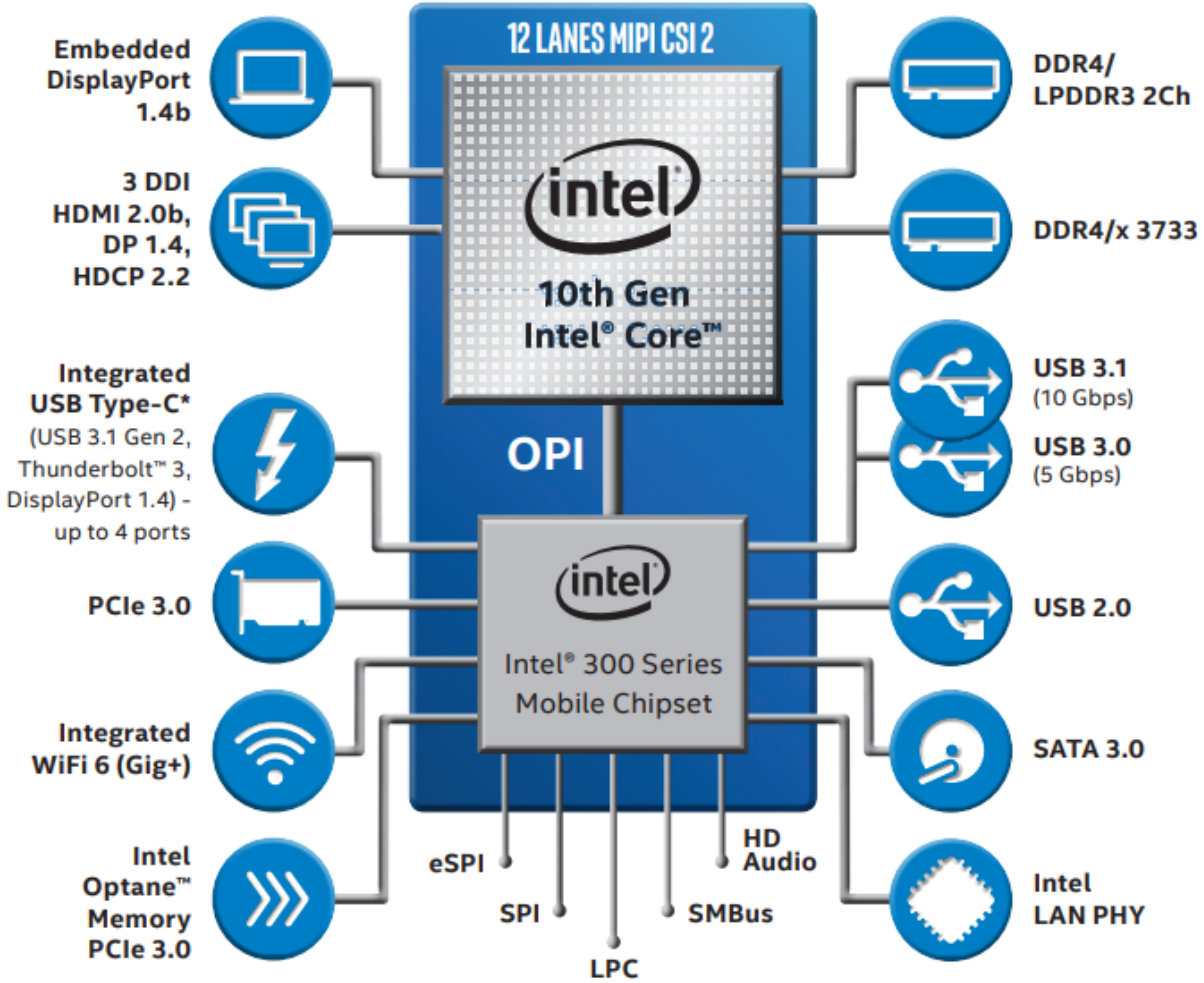
2550 Intro to cybersecurity

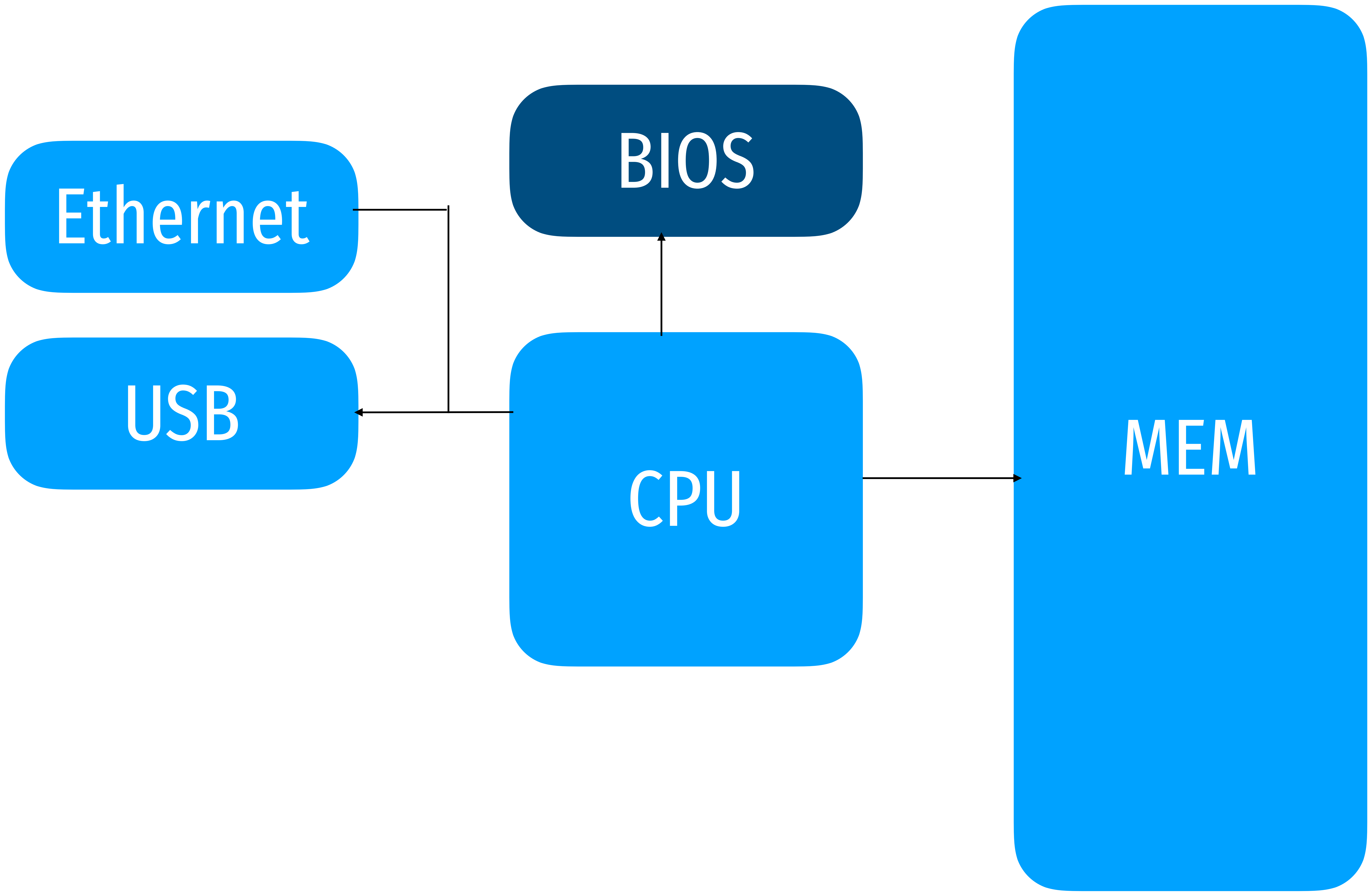
L20: systems

abhi shelat

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


IBM
IBM
IBM


12 LANES MIPI CSI 2
intel
10th Gen
Intel® Core™
OPI
intel
Intel® 300 Series
Mobile Chipset
Embedded DisplayPort 1.4b
3 DDI HDMI 2.0b, DP 1.4, HDCP 2.2
Integrated USB Type-C* (USB 3.1 Gen 2, Thunderbolt™ 3, DisplayPort 1.4) - up to 4 ports
PCIe 3.0
Integrated WiFi 6 (Gig+)
Intel Optane™ Memory PCIe 3.0
eSPI
SPI
LPC
SMBus
HD Audio
DDR4/ LPDDR3 2Ch
DDR4/x 3733
USB 3.1 (10 Gbps)
USB 3.0 (5 Gbps)
USB 2.0
SATA 3.0
Intel LAN PHY


BIOS
Ethernet
USB
CPU
MEM

What is Memory?

Memory is essentially a spreadsheet with a single column

- Every row has a number, called an address
- Every cell holds 1 byte of data

Address	Contents
114	
113	[illegible]
112	[illegible]
111	[illegible]
110	8
109	
108	[illegible]
107	[illegible]
106	[illegible]
105	[illegible]
104	
103	[illegible]
102	[illegible]
101	[illegible]
100	[illegible]

What is Memory?

Memory is essentially a spreadsheet with a single column

- Every row has a number, called an address
- Every cell holds 1 byte of data

Integers are typically four bytes

Address	Contents
114	
113	0
112	0
111	0
110	8
109	
108	[illegible]
107	[illegible]
106	[illegible]
105	[illegible]
104	
103	[illegible]
102	[illegible]
101	[illegible]
100	[illegible]

What is Memory?

Memory is essentially a spreadsheet with a single column

- Every row has a number, called an address
- Every cell holds 1 byte of data

Integers are typically four bytes

Each ASCII character is one byte, Strings are null terminated

Address	Contents
114	
113	0
112	0
111	0
110	8
109	
108	0
107	C
106	B
105	A
104	
103	[illegible]
102	[illegible]
101	[illegible]
100	[illegible]

What is Memory?

