2550 Intro to systems security

L15

abhi shelat

What happens after boot?

The kernel starts exactly one program. Every other process descends from it.

```
Ubuntu 24.04 LTS nuc tty1

nuc login: alice
Password:
alice@nuc:~$ pstree -p
systemd(1)─┬─systemd-journal(298)
           ├─systemd-udevd(341)
           ├─NetworkManager(702)
           ├─sshd(905)
           └─login(788)───bash(1402)───pstree(1530)
```

Every command you run is a child of your shell, which starts from PID 1.

1. **Kernel finishes booting**
   Memory protection, user mode and the timer are now switched on

2. **Kernel starts PID 1: systemd**
   The first user-space program; it starts all the others

3. **systemd starts agetty, sushi**
   agetty prints the “login:” prompt on the console
   sshd allows remote login

4. **login checks your password**
   Then it switches to your user ID and starts your shell

What is the OS for?



Referee

Shares the CPU, memory and devices between programs, and stops them from harming each other

One program can't read another's passwords



Illusionist

Makes each program feel like it has the whole machine: its own CPU, its own memory

300 programs share 12 cores



Glue

Provides common services, so every program doesn't reinvent them

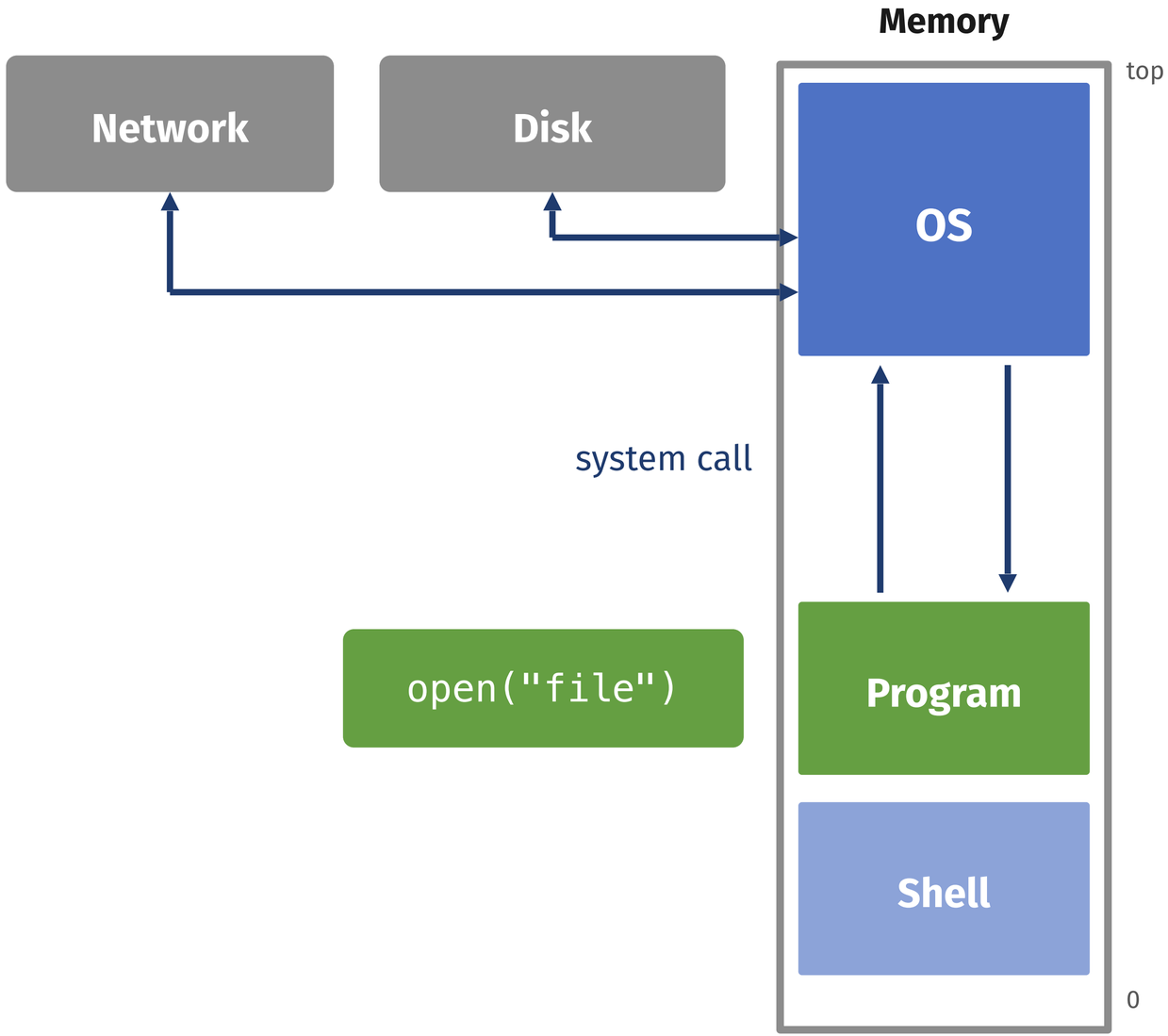
Files, networking, windows

The shell is not the OS. It's an ordinary program that uses these same services.

Framing from Anderson & Dahlin, Operating Systems: Principles and Practice.

The system model

- The OS reserves a region of memory for itself
- The shell is just another program
- Programs ask the OS for services through system calls, like open("file")
- On DOS this was optional: programs often skipped the OS and drove the hardware directly



What if nothing enforces the rules?

That’s how the CPU wakes up: real mode, no protection. DOS ran like this.

Three ways one program can hurt all the others:



Memory

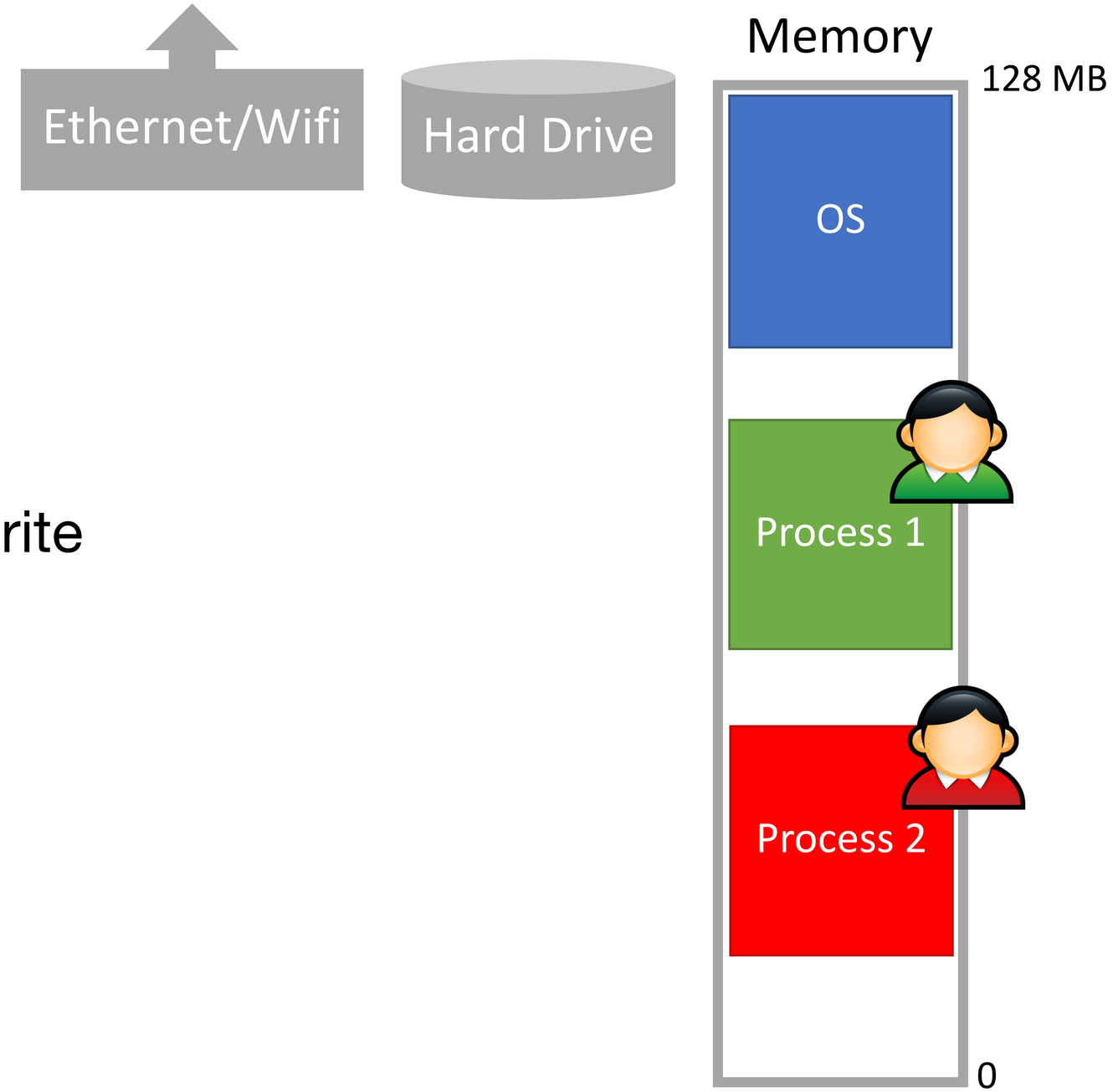


Devices



CPU time

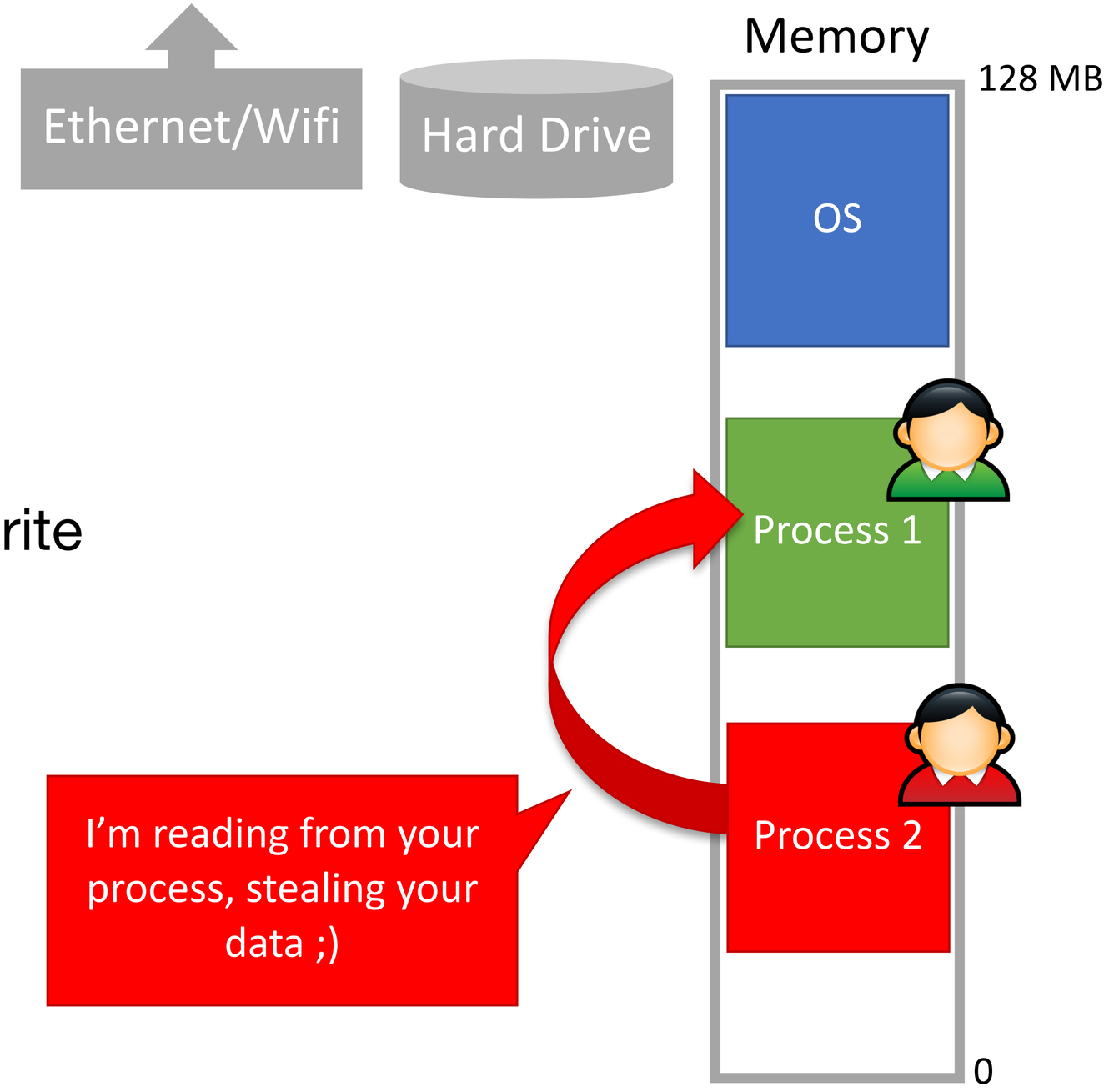
Memory Unsafety



Problem: any process can read/write any memory

Memory Unsafety

Problem: any process can read/write any memory



Problem 1: anyone can touch any memory

Malice

A malicious program can steal other programs' secrets, or plant its own code inside the OS.

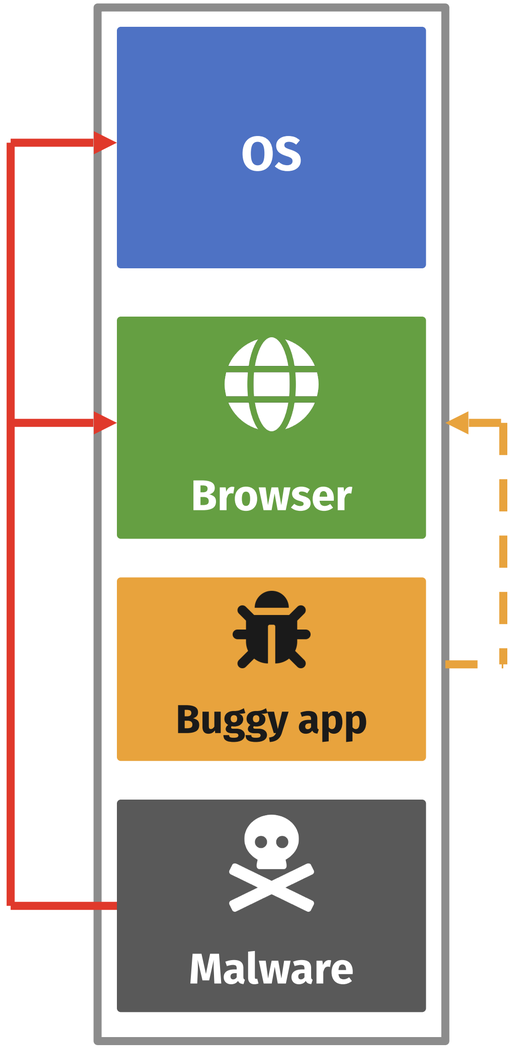
Bugs

A buggy program can corrupt another program or crash the whole machine. The crash shows up far from the cause, which makes it very hard to debug.

Overwrite the OS with malicious code

Scan the browser for passwords and card numbers

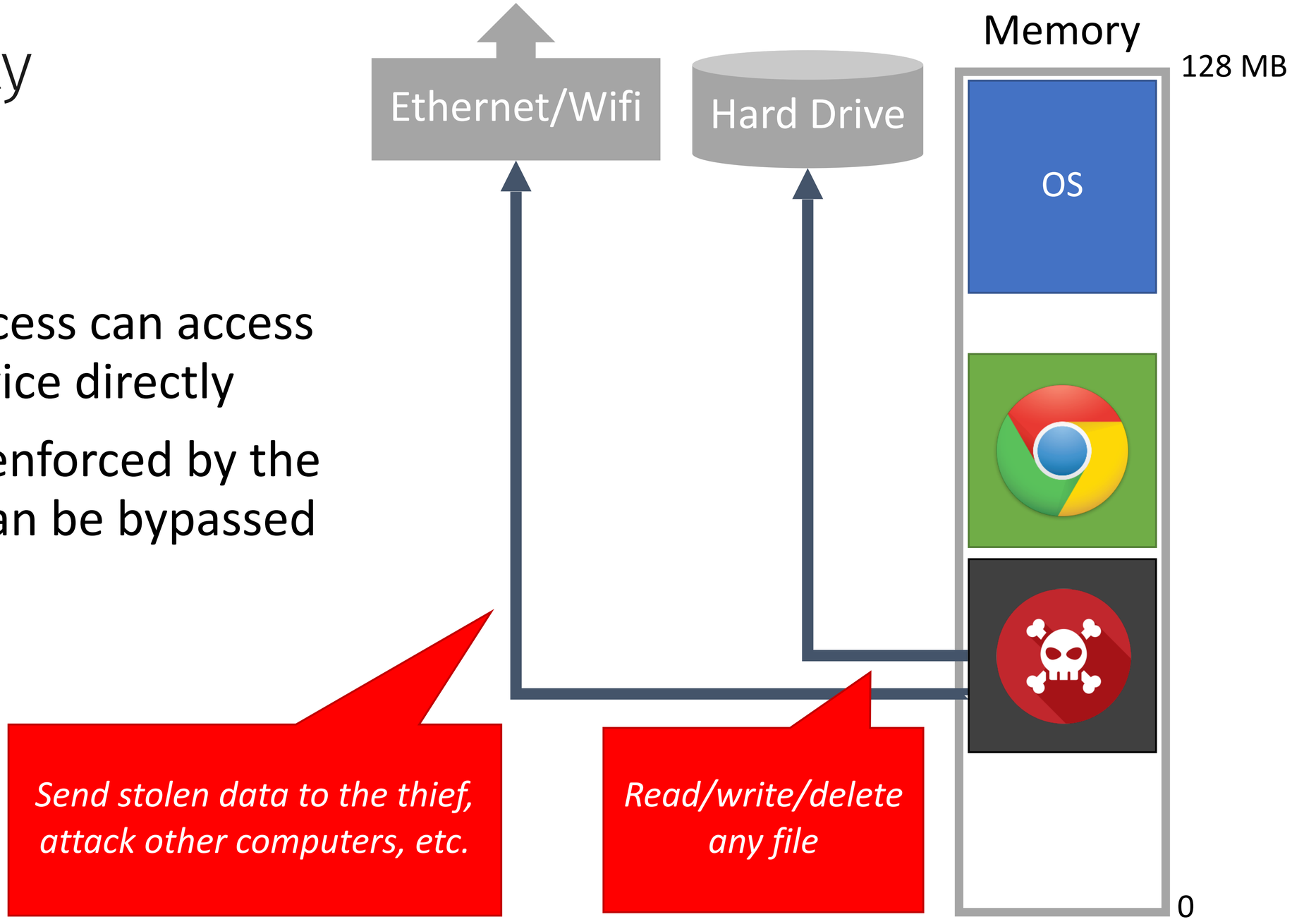
Oops: a stray pointer overwrites the browser's data



Device Unsafety

Problem: any process can access any hardware device directly

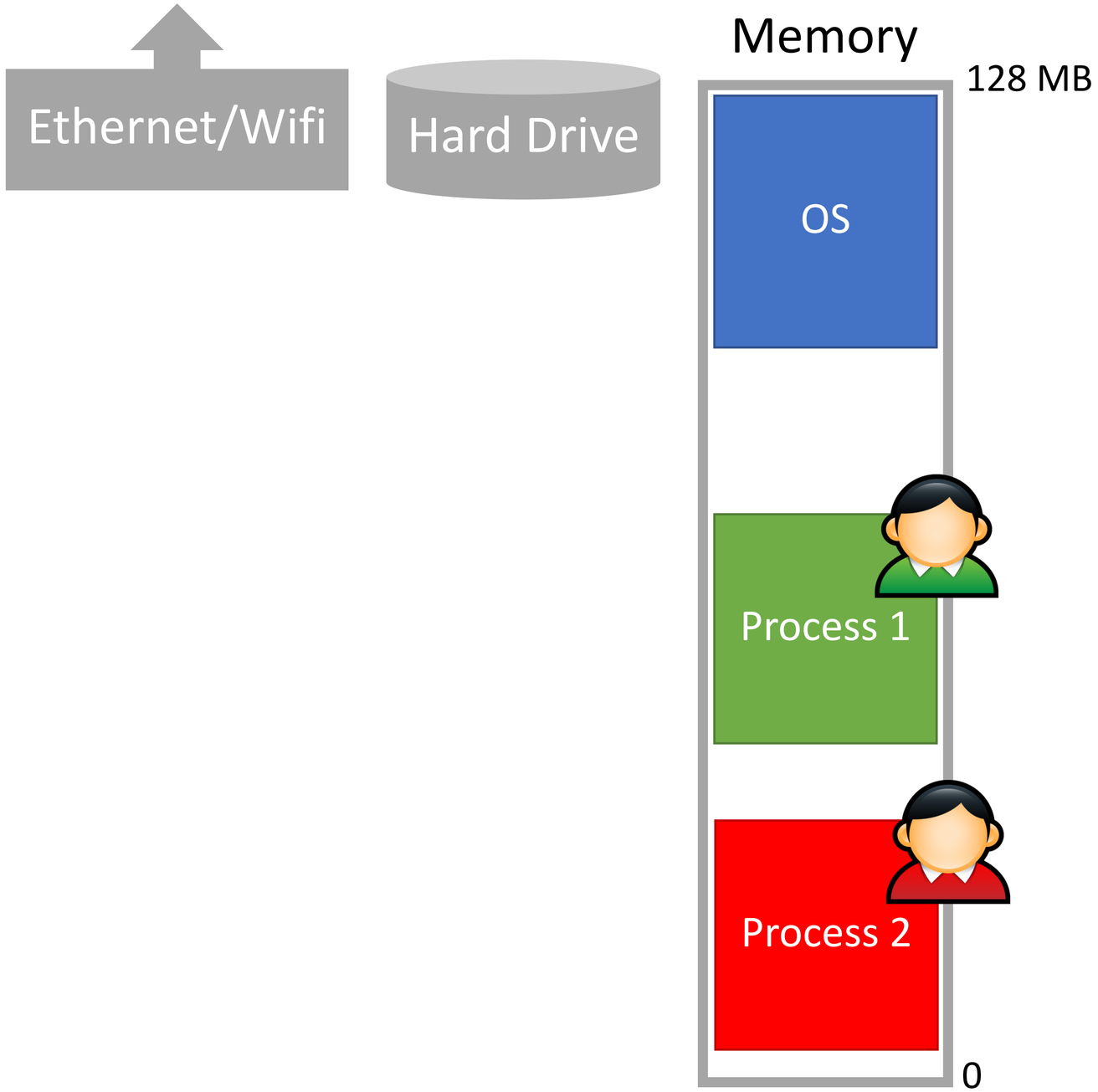
Access control is enforced by the OS, but OS APIs can be bypassed



Device Unsafety

Problem: any process can access any hardware device directly

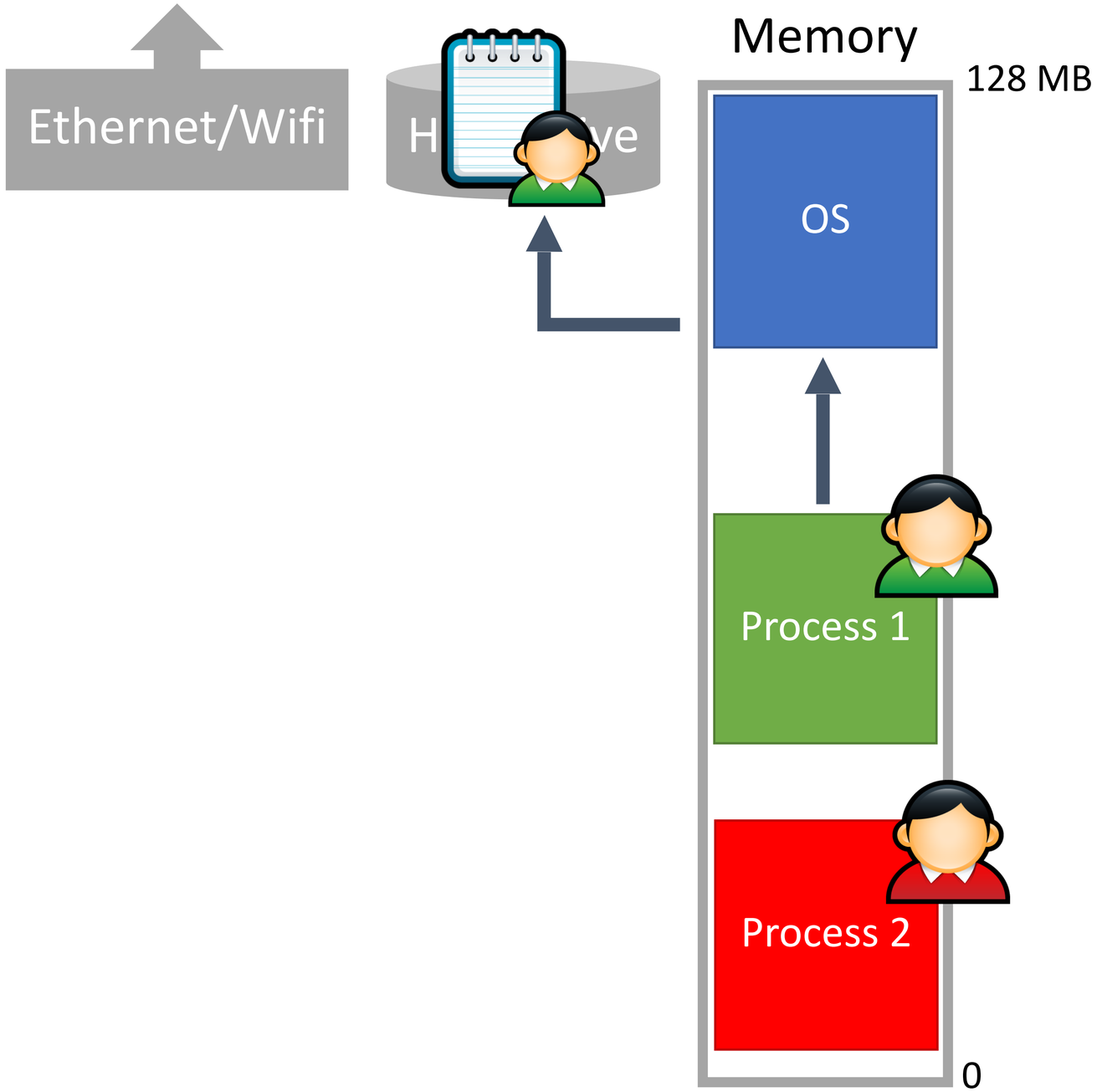
Access control is enforced by the OS, but OS APIs can be bypassed



Device Unsafety

Problem: any process can access any hardware device directly

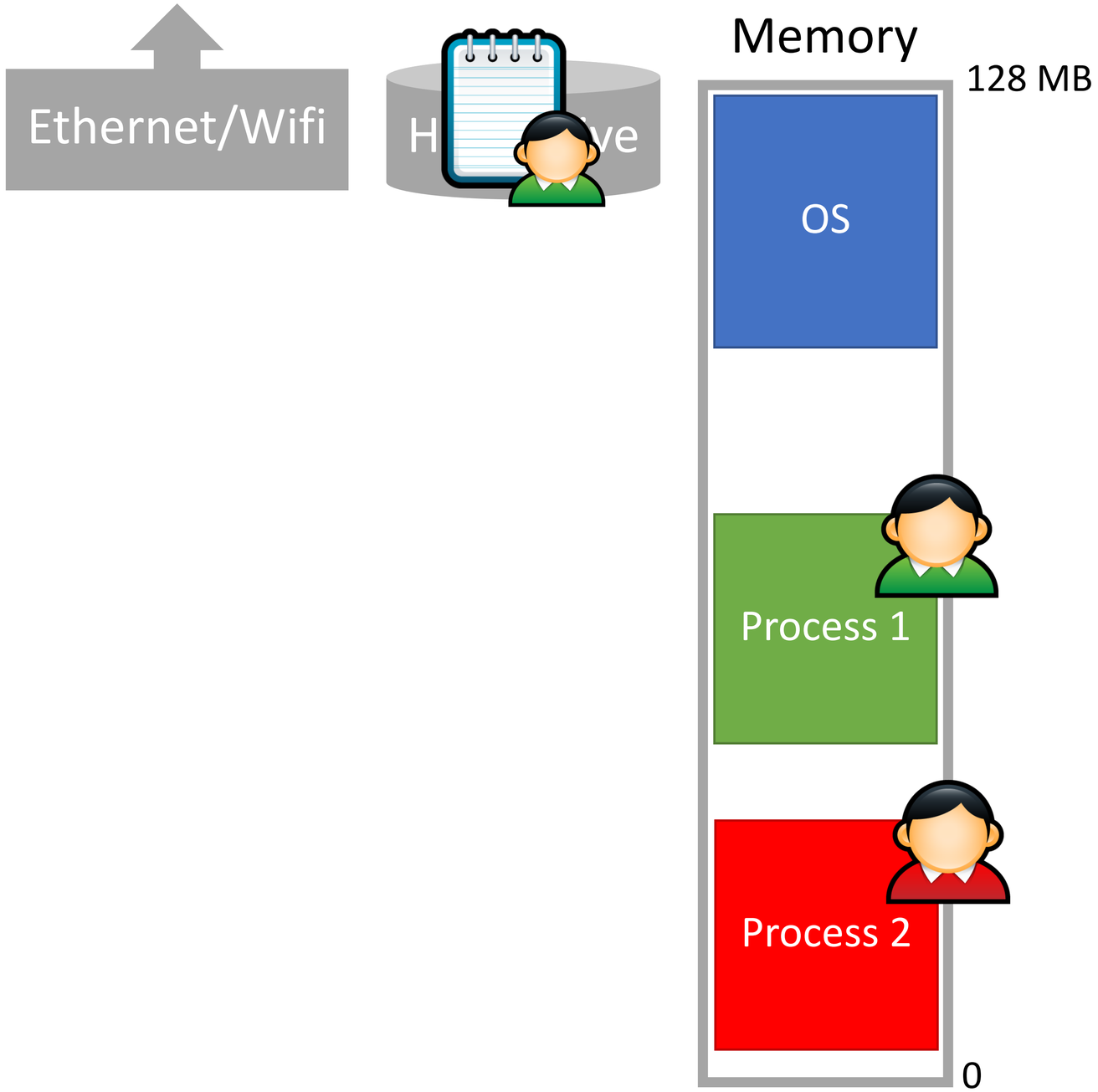
Access control is enforced by the OS, but OS APIs can be bypassed



Device Unsafety

Problem: any process can access any hardware device directly

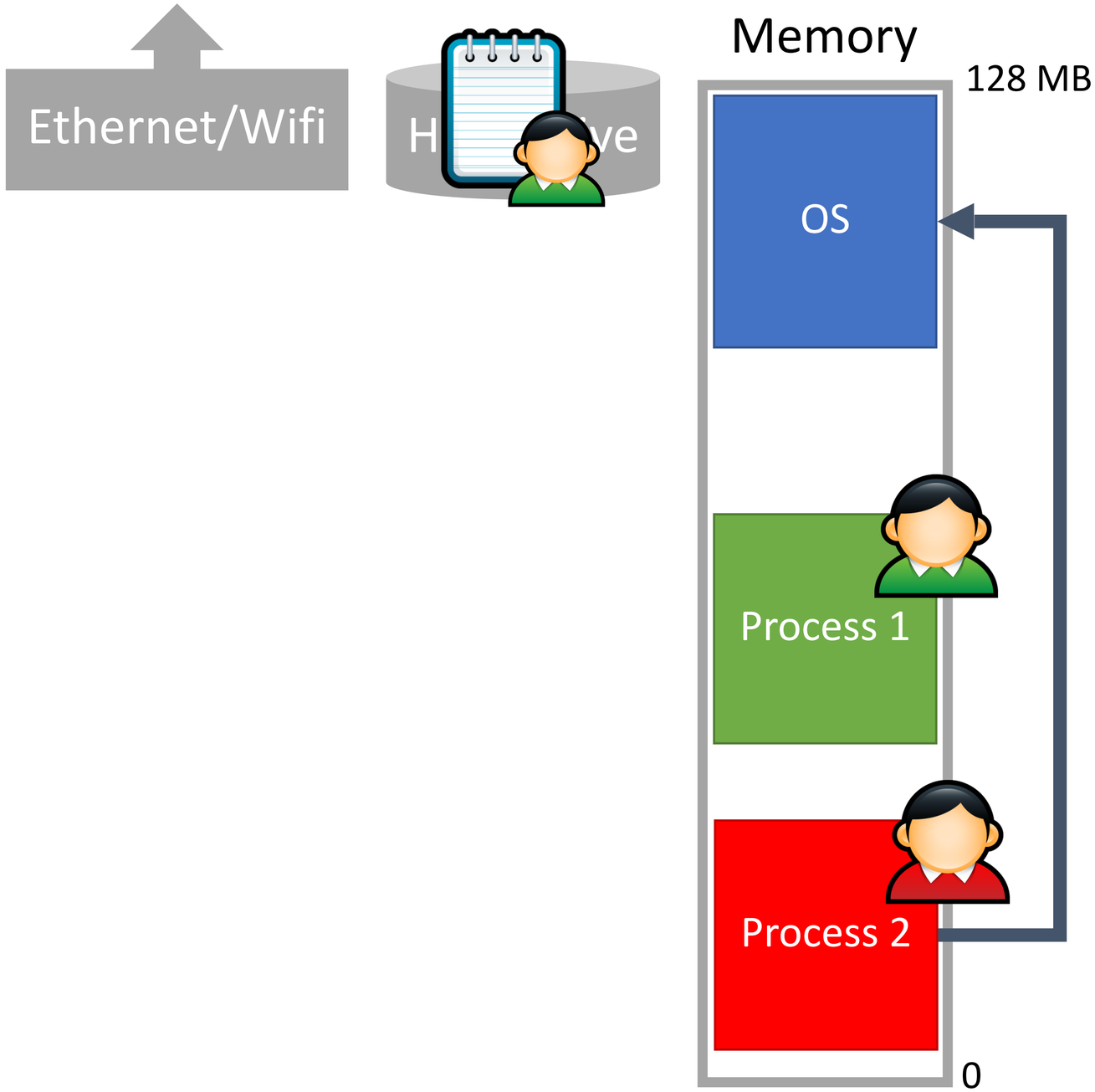
Access control is enforced by the OS, but OS APIs can be bypassed



Device Unsafety

Problem: any process can access any hardware device directly

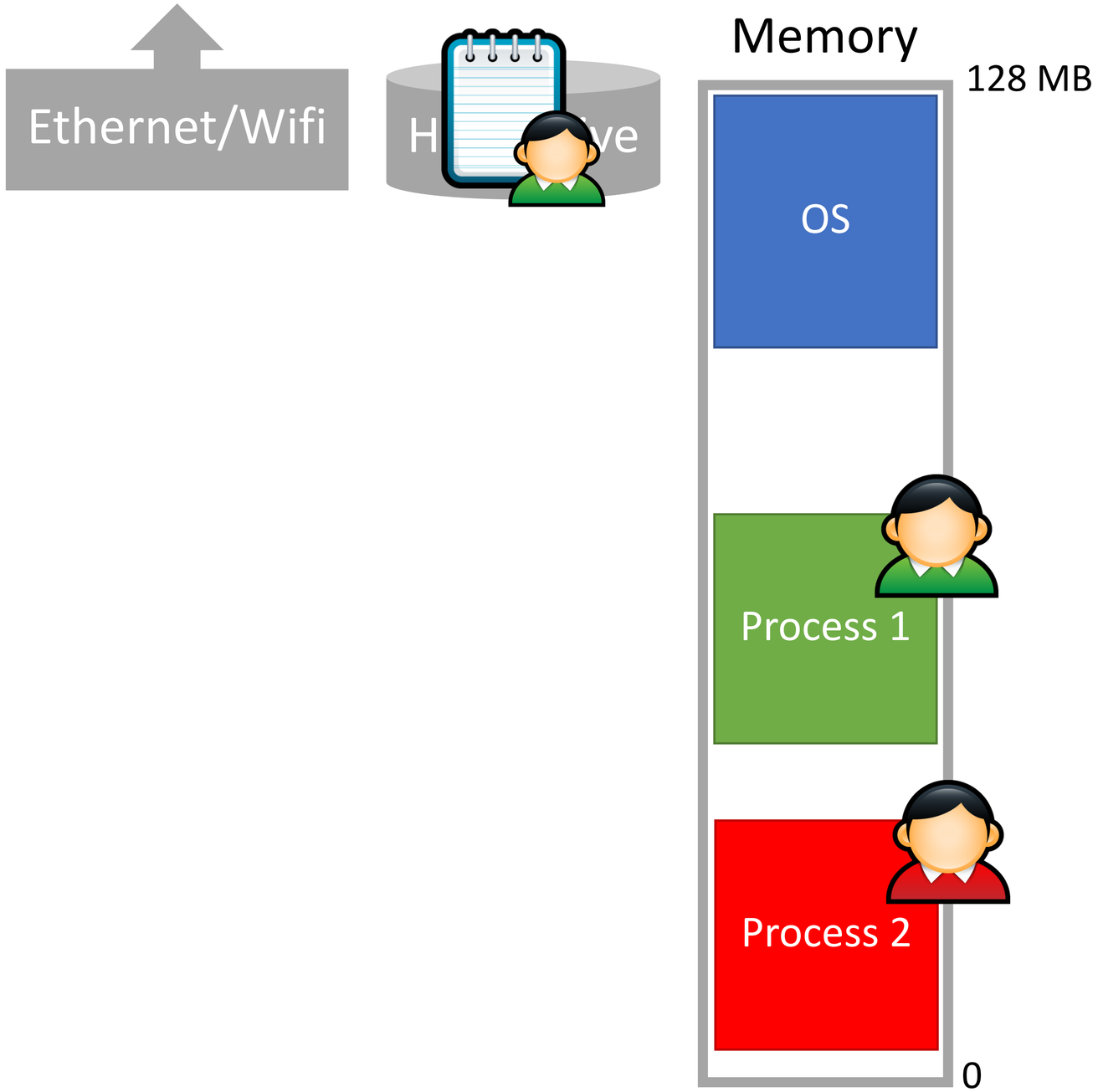
Access control is enforced by the OS, but OS APIs can be bypassed



