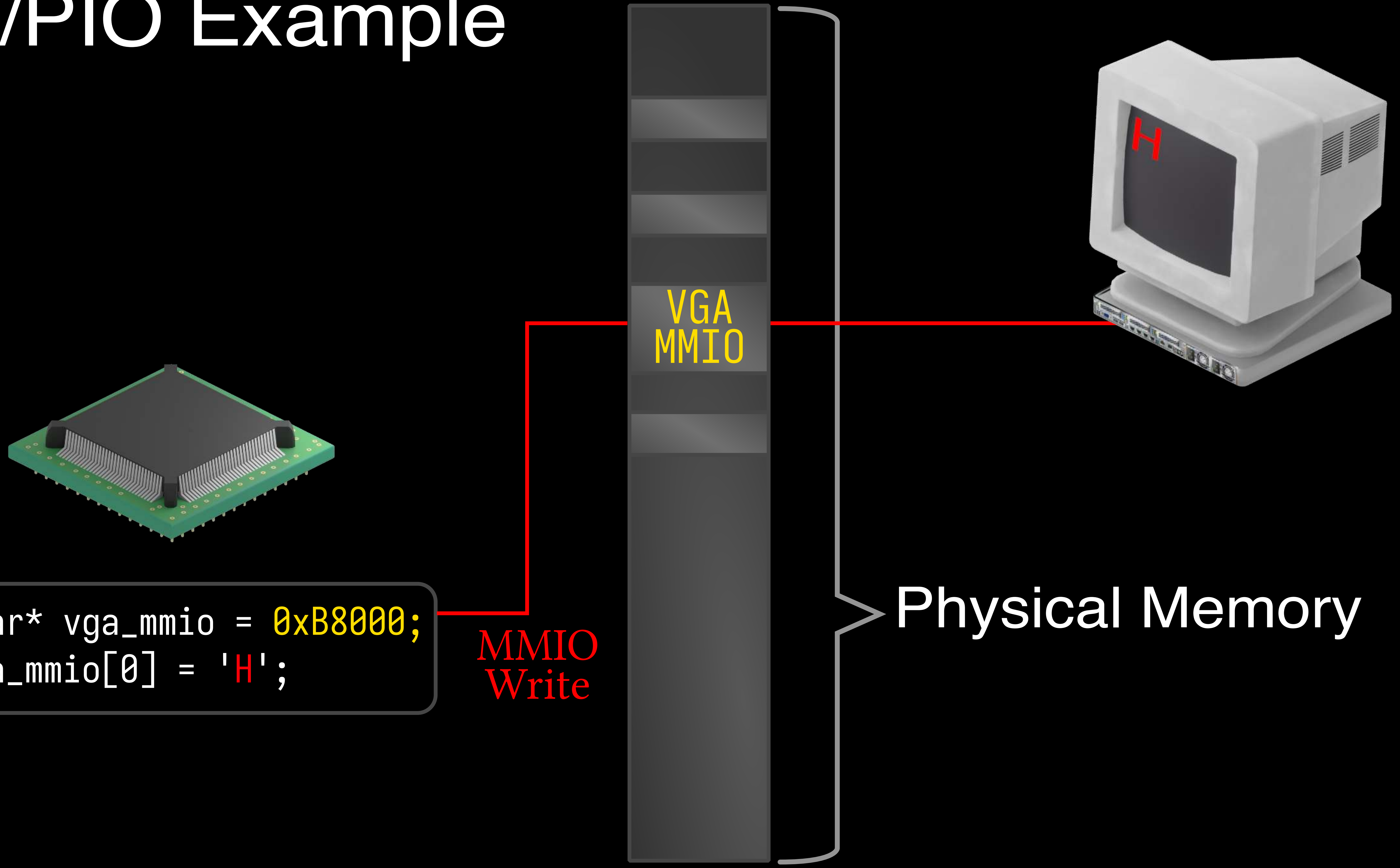


MMIO/PIO Example



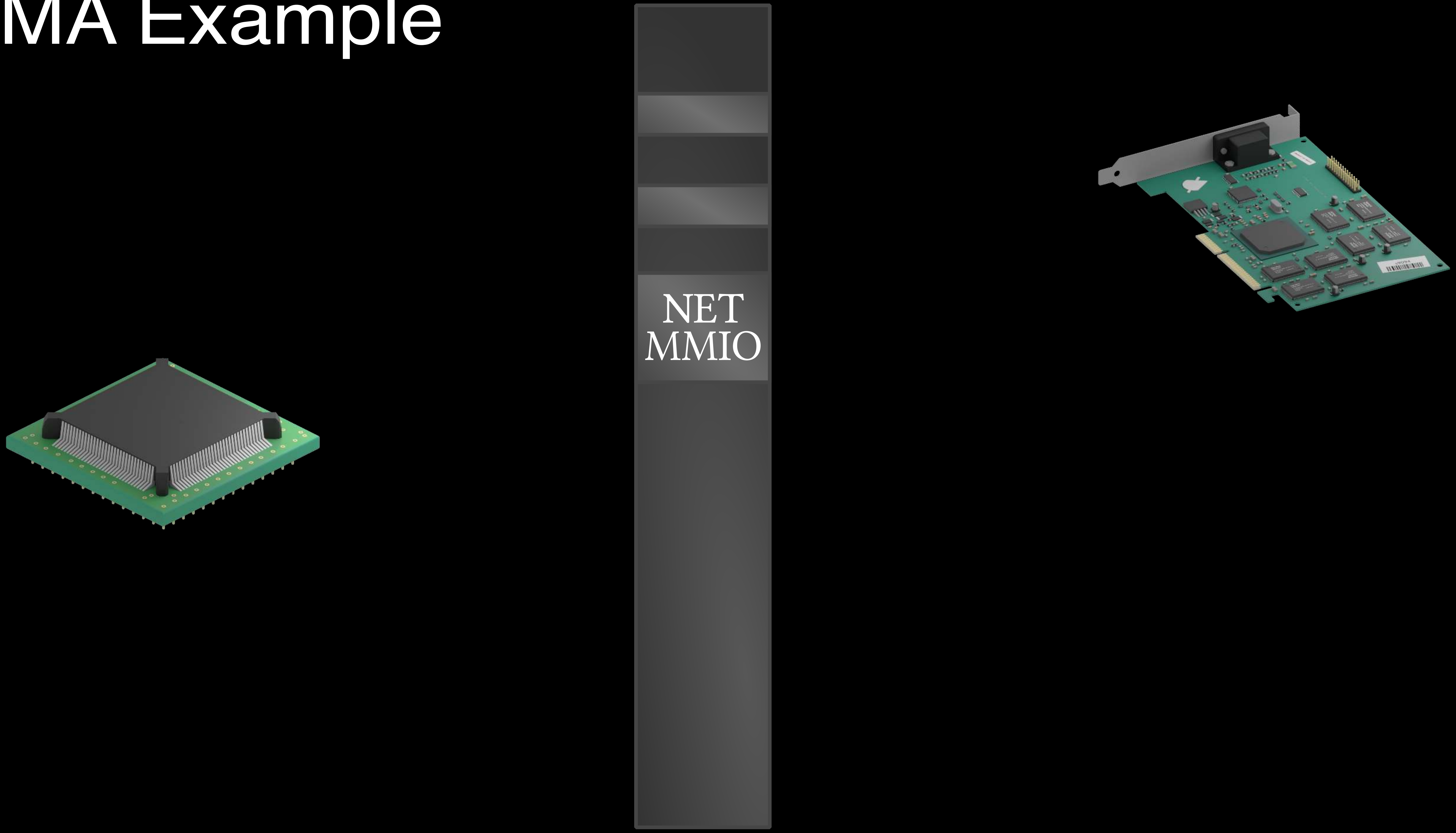

```
char* vga_mmio = 0xB8000;
vga_mmio[0] = 'H';
```

MMIO/PIO Example

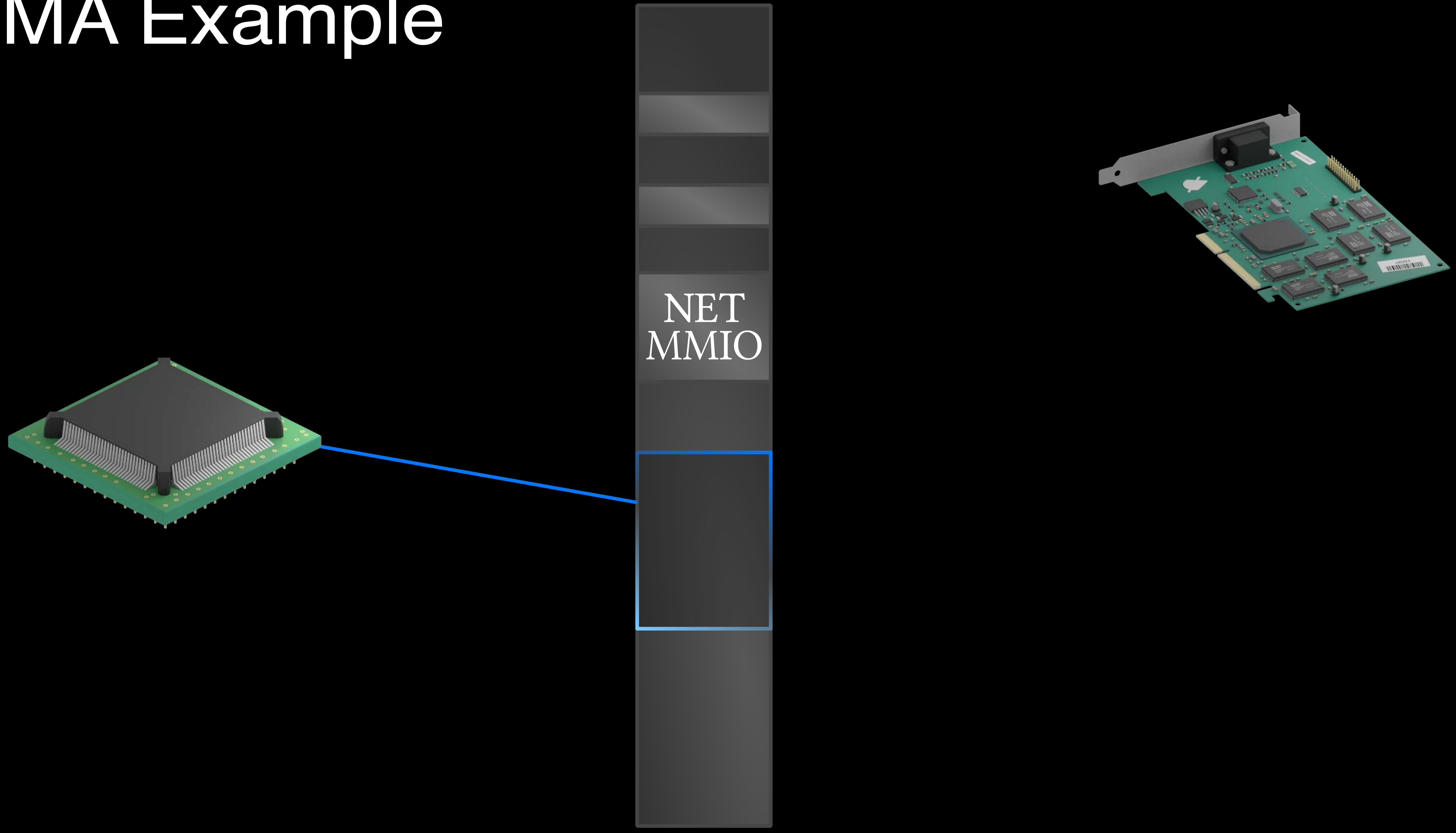



```
char* vga_mmio = 0xB8000;
vga_mmio[0] = 'H';
```

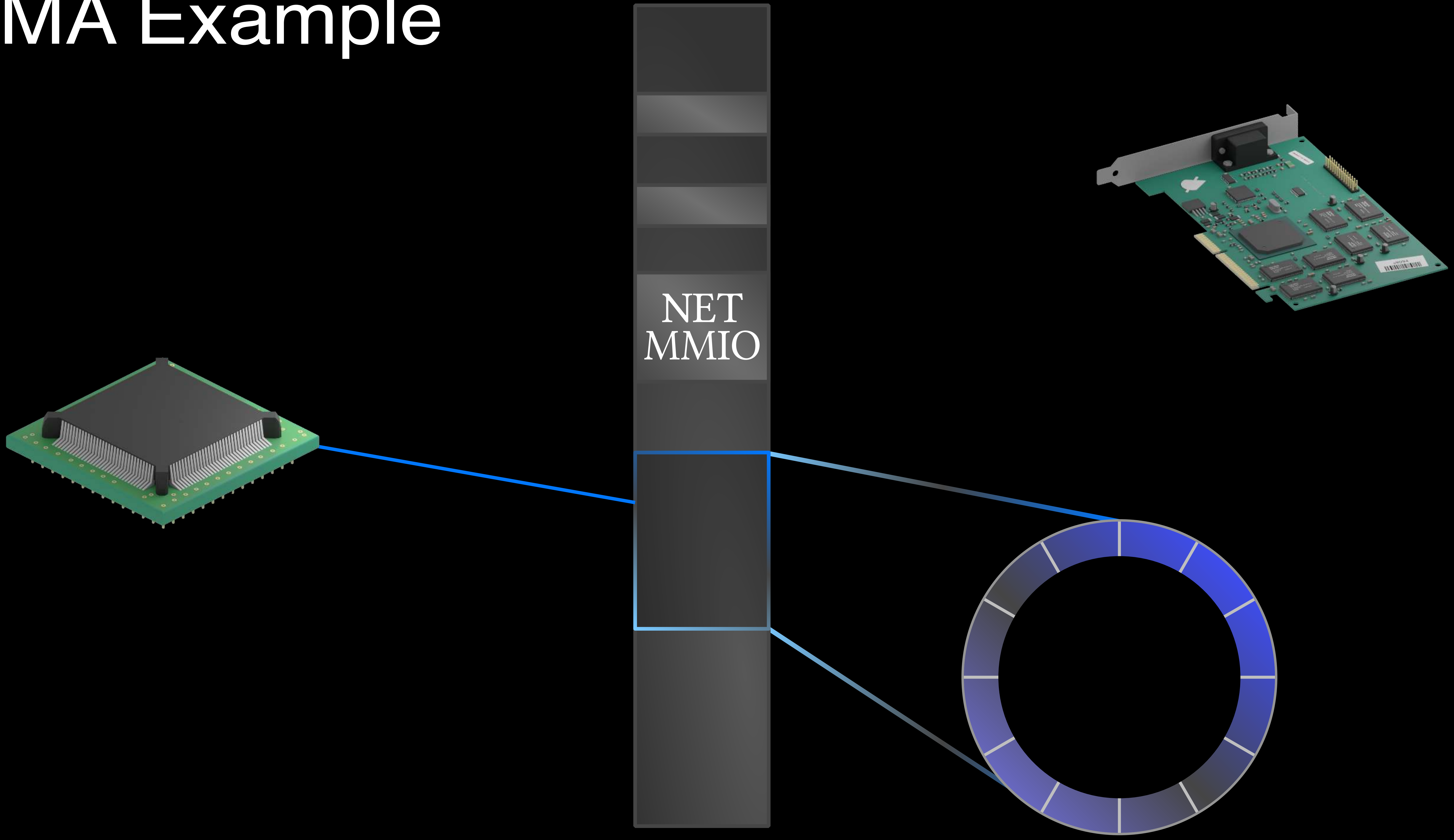
DMA Example



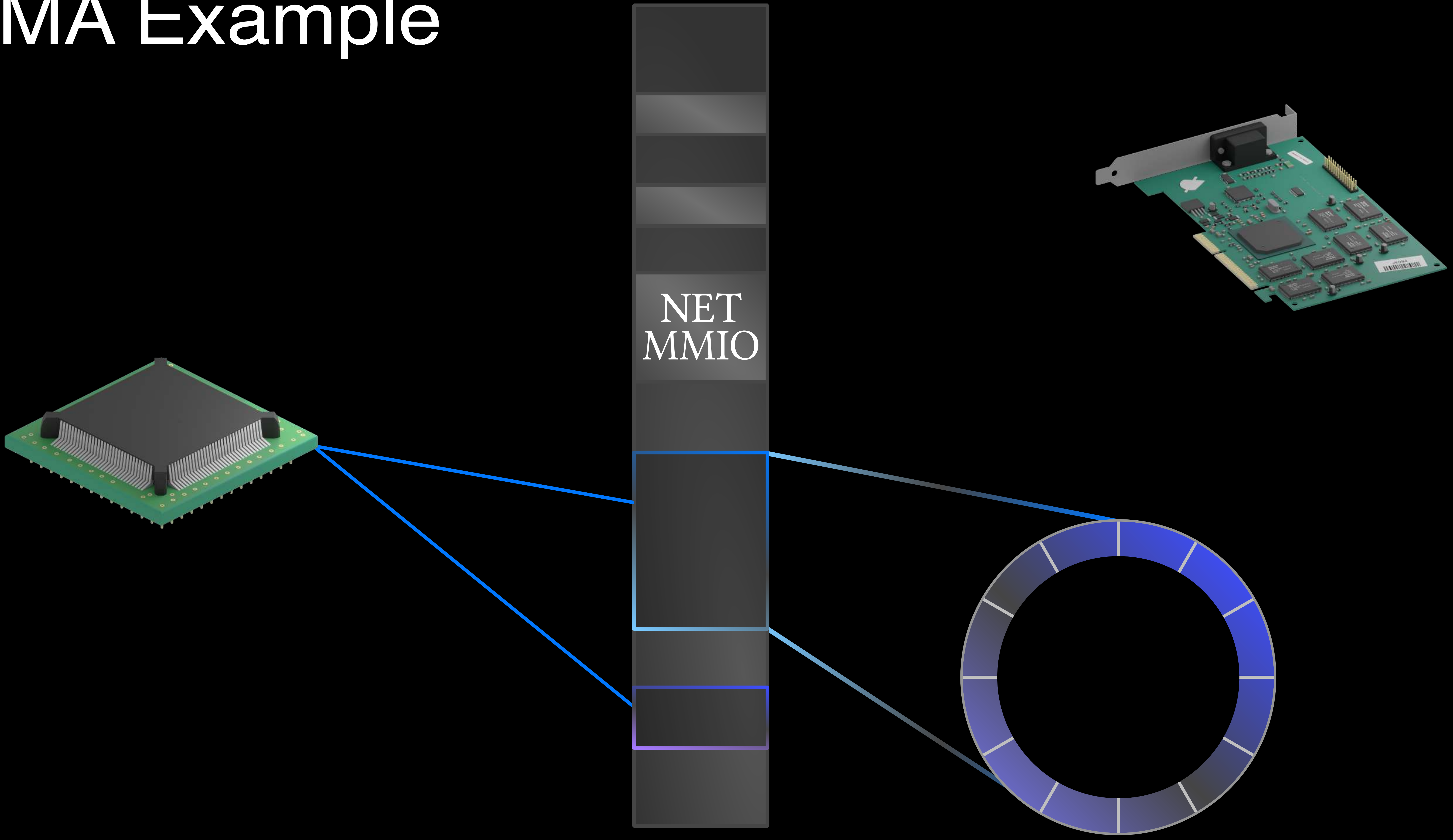
DMA Example



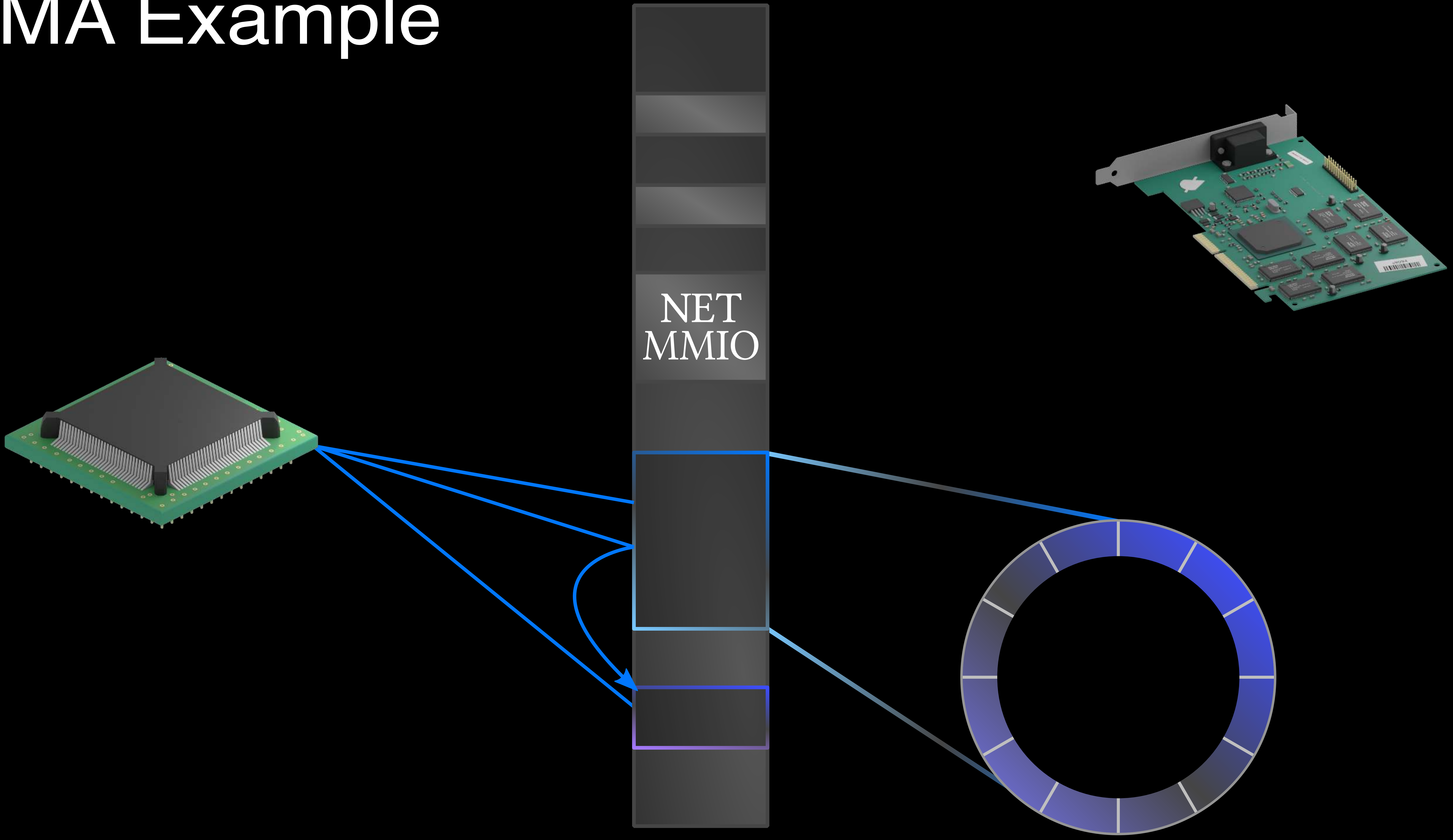
DMA Example



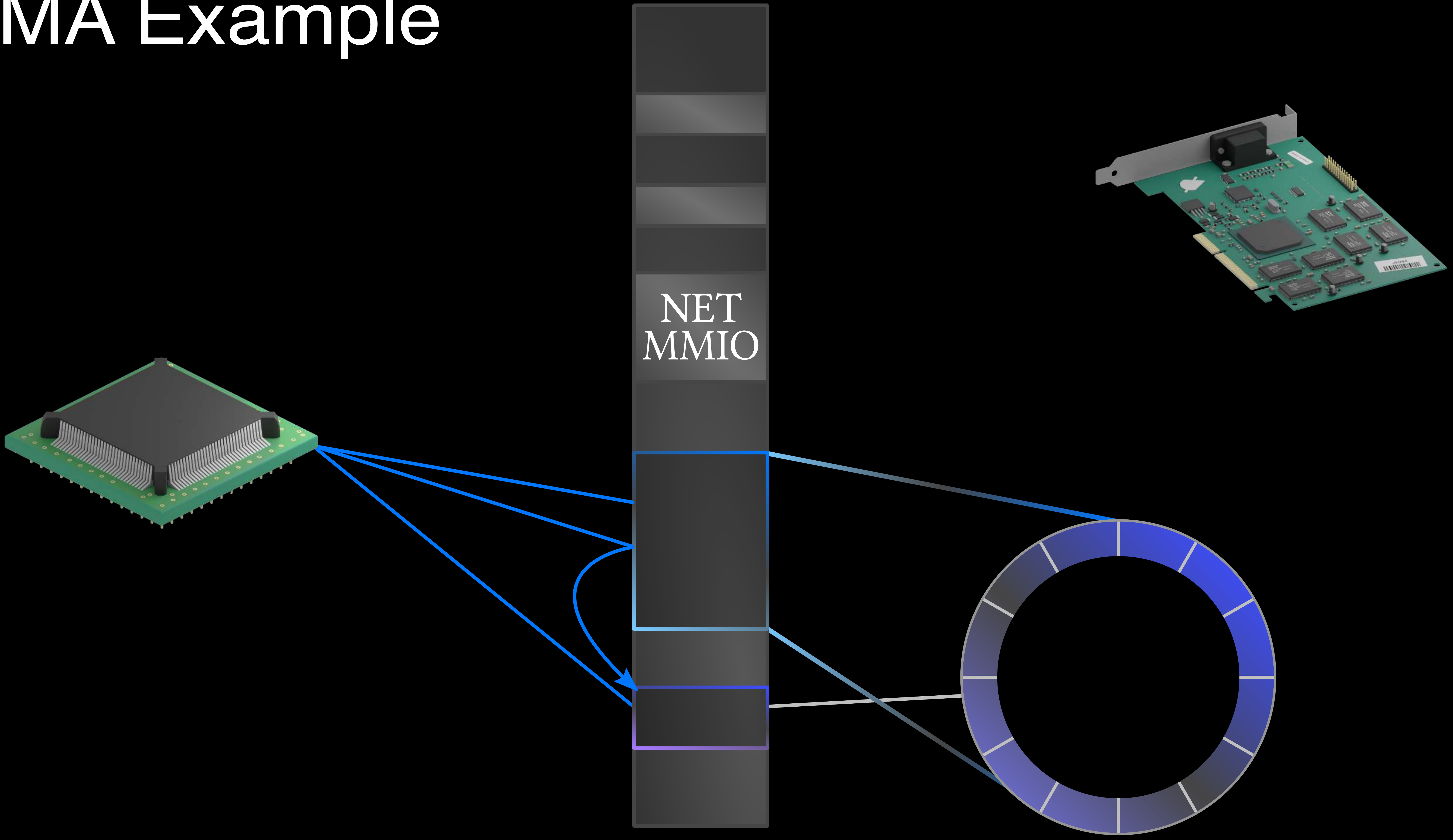
DMA Example



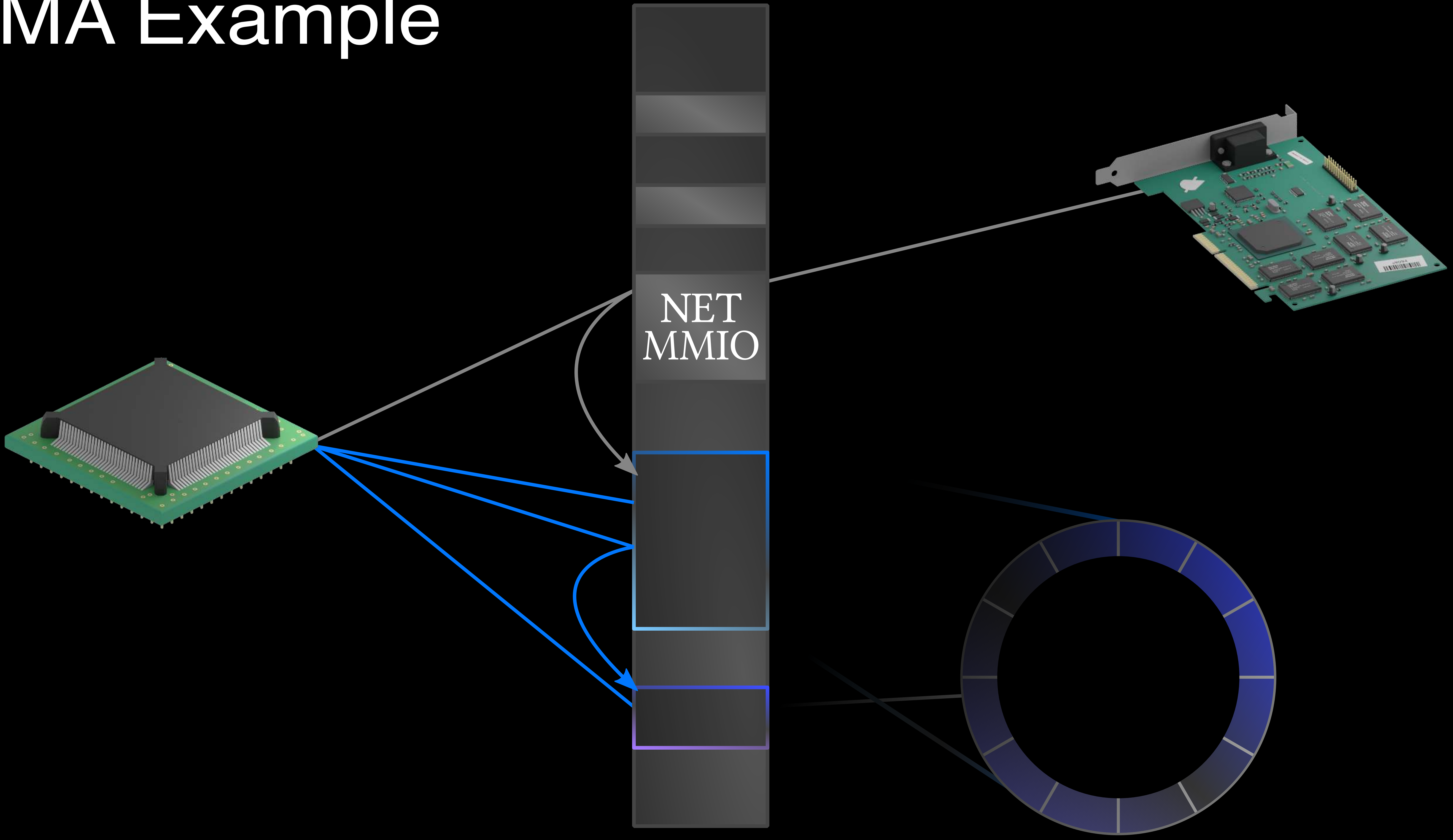
DMA Example



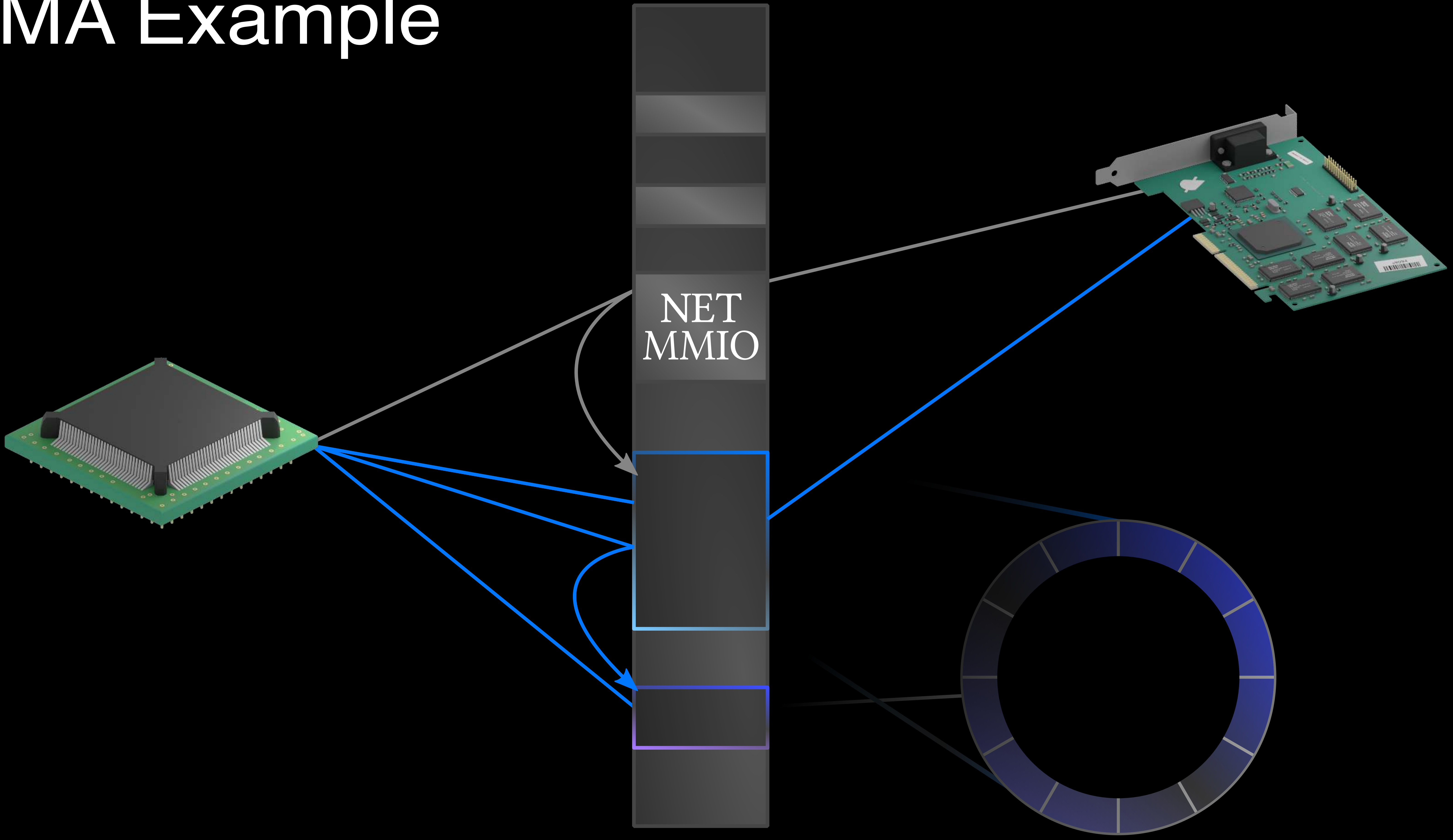
DMA Example



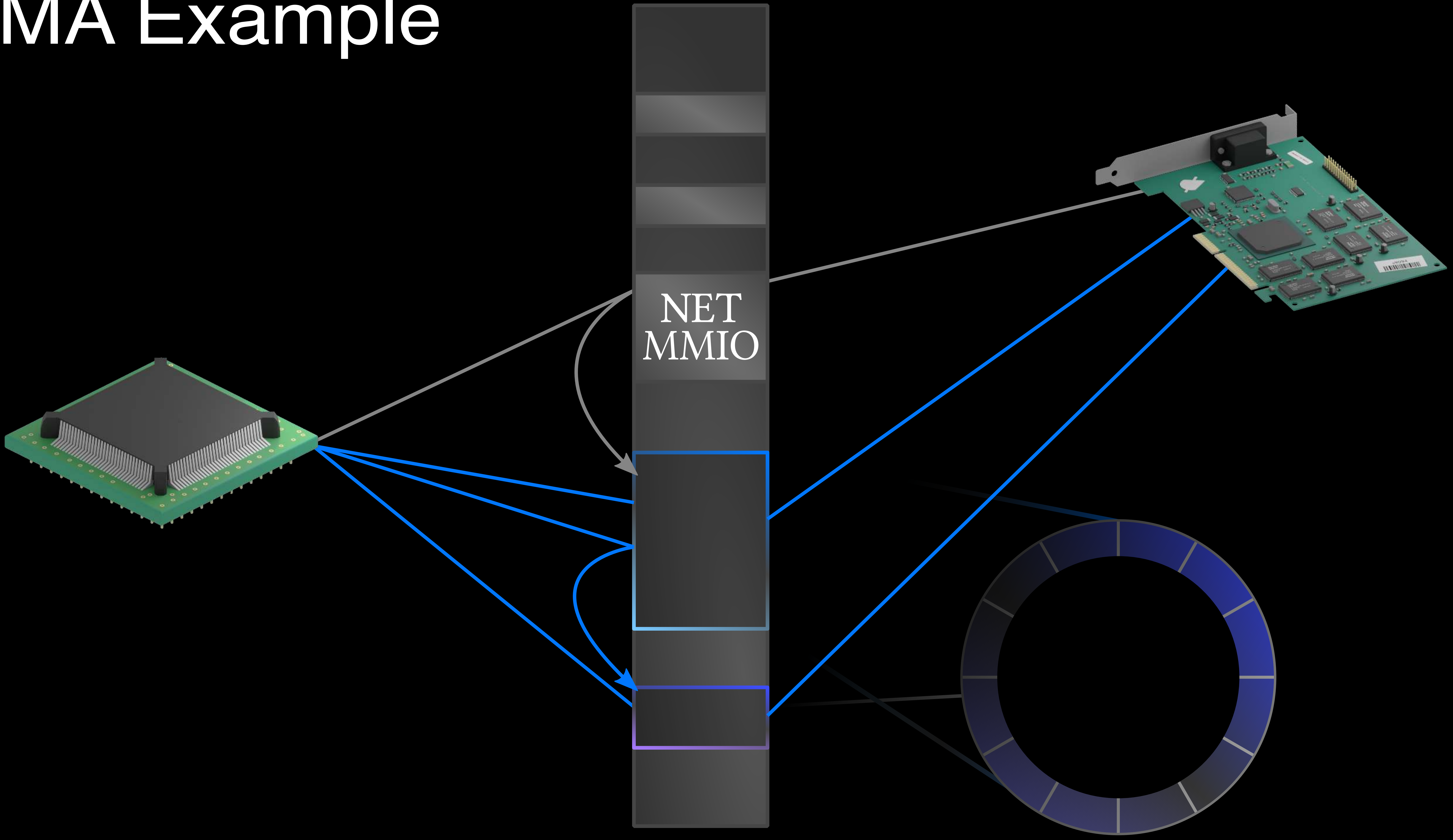
DMA Example



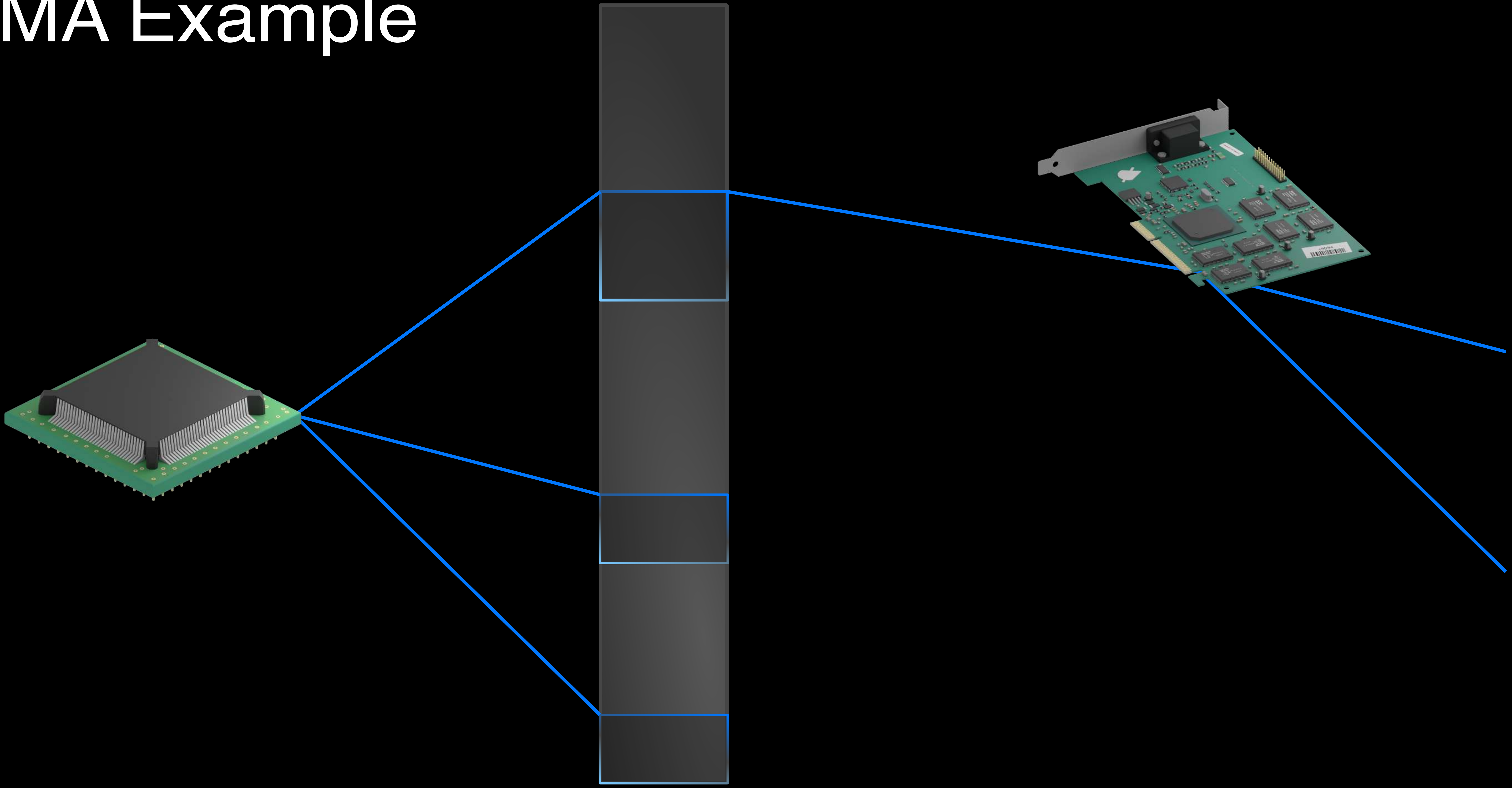
DMA Example



DMA Example



DMA Example



Input Semantics

```
outl 0xcf8 0x80001010
outl 0xcfc 0xe1020000
outl 0xcf8 0x80001014
outl 0xcf8 0x80001004
outw 0xcfc 0x7
outl 0xcf8 0x800010a2
write 0xe102003b 0xff
write 0xe1020118 0xffffffff
write 0xe1020420 0xffffffff
write 0xe1020424 0xffffffff
write 0xe102042b 0xff
write 0xe1020429 0x5 0x0015c5e5c0
write 0x5c041 0x0402e1
write 0x5c048 0x8a
write 0x5c04a 0x31
write 0x5c04b 0xff
write 0xe1020403 0xff
```

Input Semantics

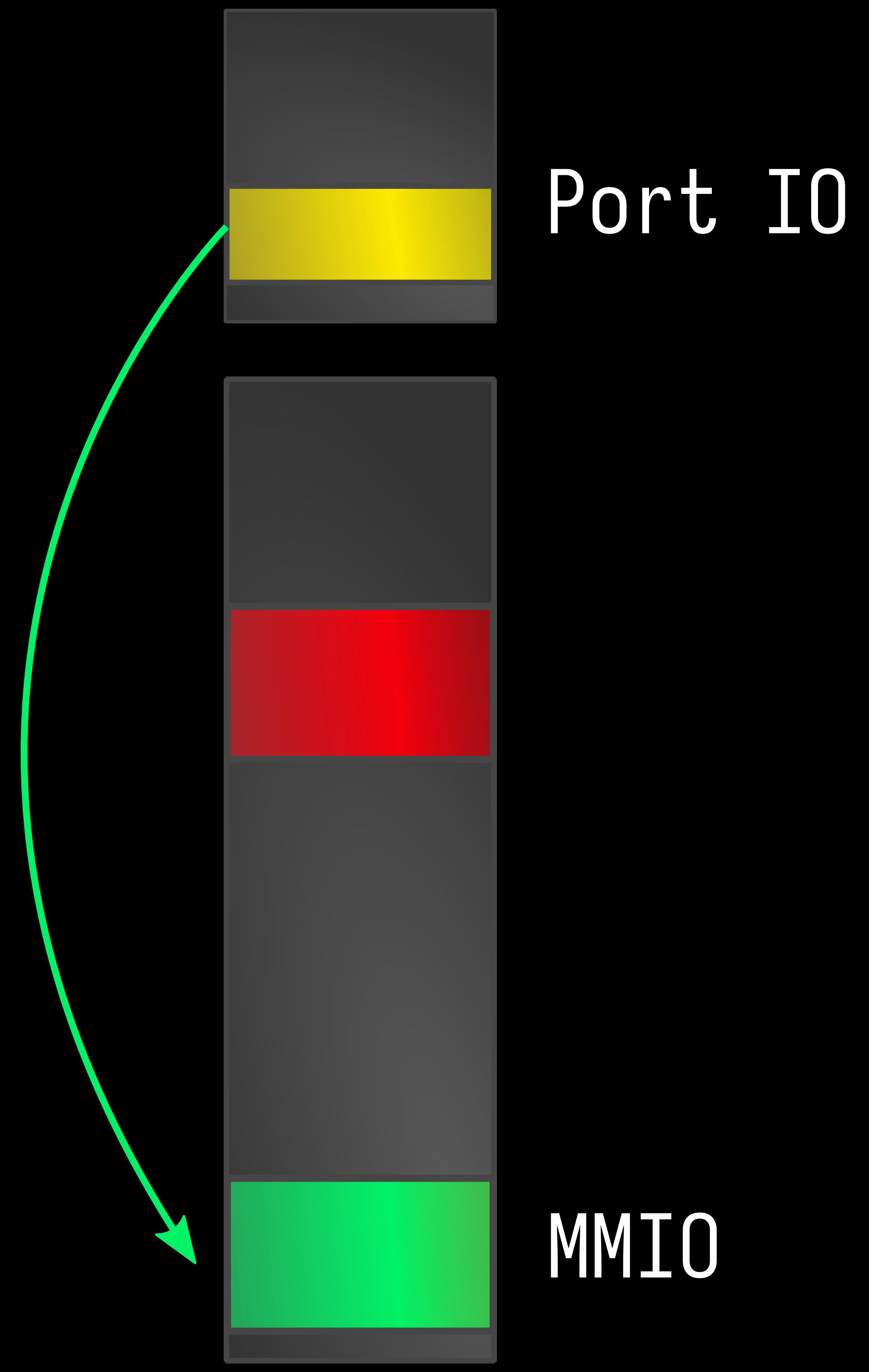
```
outl 0xcf8 0x80001010
outl 0xcfc 0xe1020000
outl 0xcf8 0x80001014
outl 0xcf8 0x80001004
outw 0xcfc 0x7
outl 0xcf8 0x800010a2
write 0xe102003b 0xff
write 0xe1020118 0xffffffff
write 0xe1020420 0xffffffff
write 0xe1020424 0xffffffff
write 0xe102042b 0xff
write 0xe1020429 0x5 0x0015c5e5c0
write 0x5c041 0x0402e1
write 0x5c048 0x8a
write 0x5c04a 0x31
write 0x5c04b 0xff
write 0xe1020403 0xff
```

Input Semantics

```
outl 0xcf8 0x80001010
outl 0xcfc 0xe1020000
outl 0xcf8 0x80001014
outl 0xcf8 0x80001004
outw 0xcfc 0x7
outl 0xcf8 0x800010a2
write 0xe102003b 0xff
write 0xe1020118 0xffffffff
write 0xe1020420 0xffffffff
write 0xe1020424 0xffffffff
write 0xe102042b 0xff
write 0xe1020429 0x5 0x0015c5e5c0
write 0x5c041 0x0402e1
write 0x5c048 0x8a
write 0x5c04a 0x31
write 0x5c04b 0xff
write 0xe1020403 0xff
```

Specifies Addr of

Creates



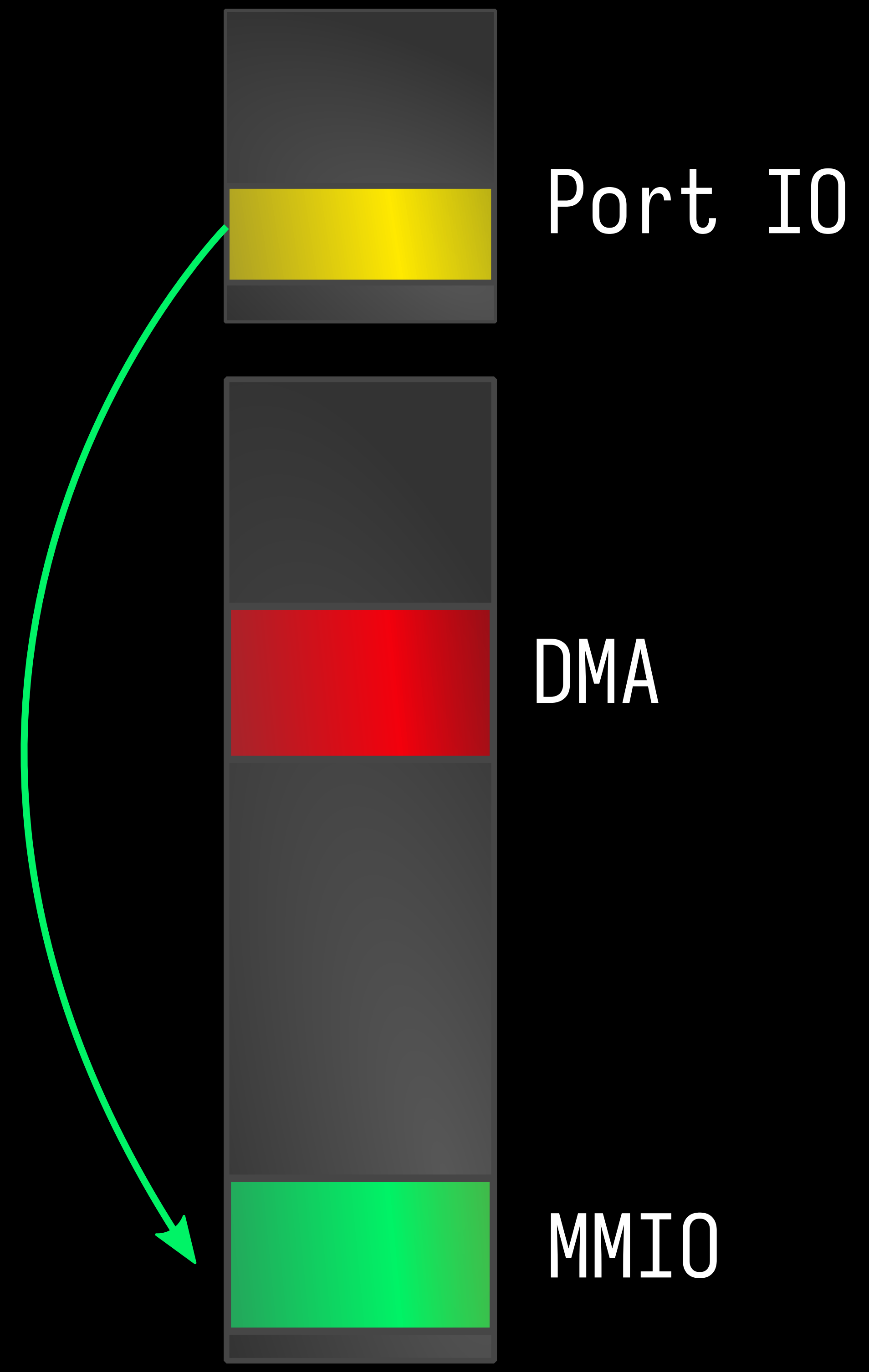
Input Semantics

```
outl 0xcf8 0x80001010
outl 0xcfc 0xe1020000
outl 0xcf8 0x80001014
outl 0xcf8 0x80001004
outw 0xcfc 0x7
outl 0xcf8 0x800010a2
write 0xe102003b 0xff
write 0xe1020118 0xffffffff
write 0xe1020420 0xffffffff
write 0xe1020424 0xffffffff
write 0xe102042b 0xff
write 0xe1020429 0x5 0x0015c5e5c0
write 0x5c041 0x0402e1
write 0x5c048 0x8a
write 0x5c04a 0x31
write 0x5c04b 0xff
write 0xe1020403 0xff
```

Specifies Addr of

Creates

Specifies Addr of



Input Semantics

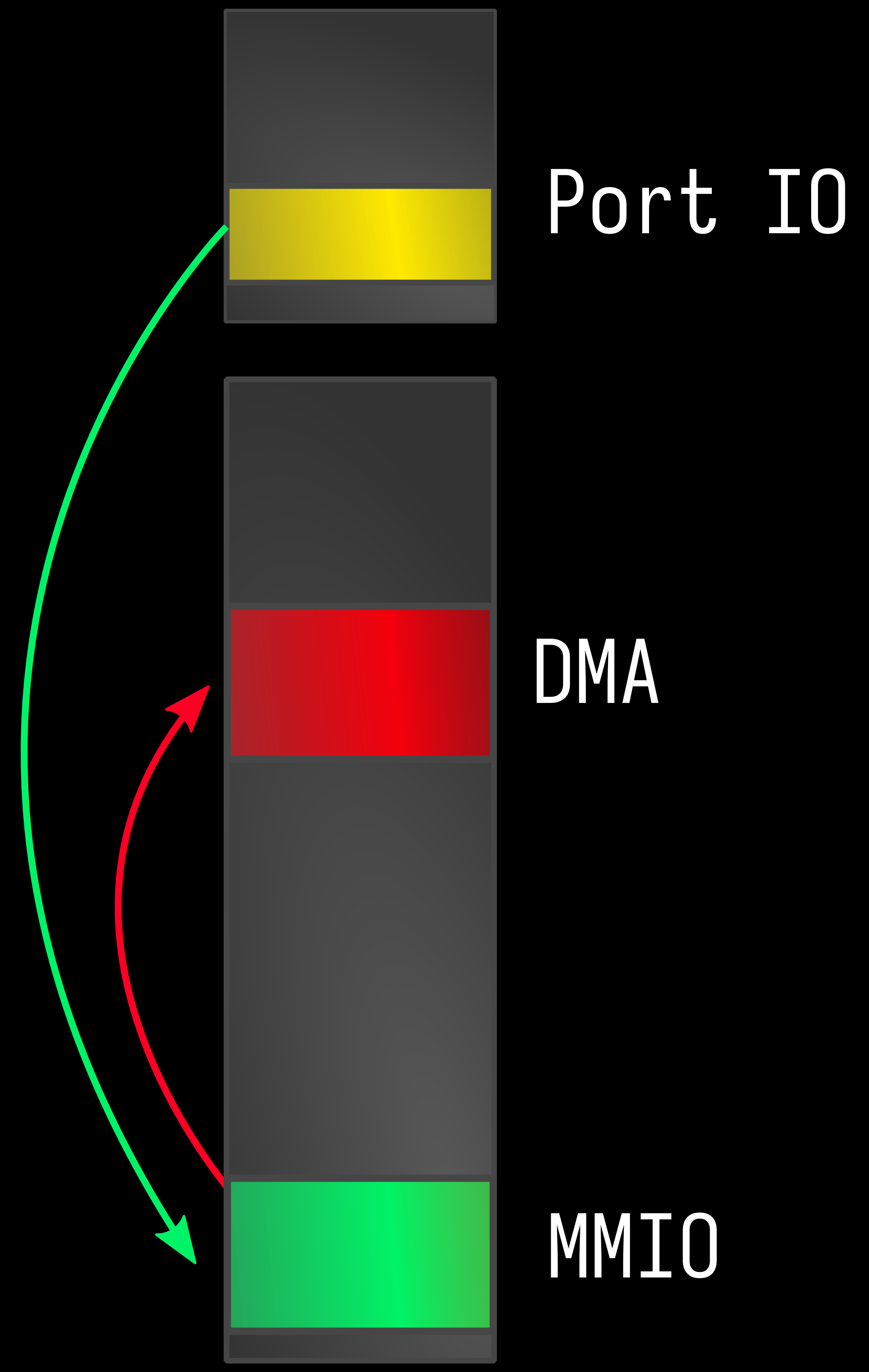
```
outl 0xcf8 0x80001010
outl 0xcfc 0xe1020000
outl 0xcf8 0x80001014
outl 0xcf8 0x80001004
outw 0xcfc 0x7
outl 0xcf8 0x800010a2
write 0xe102003b 0xff
write 0xe1020118 0xffffffff
write 0xe1020420 0xffffffff
write 0xe1020424 0xffffffff
write 0xe102042b 0xff
write 0xe1020429 0x5 0x0015c5e5c0
write 0x5c041 0x0402e1
write 0x5c048 0x8a
write 0x5c04a 0x31
write 0x5c04b 0xff
write 0xe1020403 0xff
```