Memory is essentially a spreadsheet with a single column

- Every row has a number, called an address
- Every cell holds 1 byte of data

Integers are typically four bytes

Each ASCII character is one byte, Strings are null terminated

CPUs understand instructions in assembly language

Address	Contents
114	
113	0
112	0
111	0
110	8
109	
108	0
107	C
106	B
105	A
104	
103	0xAF
102	0x3C
101	0x91
100	0xE3

What is Memory?

Memory is essentially a spreadsheet with a single column

- Every row has a number, called an address
- Every cell holds 1 byte of data

All data and running code are held in memory

```
int my_num = 8;
```

Integers are typically four bytes

Each ASCII character is one byte, Strings are null terminated

CPUs understand instructions in assembly language

Address	Contents
114	
113	0
112	0
111	0
110	8
109	
108	0
107	C
106	B
105	A
104	
103	0xAF
102	0x3C
101	0x91
100	0xE3

What is Memory?

Memory is essentially a spreadsheet with a single column

- Every row has a number, called an address
- Every cell holds 1 byte of data

All data and running code are held in memory

```
int my_num = 8;

String my_str = "ABC";
```

Address	Contents
114	
113	0
112	0
111	0
110	8
109	
108	0
107	C
106	B
105	A
104	
103	0xAF
102	0x3C
101	0x91
100	0xE3

Integers are typically four bytes

Each ASCII character is one byte, Strings are null terminated

CPUs understand instructions in assembly language

What is Memory?

Memory is essentially a spreadsheet with a single column

- Every row has a number, called an address
- Every cell holds 1 byte of data

All data and running code are held in memory

```
int my_num = 8;

String my_str = "ABC";
while (my_num > 0) my_num--;
```

Address	Contents
114	
113	0
112	0
111	0
110	8
109	
108	0
107	C
106	B
105	A
104	
103	0xAF
102	0x3C
101	0x91
100	0xE3