Device Unsafety

Problem: any process can access any hardware device directly

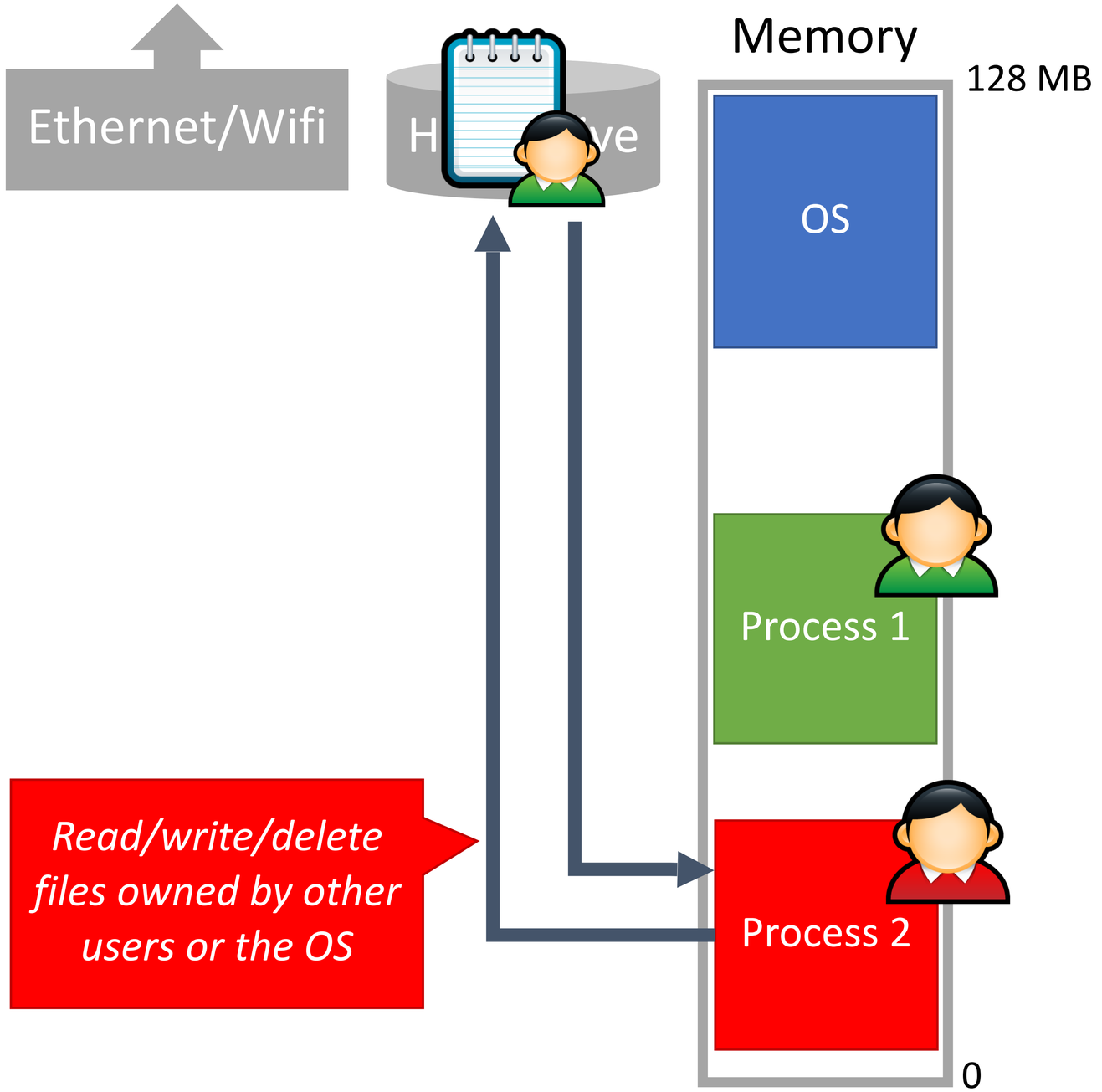
Access control is enforced by the OS, but OS APIs can be bypassed



Device Unsafety

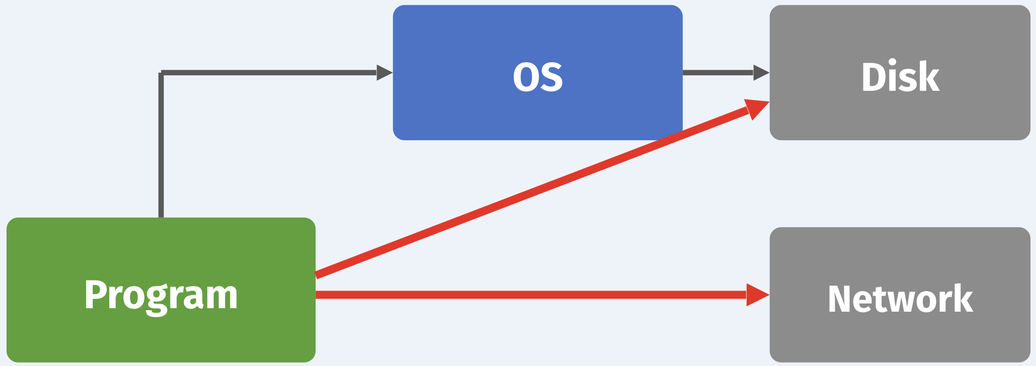
Problem: any process can access any hardware device directly

Access control is enforced by the OS, but OS APIs can be bypassed



Two more ways one program can hurt the rest

Problem 2: direct device access

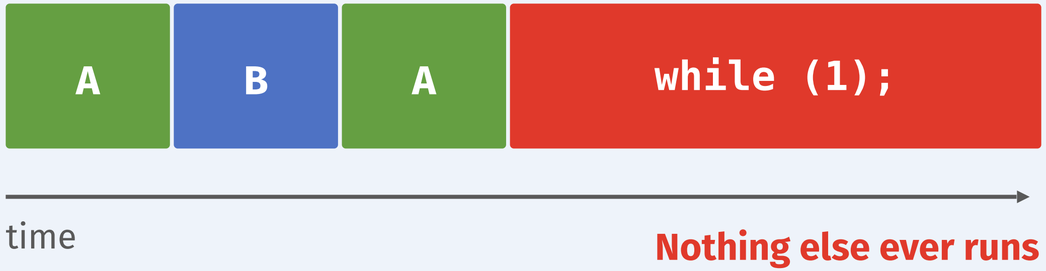


Direct, skipping the OS

- Read raw disk sectors: everyone's files, ignoring permissions
- Send any packet on the network
- Tell a device to copy memory for it (DMA)

Problem 3: hogging the CPU

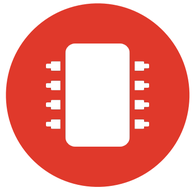
Who is running on the CPU



Nothing else ever runs

With cooperative multitasking, as in Windows 3.1 and classic Mac OS, programs had to hand the CPU back voluntarily. One infinite loop froze the whole machine.

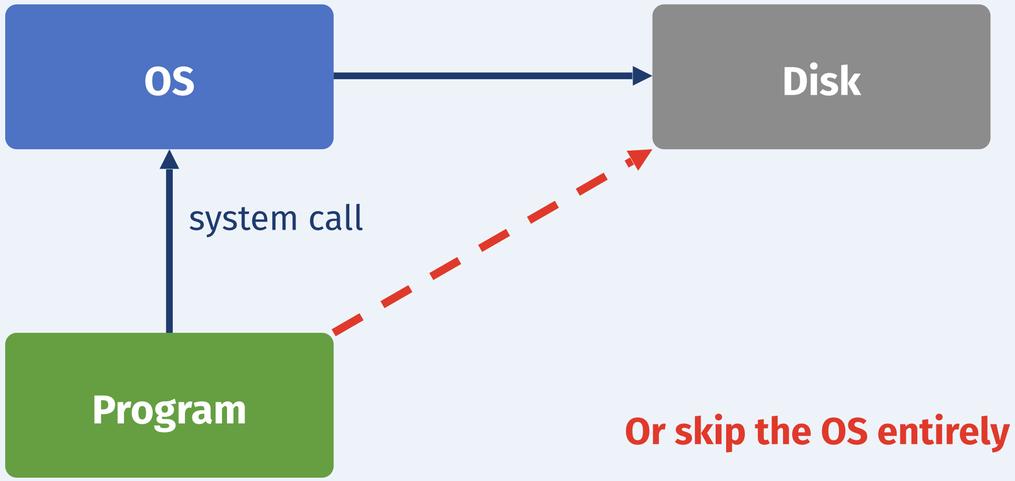
Three problems, three hardware fixes

Problem	Hardware feature	What the OS can now enforce
 **Any memory**	**MMU** address translation	Each program sees only its own memory. The OS's memory is off-limits.
 **Any device**	**User and kernel mode** privileged instructions	Only the kernel can touch devices. Programs must ask through system calls.
 **Keeps the CPU**	**Timer interrupt** fires every few ms	The kernel regains control and can switch to another program.

The kernel switches on all three during boot, before it starts PID 1.

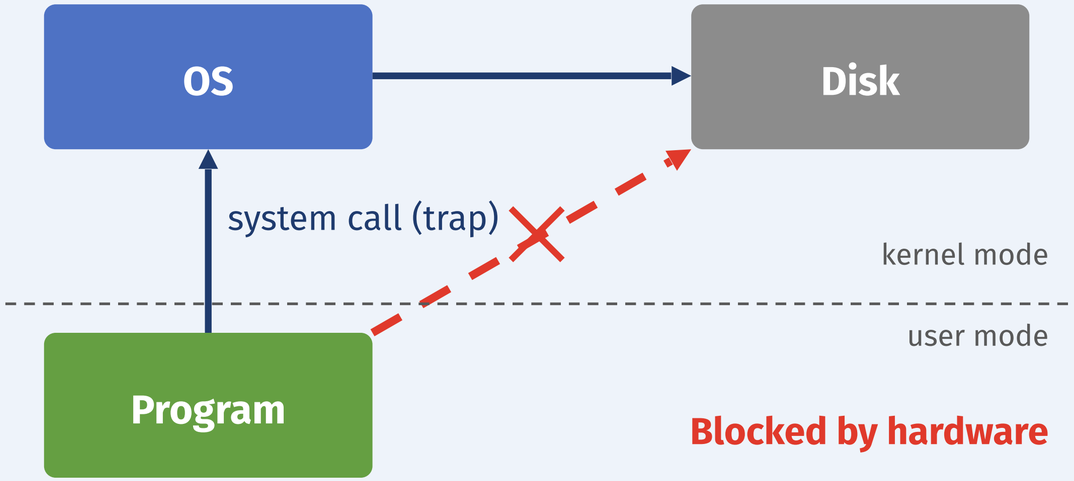
The API becomes the only door

Without protection (DOS)



System calls are a convenience

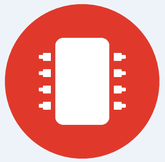
With protection



System calls are the only way in

That's why open(), read() and write() are system calls. Each one crosses from user mode into the kernel, which checks your permissions before it touches the disk.

Isolation still matters today



Meltdown (2018)

A CPU flaw let ordinary programs read kernel memory through speculative execution. The fix, KPTI, stopped mapping most of the kernel into each program's page tables.



Devices that copy memory

A Thunderbolt device can read memory directly using DMA, as the Thunderspy attacks (2020) showed. The IOMMU blocks this: the DMAR lines in our boot log describe it.



Your browser

Chrome runs each website in its own process (Site Isolation, 2018), so the OS keeps a malicious page away from your bank's tab.



Tiny computers

Many microcontrollers have no MMU. Every task shares one memory, just like DOS.

Review

Old systems did not protect memory or devices

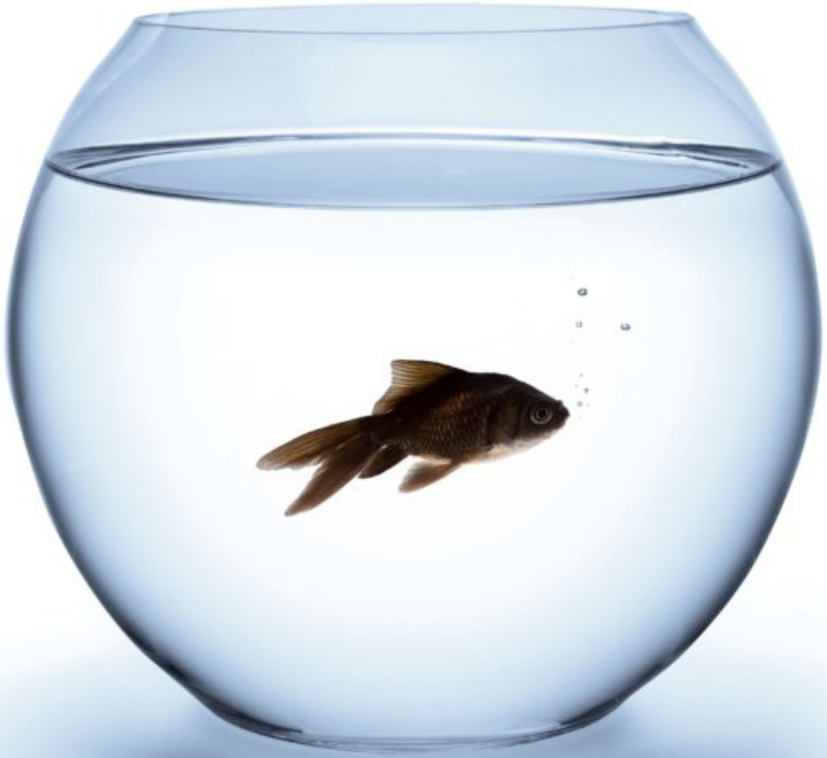
- Any process could access any memory
- Any process could access any device

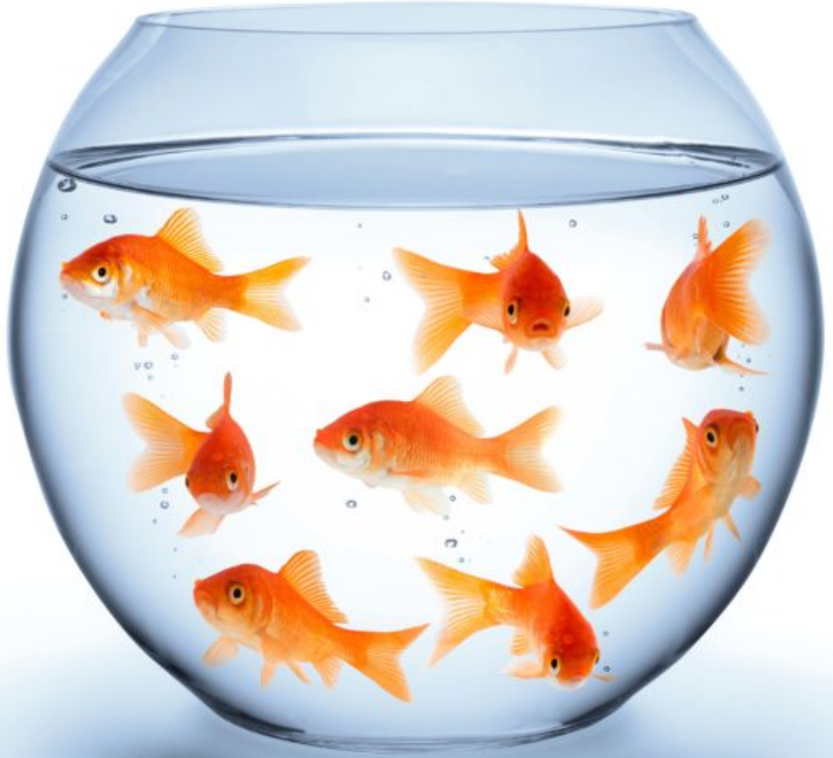
Problems

- No way to enforce access controls on users or devices
- Processes can steal from or destroy each other
- Processes can modify or destroy the OS

On old computers, systems security was literally impossible

ISOLATION





Threat Model
Principles
Intro to System Architecture
Hardware Support for Isolation
Examples

Towards Modern Architecture

To achieve systems security, we need process isolation

- Processes cannot read/write memory arbitrarily
- Processes cannot access devices directly

How do we achieve this?

Hardware support for isolation

1. Protected mode execution (a.k.a. process rings)
2. Virtual memory



Privilege rings on x86



- x86 has four rings, 0 to 3. A lower number means more privilege.
- Ring 0: the kernel, including every device driver, on Linux, Windows and macOS
- Ring 3: every program, even ones running as root
- Rings 1 and 2 are essentially unused: 64-bit page tables only know “user” (ring 3) and “supervisor” (rings 0–2)

Same idea on other CPUs

Arm: EL0 apps, EL1 kernel, EL2 hypervisor, EL3 secure firmware

RISC-V: U user, S kernel, M firmware

Why drivers in ring 0 matter

In July 2024, one faulty CrowdStrike kernel driver crashed about 8.5 million Windows PCs at once.

Below ring 0

Nicknames, not real rings: code even more privileged than the kernel, and invisible to it

Ring 3	Programs
Ring 0	Kernel
“Ring –1”	Hypervisor: runs whole operating systems as guests (VT-x, AMD-V; KVM, Hyper-V)
“Ring –2”	System Management Mode: firmware code the CPU switches to on special interrupts. The OS can’t see it.

not part of the CPU at all

“Ring –3”	A separate processor: Intel CSME or AMD PSP. Runs whenever the board has power.

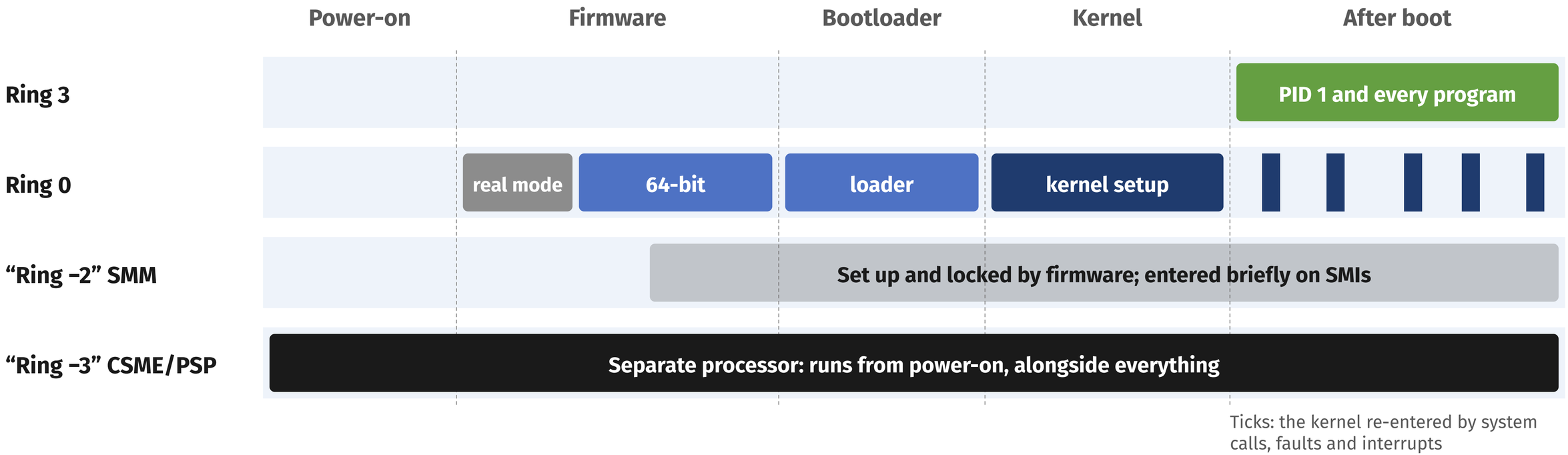
Case study: Intel’s management engine

From 2015 (ME 11) it ran its own operating system, MINIX 3, with a network stack and a web server for remote management.

In 2017, Google engineers warned that this layer sat below any OS they controlled. The same year, a flaw in that web server’s login (CVE-2017-5689) let attackers take over managed PCs over the network.

Isolation relies on correctness of the most privileged code.

Privilege levels during boot



No mode switch for the OS

UEFI firmware is already in 64-bit mode before any OS code runs.

Going down is deliberate

The kernel enters ring 3 for the first time when it starts PID 1.

Coming back up is controlled

Only through doors the kernel set up: system calls, exceptions and interrupts.

What ring 3 can’t do

Instruction	Controls	If a program could run it…
`mov to cr0`	Whether protection and paging are on	Switch protection off
`mov to cr3`	Which page tables are active	Map any memory, including the kernel’s
`lidt, wrmsr`	Interrupt handlers and the system call entry point	Point the kernel’s doors at its own code
`cli, sti`	Whether interrupts are delivered	Block the timer and keep the CPU forever
`hlt`	Stops the CPU until the next interrupt	Freeze a core
`in, out`	I/O ports, used by older devices	Talk to devices directly

Most devices don’t use in and out anymore

Their registers sit in the PCIe window on the memory map. The kernel protects them with page tables: they simply aren’t mapped into a program’s memory.

If a program tries any of these, the CPU raises a fault and the kernel takes over.

Try it: run a privileged instruction

```
$ cat hlt.c
int main(void) {
    __asm__("hlt");   // ring 0 only
    return 0;
}
$ gcc hlt.c -o hlt && ./hlt
Segmentation fault (core dumped)
$ sudo dmesg | tail -1
traps: hlt[4021] general protection
  fault ip:55d4c1a0d131 sp:7ffc3a1e9f60
  error:0 in hlt[…]
```

1. **The CPU sees hlt in ring 3**
   and refuses to run it
2. **It raises a protection fault**
   and jumps into the kernel's fault handler, in ring 0
3. **The kernel decides what to do**
   Linux sends the program the signal SIGSEGV
4. **The program dies**
   and the shell prints "Segmentation fault"

Green: ring 3. Blue: ring 0.

A program never just crashes: the CPU hands control to the kernel, and the kernel decides.

Three ways into the kernel

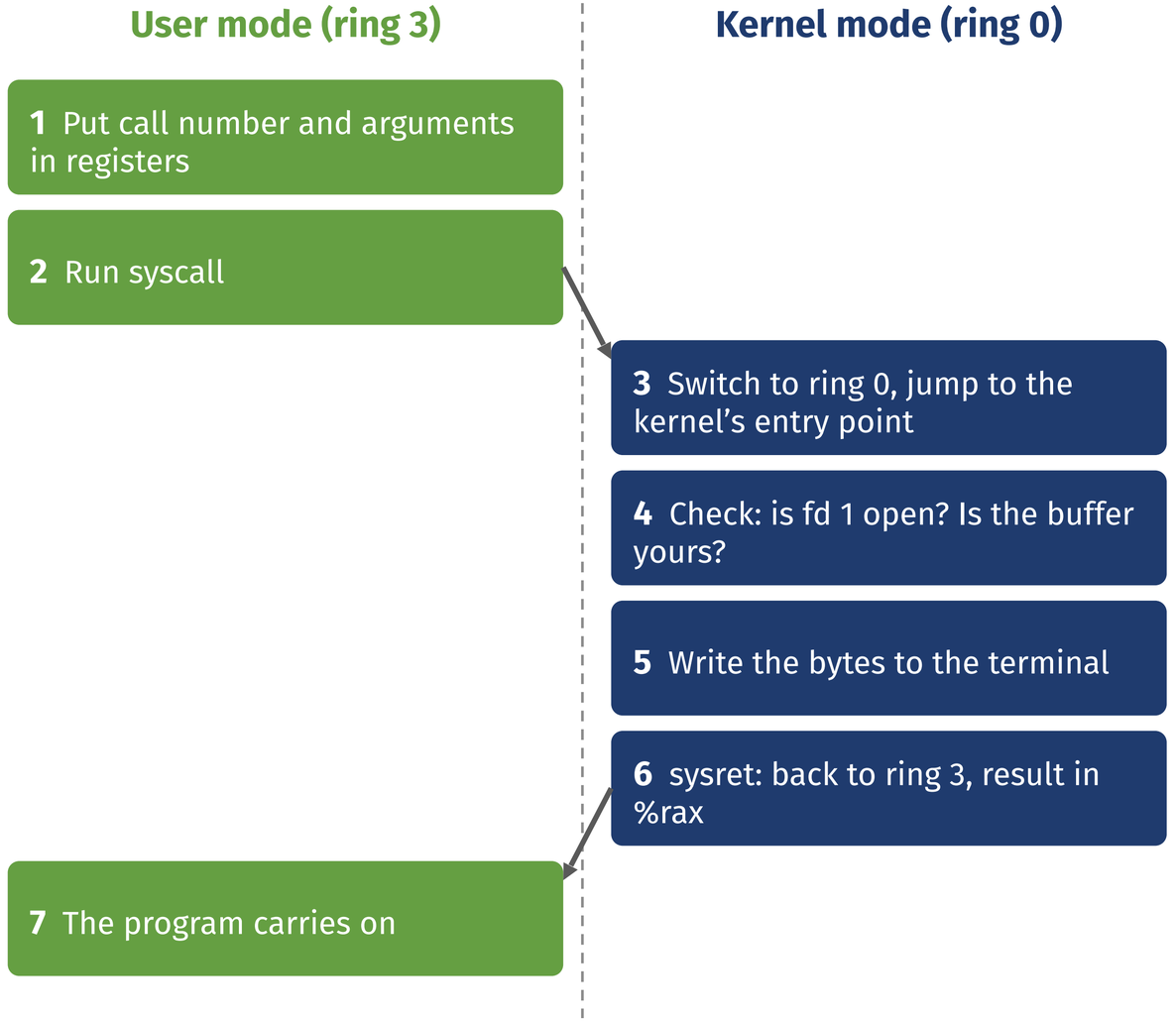
User mode (ring 3)

System call	Exception	Interrupt
The program asks for something: open a file, send a packet	The program did something the CPU won't allow or can't finish: hlt, divide by zero, unmapped memory	The outside world needs attention: the timer, a key press, a network packet
deliberate ↓	*accidental* ↓	*from hardware* ↓
Kernel mode (ring 0)		
System call entry point, set at boot	Fault handler, listed in the interrupt table	Device handler, listed in the same table

In all three, the CPU jumps to an address the kernel registered at boot, so a program can't choose where it lands in the kernel. When the kernel is done, it returns to ring 3.

A system call, step by step

```
# write(1, "hi\n", 3) on x86-64 Linux
mov  $1, %eax          # call number: write
mov  $1, %edi          # fd 1: standard output
lea  msg(%rip), %rsi   # address of "hi\n"
mov  $3, %edx          # length in bytes
syscall                # enter the kernel
# %rax now holds the result: 3
```



Older and other ways in

int 0x80 and sysenter on 32-bit x86, svc on Arm, ecall on RISC-V. Same idea everywhere: one special instruction, one fixed entry point.

See it yourself: strace

```
$ strace -e trace=openat,read,write cat /etc/hostname > /dev/null
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0"..., 832) = 832
…
openat(AT_FDCWD, "/etc/hostname", O_RDONLY) = 3
read(3, "nuc\n", 131072)                = 4
write(1, "nuc\n", 4)                    = 4
read(3, "", 131072)                     = 0
+++ exited with 0 +++
```

One line, one round trip

Each line is a trip into ring 0 and back, with the result after the =

Most of it is setup

cat first loads its libraries. The real work is the last four calls.

Try: strace -c ls

Counts every system call a program makes

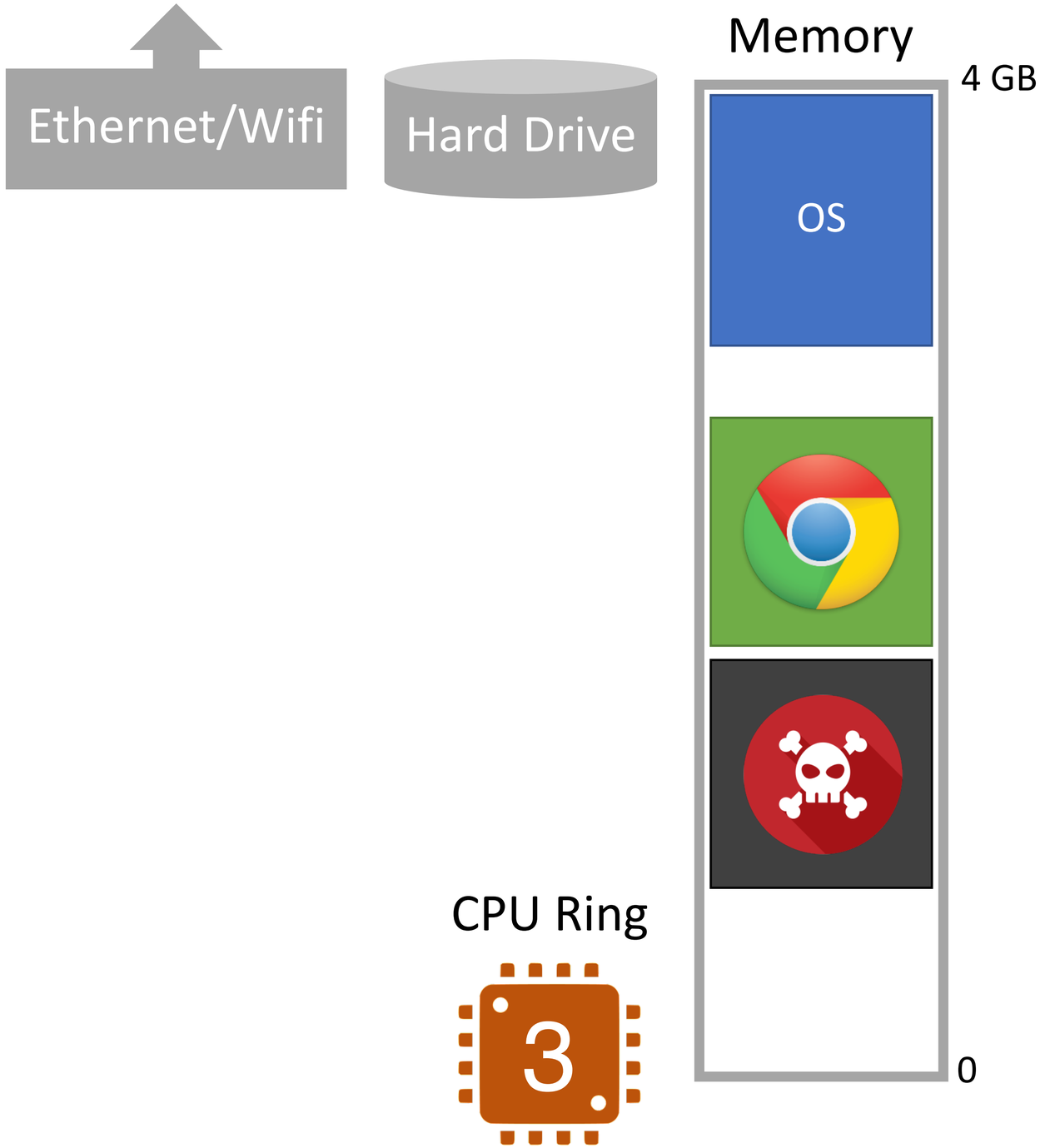
System calls are the only way access resources, so strace sees nearly everything a program does to the outside world.