Specifies Addr of

Creates

Specifies Addr of

Causes Access to



Input Semantics

```
outl 0xcf8 0x80001010
outl 0xcfc 0xe1020000
outl 0xcf8 0x80001014
outl 0xcf8 0x80001004
outw 0xcfc 0x7
outl 0xcf8 0x800010a2
write 0xe102003b 0xff
write 0xe1020118 0xffffffff
write 0xe1020420 0xffffffff
write 0xe1020424 0xffffffff
write 0xe102042b 0xff
write 0xe1020429 0x5 0x0015c5e5c0
write 0x5c041 0x0402e1
write 0x5c048 0x8a
write 0x5c04a 0x31
write 0x5c04b 0xff
write 0xe1020403 0xff
```

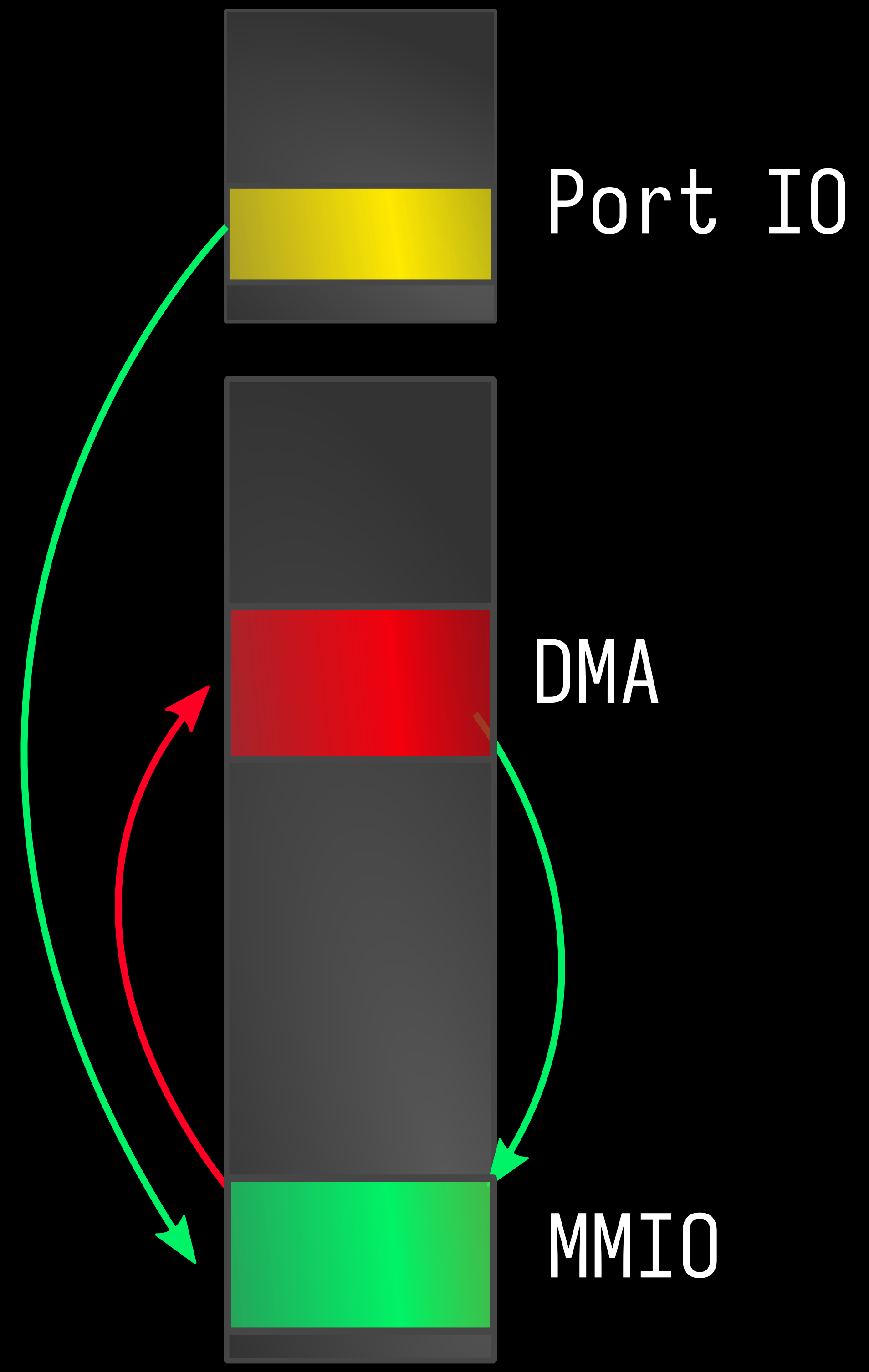
Specifies Addr of

Creates

Specifies Addr of

References

Causes Access to



Input Semantics

```
outl 0xcf8 0x80001010
outl 0xcfc 0xe1020000
outl 0xcf8 0x80001014
outl 0xcf8 0x80001004
outw 0xcfc 0x7
outl 0xcf8 0x800010a2
write 0xe102003b 0xff
write 0xe1020118 0xffffffff
write 0xe1020420 0xffffffff
write 0xe1020424 0xffffffff
write 0xe102042b 0xff
write 0xe1020429 0x5 0x0015c5e5c0
write 0x5c041 0x0402e1
write 0x5c048 0x8a
write 0x5c04a 0x31
write 0x5c04b 0xff
write 0xe1020403 0xff
```

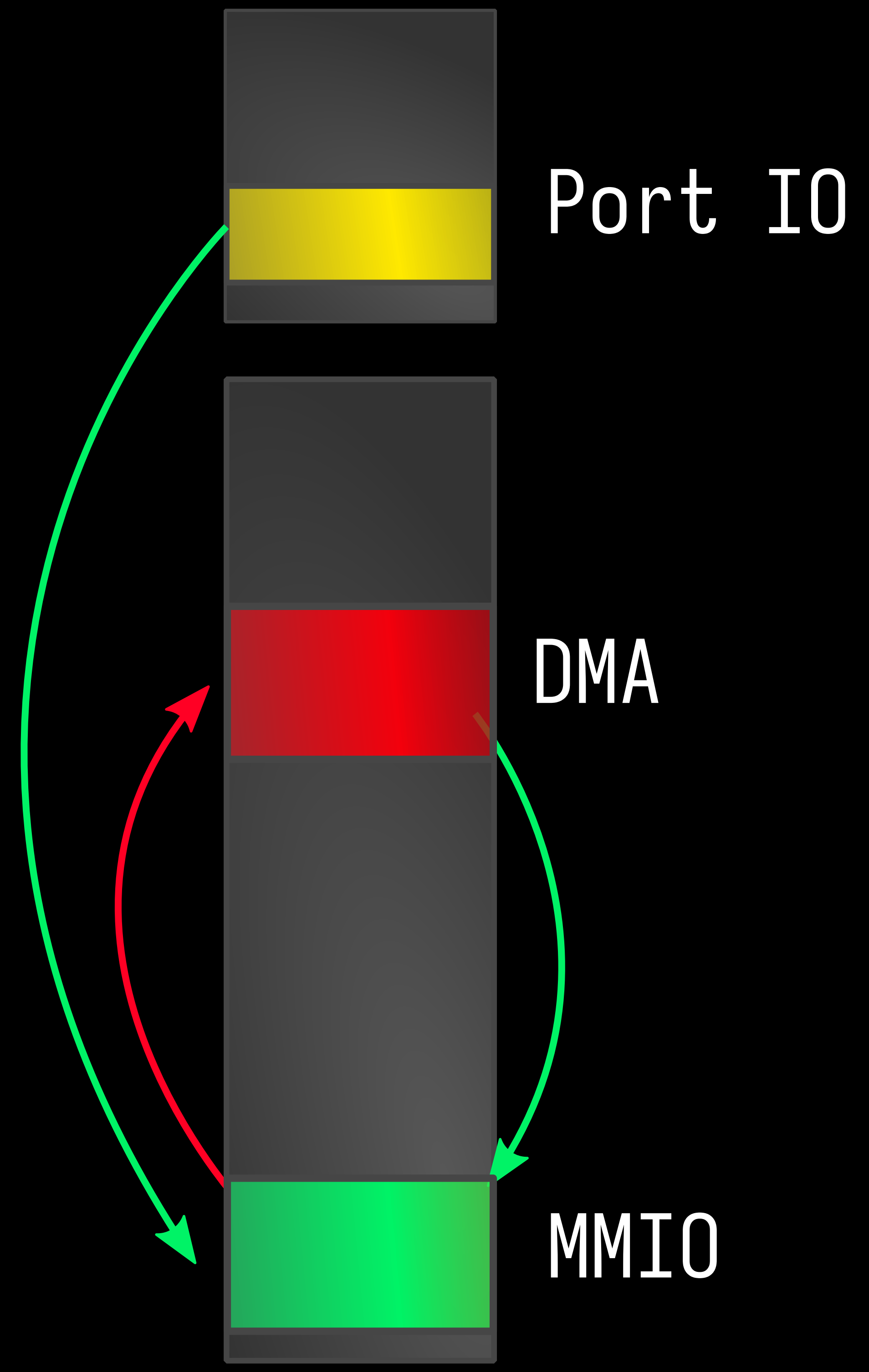
Specifies Addr of

Creates

Specifies Addr of

References

Causes Access to



Inputs can interact with virtually any address


MORPHUZZ: Bending (Input) Space to Fuzz Virtual Devices

Alexander Bulekov*† Bandan Das† Stefan Hajnoczi† Manuel Egele*
*Boston University †Red Hat
alxndr@bu.edu, bsd@redhat.com, stefanha@redhat.com, megele@bu.edu

Abstract

The security of the entire cloud ecosystem crucially depends on the isolation guarantees that hypervisors provide between guest VMs and the host system. To allow VMs to communicate with their environment, hypervisors provide a slew of virtual-devices including network interface cards and performance-optimized VIRTIO-based SCSI adapters. As these devices sit directly on the hypervisor's isolation boundary and accept potentially attacker-controlled input (e.g., from a malicious cloud tenant), bugs and vulnerabilities in the devices' implementations have the potential to render the hypervisor's isolation guarantees moot. Prior works applied fuzzing to simple virtual-devices, focusing on a narrow subset of the vast input-space and the state-of-the-art virtual-device fuzzer, Nyx, requires precise, manually-written, specifications to exercise complex devices.

In this paper we present MORPHUZZ, a generic approach that leverages insights about hypervisor design combined with coverage-guided fuzzing to find bugs in virtual device implementations. Crucially MORPHUZZ does not rely on expert knowledge specific to each device. MORPHUZZ is the first approach that automatically elicits the complex I/O behaviors of the real-world virtual devices found in modern clouds. To demonstrate this capability, we implemented MORPHUZZ in QEMU and bhyve and fuzzed 33 different virtual devices (a superset of the 16 devices analyzed by prior work). Additionally, we show that MORPHUZZ is not tied to a specific CPU architecture, by fuzzing 3 additional ARM devices. MORPHUZZ matches or exceeds coverage obtained by Nyx, for 13/16 virtual devices, and identified a superset (110) of all crashes reported by Nyx (44). We reported all newly discovered bugs to the respective developers. Notably, MORPHUZZ achieves this without initial seed-inputs, or expert guidance.

1 Introduction

hile the cloud unveils unique opportunities to IT businesses, presents a host of fundamental security issues. From a nical standpoint, virtualization is the core technol ering cloud-infrastructure. Virtualization

VMMs) multiplex the hardware resources of a physical machine (the host), between multiple Virtual Machines (VMs or guests). Cloud-ready hypervisors are complex pieces of software, tasked with isolating the software running inside a VM (i.e., a guest), from the other guests, and the hypervisor itself. Beyond the cloud, hypervisors are commonly used to sandbox applications (e.g., for malware research), and for desktop use, to run applications not supported by the host OS. Regardless the application, hypervisors are trusted with providing a layer of isolation between virtual machines and the host OS. Crucially, to provide their functionality to guests, hypervisors include a slew of implementations for virtual devices, and the code for these devices commonly executes at the privilege level of the hypervisor itself. Virtual devices play a critical role in ensuring that the guest is isolated, but due to the complexity of these devices, it can be difficult to safely implement their functionality in software. Unfortunately exploits compromising this layer of isolation (and specifically the virtual devices) are a tangible reality. In 2015, VENOM [14] was highly publicized as a VM-Escape vulnerability, which allows an attacker running within an untrusted guest to compromise the underlying hypervisor and execute code outside the security confines of the VM. VENOM is certainly not a unique example, and security researchers have identified many vulnerabilities leading to potential VM-Escape. Ranked by the size of bug bounties, VM-escapes are considered among the most critical classes of vulnerabilities, along with iOS, Android, and browser bugs [58]. Though VM-escape attacks can take advantage of weaknesses in other hypervisor components, such as shadow page tables, our work focuses on virtual-devices which are responsible for the vast majority of reported VM-escape vulnerabilities [37].

Software fuzz testing has proven to be a nique, capable of exposing vulne of software [3, 12, 15, virtual devi

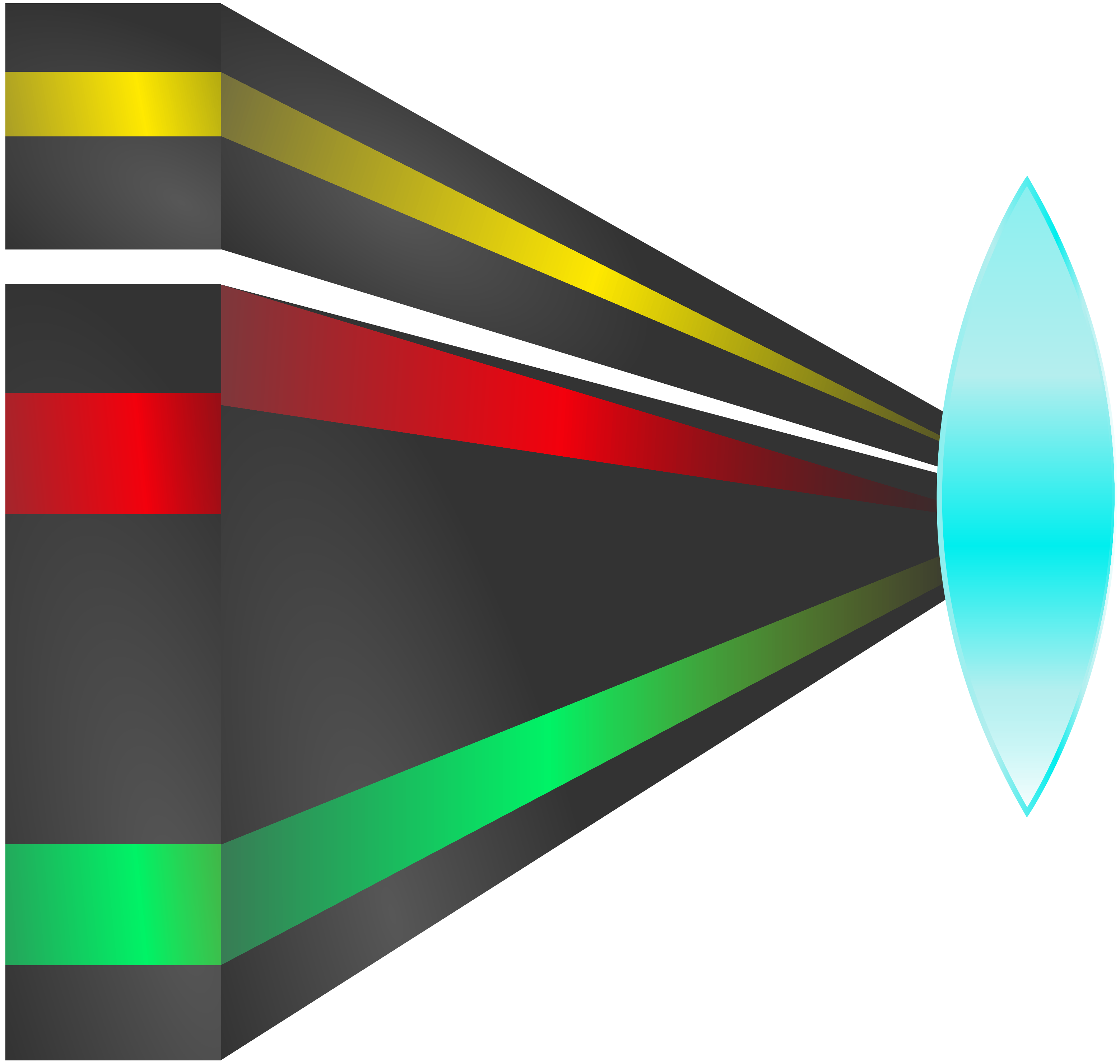
MORPHUZZ

Bending (Input) Space to Fuzz Virtual Devices

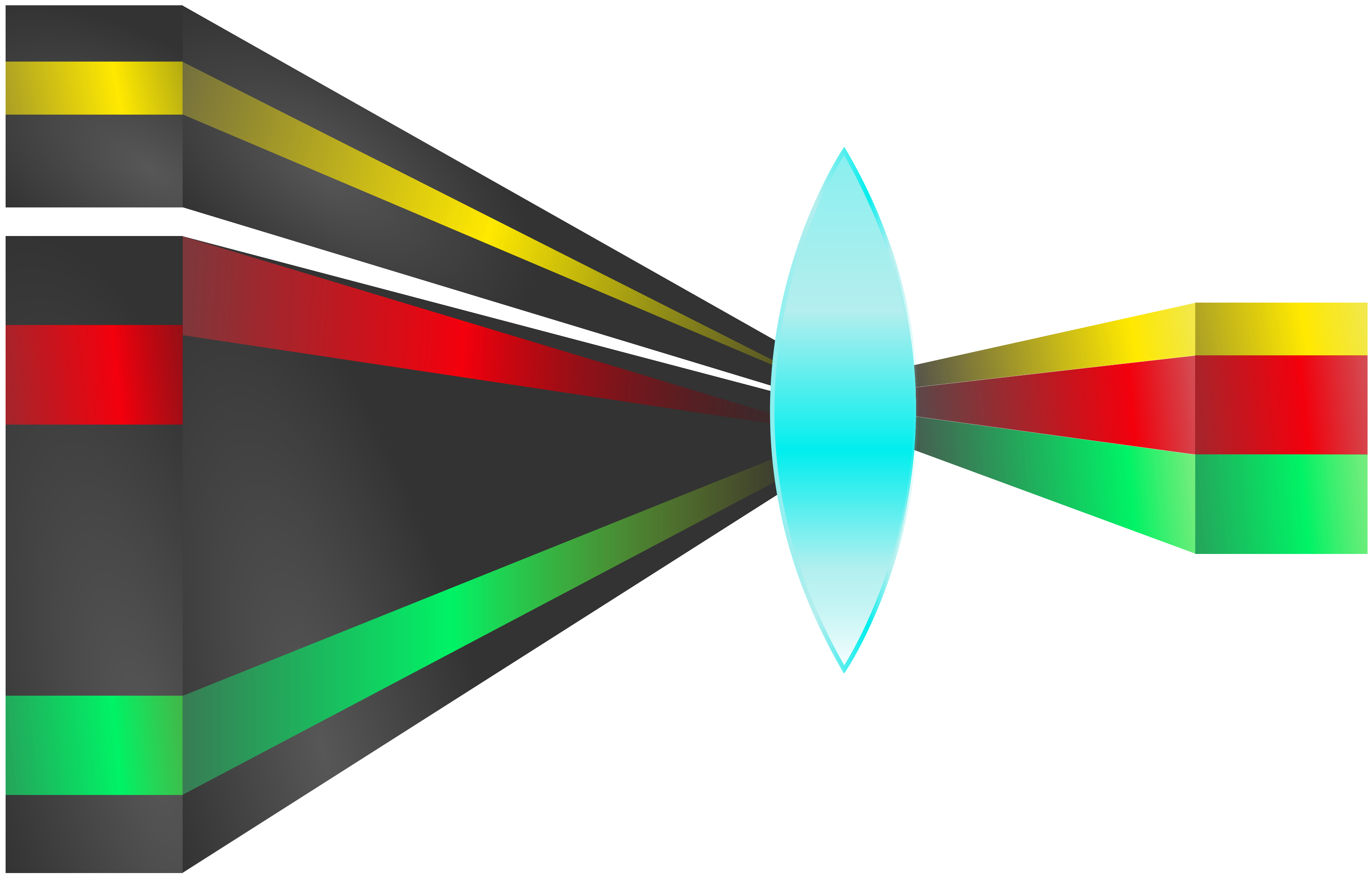
USENIX Security 2022
Bending (Input) Space to Fuzz Virtual Devices

Reshaping the Input Space

Reshaping the Input Space

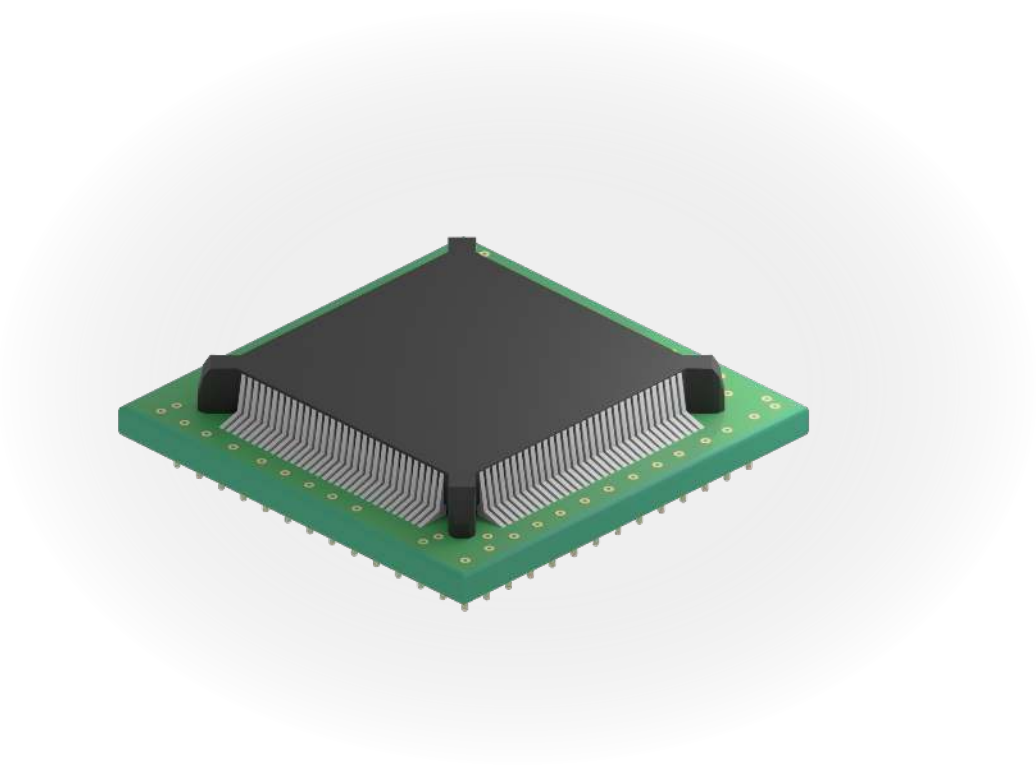


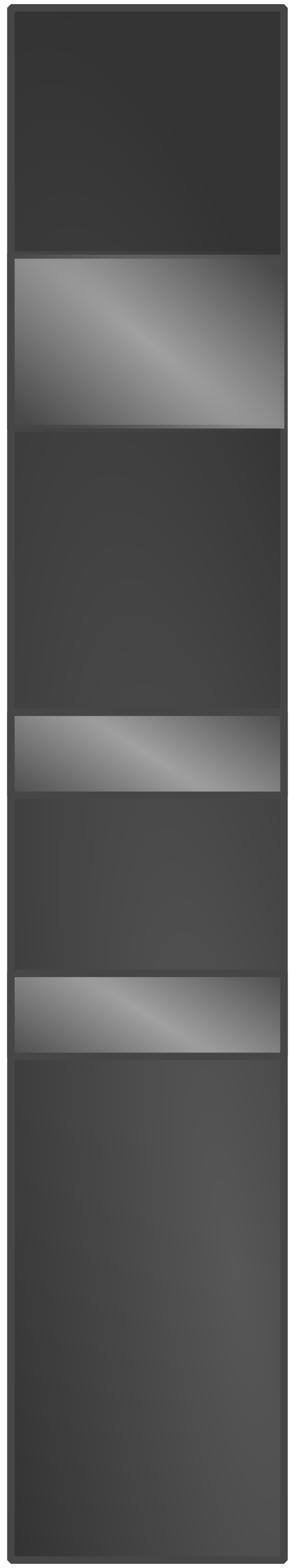
Reshaping the Input Space



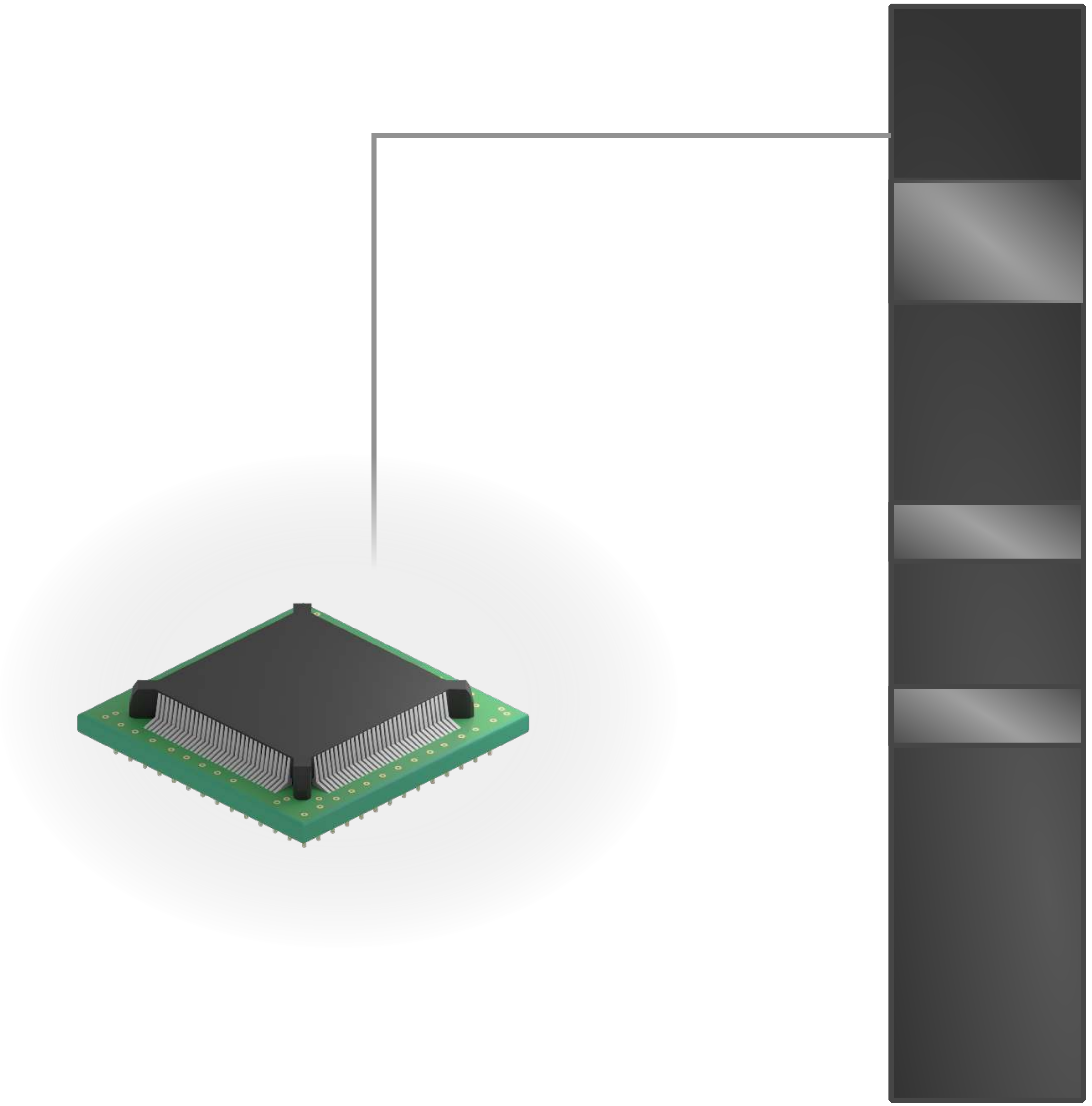
Achieve a precise view of the regions that are actively engaged in IO

Reshaping Port IO and MMIO

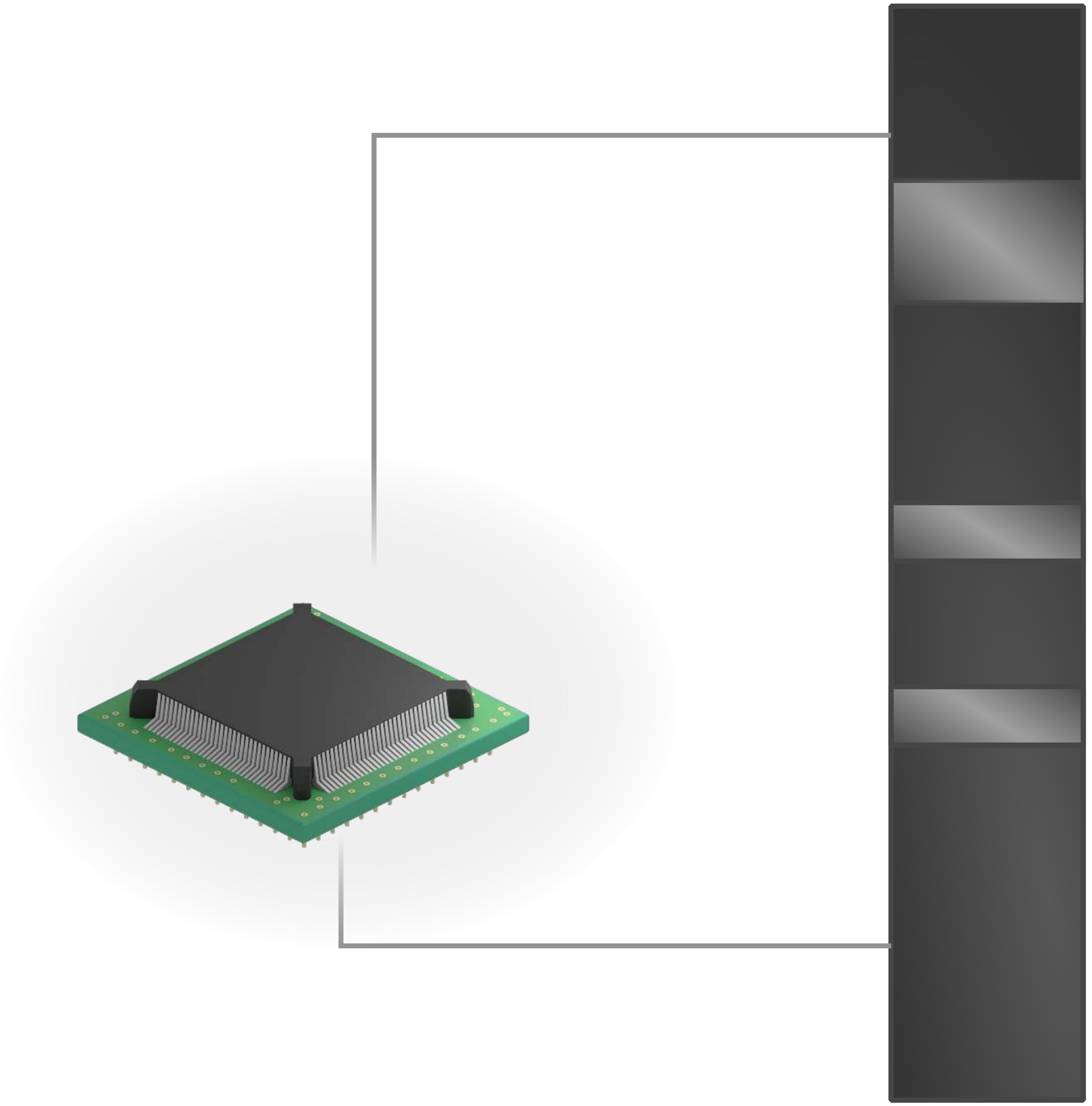




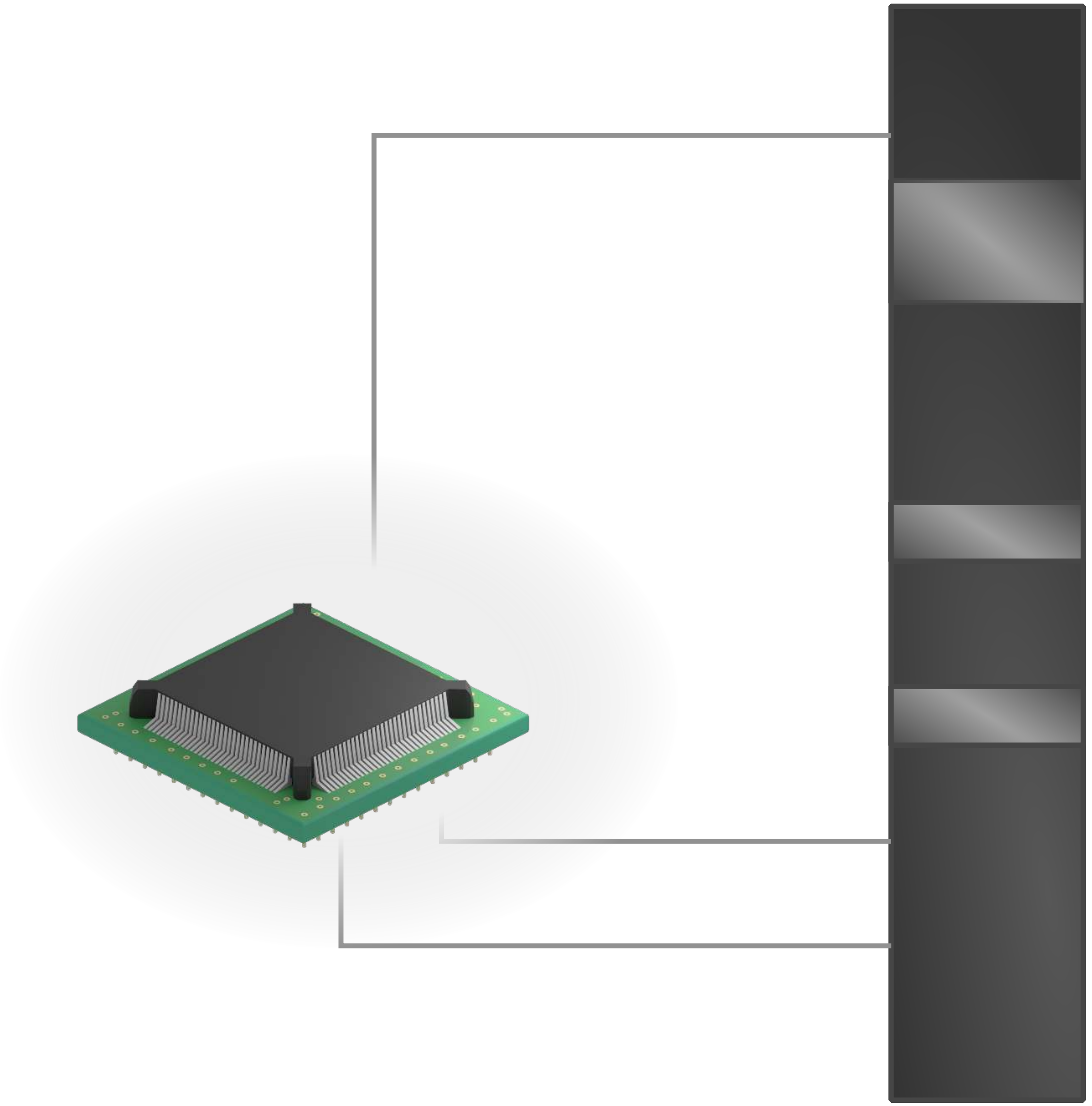
Reshaping Port IO and MMIO



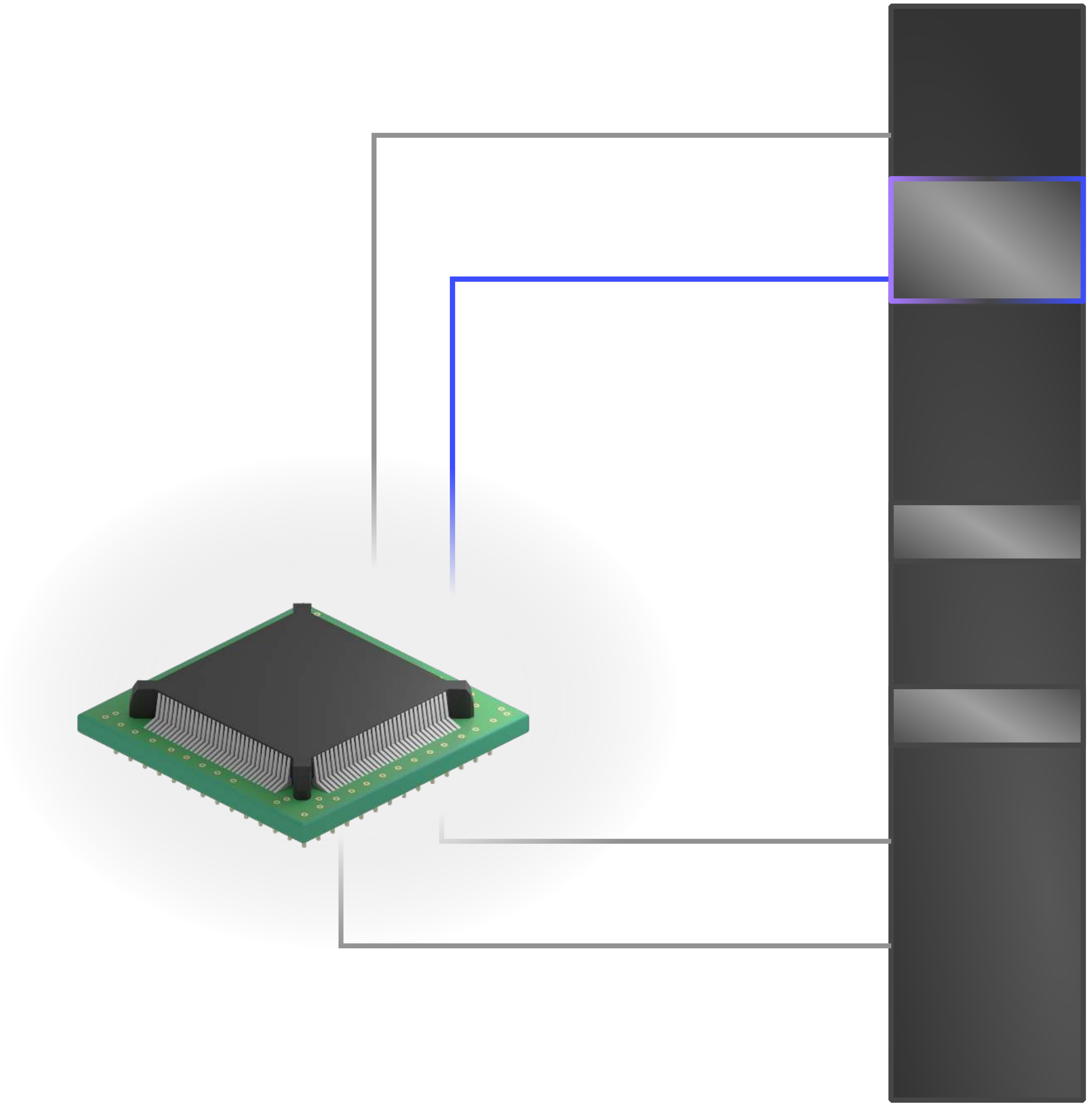
Reshaping Port IO and MMIO



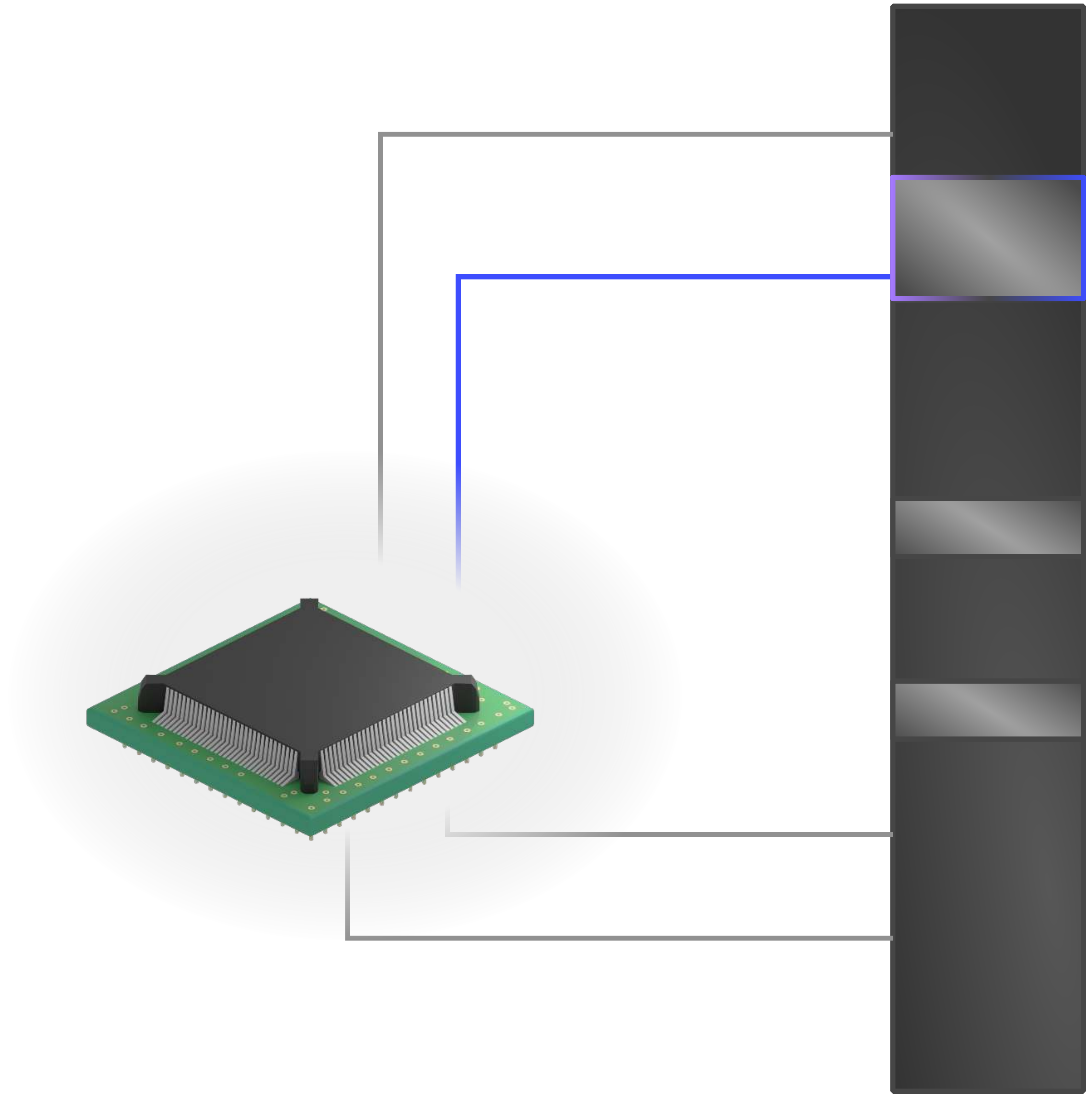
Reshaping Port IO and MMIO



Reshaping Port IO and MMIO

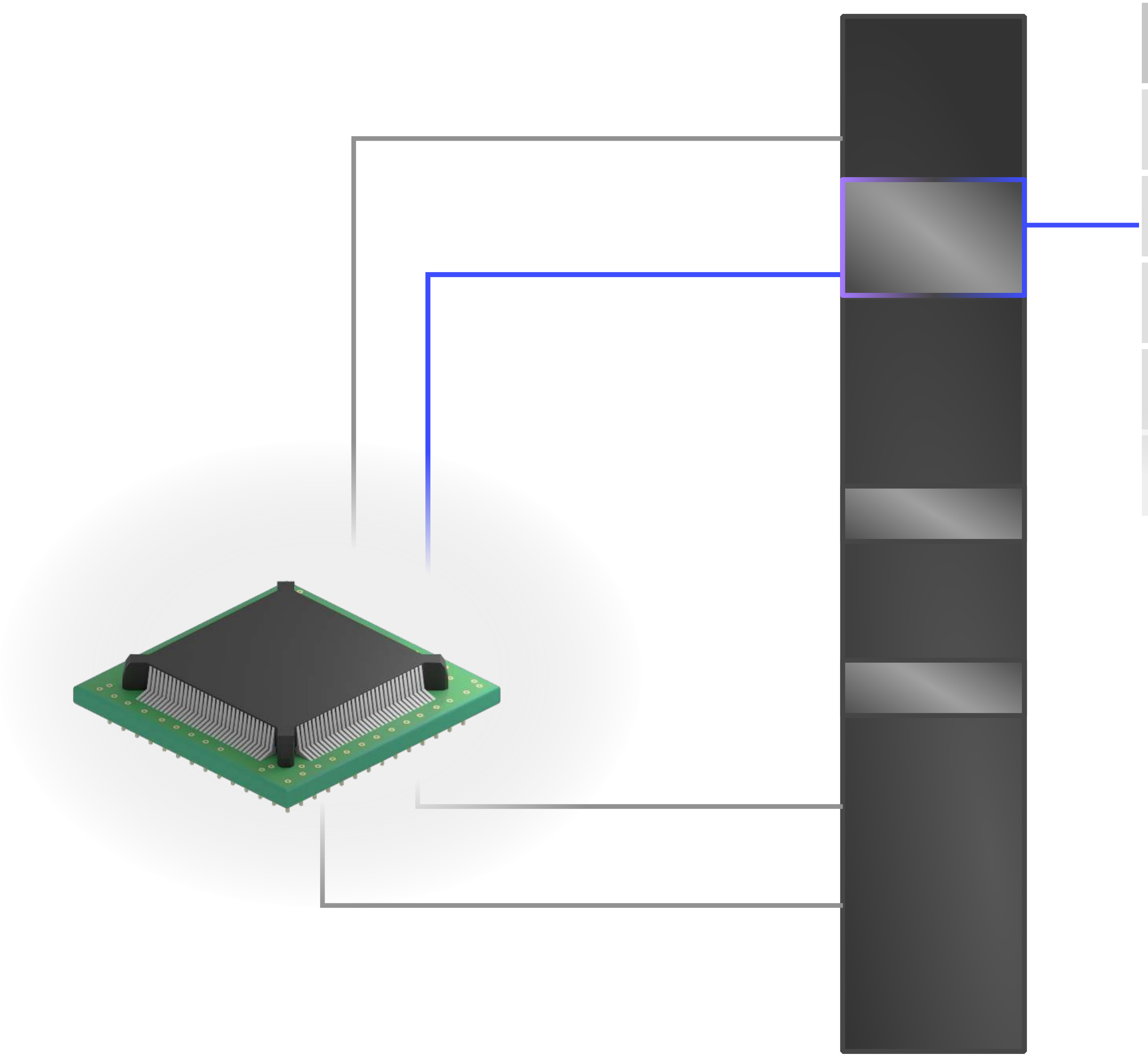


Reshaping Port IO and MMIO



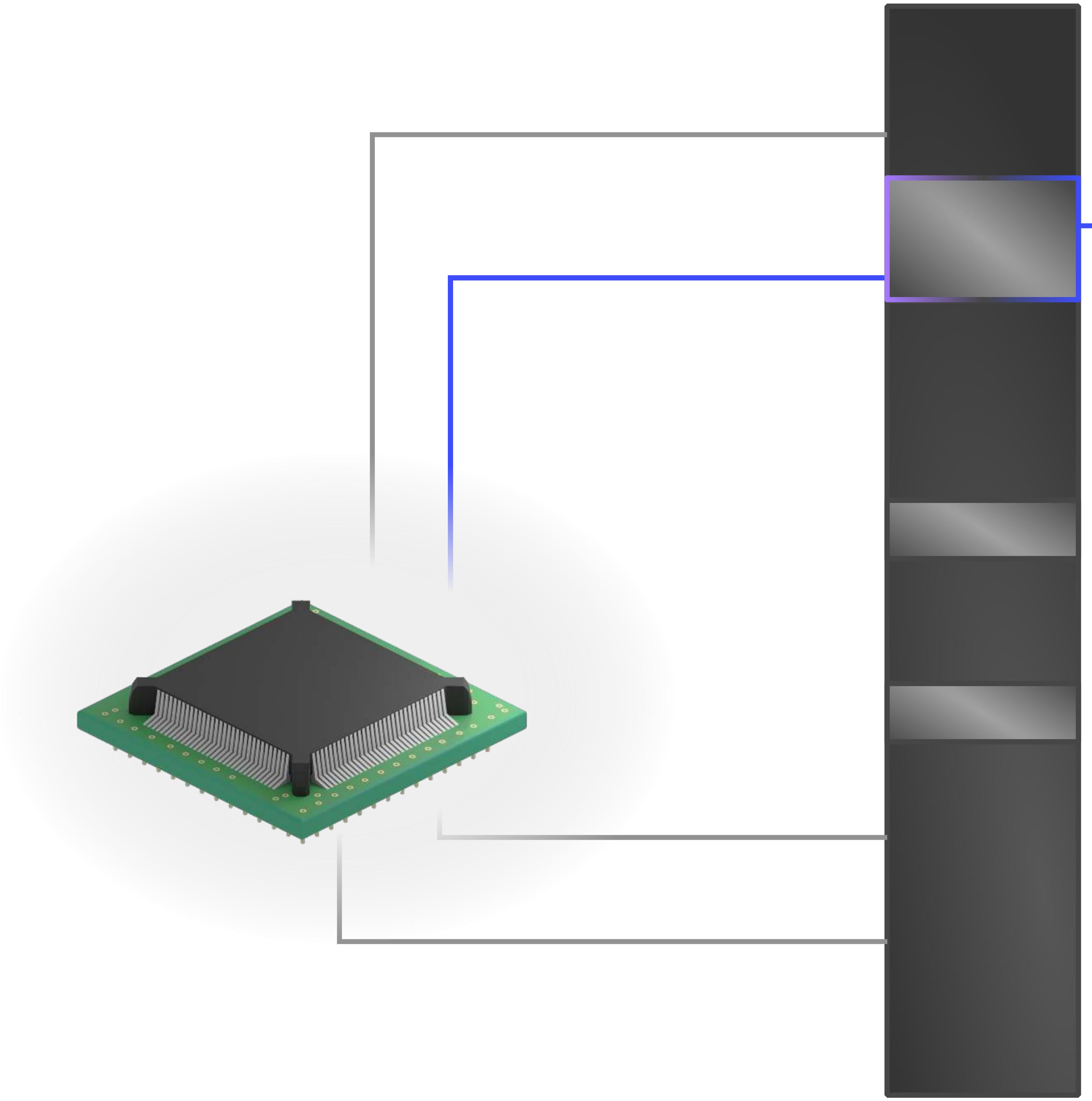
Start	End	Name
00000000	0009FFFF	RAM
000A0000	000BFFFF	VGA
000C0000	FEBBFFFF	RAM
FEBC0000	FEBDFFFF	NETWORK
[illegible]	[illegible]	RAM

Reshaping Port IO and MMIO



Start	End	Name
00000000	0009FFFF	RAM
000A0000	000BFFFF	VGA
000C0000	FEBBFFFF	RAM
FEBC0000	FEBDFFFF	NETWORK

Reshaping Port IO and MMIO

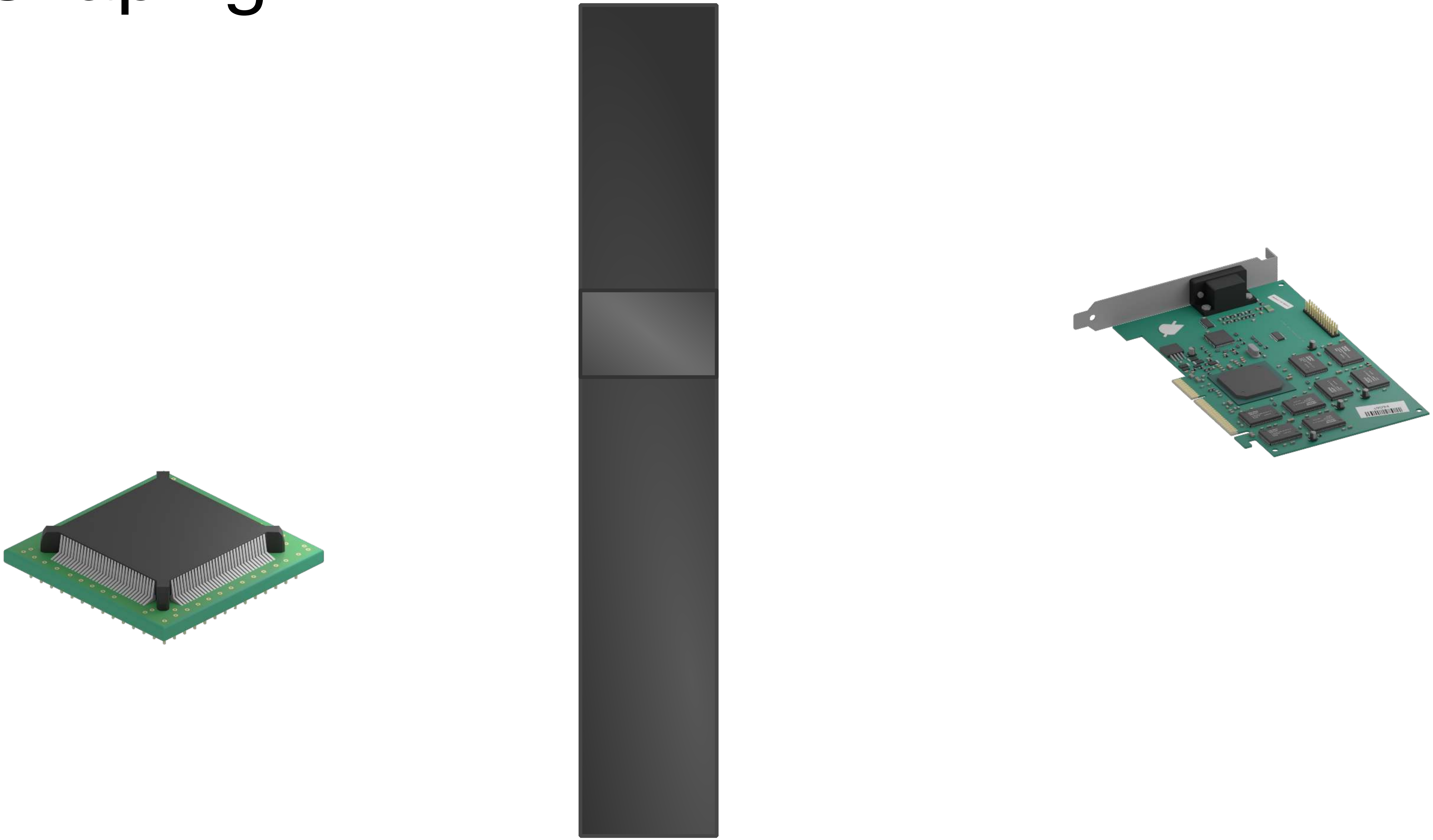


Start	End	Name
00000000	0009FFFF	RAM
000A0000	000BFFFF	VGA
000C0000	FEBBFFFF	RAM
FEBC0000	FEBDFFFF	NETWORK

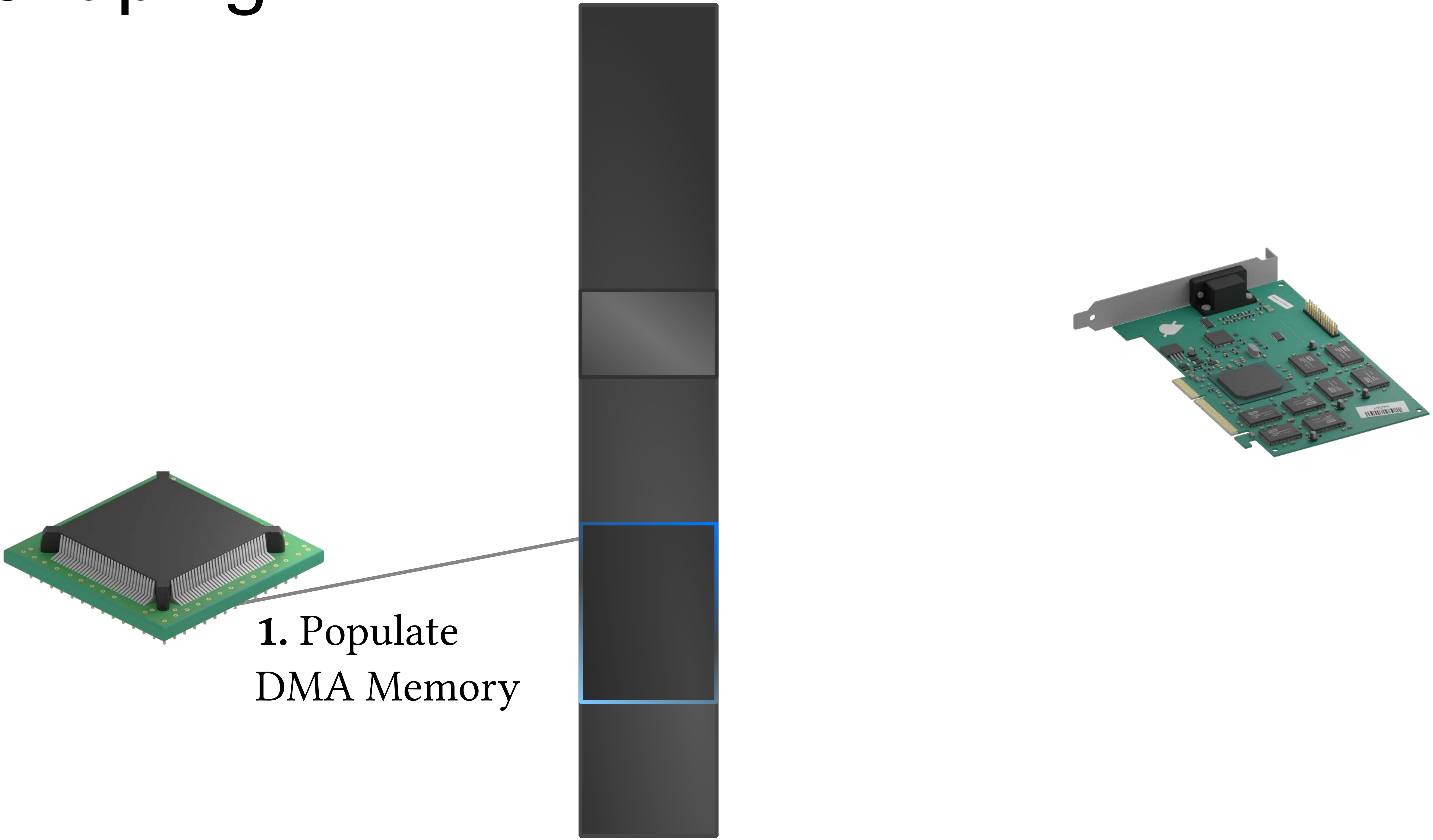
Hypervisors must track PIO/MMIO regions to trap and emulate accesses.