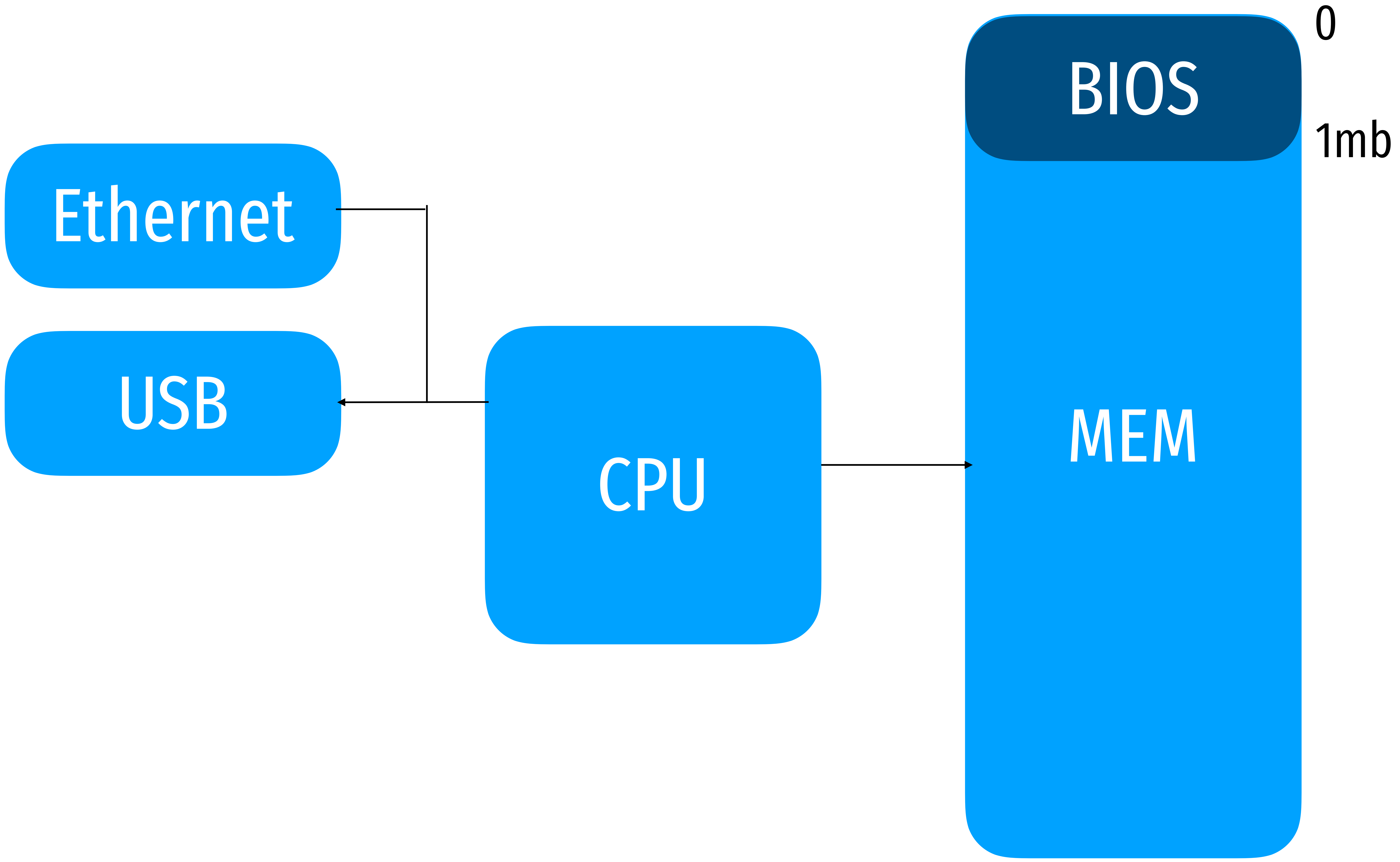
Integers are typically four bytes (addresses 110–113)

Each ASCII character is one byte, Strings are null terminated (addresses 105–108)

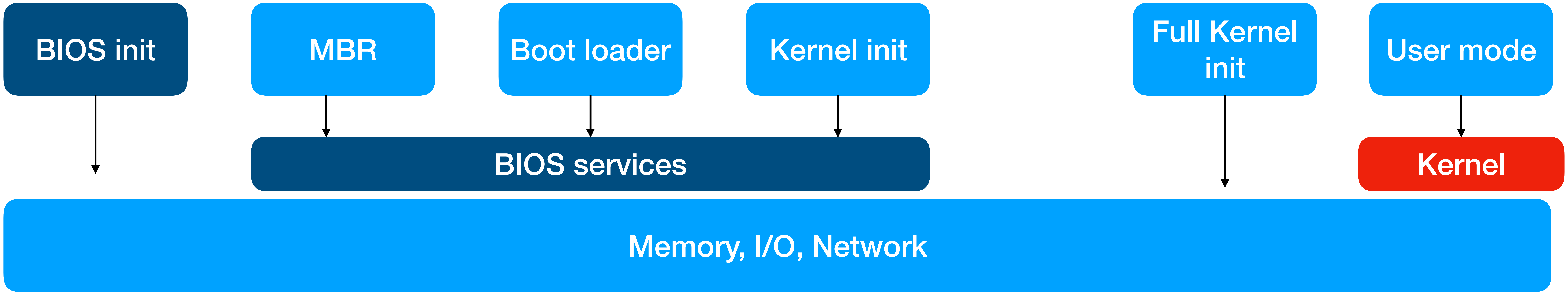
CPUs understand instructions in assembly language (addresses 100–103)

How does a computer boot?