Virtual Memory

Status Check

At this point we have protected the devices attached to the system…

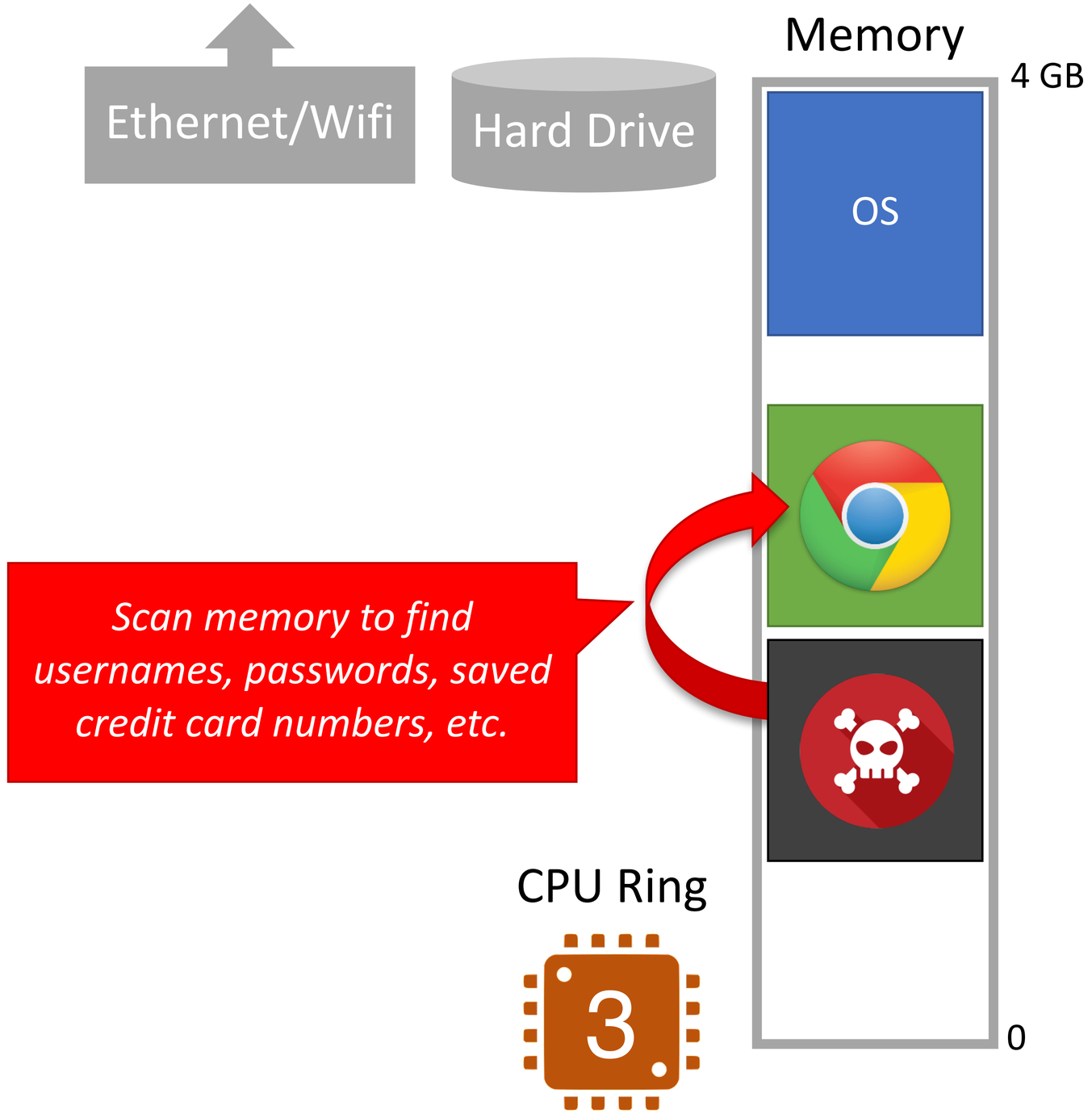
… But we have not protected memory



Status Check

At this point we have protected the devices attached to the system…

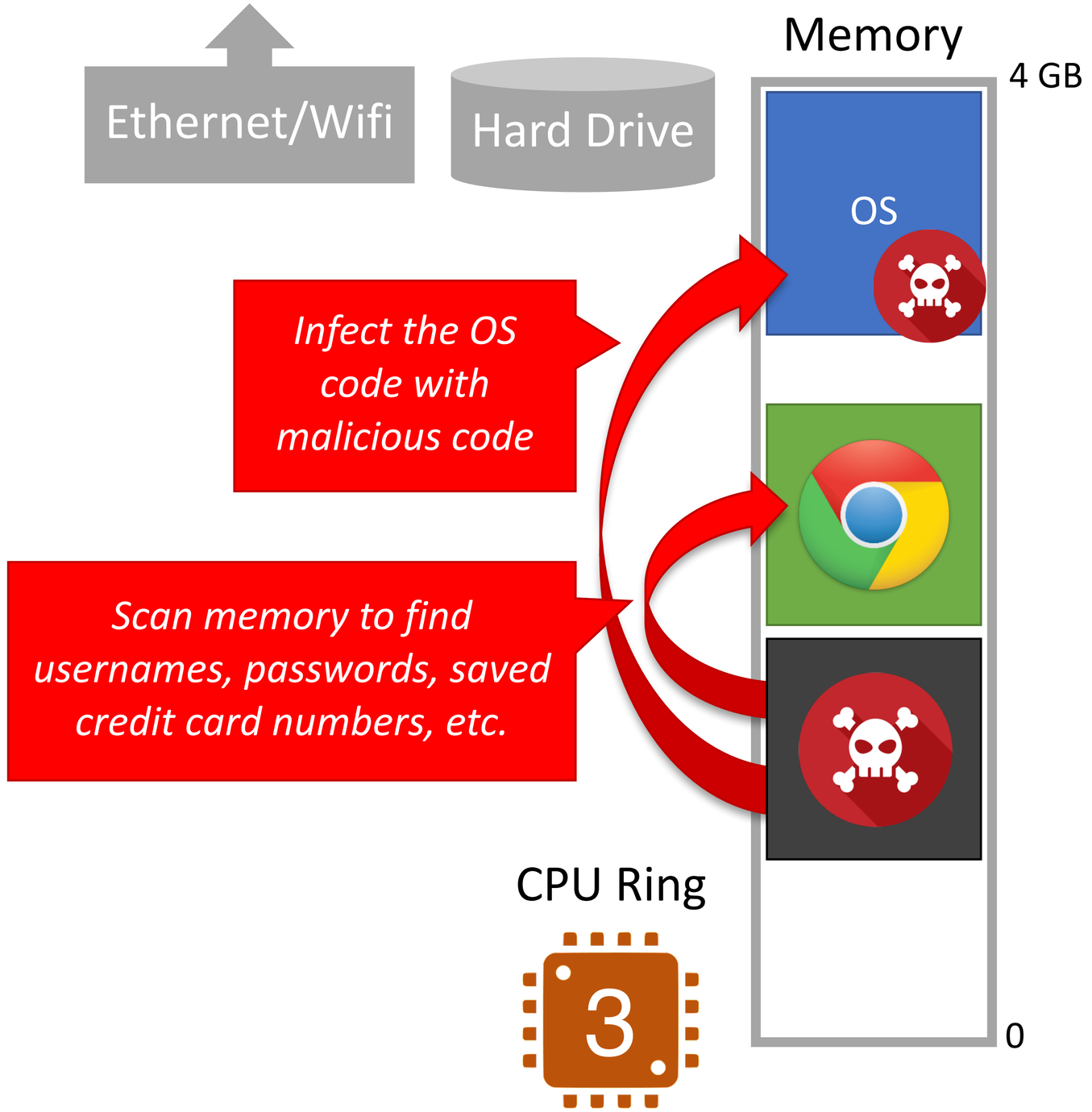
… But we have not protected memory



Status Check

At this point we have protected the devices attached to the system…

… But we have not protected memory



Memory Isolation and Virtual Memory

Modern CPUs support virtual memory

Creates the illusion that each process runs in its own, empty memory space

- Processes can not read/write memory used by other processes
- Processes can not read/write memory used by the OS

In later courses, you will learn how virtual memory is implemented

- Base and bound registers
- Segmentation
- Page tables

Today, we will do the basic version…

Every address a program uses is virtual



The illusion

Each process sees its own huge, empty address space: 256 TiB on x86-64, far more than the RAM in the machine.

The isolation

A process can only reach physical memory its page table points to, and only the kernel writes page tables.

Virtual memory is the OS's illusionist and referee at once. Every load, store and instruction fetch is translated.

Physical Memory

4 GB

OS

zoom

0

zoom

Physical Memory

4 GB

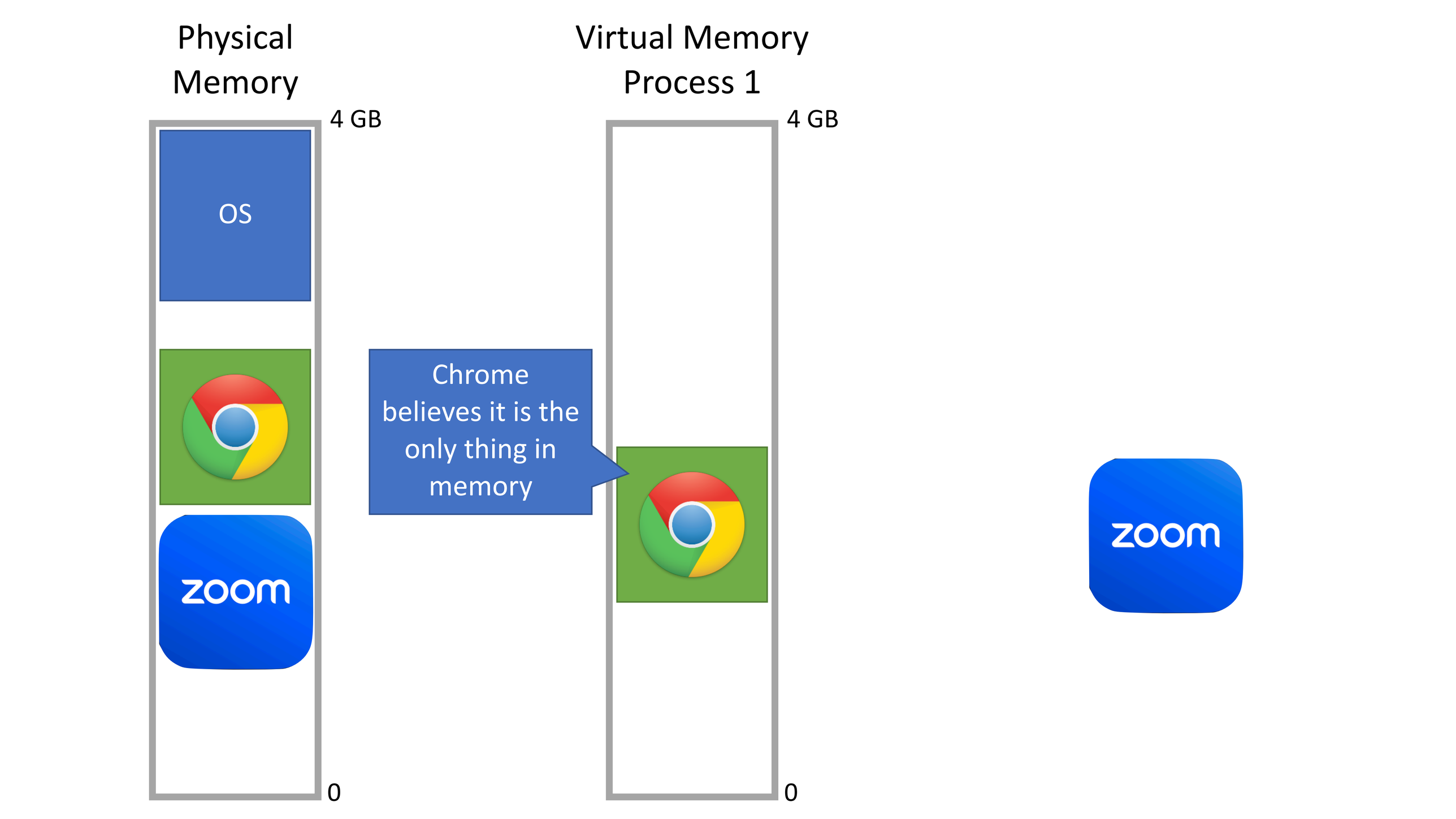
OS

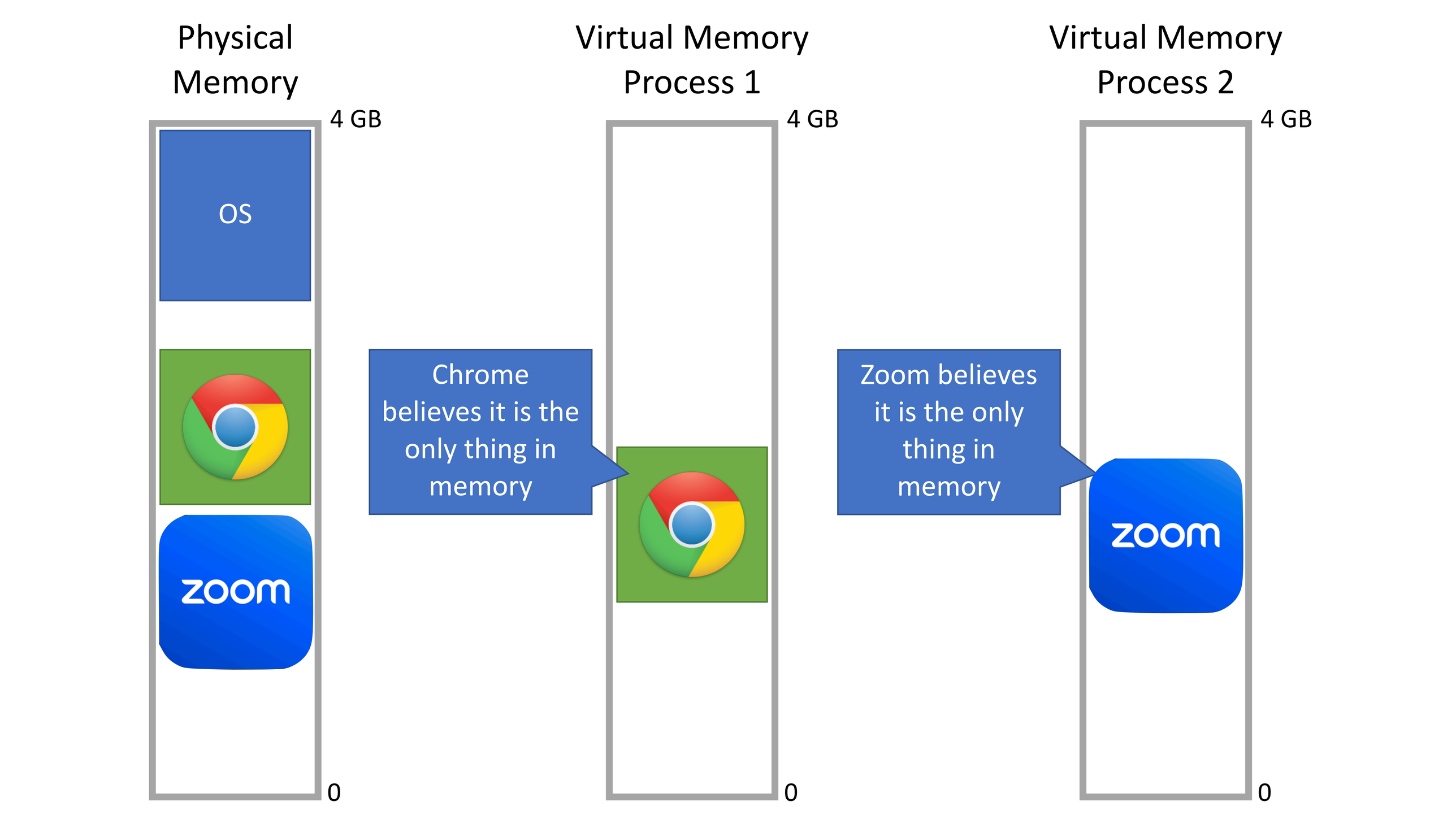




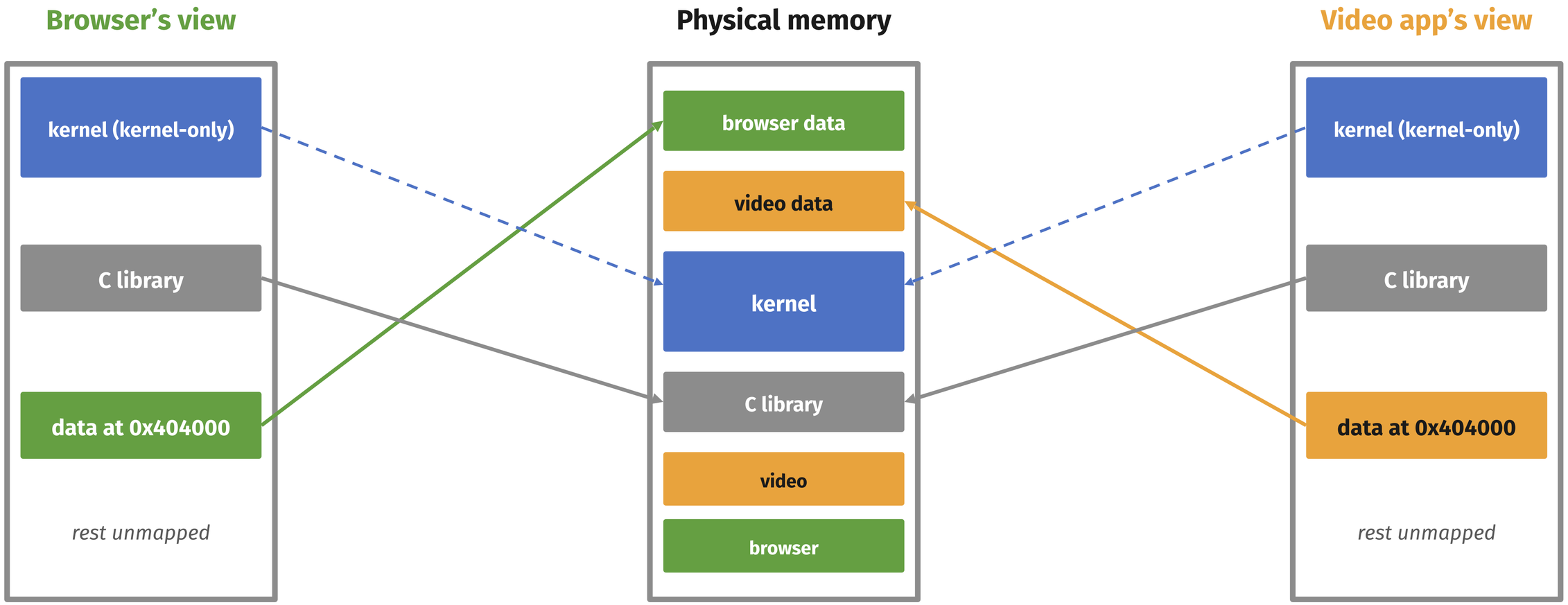
0



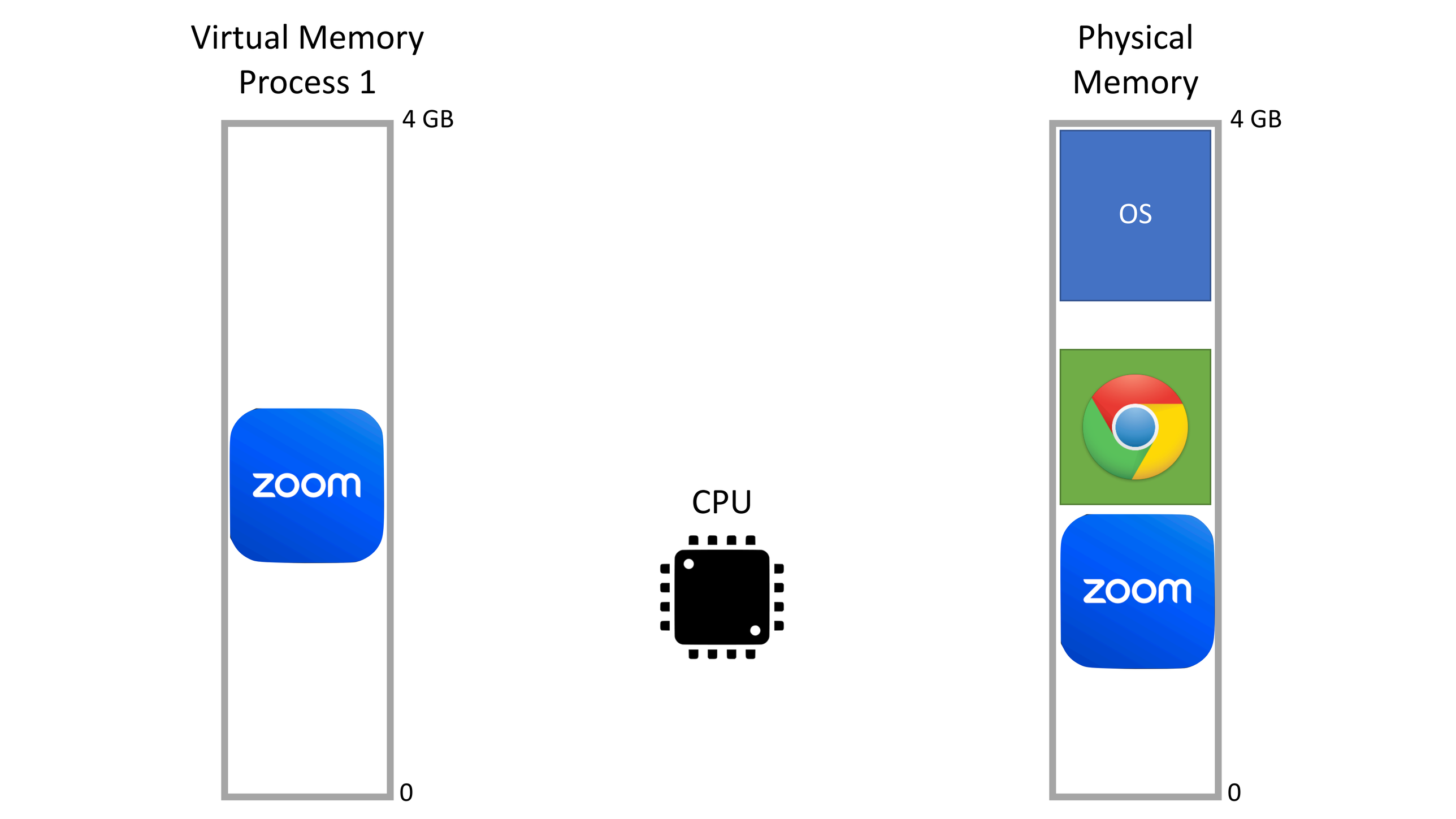

Physical Memory
4 GB
OS
zoom
0
Virtual Memory Process 1
4 GB
Chrome believes it is the only thing in memory
0
zoom

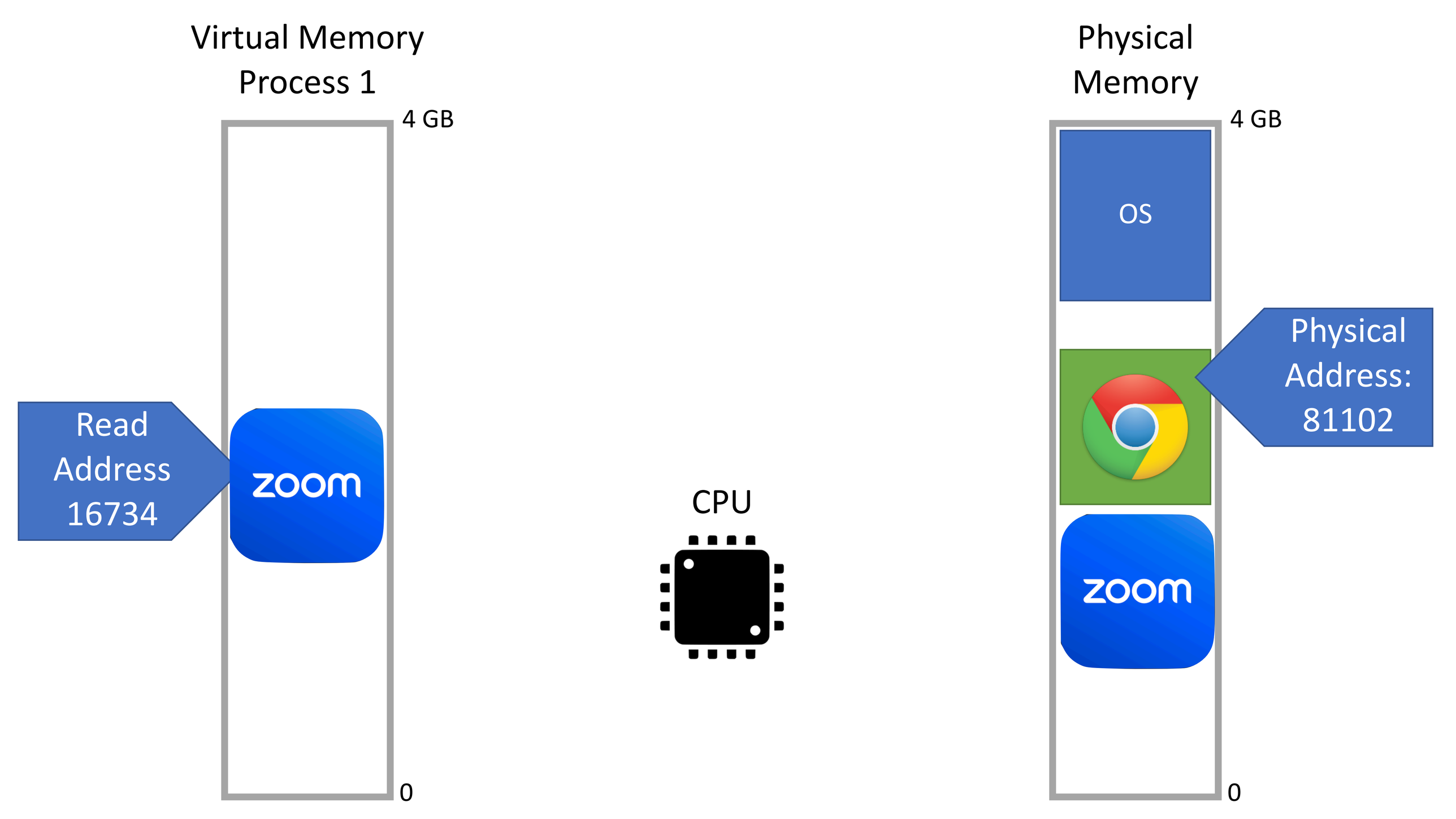

Physical Memory
4 GB
OS
zoom
0
Virtual Memory Process 1
4 GB
Chrome believes it is the only thing in memory
0
Virtual Memory Process 2
4 GB
Zoom believes it is the only thing in memory
zoom
0

Same address, different memory



Same address, different pages. One shared copy of the C library, read-only. The kernel is mapped into both, but kernel-only.


Virtual Memory
Process 1
4 GB
zoom
0
CPU
Physical
Memory
4 GB
OS
zoom
0


Virtual Memory
Process 1
4 GB
Read
Address
16734
zoom
0
CPU
Physical
Memory
4 GB
OS
Physical
Address:
81102
zoom
0

Virtual Memory
Process 1

4 GB

0

Read
Address
16734

Page Table

Virtual Addr.	Physical Addr.
16732	81100
16734	81102
16736	93568
16738	93570

CPU

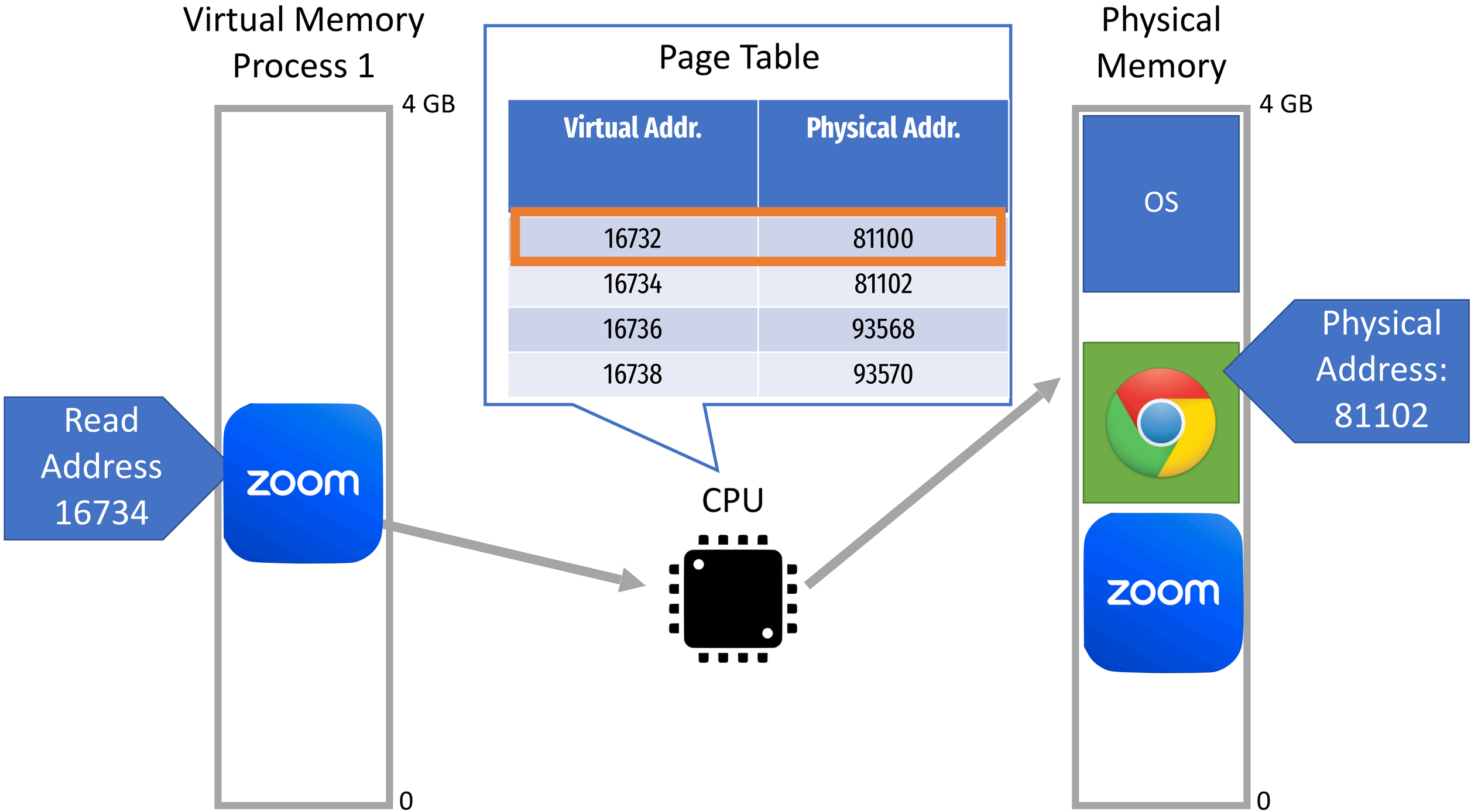
Physical
Memory

4 GB

OS

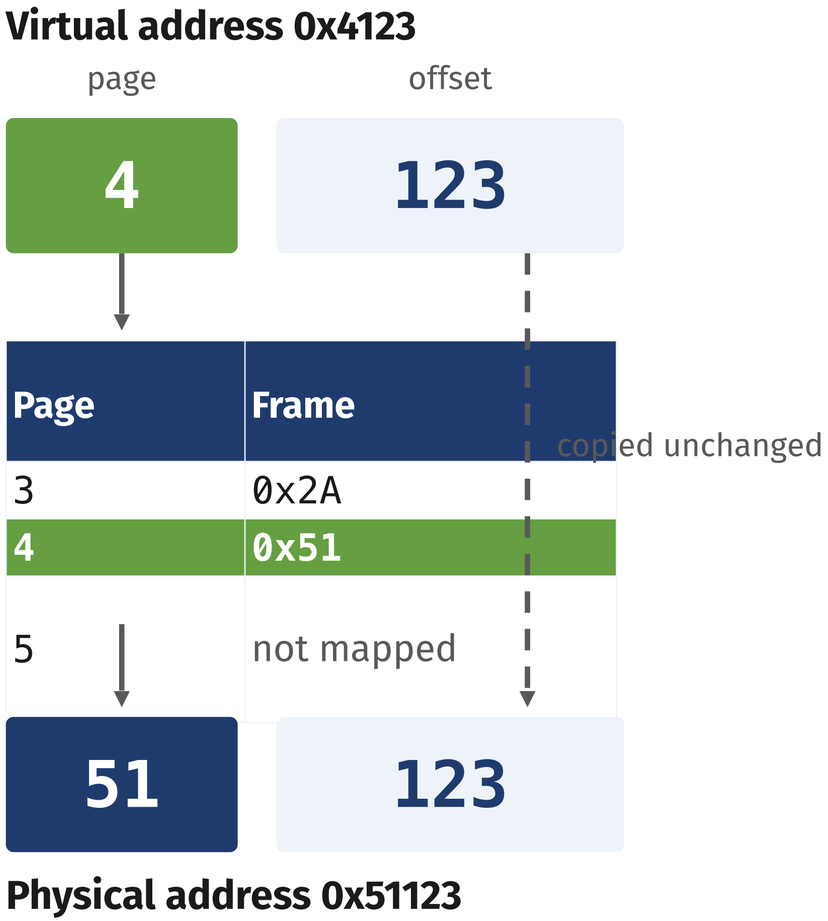
Physical
Address:
81102

0


Virtual Memory
Process 1
4 GB
0
Page Table
Virtual Addr.
Physical Addr.
16732
81100
16734
81102
16736
93568
16738
93570
Physical
Memory
4 GB
0
OS
Read
Address
16734
zoom
CPU
Physical
Address:
81102
zoom

Translation works on pages

With 4 KiB pages, the CPU translates the page number and keeps the offset



Page number + offset

With 4 KiB pages, the last 12 bits (3 hex digits) are the offset. Only the page number is looked up.

Why pages?

One entry per byte would make the table bigger than memory. One entry per 4 KiB page is 4,096 times smaller.

Real tables are trees

A flat x86-64 table covering 256 TiB would take 512 GiB, so the CPU walks a 4-level tree instead.

Each entry also holds permissions

Virtual page	Frame	Present	Write	User	Execute	What it is
`0x401000`	`0x7A2`	yes	**no**	yes	yes	Program code
`0x404000`	`0x051`	yes	yes	yes	**no**	Global variables
`0x405000`	—	not yet	yes	yes	**no**	Heap, not loaded yet
`0xffff8880…`	`0x001`	yes	yes	**no**	**no**	Kernel data

User = no

The kernel is mapped into every process, but ring 3 can’t touch it

Write = no on code

A program can’t rewrite code in place, its own or the kernel’s

Execute = no on data

Bytes an attacker sneaks into data can’t be run as code

Remember Meltdown? It leaked pages marked User = no. The fix stopped mapping most of the kernel into programs at all.

Each entry also holds permissions

Virtual page	Frame	Present	Write	User	Execute	What it is
`0x401000`	`0x7A2`	yes	**no**	yes	yes	Program code
`0x404000`	`0x051`	yes	yes	yes	**no**	Global variables
`0x405000`	—	not yet	yes	yes	**no**	Heap, not loaded yet
`0xffff8880…`	`0x001`	yes	yes	**no**	**no**	Kernel data

User = no

The kernel is mapped into every process, but ring 3 can’t touch it

Write = no on code

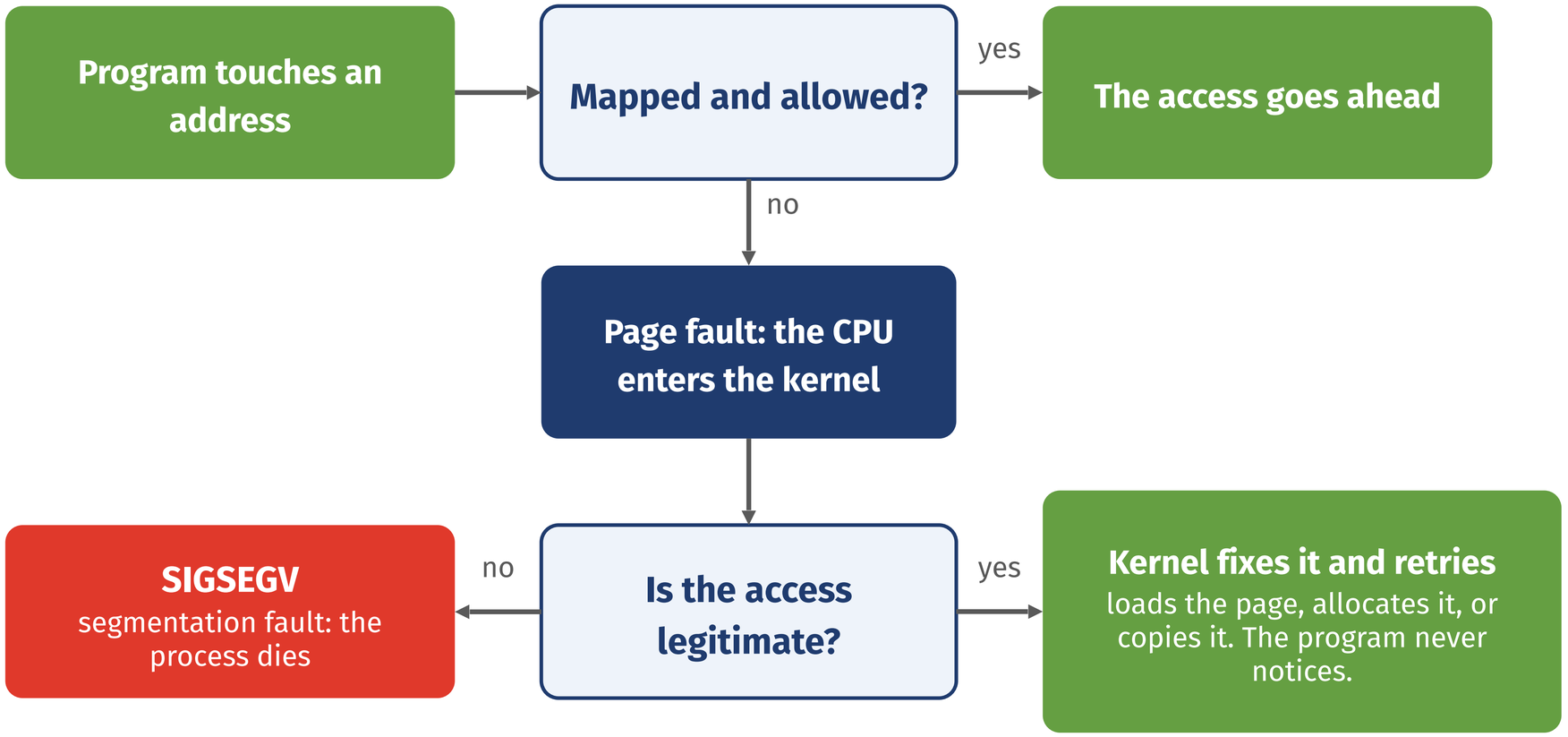
A program can’t rewrite code in place, its own or the kernel’s

Execute = no on data

Bytes an attacker sneaks into data can’t be run as code

Remember Meltdown? It leaked pages marked User = no. The fix stopped mapping most of the kernel into programs at all.

When translation fails: a page fault



Most page faults aren't errors

Pages loaded on first use, copy-on-write after fork(), memory swapped back in: a busy system handles thousands a second.