The memory layout table provides a perfect view of active IO regions.

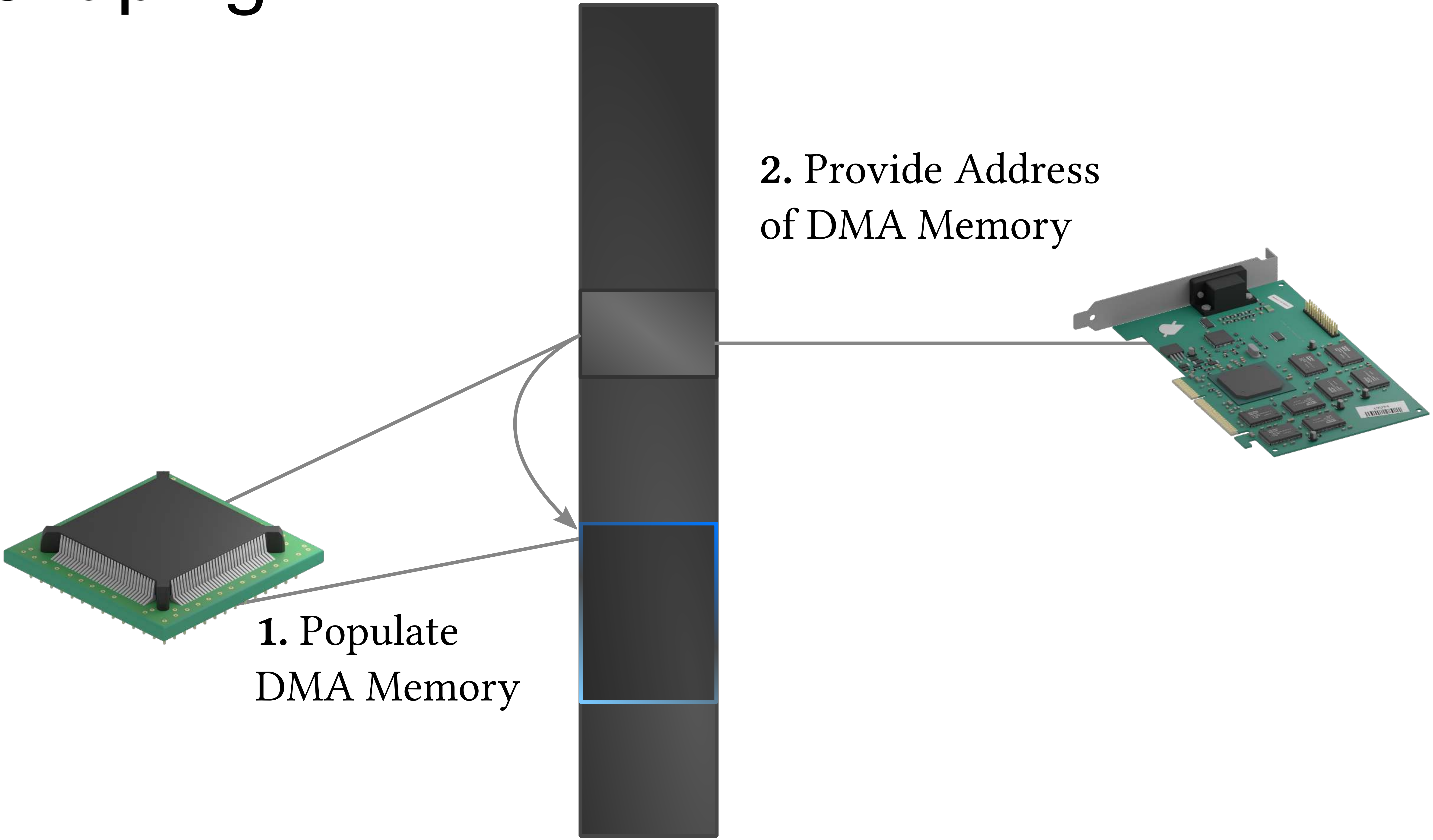
Reshaping DMA



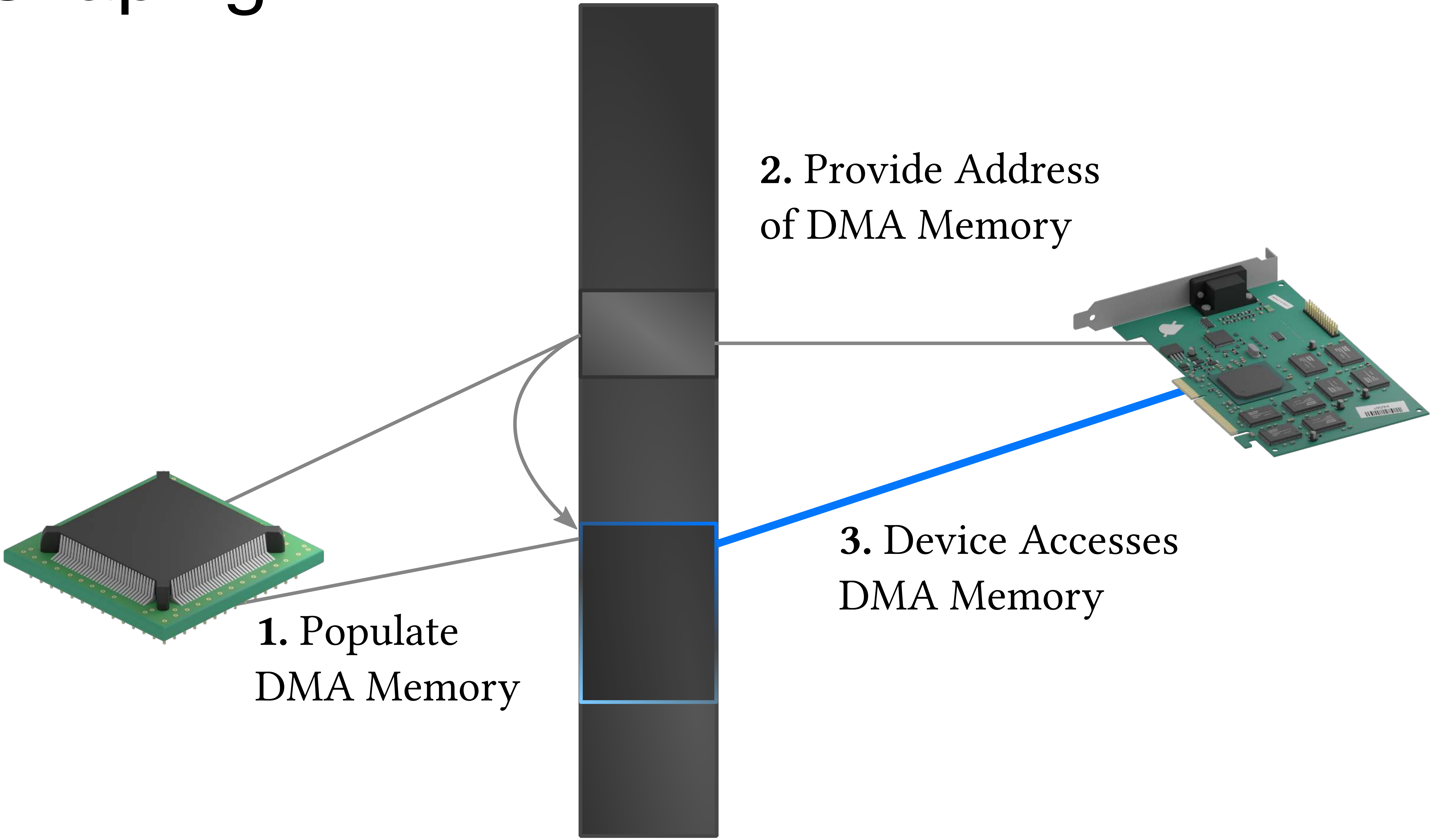
Reshaping DMA



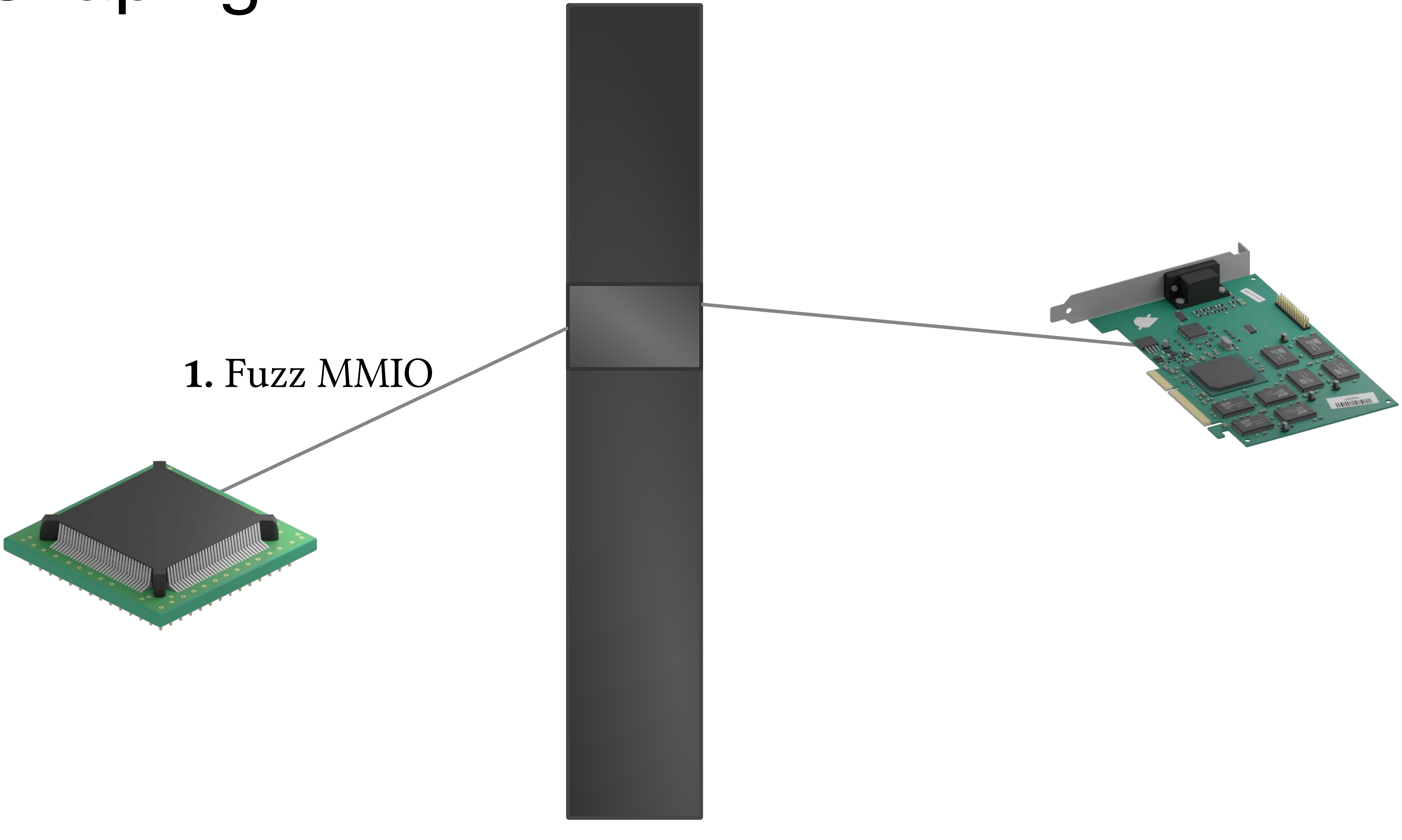
Reshaping DMA



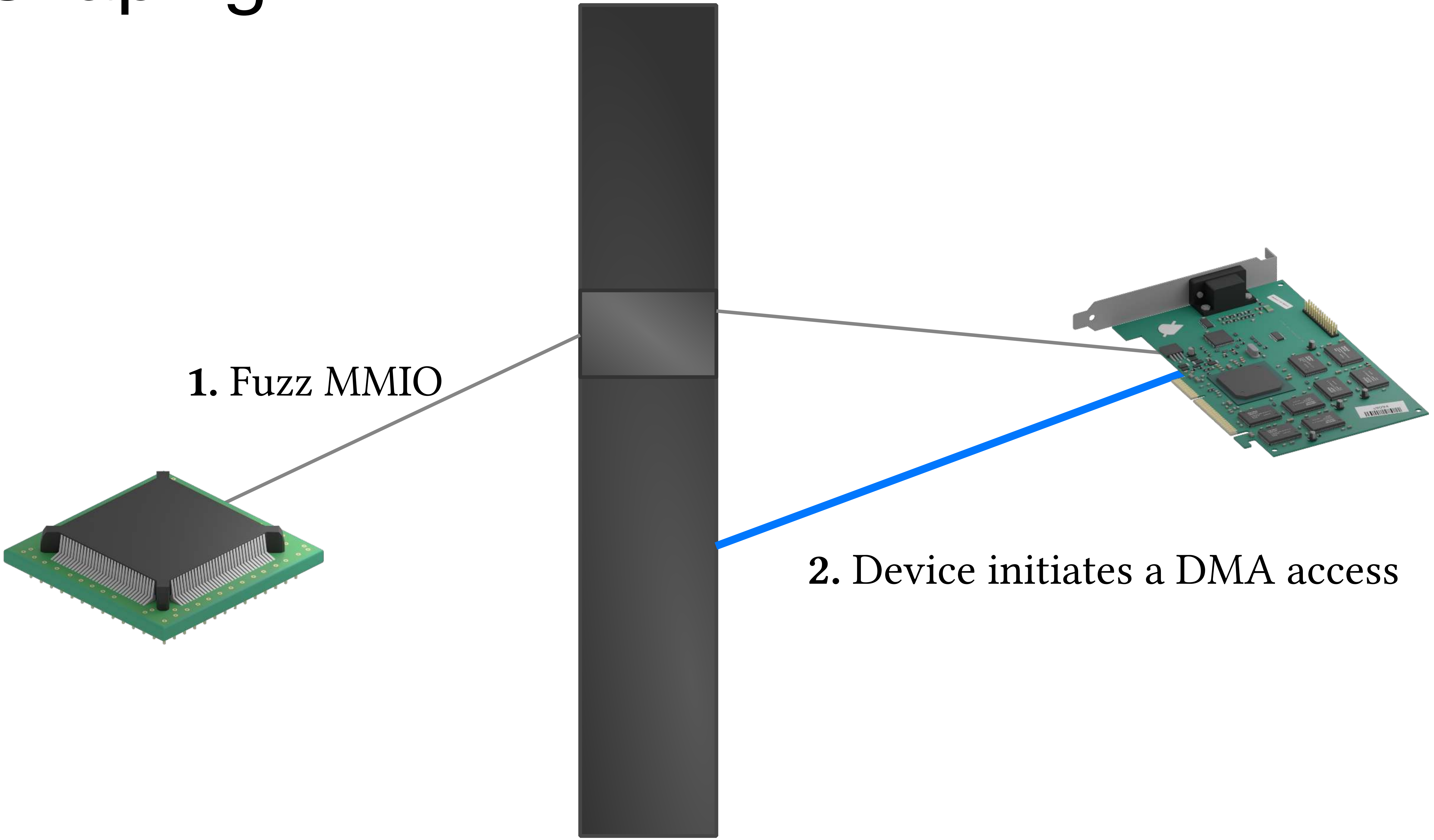
Reshaping DMA



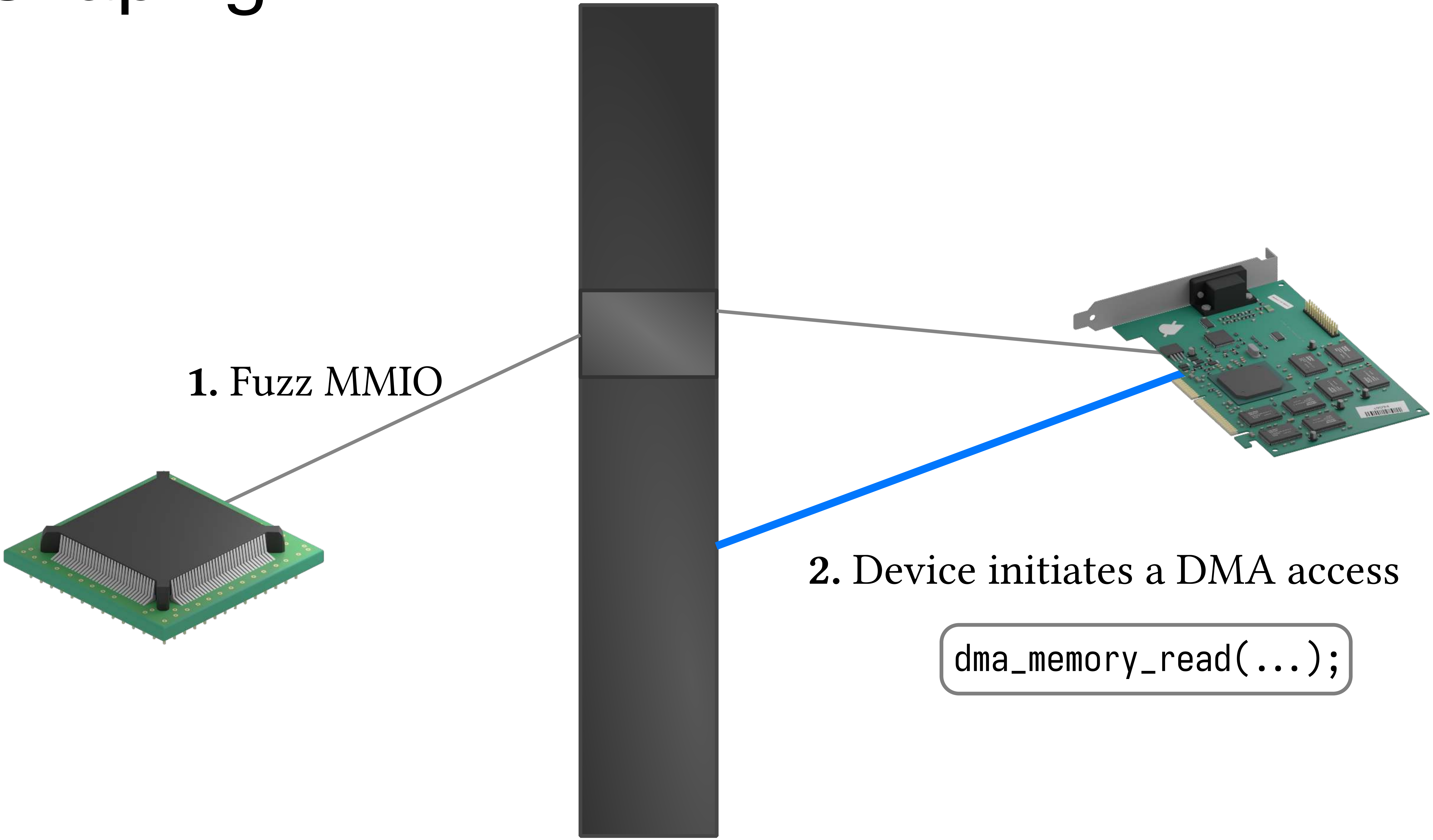
Reshaping DMA



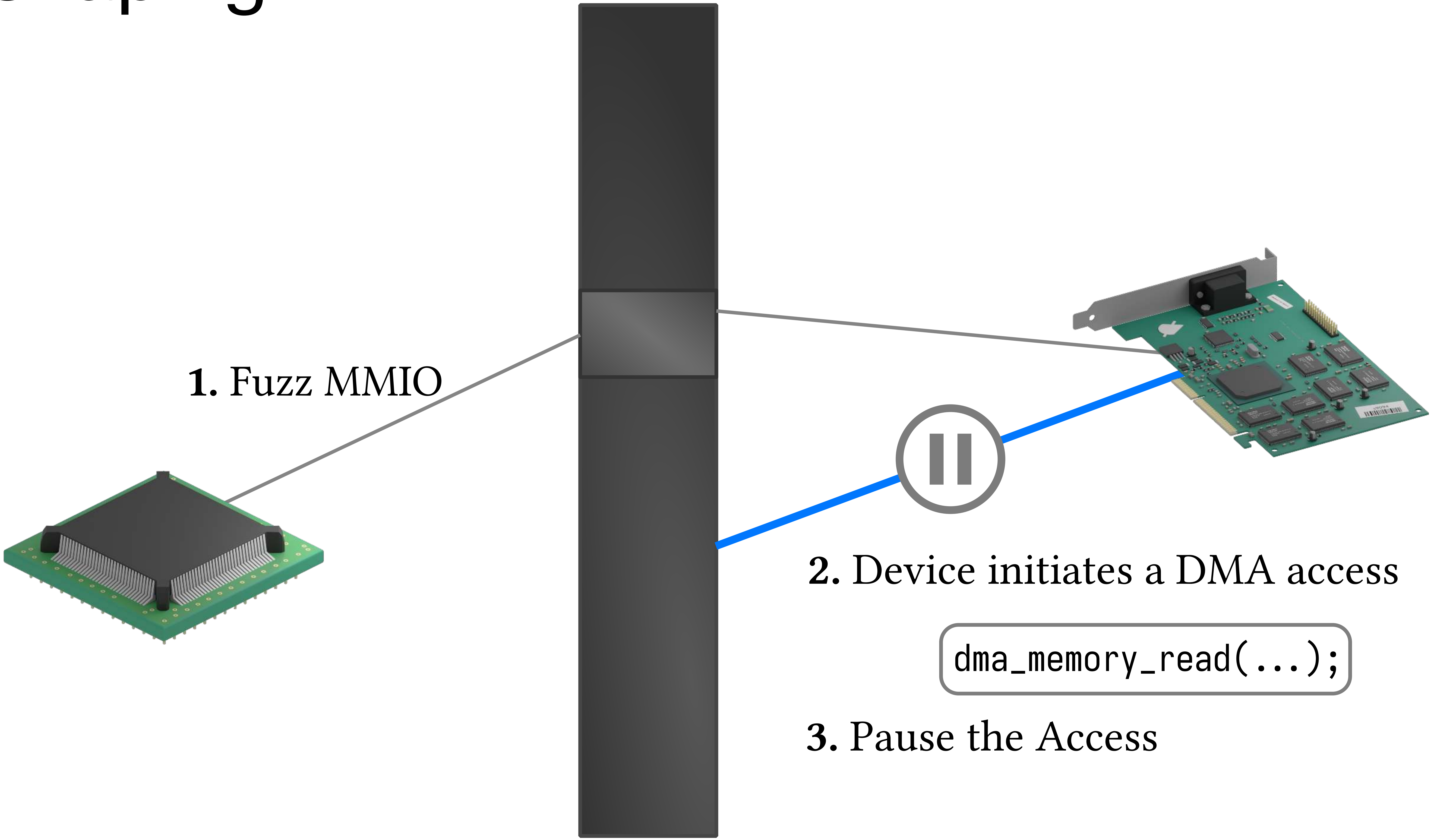
Reshaping DMA



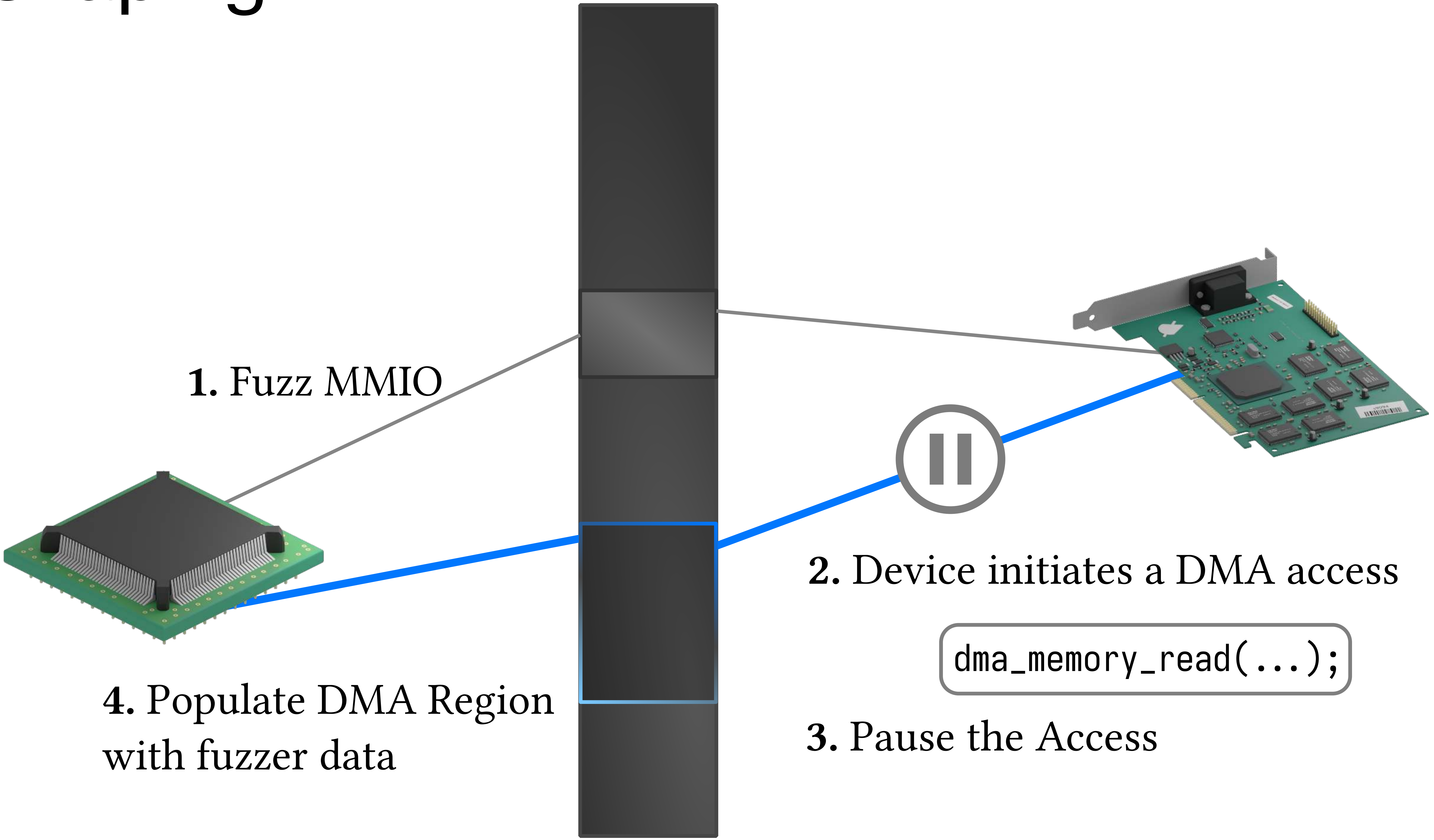
Reshaping DMA



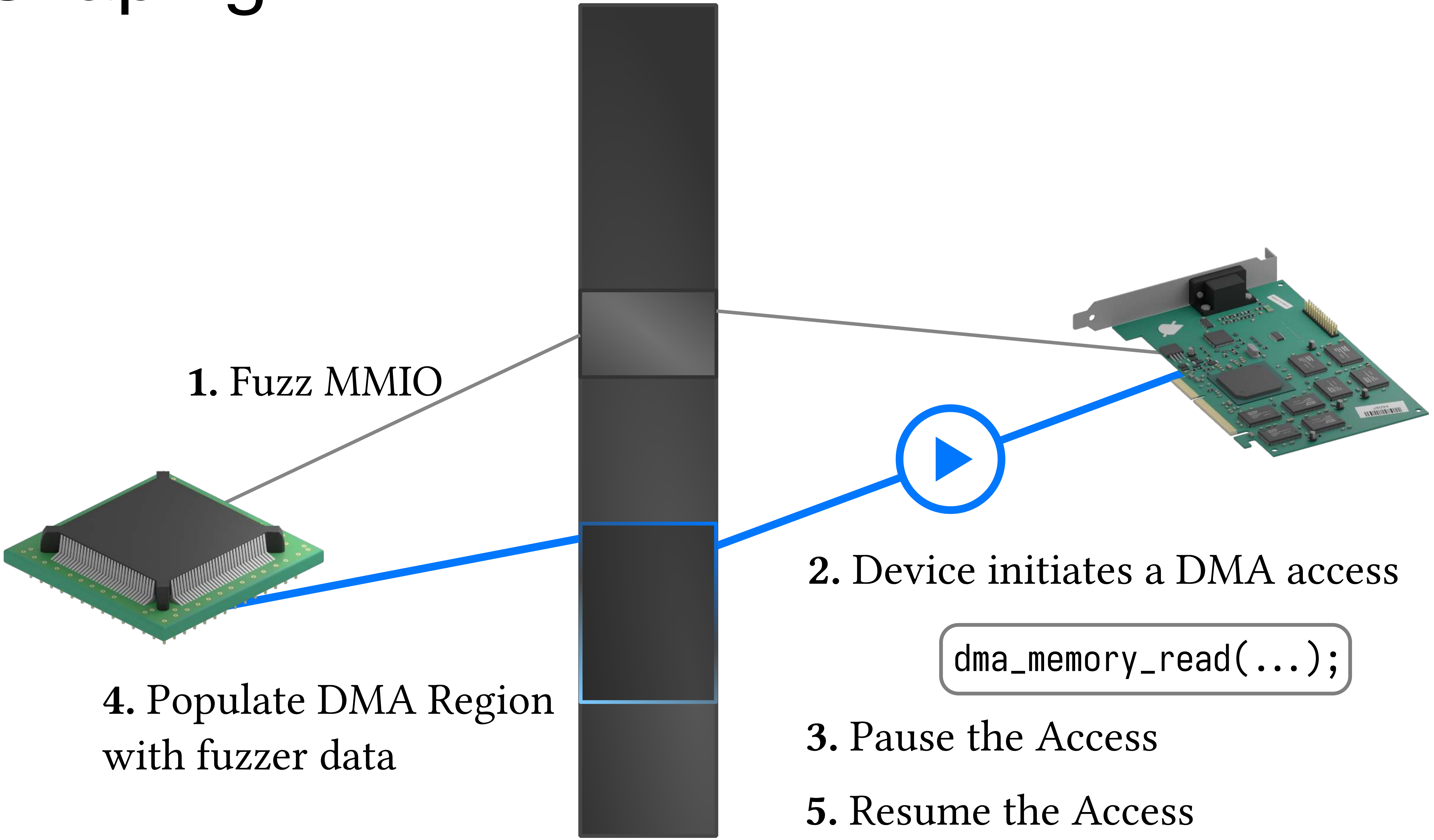
Reshaping DMA



Reshaping DMA



Reshaping DMA



Evaluation

- Two Hypervisors (QEMU and Bhyve)
- 33 Virtual Devices
- Coverage
  - Fuzzed for 24 Hours
  - 81% Overall
  - Equal/Higher coverage for 13/16 Devices
- DMA Evaluation
  - Improves Coverage for 24/33 Devices

Evaluation

- Two Hypervisors (QEMU and Bhyve)
- Coverage
  - Fuzzed for 24 Hours
  - 81% Overall
  - Equal/Higher coverage for 13/16 Devices
- DMA Evaluation
  - Improves Coverage for 24/33 Devices

Audio
ac97
cs4231a
es1370
intel-hda
sb16

IBM PC
fdc
parallel
serial

Block
ide/core
ahci
sdhci
virtio-blk
virtio-scsi
megasas
sd
scsi-disk

Network
eepro100
e1000
e1000e_core
ne2000
pcnet
rtl8139
vmxnet3
virtio-net

Graphics
virtio-gpu
cirrus_vga

USB
hcd-ehci
hcd-xhci

ARM
arm_gic
smc91c111
xgmac

bhyve
pci_xhci
virtio_block

Evaluation

- Two Hypervisors (QEMU and Bhyve)

Device	VDF‡ 25-65 Days	Hyper-Cube* 24 Hours	Nyx‡ 24 Hours	QMorphuzz 24 Hours	Bug
Source File	Cov.	Cov.	Cov.	Cov.	
Block					
ide/core	27.5%	74.87%	74.69%	78.63%	✔
ahci				80.86%	✔
sdhci	90.5%	81.15%	88.93%	84.8%	✔
virtio-blk				68.51%	✔
virtio-scsi				66.78%	✔
Average	61.67%	76.35%	78.16%	**85.76%**	
Average (All devices)				**81.08%**	

- DMA Evaluation
  - Improves Coverage for 24/33 Devices

Evaluation

- Two Hypervisors (QEMU and Bhyve)

Device	**No-DMA** 24 Hours	**Scratch-Buffer** 24 Hours	**QMorphuzz** 24 Hours
Source File	Cov.	Cov.	Cov.
Network			
eepro100	87.13%	87.13% (0.00)	89.26% (+2.13)
e1000	65.77%	66.14% (+0.37)	89.23% (+23.09)
e1000e_core	75.24%	75.84% (+0.60)	90.54% (+14.70)
ne2000	82.95%	83.47% (+0.52)	98.71% (+15.24)
pcnet	71.38%	72.72% (+1.34)	96.35% (+23.63)
Average	67.51%	69.42%	81.08%

Bugs

66 New (110 Total)

29 Assertion Failures

8 Stack Overflow

8 Null-Ptr Deref

7 UAF

7 Buffer Overflow

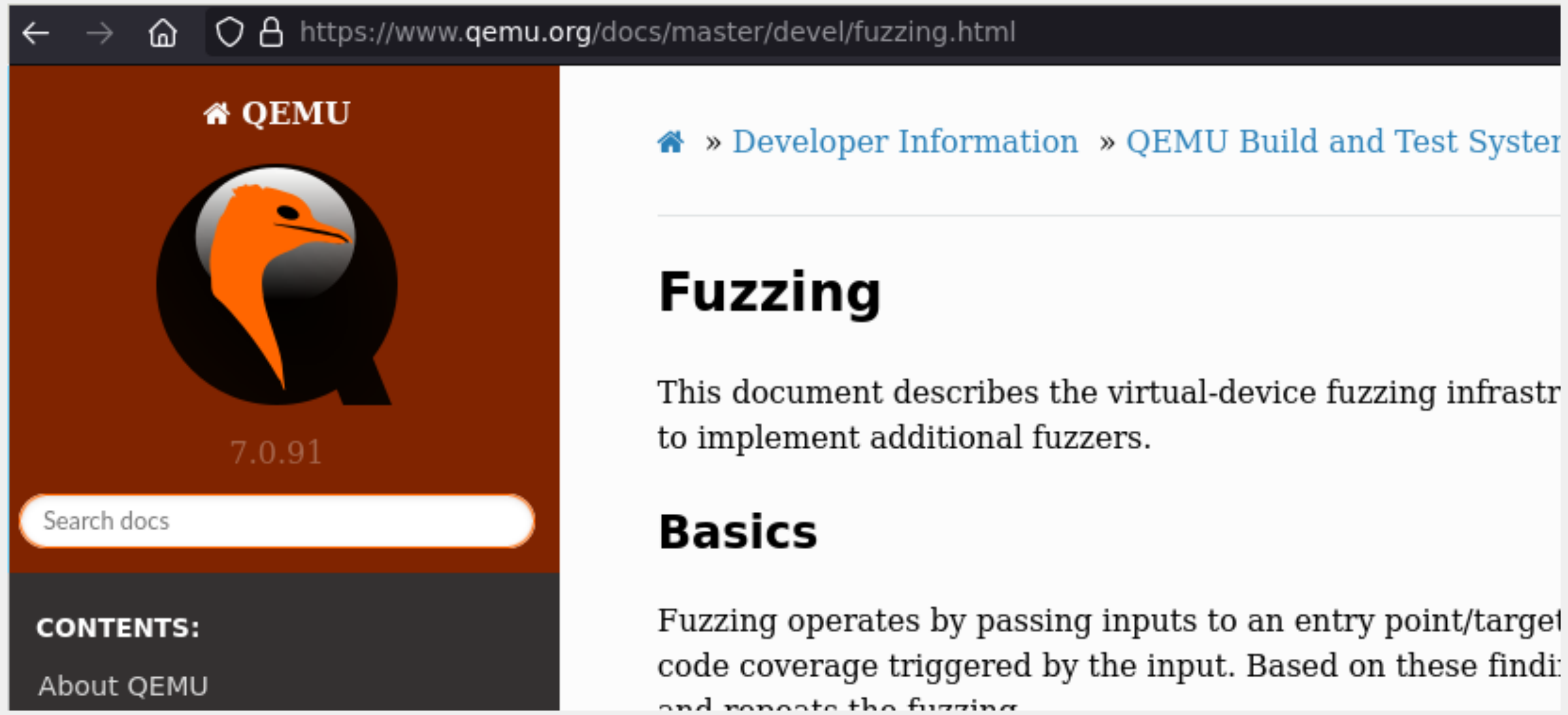
7 Other

Assertion-failure in audio_bug
Assertion-failure in mch_update_pciexbar
Assertion-failure in vmxnet3_validate_interrupt_idx
Assertion-failure in vmxnet3_validate_queues
Assertion-failure in address_space_stw_le_cached through virtio-net
Assertion-failure in address_space_stw_le_cached through virtio-blk
Assertion-failure in address_space_cache_invalidate through virtio-gpu
Assertion-failure in address_space_unmap through ahci_map_clb_address
Assertion-failure in address_space_unmap through virtio-blk
Assertion-failure in virtio_blk_reset
Assertion-failure in bdrv_aio_cancel
Assertion-failure in bmdma_active_if
Assertion-failure in e1000e_write_lgcy_rx_descr
Assertion-failure in e1000e_write_rx_descr
Assertion-failure in e1000e_write_to_rx_buffers
Assertion-failure in e1000e_intrmgr_on_throttling_timer
Assertion-failure in e1000e_intmgr_collect_delayed_causes
Assertion-failure in eth_get_gso_type through e1000e
Assertion-failure in iov_from_buf_full through e1000e
Assertion-failure in net_tx_pkt_add_raw_fragment through vmxnet3
Assertion-failure in net_tx_pkt_reset through vmxnet3
Assertion-failure in pci_bus_get_irq_level
Assertion-failure in scsi_dma_complete, with megasas
Assertion-failure in usb_detach
Assertion-failure in ati_reg_read_offs and ati_reg_write_offs
Assertion in modify_bar_registration
Assertion in unregister_mem
Assertion in pci_vtnet_proctx
Assertion in pci_vtnet_cfgwrite
Assertion-failure in gic_clear_pending_sgi
Assertion-failure in bcm2835_thermal_read
Assertion-failure in dwc2_hsotg_write
Stack-overflow in ahci_cond_start_engines
Stack-overflow in _eth_get_rss_ex_dst_addr
Stack-overflow in rtlNUMBER_transmit_one
Stack-overflow in pcnet_poll_timer
Stack-overflow in e1000_receive_iov
Stack-overflow in flatview_do_translate through e1000
Stack-overflow in intel_hda_corb_run
Stack-overflow in xhci_pci_intr_raise
Null-Ptr Deref in virtio_write_config
Null-Ptr Deref in address_space_to_flatview through ide
Null-Ptr Deref in blk_bs
Null-Ptr Deref in megasas_command_complete
Null-Ptr Deref in megasas_handle_frame
Null-Ptr Deref in tcg_handle_interrupt
Null-Ptr Deref in usb_bus_from_device
Null-Ptr Deref in vq_getchain
Null-Ptr Deref in smc91c111_writeb
Heap-use-after-free in e1000e_write_packet_to_guest
Heap use-after-free in e1000e_write_to_rx_buffers
Heap-use-after-free in ehci_flush_qh
Heap-use-after-free in usb_packet_copy
Heap-use-after-free in usb_packet_unmap
Heap-use-after-free in virtio_gpu_ctrl_response
Heap-use-after-free through double-fetch in ehci
Heap-use-after-free in gic_dist_writeb
Buffer-underflow in xhci_runtime_write
Global-buffer-overflow in mode_sense_page
Heap-buffer-overflow in sdhci_write_dataport
Heap-buffer-overflow in sdhci_data_transfer
Heap-buffer-overflow in sd_erase
Heap-buffer-overflow in msix_table_mmio_write
Heap-buffer-overflow in pcnet_receive
Memcpy-param-overlap in flatview_write_continue
Memcpy param-overlap in ip_stripoptions
Memcpy param-overlap through e1000e_write_to_rx_buffers
Memory Exhaustion in vmxnet3_activate_device
Memory Exhaustion in hpet_timer
Infinite Loop in sdhci_data_transfer
Floating-point exception in ide_set_sector

Morphuzz is Upstream in QEMU

- Continuously fuzzed on OSS-Fuzz
- 200+ Issues Reported
- Reproducers are simple to use
- Bugs are caught before release

```
git clone git.qemu.org/qemu.git
```




```
commit 283f0a05e24a5e5fab78305f783f06215390d620
Author: Thomas Huth <thuth@redhat.com>
Date:   Thu Jul 15 21:32:19 2021 +0200

    hw/net/net_tx_pkt: Fix crash detected by fuzzer

    QEMU currently crashes when it's started like this:

    cat << EOF | ./qemu-system-i386 -device vmxnet3 -nodefaults -qtest stdio
    outl 0xcf8 0x80001014
    outl 0xcfc 0xe0001000
    outl 0xcf8 0x80001018
    outl 0xcf8 0x80001004
    outw 0xcfc 0x7
    outl 0xcf8 0x80001083
    write 0x0 0x1 0xe1
    write 0x1 0x1 0xfe
    write 0x2 0x1 0xbe
    write 0x3 0x1 0xba
    writeq 0xe0001020 0xefefff5ecafe0000
    writeq 0xe0001020 0xffff5e5ccafe0002
    EOF

    It hits this assertion:

    qemu-system-i386: ../qemu/hw/net/net_tx_pkt.c:453: net_tx_pkt_reset:
     Assertion `pkt->raw' failed.

    This happens because net_tx_pkt_init() is called with max_frags == 0 and
    thus the allocation
```

Bend the virtual-device input space to make it conducive to fuzzing

Use time-tested **off-the-shelf fuzzers**

Fuzz *any* device, across all PIO, MMIO, and DMA interfaces.
No per-device analysis or descriptions, needed

MORPHUZZ

HYPERPILL: Fuzzing for Hypervisor-bugs by Leveraging the Hardware Virtualization Interface

Alexander Bulekov*†§ alxndr@bu.edu
Qiang Liu *‡ qiang.liu@epfl.ch
Manuel Egele† megele@bu.edu
Mathias Payer* mathias.payer@nebelwelt.net

*EPFL †Boston University ‡Zhejiang University §Amazon

Abstract

The security guarantees of cloud computing depend on the isolation guarantees of the underlying hypervisors. Prior works have presented effective methods for automatically identifying vulnerabilities in hypervisors. However, these approaches are limited in scope. For instance, their implementation is typically hypervisor-specific and limited by requirements for detailed grammars, access to source-code, and assumptions about hypervisor behaviors. In practice, complex closed-source and recent open-source hypervisors are often not suitable for off-the-shelf fuzzing techniques.

HYPERPILL introduces a generic approach for fuzzing arbitrary hypervisors. HYPERPILL leverages the insight that although hypervisor implementations are diverse, all hypervisors rely on the identical underlying hardware-virtualization interface to manage virtual-machines. To take advantage of the hardware-virtualization interface, HYPERPILL makes a snapshot of the hypervisor, inspects the snapshotted hardware state to enumerate the hypervisor's input-spaces, and leverages feedback-guided snapshot-fuzzing within an emulated environment to identify vulnerabilities in arbitrary hypervisors. In our evaluation, we found that beyond being the first hypervisor-fuzzer capable of identifying vulnerabilities in arbitrary hypervisors across all major attack-surfaces (i.e., PIO/MMIO/Hypercalls/DMA), HYPERPILL also outperforms state-of-the-art approaches that rely on access to source-code, due to the granularity of feedback provided by HYPERPILL's emulation-based approach. In terms of coverage, HYPERPILL outperformed past fuzzers for 10/12 QEMU devices, without the API hooking or source-code instrumentation techniques required by prior works. HYPERPILL identified 26 new bugs in recent versions of QEMU, Hyper-V, and macOS Virtualization Framework across four device-categories.

ll work was completed prior to author joining Amaz

1 Introduction

Hypervisors provide the security foundations necessary for the cloud. They enable efficient use of hardware resources, by colocating workloads from multiple tenants on the same bare-metal machines, isolated in individual virtual-machines (VMs). As such, hypervisors ensure that code running in VMs cannot violate the virtualization boundary (e.g., by performing a VM escape attack) and compromise the workloads of the other tenants, or the hypervisor itself.

Unfortunately, VM escape attacks are a tangible reality. Hundreds of bugs have been identified in the complex hypervisor code. Due to the severity of these bugs, hypervisor compromises are awarded large bug bounties, similar to other high-value targets such as web browsers and mobile devices [53]. In parallel, fuzzing has emerged as one of the most powerful techniques for automatically uncovering vulnerabilities in a large range of software [4,10,14,18,20,21,27,28,39,45,49,52]. As such, a significant amount of academic research has focused on leveraging fuzzing to automatically identify bugs in hypervisor code, so that they can be promptly fixed, preventing malicious exploitation [6, 13, 26, 29, 32, 37, 38].

State-of-the-art approaches [6, 26] are capable of automatically finding complex bugs across most major attack-surfaces (i.e., PIO/MMIO/DMA). However, these approaches rely on access and manual modifications to hypervisor source-code to effectively fuzz virtual-devices. Even with access to source-code, porting current methods to new targets is a non-trivial process that requires considerable manual effort by an expert. Furthermore, most fuzzers do not handle the hypercall attack-surface as hypercalls are often implemented in a separate component from the core device-emulation (e.g., in the OS kernel), for performance reasons. Thus, even the open-source targets such as QEMU and ... extensively fuzzed, closed-... macOS Virtuali... fuzz...