https://youtu.be/MsKb0gR-4AM?t=36

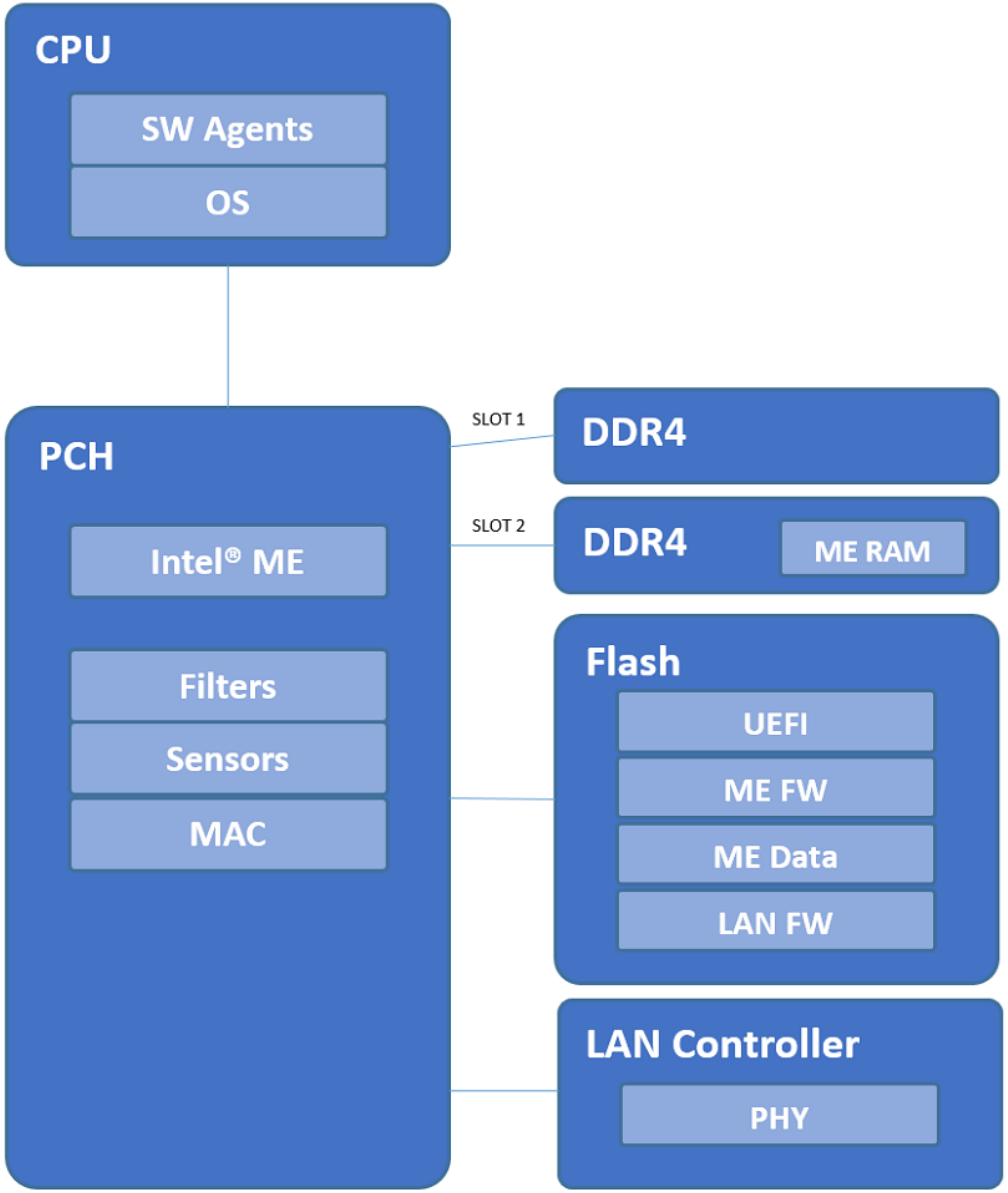

0
BIOS
1mb
Ethernet
USB
CPU
MEM

System Model: how does a computer boot?



https://www.intel.com/content/www/us/en/intelligent-systems/intel-boot-loader-development-kit/minimal-intel-architecture-boot-loader-paper.html