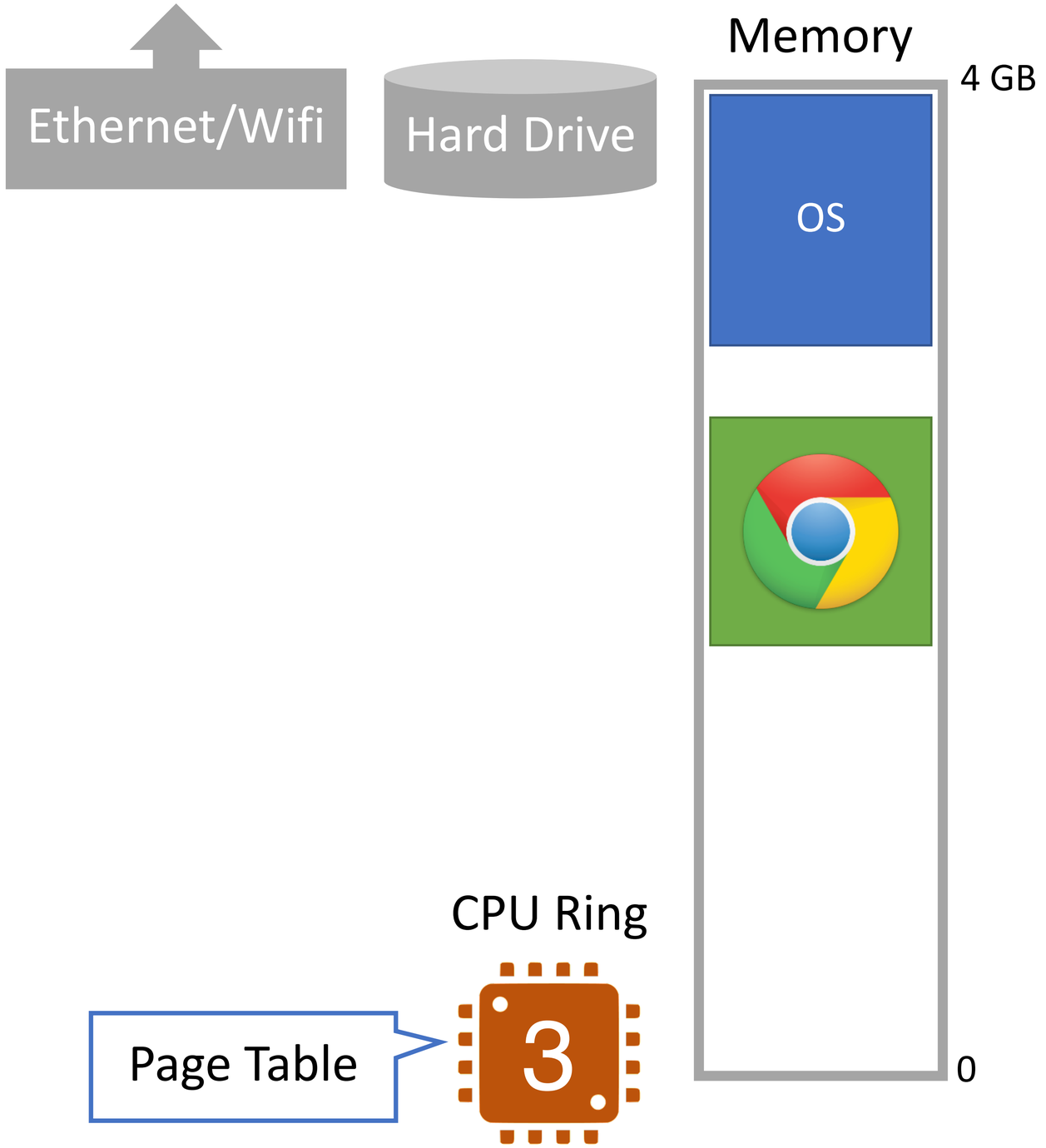
Page fault ≠ segfault

One is a CPU exception; the other is the kernel's verdict.

VM in Action

Processes can only read/ write within their own virtual memory

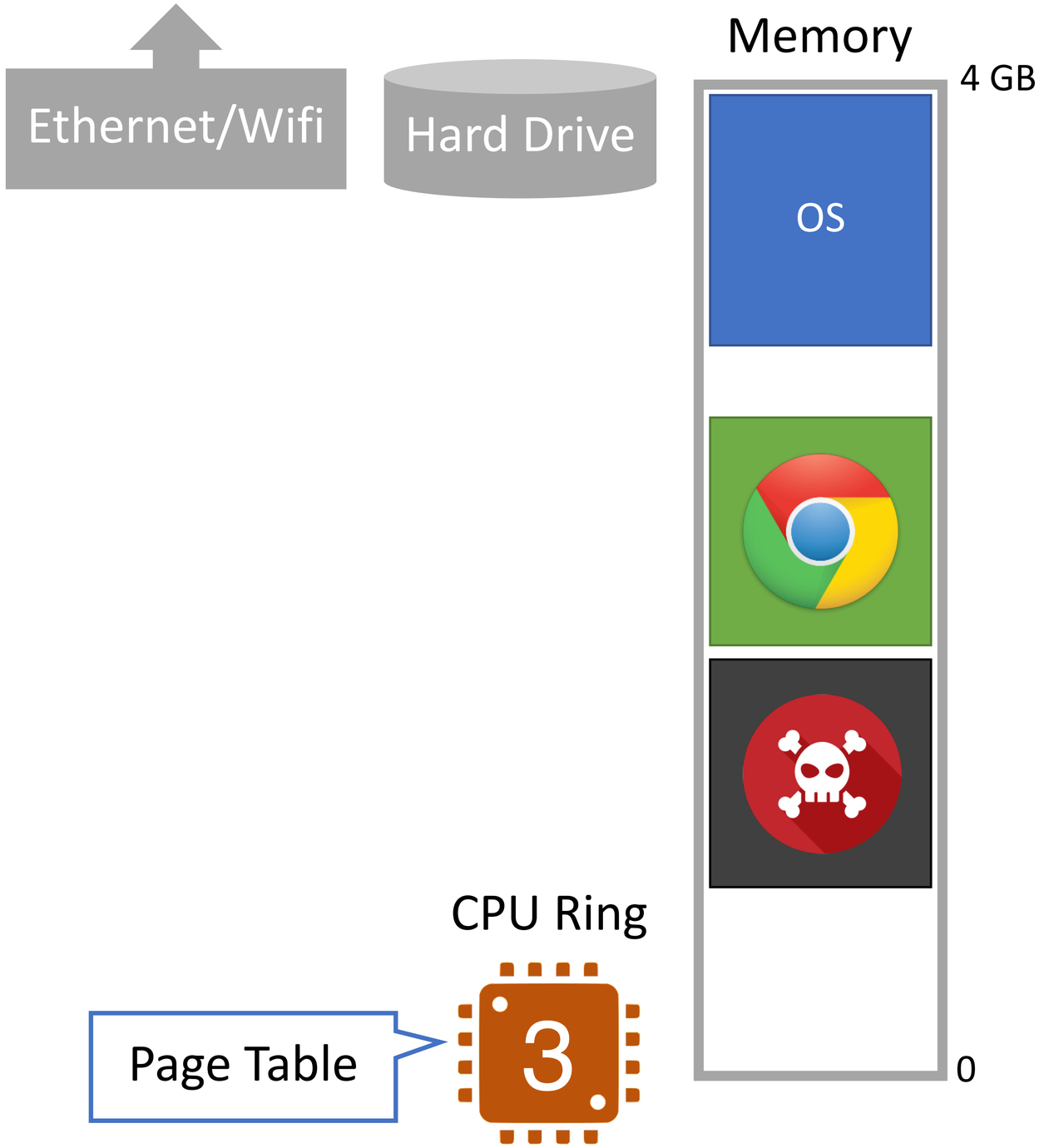
Processes cannot change their own page tables



VM in Action

Processes can only read/ write within their own virtual memory

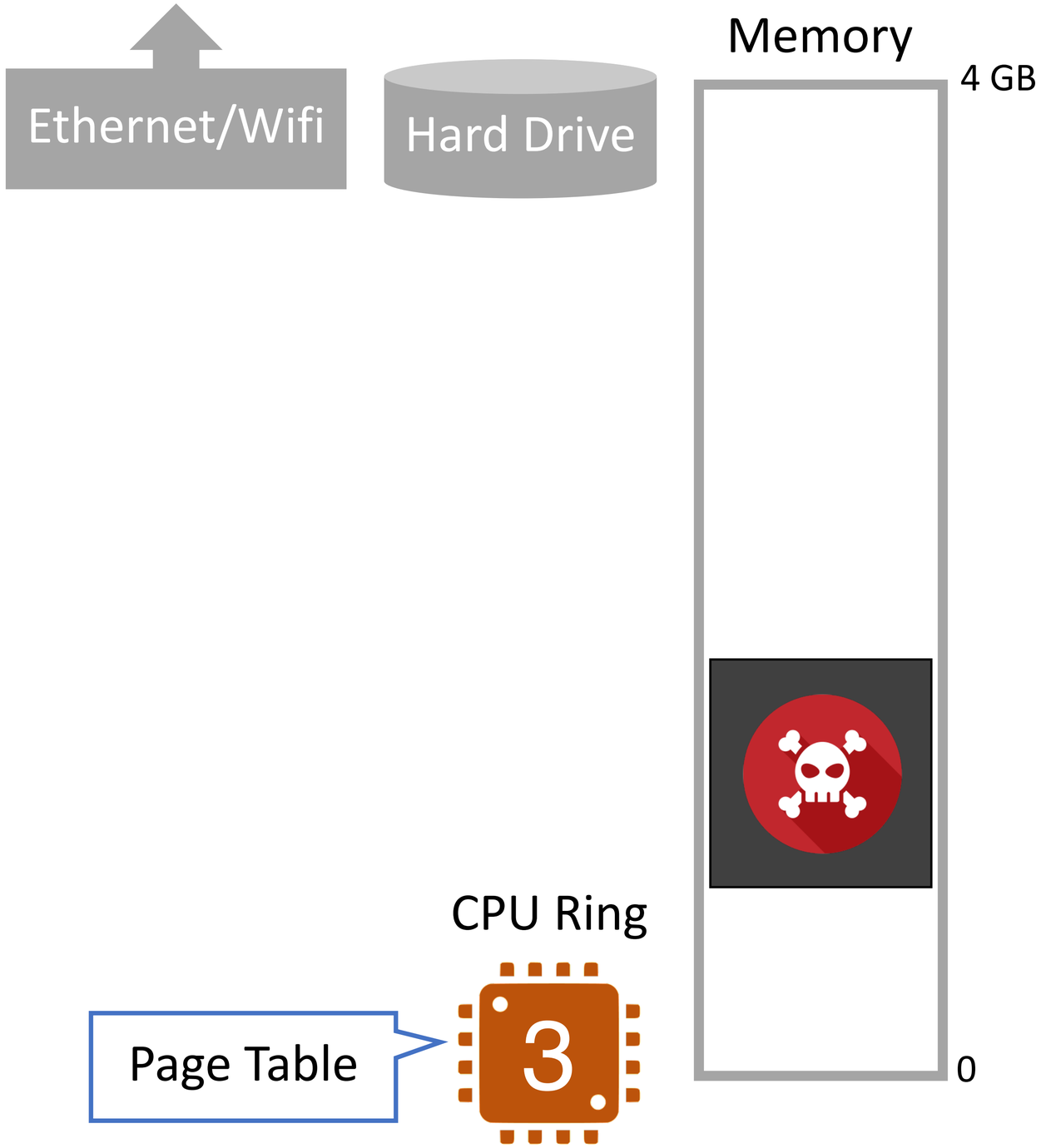
Processes cannot change their own page tables



VM in Action

Processes can only read/ write within their own virtual memory

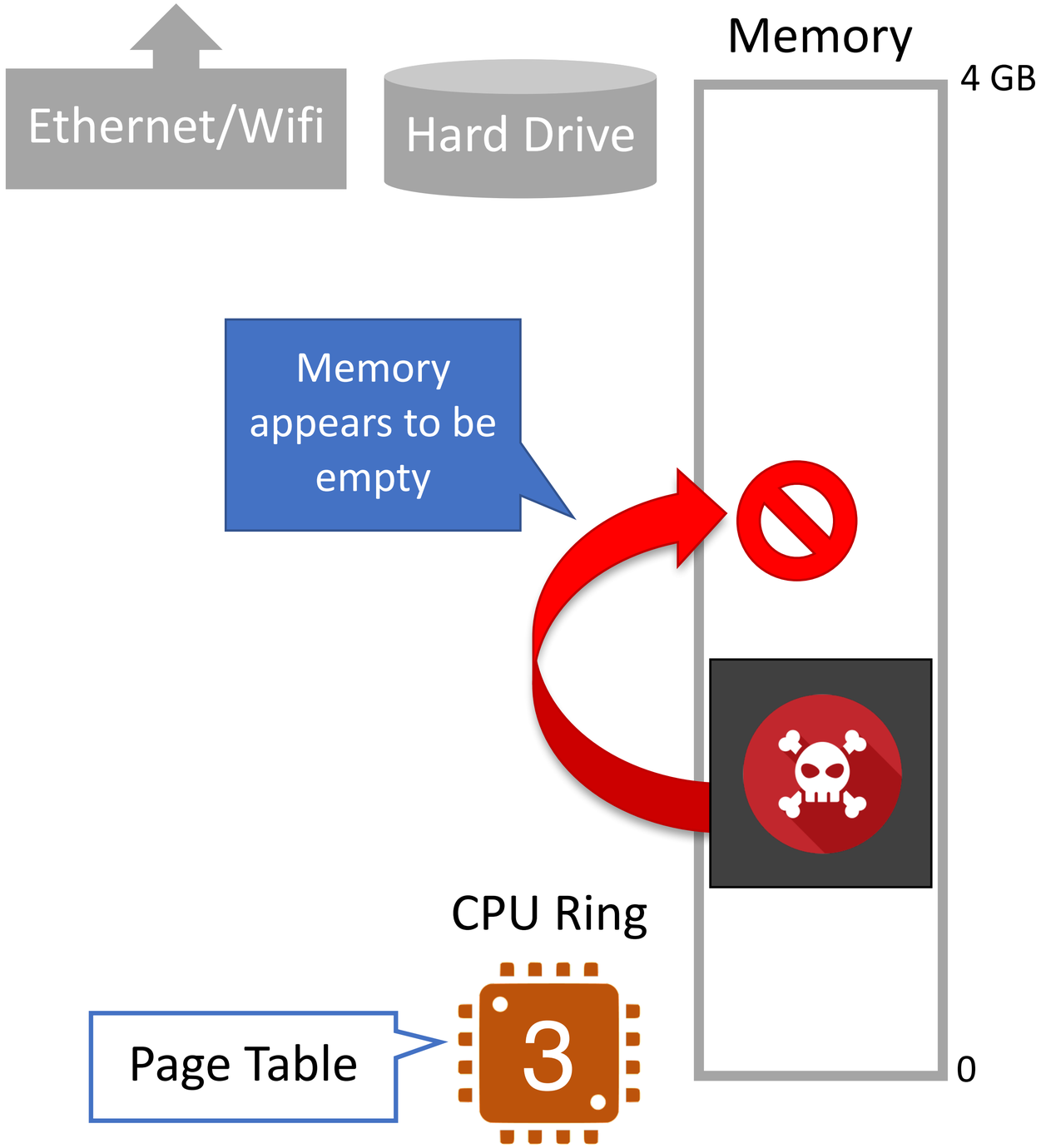
Processes cannot change their own page tables



VM in Action

Processes can only read/ write within their own virtual memory

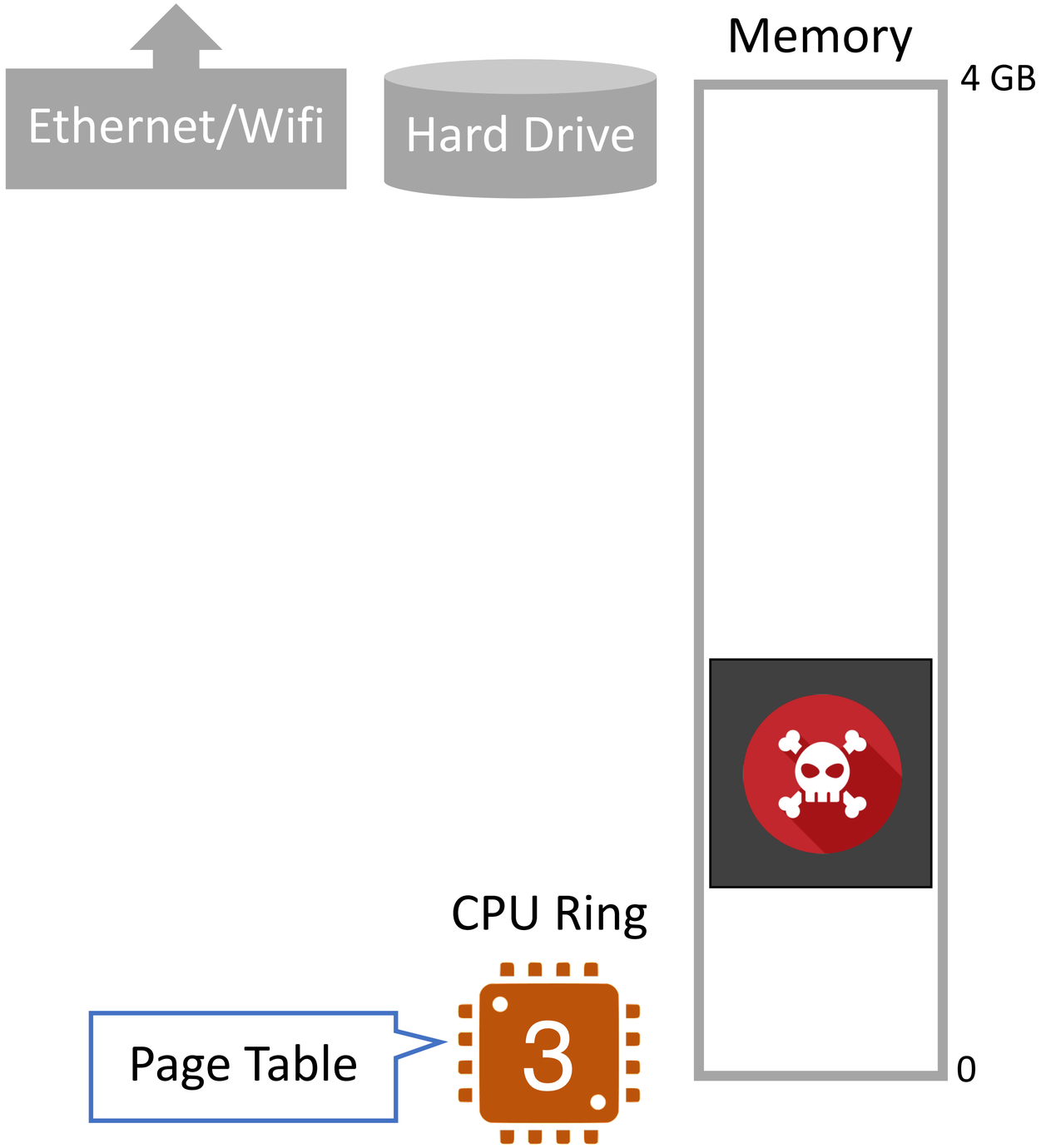
Processes cannot change their own page tables



VM in Action

Processes can only read/ write within their own virtual memory

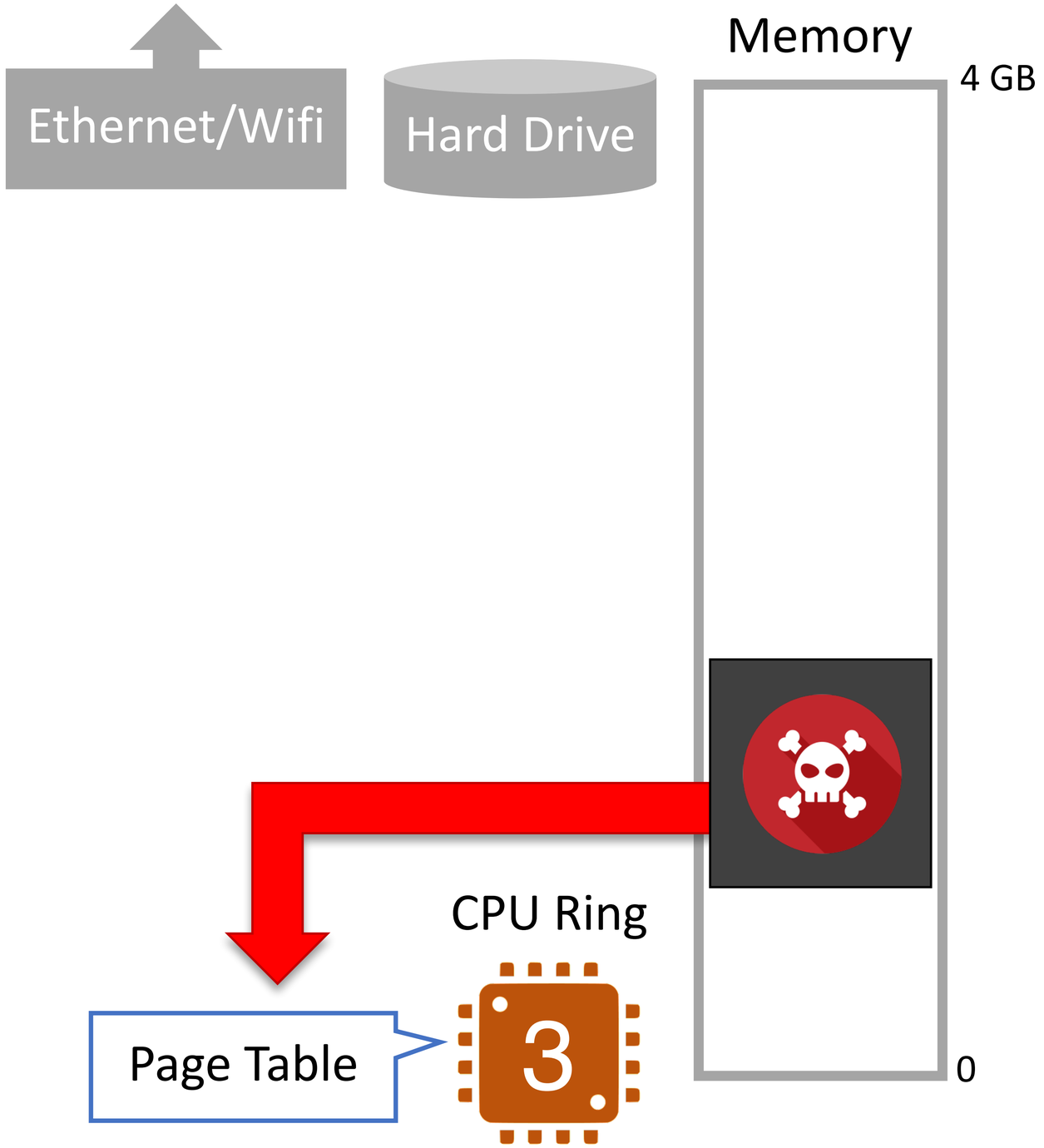
Processes cannot change their own page tables



VM in Action

Processes can only read/ write within their own virtual memory

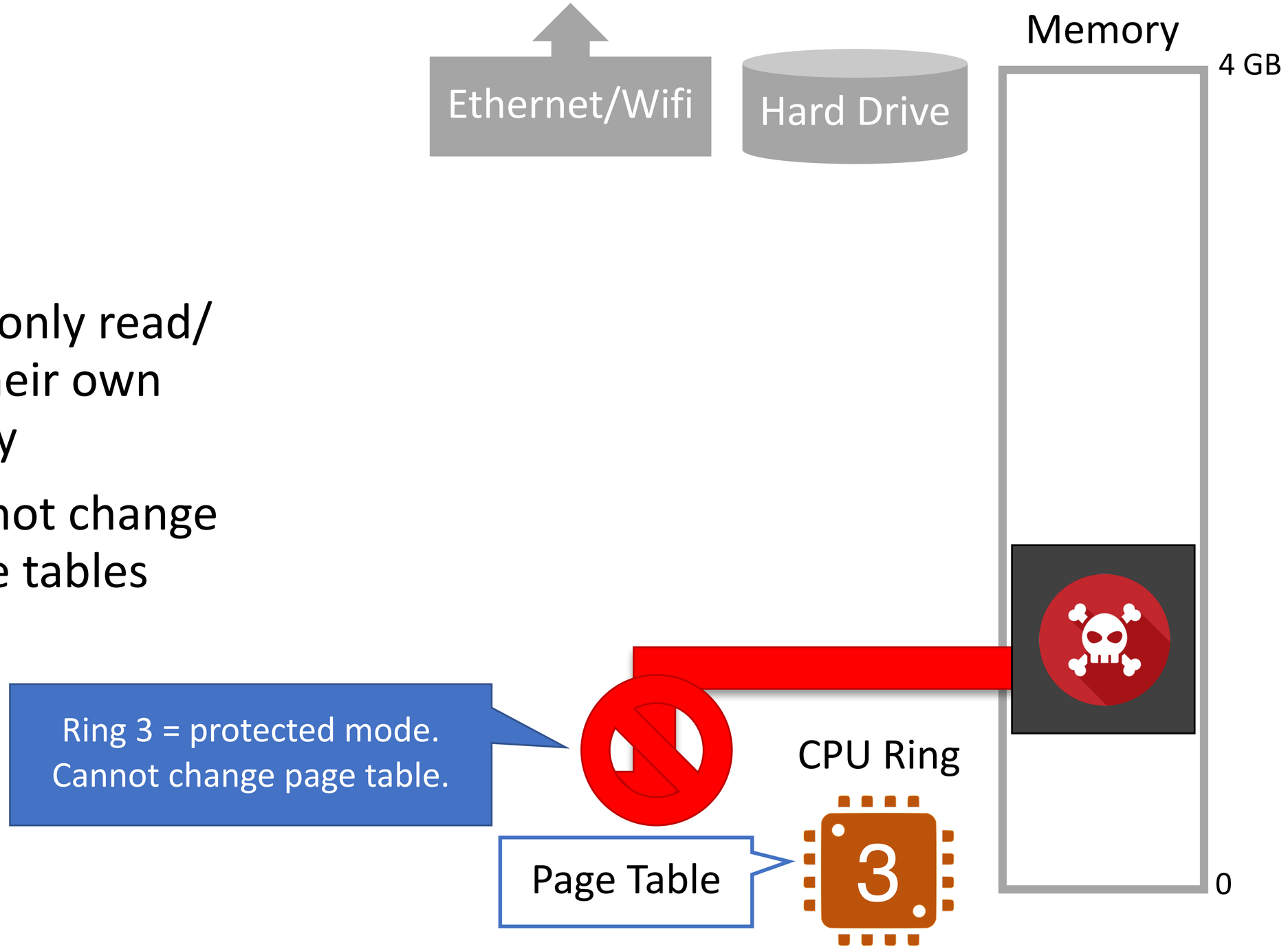
Processes cannot change their own page tables



VM in Action

Processes can only read/ write within their own virtual memory

Processes cannot change their own page tables



In-class exercise: be the MMU

In pairs, 10 minutes. With 4 KiB pages, the first hex digit is the page number.

Process A's page table

Page	Frame	Allowed
0	not mapped	—
1	7	read, execute (code)
2	4	read, write (data)
3	not loaded yet	read, write
F	0	kernel only

Translate each access, or say what happens

Access	Physical address, or what happens?
`Read 0x1234`	
`Write 0x2010`	
`Write 0x1234`	
`Read 0x0004`	
`Write 0x3008`	

Then: in Process B's table, page 2 maps to frame 9. Where does B's read of 0x2010 go? B wants A's data in frame 4. What address should it use?

Be the MMU: answers

Page = first hex digit. Offset = the other three, copied unchanged.

Process A's page table

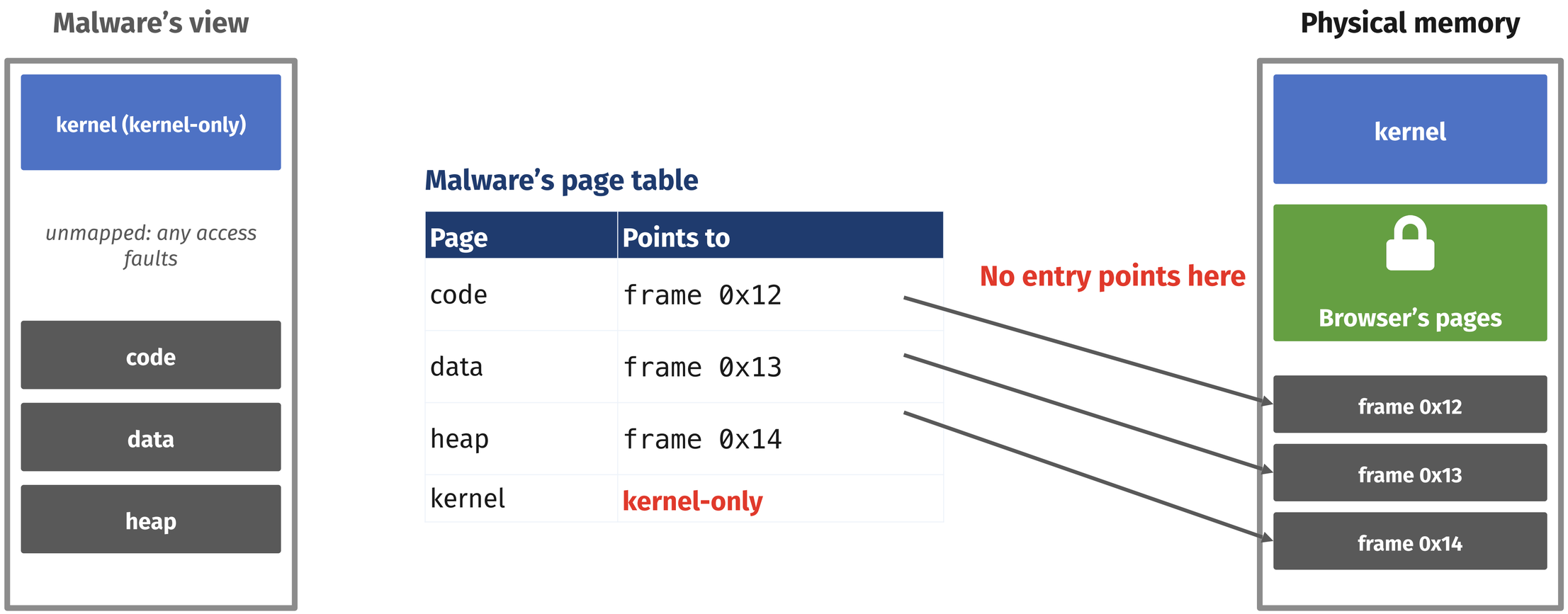
Page	Frame	Allowed
0	not mapped	—
1	7	read, execute (code)
2	4	read, write (data)
3	not loaded yet	read, write
F	0	kernel only

Answers

Access	Result
`Read 0x1234`	**Physical 0x7234**
`Write 0x2010`	**Physical 0x4010**
`Write 0x1234`	**SIGSEGV: code is read-only**
`Read 0x0004`	**SIGSEGV: unmapped (null pointer)**
`Write 0x3008`	**Page fault, handled: no crash**
`Read 0xF000`	**SIGSEGV: kernel only**

Process B: 0x2010 goes to 0x9010, the same address as A but different memory. To reach frame 4, B needs a table entry pointing there. It has none, and only the kernel can add one.

VM in action: the attacker's view



There is no address the malware can use to reach the browser. Every address hits its own pages or faults.

What virtual memory doesn’t protect

Bugs inside one process

A buffer overflow in the browser stays inside the browser’s own memory, where VM can’t help. Memory tagging (MTE, Apple’s MIE) targets exactly these bugs.

Kernel bugs

The kernel writes the page tables, so an exploit that runs code in the kernel can map anything it likes.

Hardware attacks

Meltdown read kernel-only pages through speculation. Rowhammer flips bits in RAM, and has been used to flip bits in page tables.

Devices using DMA

Devices read and write memory without going through the CPU’s MMU, unless the IOMMU translates and limits their accesses too.

Try it: look at an address space

```
$ cat /proc/self/maps                                  (trimmed)
5603f1a4b000-5603f1a4d000 r--p 00000000 08:02 1835   /usr/bin/cat
5603f1a4d000-5603f1a52000 r-xp 00002000 08:02 1835   /usr/bin/cat
5603f1a59000-5603f1a5a000 rw-p 0000d000 08:02 1835   /usr/bin/cat
5603f2c1e000-5603f2c3f000 rw-p 00000000 00:00 0      [heap]
7f2d8c800000-7f2d8c828000 r--p 00000000 08:02 2101   /usr/lib/x86_64-linux-gnu/libc.so.6
7f2d8c828000-7f2d8c9bd000 r-xp 00028000 08:02 2101   /usr/lib/x86_64-linux-gnu/libc.so.6
7ffc5b2e1000-7ffc5b302000 rw-p 00000000 00:00 0      [stack]
```

Read the permissions

r-xp is code: readable and executable, not writable. Is any region both writable and executable?

Run it twice

Every address changes. Why would the OS randomize the layout (ASLR)?

Look for the kernel

It never appears: it’s mapped, but kernel-only, so it isn’t part of the program’s own regions.