HYPERPILL

Fuzzing for Hypervisor-bugs by Leveraging the Hardware Virtualization Interface

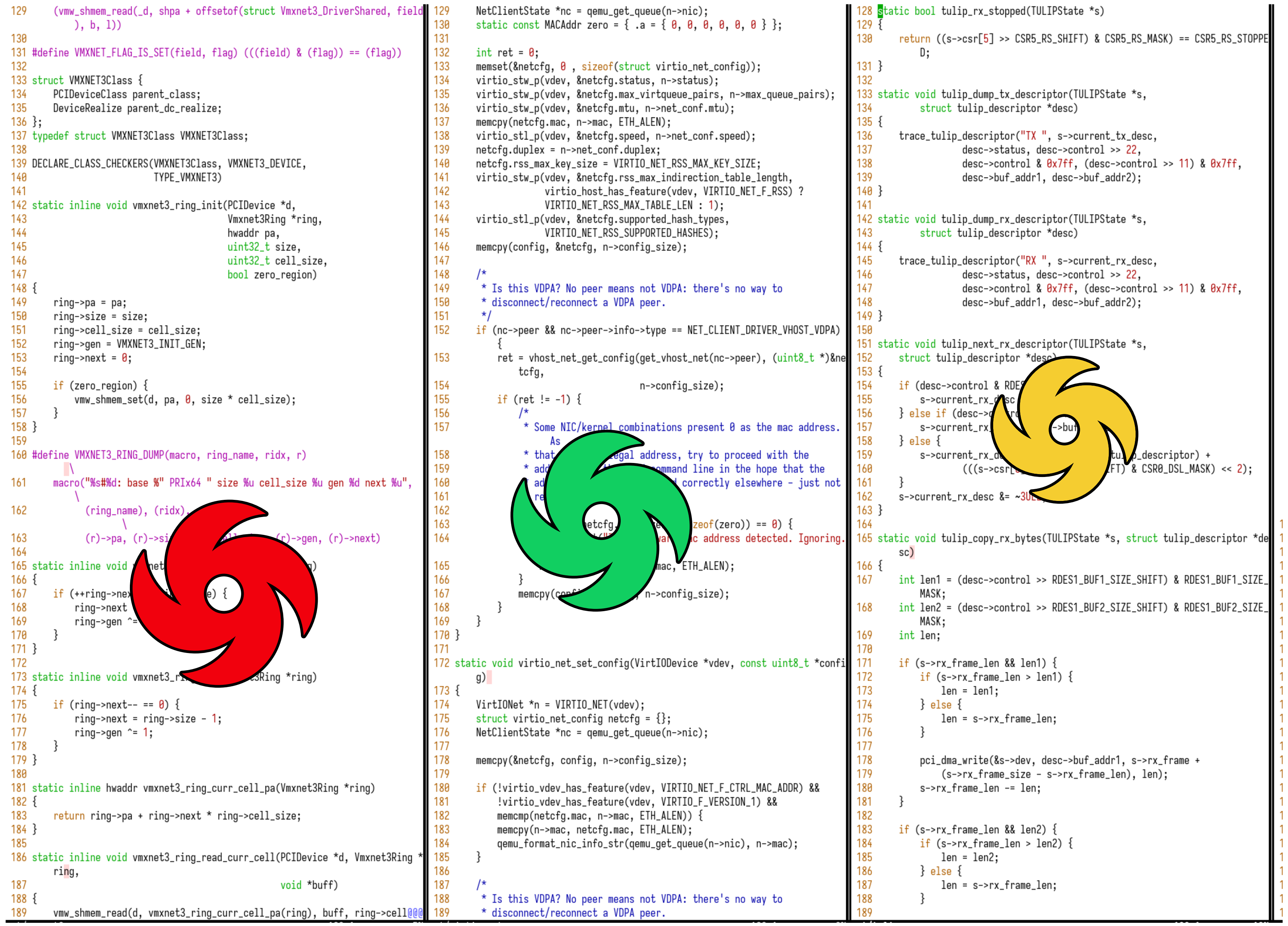
USENIX Security 2024

Other Hypervisors

Other Hypervisors

QEMU

Bhyve



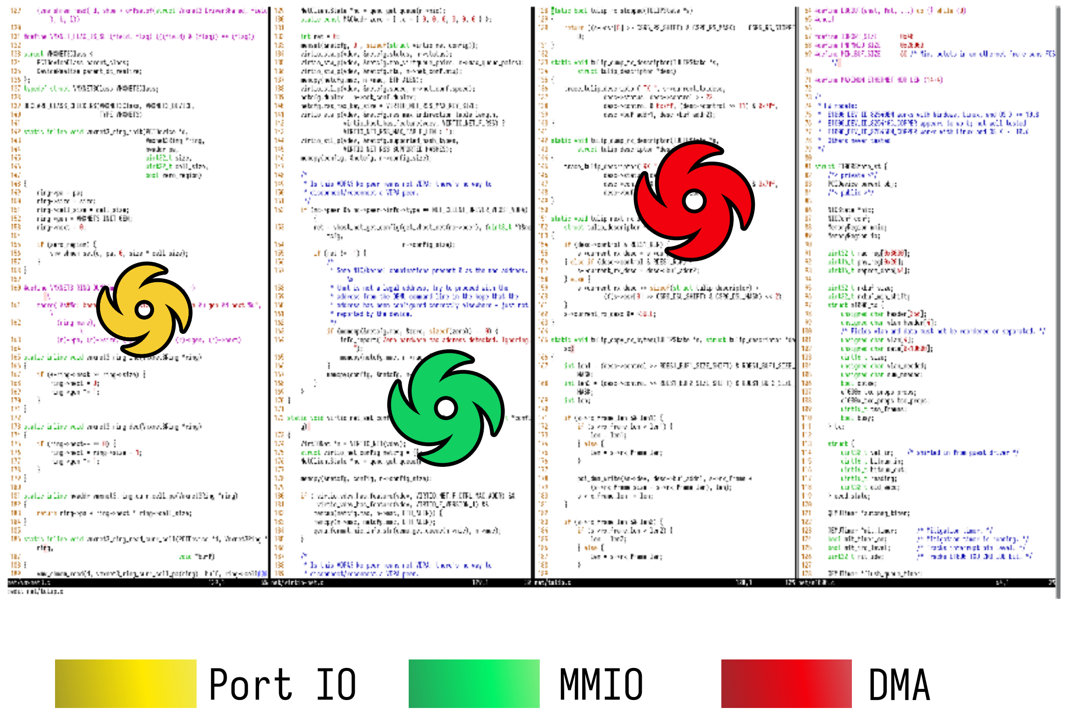
Port IO

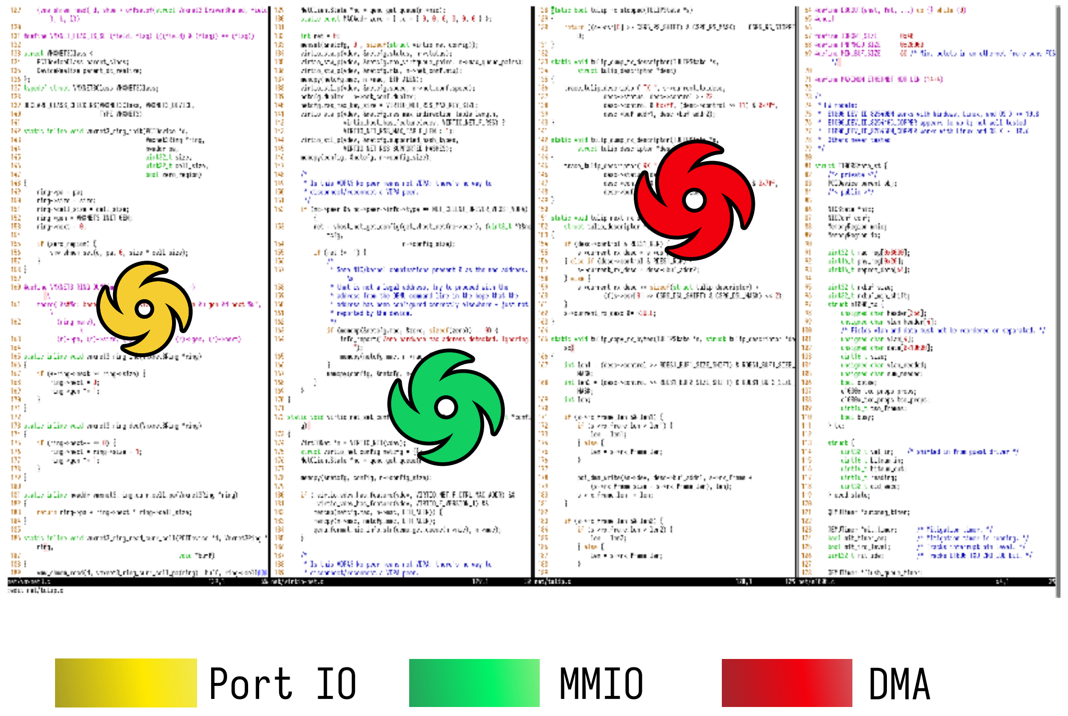
MMIO

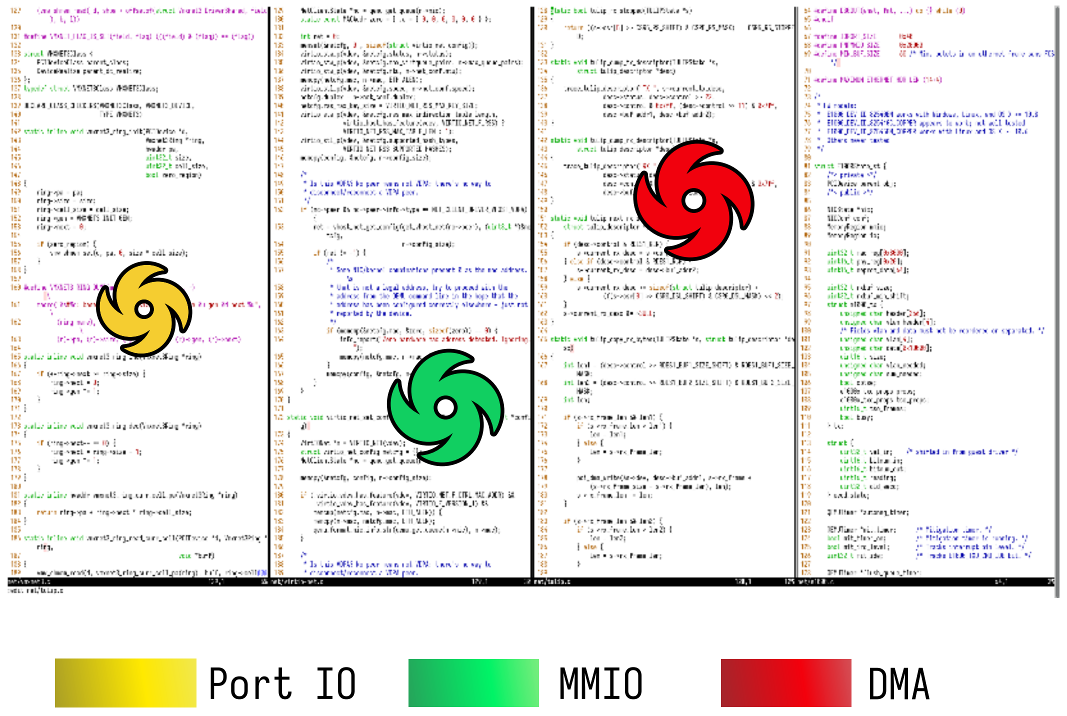
DMA

Reshaping

Other Hypervisors







Other Hypervisors

Microsoft Hyper-V

vmware®

Parallels™


```
00007fe0: 8b35 5d44 dcd9 c3a6 e007 8ca5 01fc d5df  .5]D............
00007ff0: 68eb 956d 69f2 3dff ccba c94d ba95 931c  h..mi.=....M....
00008000: 6f99 64a3 8888 13a3 b84f dcb6 cc9e 5071  o.d......O....Pq
00008010: cdba c94d 6c9e 7aaa cfba c94d b85b 938b  ...Ml.z....M.[..
00008020: f93c 81a9 d0ba c94d f9cb 8d8d 5a9b e8b0  .<.....M....Z...
00008030: cac2 d3b8 dcb7 5e4c e15f 8598 74fe 7497  ......^L._..t.t.
00008040: fc39 d79c 5b76 053d d2ba c94d 8883 b0e8  .9..[v.=...M....
00008050: 2575 a29f 2551 64ff e0d9 fa85 b7bf 4d8c  %u..%Qd.......M.
00008060: 7e4f 3c2a 7413 4b0b c2b1 14b7 2e43 d893  ~O<*t.K......C..
00008070: 6d16 bc16 9879 87af f2c1 2a2b a19f a90f  m....y....*+....
00008080: 265a e8ab 6828 5bb3 555f 2673 3a0d 85b3  &Z..h([.U_&s:...
00008090: 3140 4702 d79a 4813 5c52 6a8d c95a 414e  1@G...H.\Rj..ZAN
000080a0: 10e3 77ea 62cc e665 edda c8b7 6044 ce3e  ..w.b..e....`D.>
000080b0: a883 3d41 6083 4d73 42dc 4e7e fe8d 698e  ..=A`.MsB.N~..i.
000080c0: 30bb d143 3bc2 cee6 a444 6ecb e4a6 d4dc  0..C;....Dn.....
000080d0: 7d3b f166 9580 42cb 0ddb 60ea 7e5d 5d1e  };.f..B...`.~]].
000080e0: c421 5845 be42 31e0 6352 6a8d df4b 7d81  .!XE.B1.cRj..K}.
000080f0: e0d7 9f9a 8c34 feea 7bc9 1403 6fea da25  .....4..{...o..%
00008100: 4237 5fe9 67c9 421c 315b 851a 4ec5 a38c  B7_.g.B.1[..N...
00008110: b21b 1360 f345 7fba 17bc 55e2 f670 205e  ...`.E....U..p ^
00008120: b9c7 5bdf 648b d75a 051e 5969 527d 3165  ..[.d..Z..YiR}1e
00008130: 23c6 d3b3 fa88 668a 0ddf 5683 309c dadf  #.....f...V.0...
00008140: 1421 d944 d1e5 d412 4b04 750f 4aba 9a7c  .!.D....K.u.J..|
00008150: 43cd ffa5 2a4b 1521 9ecf a439 375d 1154  C...*K.!...97].T
00008160: 4704 581e 04d1 5a69 335d 8757 df96 45b7  G.X...Zi3].W..E.
00008170: 74a7 d4e9 4a68 1266 82e6 5d1e fbdf daa8  t...Jh.f..].....
00008180: 68ff 6a03 0cf1 39ad 770a a100 eec1 2935  h.j...9.w.....)5
00008190: 238b 8fa6 b2e4 b68a e158 1fc4 ba9a 58ec  #........X....X.
000081a0: 29ee 64bc 5d47 79f0 87c5 cd59 3f2d c613  ).d.]Gy....Y?-..
000081b0: 5e88 3678 6872 cb13 8241 8060 e0ad e8c3  ^.6xhr...A.`....
000081c0: d1f6 5710 37fc 9aea 0f31 8c35 f6b6 f928  ..W.7....1.5...(
000081d0: 892e 893c ea0e 9eca d165 ce6d ef5e 11a5  ...<.....e.m.^..
000081e0: 64fd 1810 460c 670b 693a f4e9 e9fc 483e  d...F.g.i:....H>
000081f0: ca6c 73e3 c42e c99d 96ed 1881 bb54 7cf0  .ls..........T|.
00008200: 8be2 c613 861f 85fc 72fa 811b 092d 4dea  ........r....-M.
00008210: 567b 80e7 095a 9248 3cc8 7a5c fc25 7c53  V{...Z.H<.z\.%|S
00008220: 95de 5151 3c27 ca2b ba07 bc5a c210 c9c9  ..QQ<'.+...Z....
00008230: 0519 bc3e 1cb5 b2c4 4be3 cf89 4727 6934  ...>....K...G'i4
00008240: 88b9 e2c5 b2ac 8773 ece9 432b 99c2 fb28  .......s..C+...(
00008250: 82d4 b218 cd6a 98bb 7a4a 1595 da85 c370  .....j..zJ.....p
```



```
00007fe0: 8b35 5d44 dcd9 c3a6 e007 8ca5 01fc d5df  .5]D............
00007ff0: 68eb 956d 69f2 3dff ccba c94d ba95 931c  h..mi.=....M....
00008000: 6f99 64a3 8888 13a3 b84f dcb6 cc9e 5071  o.d......O....Pq
00008010: cdba c94d 6c9e 7aaa cfba c94d b85b 938b  ...Ml.z....M.[..
00008020: f93c 81a9 d0ba c94d f9cb 8d8d 5a9b e8b0  .<.....M....Z...
00008030: cac2 d3b8 dcb7 5e4c e15f 8598 74fe 7497  ......^L._..t.t.
00008040: fc39 d79c 5b76 053d d2ba c94d 8883 b0e8  .9..[v.=...M....
00008050: 2575 a29f 2551 64ff e0d9 fa85 b7bf 4d8c  %u..%Qd.......M.
00008060: 7e4f 3c2a 7413 4b0b c2b1 14b7 2e43 d893  ~O<*t.K......C..
00008070: 6d16 bc16 9879 87af f2c1 2a2b a19f a90f  m....y....*+....
00008080: 265a e8ab 6828 5bb3 555f 2673 3a0d 85b3  &Z..h([.U_&s:...
00008090: 3140 4702 d79a 4813 5c52 6a8d c95a 414e  1@G...H.\Rj..ZAN
000080a0: 10e3 77ea 62cc e665 edda c8b7 6044 ce3e  ..w.b..e....`D.>
000080b0: a883 3d41 6083 4d73 42dc 4e7e fe8d 698e  ..=A`.MsB.N~..i.
000080c0: 30bb d143 3bc2 cee6 a444 6ecb e4a6 d4dc  0..C;....Dn.....
000080d0: 7d3b f166 9580 42cb 0ddb 60ea 7e5d 5d1e  };.f..B...`.~]].
000080e0: c421 5845 be42 31e0 6352 6a8d df4b 7d81  .!XE.B1.cRj..K}.
000080f0: e0d7 9f9a 8c34 feea 7bc9 1403 6fea da25  .....4..{...o..%
00008100: 4237 5fe9 67c9 421c 315b 851a 4ec5 a38c  B7_.g.B.1[..N...
00008110: b21b 1360 f345 7fba 17bc 55e2 f670 205e  ...`.E....U..p ^
00008120: b9c7 5bdf 648b d75a 051e 5969 527d 3165  ..[.d..Z..YiR}1e
00008130: 23c6 d3b3 fa88 668a 0ddf 5683 309c dadf  #.....f...V.0...
00008140: 1421 d944 d1e5 d412 4b04 750f 4aba 9a7c  .!.D....K.u.J..|
00008150: 43cd ffa5 2a4b 1521 9ecf a439 375d 1154  C...*K.!...97].T
00008160: 4704 581e 04d1 5a69 335d 8757 df96 45b7  G.X...Zi3].W..E.
00008170: 74a7 d4e9 4a68 1266 82e6 5d1e fbdf daa8  t...Jh.f..].....
00008180: 68ff 6a03 0cf1 39ad 770a a100 eec1 2935  h.j...9.w.....)5
00008190: 238b 8fa6 b2e4 b68a e158 1fc4 ba9a 58ec  #........X....X.
000081a0: 29ee 64bc 5d47 79f0 87c5 cd59 3f2d c613  ).d.]Gy....Y?-..
000081b0: 5e88 3678 6872 cb13 8241 8060 e0ad e8c3  ^.6xhr...A.`....
000081c0: d1f6 5710 37fc 9aea 0f31 8c35 f6b6 f928  ..W.7....1.5...(
000081d0: 892e 893c ea0e 9eca d165 ce6d ef5e 11a5  ...<.....e.m.^..
000081e0: 64fd 1810 460c 670b 693a f4e9 e9fc 483e  d...F.g.i:....H>
000081f0: ca6c 73e3 c42e c99d 96ed 1881 bb54 7cf0  .ls..........T|.
00008200: 8be2 c613 861f 85fc 72fa 811b 092d 4dea  ........r....-M.
00008210: 567b 80e7 095a 9248 3cc8 7a5c fc25 7c53  V{...Z.H<.z\.%|S
00008220: 95de 5151 3c27 ca2b ba07 bc5a c210 c9c9  ..QQ<'.+...Z....
00008230: 0519 bc3e 1cb5 b2c4 4be3 cf89 4727 6934  ...>....K...G'i4
00008240: 88b9 e2c5 b2ac 8773 ece9 432b 99c2 fb28  .......s..C+...(
00008250: 82d4 b218 cd6a 98bb 7a4a 1595 da85 c370  .....j..zJ.....p
```

Other Hypervisors

VirtualBox

Firecracker

crosvm

Microsoft Hyper-V

Parallels

QEMU

vmware

Bhyve

Other Hypervisors

Fuzzer	Open-Source	Type-1	Language	Execution Environment	PV Platform	Hypercall identifier	EPT MMIO Mechanism
QEMU-KVM	●	○	C	K+U	VIRTIO	RAX	V+M
FIRECRACKER	●	○	Rust	K+U	VIRTIO	RAX	V+M
VIRTUALBOX	●	○	C++	K+U	VIRTIO/VMBus	RAX/RCX	M
VMWARE (PC)	○	○	C,C++	K+U	VMWare	RDX	
VMWARE ESX	○	●	C	K	VMWare	RDX	
MACOS	○	○	?	U	VIRTIO	RAX	V
HYPER-V	○	●	C++,?	VM+K+U	VMBus	RCX	V

Bhyve

Other Hypervisors

VirtualBox

Firecracker

crosvm

Microsoft Hyper-V

Parallels

QEMU

vmware

Bhyve

Other Hypervisors

intel

Intel® 64 and IA-32 Architectures Software Developer's Manual

Volume 3C:
System Programming Guide, Part 3

VirtualBox

Firecracker

crosvm

Microsoft Hyper-V

Parallels

QEMU

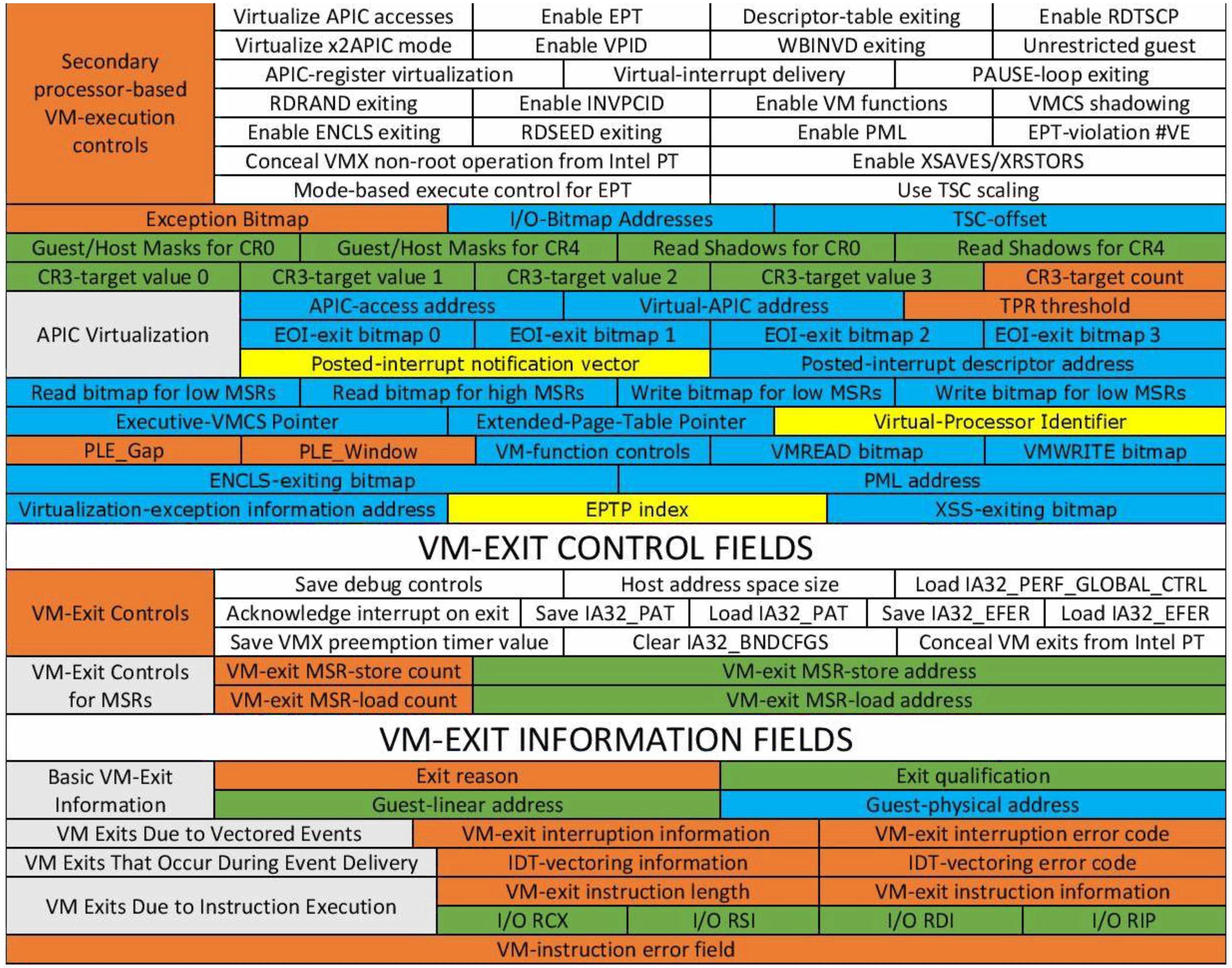
vmware

Bhyve

Other Hypervisors

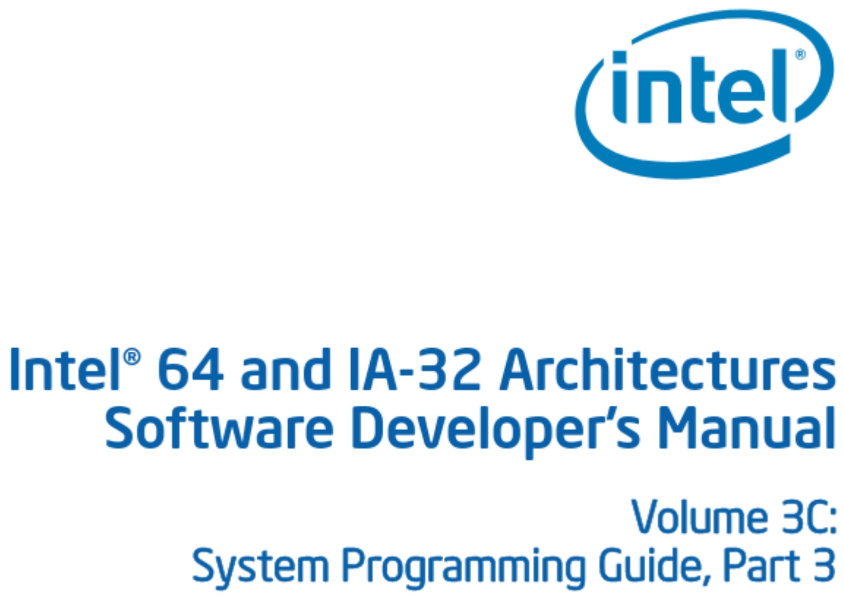
Intel® 64 and IA-32 Architectures Software Developer's Manual

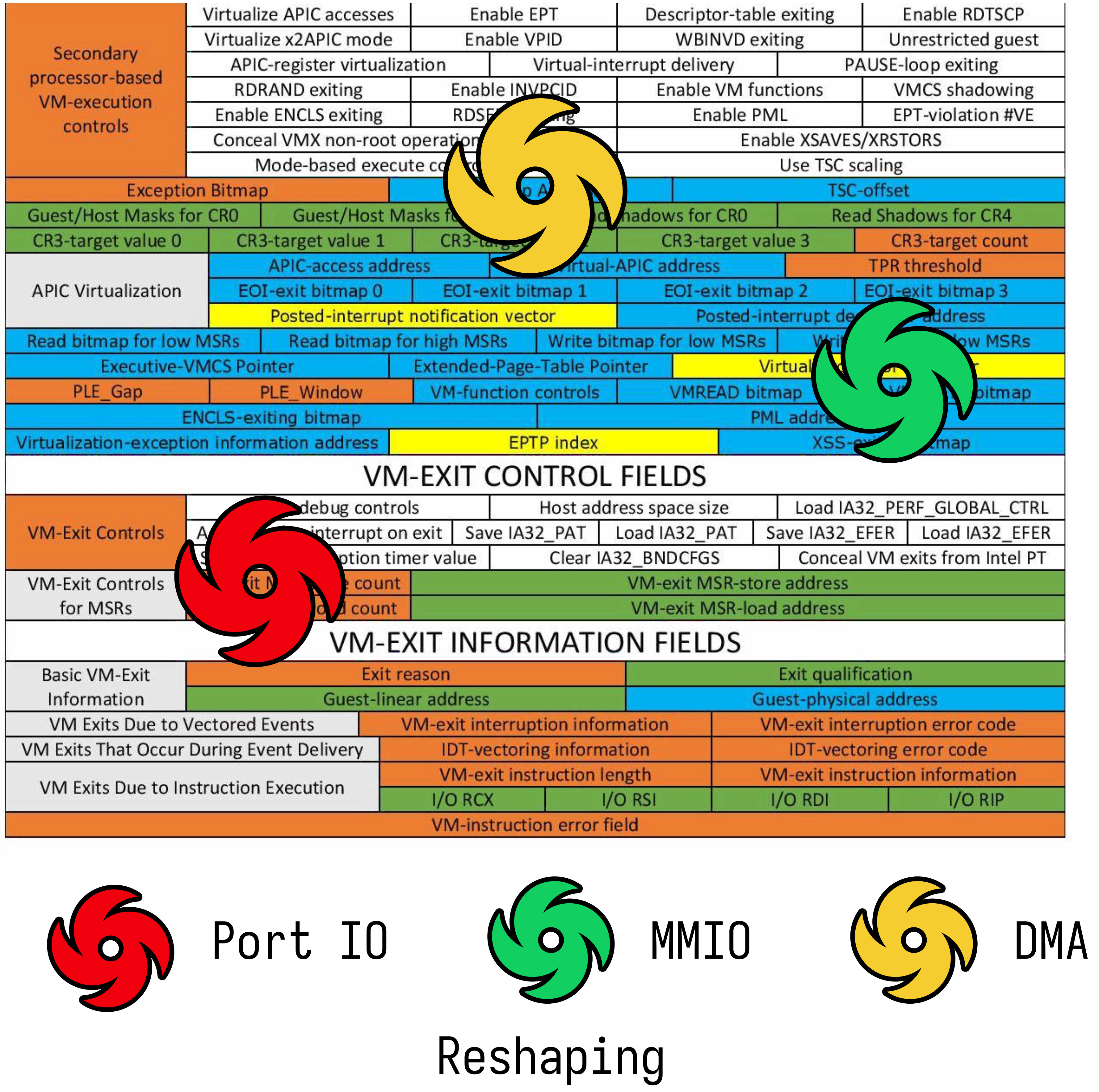
Volume 3C: System Programming Guide, Part 3



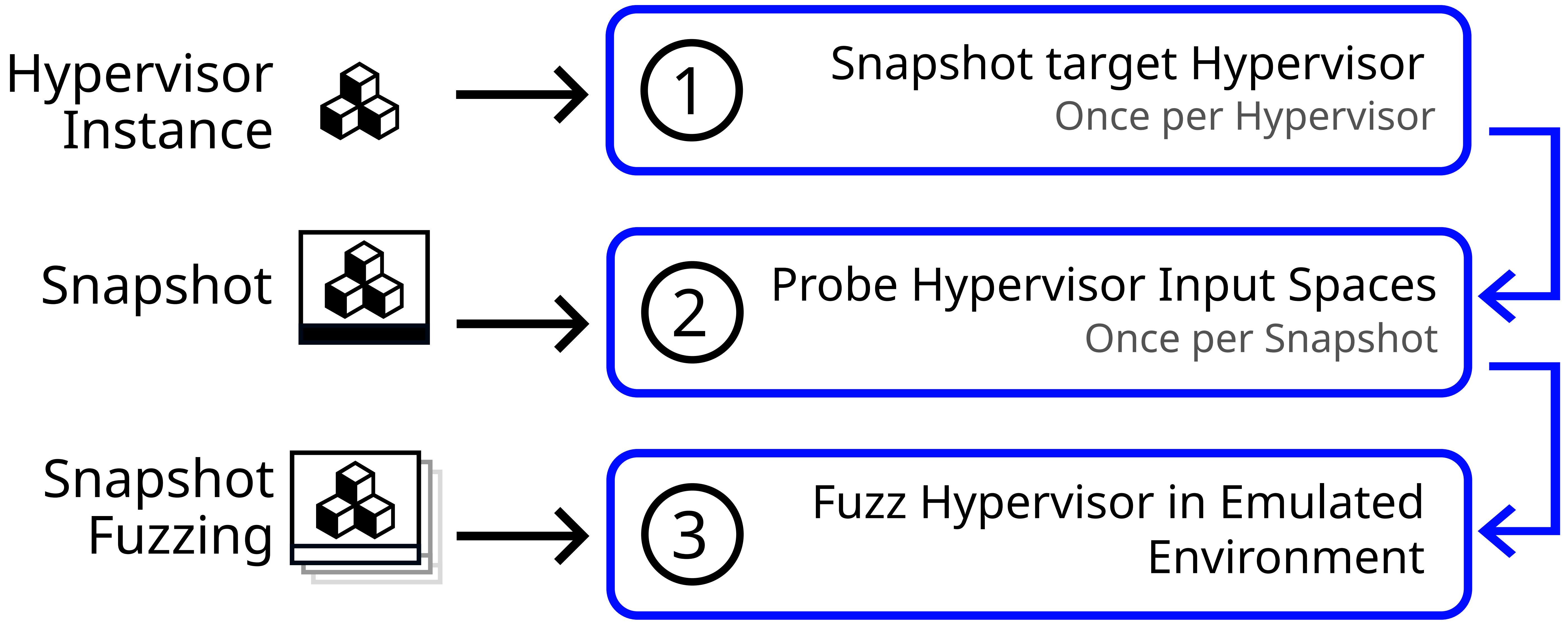
Firecracker · crosvm · Microsoft Hyper-V · Parallels · QEMU · vmware · Bhyve

Other Hypervisors





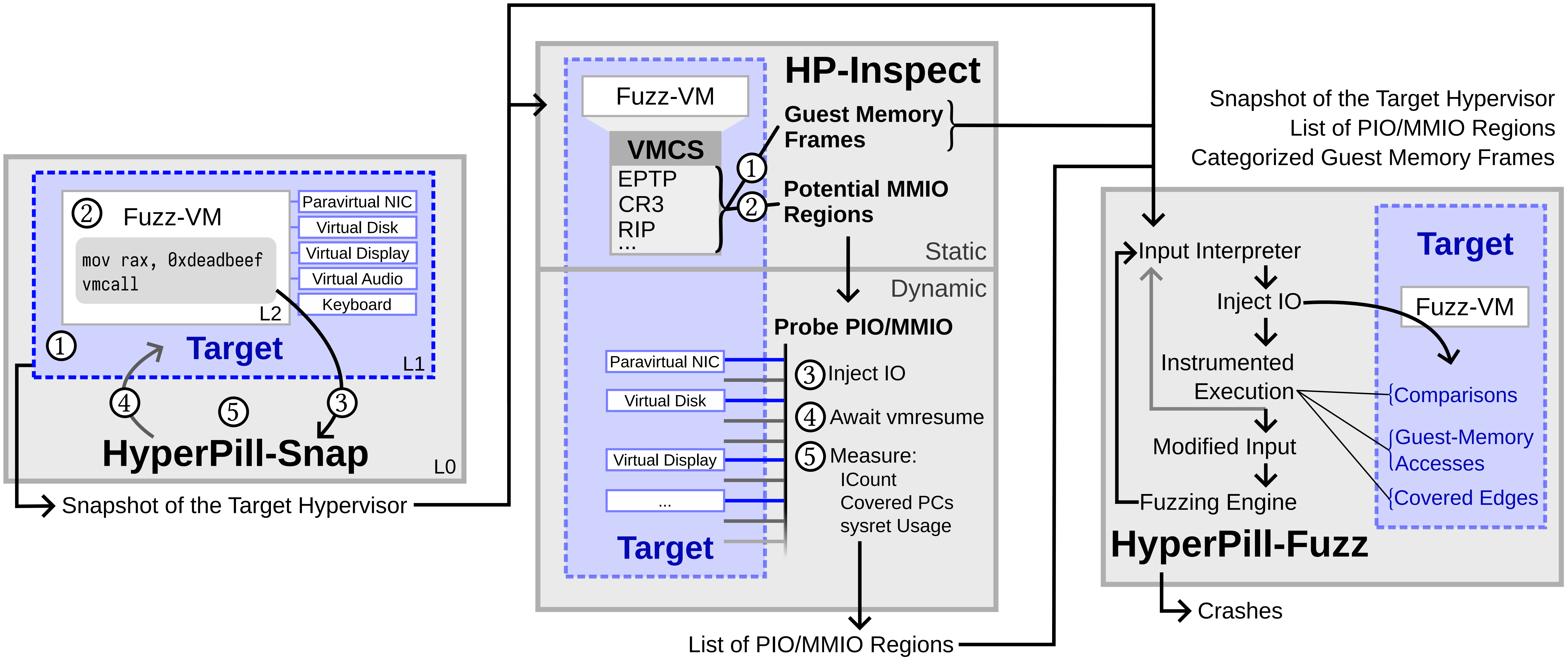
Overview


Hypervisor Instance
Snapshot
Snapshot Fuzzing
1
Snapshot target Hypervisor
Once per Hypervisor
2
Probe Hypervisor Input Spaces
Once per Snapshot
3
Fuzz Hypervisor in Emulated Environment

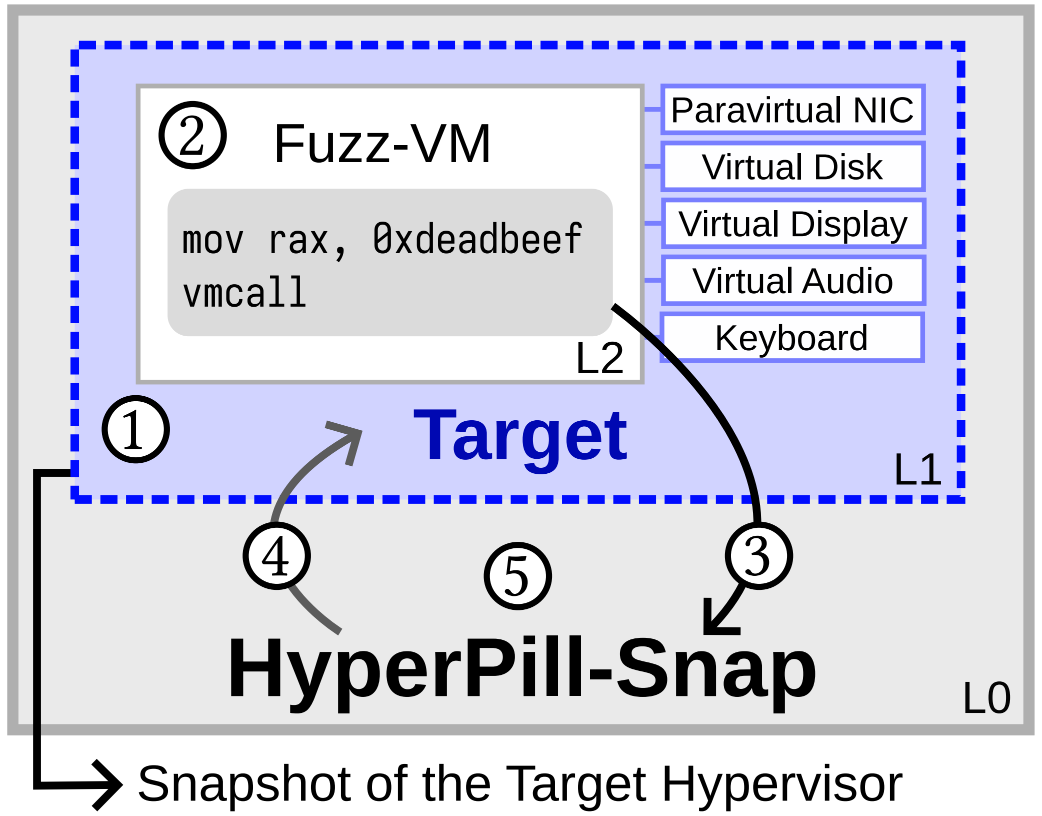
1 Snapshot target Hypervisor
Once per Hypervisor

2 Probe Hypervisor Input Spaces
Once per Snapshot

3 Fuzz Hypervisor in Emulated Environment

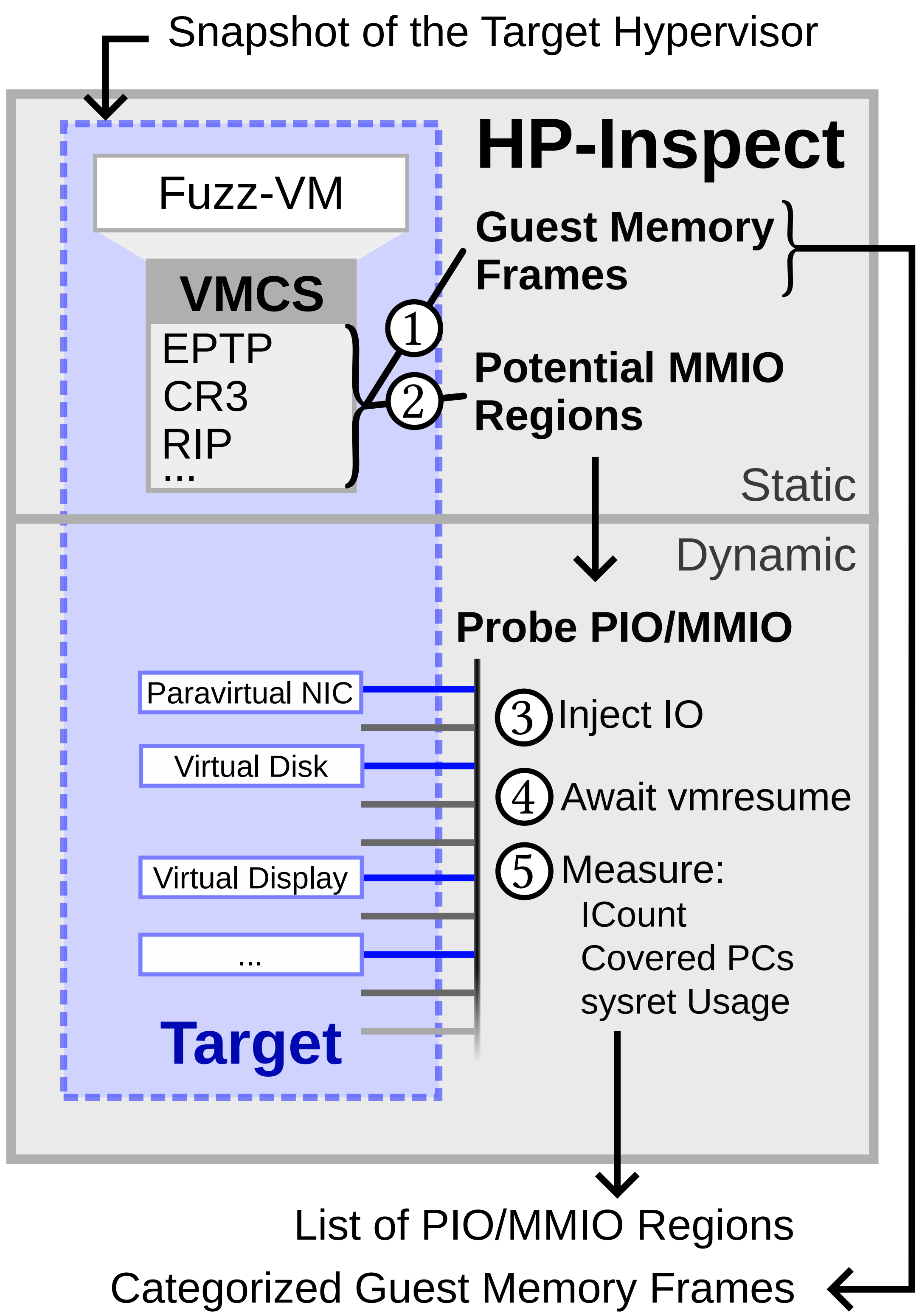


1. Make a Snapshot



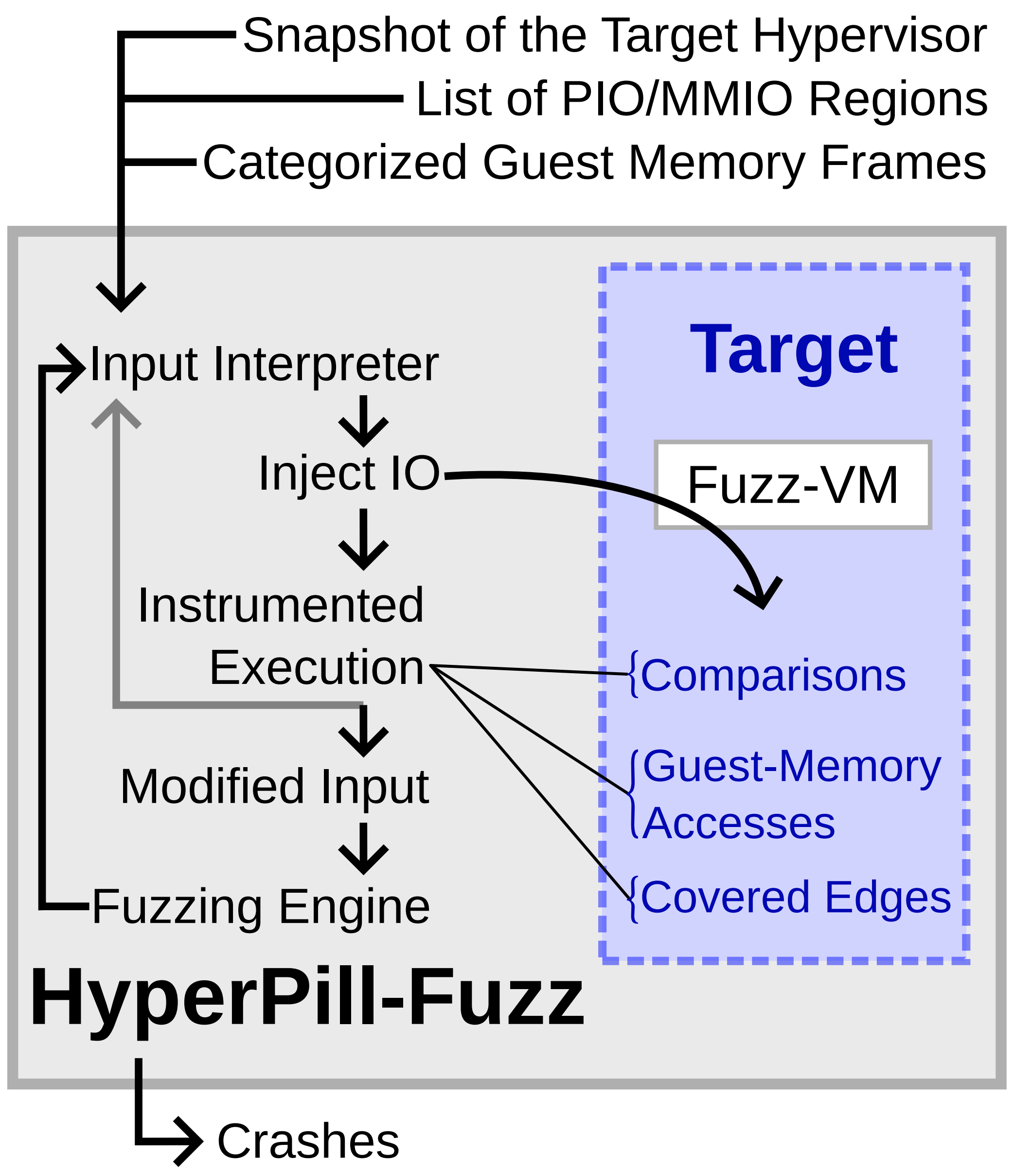
1. Run the target hypervisor (L1) nested in KVM (L0)
2. Configure/Start a VM in L1 (L2)
3. Invoke a special "snapshot" hypercall in (L2)
4. **Collect a memory/register snapshot of L1, just as it is about to handle the hypercall VM exit from L2**.

2. Enumerate the Input Spaces



1. Load the snapshot into an emulator (Bochs)
2. Inspect the VMCS that L1 created for L2 to identify **MMIO Regions** and physical **frames allocated to L2's memory**
3. Perform probing of IO input-space to identify **active ports** by tracking icounts, covered PCs and exits to userspace.

3. Fuzz the Hypervisor



1. Load the snapshot into the emulator
2. Modify the register/VMCS state to inject fuzzer-generated **PIO/MMIO**
3. Resume the hypervisor and wait for it to handle the VMExit.
4. Instead of running the L2 VM, immeditately inject another fuzzer-provided VMexit.
5. When the hypervisor reads from L2's memory, fill the read with fuzzer-provided data. (**DMA**)
6. Once the whole input is executed, reload the snapshot

Results: **QEMU Coverage**

	Morphuzz		ViDeZZo		HYPERPILL		
	12 Cores 24 Hours						
Device	Branch Coverage (Executions/Second)						Bug
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: **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

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

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

No Grammar, No Problem: Towards Fuzzing the Linux Kernel without System-Call Descriptions

Alexander Bulekov[†][*] alxndr@bu.edu

Bandan Das[*] bsd@redhat.com

Stefan Hajnoczi[*] stefanha@redhat.com

Manuel Egele[†] megele@bu.edu

[†]Boston University [*]Red Hat

Abstract—The integrity of the entire computing ecosystem depends on the security of our operating systems (OSes). Unfortunately, due to the scale and complexity of OS code, hundreds of security issues are found in OSes, every year [32]. As such, operating systems have constantly been prime use-cases for applying security-analysis tools. In recent years, fuzz-testing has appeared as the dominant technique for automatically finding security issues in software. As such, fuzzing has been adapted to find thousands of bugs in kernels [14]. However, modern OS fuzzers, such as Syzkaller, rely on precise, extensive, manually-created harnesses and grammars for each interface fuzzed within the kernel. Due to this reliance on grammars, current OS fuzzers are faced with scaling-issues.

In this paper, we present FuzzNG, our generic approach to fuzzing system-calls on OSes. Unlike Syzkaller, FuzzNG does not require intricate descriptions of system-call interfaces in order to function. Instead FuzzNG leverages fundamental kernel design features in order to reshape and simplify the fuzzer's input-space. As such FuzzNG only requires a small config, for each new target: essentially a list of files and system-call numbers the fuzzer should explore.

We implemented FuzzNG for the Linux kernel. Testing FuzzNG over 10 Linux components with extensive descriptions in Syzkaller showed that, on average, FuzzNG achieves 102.5% of Syzkaller's coverage. FuzzNG found 9 new bugs (5 in components that Syzkaller had already fuzzed extensively, for years). Additionally, FuzzNG's lightweight configs are less than 1.7% the size of Syzkaller's manually-written grammars. Crucially, FuzzNG achieves this without initial seed-inputs, or expert guidance.

I. Introduction

The Operating System continues to serve as one of the most security-critical building blocks in modern computing. The OS' role in managing resources and enforcing isolation between applications makes it a target for attackers who seek to violate OS-provided guarantees. Recognizing the critical nature of OS security, fuzzers have identified and helped fix thousands of bugs in OS kernels. Recently, the success of OS fuzzers has emphasized difficulty of writing secure low-level code, and has even spurred initiatives such as support for safer languages in the Linux kernel, and the usage of hardware-features such as Memory Tagging to enable advanced low-overhead defenses against memory-corruption issues [23], [44]. Most OS fuzzers focus on the critical system-call interface, which enables user-space applications to request services from the kernel.

Syzkaller[14], the most prolific system-call fuzzer, has become an integral component of the Linux Kernel development lifecycle, with over 2,700 mentions in kernel commit messages. As such, syzkaller, itself, has grown to be a sizeable project, with over 200 contributors. Crucially, Syzkaller can only fuzz system-calls that are sufficiently described by a "syzlang" grammar. These grammars encode and annotate the types of resources provided as inputs and returned as outputs, by system-calls. Therefore, much of the syzkaller community's work is focused around developing and refining "syzlang" descriptions for system-calls, which are essential to Syzkaller's success.

Developing such grammars is a manual process, and requires detailed knowledge about the interface (i.e., set of system calls) in question. As such, grammars are prone to human-error, and can lead to gaps in coverage, or over-fitting (preventing the fuzzer from exploring all states and scenarios in which code could be covered). Additionally, syzkaller sometimes requires writing supplementary harnessing code to fuzz particularly complex interfaces. For example, to fuzz the Linux Kernel Virtual Machine (KVM) interface, which powers security-critical virtualization software, Syzkaller developers committed 891 lines of detailed syscall descriptions, 243 KVM-related constants, and a further 879 lines of KVM-specific C harnessing code (illustrated in Figure 1). Even though Syzkaller features tens-of-thousands of hand-crafted "syzlang" rules, the current process cannot scale to fuzz the millions of lines of code added to the Linux Kernel each year [33]. Academic works have recognized Syzkaller's scalability problem with manually-created syzlang grammars, and have focused on automatically generating grammars. Works such as Difuze, IMF, SyzGen, and KSG apply static and dynamic-analysis techniques to automatically generate system-call descriptions [12], [18], [9], [51]. Difuze, IMF and SyzGen are designed and evaluated against interfaces, such as Android Drivers, and macOS APIs, for which no ... line manual-descriptions exist. KSG's descripti... improve Syzkaller's coverage, howe... been released and upstrea... efforts continue t... Some ...