More details



Layout of memory at boot

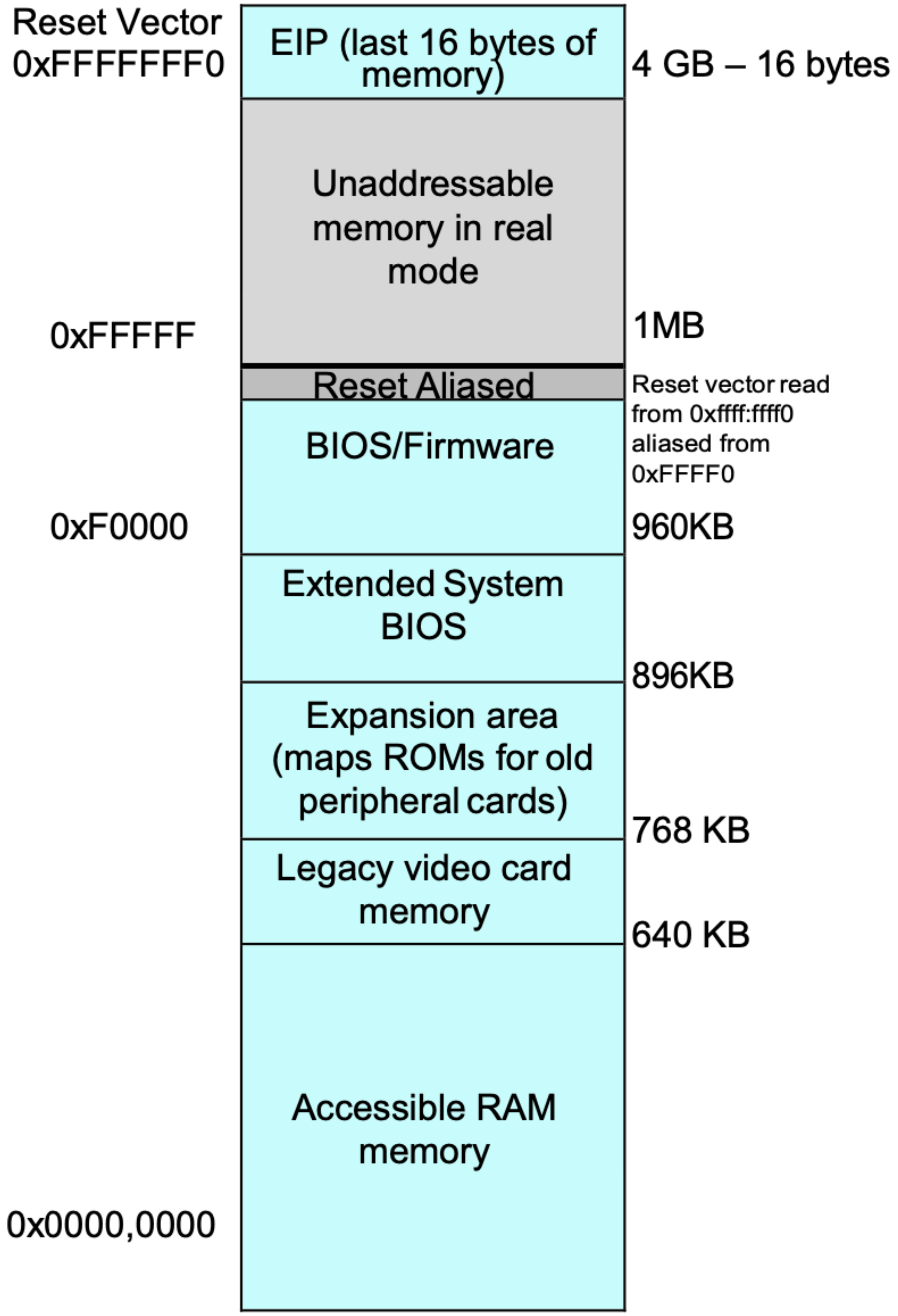
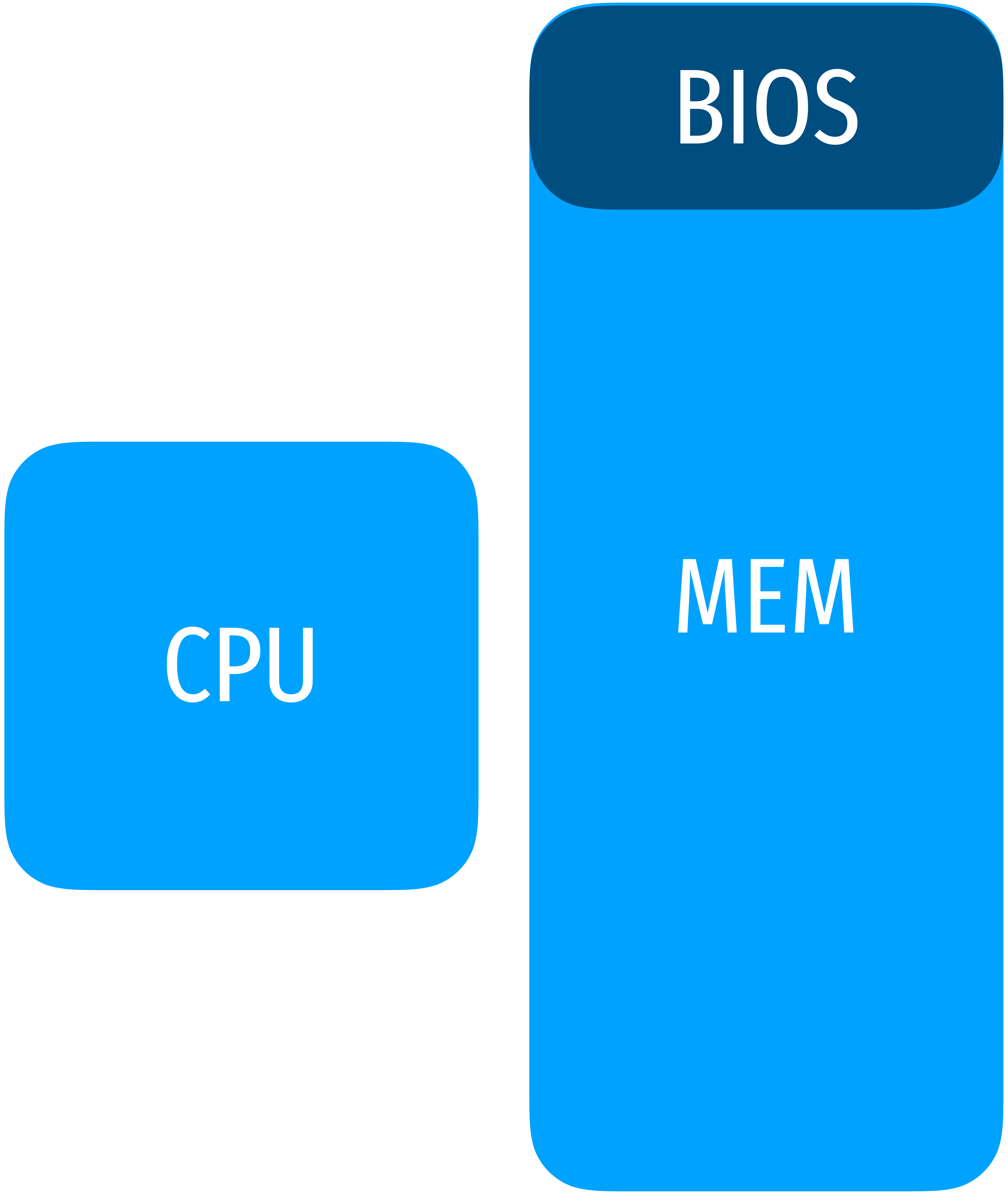