Try it: one address, two processes

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int x;

int main(int argc, char **argv) {
    x = atoi(argv[1]);
    printf("pid %d: &x = %p, x = %d\n",
           getpid(), (void *)&x, x);
    sleep(20);
    printf("pid %d: x is still %d\n", getpid(), x);
}
```

```
$ gcc -no-pie same.c -o same
$ ./same 1 & ./same 2
pid 4101: &x = 0x40402c, x = 1
pid 4102: &x = 0x40402c, x = 2
pid 4101: x is still 1
pid 4102: x is still 2
```

-no-pie fixes the program's address, so both copies put x in the same place.

Question

Same address, different values. Where does each x really live?

Each process has its own page table, so 0x40402c leads to a different physical page in each one.

Try it: decode a segfault

```
#include <stdlib.h>
int main(int argc, char **argv) {
    char *s = "hello";
    switch (argv[1][0]) {
    case 'r': return *(volatile char *)0;
    case 'w': *(volatile char *)0 = 1; break;
    case 's': s[0] = 'H'; break;
    case 'x': ((void (*)(void))malloc(64))(); break;
    }
}
```

```
$ ./segv w
Segmentation fault (core dumped)
$ sudo dmesg | tail -1
segv[5120]: segfault at 0 ip … error 6
```

Run	What it does	Predict: error
r	read a null pointer	**4**
w	write a null pointer	**6**
s	write a string literal	**7**
x	jump into heap data	**15**

The error is a hex bit field

```
1  the page was present (permissions)
2  it was a write
4  it came from user mode
10 it was an instruction fetch
```

In later courses

Each design fixed the previous one’s biggest problem

Base and bound

How

Add a base to every address; fault past a limit

Problem

One contiguous block per process: memory fragments

Segmentation

How

Several base-and-bound pairs: code, data, stack

Problem

Segments are still variable-sized and fragment

Paging

How

Fixed 4 KiB pages, mapped one by one

Problem

A flat table for 256 TiB would be huge

Multi-level tables + TLB

How

A tree of small tables, plus a cache of recent translations

Today

What x86-64 and Arm use today

Threat Model
Intro to System Architecture
Hardware Support for Isolation
Examples
Principles

Review

At this point, we have achieved process isolation

- Protected mode execution prevents direct device access
- Virtual memory prevents direct memory access

Requires CPU support

- All moderns CPUs support these techniques

Requires OS support

- All moderns OS support these techniques
- OS controls process rings and page tables



Review

At this point, we have achieved process isolation

- Protected mode execution prevents direct device access
- Virtual memory prevents direct memory access

Requires CPU support

- All moderns CPUs support these techniques

Requires OS support

- All moderns OS support these techniques
- OS controls process rings and page tables

Warning: bugs in the OS may compromise process isolation



Processes on a linux machine

```
PID TTY      STAT   TIME COMMAND
      1 ?        Ss     0:02 /sbin/init splash
      2 ?        S      0:00 [kthreadd]
      3 ?        I<     0:00 [rcu_gp]
      4 ?        I<     0:00 [rcu_par_gp]
      6 ?        I<     0:00 [kworker/0:0H-kblockd]
      7 ?        I      0:00 [kworker/0:1-events]
      9 ?        I<     0:00 [mm_percpu_wq]
     10 ?        S      0:00 [ksoftirqd/0]
     11 ?        I      0:00 [rcu_sched]
     12 ?        S      0:00 [migration/0]
     13 ?        S      0:00 [idle_inject/0]
     14 ?        S      0:00 [cpuhp/0]
     15 ?        S      0:00 [kdevtmpfs]
     16 ?        I<     0:00 [netns]
     17 ?        S      0:00 [rcu_tasks_kthre]
     18 ?        S      0:00 [kauditd]
     19 ?        S      0:00 [khungtaskd]
     20 ?        S      0:00 [oom_reaper]
     21 ?        I<     0:00 [writeback]
     22 ?        S      0:00 [kcompactd0]
     23 ?        SN     0:00 [ksmd]
     24 ?        SN     0:00 [khugepaged]
     70 ?        I<     0:00 [kintegrityd]
     71 ?        I<     0:00 [kblockd]
     72 ?        I<     0:00 [blkcg_punt_bio]
     73 ?        I<     0:00 [tpm_dev_wq]
     74 ?        I<     0:00 [ata_sff]
     75 ?        I<     0:00 [md]
     76 ?        I<     0:00 [edac-poller]
     77 ?        I<     0:00 [devfreq_wq]
     78 ?        S      0:00 [watchdogd]
     81 ?        S      0:00 [kswapd0]
     82 ?        S      0:00 [ecryptfs-kthrea]
     84 ?        I<     0:00 [kthrotld]
     85 ?        I<     0:00 [acpi_thermal_pm]
     86 ?        S      0:00 [scsi_eh_0]
     87 ?        I<     0:00 [scsi_tmf_0]
     88 ?        S      0:00 [scsi_eh_1]
     89 ?        I<     0:00 [scsi_tmf_1]
     91 ?        I<     0:00 [vfio-irqfd-clea]
     92 ?        I      0:00 [kworker/u2:3-events_unbound]
     93 ?        I<     0:00 [ipv6_addrconf]
    102 ?        I<     0:00 [kstrp]
    105 ?        I<     0:00 [kworker/u3:0]
    118 ?        I<     0:00 [charger_manager]
    160 ?        I      0:00 [kworker/0:2-mm_percpu_wq]
    163 ?        S      0:00 [scsi_eh_2]
    164 ?        I<     0:00 [scsi_tmf_2]
    165 ?        I<     0:00 [kworker/0:1H-kblockd]
    186 ?        S      0:00 [jbd2/sda5-8]
    187 ?        I<     0:00 [ext4-rsv-conver]
    226 ?        S<s    0:00 /lib/systemd/systemd-journald
    250 ?        Ss     0:00 /lib/systemd/systemd-udevd
    251 ?        S      0:00 [irq/18-vmwgfx]
    252 ?        I<     0:00 [ttm_swap]
    261 ?        S<     0:00 [loop0]
    268 ?        S<     0:00 [loop3]
    279 ?        S<     0:00 [loop4]
    283 ?        S<     0:00 [loop5]
    296 ?        S<     0:00 [loop7]
    297 ?        S<     0:00 [loop8]
    298 ?        S<     0:00 [loop9]
    300 ?        S<     0:00 [loop10]
    302 ?        S<     0:00 [loop11]
    303 ?        S<     0:00 [loop12]
    332 ?        I<     0:00 [iprt-VBoxWQueue]
    355 ?        I<     0:00 [cryptd]
    547 ?        Ss     0:00 /lib/systemd/systemd-resolved
    548 ?        Ssl    0:00 /lib/systemd/systemd-timesyncd
    585 ?        Ssl    0:00 /usr/lib/accountsservice/accounts-daemon
    586 ?        Ss     0:00 /usr/sbin/acpid
    591 ?        Ss     0:00 avahi-daemon: running [abhi-VirtualBox.local]
    592 ?        Ss     0:00 /usr/sbin/cron -f
    596 ?        Ss     0:01 /usr/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile —systemd-activa…
    597 ?        Ssl    0:00 /usr/sbin/NetworkManager --no-daemon
    617 ?        Ss     0:00 /usr/bin/python3 /usr/bin/networkd-dispatcher --run-startup-triggers
    618 ?        Ssl    0:00 /usr/lib/policykit-1/polkitd --no-debug
    625 ?        Ssl    0:00 /usr/sbin/rsyslogd -n -iNONE
    628 ?        Ssl    0:12 /usr/lib/snapd/snapd
    629 ?        Ssl    0:00 /usr/libexec/switcheroo-control
    630 ?        Ss     0:00 /lib/systemd/systemd-logind
    634 ?        Ssl    0:00 /usr/lib/udisks2/udisksd
    641 ?        Ss     0:00 /sbin/wpa_supplicant -u -s -O /run/wpa_supplicant
    653 ?        S      0:00 avahi-daemon: chroot helper
    699 ?        Ssl    0:00 /usr/sbin/ModemManager --filter-policy=strict
    703 ?        Ssl    0:00 /usr/sbin/cups-browsed
    717 ?        Ss     0:00 /usr/sbin/cupsd -l
    720 ?        Ssl    0:00 /usr/bin/python3 /usr/share/unattended-upgrades/unattended-upgrade-shutdown --wait-for-signal
    753 ?        Ssl    0:00 /usr/bin/whoopsie -f
    758 ?        Ss     0:00 /usr/sbin/kerneloops --test
    763 ?        Ss     0:00 /usr/sbin/kerneloops
    961 ?        Ssl    0:00 /usr/sbin/gdm3
    974 ?        Sl     0:00 /usr/sbin/VBoxService --pidfile /var/run/vboxadd-service.sh
   1036 ?        SNsl   0:00 /usr/libexec/rtkit-daemon
   1097 ?        Ssl    0:00 /usr/lib/upower/upowerd
   1297 ?        Ssl    0:00 /usr/libexec/colord
   1350 ?        Sl     0:00 gdm-session-worker [pam/gdm-password]
   1369 ?        Ss     0:00 /lib/systemd/systemd --user
   1370 ?        S      0:00 (sd-pam)
   1379 ?        S<sl   0:00 /usr/bin/pulseaudio --daemonize=no --log-target=journal
   1382 ?        Sl     0:00 /usr/bin/gnome-keyring-daemon --daemonize --login
   1385 ?        SNsl   0:00 /usr/libexec/tracker-miner-fs
   1387 ?        Ss     0:00 /usr/bin/dbus-daemon --session --address=systemd: --nofork --nopidfile —systemd-ac…
   1391 ?        Ssl    0:00 /usr/libexec/gvfsd
   1396 ?        Sl     0:00 /usr/libexec/gvfsd-fuse /run/user/1000/gvfs -f -o big_writes
   1409 ?        Ssl    0:00 /usr/libexec/gvfs-udisks2-volume-monitor
   1420 tty2     Ssl+   0:00 /usr/lib/gdm3/gdm-x-session --run-script env GNOME_SHELL_SESSION_MODE=ubuntu /usr…
   1423 tty2     Sl+    0:02 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/Xauthority …
   1424 ?        Ssl    0:00 /usr/libexec/gvfs-afc-volume-monitor
   1430 ?        Ssl    0:00 /usr/libexec/gvfs-gphoto2-volume-monitor
   1434 ?        Ssl    0:00 /usr/libexec/gvfs-goa-volume-monitor
   1438 ?        Sl     0:00 /usr/libexec/goa-daemon
   1446 ?        Sl     0:00 /usr/libexec/goa-identity-service
   1449 ?        Ssl    0:00 /usr/libexec/gvfs-mtp-volume-monitor
   1478 tty2     Sl+    0:00 /usr/libexec/gnome-session-binary --systemd --systemd --session=ubuntu
   1550 ?        S      0:00 /usr/bin/VBoxClient --clipboard
   1551 ?        Sl     0:00 /usr/bin/VBoxClient --clipboard
   1562 ?        S      0:00 /usr/bin/VBoxClient --seamless
   1563 ?        Sl     0:00 /usr/bin/VBoxClient --seamless
   1569 ?        S      0:00 /usr/bin/VBoxClient --draganddrop
   1570 ?        Sl     0:02 /usr/bin/VBoxClient --draganddrop
   1577 ?        S      0:00 /usr/bin/VBoxClient --vmsvga
   1578 ?        Sl     0:00 /usr/bin/VBoxClient --vmsvga
   1585 ?        Ss     0:00 /usr/bin/ssh-agent /usr/bin/im-launch env GNOME_SHELL_SESSION_MODE=ubuntu /usr/bin/gnome-session --systemd --session=ubuntu
   1604 ?        Ssl    0:00 /usr/libexec/at-spi-bus-launcher
   1610 ?        S      0:00 /usr/bin/dbus-daemon --config-file=/usr/share/defaults/at-spi2/accessibility.conf --nofork --print-address 3
   1614 ?        Ssl    0:00 /usr/libexec/gnome-session-ctl --monitor
   1621 ?        Ssl    0:00 /usr/libexec/gnome-session-binary --systemd-service --session=ubuntu
   1635 ?        Ssl    0:07 /usr/bin/gnome-shell
   1678 ?        Sl     0:00 ibus-daemon --panel disable --xim
   1689 ?        Sl     0:00 /usr/libexec/ibus-dconf
   1690 ?        Sl     0:00 /usr/libexec/ibus-extension-gtk3
   1693 ?        Sl     0:00 /usr/libexec/ibus-x11 --kill-daemon
   1695 ?        Sl     0:00 /usr/libexec/ibus-portal
   1731 ?        Sl     0:00 /usr/libexec/at-spi2-registryd --use-gnome-session
   1737 ?        S<     0:00 [loop13]
   1740 ?        Ssl    0:00 /usr/libexec/xdg-permission-store
   1745 ?        Sl     0:00 /usr/libexec/gnome-shell-calendar-server
   1751 ?        Ssl    0:00 /usr/libexec/evolution-source-registry
   1762 ?        Ssl    0:00 /usr/libexec/evolution-calendar-factory
   1771 ?        Sl     0:00 /usr/libexec/dconf-service
   1776 ?        Ssl    0:00 /usr/libexec/evolution-addressbook-factory
   1804 ?        Sl     0:00 /usr/libexec/gvfsd-trash --spawner :1.3 /org/gtk/gvfs
   1807 ?        Sl     0:00 /usr/bin/gjs /usr/share/gnome-shell/org.gnome.Shell.N
   1847 ?        Ssl    0:00 /usr/libexec/gsd-a11y-settings
   1850 ?        Ssl    0:00 /usr/libexec/gsd-color
   1851 ?        Ssl    0:00 /usr/libexec/gsd-datetime
   1856 ?        Ssl    0:00 /usr/libexec/gsd-housekeeping
   1858 ?        Ssl    0:00 /usr/libexec/gsd-keyboard
   1861 ?        Ssl    0:00 /usr/libexec/gsd-media-keys
   1863 ?        Ssl    0:00 /usr/libexec/gsd-power
   1864 ?        Ssl    0:00 /usr/libexec/gsd-print-notifications
   1866 ?        Ssl    0:00 /usr/libexec/gsd-rfkill
   1875 ?        Ssl    0:00 /usr/libexec/gsd-screensaver-proxy
   1879 ?        Ssl    0:00 /usr/libexec/gsd-sharing
   1881 ?        Ssl    0:00 /usr/libexec/gsd-smartcard
   1885 ?        Sl     0:00 /usr/libexec/evolution-data-server/evolution-alarm-no
   1890 ?        Ssl    0:00 /usr/libexec/gsd-sound
   1895 ?        Ssl    0:00 /usr/libexec/gsd-usb-protection
   1899 ?        Ssl    0:00 /usr/libexec/gsd-wacom
   1902 ?        Ssl    0:00 /usr/libexec/gsd-wwan
   1904 ?        Ssl    0:00 /usr/libexec/gsd-xsettings
   1930 ?        Sl     0:00 /usr/libexec/gsd-disk-utility-notify
   1947 ?        Sl     0:00 /usr/libexec/gsd-printer
   2005 ?        Sl     0:00 /usr/libexec/ibus-engine-simple
   2137 ?        S<     0:00 [loop2]
   2200 ?        Ssl    0:01 /usr/libexec/gnome-terminal-server
   2208 pts/0    Ss     0:00 bash
   2338 ?        S<     0:00 [loop1]
   2389 ?        Ssl    0:00 /usr/libexec/gvfsd-metadata
   2392 ?        Sl     0:00 update-notifier
   2458 ?        SNl    0:02 /usr/bin/python3 /usr/bin/update-manager --no-update
   2610 ?        I      0:00 [kworker/u2:0-events_unbound]
   2707 ?        I      0:00 [kworker/0:0-events]
   2708 ?        I      0:00 [kworker/0:3-cgroup_destroy]
   3107 ?        I      0:00 [kworker/u2:1]
   3248 ?        Ss     0:00 sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startu
   4244 ?        Ssl    0:00 /usr/lib/packagekit/packagekitd
   4334 pts/0    R+     0:00 ps ax
```

Towards Secure Systems

Now that we have process isolation, we can build more complex security features



File Access Control



Anti-virus



Firewall

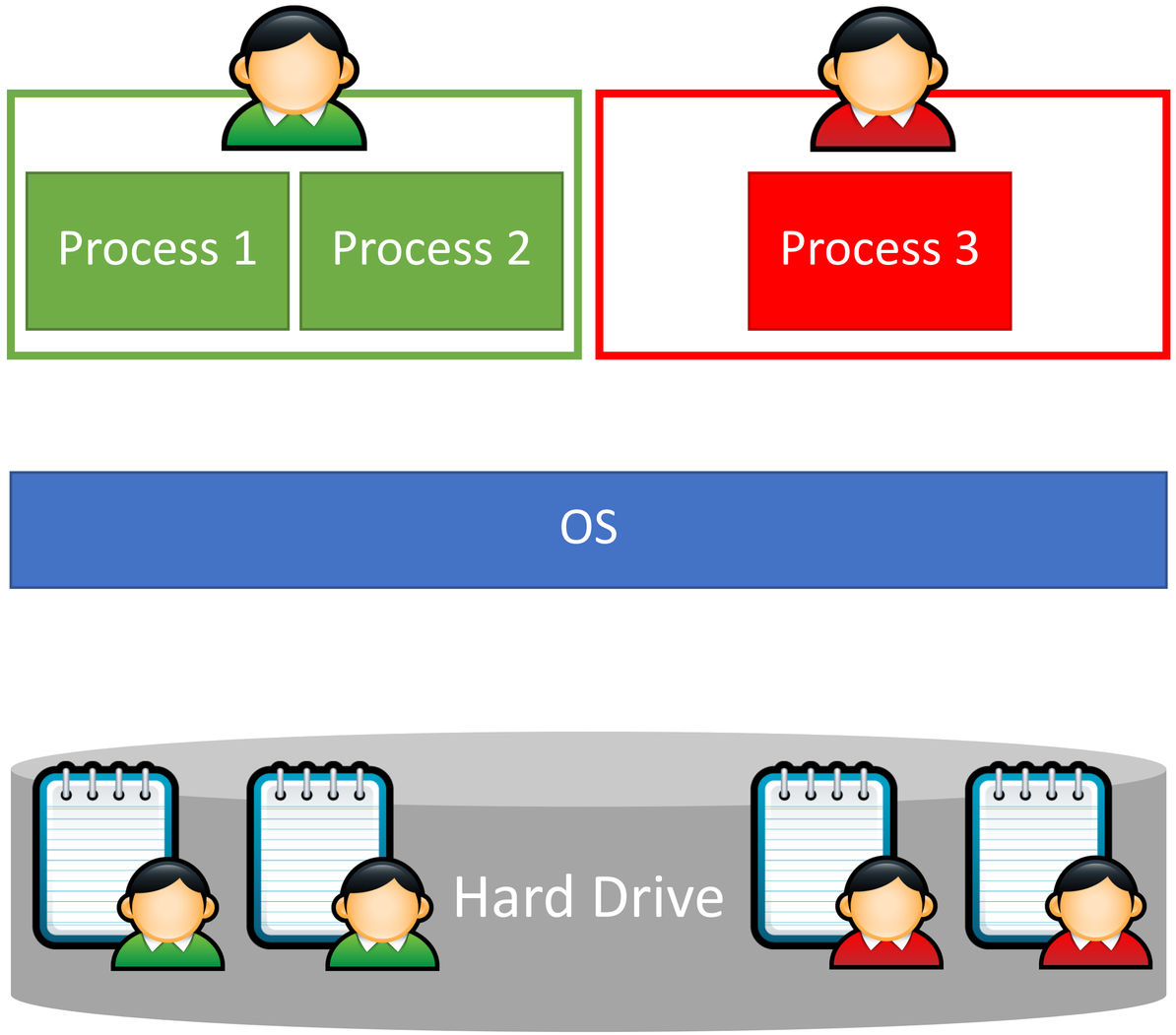


Secure Logging

File Access Control

All disk access is mediated by the OS

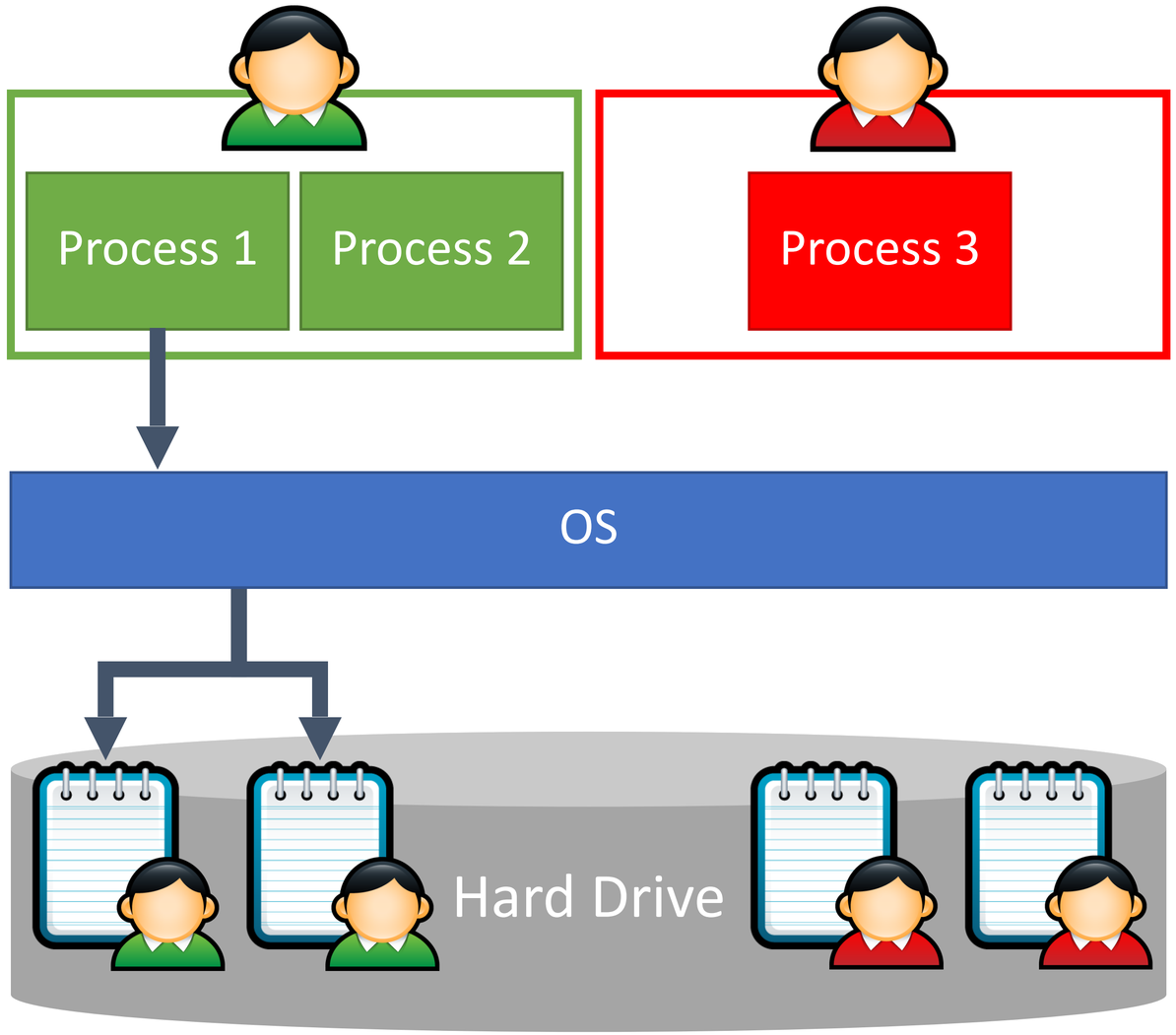
OS enforces access controls



File Access Control

All disk access is mediated by the OS

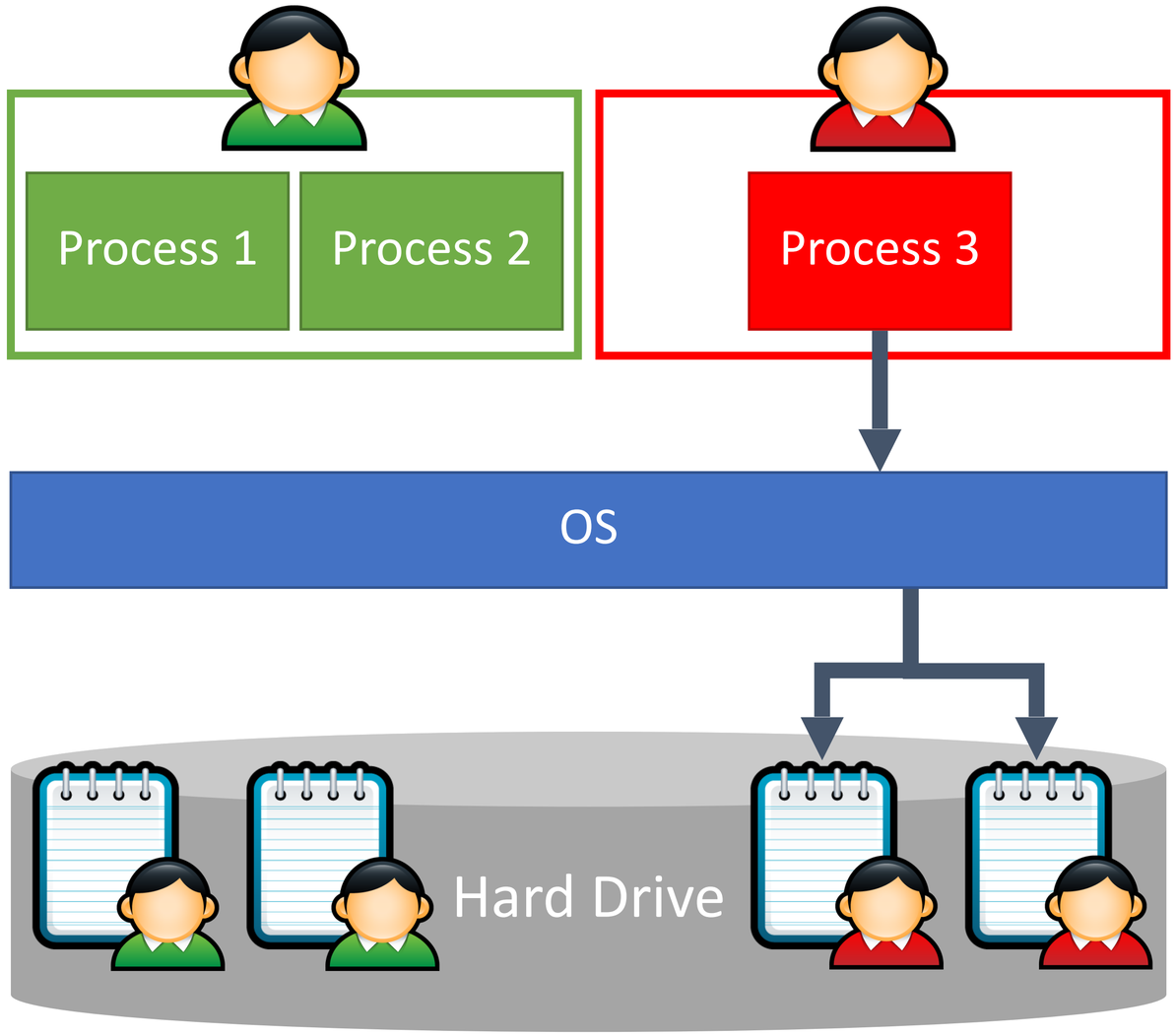
OS enforces access controls



File Access Control

All disk access is mediated by the OS

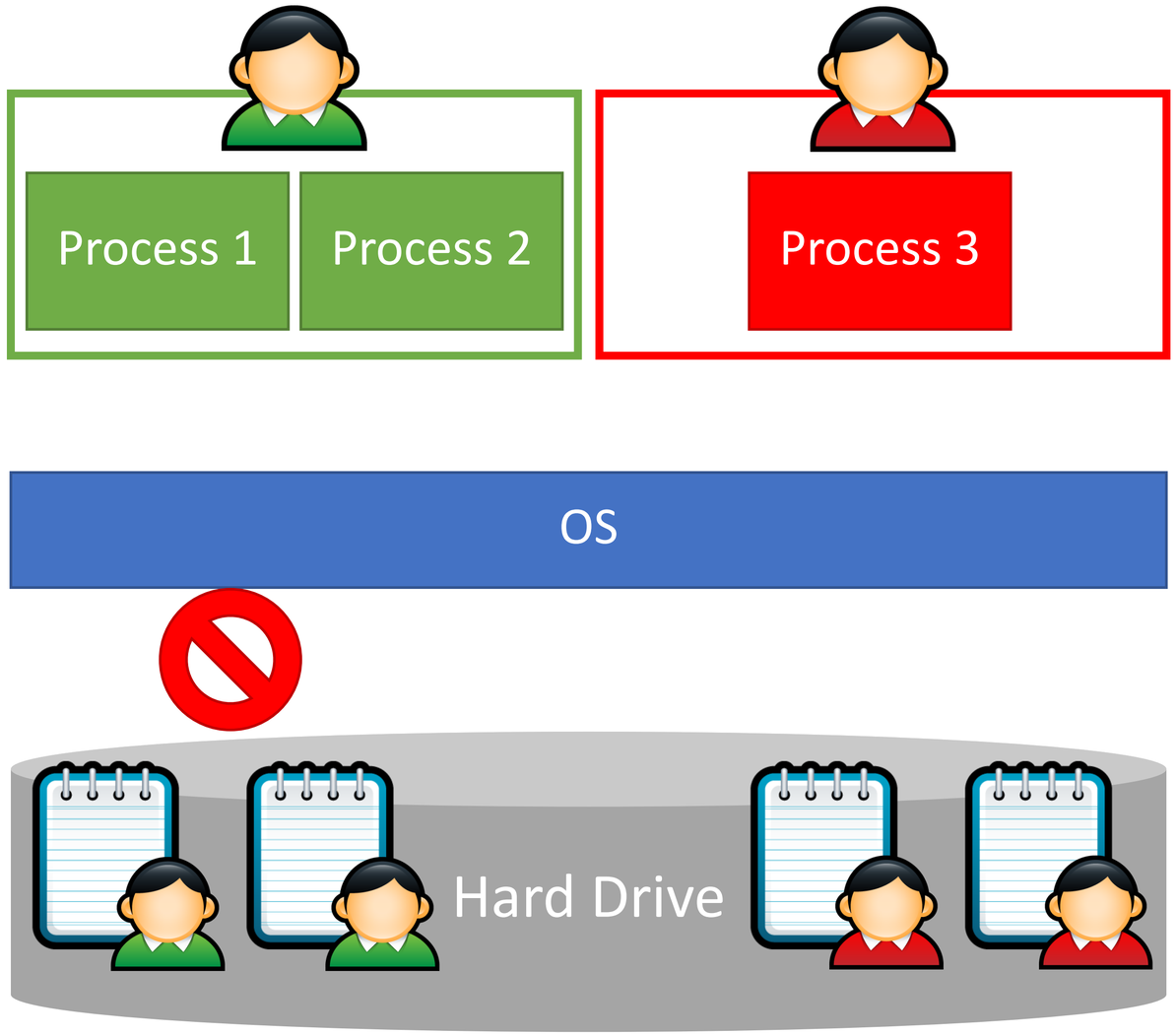
OS enforces access controls



File Access Control

All disk access is mediated by the OS

OS enforces access controls

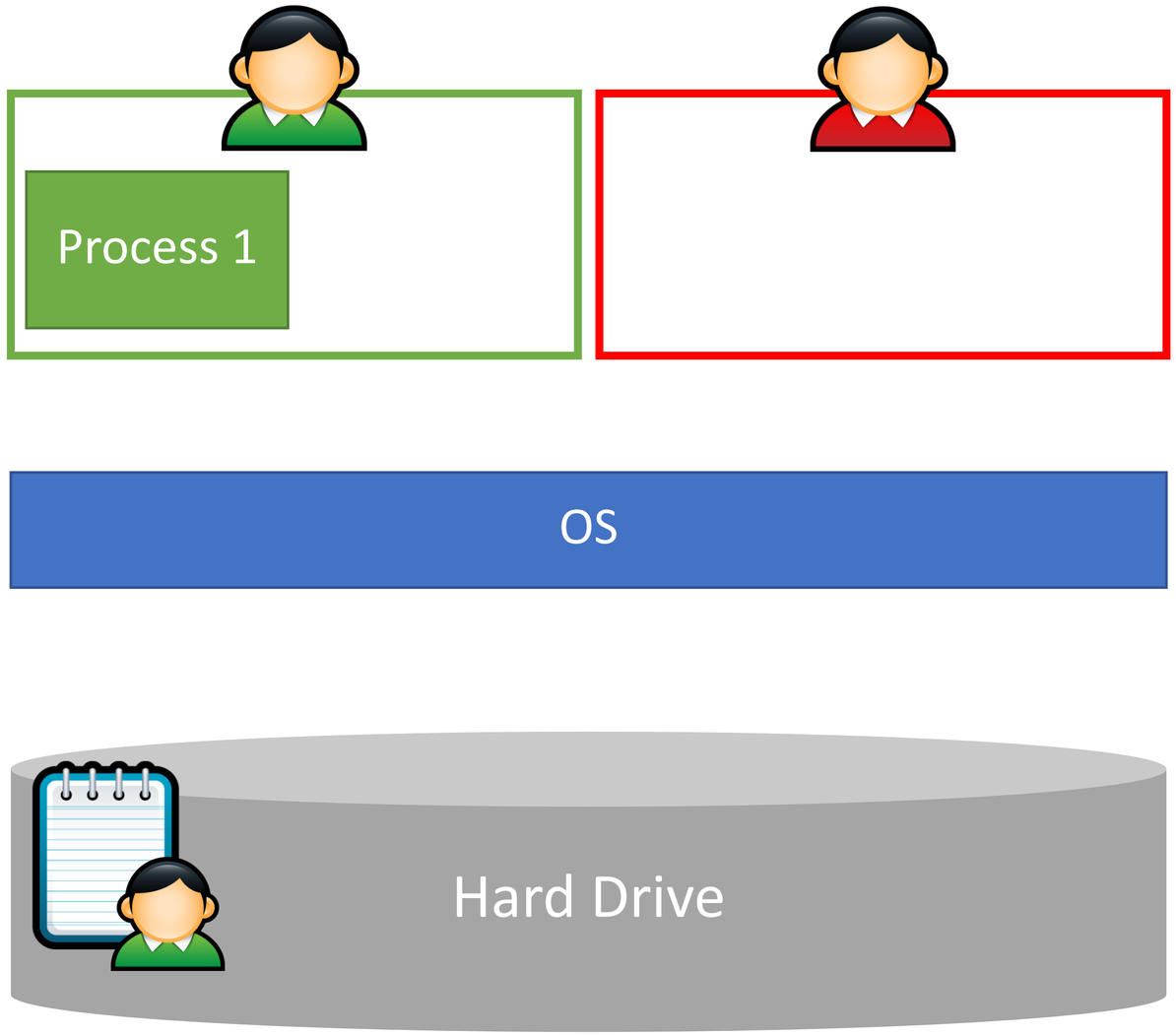


Insecure Logging

Suppose Process 1 writes information to a log file

Malware can still destroy the log

- Add or remove entries
- Add fake entries
- Delete the whole log

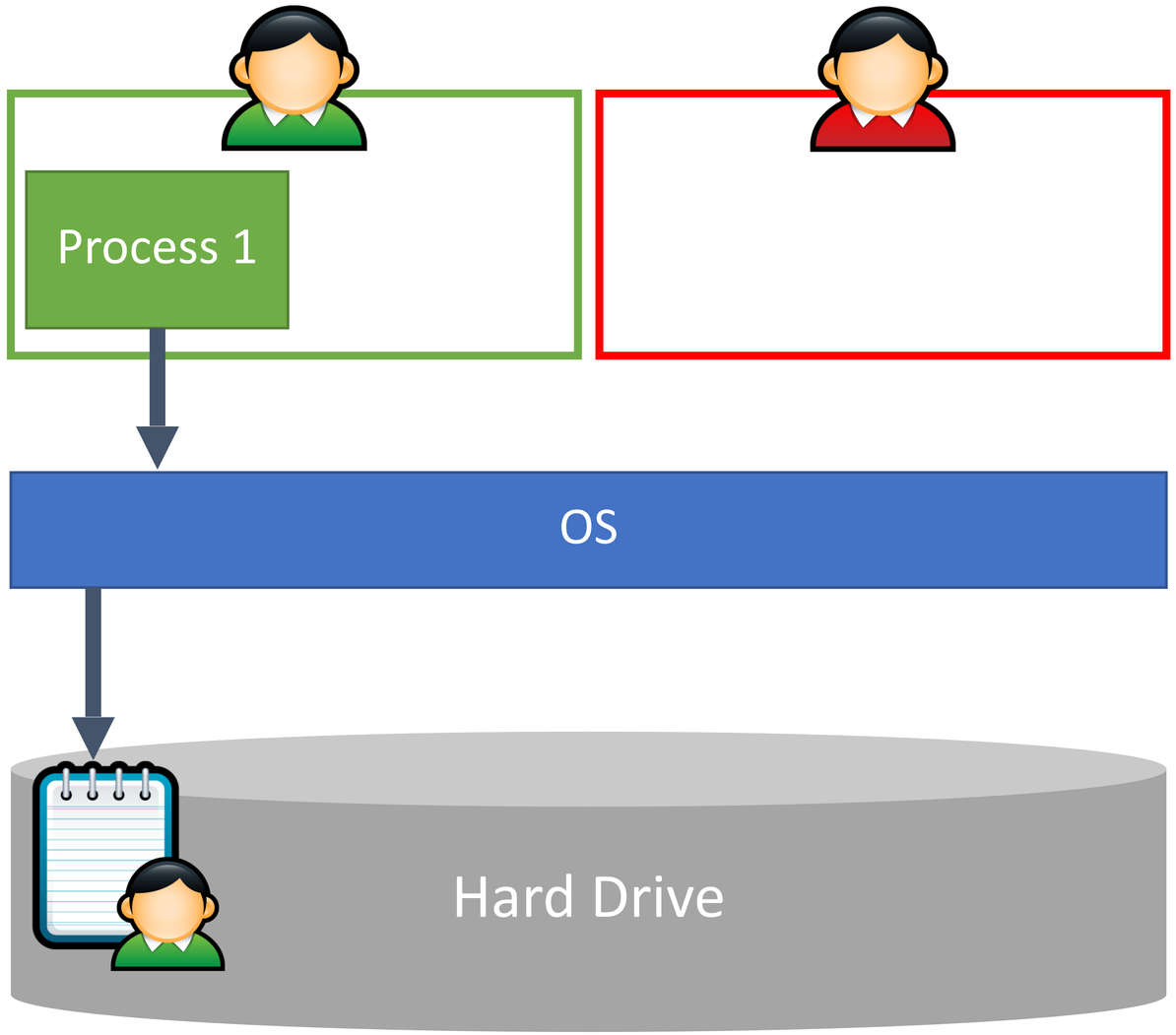


Insecure Logging

Suppose Process 1 writes information to a log file

Malware can still destroy the log

- Add or remove entries
- Add fake entries
- Delete the whole log

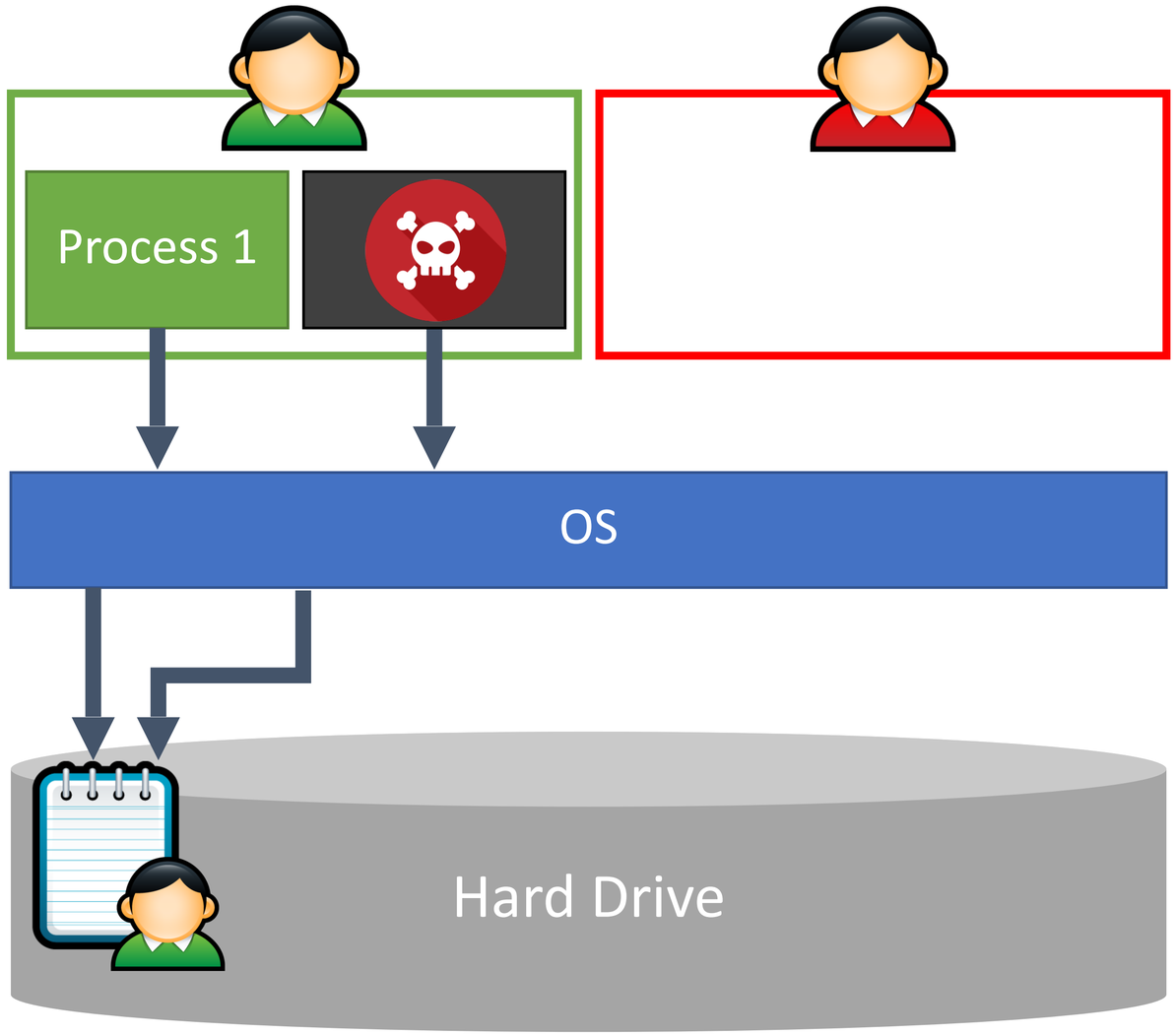


Insecure Logging

Suppose Process 1 writes information to a log file

Malware can still destroy the log

- Add or remove entries
- Add fake entries
- Delete the whole log

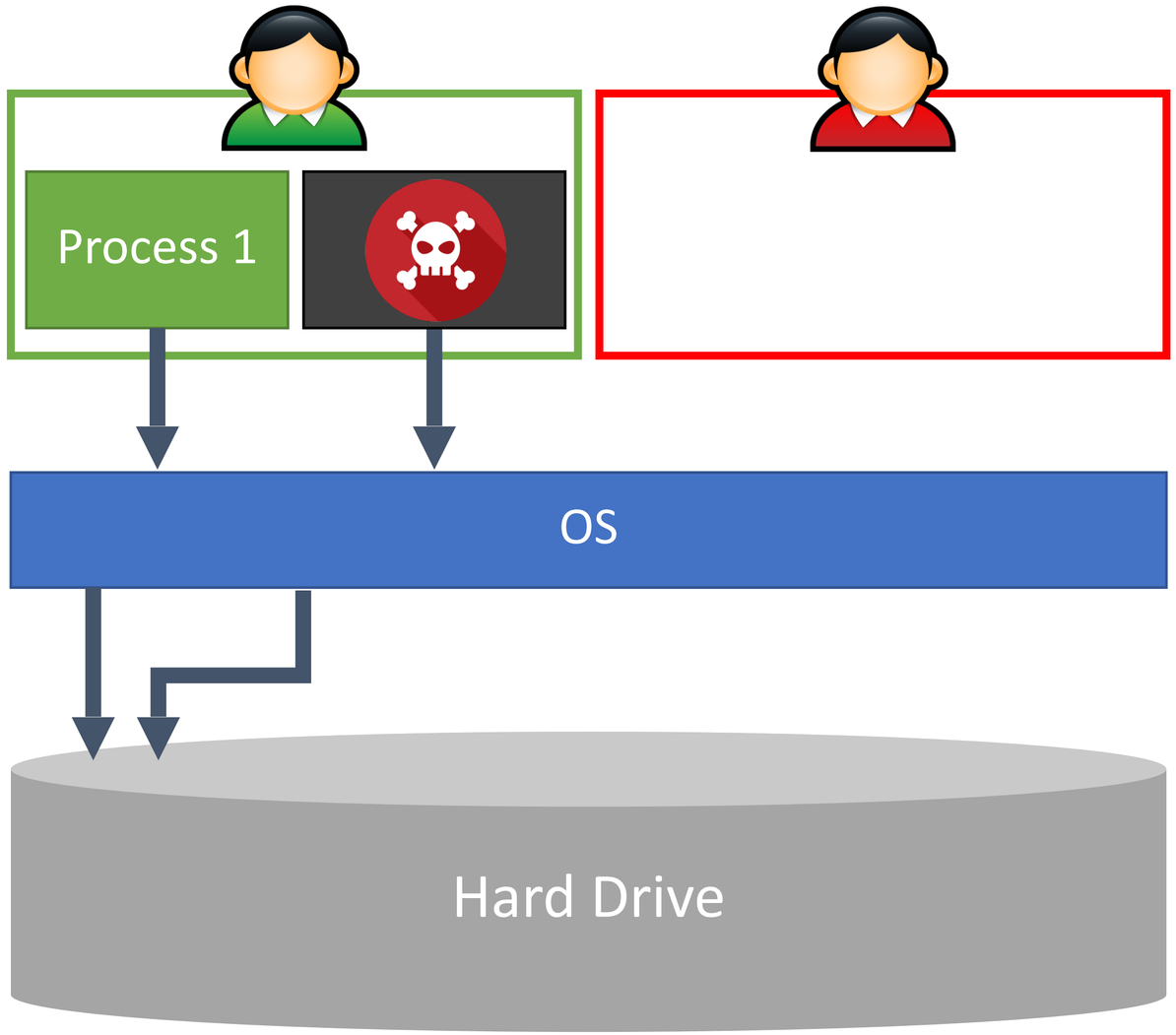


Insecure Logging

Suppose Process 1 writes information to a log file

Malware can still destroy the log

- Add or remove entries
- Add fake entries
- Delete the whole log



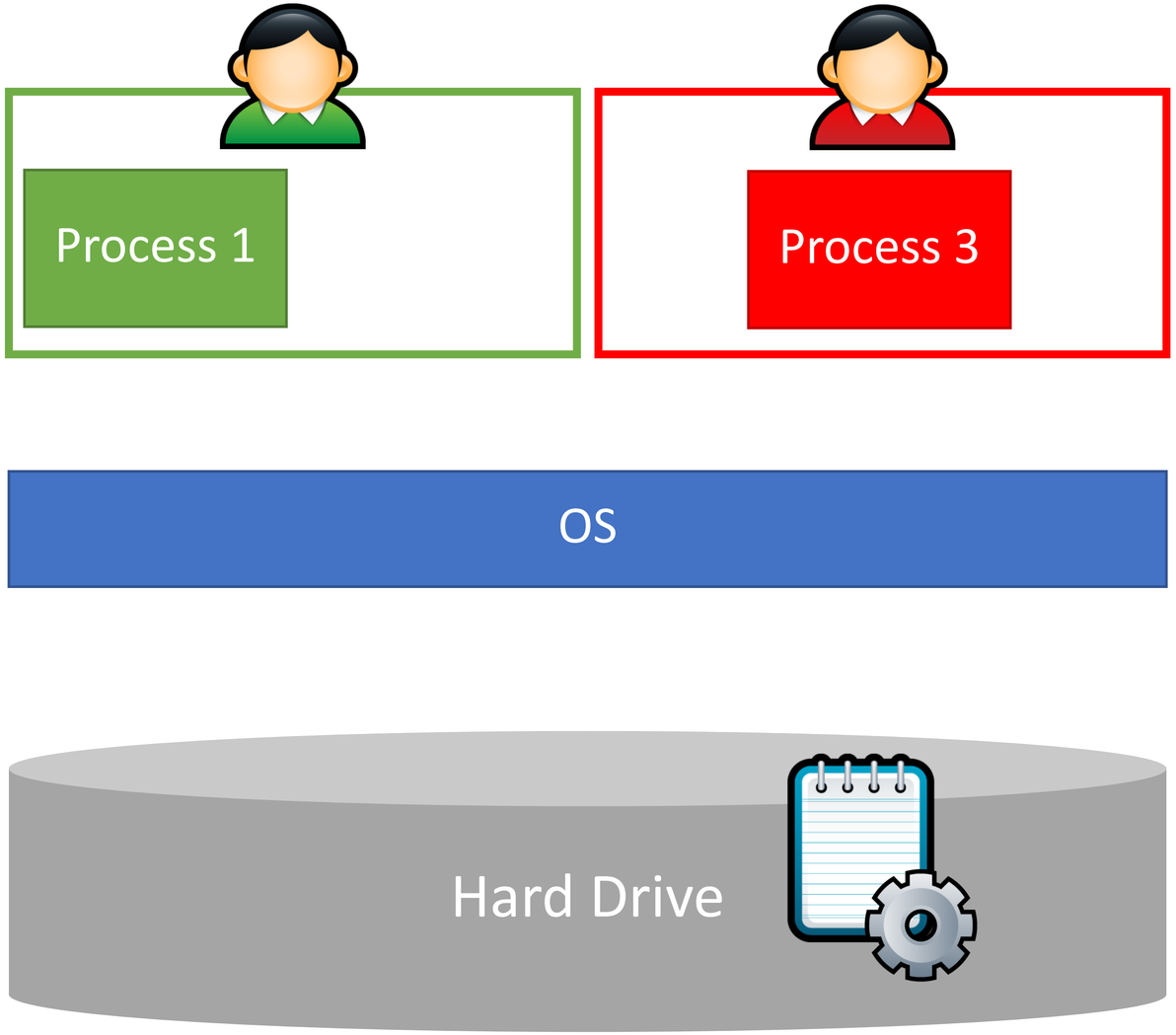
Secure Logging



OS maintains a system log

Processes may write entries to the log using an OS API

Processes may not delete entries or the log



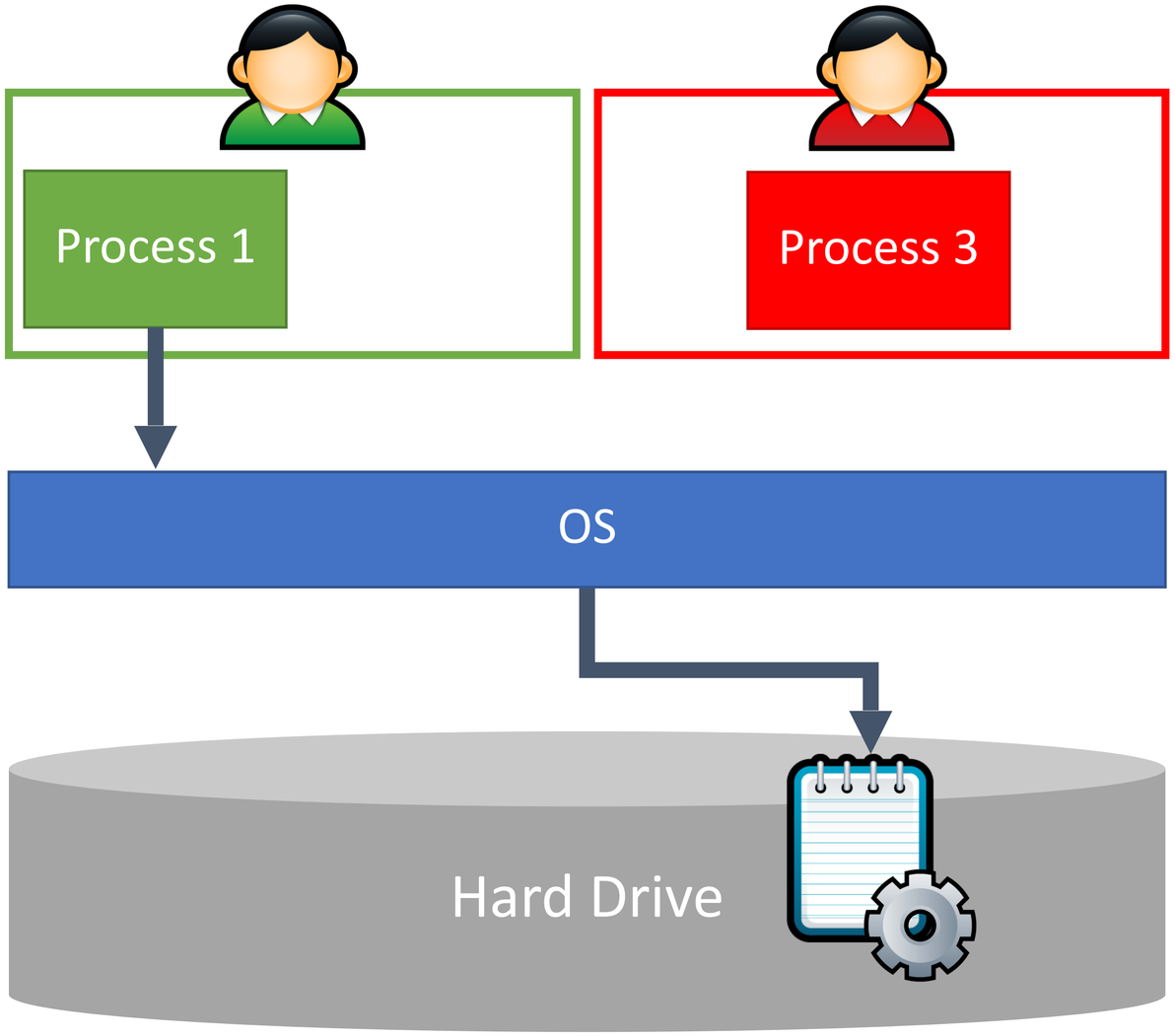
Secure Logging



OS maintains a system log

Processes may write entries to the log using an OS API

Processes may not delete entries or the log



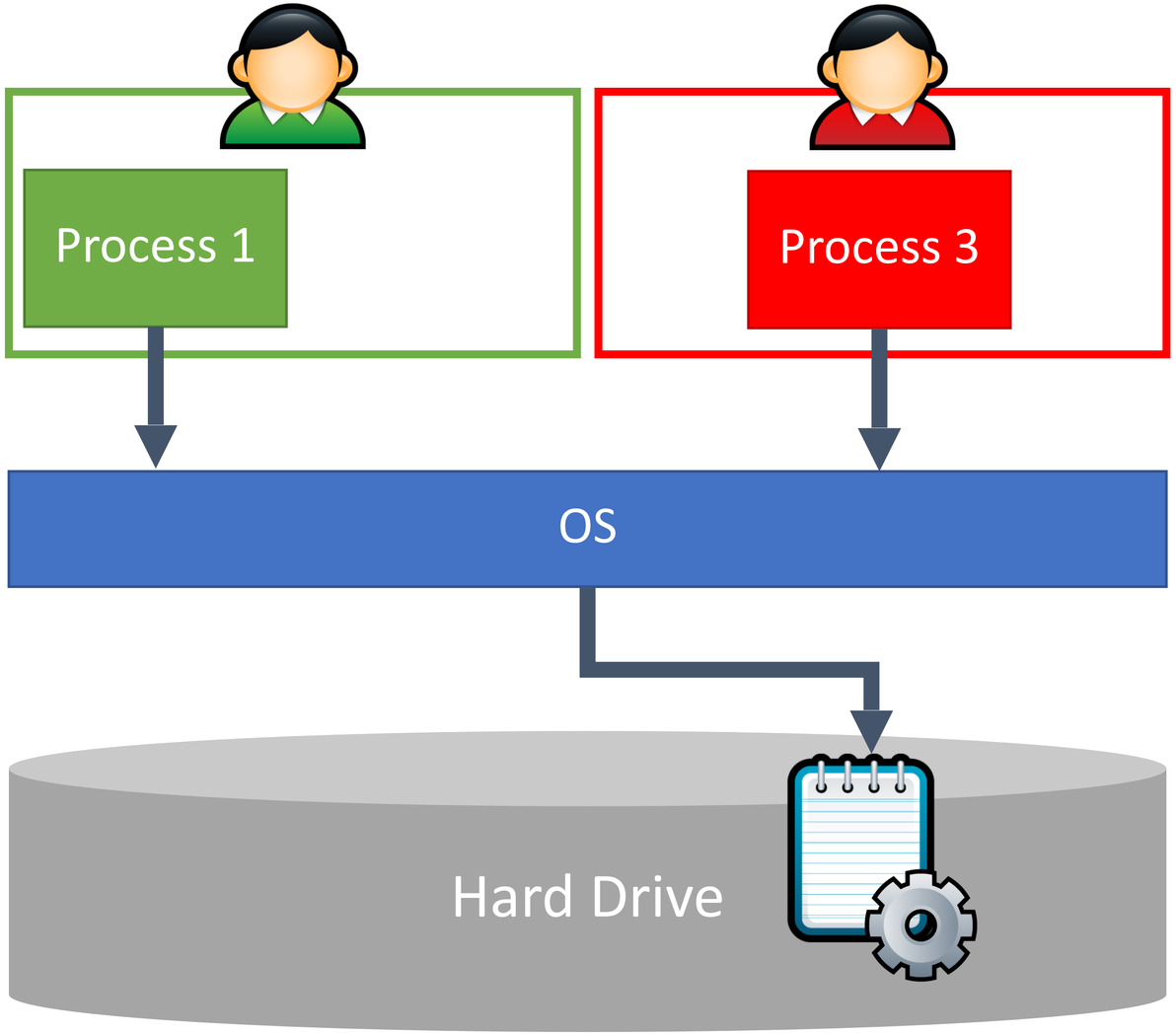
Secure Logging



OS maintains a system log

Processes may write entries to the log using an OS API

Processes may not delete entries or the log



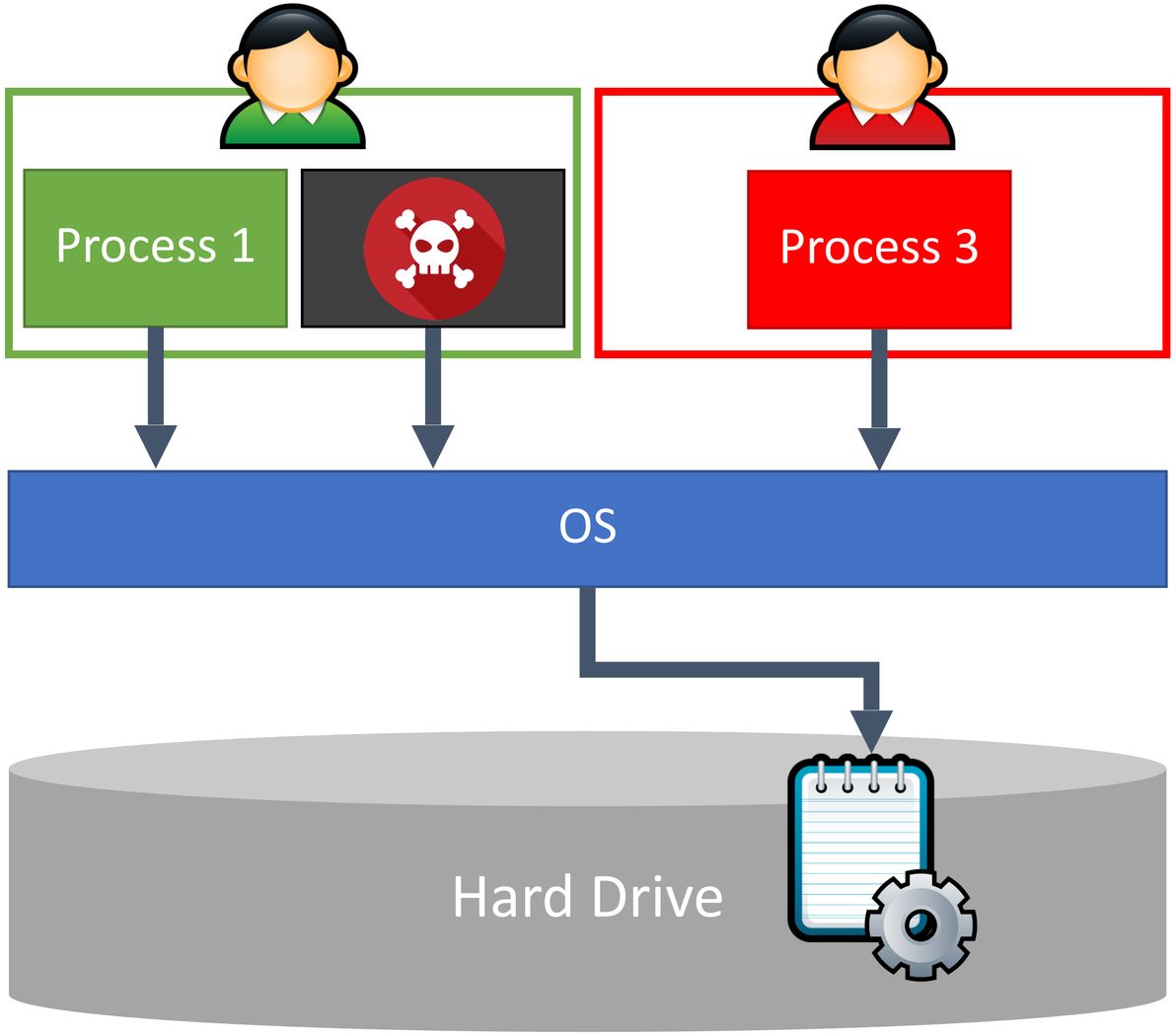
Secure Logging



OS maintains a system log

Processes may write entries to the log using an OS API

Processes may not delete entries or the log



Linux logging