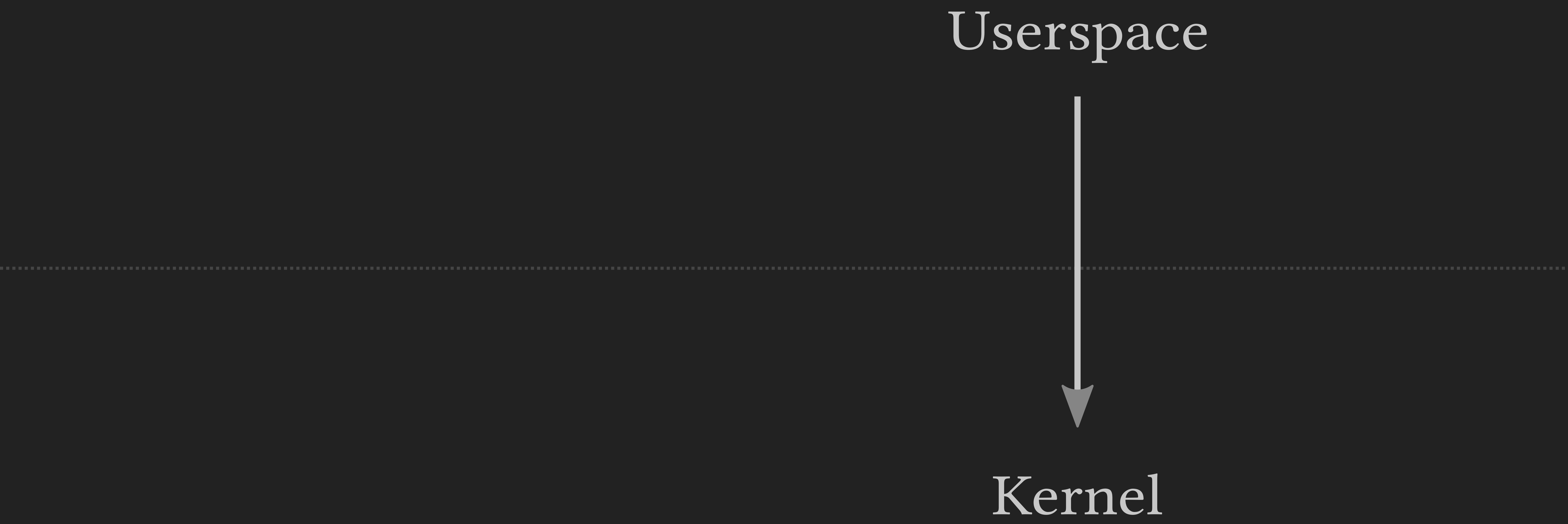
...ork and Distributed System Security (NDSS) Symposium 2023
...ruary - 3 March 2023, San Diego, CA, USA
...-891562-83-5
...x.doi.org/10.14722/ndss.2023.24688
...ss-symposium.org

FuzzNG

No Grammar, No Problem: Towards Fuzzing the Linux Kernel without System-Call Descriptions

NDSS 2023

Fuzzing System Calls



Fuzzing System Calls

Fuzzing System Calls

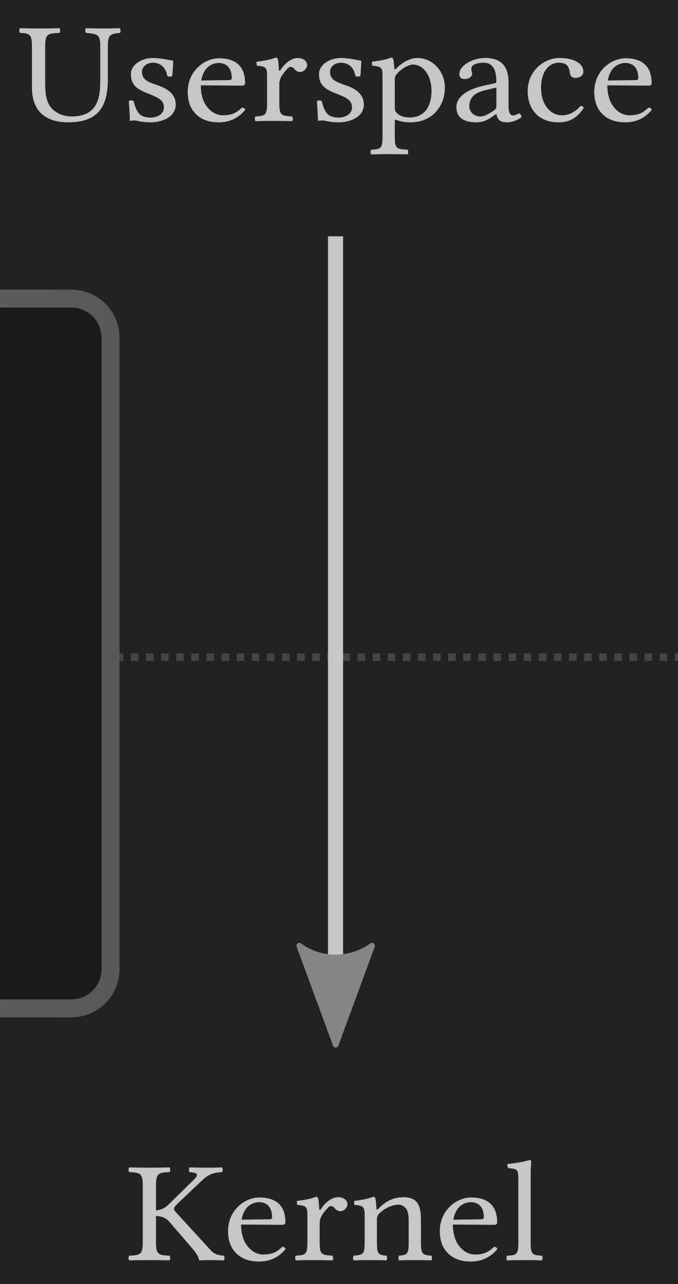
```
Adding a New System Call
----------------------------------------

For more sophisticated system calls that involve a larger number of arguments,
it's preferred to encapsulate the majority of the arguments into a structure
that is passed in by pointer.  Such a structure can cope with future extension
by including a size argument in the structure

...

If your new system call allows userspace to refer to a kernel object, it
should use a file descriptor as the handle for that object -- don't invent a
new type of userspace object handle when the kernel already has mechanisms and
well-defined semantics for using file descriptors.
```

linux-kernel/Documentation/process/adding-syscalls.rst

Fuzzing System Calls

Fuzzing System Calls

```
fd = open("/var/log/messages", O_RDONLY)
read(fd, buf, 100);
```



Fuzzing System Calls

Userspace

```
fd = open("/var/log/messages", O_RDONLY)
read(fd, buf, 100);

syscall 0x2 0xce51020 0x0 0x0 0x0 0x0
syscall 0x0     0x55a4e3c2a000 100 0x0 0x0
```

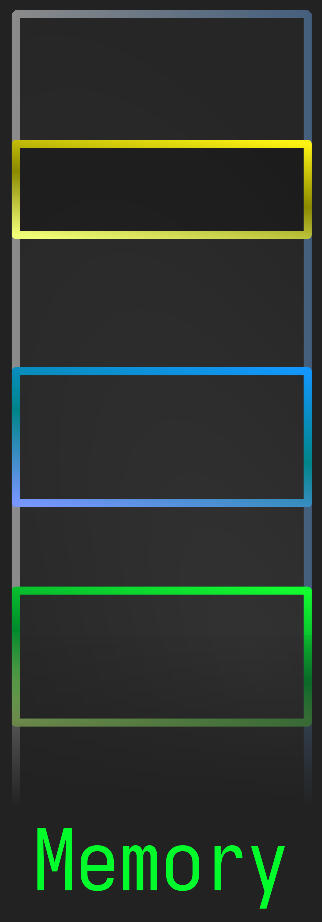
Kernel

Fuzzing System Calls

```
fd = open("/var/log/messages", O_RDONLY)
read(fd, buf, 100);

syscall 0x2 0xce51020 0x0 0x0 0x0 0x0
syscall 0x0     0x55a4e3c2a000 100 0x0 0x0
```

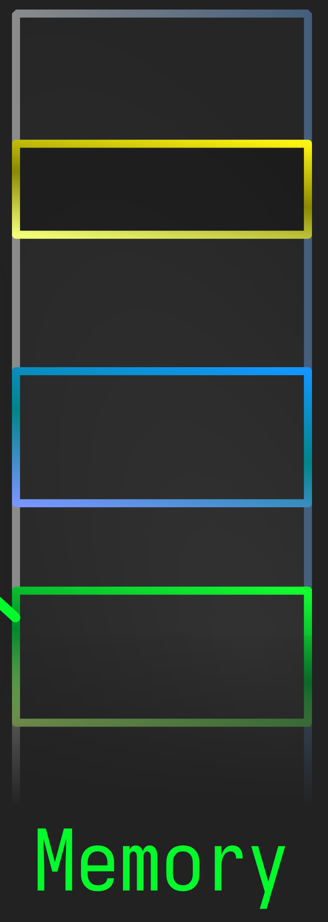
Userspace

Kernel



Fuzzing System Calls

```
0ce51010 5548 89e5 9090 5dc3 3030 3030 3030 0a00  UH....].000000..
0ce51020 2f76 6172 2f6c 6f67 2f6d 6573 7361 6765  /var/log/message
0ce51030 7300 4572 726f 7220 6f70 656e 696e 6720  s.Error opening
0ce51040 6669 6c65 2e00 5265 6164 696e 6720 7468  file..Reading th
```



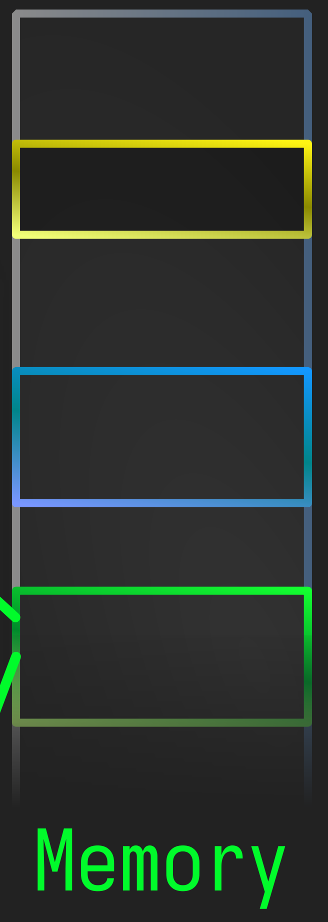

```
fd = open("/var/log/messages", O_RDONLY)
read(fd, buf, 100);

syscall 0x2 0xce51020 0x0 0x0 0x0 0x0
syscall 0x0     0x55a4e3c2a000 100 0x0 0x0
```

Kernel

Fuzzing System Calls

```
0ce51010 5548 89e5 9090 5dc3 3030 3030 3030 0a00  UH....].000000..
0ce51020 2f76 6172 2f6c 6f67 2f6d 6573 7361 6765  /var/log/message
0ce51030 7300 4572 726f 7220 6f70 656e 696e 6720  s.Error opening
0ce51040 6669 6c65 2e00 5265 6164 696e 6720 7468  file..Reading th
```

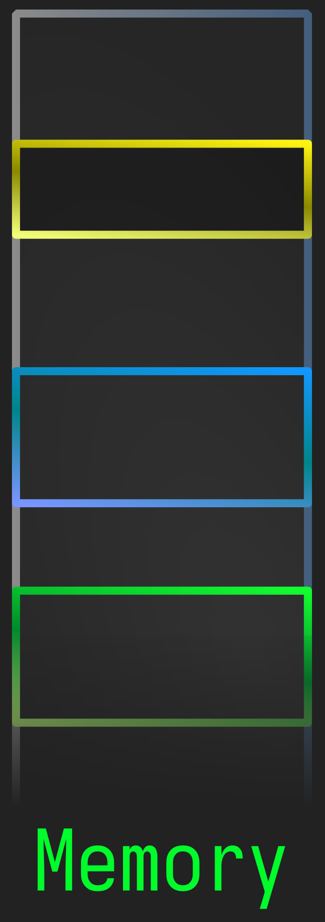
```
fd = open("/var/log/messages", O_RDONLY)
read(fd, buf, 100);

syscall 0x2 0xce51020 0x0 0x0 0x0 0x0
syscall 0x0     0x55a4e3c2a000 100 0x0 0x0
```



Kernel

Fuzzing System Calls

Userspace

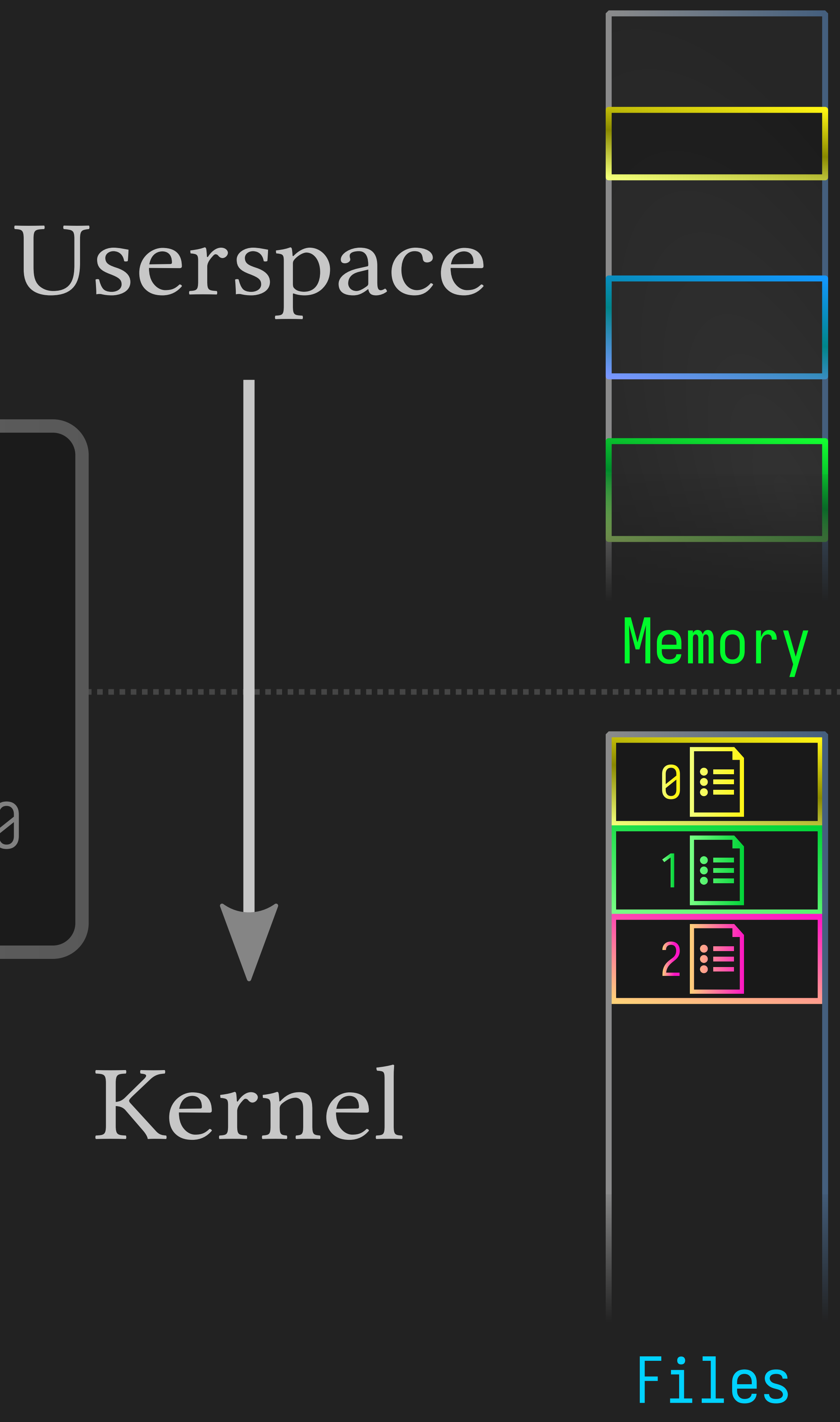



```
fd = open("/var/log/messages", O_RDONLY)
read(fd, buf, 100);

syscall 0x2 0xce51020 0x0 0x0 0x0 0x0
syscall 0x0     0x55a4e3c2a000 100 0x0 0x0
```

Kernel

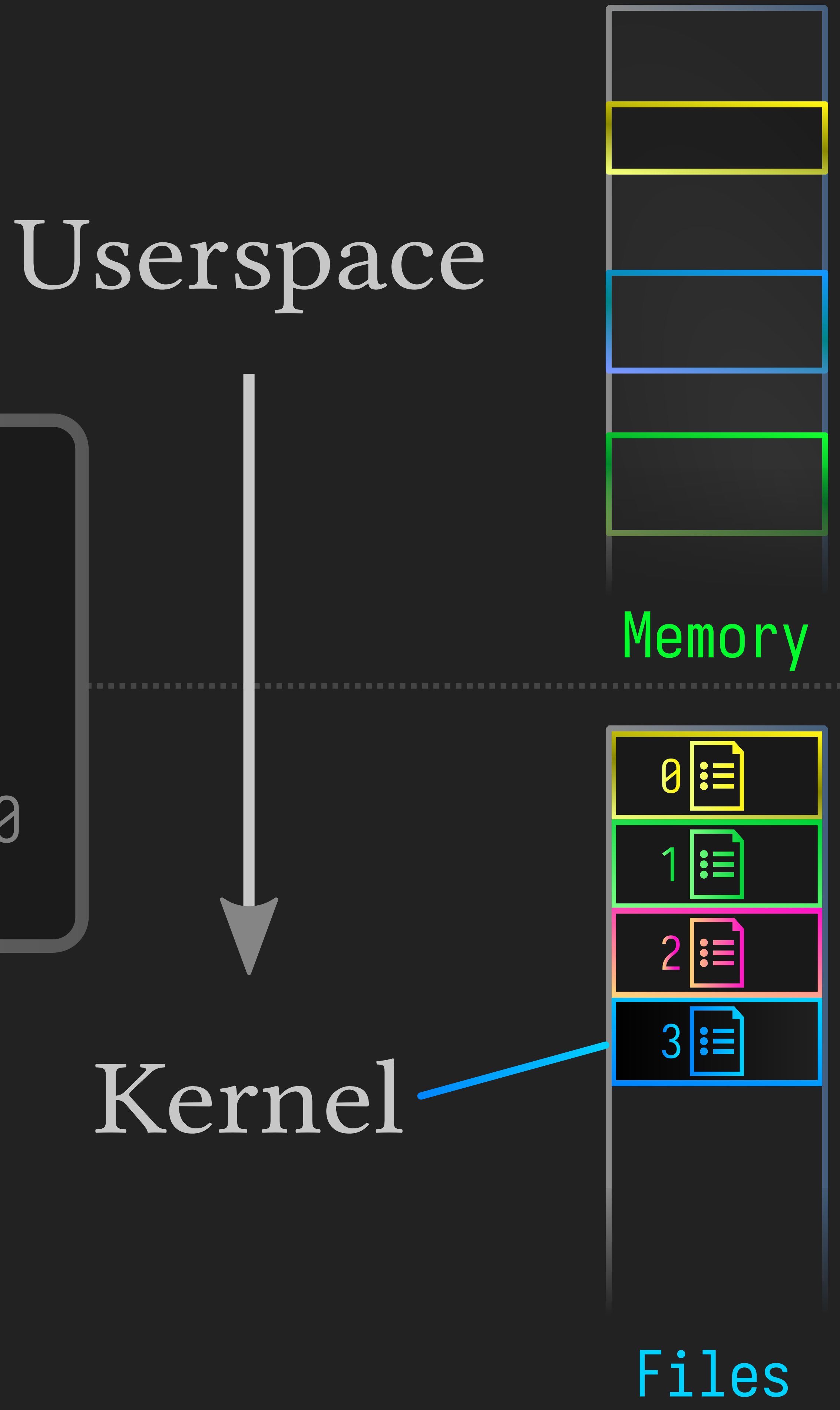
Fuzzing System Calls

```
fd = open("/var/log/messages", O_RDONLY)
read(fd, buf, 100);

syscall 0x2 0xce51020 0x0 0x0 0x0 0x0
syscall 0x0     0x55a4e3c2a000 100 0x0 0x0
```



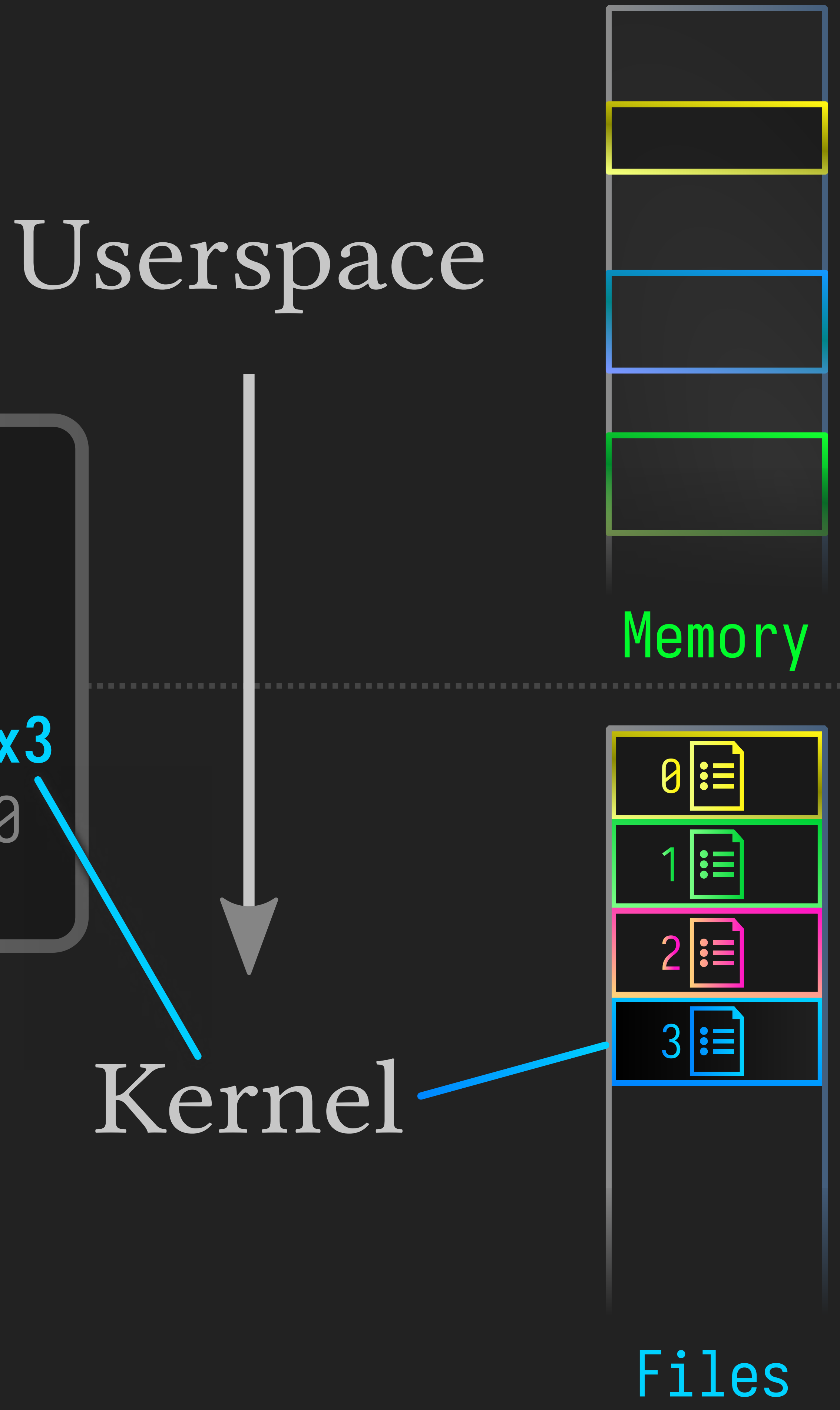
Fuzzing System Calls

```
fd = open("/var/log/messages", O_RDONLY)
read(fd, buf, 100);

syscall 0x2 0xce51020 0x0 0x0 0x0 0x0
syscall 0x0     0x55a4e3c2a000 100 0x0 0x0
```



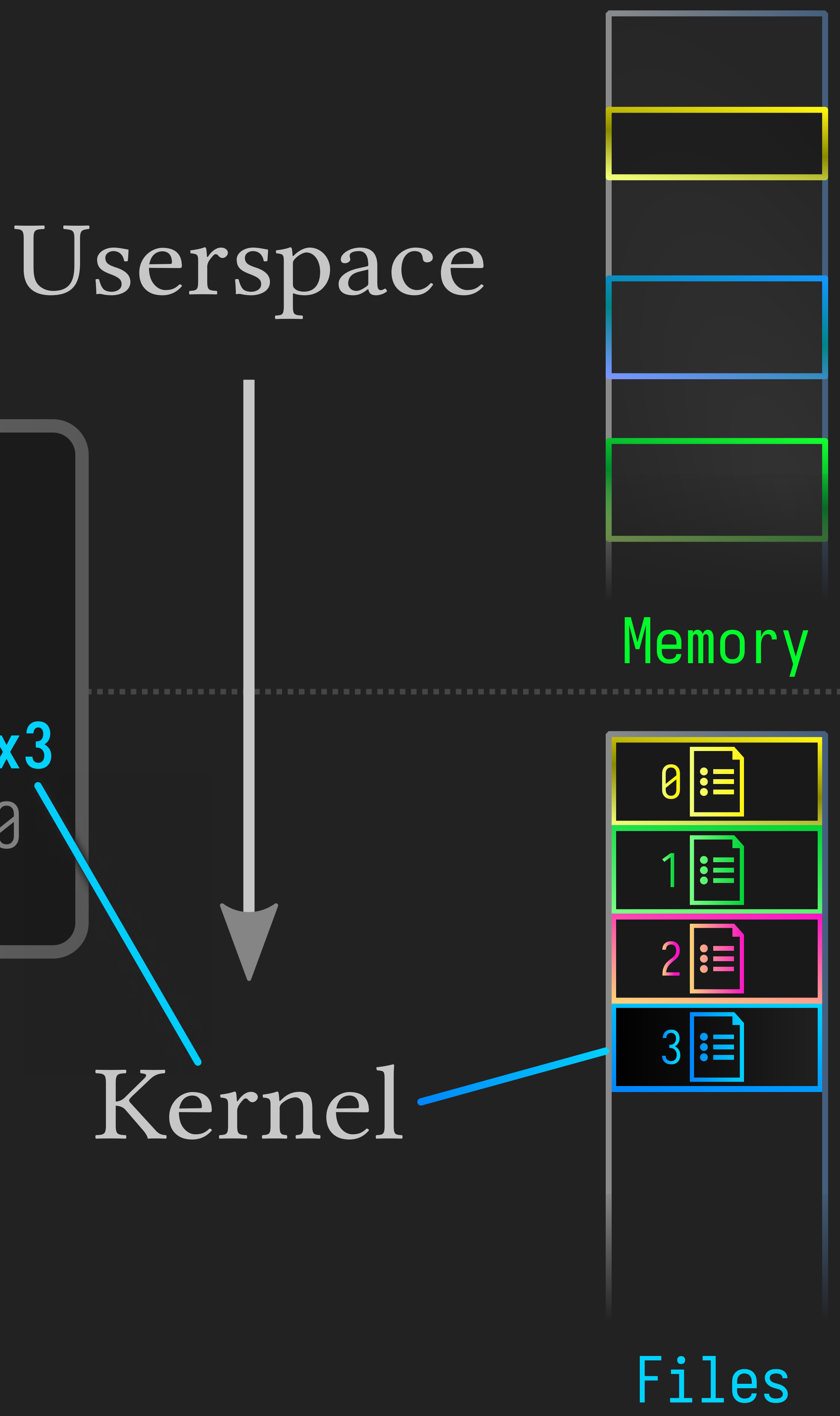
Fuzzing System Calls

```
fd = open("/var/log/messages", O_RDONLY)
read(fd, buf, 100);

syscall 0x2 0xce51020 0x0 0x0 0x0 0x0 = 0x3
syscall 0x0     0x55a4e3c2a000 100 0x0 0x0
```

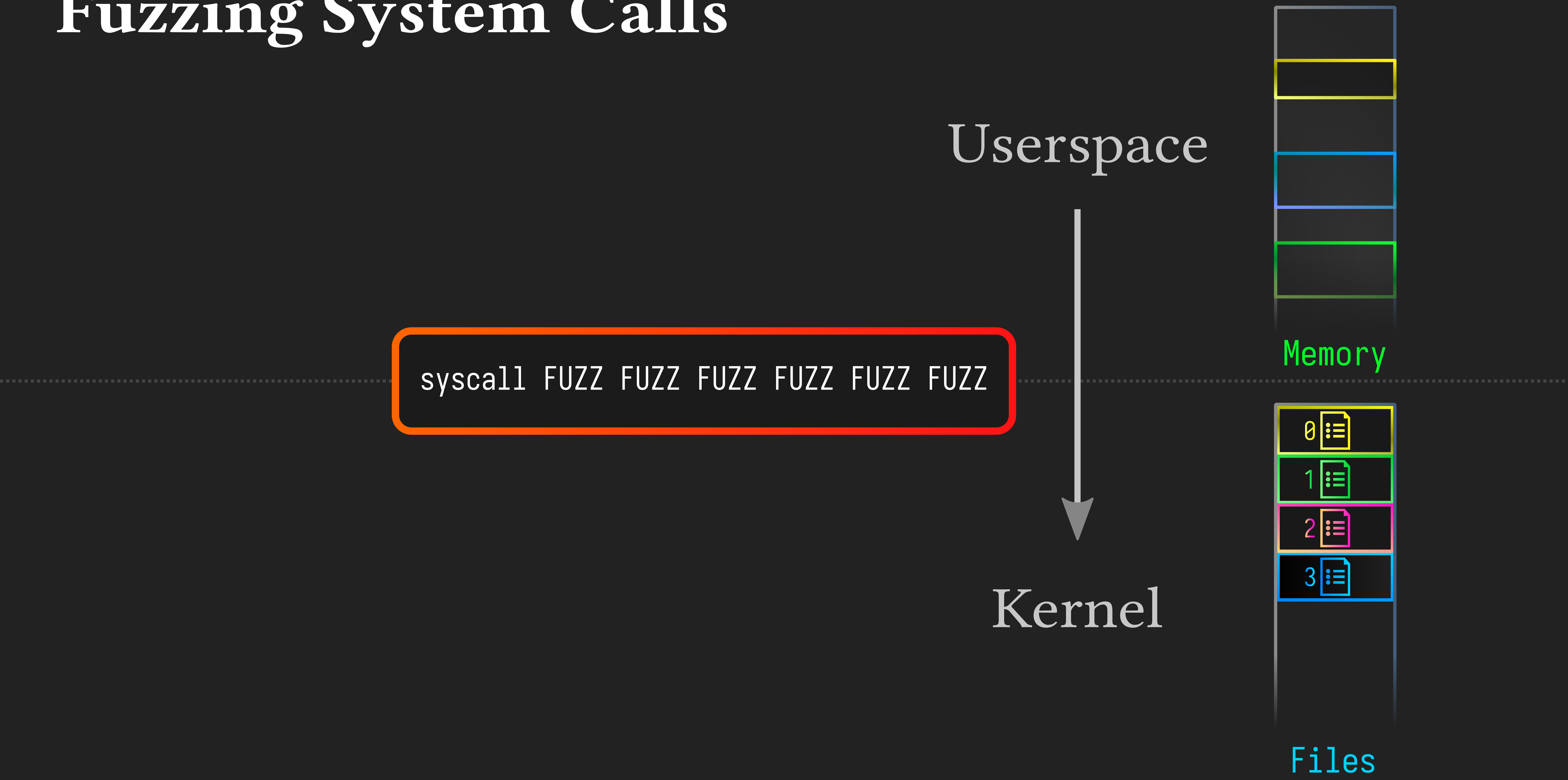


Fuzzing System Calls

```
fd = open("/var/log/messages", O_RDONLY)
read(fd, buf, 100);

syscall 0x2 0xce51020 0x0 0x0 0x0 0x0 = 0x3
syscall 0x0 0x3 0x55a4e3c2a000 100 0x0 0x0
```



Fuzzing System Calls



Fuzzing System Calls

Userspace

`syscall FUZZ FUZZ FUZZ FUZZ FUZZ FUZZ`

Pointers and **File-Descriptors**
result in an **enormous** system-call input-space

System-Call Grammars

```
syz_io_uring_setup(entries int32[1:IORING_MAX_ENTRIES], params ptr[inout, io_uring_params], addr_ring vma, addr_sqes vma, ring_ptr ptr[out, ring_ptr], sqes_ptr ptr[out, sqes_ptr]) fd_io_uring

io_uring_setup(entries int32[1:IORING_MAX_ENTRIES], params ptr[inout, io_uring_params]) fd_io_uring
io_uring_enter(fd fd_io_uring, to_submit int32[0:IORING_MAX_ENTRIES], min_complete int32[0:IORING_MAX_CQ_ENTRIES], flags flags[io_uring_enter_flags], sigmask ptr[in, sigset_t], size len[sigmask])
io_uring_register$IORING_REGISTER_BUFFERS(fd fd_io_uring, opcode const[IORING_REGISTER_BUFFERS], arg ptr[in, array[iovec_out]], nr_args len[arg])
io_uring_register$IORING_UNREGISTER_BUFFERS(fd fd_io_uring, opcode const[IORING_UNREGISTER_BUFFERS], arg const[0], nr_args const[0])
io_uring_register$IORING_REGISTER_FILES(fd fd_io_uring, opcode const[IORING_REGISTER_FILES], arg ptr[in, array[fd]], nr_args len[arg])
io_uring_register$IORING_UNREGISTER_FILES(fd fd_io_uring, opcode const[IORING_UNREGISTER_FILES], arg const[0], nr_args const[0])
io_uring_register$IORING_REGISTER_EVENTFD(fd fd_io_uring, opcode const[IORING_REGISTER_EVENTFD], arg ptr[in, fd_event], nr_args const[1])
io_uring_register$IORING_UNREGISTER_EVENTFD(fd fd_io_uring, opcode const[IORING_UNREGISTER_EVENTFD], arg const[0], nr_args const[0])
io_uring_register$IORING_REGISTER_FILES_UPDATE(fd fd_io_uring, opcode const[IORING_REGISTER_FILES_UPDATE], arg ptr[in, io_uring_files_update], nr_args len[arg:fds])
io_uring_register$IORING_REGISTER_EVENTFD_ASYNC(fd fd_io_uring, opcode const[IORING_REGISTER_EVENTFD_ASYNC], arg ptr[in, fd_event], nr_args const[1])
io_uring_register$IORING_REGISTER_PROBE(fd fd_io_uring, opcode const[IORING_REGISTER_PROBE], arg ptr[inout, io_uring_probe], nr_args len[arg:ops])
io_uring_register$IORING_REGISTER_PERSONALITY(fd fd_io_uring, opcode const[IORING_REGISTER_PERSONALITY], arg const[0], nr_args const[0]) ioring_personality_id
io_uring_register$IORING_UNREGISTER_PERSONALITY(fd fd_io_uring, opcode const[IORING_UNREGISTER_PERSONALITY], arg const[0], nr_args ioring_personality_id)

# The mmap'ed area for SQ and CQ rings are really the same -- the difference is
# accounted for with the usage of offsets.
mmap$IORING_OFF_SQ_RING(addr vma, len len[addr], prot flags[mmap_prot], flags flags[mmap_flags], fd fd_io_uring, offset const[IORING_OFF_SQ_RING]) ring_ptr
mmap$IORING_OFF_CQ_RING(addr vma, len len[addr], prot flags[mmap_prot], flags flags[mmap_flags], fd fd_io_uring, offset const[IORING_OFF_CQ_RING]) ring_ptr
mmap$IORING_OFF_SQES(addr vma, len len[addr], prot flags[mmap_prot], flags flags[mmap_flags], fd fd_io_uring, offset const[IORING_OFF_SQES]) sqes_ptr

# If no flags are specified(0), the io_uring instance is setup for interrupt driven IO.
io_uring_setup_flags = 0, IORING_SETUP_IOPOLL, IORING_SETUP_SQPOLL, IORING_SETUP_SQ_AFF, IORING_SETUP_CQSIZE, IORING_SETUP_CLAMP, IORING_SETUP_ATTACH_WQ
io_uring_enter_flags = IORING_ENTER_GETEVENTS, IORING_ENTER_SQ_WAKEUP
_ = __NR_mmap2

# Once an io_uring is set up by calling io_uring_setup, the offsets to the member fields
# to be used on the mmap'ed area are set in structs io_sqring_offsets and io_cqring_offsets.
# Except io_sqring_offsets.array, the offsets are static while all depend on how struct io_rings
# is organized in code. The offsets can be marked as resources in syzkaller descriptions but
# this makes it difficult to generate correct programs by the fuzzer. Thus, the offsets are
# hard-coded here (and in the executor).
define SQ_HEAD_OFFSET 0
define SQ_TAIL_OFFSET 64
define SQ_RING_MASK_OFFSET 256
define SQ_RING_ENTRIES_OFFSET 264
define SQ_FLAGS_OFFSET 276
```

System-Call Grammars

```
io_uring_setup(entries int32[1:IORING_MAX_ENTRIES],
               params ptr[inout, io_uring_params]) fd_io_uring
```

System-Call Grammars

```
io_uring_setup(entries int32[1:IORING_MAX_ENTRIES],
               params ptr[inout, io_uring_params]) fd_io_uring

io_uring_register$IORING_REGISTER_PROBE(fd fd_io_uring,
                                        opcode const[IORING_REGISTER_PROBE],
                                        arg ptr[inout, io_uring_probe], nr_args len[arg:ops])
```

System-Call Grammars

```
io_uring_setup(entries int32[1:IORING_MAX_ENTRIES],
               params ptr[inout, io_uring_params]) fd_io_uring

io_uring_register$IORING_REGISTER_PROBE(fd fd_io_uring,
                                        opcode const[IORING_REGISTER_PROBE],
                                        arg ptr[inout, io_uring_probe], nr_args len[arg:ops])
io_uring_probe {
    last_op const[0, int8]
    ops_len const[0, int8]
    resv    const[0, int16]
    resv2   array[const[0, int32], 3]
    ops array[io_uring_probe_op, 0:IORING_OP_LAST]
}
io_uring_probe_op {
    op  const[0, int8]
    resv    const[0, int8]
    flags   const[0, int16]
    resv2   const[0, int32]
}
```

System-Call Grammars

io_uring_enter(2)   Linux Programmer's Manual   io_uring_enter(2)

NAME

io_uring_enter - initiate and/or complete asynchronous I/O

SYNOPSIS

```
#include <liburing.h>

int io_uring_enter(unsigned int fd, unsigned int to_submit,
                   unsigned int min_complete, unsigned int
flags,
                   sigset_t *sig);

int io_uring_enter2(unsigned int fd, unsigned int to_submit,
                    unsigned int min_complete, unsigned int
flags,
                    sigset_t *sig, size_t sz);
```

DESCRIPTION

io_uring_enter(2) is used to initiate and complete I/O using the shared submission and completion queues setup by a call to io_uring_setup(2). A single call can both submit new I/O and wait for completions of I/O initiated by this call or previous calls to io_uring_enter(2).

fd is the file descriptor returned by io_uring_setup(2). *to_submit* specifies the number of I/Os to submit from the submission queue. *flags* is a bitmask of the following values:

IORING_ENTER_GETEVENTS
If this flag is set, then the system call will wait for the specified number of events in *min_complete* before returning. This flag can be set along with *to_submit* to both submit and complete events in a single system call.

System-Call Grammars

```
io_uring_enter(2)    Linux Programmer's Manual    io_uring_enter(2)

NAME

  io_uring_enter - initiate and/or complete asynchronous I/O

SYNOPSIS

    #include <liburing.h>

    int io_uring_enter(unsigned int fd, unsigned int to_submit,
                       unsigned int min_complete, unsigned int
    flags,
                       sigset_t *sig);

    int io_uring_enter2(unsigned int fd, unsigned int to_submit,
                        unsigned int min_complete, unsigned int
    flags,
                        sigset_t *sig, size_t sz);

DESCRIPTION

  io_uring_enter(2) is used to initiate and complete I/O using the
  shared submission and completion queues setup by a call to
  io_uring_setup(2). A single call can both submit new I/O and
  wait for completions of I/O initiated by this call or previous
  calls to io_uring_enter(2).

  fd is the file descriptor returned by io_uring_setup(2).
  to_submit specifies the number of I/Os to submit from the
  submission queue. flags is a bitmask of the following values:

IORING_ENTER_GETEVENTS
         If this flag is set, then the system call will wait for
         the specified number of events in min_complete before
         returning. This flag can be set along with to_submit to
         both submit and complete events in a single system call.
```

```c
SYSCALL_DEFINE6(io_uring_enter, unsigned int, fd, u32, to_submit,
        u32, min_complete, u32, flags, const void __user *, argp,
        size_t, argsz)
{
    struct io_ring_ctx *ctx;
    long ret = -EBADF;
    int submitted = 0;
    struct fd f;

    io_run_task_work();

    if (flags & ~(IORING_ENTER_GETEVENTS | IORING_ENTER_SQ_WAKEUP |
            IORING_ENTER_SQ_WAIT | IORING_ENTER_EXT_ARG))
        return -EINVAL;

    f = fdget(fd);
    if (!f.file)
        return -EBADF;

    ret = -EOPNOTSUPP;
    if (f.file->f_op != &io_uring_fops)
        goto out_fput;

    ret = -ENXIO;
    ctx = f.file->private_data;
    if (!percpu_ref_tryget(&ctx->refs))
        goto out_fput;
```

System-Call Grammars

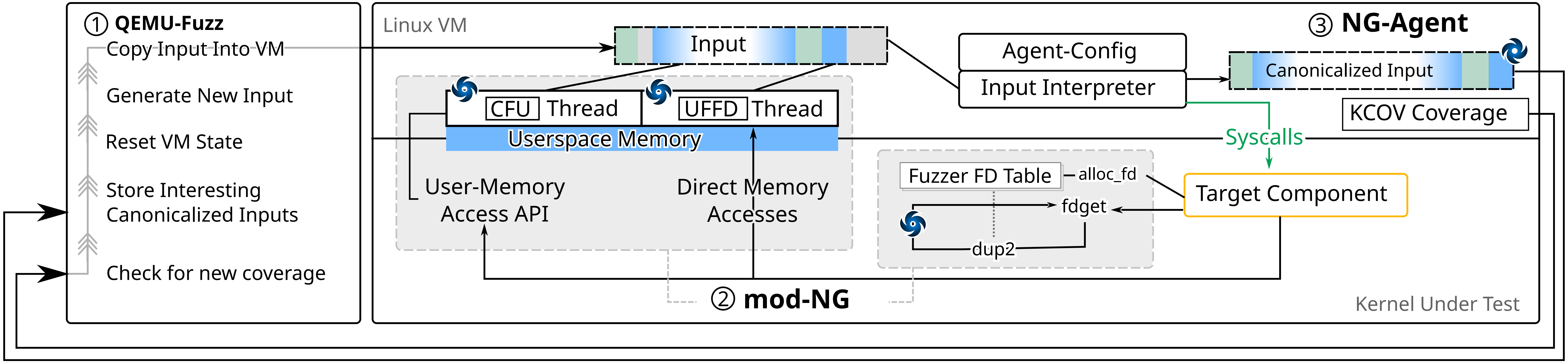
4000+ Lines of Code to Describe a Single Subsystem (KVM)

Current System-Call Fuzzers rely on detailed grammars to describe pointer and file-descriptor arguments

FuzzNG

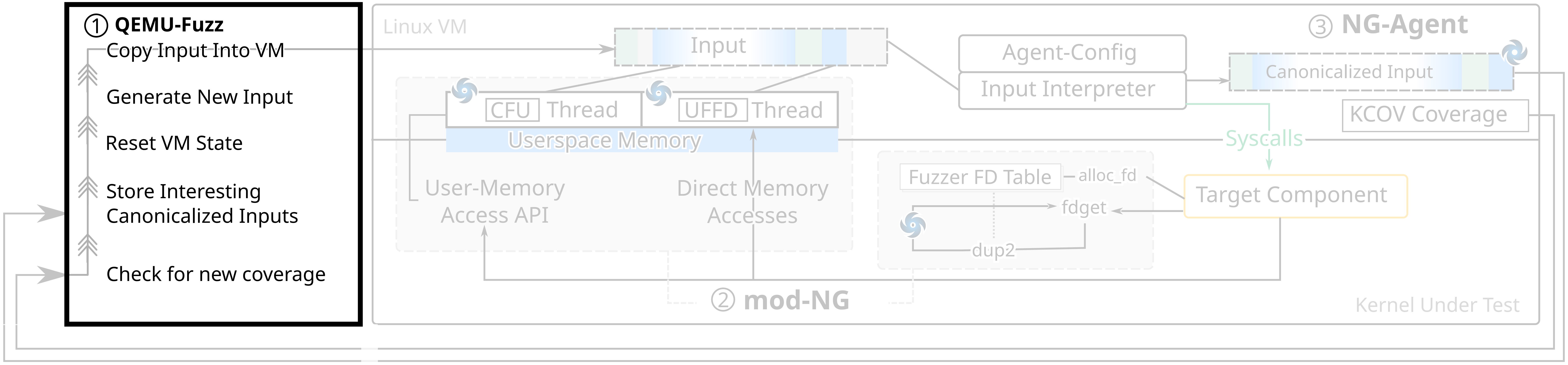
Reshape the pointer and file-descriptor input-spaces to make system-calls conducive to off-the-shelf fuzzing methods

FuzzNG



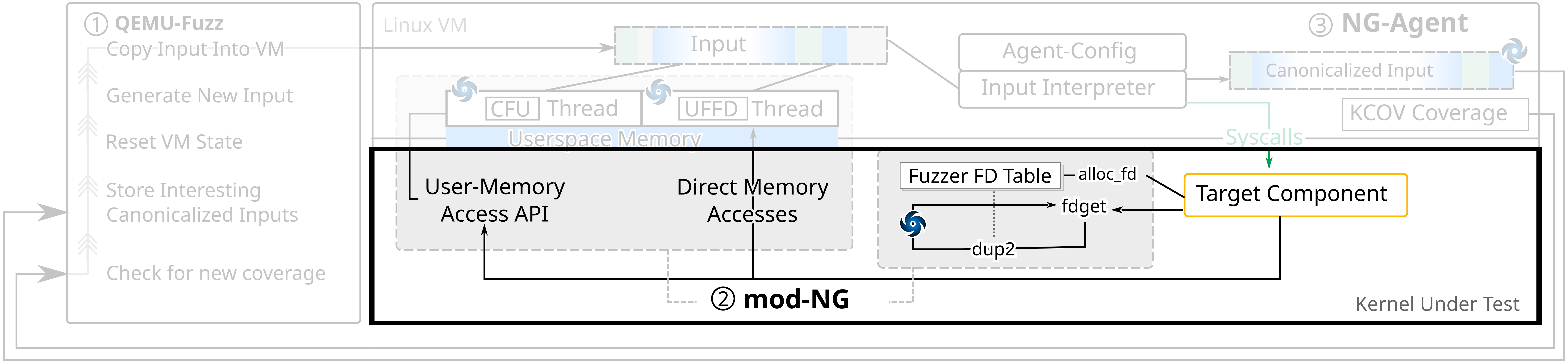
① QEMU-Fuzz
Copy Input Into VM
Generate New Input
Reset VM State
Store Interesting Canonicalized Inputs
Check for new coverage
Linux VM
Input
CFU Thread
UFFD Thread
Userspace Memory
User-Memory Access API
Direct Memory Accesses
Agent-Config
Input Interpreter
③ NG-Agent
Canonicalized Input
KCOV Coverage
Syscalls
Fuzzer FD Table
alloc_fd
fdget
dup2
Target Component
② mod-NG
Kernel Under Test

FuzzNG



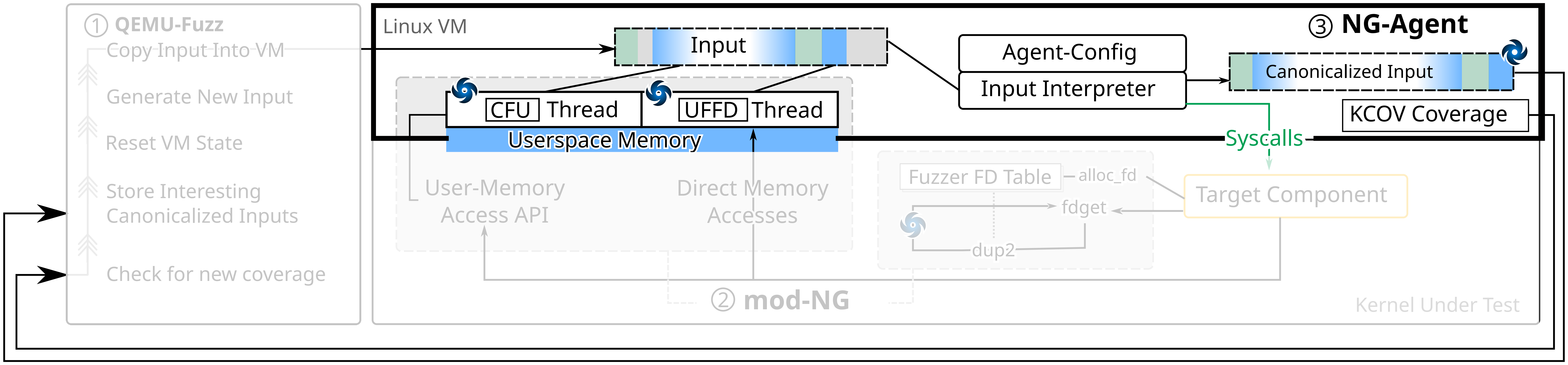
① QEMU-Fuzz
Copy Input Into VM
Generate New Input
Reset VM State
Store Interesting Canonicalized Inputs
Check for new coverage
Linux VM
Input
CFU Thread
UFFD Thread
Userspace Memory
User-Memory Access API
Direct Memory Accesses
② mod-NG
Agent-Config
Input Interpreter
③ NG-Agent
Canonicalized Input
KCOV Coverage
Syscalls
Fuzzer FD Table
alloc_fd
fdget
dup2
Target Component
Kernel Under Test

FuzzNG



① QEMU-Fuzz
Copy Input Into VM
Generate New Input
Reset VM State
Store Interesting Canonicalized Inputs
Check for new coverage
Linux VM
Input
CFU Thread
UFFD Thread
Userspace Memory
Agent-Config
Input Interpreter
③ NG-Agent
Canonicalized Input
KCOV Coverage
Syscalls
User-Memory Access API
Direct Memory Accesses
Fuzzer FD Table
alloc_fd
fdget
dup2
Target Component
② mod-NG
Kernel Under Test

FuzzNG



① QEMU-Fuzz
Copy Input Into VM
Generate New Input
Reset VM State
Store Interesting Canonicalized Inputs
Check for new coverage
Linux VM
Input
CFU Thread
UFFD Thread
Userspace Memory
Agent-Config
Input Interpreter
③ NG-Agent
Canonicalized Input
KCOV Coverage
Syscalls
User-Memory Access API
Direct Memory Accesses
Fuzzer FD Table
alloc_fd
fdget
dup2
Target Component
② mod-NG
Kernel Under Test

Reshaping the Pointer and File-Descriptor Input Spaces

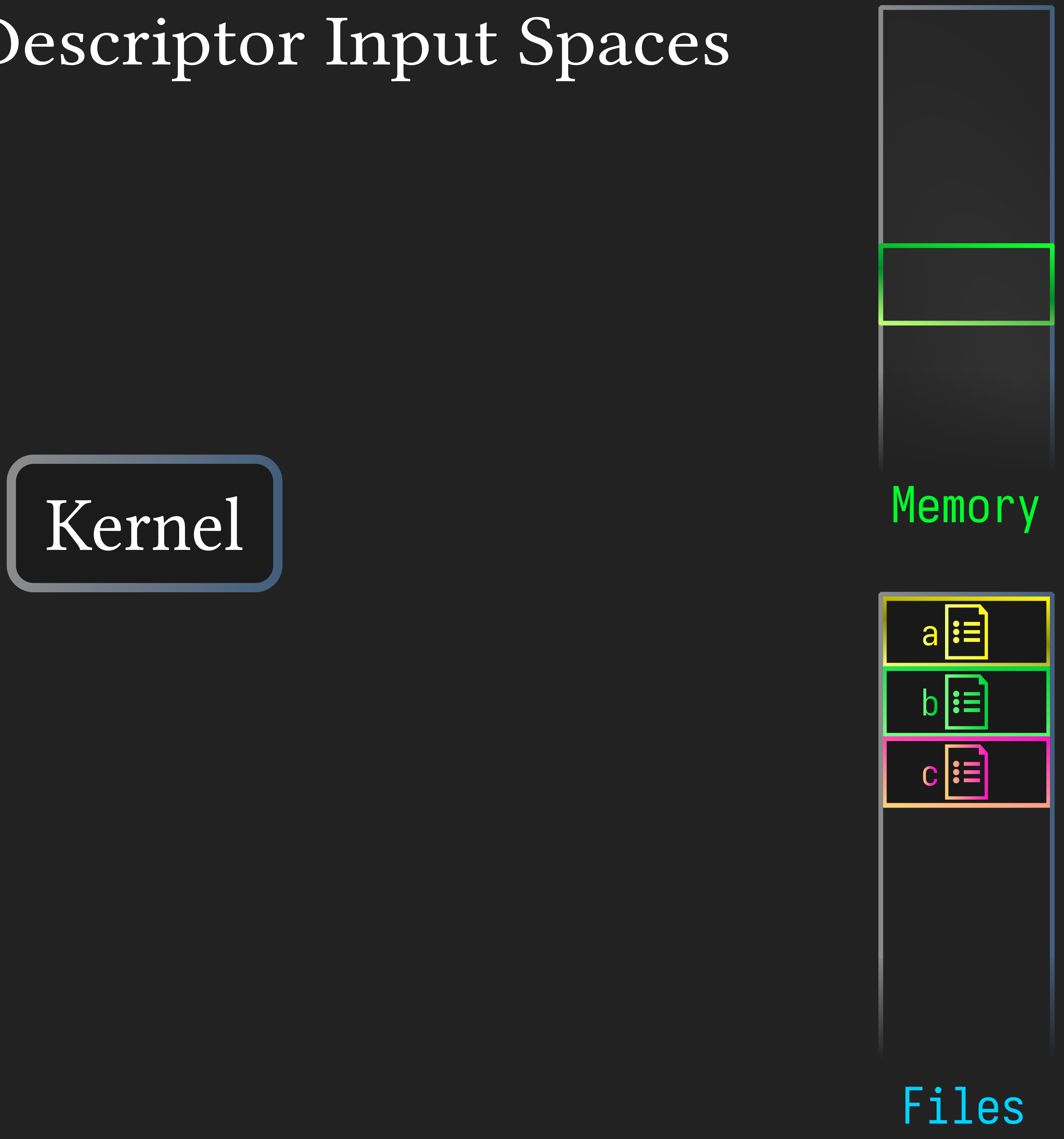
```
syscall FUZZ FUZZ FUZZ FUZZ FUZZ FUZZ
```

Reshaping the Pointer and File-Descriptor Input Spaces

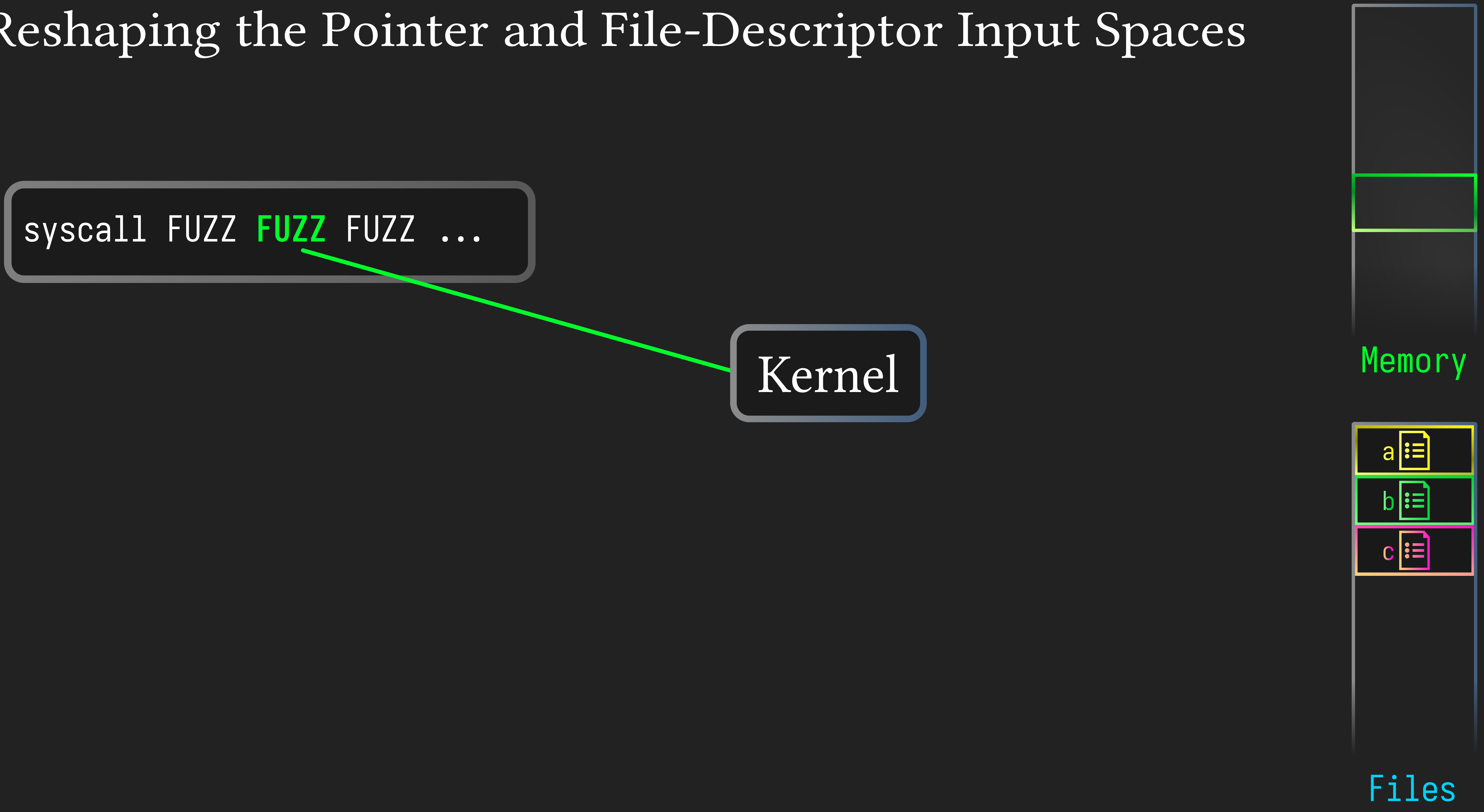
```
syscall FUZZ FUZZ FUZZ FUZZ FUZZ FUZZ
```

What will it take to make **fuzzer-generated pointers and file-descriptors** result in meaningful target behaviors?