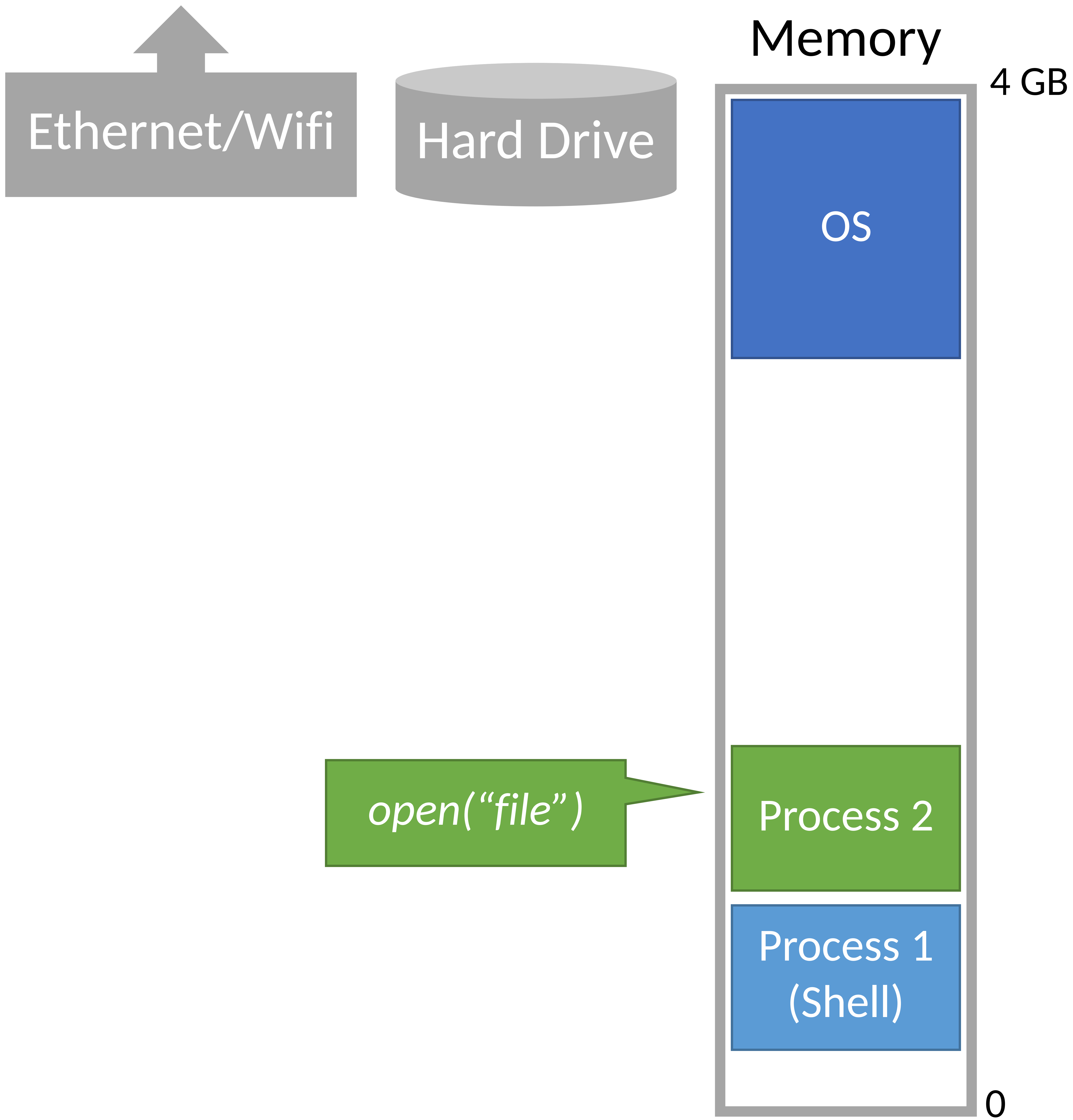
Figure 3 Intel® Architecture Memory Map at Power On

Details

CPU begins executing at f.fff0
BIOS firmware begins init of hw
Applies microcode patches
Execute Firmware Support Pkg (blob)
[Ram is setup]
Copy firmware to RAM
Begin executing in RAM
Setup interrupts, timers, clocks
Bring up other cores
Setup PCI
Setup ACPI tables
Execute OS loader