```
abhi@abhi-VirtualBox:~$ ls /var/log
alternatives.log        boot.log       dist-upgrade    gdm3              private              vboxadd-install.log
alternatives.log.1      boot.log.1     dmesg           gpu-manager.log   speech-dispatcher    vboxadd-setup.log
alternatives.log.2.gz   boot.log.2     dmesg.0         hp                syslog               vboxadd-setup.log.1
apparmor                boot.log.3     dmesg.1.gz      installer         syslog.1             vboxadd-setup.log.2
apport.log              boot.log.4     dmesg.2.gz      journal           syslog.2.gz          vboxadd-setup.log.3
apport.log.1            boot.log.5     dmesg.3.gz      kern.log          syslog.3.gz          vboxadd-setup.log.4
apt                     boot.log.6     dmesg.4.gz      kern.log.1        syslog.4.gz          vboxadd-uninstall.log
auth.log                boot.log.7     dpkg.log        kern.log.2.gz     syslog.5.gz          wtmp
auth.log.1              bootstrap.log  dpkg.log.1      kern.log.3.gz     syslog.6.gz
auth.log.2.gz           btmp           dpkg.log.2.gz   kern.log.4.gz     syslog.7.gz
auth.log.3.gz           btmp.1         faillog         lastlog           ubuntu-advantage.log
auth.log.4.gz           cups           fontconfig.log  openvpn           unattended-upgrades
abhi@abhi-VirtualBox:~$
```

Syslog

```
/var/log/syslog

/var/log/auth.log


/var/log/kern.log

/var/log/cron
```

syslog

```
Nov 10 10:20:24 abhi-VirtualBox systemd[1]: Reloading.
Nov 10 10:20:24 abhi-VirtualBox systemd[1]: /lib/systemd/system/dbus.socket:5: ListenStream= references a path below legacy directory /var/run/, updating /
var/run/dbus/system_bus_socket → /run/dbus/system_bus_socket; please update the unit file accordingly.
Nov 10 10:20:24 abhi-VirtualBox systemd[1]: Starting Daily apt upgrade and clean activities…
Nov 10 10:20:24 abhi-VirtualBox systemd[1]: Starting OpenBSD Secure Shell server…
Nov 10 10:20:24 abhi-VirtualBox systemd[1]: Started OpenBSD Secure Shell server.
Nov 10 10:20:24 abhi-VirtualBox systemd[1]: apt-daily-upgrade.service: Succeeded.
Nov 10 10:20:24 abhi-VirtualBox systemd[1]: Finished Daily apt upgrade and clean activities.
Nov 10 10:20:24 abhi-VirtualBox systemd[1]: Reloading.
Nov 10 10:20:24 abhi-VirtualBox systemd[1]: /lib/systemd/system/dbus.socket:5: ListenStream= references a path below legacy directory /var/run/, updating /
var/run/dbus/system_bus_socket → /run/dbus/system_bus_socket; please update the unit file accordingly.
Nov 10 10:20:28 abhi-VirtualBox dbus-daemon[596]: [system] Activating via systemd: service name='org.freedesktop.PackageKit' unit='packagekit.service'
requested by ':1.111' (uid=0 pid=4241 comm="/usr/bin/gdbus call --system --dest org.freedeskto" label="unconfined")
```

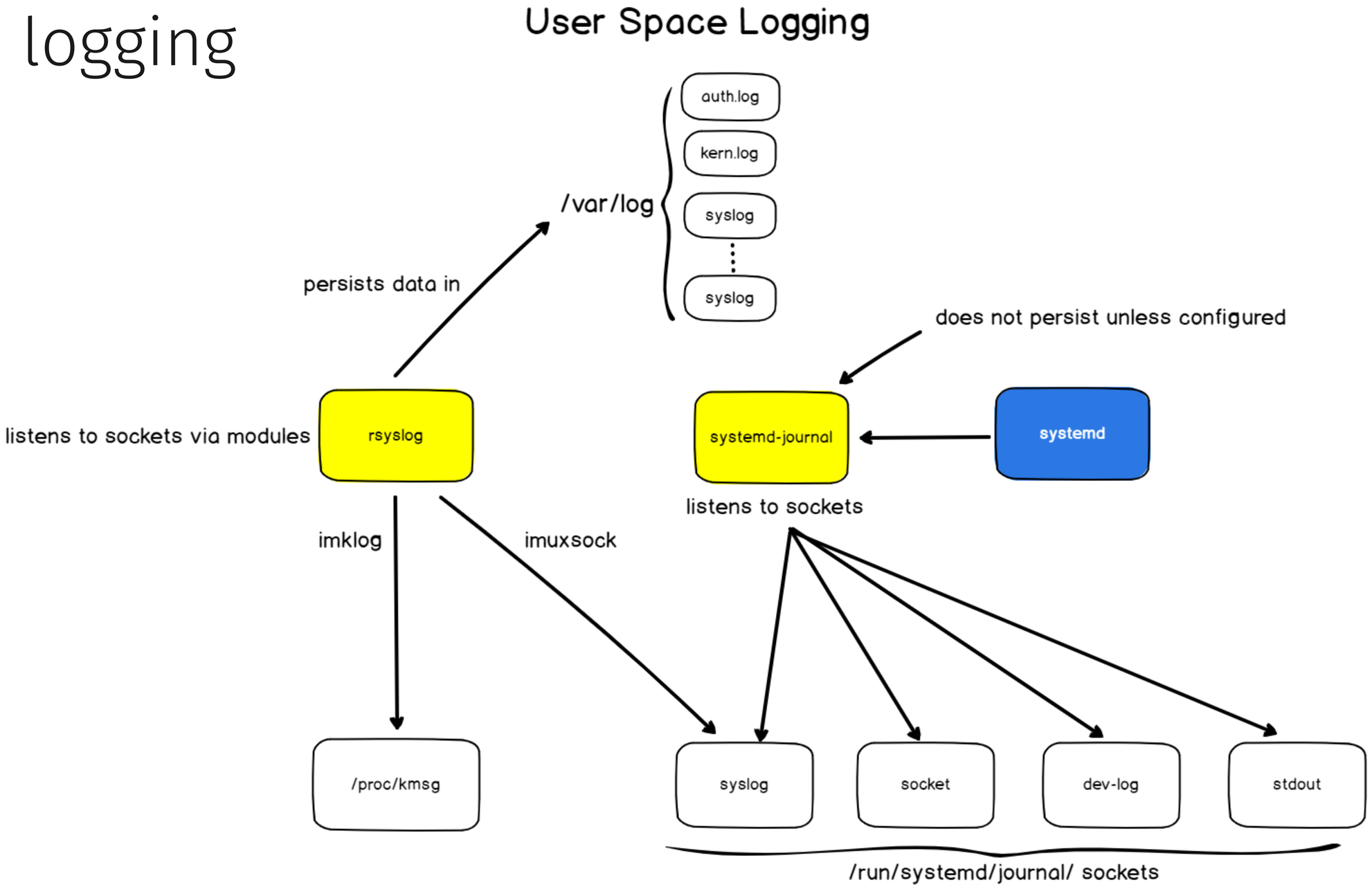
auth.log

```
Nov 10 10:18:55 abhi-VirtualBox sudo: pam_unix(sudo:session): session opened for user root by (uid=0)
Nov 10 10:18:55 abhi-VirtualBox sudo: pam_unix(sudo:session): session closed for user root
Nov 10 10:19:01 abhi-VirtualBox sudo:     abhi : TTY=pts/0 ; PWD=/home/abhi ; USER=root ; COMMAND=/usr/bin/apt install sshd
Nov 10 10:19:01 abhi-VirtualBox sudo: pam_unix(sudo:session): session opened for user root by (uid=0)
Nov 10 10:19:02 abhi-VirtualBox sudo: pam_unix(sudo:session): session closed for user root
Nov 10 10:20:16 abhi-VirtualBox sudo:     abhi : TTY=pts/0 ; PWD=/home/abhi ; USER=root ; COMMAND=/usr/bin/apt install openssh-server
Nov 10 10:20:16 abhi-VirtualBox sudo: pam_unix(sudo:session): session opened for user root by (uid=0)
Nov 10 10:20:22 abhi-VirtualBox useradd[3079]: new user: name=sshd, UID=126, GID=65534, home=/run/sshd, shell=/usr/sbin/nologin, from=none
Nov 10 10:20:22 abhi-VirtualBox usermod[3087]: change user 'sshd' password
Nov 10 10:20:22 abhi-VirtualBox chage[3094]: changed password expiry for sshd
Nov
```

Kernel logging

```
[    0.000406] MTRR variable ranges enabled:
[    0.000409]   0 base 0080000000 mask 7F80000000 uncachable
[    0.000410]   1 base 007C000000 mask 7FFC000000 uncachable
[    0.000412]   2 base 007A000000 mask 7FFE000000 uncachable
[    0.000413]   3 base 0079000000 mask 7FFF000000 uncachable
[    0.000415]   4 base 0078800000 mask 7FFF800000 uncachable
[    0.000417]   5 base 2000000000 mask 6000000000 uncachable
[    0.000418]   6 base 1000000000 mask 7000000000 uncachable
[    0.000420]   7 base 4000000000 mask 4000000000 uncachable
[    0.000421]   8 disabled
[    0.000422]   9 disabled
[    0.001256] x86/PAT: Configuration [0-7]: WB  WC  UC- UC  WB  WP  UC- WT
[    0.001570] last_pfn = 0x6fc4f max_arch_pfn = 0x400000000
[    0.021713] esrt: Reserving ESRT space from 0x0000000068146518 to 0x0000000068146550.
[    0.021729] e820: update [mem 0x68146000-0x68146fff] usable ==> reserved
[    0.021847] check: Scanning 1 areas for low memory corruption
[    0.021853] Using GB pages for direct mapping
[    0.022564] RAMDISK: [mem 0x3ce54000-0x3fffdfff]
[    0.022580] ACPI: Early table checksum verification disabled
[    0.022584] ACPI: RSDP 0x000000006F117014 000024 (v02 INTEL )
[    0.022589] ACPI: XSDT 0x000000006F116728 0000CC (v01 INTEL  NUC9i5FN 00000020 AMI  01000013)
[    0.022598] ACPI: FACP 0x000000006F0D2000 000114 (v06 INTEL  NUC9i5FN 00000020 AMI  00010013)
[    0.022606] ACPI: DSDT 0x000000006F08F000 042561 (v02 INTEL  NUC9i5FN 00000020 INTL 20160527)
[    0.022610] ACPI: FACS 0x000000006F1B1000 000040
[    0.022614] ACPI: MCFG 0x000000006F0D5000 00003C (v01 INTEL  NUC9i5FN 00000020 MSFT 00000097)
[    0.022618] ACPI: SSDT 0x000000006F0D3000 001B4A (v02 INTEL  NUC9i5FN 00000020 INTL 20160527)
[    0.022623] ACPI: FIDT 0x000000006F08E000 00009C (v01 INTEL  NUC9i5FN 00000020 AMI  00010013)
[    0.022627] ACPI: SSDT 0x000000006F08A000 0031C6 (v02 INTEL  NUC9i5FN 00000020 INTL 20160527)
[    0.022631] ACPI: HPET 0x000000006F0D7000 000038 (v01 INTEL  NUC9i5FN 00000020 AMI  01000013)
[    0.022635] ACPI: SSDT 0x000000006F086000 0033B4 (v02 INTEL  NUC9i5FN 00000020 INTL 20160527)
[    0.022639] ACPI: SSDT 0x000000006F084000 00147B (v02 INTEL  NUC9i5FN 00000020 INTL 20160527)
[    0.022643] ACPI: SSDT 0x000000006F080000 0032BD (v02 INTEL  NUC9i5FN 00000020 INTL 20160527)
[    0.022648] ACPI: NHLT 0x000000006F0D6000 00002D (v00 INTEL  NUC9i5FN 00000020 AMI  01000013)
[    0.022652] ACPI: LPIT 0x000000006F07F000 000094 (v01 INTEL  NUC9i5FN 00000020 AMI  01000013)
[    0.022656] ACPI: SSDT 0x000000006F07B000 002720 (v02 INTEL  NUC9i5FN 00000020 INTL 20160527)
[    0.022660] ACPI: SSDT 0x000000006F07A000 00087C (v02 INTEL  NUC9i5FN 00000020 INTL 20160527)
[    0.022664] ACPI: DBGP 0x000000006F079000 000034 (v01 INTEL  NUC9i5FN 00000020 AMI  01000013)
[    0.022668] ACPI: DBG2 0x000000006F078000 000054 (v00 INTEL  NUC9i5FN 00000020 AMI  01000013)
[    0.022672] ACPI: SSDT 0x000000006F076000 001B66 (v02 INTEL  NUC9i5FN 00000020 INTL 20160527)
[    0.022677] ACPI: TPM2 0x000000006F074000 00004C (v04 INTEL  NUC9i5FN 00000020 AMI  00000000)
[    0.022681] ACPI: DMAR 0x000000006F075000 0000A8 (v01 INTEL  NUC9i5FN 00000020      01000013)
[    0.022685] ACPI: WSMT 0x000000006F07E000 000028 (v01 INTEL  NUC9i5FN 00000020 AMI  00010013)
[    0.022689] ACPI: APIC 0x000000006F073000 0000F4 (v04 INTEL  NUC9i5FN 00000020 AMI  00010013)
[    0.022693] ACPI: FPDT 0x000000006F072000 000044 (v01 INTEL  NUC9i5FN 00000020 AMI  01000013)
[    0.022707] ACPI: Local APIC address 0xfee00000
[    0.023236] No NUMA configuration found
[    0.023238] Faking a node at [mem 0x0000000000000000-0x0000000880ffffff]
[    0.023254] NODE_DATA(0) allocated [mem 0x880fd5000-0x880ffffff]
[    0.023698] Zone ranges:
[    0.023700]   DMA      [mem 0x0000000000001000-0x0000000000ffffff]
[    0.023702]   DMA32    [mem 0x0000000001000000-0x00000000ffffffff]
[    0.023703]   Normal   [mem 0x0000000100000000-0x0000000880ffffff]
[    0.023705]   Device   empty
[    0.023706] Movable zone start for each node
[    0.023711] Early memory node ranges
[    0.023713]   node   0: [mem 0x0000000000001000-0x000000000009efff]
[    0.023714]   node   0: [mem 0x0000000000100000-0x000000006cf53fff]
[    0.023716]   node   0: [mem 0x000000006fc4e000-0x000000006fc4efff]
[    0.023717]   node   0: [mem 0x0000000100000000-0x0000000880ffffff]
[    0.024270] Zeroed struct page in unavailable ranges: 41229 pages
[    0.024272] Initmem setup node 0 [mem 0x0000000000001000-0x0000000880ffffff]
[    0.024274] On node 0 totalpages: 8314611
[    0.024276]   DMA zone: 64 pages used for memmap
[    0.024277]   DMA zone: 25 pages reserved
[    0.024278]   DMA zone: 3998 pages, LIFO batch:0
[    0.024379]   DMA32 zone: 6910 pages used for memmap
[    0.024380]   DMA32 zone: 442197 pages, LIFO batch:63
[    0.039990]   Normal zone: 122944 pages used for memmap
[    0.039991]   Normal zone: 7868416 pages, LIFO batch:63
```

User space logging



https://devconnected.com/linux-logging-complete-guide/