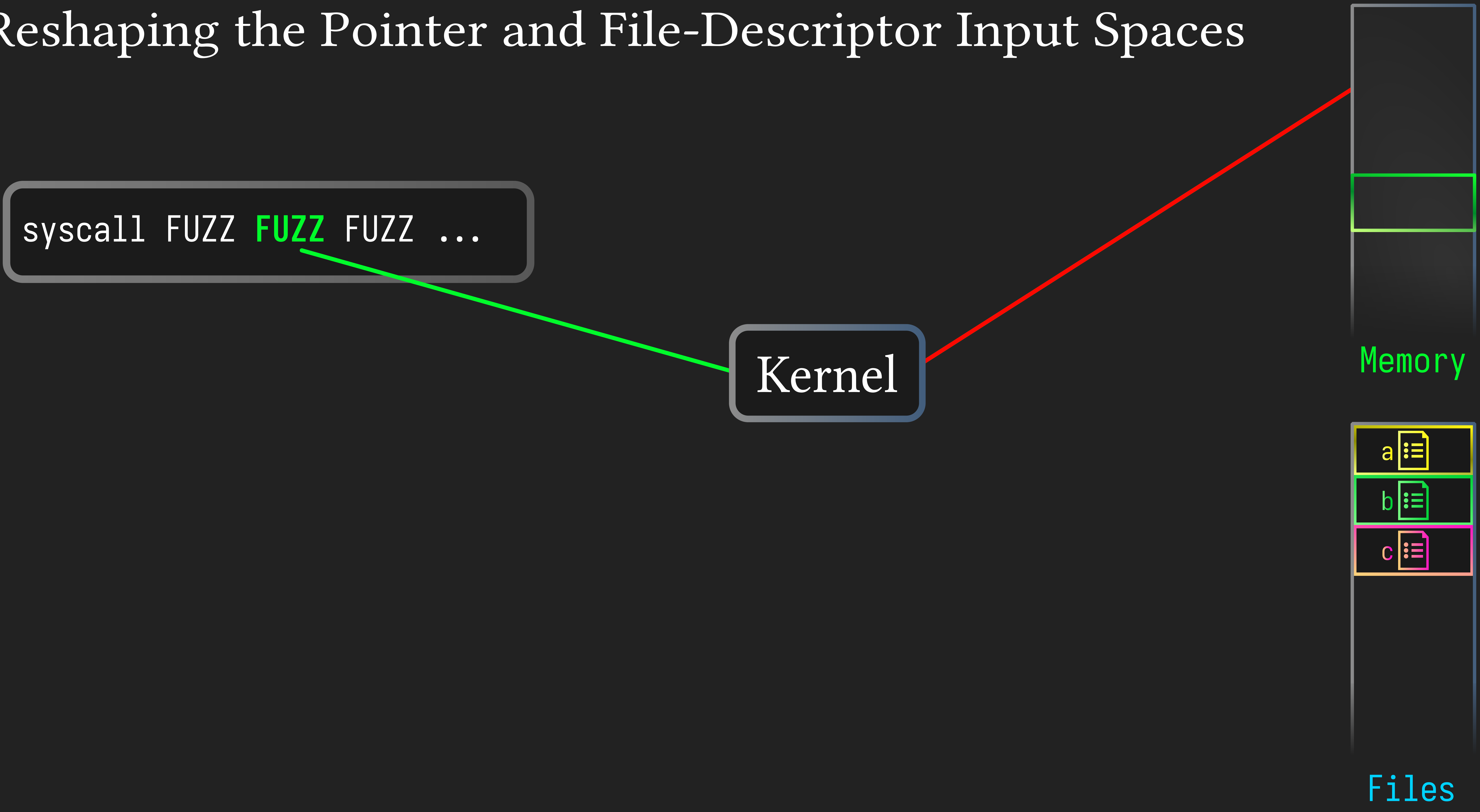
Reshaping the Pointer and File-Descriptor Input Spaces



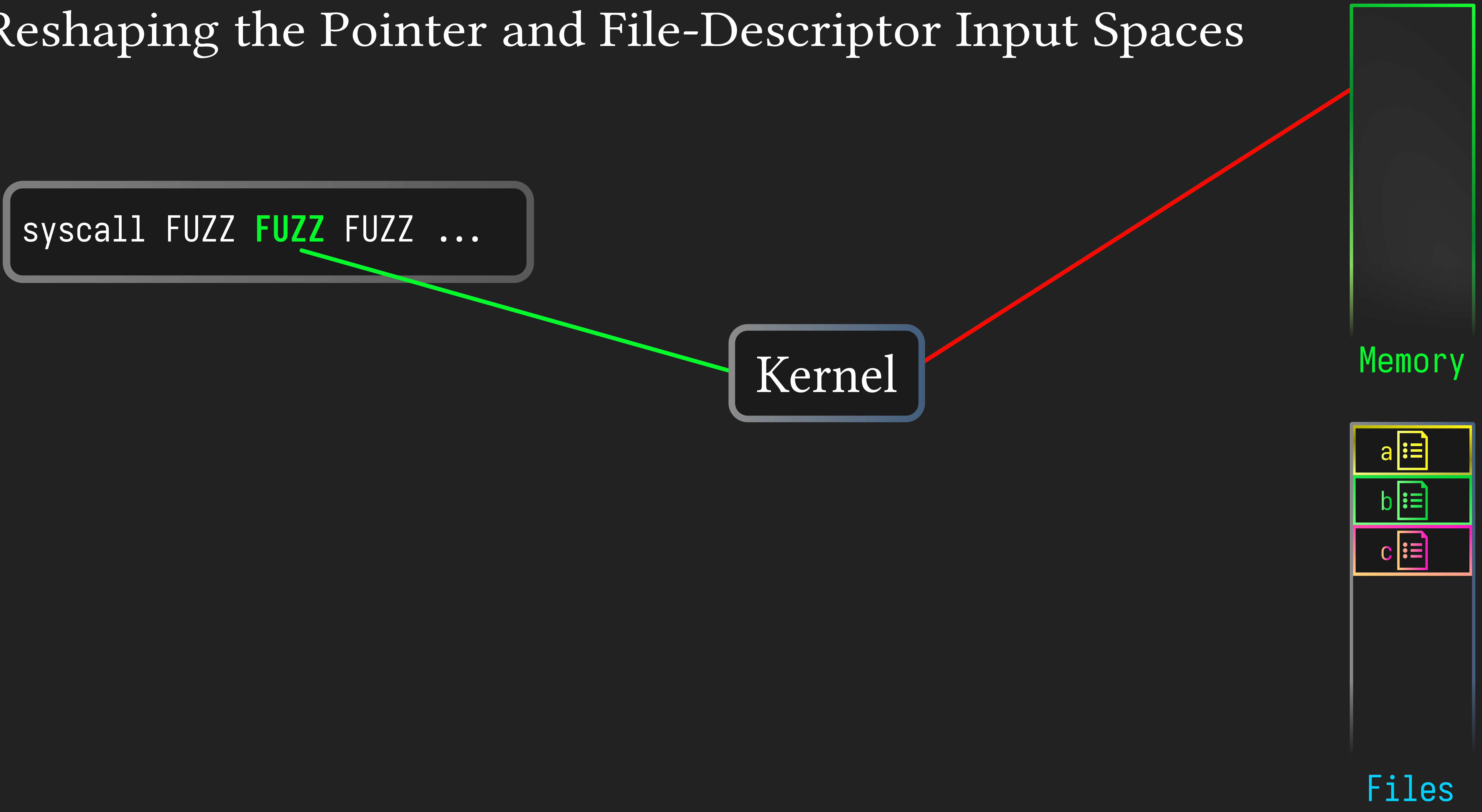
Reshaping the Pointer and File-Descriptor Input Spaces



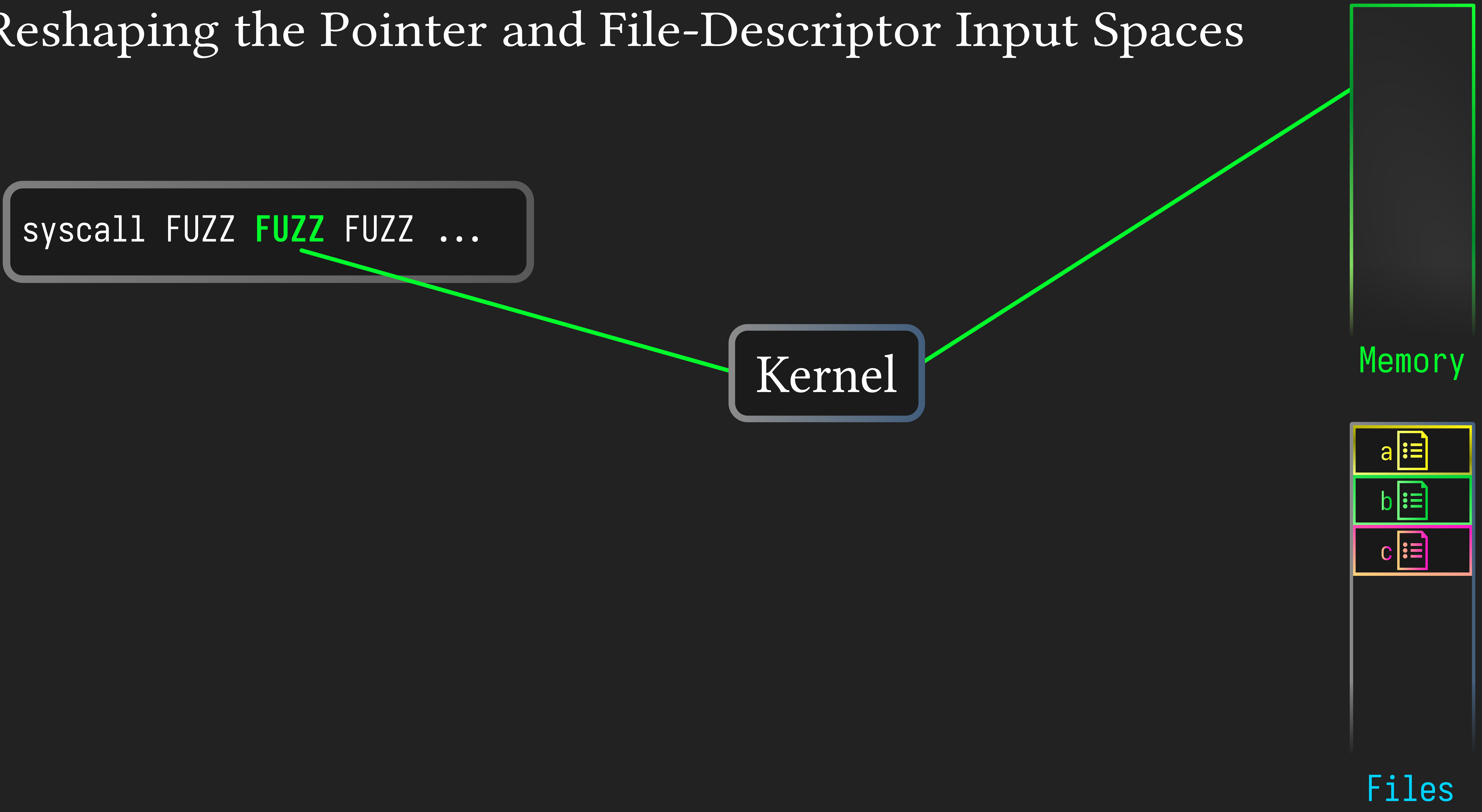
Reshaping the Pointer and File-Descriptor Input Spaces



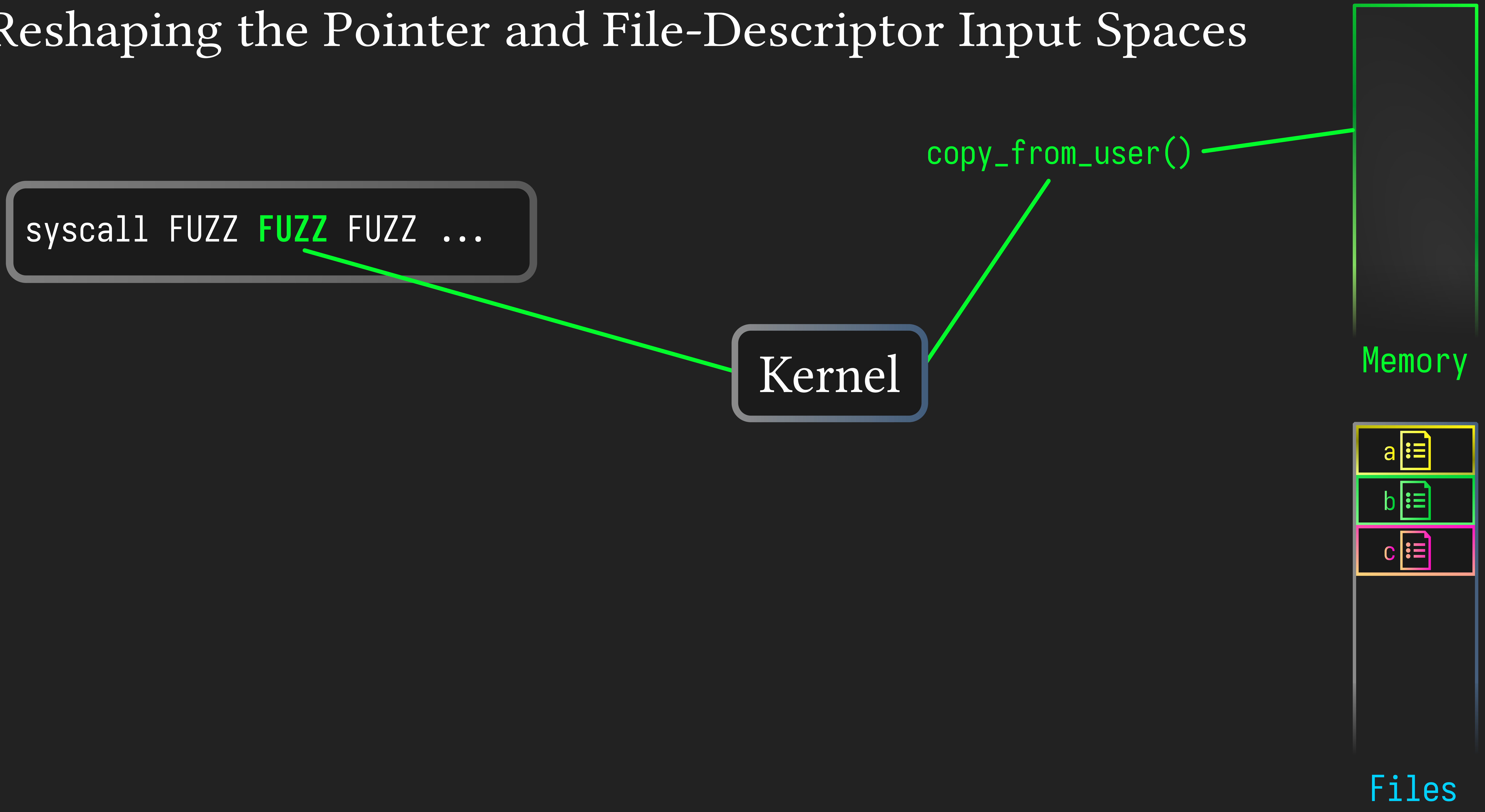
Reshaping the Pointer and File-Descriptor Input Spaces



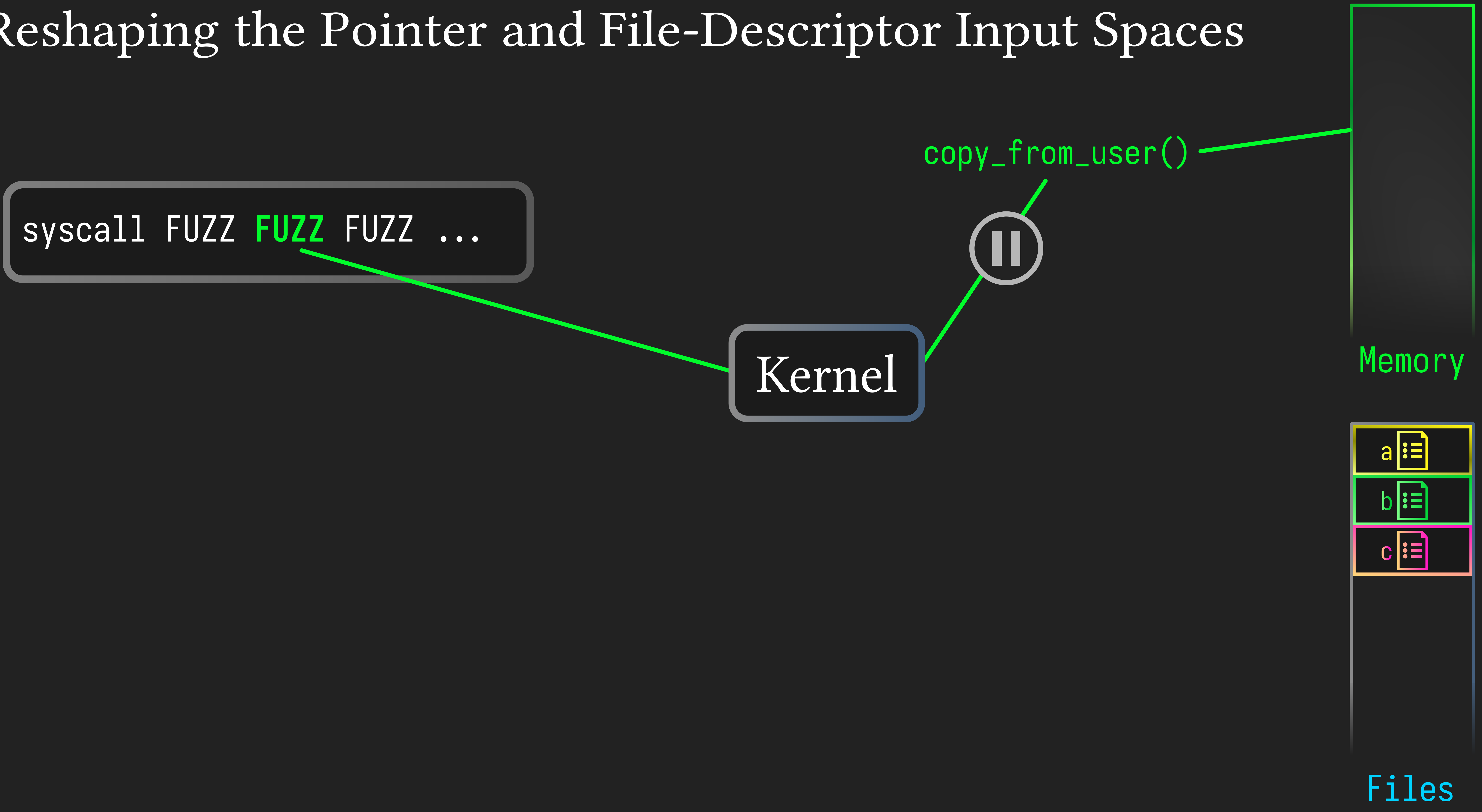
Reshaping the Pointer and File-Descriptor Input Spaces



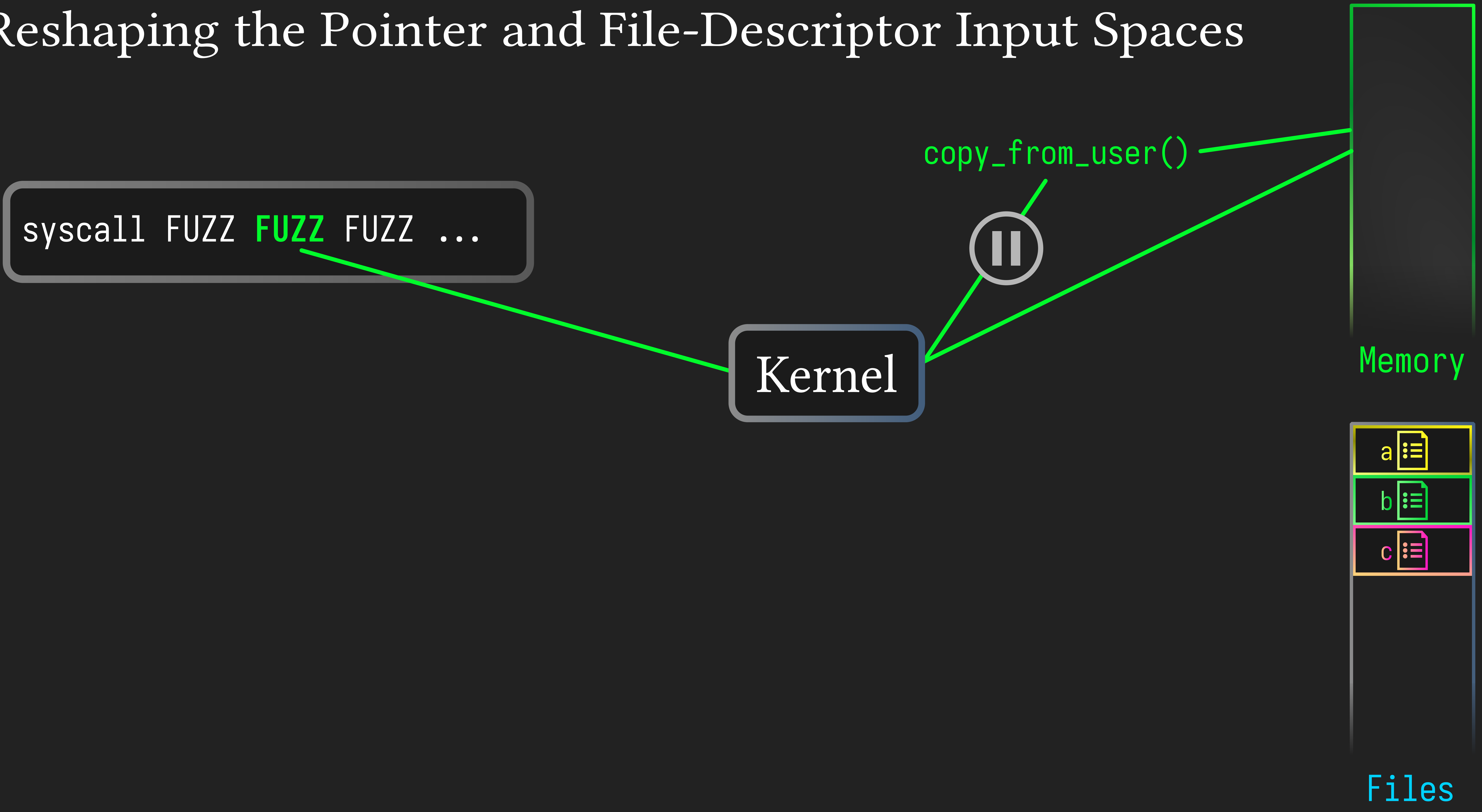
Reshaping the Pointer and File-Descriptor Input Spaces



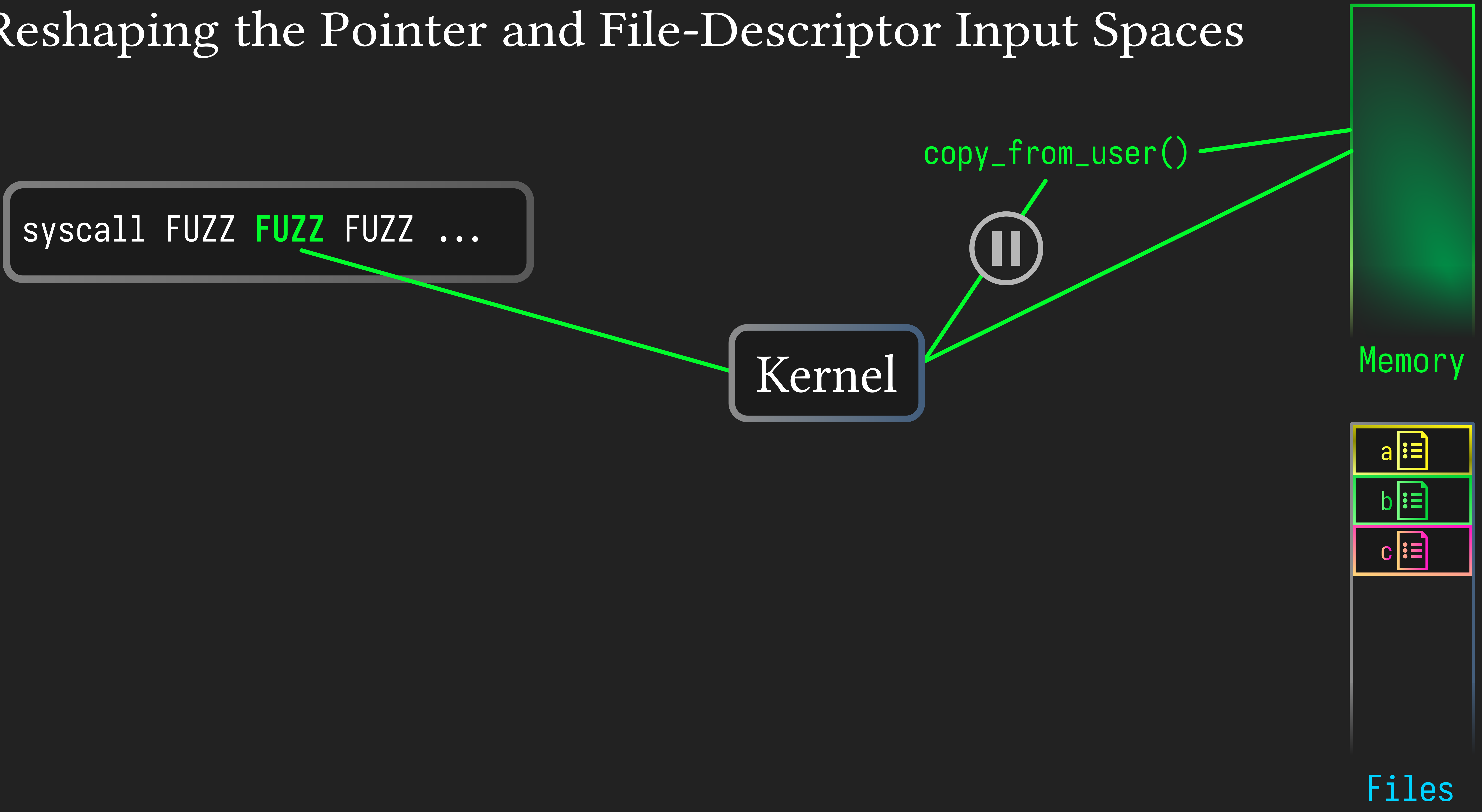
Reshaping the Pointer and File-Descriptor Input Spaces



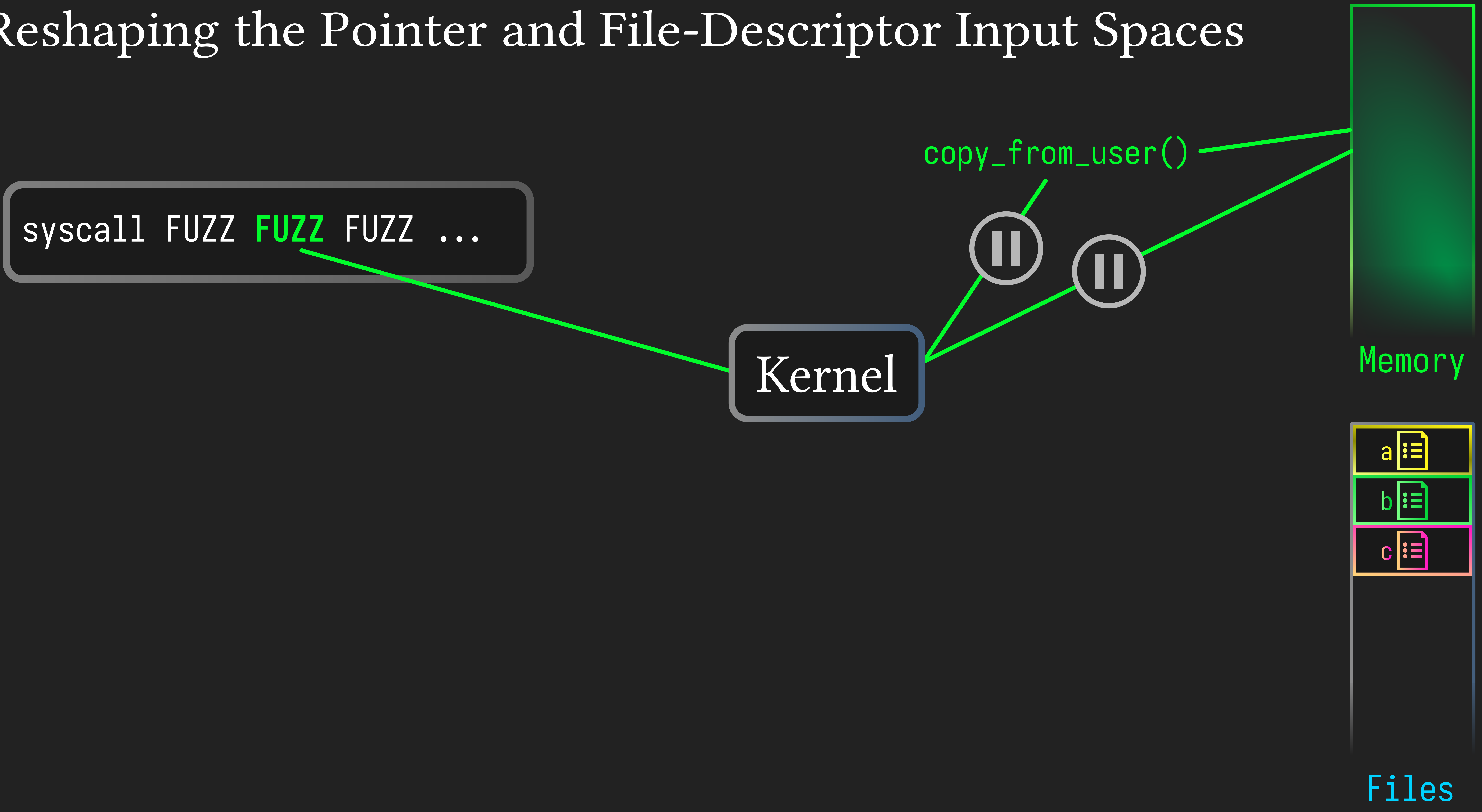
Reshaping the Pointer and File-Descriptor Input Spaces



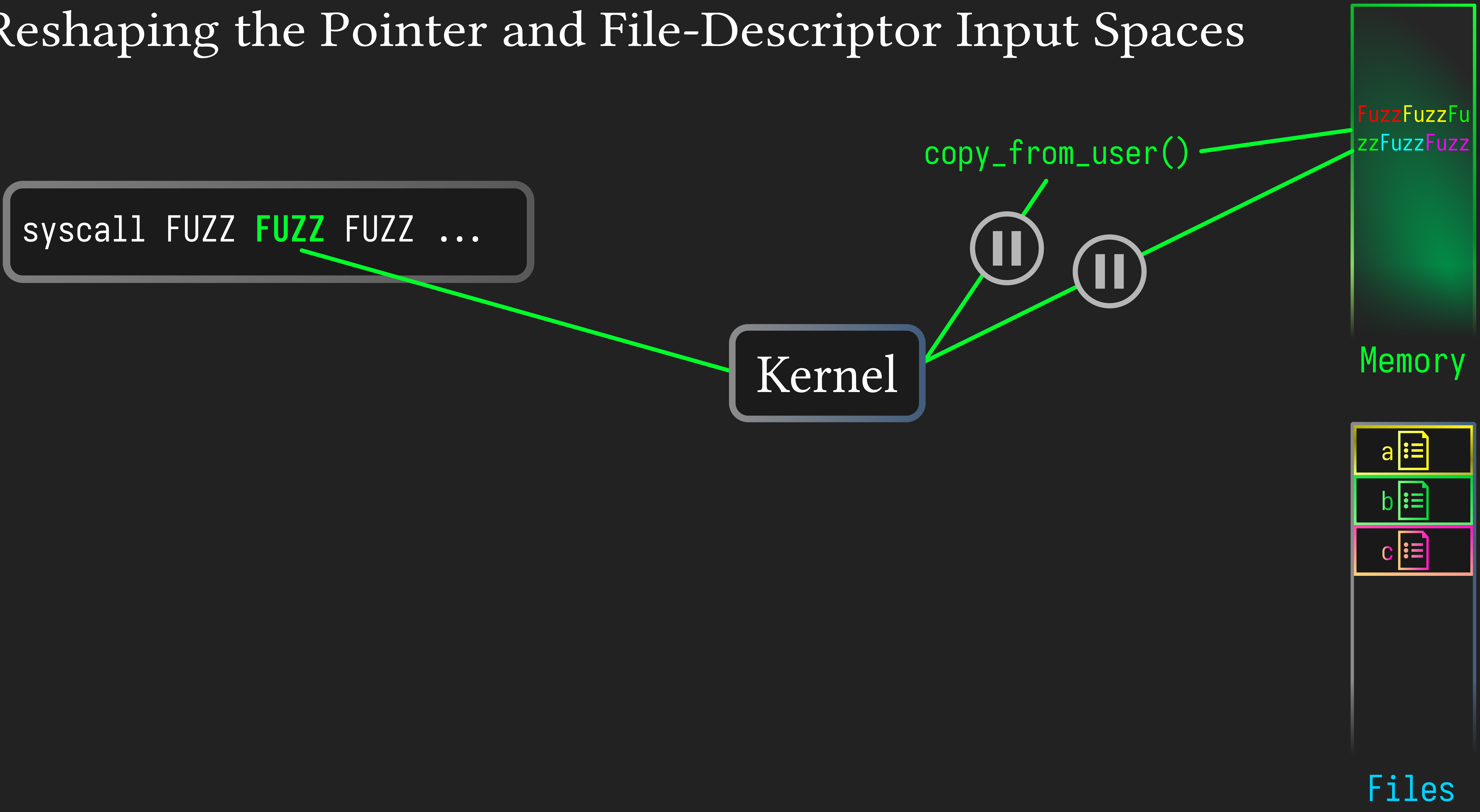
Reshaping the Pointer and File-Descriptor Input Spaces



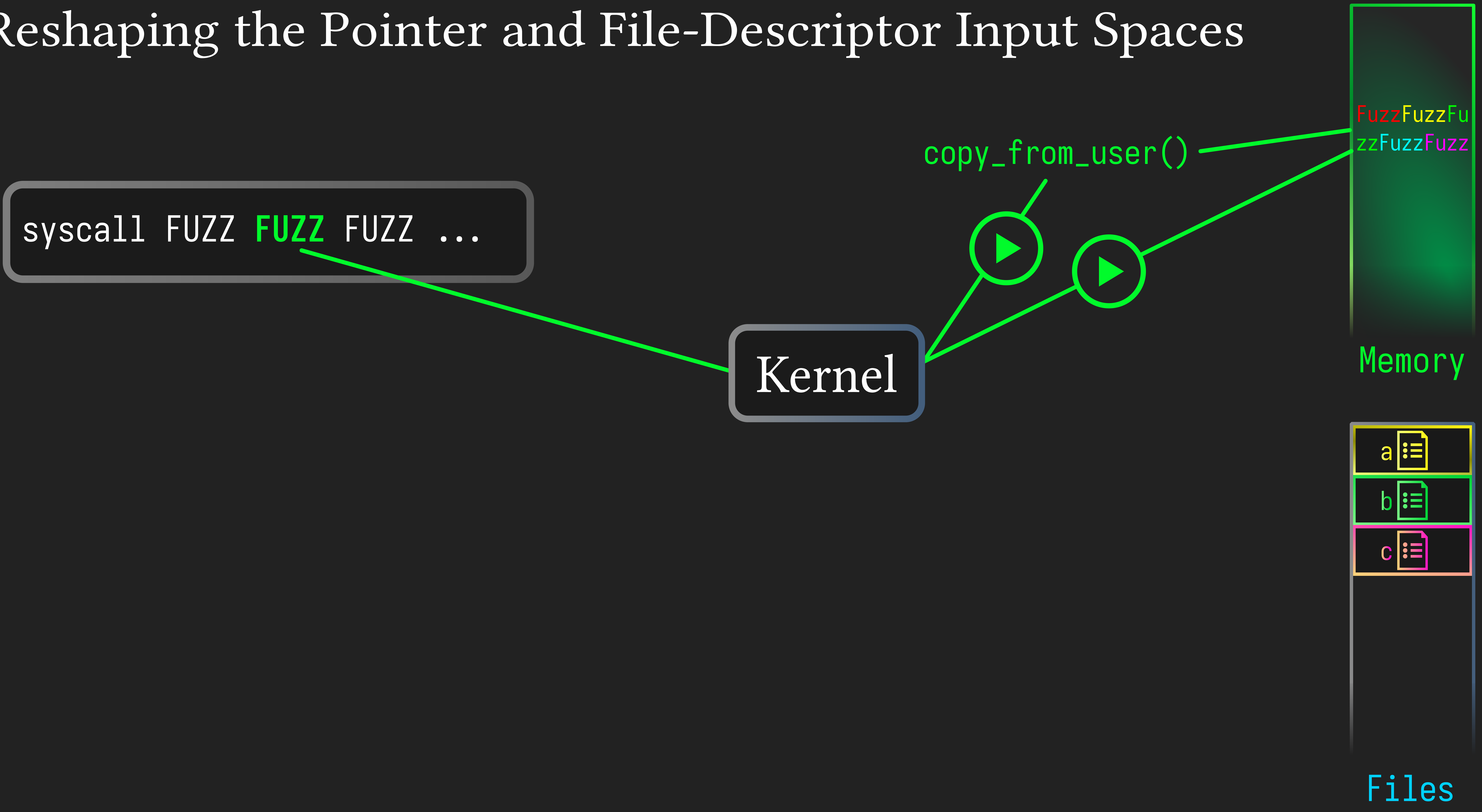
Reshaping the Pointer and File-Descriptor Input Spaces



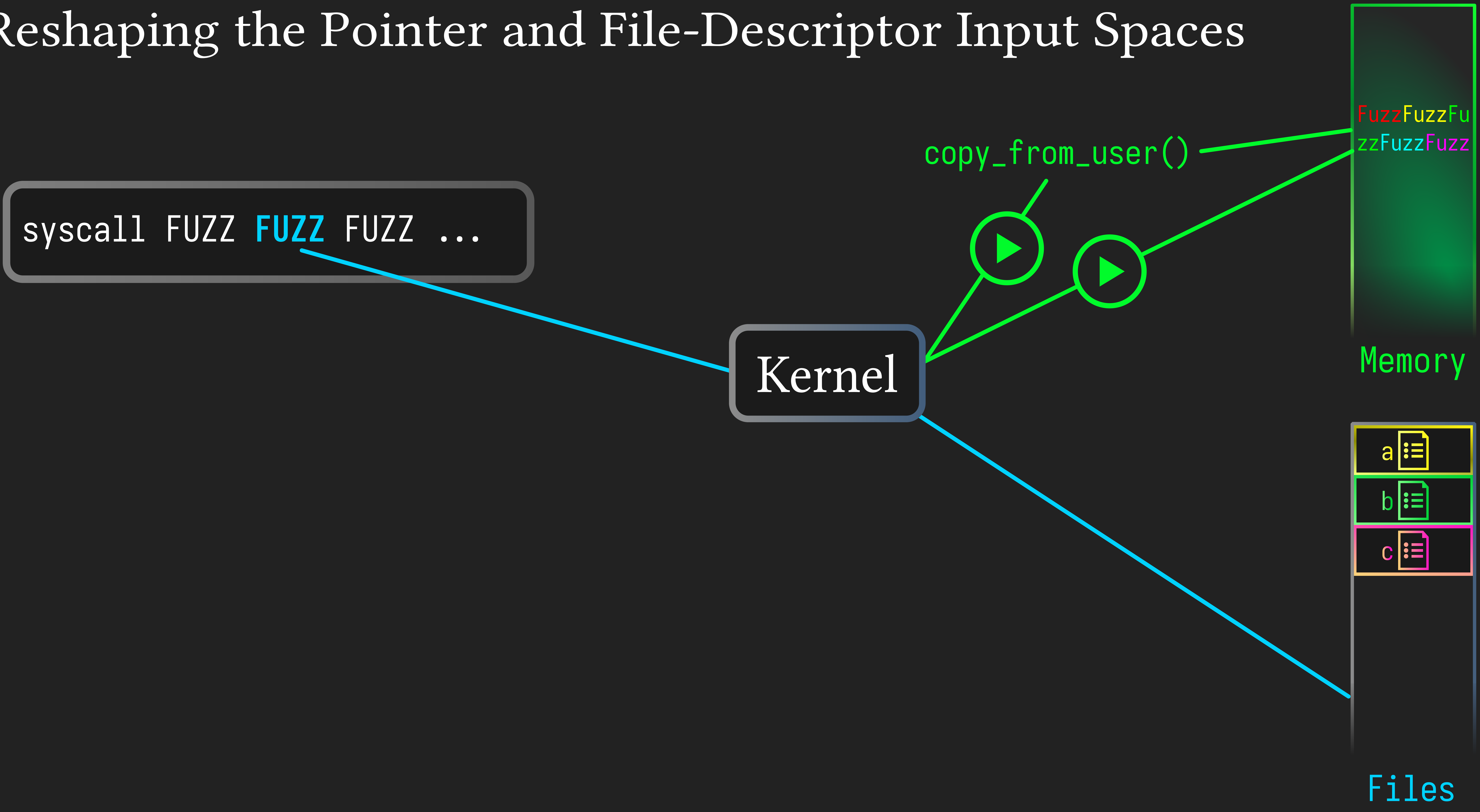
Reshaping the Pointer and File-Descriptor Input Spaces



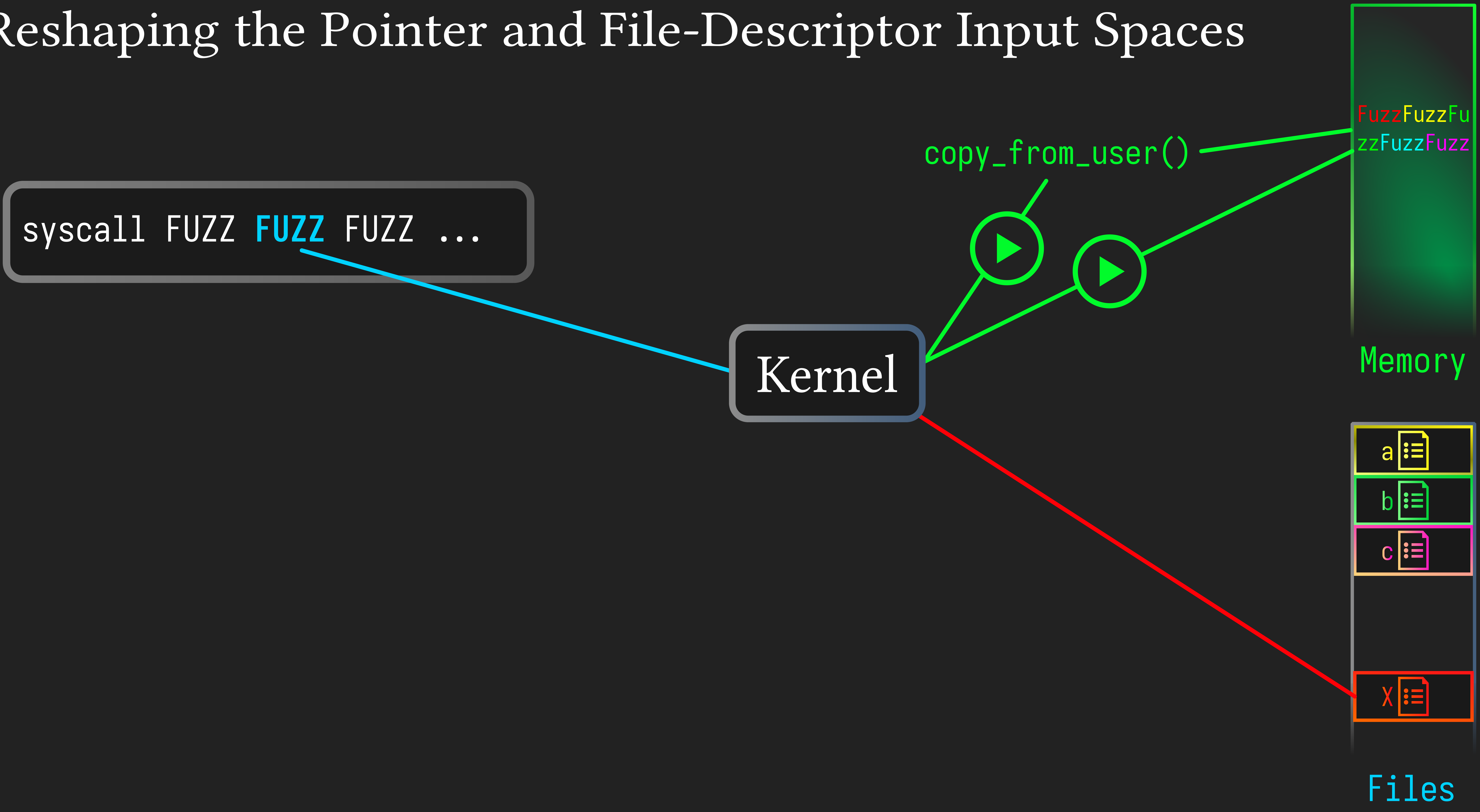
Reshaping the Pointer and File-Descriptor Input Spaces



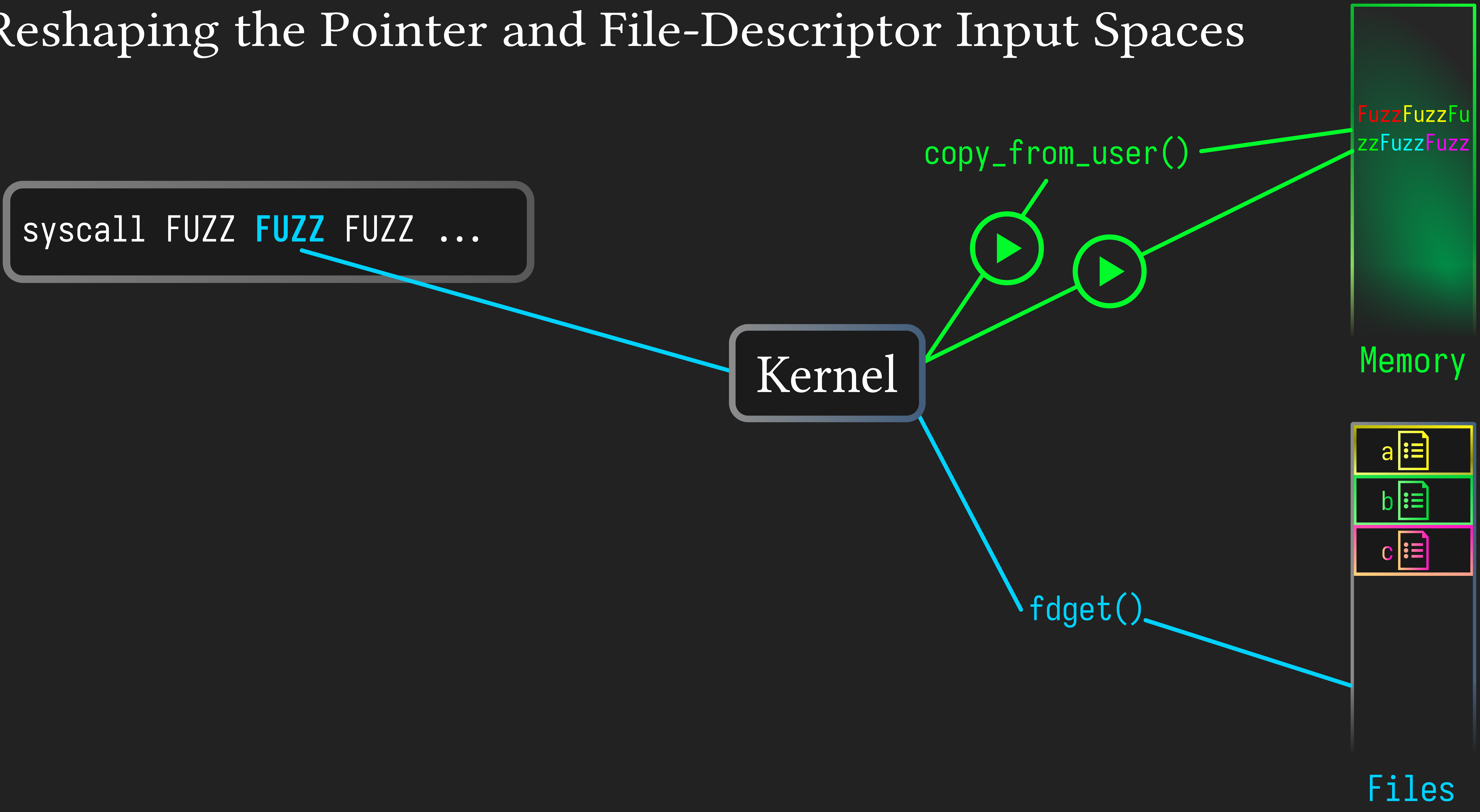
Reshaping the Pointer and File-Descriptor Input Spaces



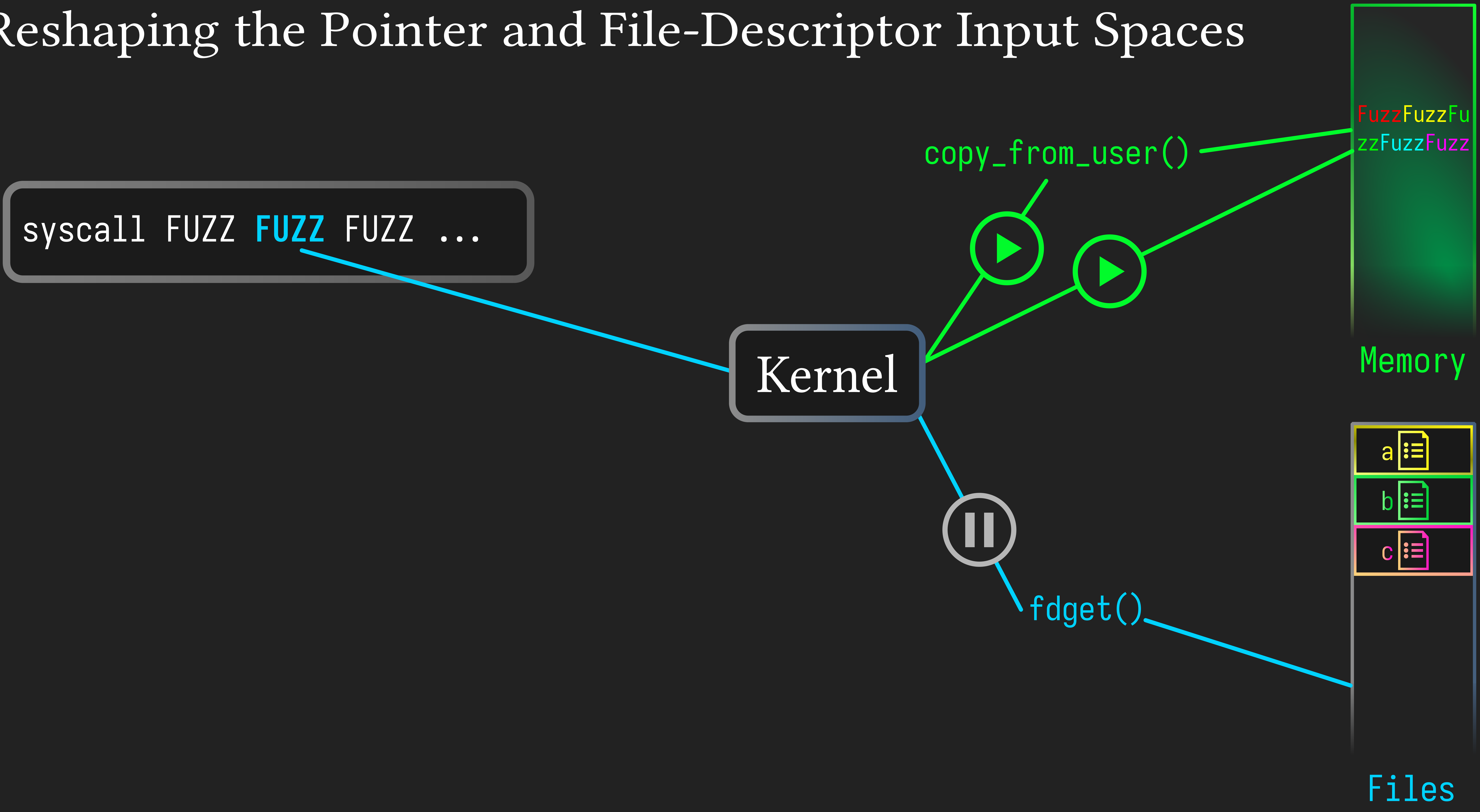
Reshaping the Pointer and File-Descriptor Input Spaces