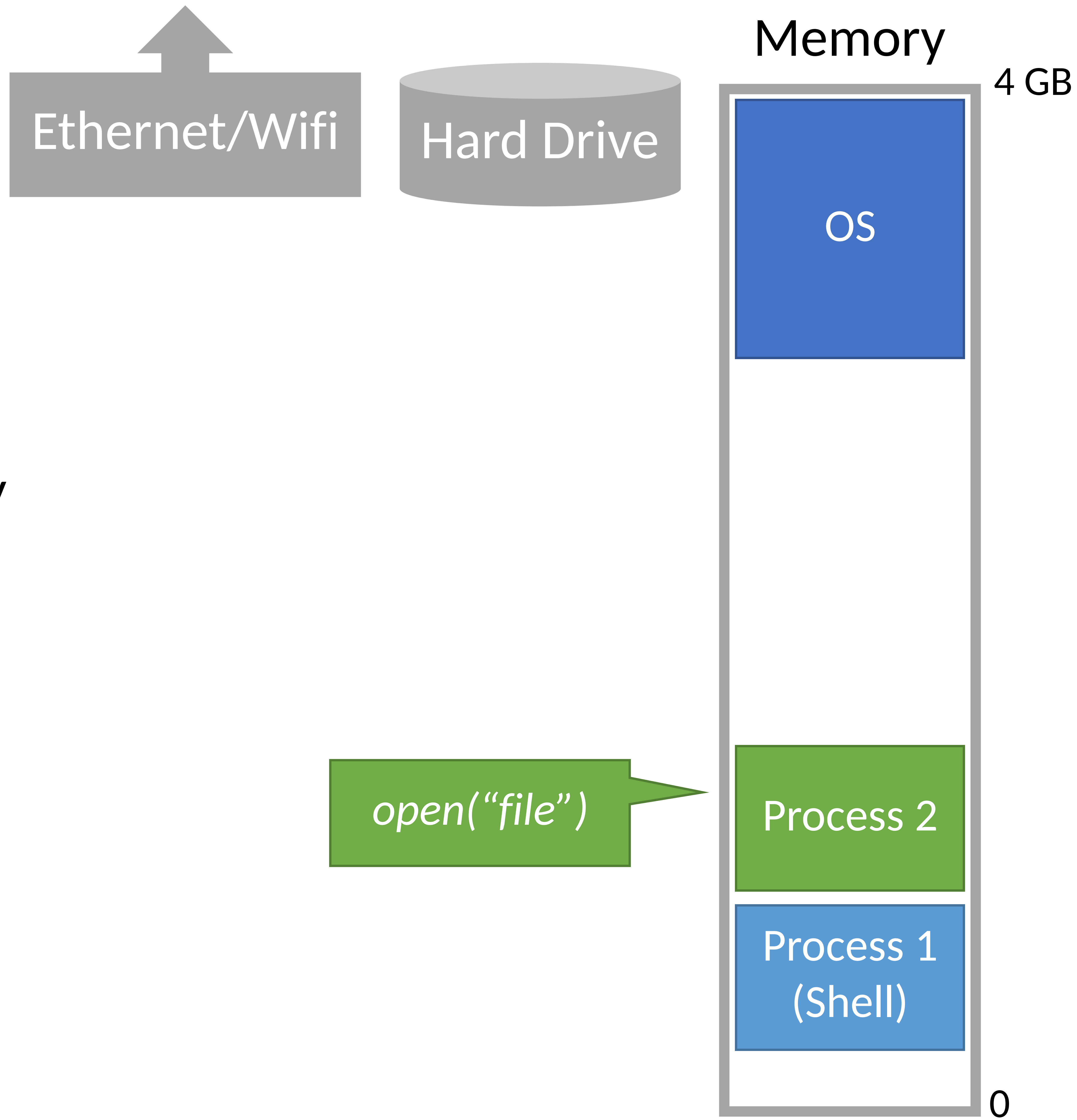
System Model



System Model

On bootup, the Operating System (OS) loads itself into memory

- eg. DOS (before hw isolation)
- Typically places itself in high memory



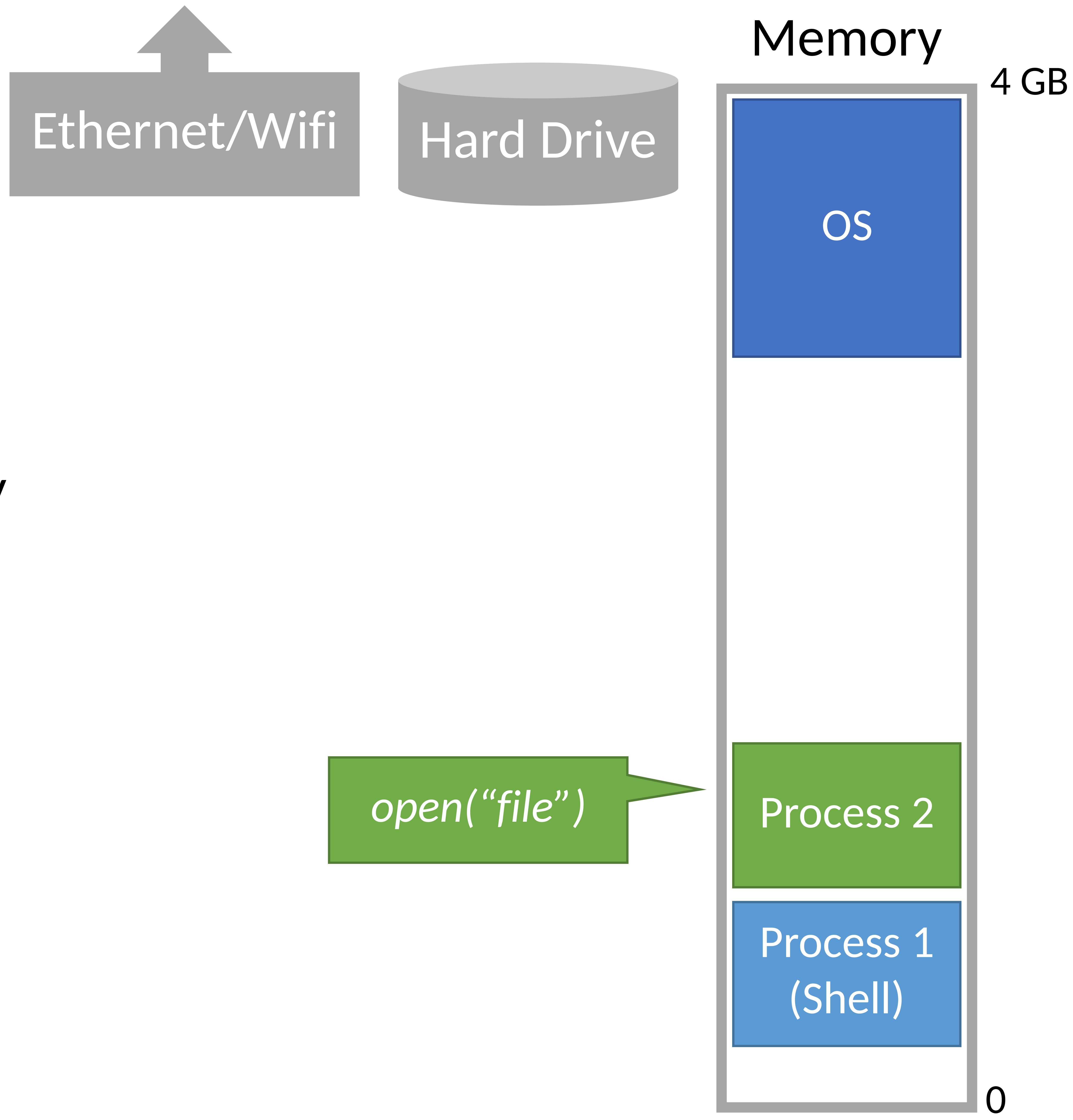
System Model

On bootup, the Operating System (OS) loads itself into memory

- eg. DOS (before hw isolation)
- Typically places itself in high memory

What is the role of the OS?

- Allow the user to run processes
- Often comes with a shell
  - Text shell like bash
  - Graphical shell like the Windows desktop
- Provides APIs to access devices
  - Offered as a convenience to application developers