```
abhi@abhi-VirtualBox:~$ ps ax | grep log
    596 ?        Ss     0:01 /usr/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation --syslog-only
    625 ?        Ssl    0:00 /usr/sbin/rsyslogd -n -iNONE
    630 ?        Ss     0:00 /lib/systemd/systemd-logind
   1379 ?        S<sl   0:00 /usr/bin/pulseaudio --daemonize=no --log-target=journal
   1382 ?        Sl     0:00 /usr/bin/gnome-keyring-daemon --daemonize --login
   1387 ?        Ss     0:00 /usr/bin/dbus-daemon --session --address=systemd: --nofork --nopidfile --systemd-activation --syslog-only
   4558 pts/1    S+     0:00 grep --color=auto log
abhi@abhi-VirtualBox:~$
```

systemd-journal

```
abhi@abhi-VirtualBox:~$ ls -al /run/
acpid.pid           gdm3/               plymouth/           thermald/
acpid.socket        gdm3.pid            sendsigs.omit.d/    tmpfiles.d/
alsa/               initctl             shm/                udev/
avahi-daemon/       initramfs/          snapd-snap.socket   udisks2/
blkid/              lock/               snapd.socket        ufw.lock
console-setup/      log/                speech-dispatcher/  user/
crond.pid           mount/              spice-vdagentd/     utmp
crond.reboot        NetworkManager/     sshd/               uuidd/
cups/               openvpn/            sshd.pid            vboxadd-service.sh
dbus/               openvpn-client/     sudo/
fsck/               openvpn-server/     systemd/
abhi@abhi-VirtualBox:~$ ls -al /run/log/
total 0
drwxr-xr-x   3 root root             60 Nov 10 10:03 .
drwxr-xr-x  31 root root            880 Nov 10 10:20 ..
drwxr-sr-x+  2 root systemd-journal  40 Nov 10 10:03 journal
abhi@abhi-VirtualBox:~$ ls -al /run/log/journal/
total 0
drwxr-sr-x+ 2 root systemd-journal 40 Nov 10 10:03 .
drwxr-xr-x  3 root root            60 Nov 10 10:03 ..
abhi@abhi-VirtualBox:~$
```

journalctl

abhi ssh ◂ -bash

JOURNALCTL(1) journalctl JOURNALCTL(1)

NAME

journalctl - Query the systemd journal

SYNOPSIS

journalctl [OPTIONS ...] [MATCHES ...]

DESCRIPTION

journalctl may be used to query the contents of the **systemd**(1) journal as written by **systemd-journald.service**(8).

If called without parameters, it will show the full contents of the journal, starting with the oldest entry collected.

If one or more match arguments are passed, the output is filtered accordingly. A match is in the format "FIELD=VALUE", e.g. "_SYSTEMD_UNIT=httpd.service", referring to the components of a structured journal entry. See **systemd.journal-fields**(7) for a list of well-known fields. If multiple matches are specified matching different fields, the log entries are filtered by both, i.e. the resulting output will show only entries matching all the specified matches of this kind. If two matches apply to the same field, then they are automatically matched as alternatives, i.e. the resulting output will show entries matching any of the specified matches for the same field. Finally, the character "+" may appear as a separate word between other terms on the command line. This causes all matches before and after to be combined in a disjunction (i.e. logical OR).

File Access Control



All disk access is mediated by the OS

OS enforces access controls





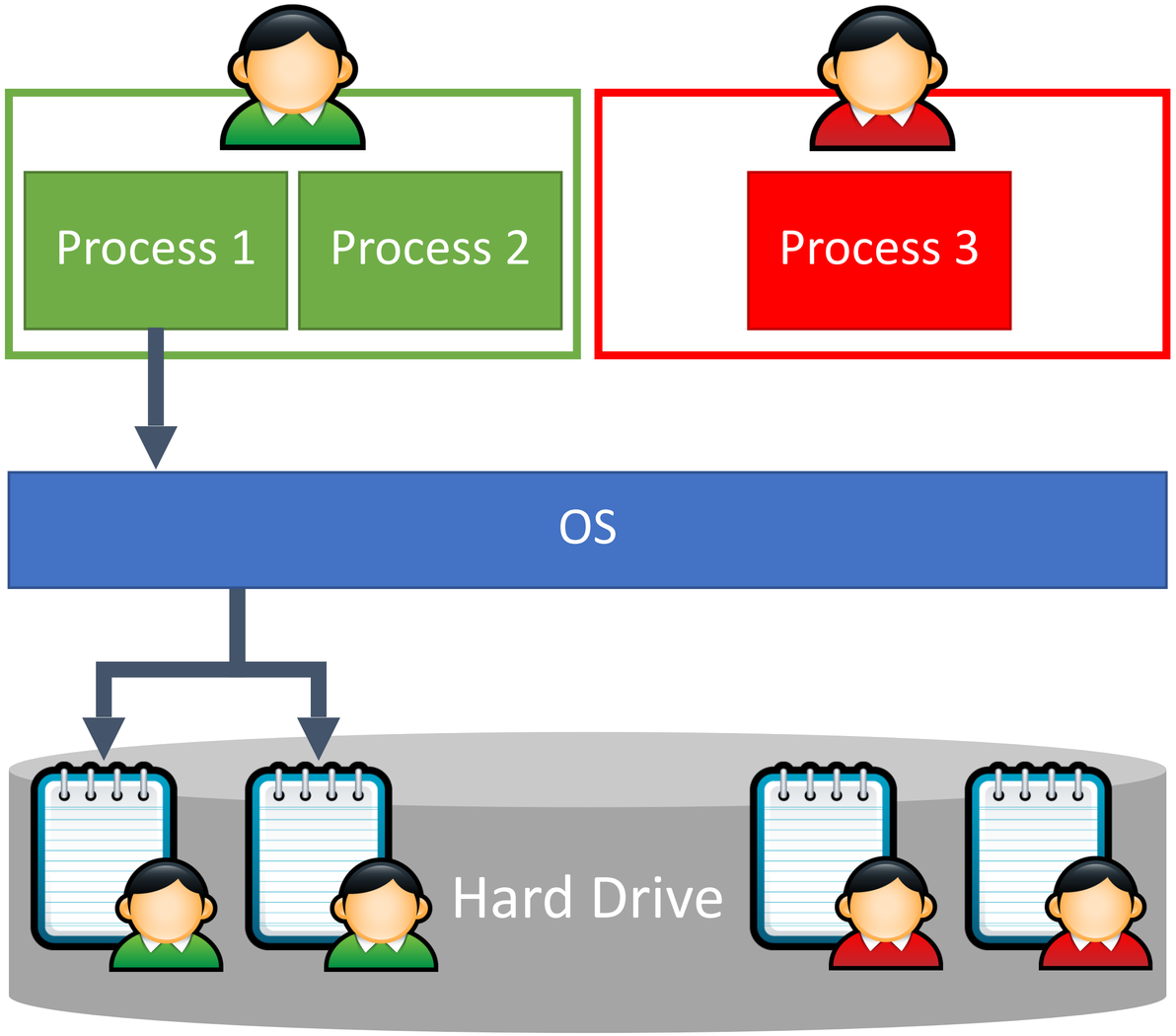


File Access Control



All disk access is mediated by the OS

OS enforces access controls

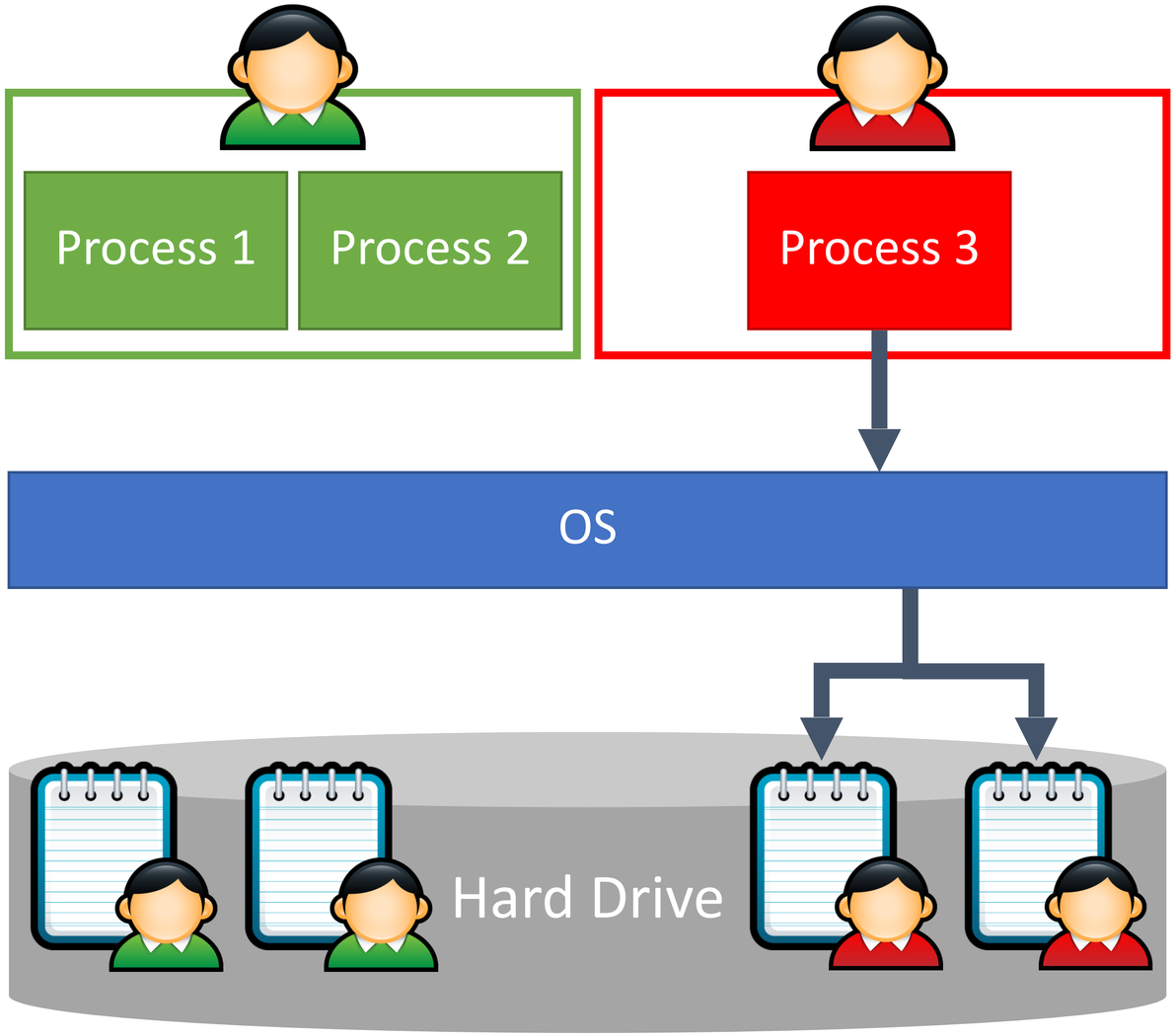


File Access Control



All disk access is mediated by the OS

OS enforces access controls

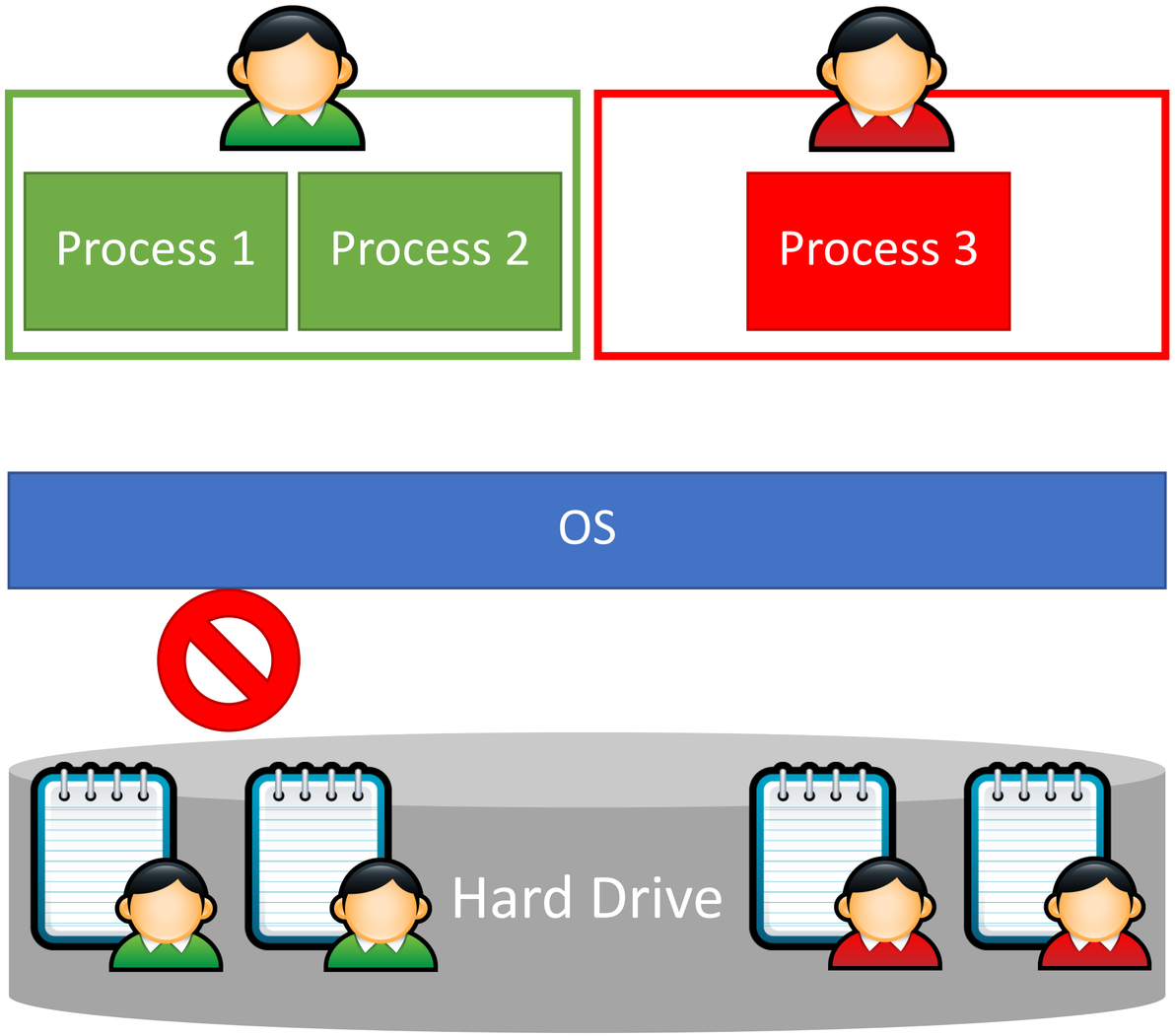


File Access Control



All disk access is mediated by the OS

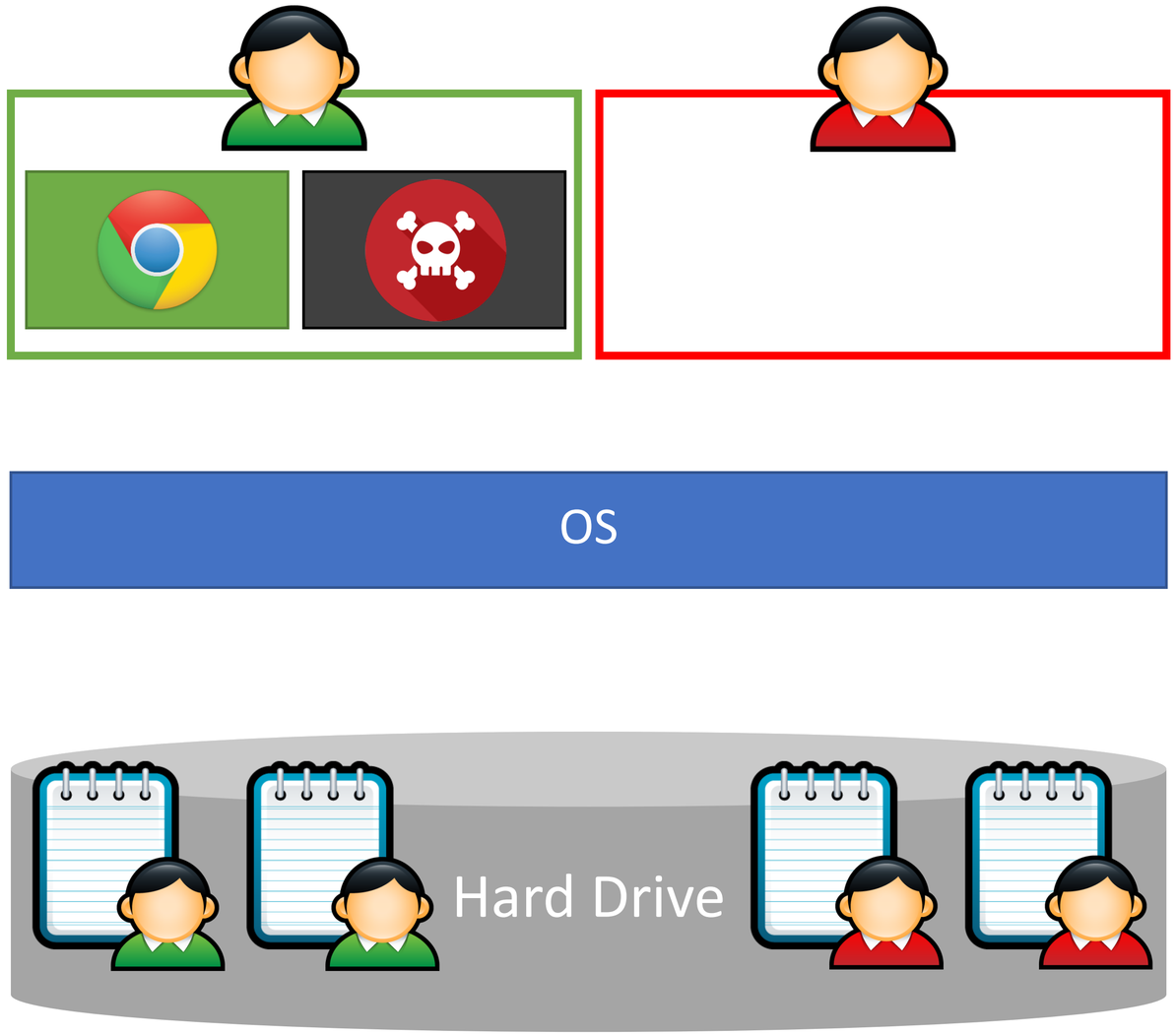
OS enforces access controls



Limitations

Malware can still cause damage

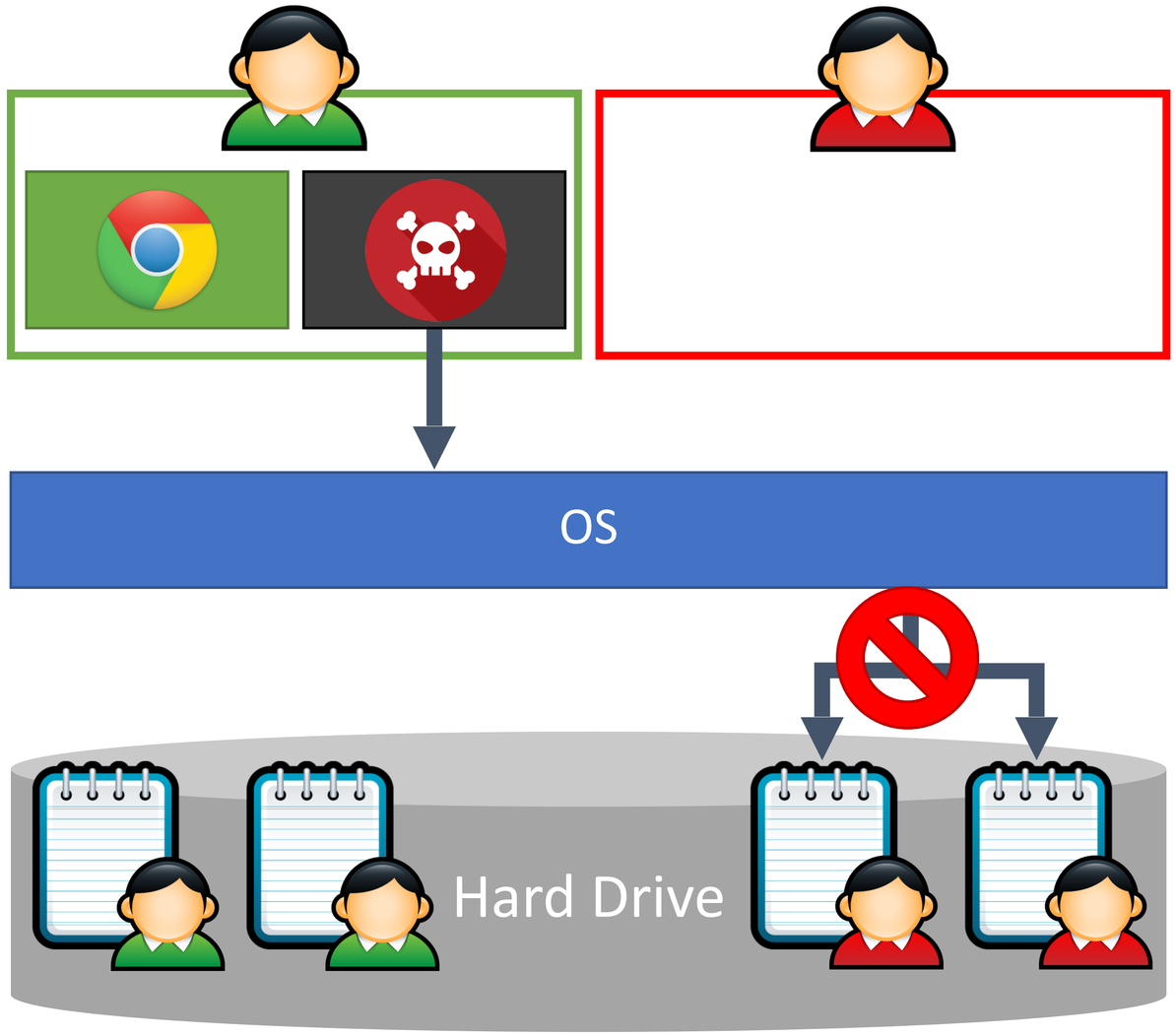
Discretionary access control means that isolation is incomplete



Limitations

Malware can still cause damage

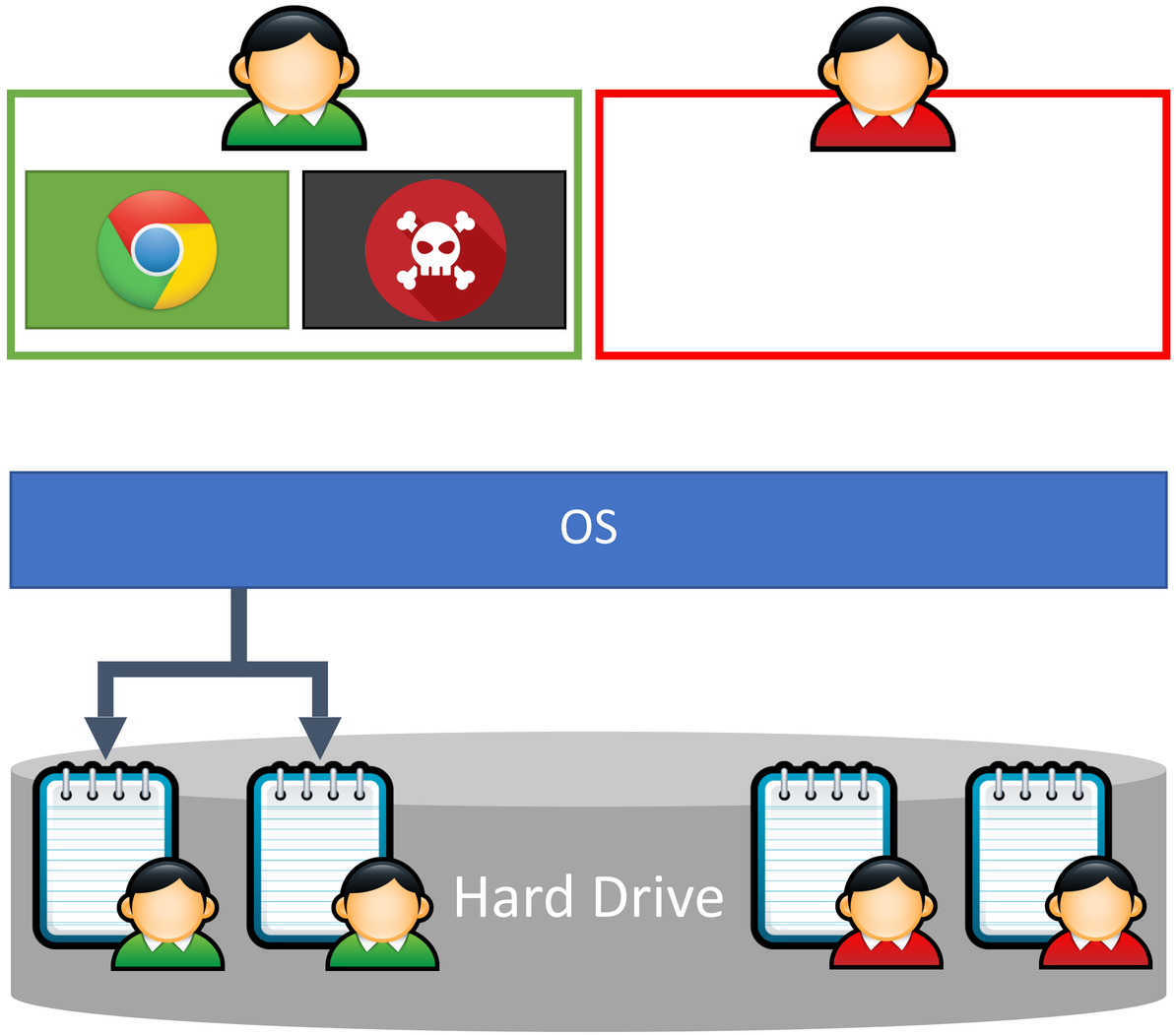
Discretionary access control means that isolation is incomplete



Limitations

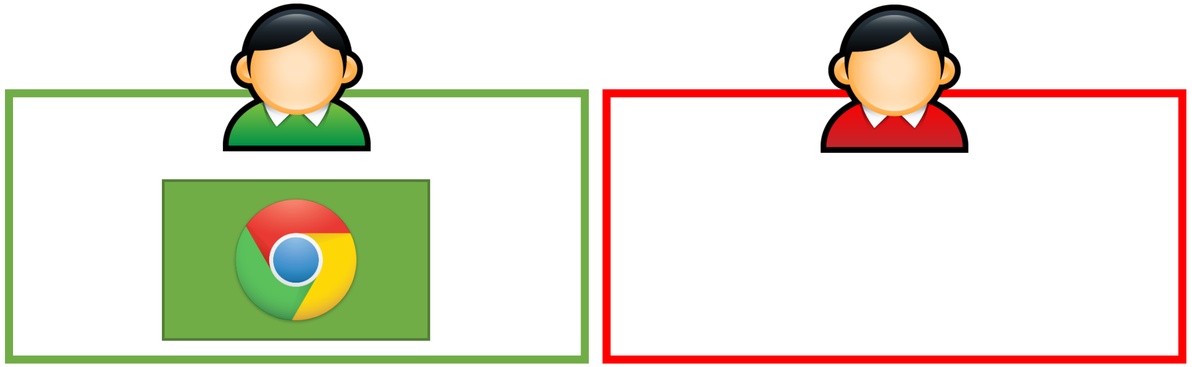
Malware can still cause damage

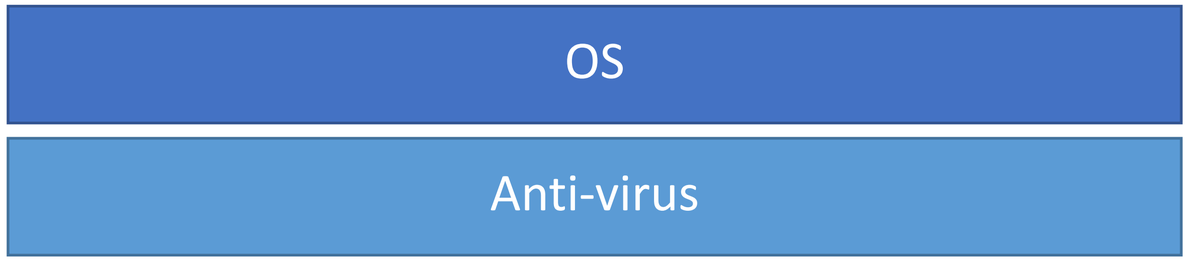
Discretionary access control means that isolation is incomplete



Anti-virus



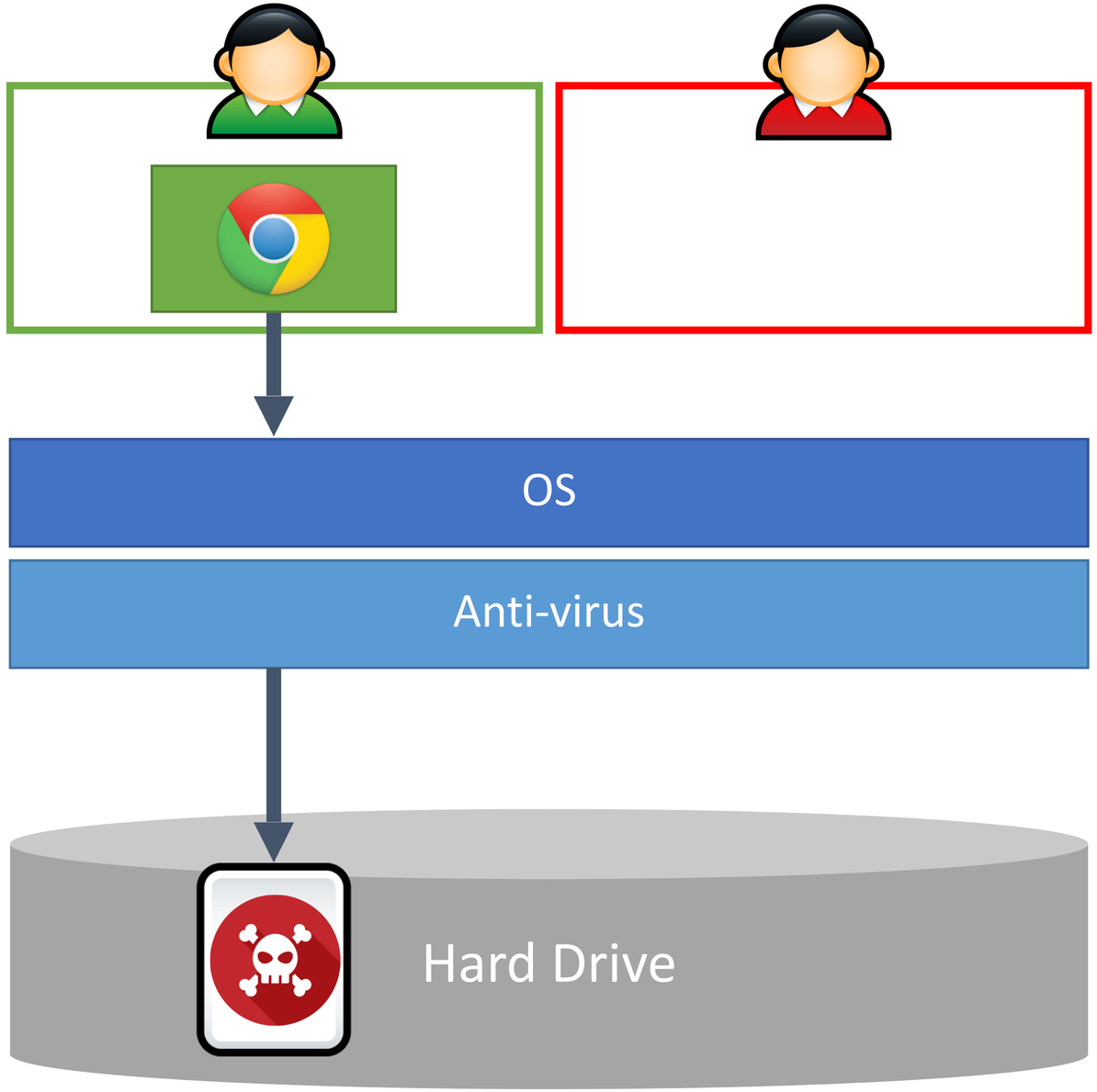






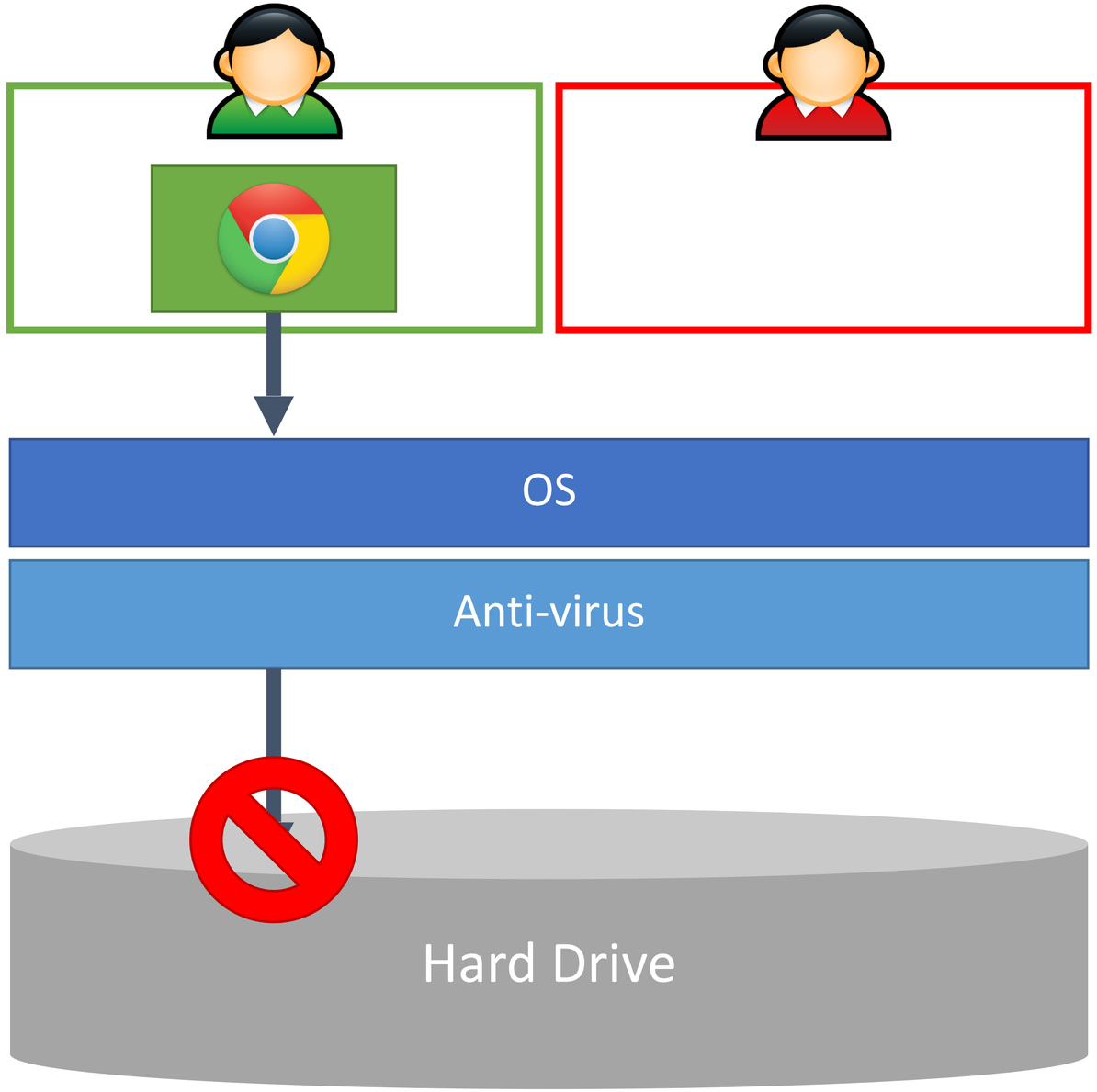
Anti-virus





Anti-virus





Anti-virus



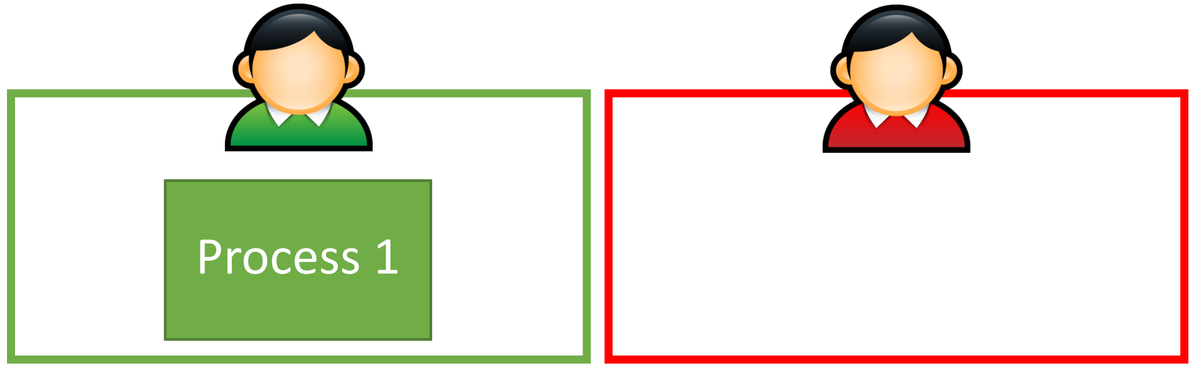
Anti-virus process is privileged

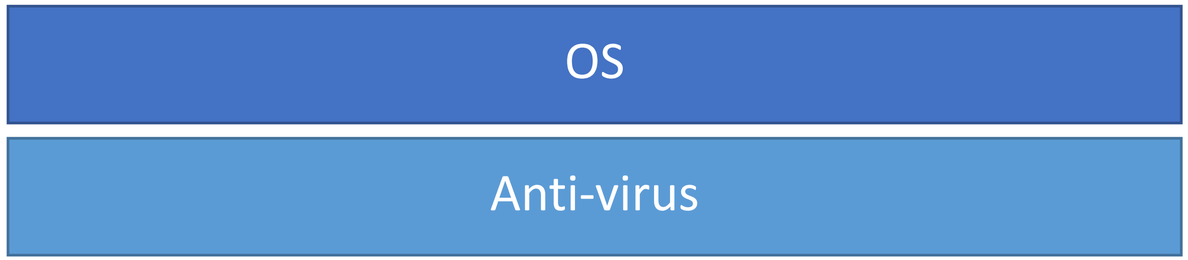
- Typically runs in Ring 0

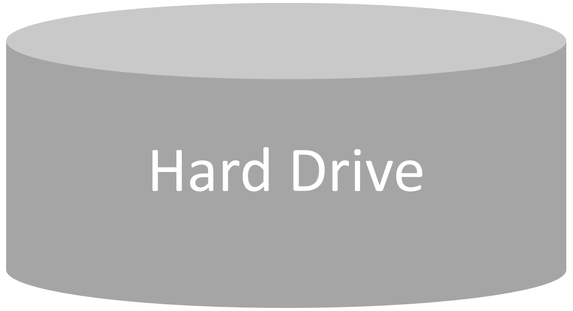
Scans all files looking for signatures

- Each signature uniquely identifies a piece of malware

Files scanned on creation and access







Anti-virus



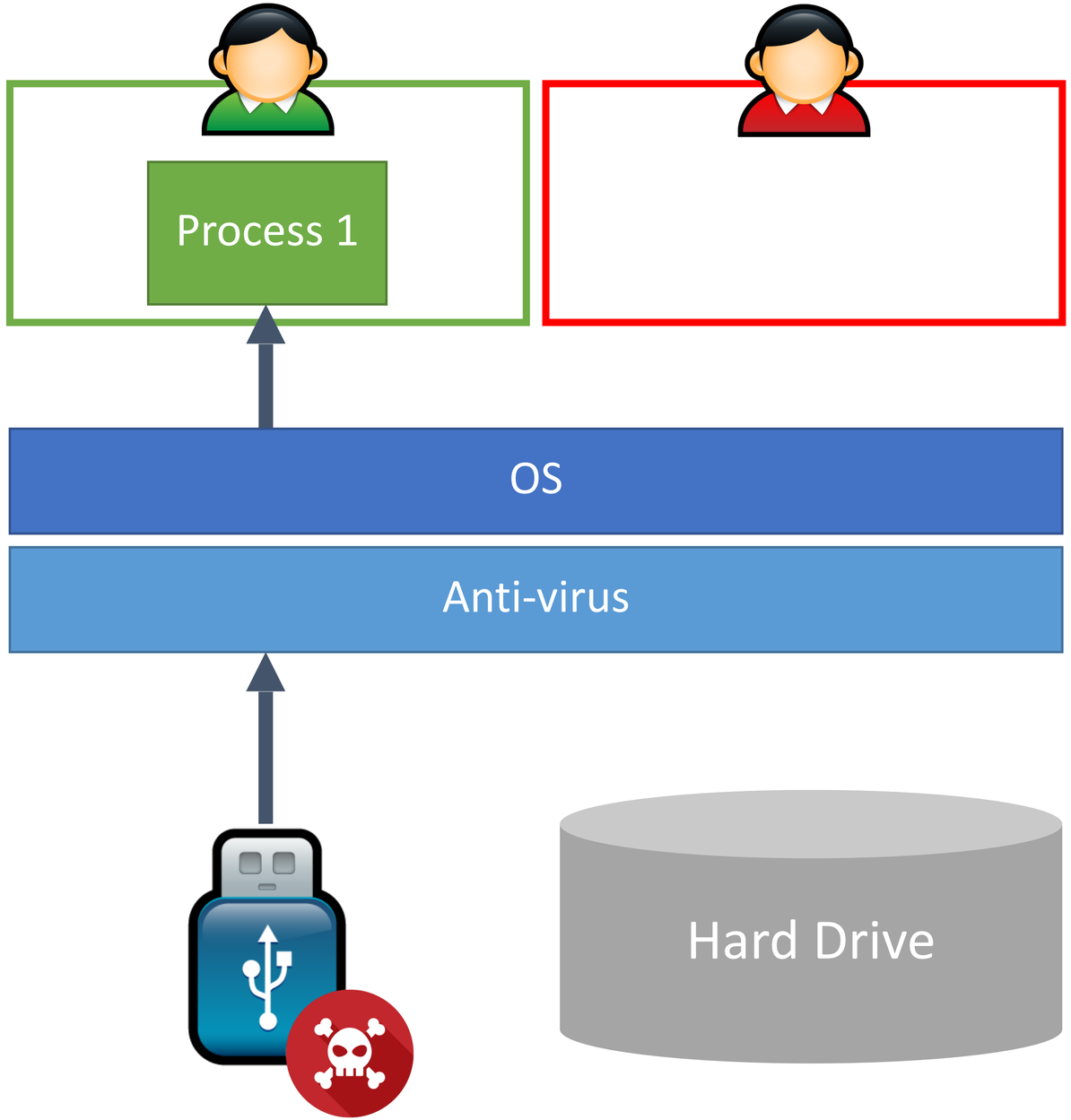
Anti-virus process is privileged

- Typically runs in Ring 0

Scans all files looking for signatures

- Each signature uniquely identifies a piece of malware

Files scanned on creation and access



Anti-virus



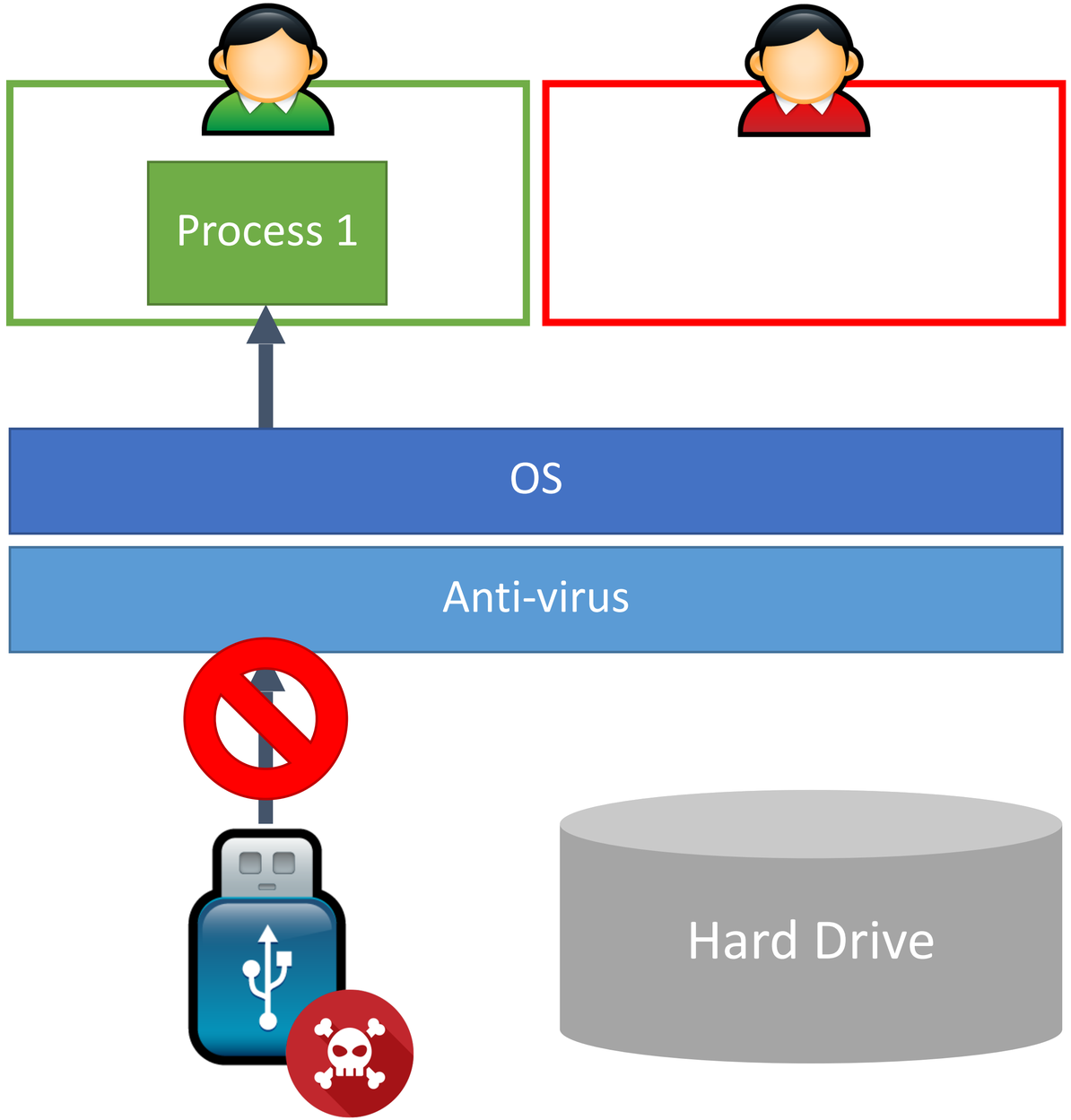
Anti-virus process is privileged

- Typically runs in Ring 0

Scans all files looking for signatures

- Each signature uniquely identifies a piece of malware

Files scanned on creation and access



Signature-based Detection

Key idea: identify invariants that correspond to malicious code or data

Example – anti-virus signatures

- List of code snippets that are unique to known malware

Problems with signatures

Signature-based Detection

Key idea: identify invariants that correspond to malicious code or data

Example – anti-virus signatures

- List of code snippets that are unique to known malware

Problems with signatures

- Must be updated frequently
- May cause false positives
  - Accidental overlaps with good programs and benign network traffic

Avast Malware Signature Update Breaks Installed Programs

Users of the free version of Avast antivirus unscathed

May 7, 2015 13:55 GMT · By Ionut Ilascu · Share:

A bad virus definition update from Avast released on Wednesday caused a lot of trouble, as it mistook various components in legitimate programs installed on the machine for malware.

The list of valid software affected by the signature update includes Firefox, iTunes, NVIDIA drivers, Google Chrome, Adobe Flash Player, Skype, Opera, TeamViewer, ATI drivers, as well as products from Corel and components of Microsoft Office.

Crowdstrike

What Happened

The CrowdStrike Falcon sensor delivers AI and machine learning to protect customer systems by identifying and remediating the latest advanced threats. In February 2024, CrowdStrike introduced a new sensor capability to enable visibility into possible novel attack techniques that may abuse certain Windows mechanisms. This capability pre-defined a set of fields for Rapid Response Content to gather data. As outlined in the RCA, this new sensor capability was developed and tested according to our standard software development processes.

On March 5, 2024, following a successful stress test, the first Rapid Response Content for Channel File 291 was released to production as part of a content configuration update, with three additional Rapid Response updates deployed between April 8, 2024 and April 24, 2024. These performed as expected in production.

On July 19, 2024, a Rapid Response Content update was delivered to certain Windows hosts, evolving the new capability first released in February 2024. The sensor expected 20 input fields, while the update provided 21 input fields. In this instance, the mismatch resulted in an out-of-bounds memory read, causing a system crash. Our analysis, together with a third-party review, confirmed this bug is not exploitable by a threat actor.

While this scenario with Channel File 291 is now incapable of recurring, it informs the process improvements and mitigation steps that CrowdStrike is deploying to ensure further enhanced resilience.

What happened?

The widespread Windows outage was due to a flaw in CrowdStrike's Falcon Sensor update that led to Microsoft Windows devices experiencing the Blue Screen of Death (BSOD), rendering systems unusable and causing widespread operational disruptions. This flaw rendered Microsoft Windows systems running CrowdStrike's software inoperable, with Windows systems failing to boot.

CrowdStrike pushed out the flawed update to Windows systems, causing outages in Australia, followed by Europe, grounding air travel and taking U.K. broadcaster SkyNews offline, Microsoft 365 outages, and taking down Windows systems running CrowdStrike software across the globe. Linux and MacOS systems were not impacted. The update from CrowdStrike was composed of "content", rather than software. The specifics of CrowdStirke's content update which caused Windows to crash are used by Falcon sensors executing on the host device.

CrowdStrike responded quickly by issuing an update fix and actively worked with customers around the globe to restore Windows systems. July 19, 2024, at 5:45 a.m. ET, CrowdStrike CEO George Kurtz posted a notice on social media platform X acknowledging the outage, indicating the root cause had been identified, and a fix update was available.

Remediation recommendations from CrowdStrike required rebooting affected Windows systems which would also receive the update. Virtualized, cloud-based servers and devices that can be remotely power cycled to apply the update. Windows devices still crashing required administrators to boot Windows into Safe Mode or the Windows Recovery Environment, navigate to a CrowdStrike directory, and manually delete the flawed update file. One aspect of why the CrowdStrike outage was so impactful was due to unmanaged and remote Windows devices, which required administrators' physical access to the devices. Further remediation details are available here.

However, amid the incident response and remediation efforts, it is crucial to recognize that the most significant flaw in this CrowdStrike outage was not merely the defective update but also the deployment process that allowed a service-impacting update to affect such a large set of customers

Avoiding Anti-virus

Malware authors go to great length to avoid detection by AV

Polymorphism

- Viral code mutates after every infection

Avoiding Anti-virus

Malware authors go to great length to avoid detection by AV

Polymorphism

- Viral code mutates after every infection