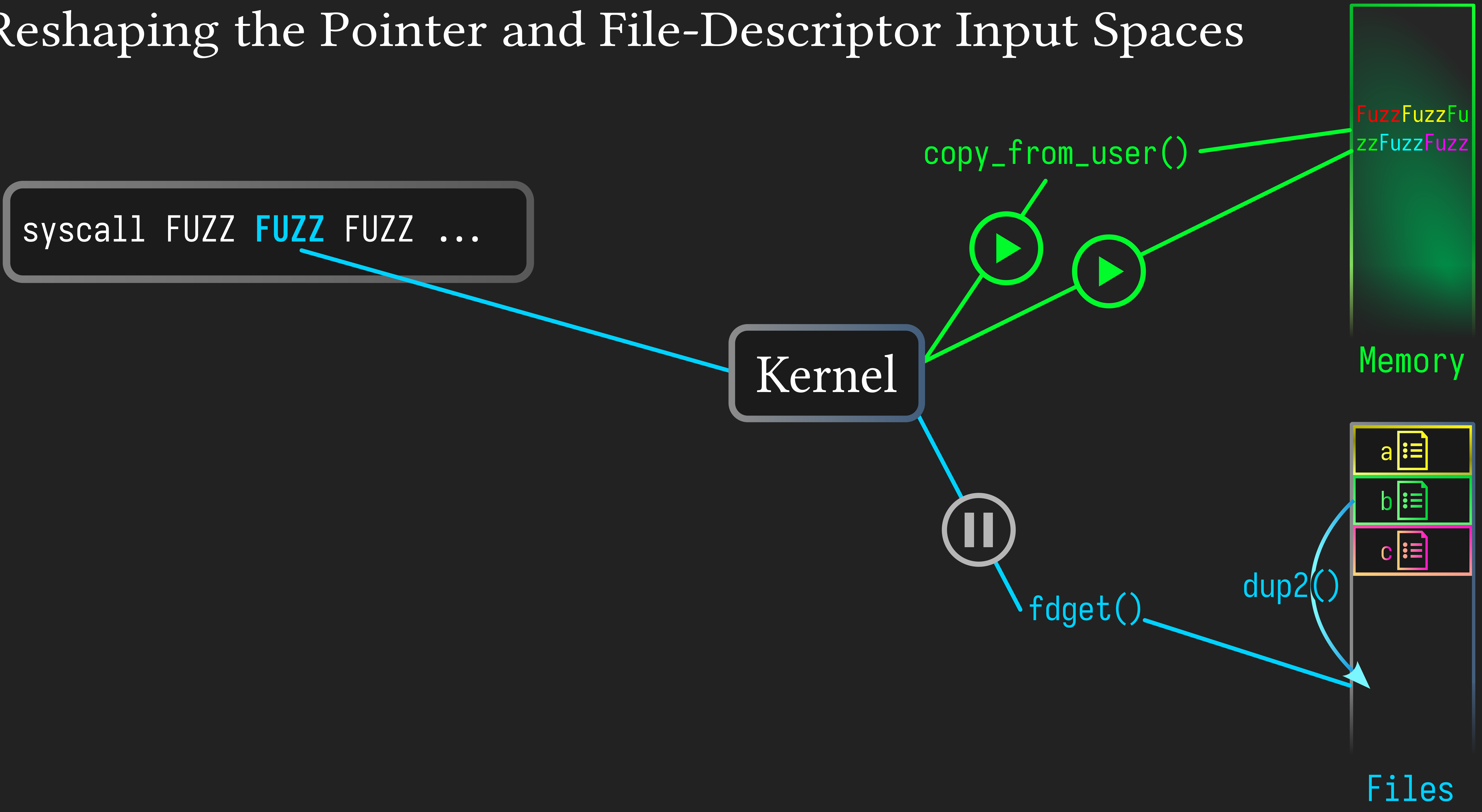
Reshaping the Pointer and File-Descriptor Input Spaces



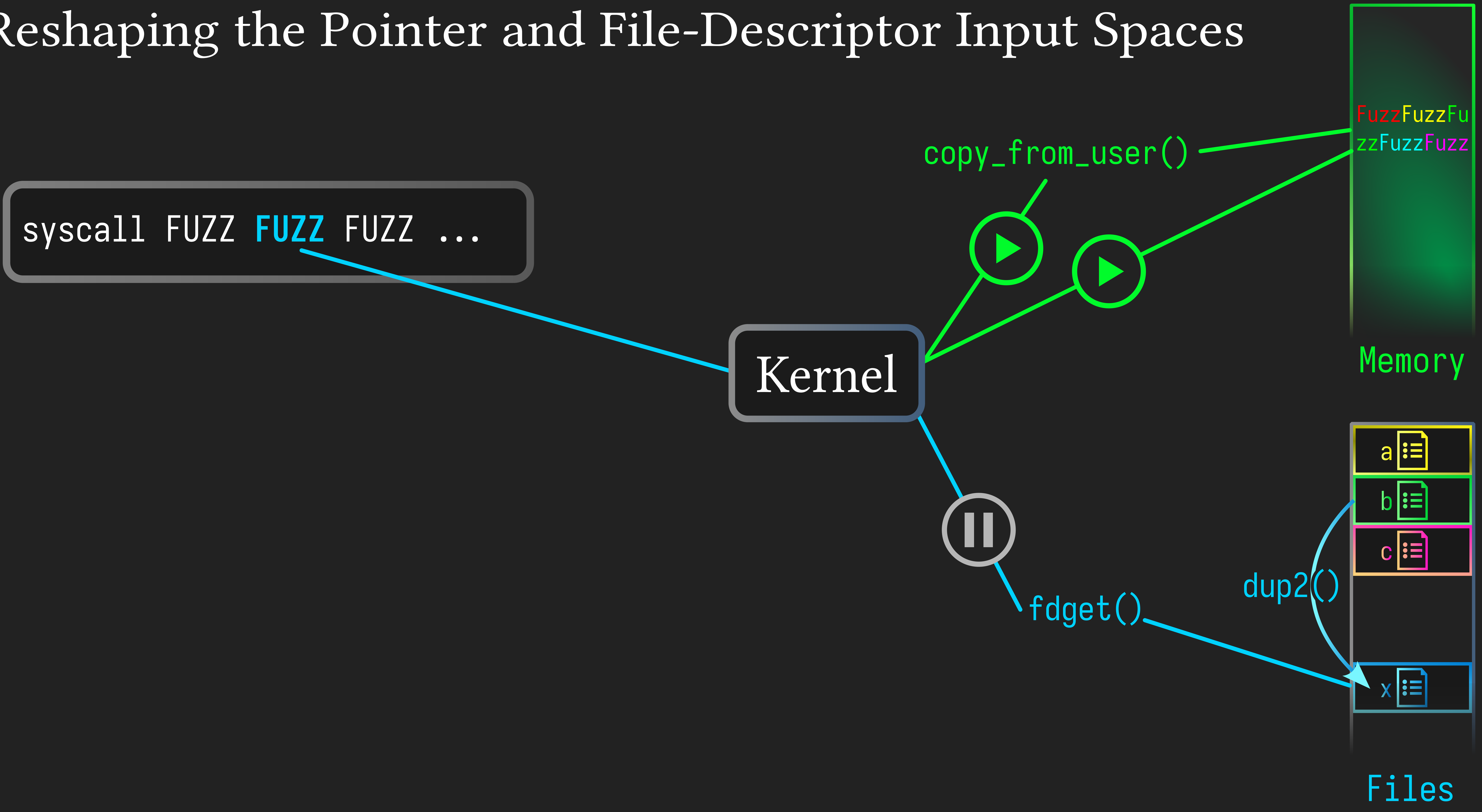
Reshaping the Pointer and File-Descriptor Input Spaces



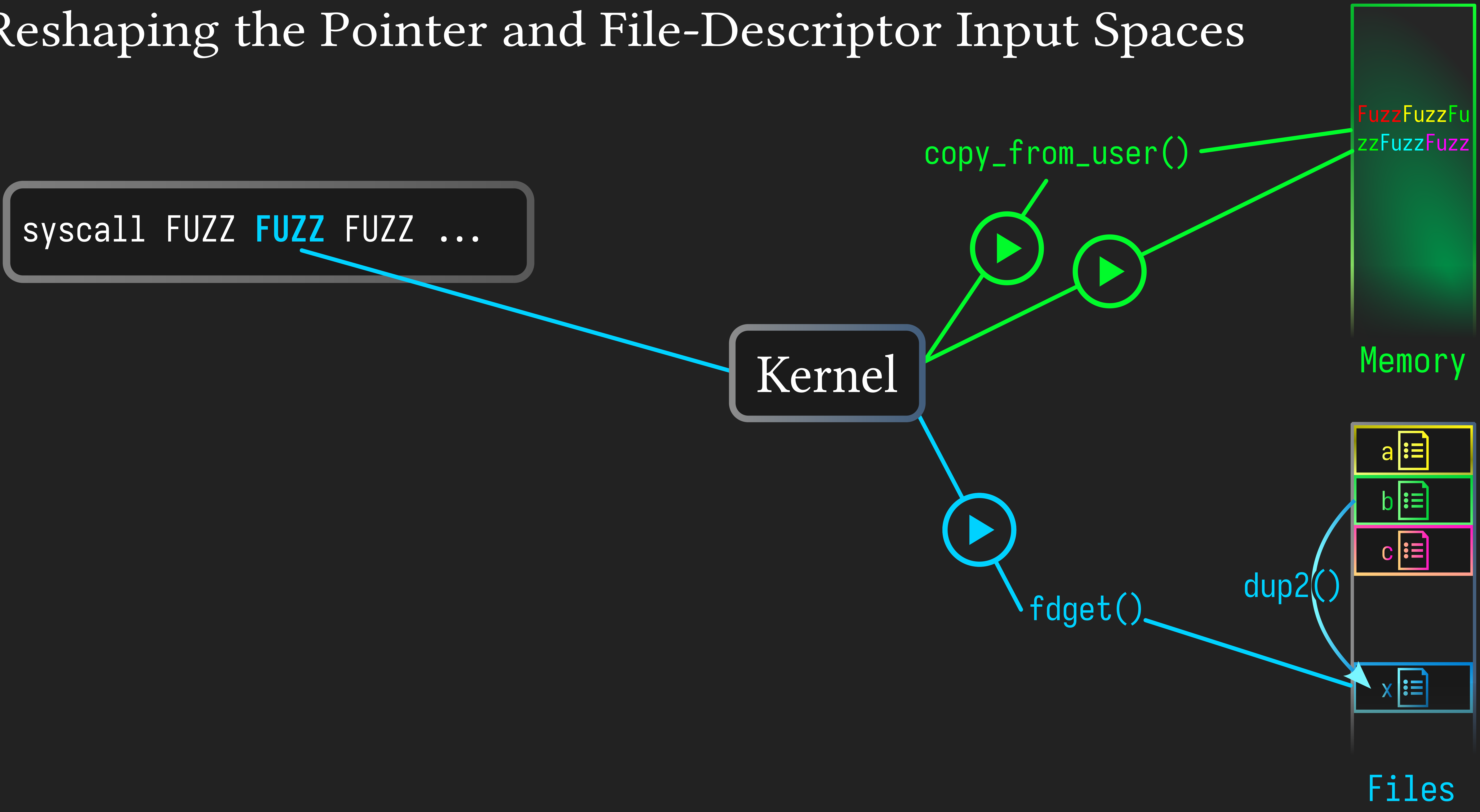
Reshaping the Pointer and File-Descriptor Input Spaces



Reshaping the Pointer and File-Descriptor Input Spaces

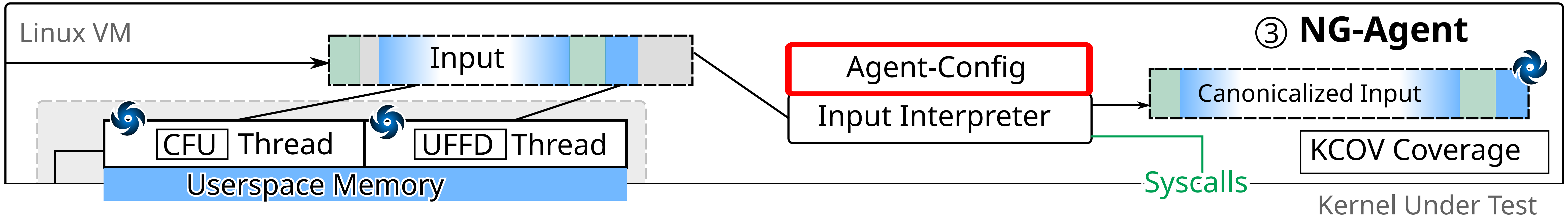


Reshaping the Pointer and File-Descriptor Input Spaces



Config Setup

```
files = "/dev/kvm", O_RDWR
ioctl[-1, -1, -1]
mmap[0, 0xF000, PROT_READ | PROT_WRITE, MAP_SHARED|MAP_POPULATE, 0xFFFF, -1]
close[-1]
fstat[-1, -1]
read[-1, -1, 0xFFFF]
write[-1, -1, 0xFFFF]
```



NG-Agent

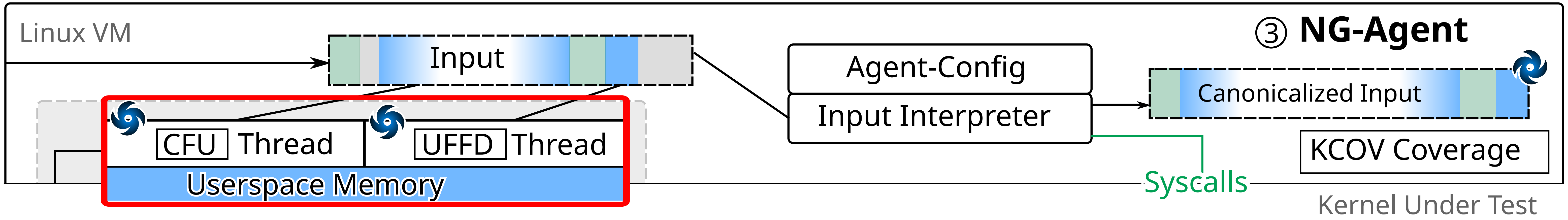
```
files = "/dev/kvm", O_RDWR
ioctl[-1, -1, -1]
mmap[0, 0xF000, PROT_READ | PROT_WRITE, MAP_SHARED|MAP_POPULATE, 0xFFFF, -1]
close[-1]
fstat[-1, -1]
read[-1, -1, 0xFFFF]
write[-1, -1, 0xFFFF]
```

NG-Agent

Setup

Inflate Memory

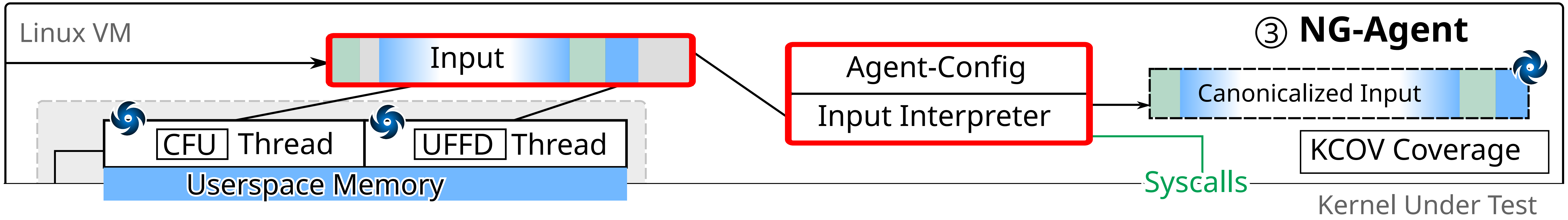
Start up threads to fill memory-accesses



NG-Agent

Interpreter

```
files = "/dev/kvm", O_RDWR
ioctl[-1, -1, -1]
mmap[0, 0xF000, PROT_READ | PROT_WRITE, MAP_SHARED|MAP_POPULATE, 0xFFFF, -1]
close[-1]
fstat[-1, -1]
read[-1, -1, 0xFFFF]
write[-1, -1, 0xFFFF]
```

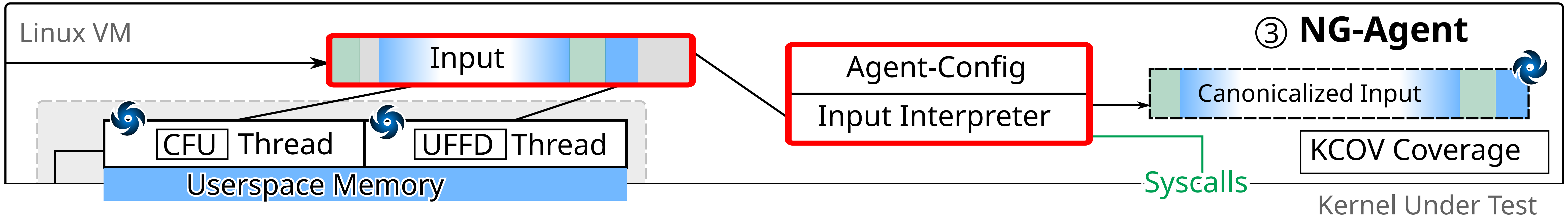


NG-Agent

Interpreter

```
0011001010001000010011010100110110111011111111110011001010001...
```

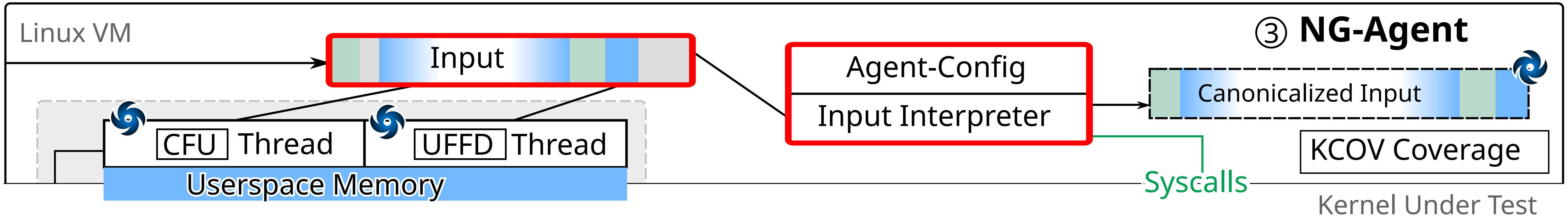
```
files = "/dev/kvm", O_RDWR
ioctl[-1, -1, -1]
mmap[0, 0xF000, PROT_READ | PROT_WRITE, MAP_SHARED|MAP_POPULATE, 0xFFFF, -1]
close[-1]
fstat[-1, -1]
read[-1, -1, 0xFFFF]
write[-1, -1, 0xFFFF]
```



NG-Agent

Interpreter

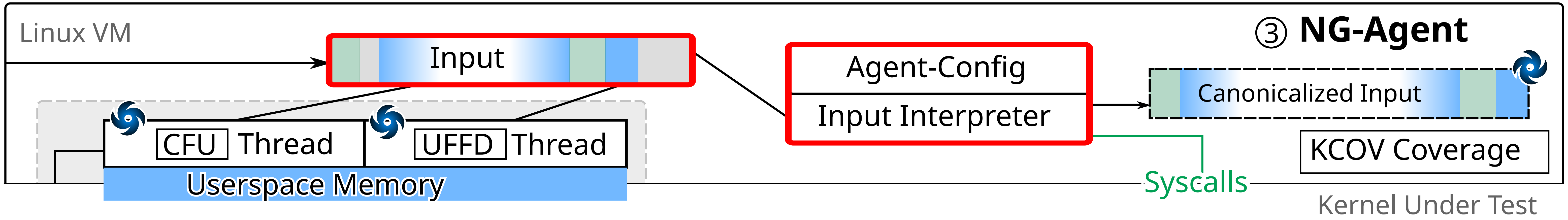
```
files = "/dev/kvm", O_RDWR
ioctl[-1, -1, -1]
mmap[0, 0xF000, PROT_READ | PROT_WRITE, MAP_SHARED|MAP_POPULATE, 0xFFFF, -1]
close[-1]
fstat[-1, -1]
read[-1, -1, 0xFFFF]
write[-1, -1, 0xFFFF]
```



NG-Agent

Interpreter

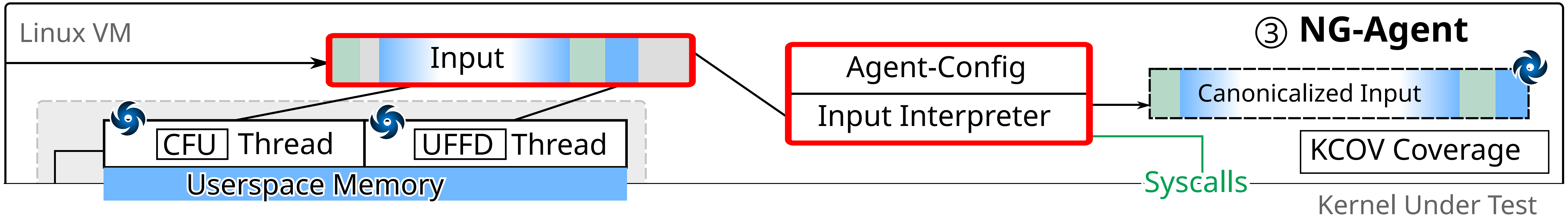
```
files = "/dev/kvm", O_RDWR
ioctl[-1, -1, -1]
mmap[0, 0xF000, PROT_READ | PROT_WRITE, MAP_SHARED|MAP_POPULATE, 0xFFFF, -1]
close[-1]
fstat[-1, -1]
read[-1, -1, 0xFFFF]
write[-1, -1, 0xFFFF]
```



NG-Agent

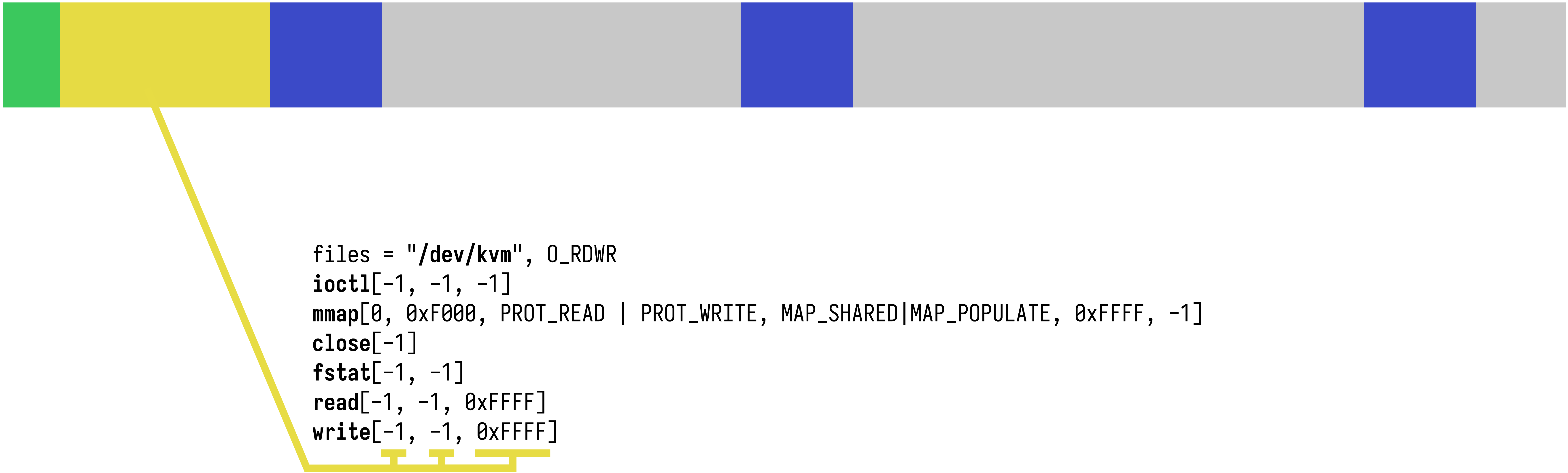
Interpreter

```
files = "/dev/kvm", O_RDWR
ioctl[-1, -1, -1]
mmap[0, 0xF000, PROT_READ | PROT_WRITE, MAP_SHARED|MAP_POPULATE, 0xFFFF, -1]
close[-1]
fstat[-1, -1]
read[-1, -1, 0xFFFF]
write[-1, -1, 0xFFFF]
```

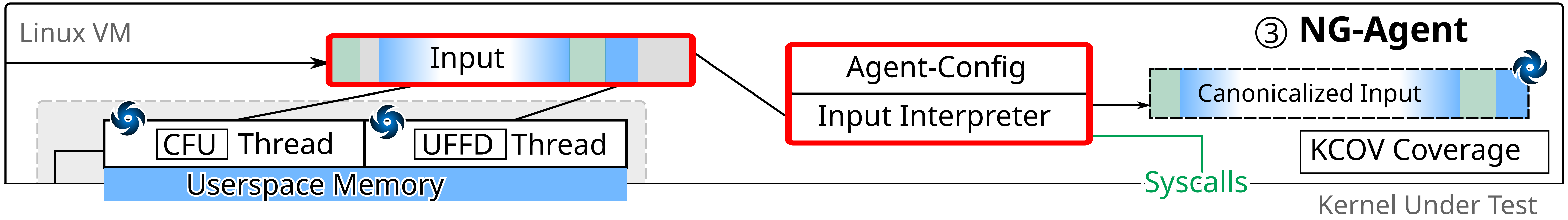


NG-Agent

Interpreter



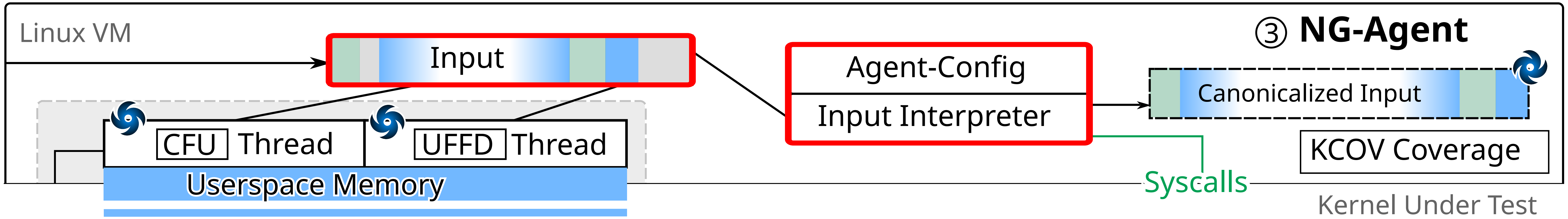
```
files = "/dev/kvm", O_RDWR
ioctl[-1, -1, -1]
mmap[0, 0xF000, PROT_READ | PROT_WRITE, MAP_SHARED|MAP_POPULATE, 0xFFFF, -1]
close[-1]
fstat[-1, -1]
read[-1, -1, 0xFFFF]
write[-1, -1, 0xFFFF]
```



NG-Agent

Interpreter

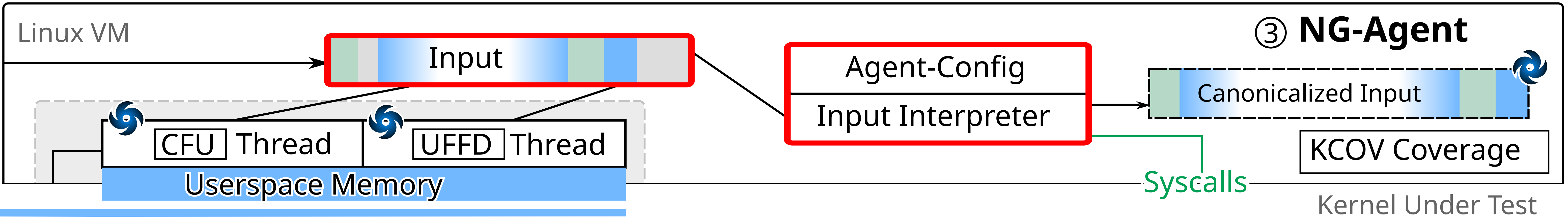
```
files = "/dev/kvm", O_RDWR
ioctl[-1, -1, -1]
mmap[0, 0xF000, PROT_READ | PROT_WRITE, MAP_SHARED|MAP_POPULATE, 0xFFFF, -1]
close[-1]
fstat[-1, -1]
read[-1, -1, 0xFFFF]
write[-1, -1, 0xFFFF]
```



NG-Agent

Interpreter

```
files = "/dev/kvm", O_RDWR
ioctl[-1, -1, -1]
mmap[0, 0xF000, PROT_READ | PROT_WRITE, MAP_SHARED|MAP_POPULATE, 0xFFFF, -1]
close[-1]
fstat[-1, -1]
read[-1, -1, 0xFFFF]
write[-1, -1, 0xFFFF]
```



Results

- Linux 5.12

Results

- Linux 5.12
- 13 Components

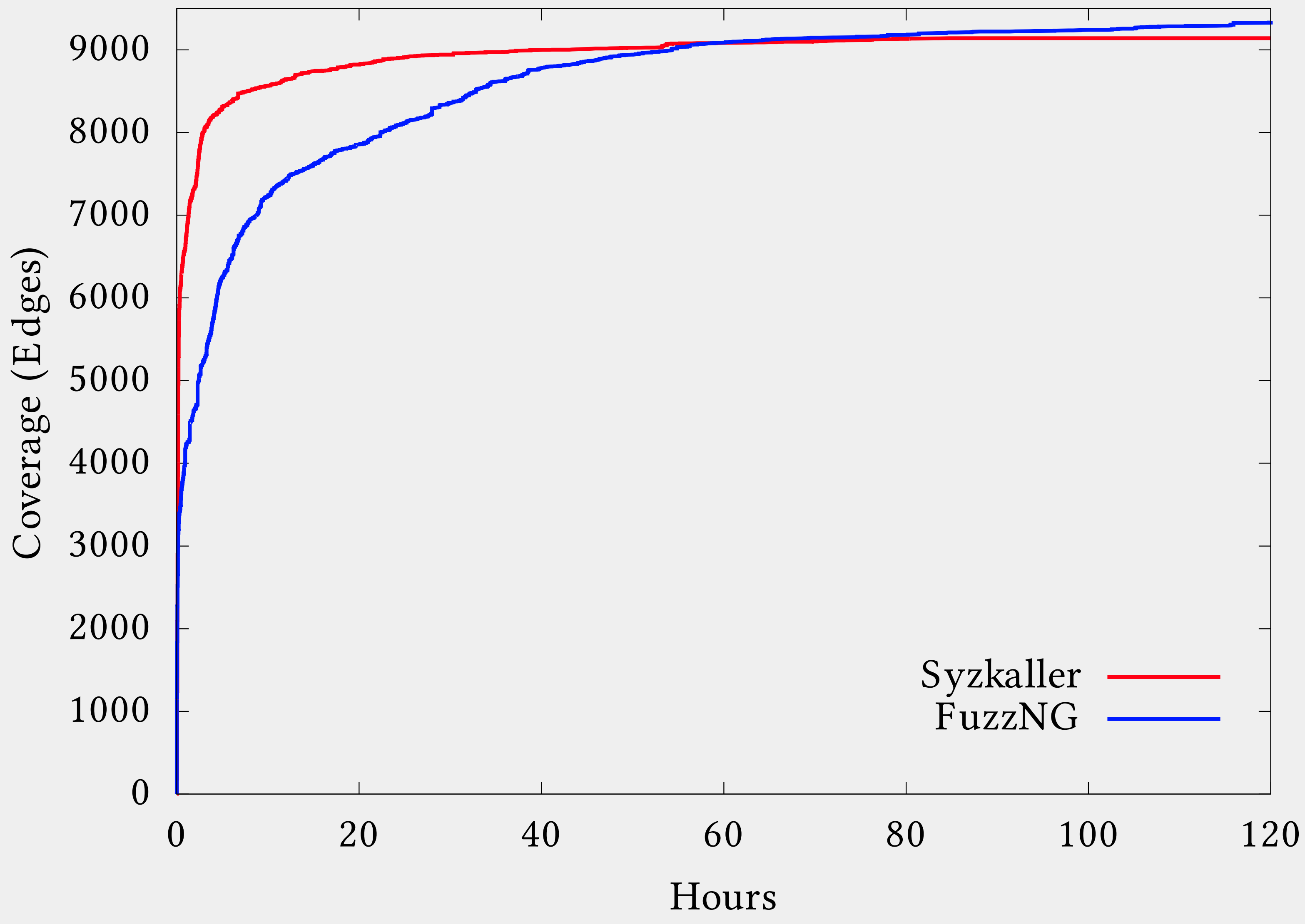
```
bpf
video4linux
rdma
binder
cdrom
kvm
vhost_net
drm
io_uring
vt_ioctl
ptmx
nvme
megaraid
```

Results

- Linux 5.12
- 13 Components
- Coverage
  - Fuzzed for 7 Days on 20 Cores
  - 102.5% of Syzkaller's Coverage
  - Configurations <1.7% of Syzkaller's

```
bpf
video4linux
rdma
binder
cdrom
kvm
vhost_net
drm
io_uring
vt_ioctl
ptmx
nvme
megaraid
```

Results



Component	Max Cov	Syzkaller		FuzzNG	
		Edge Count	Syzlang LoC	Edge Count	Config LoC
bpf	15359	3623	864	3572	1
video4linux	1004	563	381	567	4
rdma	4014	562	1474	591	5
binder	2506	340	272	344	6
cdrom	956	138	351	144	5
kvm	34924	9213	891	9468	7
vhost_net	415	218	157	225	9
drm	12503	2296	745	2138	7
io_uring	3413	982	343	1003	6
vt_ioctl	332	142	381	162	9
Average				**102.53%**	**1.67%**
Geo. Mean				**102.41%**	**1.09%**

Bend the system-call input space to make it conducive to fuzzing

Use time-tested **off-the-shelf fuzzers**

Competitive fuzzing performance with **tiny component configs**

FuzzNG

Recap:

Target		MORPHUZZ	HyperPill	FuzzNG
	Hypervisor	✓	✓	
	Kernel			✓

Reshaping applies to independent, complex targets

Recap:

		Morphuzz	HyperPill	FuzzNG
Target	Hypervisor	✓	✓	
	Kernel			✓
Reshaping Level	Source-Level	✓		✓
	ISA-Level		✓	

Reshaping can be applied even without access to a target's source

Recap:

		MORPHUZZ	HyperPill	FuzzNG
Target	Hypervisor	✓	✓	
	Kernel			✓
Reshaping Level	Source-Level	✓		✓
	ISA-Level		✓	
Fuzzing Techniques	VM-Snapshotting		✓	✓
	Process-Based	✓		

Reshaping composes with diverse fuzzing techniques

Recap:

		MORPHUZZ	HyperPill	FuzzNG
Target	Hypervisor	✓	✓	
	Kernel			✓
Reshaping Level	Source-Level	✓		✓
	ISA-Level		✓	
Fuzzing Techniques	VM-Snapshotting		✓	✓
	Process-Based	✓		
	Emulation		✓	
	Native Execution	✓		✓

Reshaping composes with diverse fuzzing techniques

Is reshaped fuzzing...

RQ1: effective at finding bugs?

RQ2: competitve with other approaches on coverage-achieved?

RQ3: applicable to a diverse set of targets?

RQ4: beneficial even when grammars exist?

RQ5: compatible with other SoTA fuzzing techniques?

Is reshaped fuzzing...

RQ1: effective at finding bugs?

RQ2: competitve with other approaches on coverage-achieved?

RQ3: applicable to a diverse set of targets?

RQ4: beneficial even when grammars exist?

RQ5: compatible with other SoTA fuzzing techniques?

Over 100 new bugs found.

Bugs found in code that was already "covered" by grammar-based fuzzers.

Is reshaped fuzzing...

RQ1: effective at finding bugs?

RQ2: competitve with other approaches on coverage-achieved?

RQ3: applicable to a diverse set of targets?

RQ4: beneficial even when grammars exist?

RQ5: compatible with other SoTA fuzzing techniques?

Is reshaped fuzzing...

RQ1: effective at finding bugs?

RQ2: competitve with other approaches on coverage-achieved?

RQ3: applicable to a diverse set of targets?

RQ4: beneficial even when grammars exist?

RQ5: compatible with other SoTA fuzzing techniques?

Morphuzz, FuzzNG are extensible to other targets.

HyperPill applies to an entire class of diverse targets.

Is reshaped fuzzing...

RQ1: effective at finding bugs?

RQ2: competitve with other approaches on coverage-achieved?

RQ3: applicable to a diverse set of targets?

RQ4: beneficial even when grammars exist?

RQ5: compatible with other SoTA fuzzing techniques?

Morphuzz and FuzzNG found bugs in code and components that had already been fuzzed by grammar-based approaches.

Is reshaped fuzzing...

RQ1: effective at finding bugs?

RQ2: competitve with other approaches on coverage-achieved?

RQ3: applicable to a diverse set of targets?

RQ4: beneficial even when grammars exist?

RQ5: compatible with other SoTA fuzzing techniques?

Our reshaping-based fuzzers integrate with diverse state-of-the-art techiques, such as fork-servers, full-system-fuzzing and emulator-based fuzzing.

Is reshaped fuzzing...

RQ1: effective at finding bugs?

RQ2: competitve with other approaches on coverage-achieved?

RQ3: applicable to a diverse set of targets?

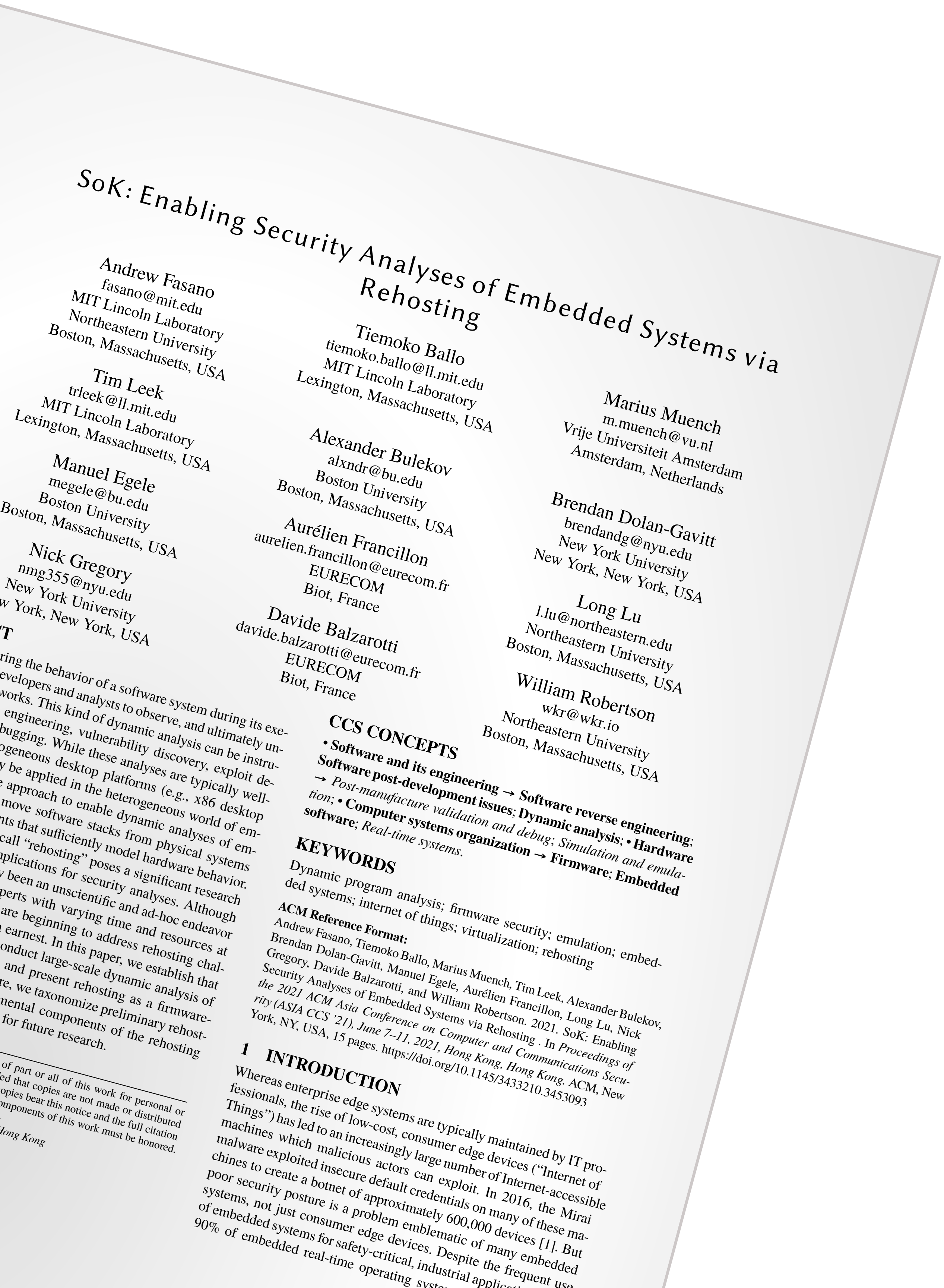
RQ4: beneficial even when grammars exist?

RQ5: compatible with other SoTA fuzzing techniques?

Thesis:

Input-space reshaping is more effective than grammar-based harnessing approaches for fuzzing complex targets.

SoK: Enabling Security Analyses of Embedded Systems via **Rehosting**


SoK: Enabling Security Analyses of Embedded Systems via Rehosting

Andrew Fasano
fasano@mit.edu
MIT Lincoln Laboratory
Northeastern University
Boston, Massachusetts, USA

Tiemoko Ballo
tiemoko.ballo@ll.mit.edu
MIT Lincoln Laboratory
Lexington, Massachusetts, USA

Marius Muench
m.muench@vu.nl
Vrije Universiteit Amsterdam
Amsterdam, Netherlands

Tim Leek
tleek@ll.mit.edu
MIT Lincoln Laboratory
Lexington, Massachusetts, USA

Alexander Bulekov
alxndr@bu.edu
Boston University
Boston, Massachusetts, USA

Brendan Dolan-Gavitt
brendandg@nyu.edu
New York University
New York, New York, USA

Manuel Egele
megele@bu.edu
Boston University
Boston, Massachusetts, USA

Aurélien Francillon
aurelien.francillon@eurecom.fr
EURECOM
Biot, France

Long Lu
l.lu@northeastern.edu
Northeastern University
Boston, Massachusetts, USA

Nick Gregory
nmg355@nyu.edu
New York University
New York, New York, USA

Davide Balzarotti
davide.balzarotti@eurecom.fr
EURECOM
Biot, France

William Robertson
wkr@wkr.io
Northeastern University
Boston, Massachusetts, USA

CCS CONCEPTS
• Software and its engineering → Software reverse engineering; Software post-development issues; Dynamic analysis; • Hardware → Post-manufacture validation and debug; Simulation and emulation; • Computer systems organization → Firmware; Embedded software; Real-time systems.

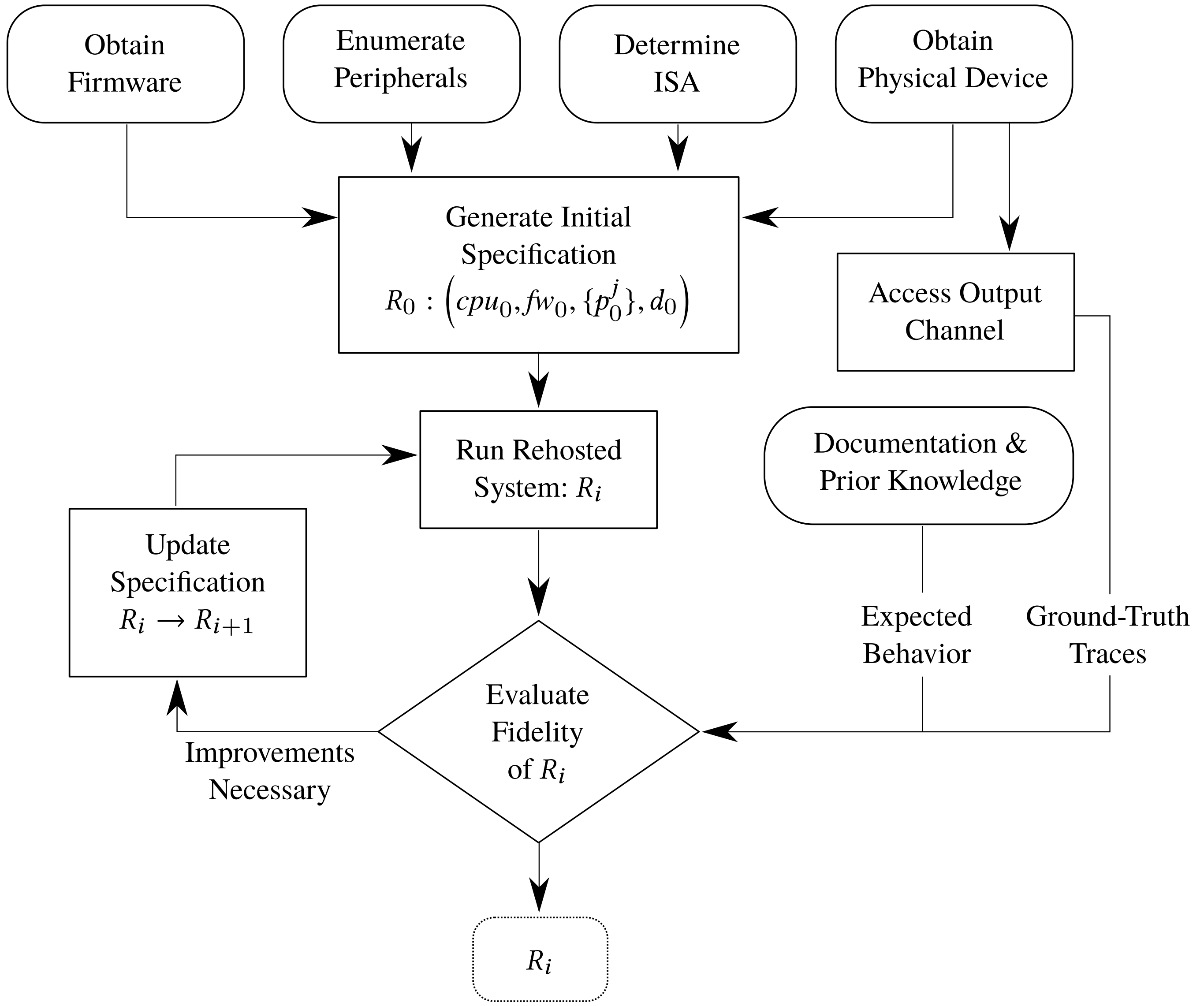
KEYWORDS
Dynamic program analysis; firmware security; emulation; embedded systems; internet of things; virtualization; rehosting

ACM Reference Format:
Andrew Fasano, Tiemoko Ballo, Marius Muench, Tim Leek, Alexander Bulekov, Brendan Dolan-Gavitt, Manuel Egele, Aurélien Francillon, Long Lu, Nick Gregory, Davide Balzarotti, and William Robertson. 2021. SoK: Enabling Security Analyses of Embedded Systems via Rehosting . In Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security (ASIA CCS '21), June 7–11, 2021, Hong Kong, Hong Kong. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3433210.3453093

1 INTRODUCTION
Whereas enterprise edge systems are typically maintained by IT professionals, the rise of low-cost, consumer edge devices ("Internet of Things") has led to an increasingly large number of Internet-accessible machines which malicious actors can exploit. In 2016, the Mirai malware exploited insecure default credentials on many of these machines to create a botnet of approximately 600,000 devices [1]. But poor security posture is a problem emblematic of many embedded systems, not just consumer edge devices. Despite the frequent use of embedded systems for safety-critical, industrial applications, 90% of embedded real-time operating systems

Introduces the benefits and challenges of of virtualizing embedded software

Identifies the essential steps in the rehosting process and a high-level, iterative process for rehosting embedded systems.



Saphire: Sandboxing PHP Applications with Tailored System Call Allowlists


Saphire: Sandboxing PHP Applications with Tailored System Call Allowlists

Alexander Bulekov* Rasoul Jahanshahi* Manuel Egele
*Equal contribution joint first authors
Boston University
{alxndr,rasoulj,megele}@bu.edu

Abstract

1 Introduction

Interpreted languages, such as PHP and JavaScript, are the foundation of modern-day computing. This is particularly true for the web, where online social networks, eCommerce, and online news attract the attention of billions of daily users. The ensuing swaths of personal, financial, and otherwise sensitive information held by these entities, make web sites attractive targets for cyber attacks. Beyond localized leaks of information, web apps and the interpreted languages that power them have also been at the core of data breaches that affect society at large. In 2015 attackers allegedly leveraged vulnerabilities in plugins of the WordPress and Drupal web apps to leak what has become known as the "Panama Papers" [36]. As testament to this crisis, Symantec reports [10] that in 2017, one in every 13 web requests was malicious. What exacerbates the situation is that, according to W3Techs [43], nine out of ten most popular web-development languages are interpreted. Furthermore, 2017 saw a 400% year-over-year increase [24] of reported vulnerabilities in the top four most popular content management systems. All four are interpreted web apps and have attracted significant attention from attackers.

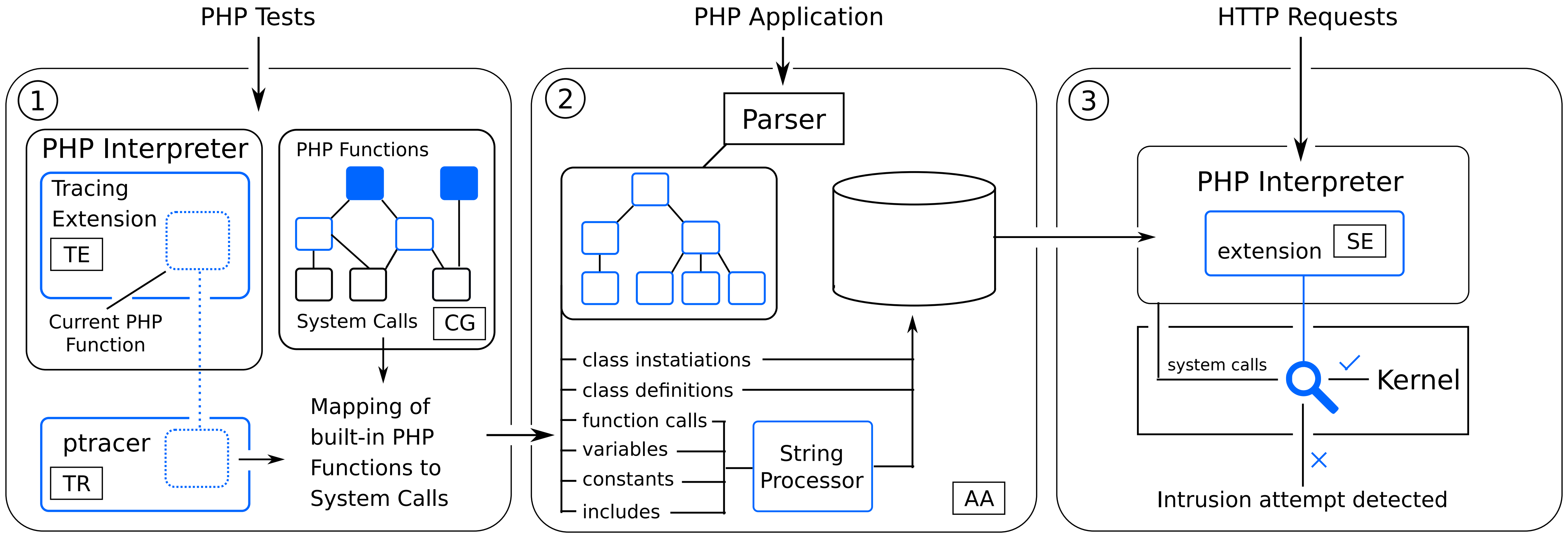
Arbitrary code-executions(ACE) are the most dangerous class of application vulnerabilities, as they allow an attacker to take complete control over the running application. The root issue that makes ACE so hazardous is the fact that modern interpreted applications do not adhere to the principle of least privilege (PoLP) [46]. An attacker's exploit executes with ambient authority and is constrained only by the operating systems' access control mechanisms.

Some at-risk projects have recognized this problem and taken steps to intentionally reduce the run-time privileges of their software. By relinquishing access to unneeded system resources and API's, the software reduces the potential impact of a vulnerability. This practice has been widely adopted by native applications such as Chrome, Firefox, Tor, OEM… and OpenSSH, but it is not in common use … applications. The reason for this is … introduce a layer of abs…

Three stage approach to automatically protect a vulnerable-web application against exploitation.

Leverages, seccomp, a built-in feature of the kernel

Blocked every exploit in our dataset



Fasano, Andrew, Tiemoko Ballo, Marius Muench, Tim Leek, Alexander Bulekov, Brendan Dolan-Gavitt, Manuel Egele et al. "Sok: Enabling security analyses of embedded systems via rehosting." In Proceedings of the 2021 ACM Asia conference on computer and communications security.