```
b = a + 10
```

Avoiding Anti-virus

Malware authors go to great length to avoid detection by AV

Polymorphism

- Viral code mutates after every infection

```
b = a + 10          b = a + 5 + 5
```

Avoiding Anti-virus

Malware authors go to great length to avoid detection by AV

Polymorphism

- Viral code mutates after every infection

```
b = a + 10        b = a + 5 + 5     b = (2 * a + 20) / 2
```

Avoiding Anti-virus

Malware authors go to great length to avoid detection by AV

Polymorphism

- Viral code mutates after every infection

```
b = a + 10        b = a + 5 + 5     b = (2 * a + 20) / 2
```

Packing

- Malware code is encrypted, key is changed every infection
- Decryption code is vulnerable to signature construction
- Polymorphism may be used to mutate the decryption code

Firewall

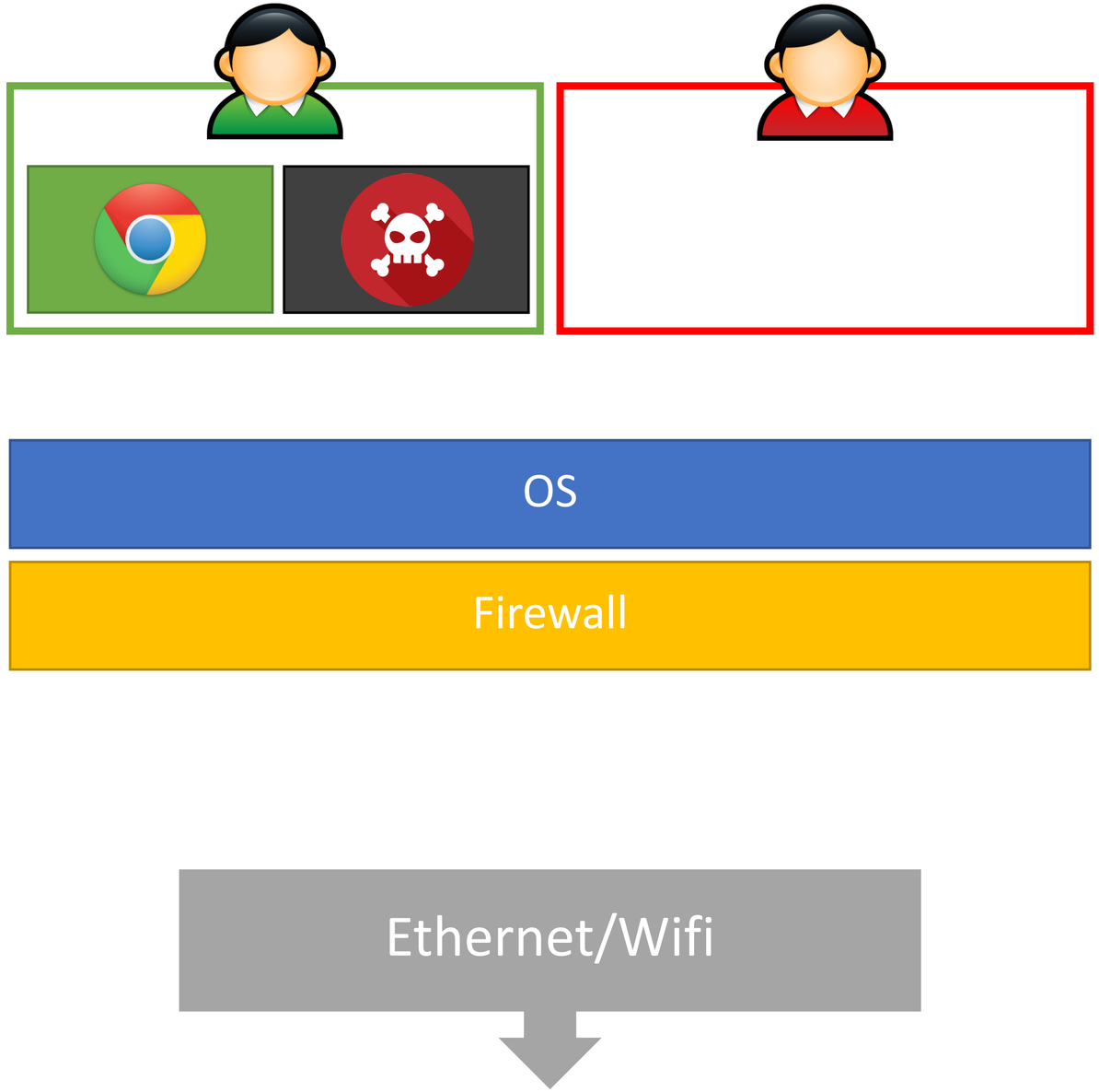
Firewall process is privileged

- Often runs in Ring 0

Selectively blocks network traffic

- By process
- By port
- By IP address
- By packet content

Inspects outgoing and incoming network traffic



Firewall

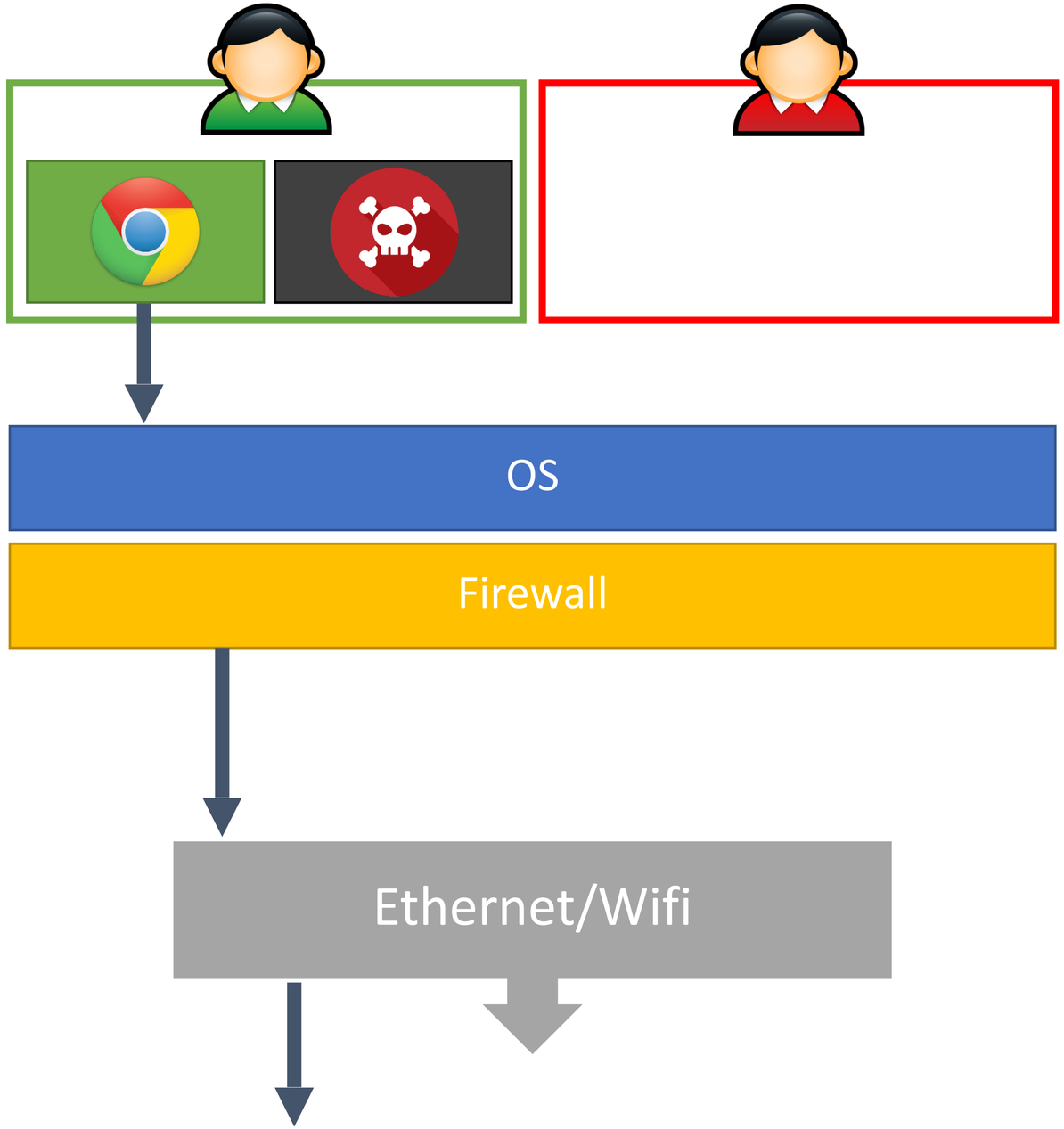
Firewall process is privileged

- Often runs in Ring 0

Selectively blocks network traffic

- By process
- By port
- By IP address
- By packet content

Inspects outgoing and incoming network traffic



Firewall

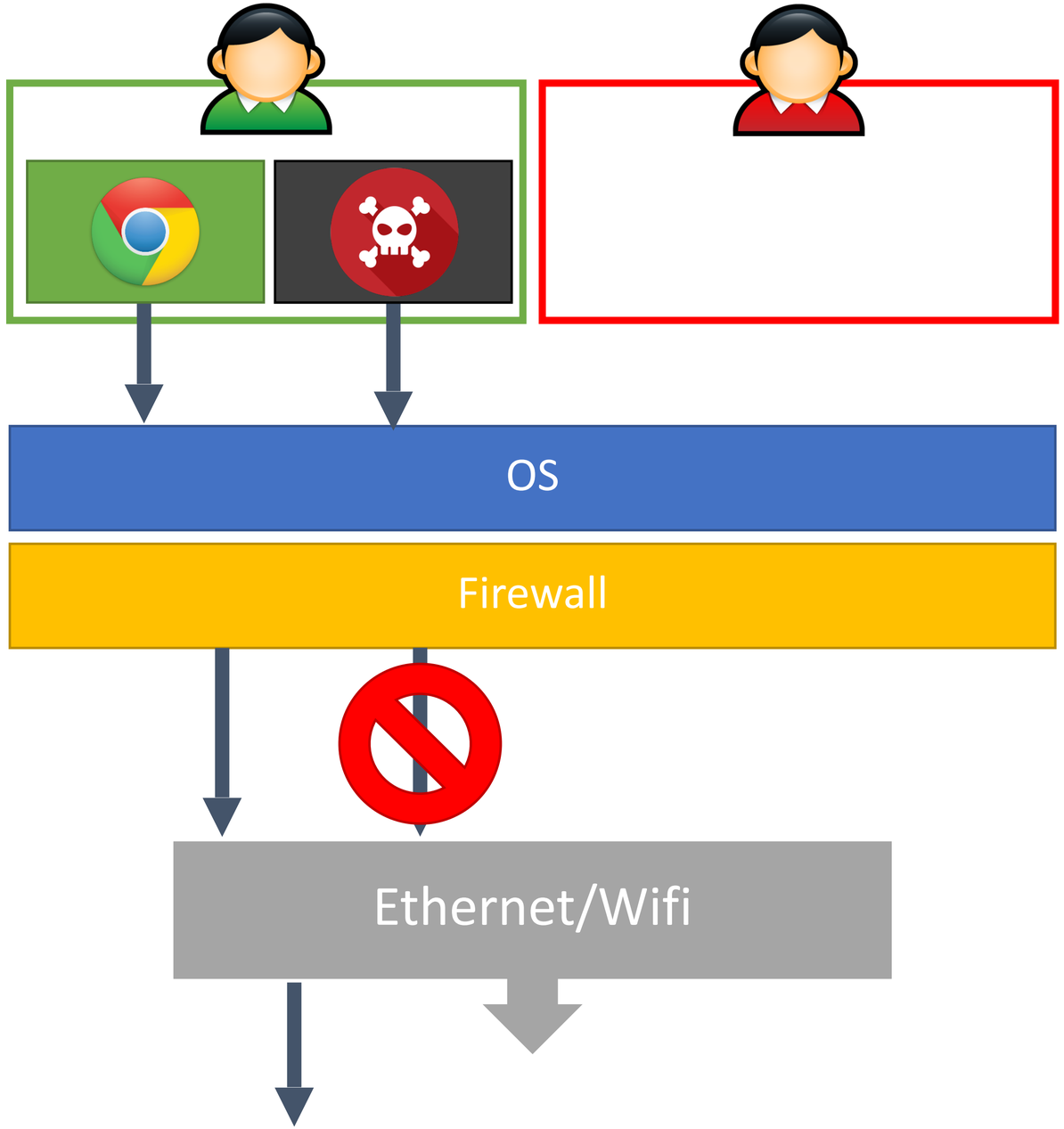
Firewall process is privileged

- Often runs in Ring 0

Selectively blocks network traffic

- By process
- By port
- By IP address
- By packet content

Inspects outgoing and incoming network traffic



Firewall

Firewall process is privileged

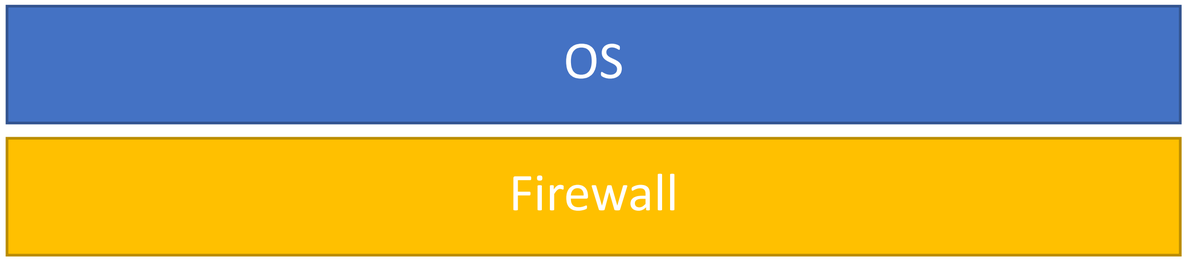
- Often runs in Ring 0

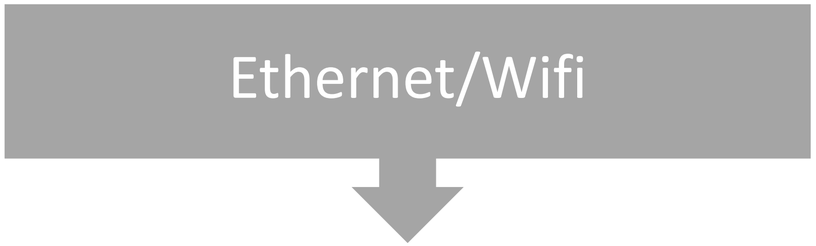
Selectively blocks network traffic

- By process
- By port
- By IP address
- By packet content

Inspects outgoing and incoming network traffic







Firewall

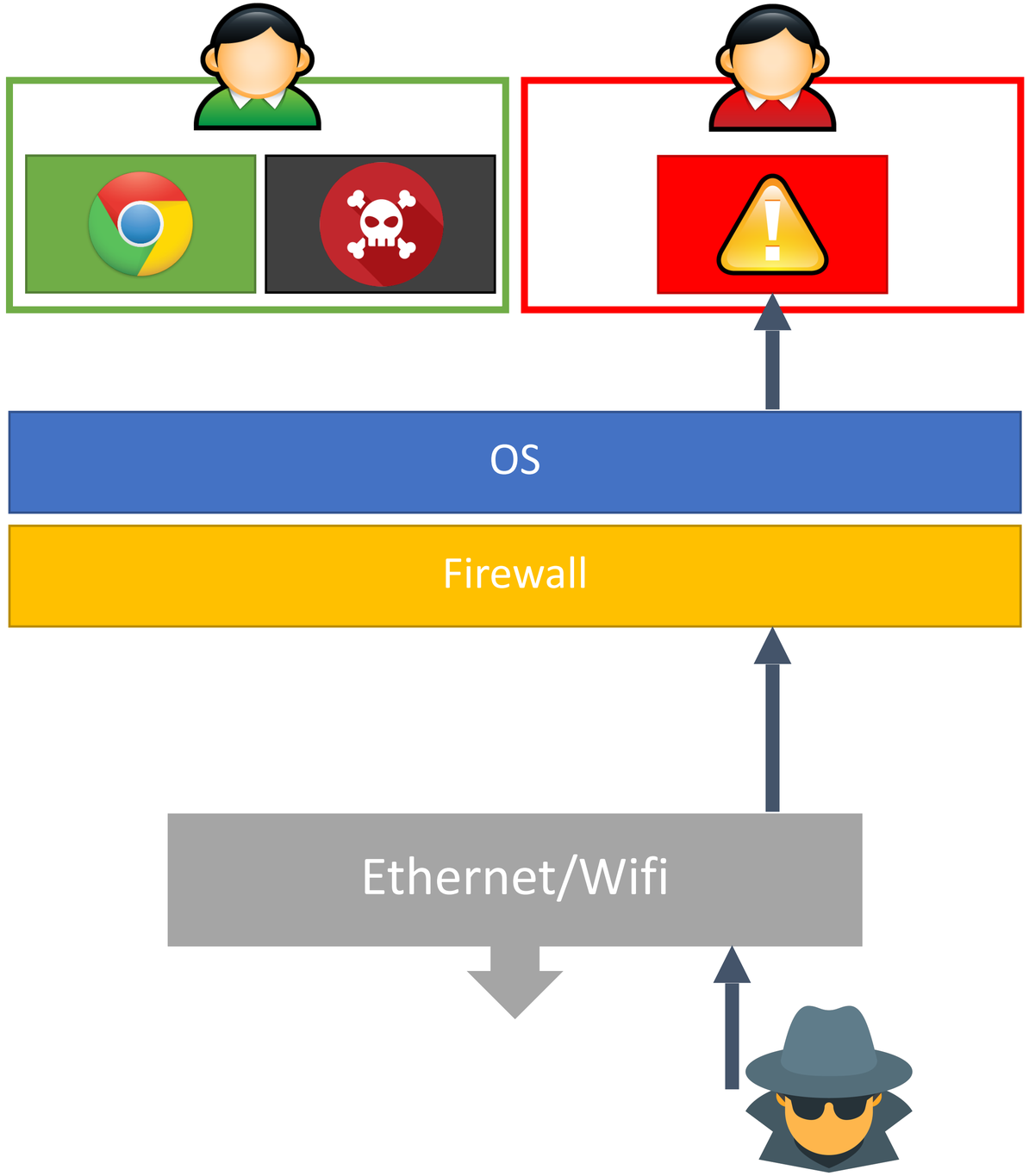
Firewall process is privileged

- Often runs in Ring 0

Selectively blocks network traffic

- By process
- By port
- By IP address
- By packet content

Inspects outgoing and incoming network traffic



Firewall

Firewall process is privileged

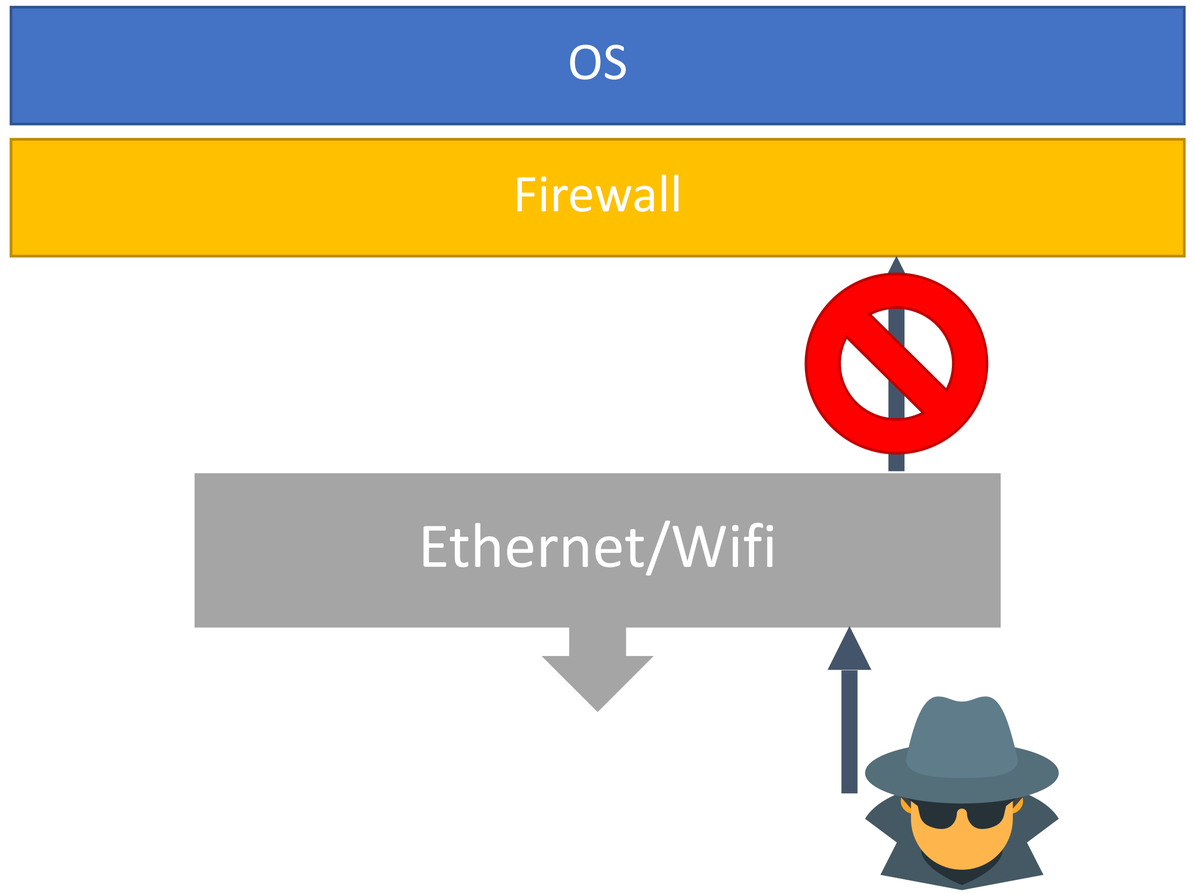
- Often runs in Ring 0

Selectively blocks network traffic

- By process
- By port
- By IP address
- By packet content

Inspects outgoing and incoming network traffic





Network Intrusion Detection Systems

NIDS for short

Snort

- Open source intrusion prevention system capable of real-time traffic analysis and packet logging
- Identifies malicious network traffic using signatures



Bro

- Open source network monitoring, analysis, and logging framework
- Can be used to implement signature based detection
- Capable of more complex analysis