Bulekov, Alexander, Rasoul Jahanshahi, and Manuel Egele. "Saphire: Sandboxing {PHP} applications with tailored system call allowlists." In USENIX Security Symposium (USENIX Security 21)

Bulekov, Alexander, Bandan Das, Stefan Hajnoczi, and Manuel Egele. "MORPHUZZ: Bending (input) space to fuzz virtual devices." In USENIX Security Symposium (USENIX Security 22)

Bulekov, Alexander, Bandan Das, Stefan Hajnoczi, and Manuel Egele. "No grammar, no problem: Towards fuzzing the linux kernel without system-call descriptions." In Network and Distributed System Security (NDSS) Symposium 2023

Bulekov, Alexander, Qiang Liu, Manuel Egele, and Mathias Payer Hyperpill: Fuzzing for hypervisor-bugs by leveraging the hardware virtualization interface (to appear). In USENIX Security Symposium (USENIX Security 24)

Thank you

Prof. Manuel Egele Prof. Gianluca Stringhini

Prof. Orran Krieger Prof. Mathias Payer

Thank you

Prof. Manuel Egele Prof. Gianluca Stringhini

Prof. Orran Krieger Prof. Mathias Payer

Qiang Liu
Rasoul Jahanshahi
Stefan Hajnoczi
Bandan Das
Andrew Fasano
Tiemoko Ballo
Jeremy Liu
Darren Kenny
Marius Muench
Tim Leek
Brendan Dolan-Gavitt
William Robertson
Aurélien Francillon
Long Lu
Davide Balzarotti
Nick Gregory

Thank you

Prof. Manuel Egele Prof. Gianluca Stringhini

Prof. Orran Krieger Prof. Mathias Payer

Qiang Liu	Rasoul Jahanshahi	Stefan Hajnoczi	Bandan Das
Andrew Fasano	Tiemoko Ballo	Jeremy Liu	Darren Kenny
Marius Muench	Tim Leek	Brendan Dolan-Gavitt	William Robertson
Aurélien Francillon	Long Lu	Davide Balzarotti	Nick Gregory

SeclaBU

Thank you

Prof. Manuel Egele Prof. Gianluca Stringhini

Prof. Orran Krieger Prof. Mathias Payer

Qiang Liu	Rasoul Jahanshahi	Stefan Hajnoczi	Bandan Das
Andrew Fasano	Tiemoko Ballo	Jeremy Liu	Darren Kenny
Marius Muench	Tim Leek	Brendan Dolan-Gavitt	William Robertson
Aurélien Francillon	Long Lu	Davide Balzarotti	Nick Gregory

SeclaBU

Thank you

Prof. Manuel Egele Prof. Gianluca Stringhini

Prof. Orran Krieger Prof. Mathias Payer

Qiang Liu	Rasoul Jahanshahi	Stefan Hajnoczi	Bandan Das
Andrew Fasano	Tiemoko Ballo	Jeremy Liu	Darren Kenny
Marius Muench	Tim Leek	Brendan Dolan-Gavitt	William Robertson
Aurélien Francillon	Long Lu	Davide Balzarotti	Nick Gregory

Red Hat Research

BU RCS

SeclaBU

Thank you

Prof. Manuel Egele Prof. Gianluca Stringhini

Prof. Orran Krieger Prof. Mathias Payer

Qiang Liu, Rasoul Jahanshahi, Stefan Hajnoczi, Bandan Das

Andrew Fasano, Tiemoko Ballo, Jeremy Liu, Darren Kenny

Marius Muench, Tim Leek, Brendan Dolan-Gavitt, William Robertson

Aurélien Francillon, Long Lu, Davide Balzarotti, Nick Gregory

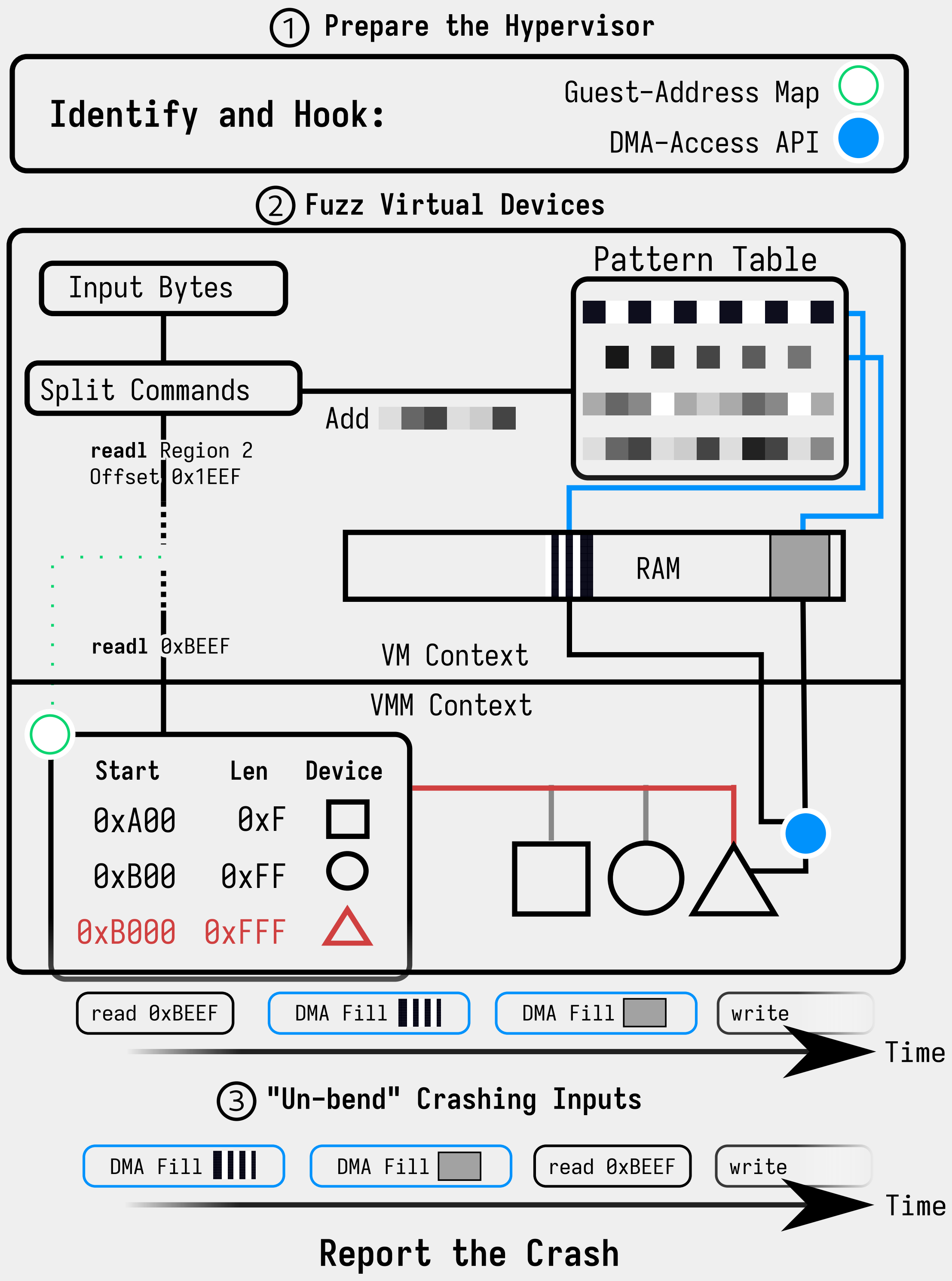
Red Hat Research

SeclaBU

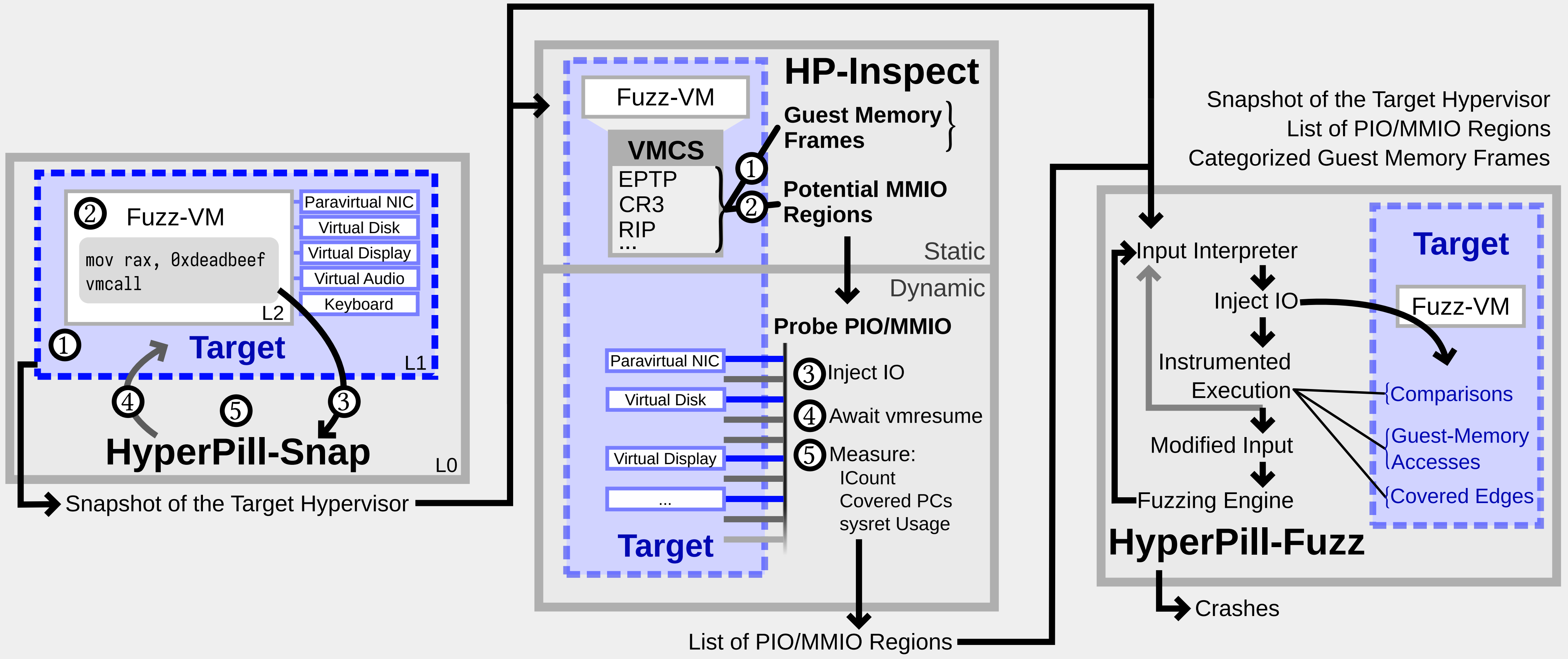
My Family

BU RCS

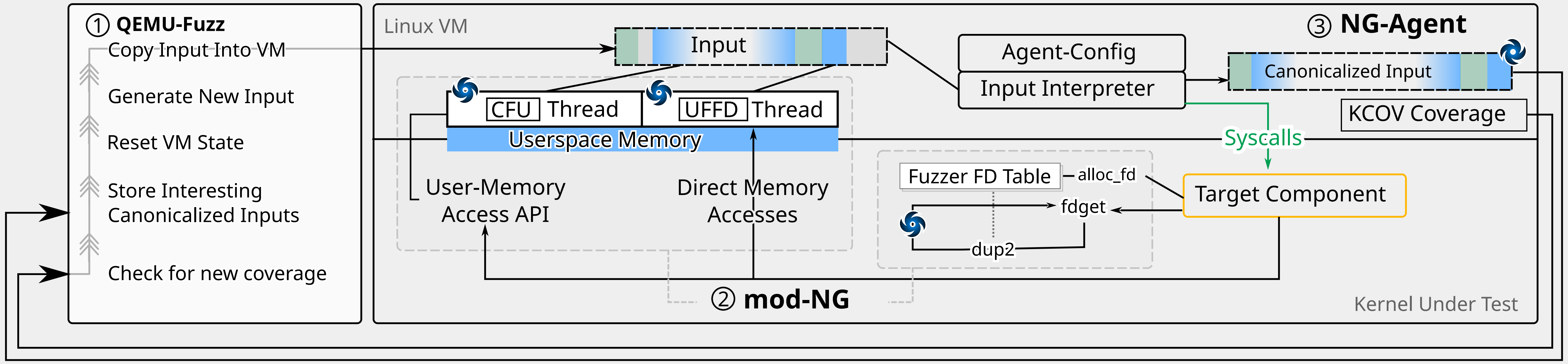
Thank you!



MORPHUZZ



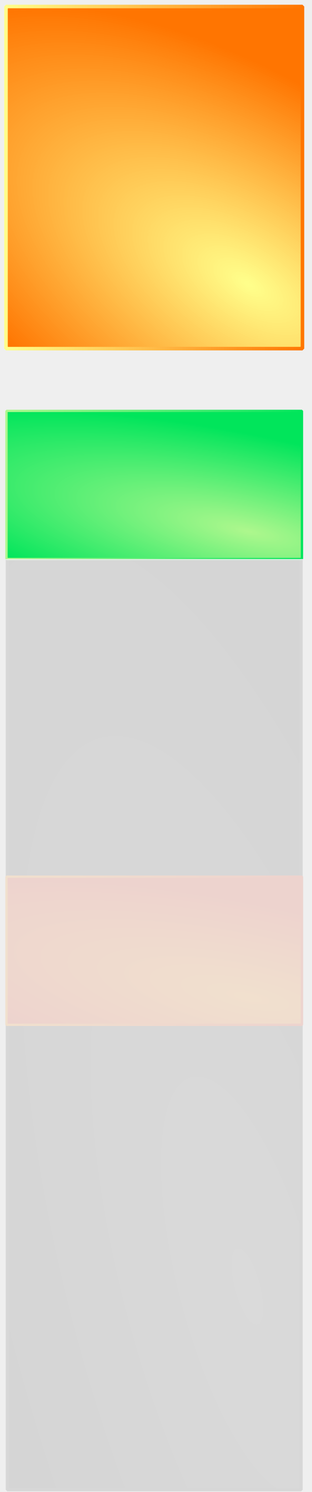
HyperPill



FuzzNG

Prior Approaches

Prior Approaches

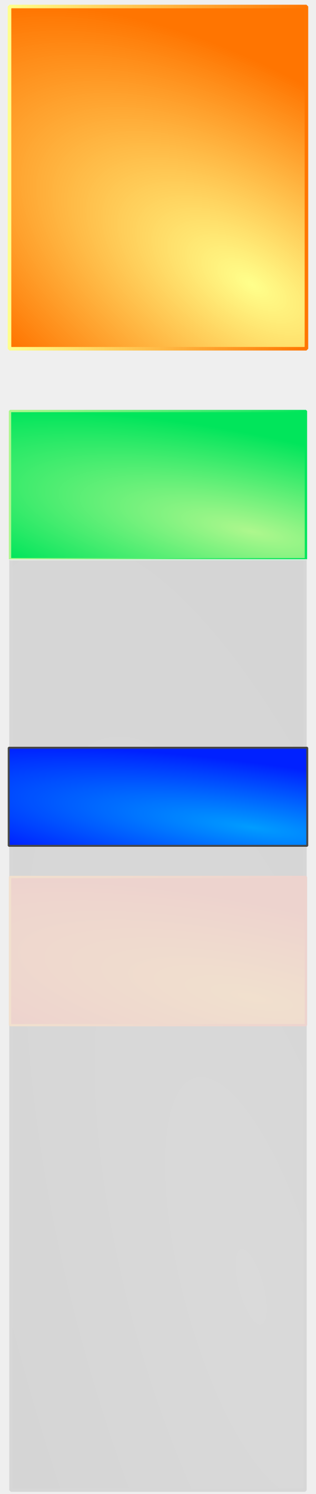


Ignore DMA

Tavis Ormandy. **IOFuzz** (2007)

Henderson et al. **VDF** (Raid ‘17)

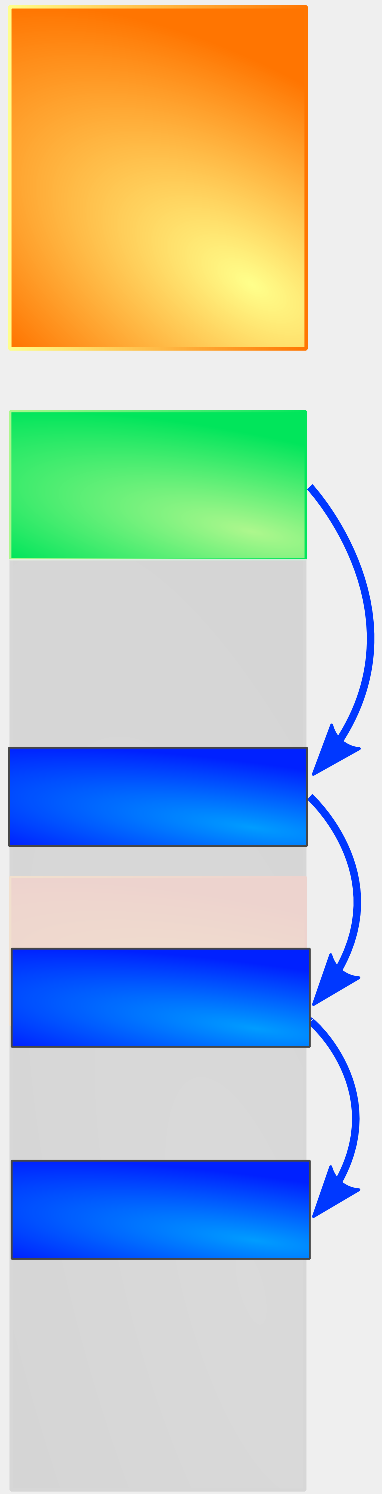
Prior Approaches



Scratch-Buffer to Stimulate DMA

Schumilo et al. **HYPER-CUBE** (NDSS '20)
Schumilo et al. **Nyx** (Sec '21)

Prior Approaches



Per-Device Grammars

Schumilo et al. **Nyx** (Sec ’21)

Grammars - Virtual Devices

eXtensible Host Controller Interface for Universal Serial Bus

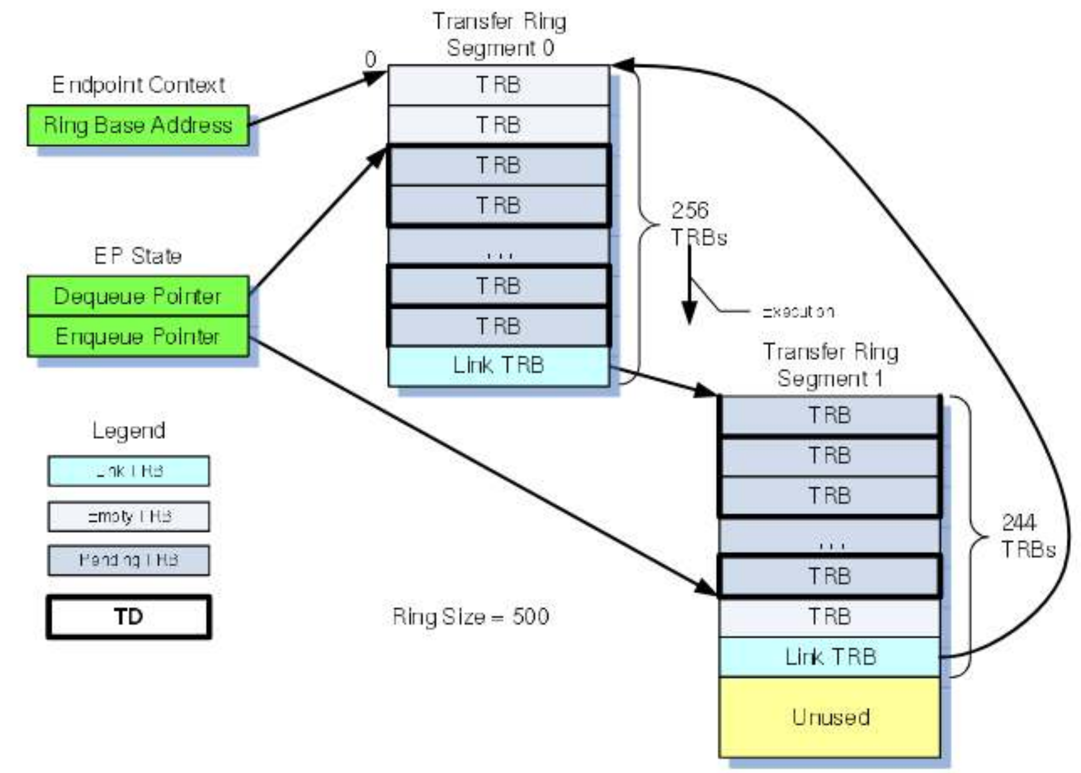
(xHCI)

Requirements Specification

May 2019

Revision 1.2

Figure 4-15: Link TRB Example



Transfer Descriptors (Chained TRBs) may cross Segment boundaries.

Refer to section 4.11.7 for how the Chain (CH) flag shall be set in a Link TRB. In a Transfer Ring a Link TRB is always assumed to be linked to the first TRB of the next segment. If the Chain bit (CH) of the previous TRB is '1', then the multi-TRB TD that it defines spans segments and shall continue with the first TRB of the next segment. In a Command Ring the Link TRB Chain bit (CH) is ignored by the xHC.

As software advances its Enqueue Pointer and advances over a Link TRB, the Cycle (C) bit shall be updated with the value of the PCS flag.

The Interrupt On Completion (IOC) flag of a Link TRB may be used by system software to generate an event indicating the Dequeue Pointer has reached the Link TRB. This feature provides software with the ability to track the Dequeue Pointer as a function of segment boundary crossings.

Note: A TD Fragment shall not span segments. Refer to section 4.11.7.1.

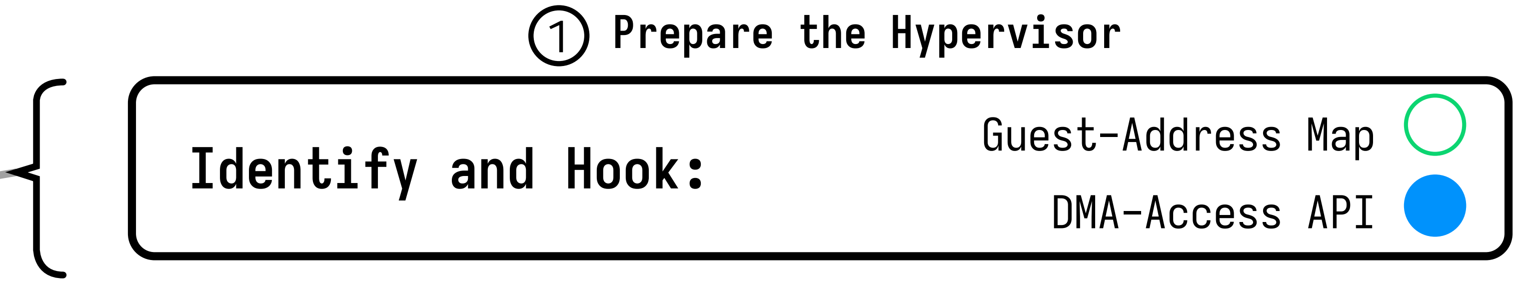
When the Link TRB resides on a Transfer Ring the Interrupt On Completion (IOC) flag of a Link TRB may be used by system software to generate a Transfer Event, where the Transfer Event Slot ID and Endpoint ID shall reflect the slot and

229

Grammars - Virtual Devices

```
t_dcb_ctx = s.edge_type("dcb_ctx", c_type= "uint32_t")
new_dcb_ctx=" *output_0 = (uint32_t) slab_alloc_page_aligend(0x1000);\n" +\
    "uint32_t* dcbaa = (uint32_t* )*borrow_0;\n"+\
    "dcbaa[*data_slot] = *output_0;\n"

s.node_type("new_dcb_ctx", outputs=[t_dcb_ctx], borrows=[t_dcbaa], data=s.data_u8("slot"), codegen_args=new_dcb_ctx)

new_dcb_ctx_broken_1="*((uint32_t**)borrow_0)[*data_u8] = (uint32_t)0x%x;\n"%(base)
s.node_type("new_dcb_ctx_broken_1", borrows=[t_dcbaa], data=d_u8, codegen_args=new_dcb_ctx_broken_1)

new_dcb_ctx_broken_2=" *((uint32_t**)borrow_0)[*data_u8] = (uint32_t)0xFFFFFFFFFFFFFF00;\n"
s.node_type("new_dcb_ctx_broken_2", borrows=[t_dcbaa], data=d_u8, codegen_args=new_dcb_ctx_broken_2)


d_slot_context = s.data_struct("d_slot_context")
d_slot_context.u32("data_a")
d_slot_context.u32("data_b")
d_slot_context.u32("data_c")
d_slot_context.u32("data_d")
d_slot_context.finalize()

new_slot_context = "" +\
  "*((uint32_t*)(*borrow_0 + 0)) = (uint32_t)data_d_slot_context->data_a;\n"+\
  "*((uint32_t*)(*borrow_0 + 4)) = (uint32_t)data_d_slot_context->data_b;\n"+\
  "*((uint32_t*)(*borrow_0 + 8)) = (uint32_t)data_d_slot_context->data_c;\n"+\
  "*((uint32_t*)(*borrow_0 + 12)) = (uint32_t)data_d_slot_context->data_d;\n"
s.node_type("new_slot_context", borrows=[t_dcb_ctx], data=d_slot_context, codegen_args=new_slot_context)

d_slot_context = s.data_struct("d_ep_context")
d_slot_context.u32("data_a")
d_slot_context.u32("data_b")
d_slot_context.u32("data_c")
d_slot_context.u8("ep_identifier")
d_slot_context.finalize()

new_ep_context = "" +\
  "*((uint32_t*)(*borrow_0 + 0  + (0x20 * (1+(uint32_t)data_d_ep_context->ep_identifier%0x30)))) = (uint32_t)data_d_ep_context->data_a;\n"+\
  "*((uint32_t*)(*borrow_0 + 4  + (0x20 * (1+(uint32_t)data_d_ep_context->ep_identifier%0x30)))) = (uint32_t)data_d_ep_context->data_b;\n"+\
  "*((uint32_t*)(*borrow_0 + 16 + (0x20 * (1+(uint32_t)data_d_ep_context->ep_identifier%0x30)))) = (uint32_t)data_d_ep_context->data_c;\n"+\
  "*((uint32_t*)(*borrow_0 + 8  + (0x20 * (1+(uint32_t)data_d_ep_context->ep_identifier%0x30)))) = *borrow_1;\n" + \
  "*((uint32_t*)(*borrow_0 + 12 + (0x20 * (1+(uint32_t)data_d_ep_context->ep_identifier%0x30)))) = 0;\n"
s.node_type("new_ep_context", borrows=[t_dcb_ctx, t_trb_ring], data=d_slot_context, codegen_args=new_ep_context)
```

Morphuzz Components



① Hypervisor Hooks

② Virtual Device Fuzzer Harness

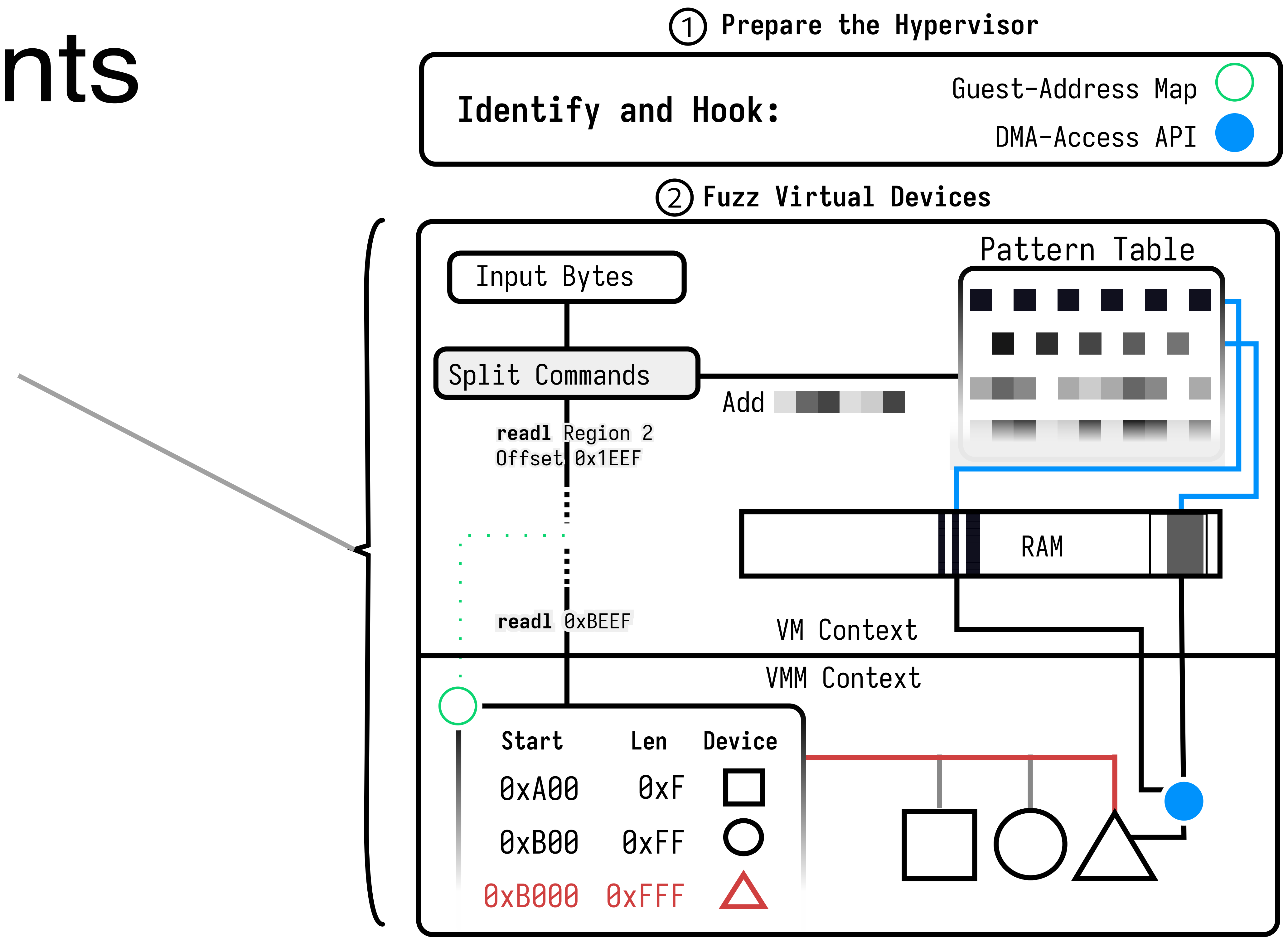
- libFuzzer
- Input interpreter
- We added a fork-server

③ Crash "Unbending"

Transform DMA back into an asynchronous operation

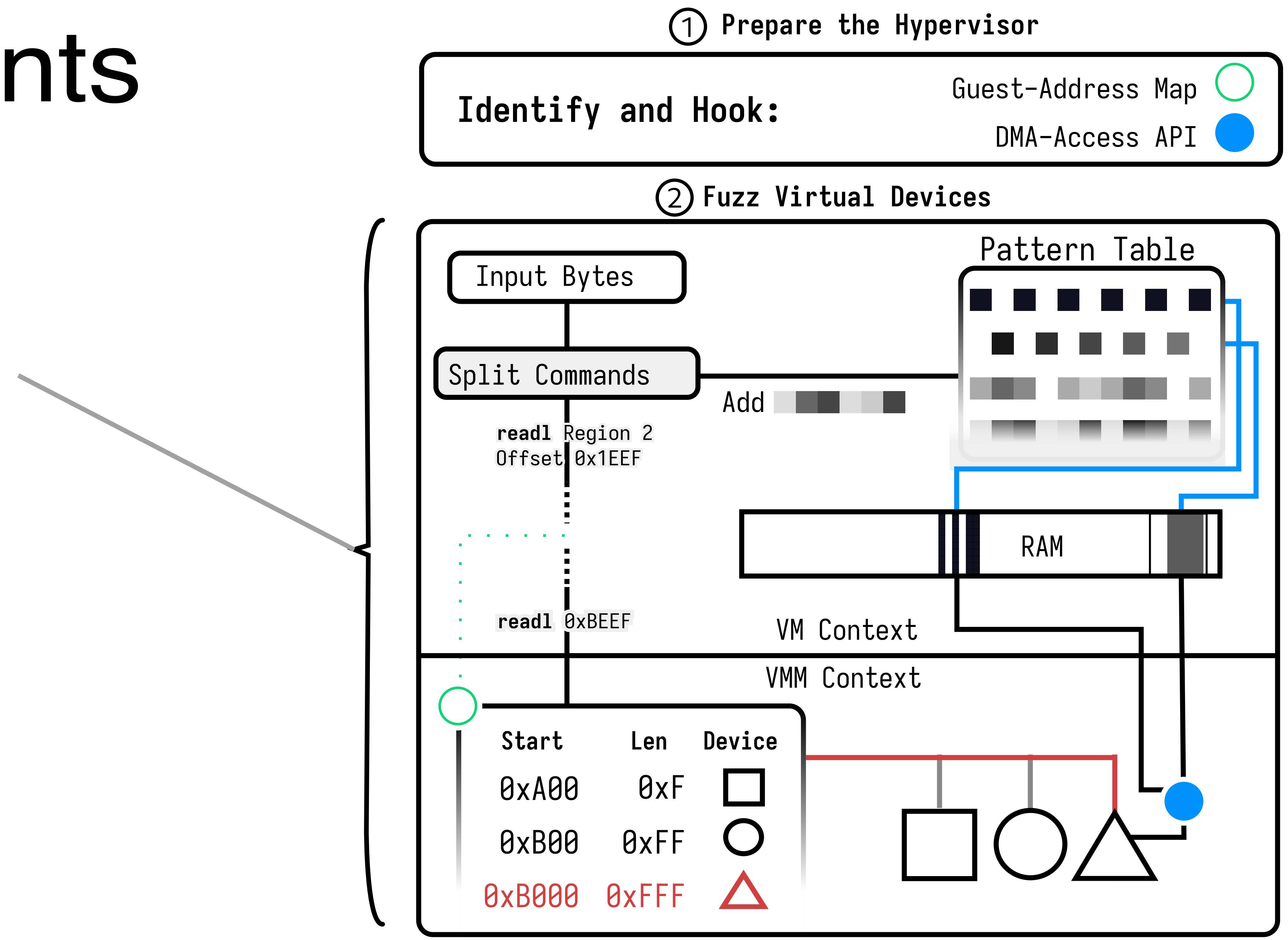
Morphuzz Components

① Hypervisor Hooks



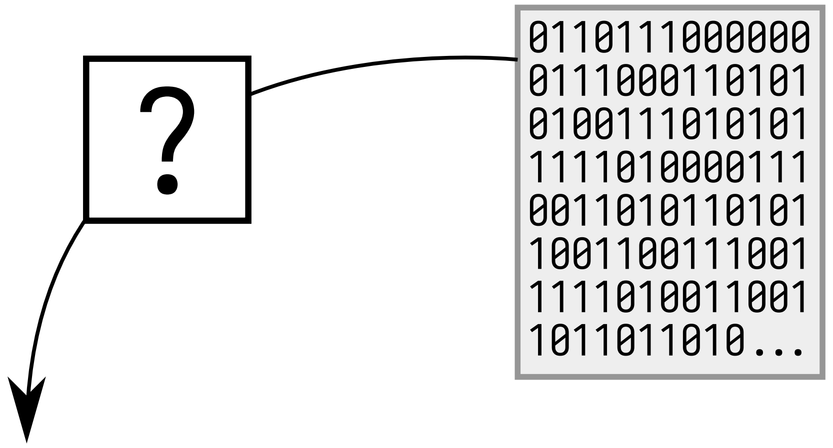
Morphuzz Components

① Hypervisor Hooks

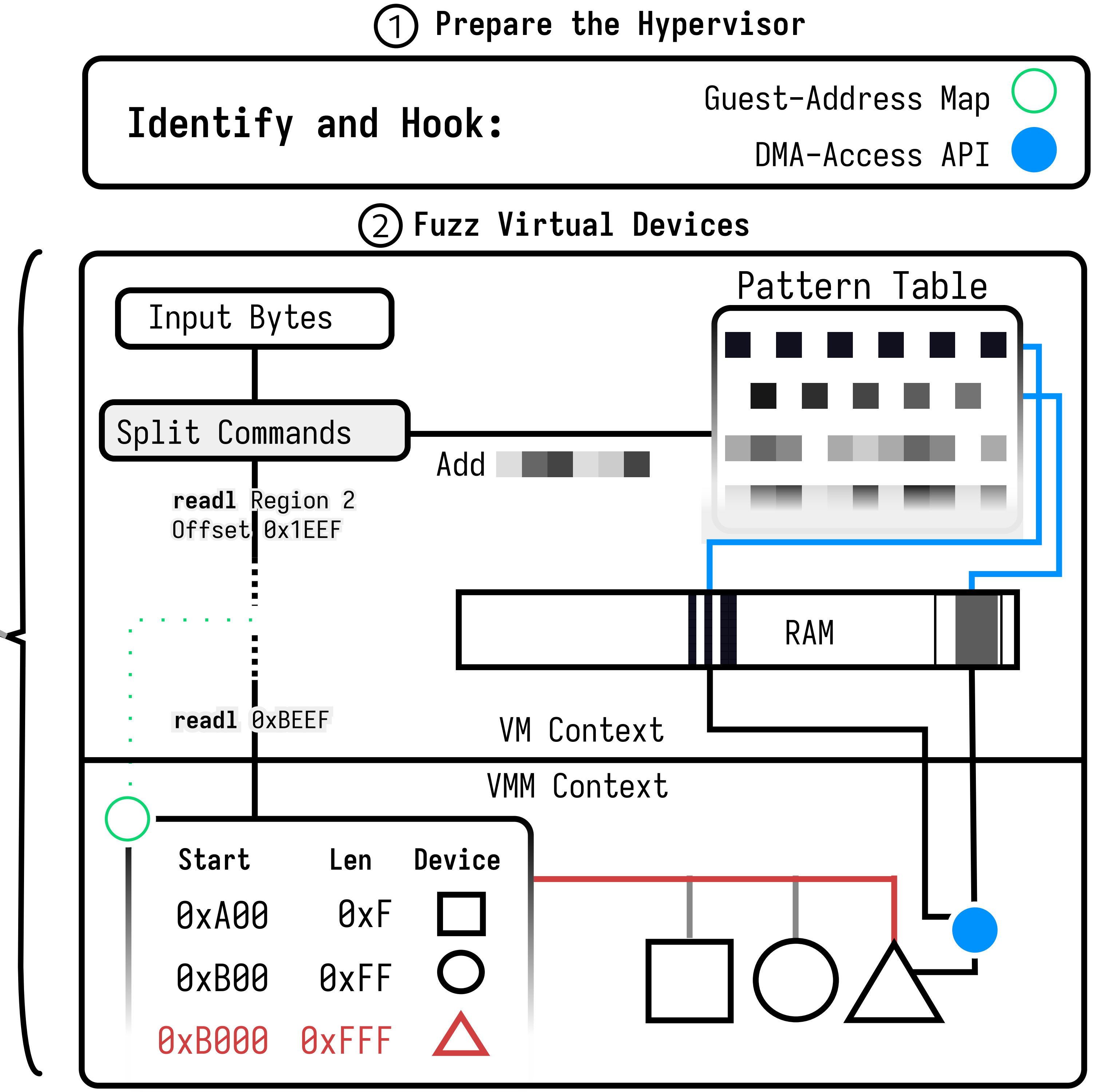


Morphuzz Components

① Hypervisor Hooks

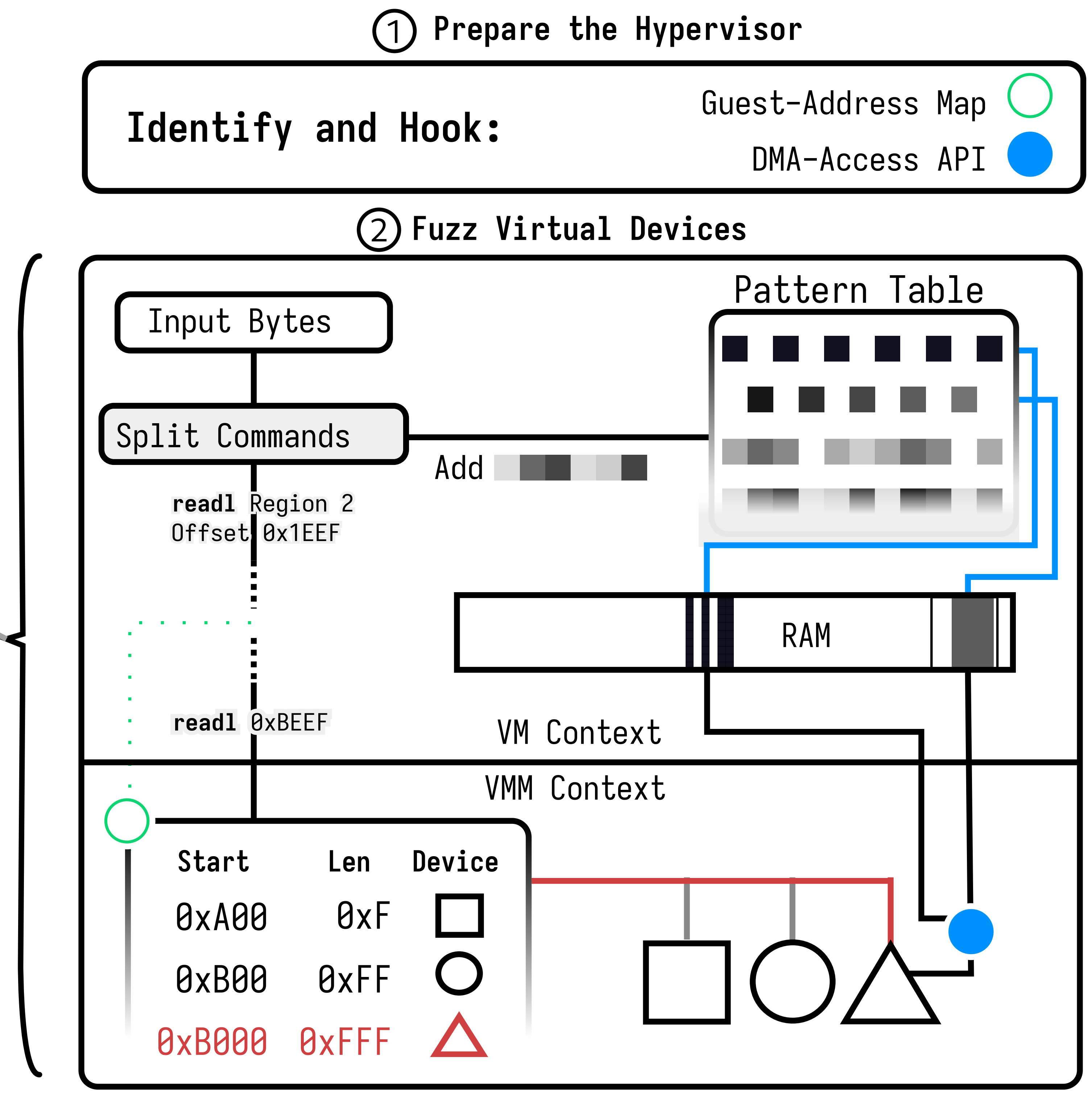



```
01 outl 0xcf8 0x80001010
02 outl 0xcfc 0xc0200
03 outl 0xcf8 0x80001004
04 outl 0xcfc 0x1c77695e
```



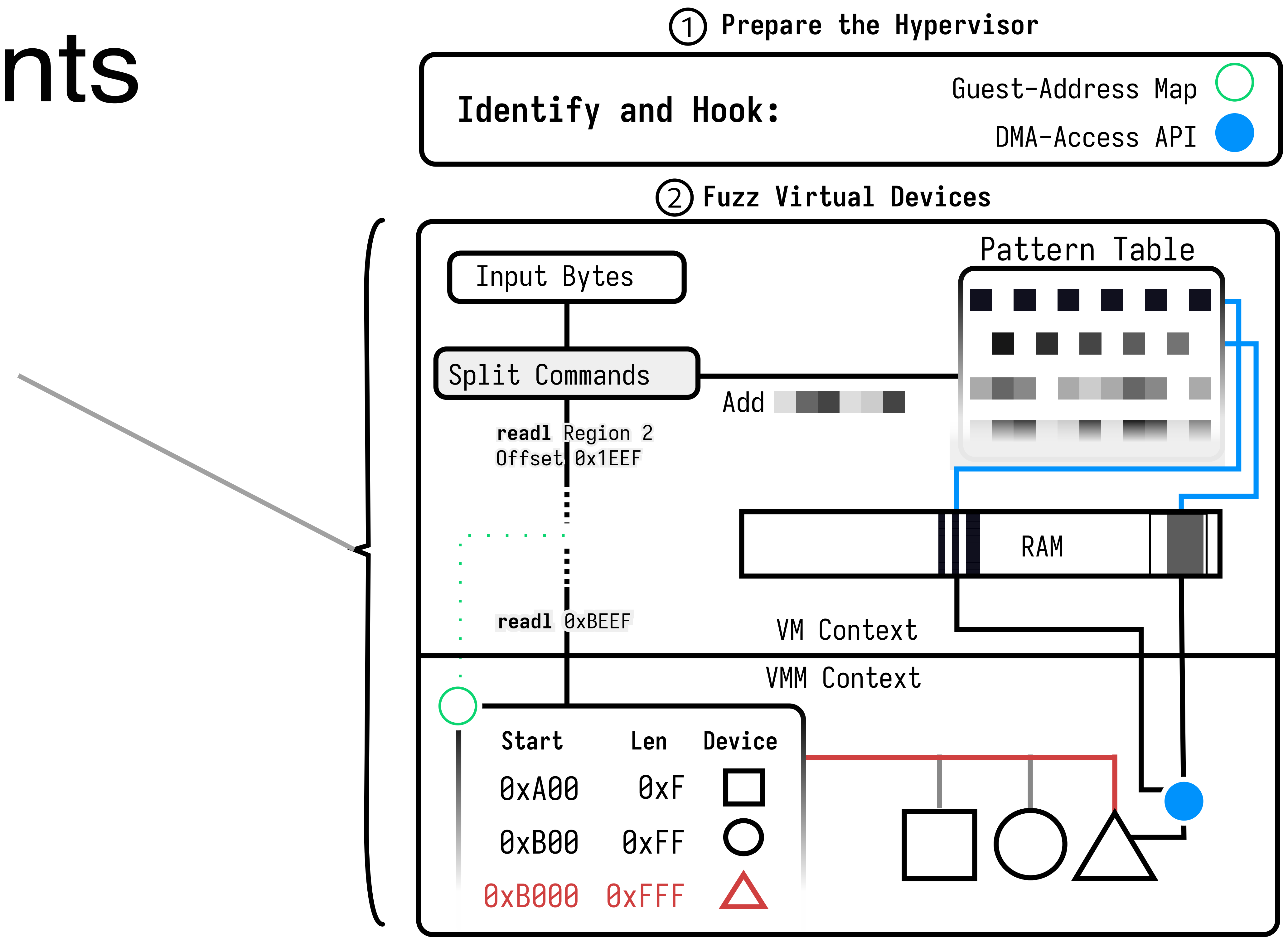
Morphuzz Components

① Hypervisor Hooks



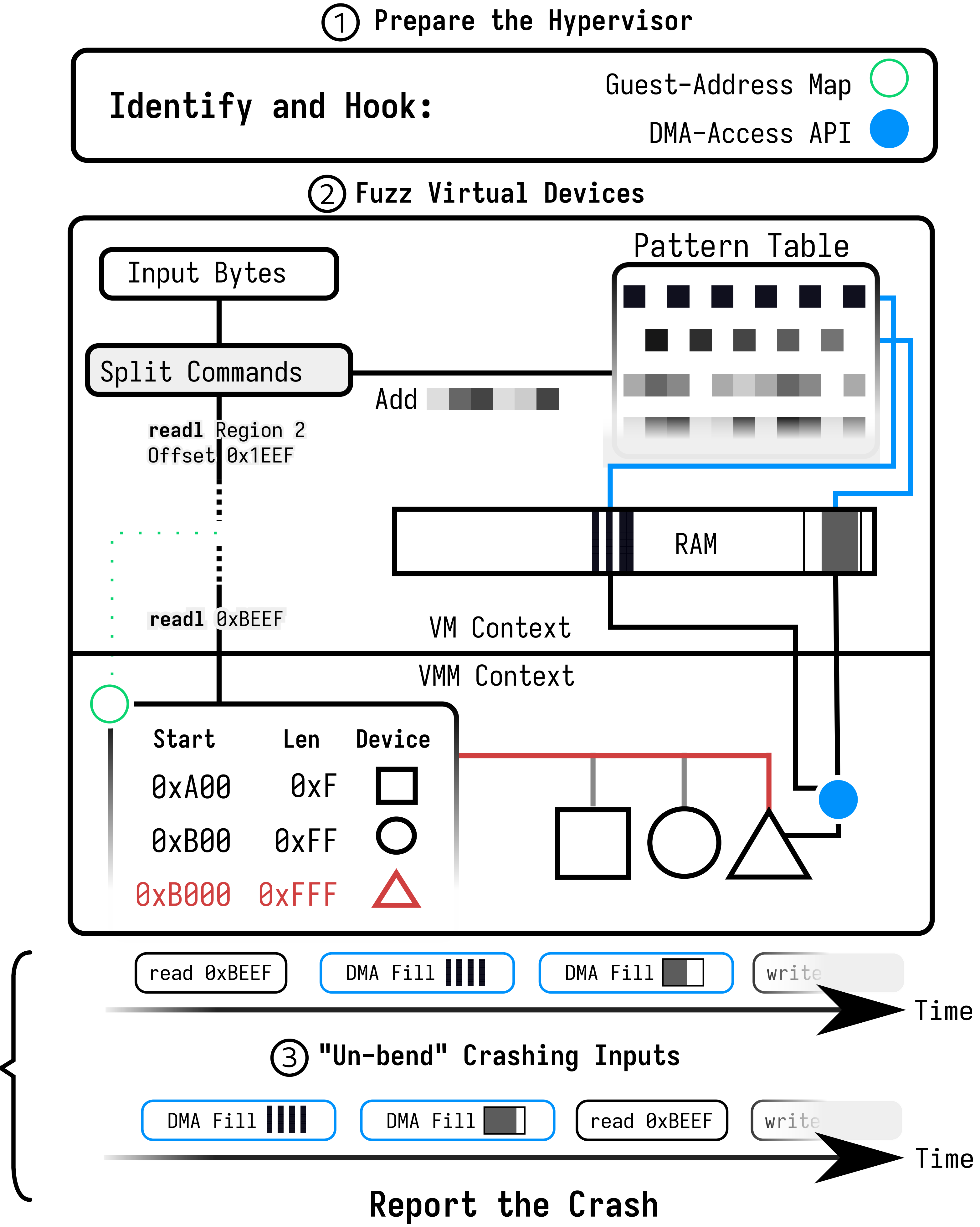
Morphuzz Components

① Hypervisor Hooks



Morphuzz Components

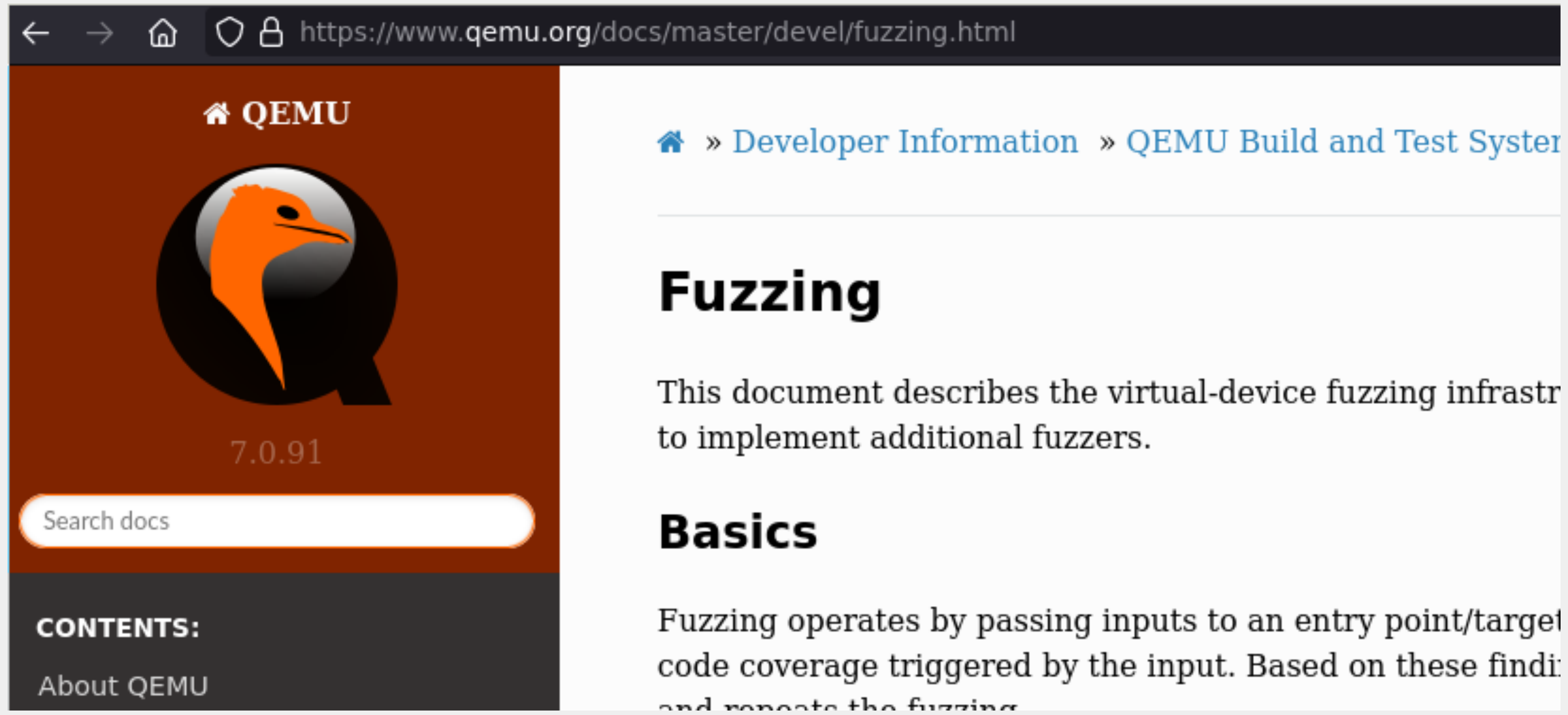
① Hypervisor Hooks



Morphuzz is Upstream in QEMU

- Continuously fuzzed on OSS-Fuzz
- 200+ Issues Reported
- Reproducers are simple to use
- Bugs are caught before release

```
git clone git.qemu.org/qemu.git
```




```
commit 283f0a05e24a5e5fab78305f783f06215390d620
Author: Thomas Huth <thuth@redhat.com>
Date:   Thu Jul 15 21:32:19 2021 +0200

    hw/net/net_tx_pkt: Fix crash detected by fuzzer

    QEMU currently crashes when it's started like this:

    cat << EOF | ./qemu-system-i386 -device vmxnet3 -nodefaults -qtest stdio
    outl 0xcf8 0x80001014
    outl 0xcfc 0xe0001000
    outl 0xcf8 0x80001018
    outl 0xcf8 0x80001004
    outw 0xcfc 0x7
    outl 0xcf8 0x80001083
    write 0x0 0x1 0xe1
    write 0x1 0x1 0xfe
    write 0x2 0x1 0xbe
    write 0x3 0x1 0xba
    writeq 0xe0001020 0xefefff5ecafe0000
    writeq 0xe0001020 0xffff5e5ccafe0002
    EOF

    It hits this assertion:

    qemu-system-i386: ../qemu/hw/net/net_tx_pkt.c:453: net_tx_pkt_reset:
     Assertion `pkt->raw' failed.

    This happens because net_tx_pkt_init() is called with max_frags == 0 and
    thus the allocation
```