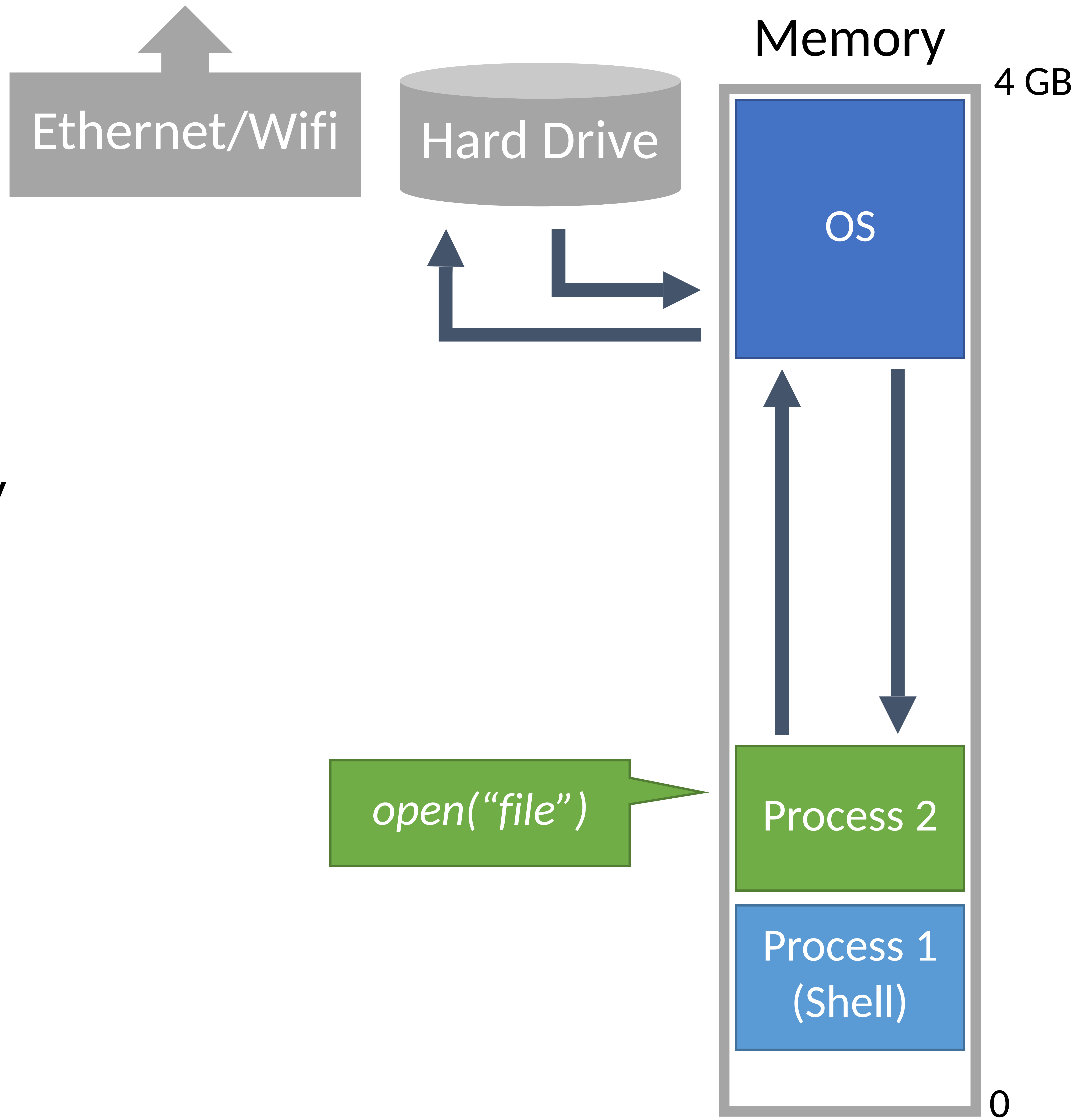
System Model

On bootup, the Operating System (OS) loads itself into memory

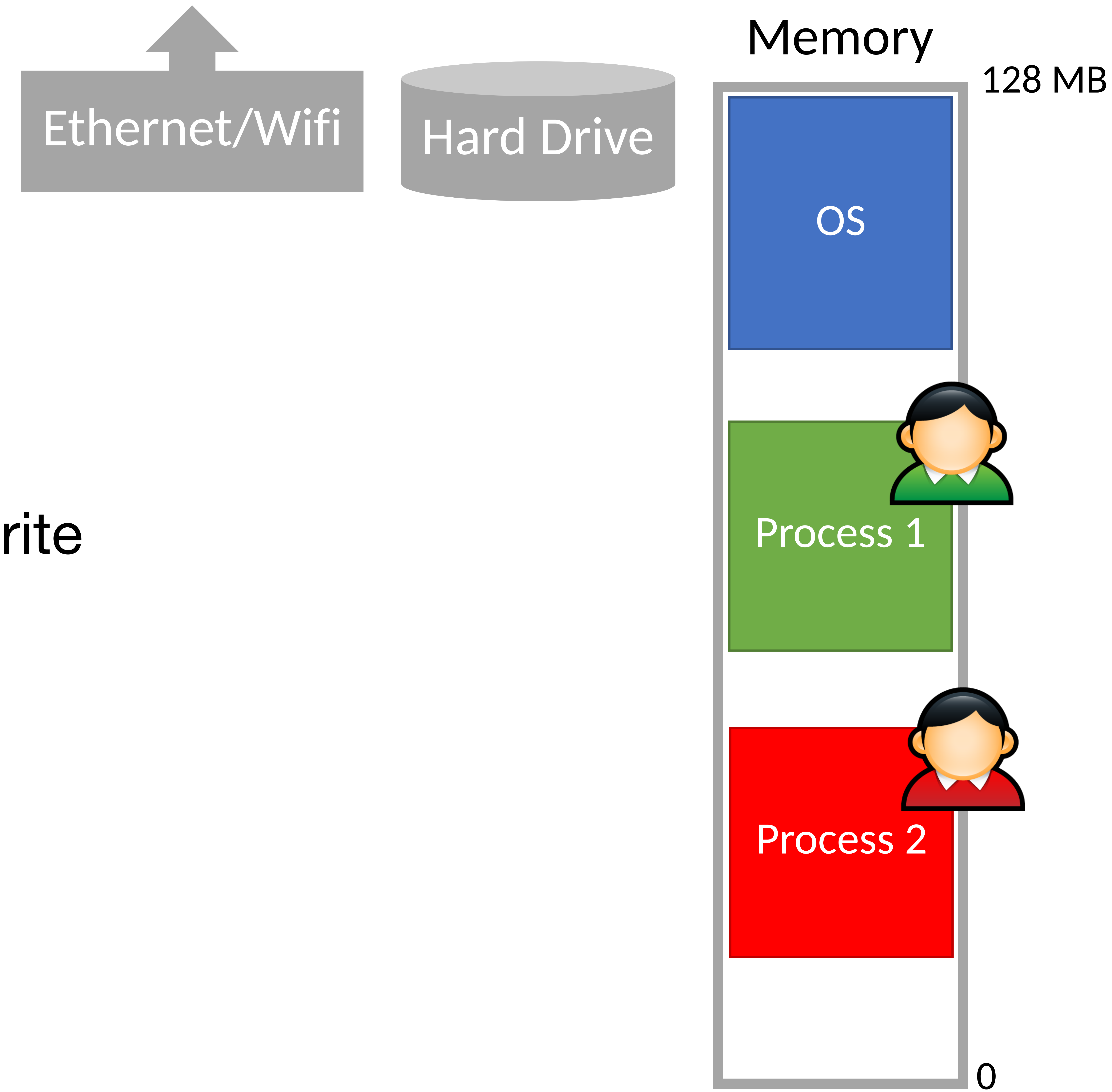
- eg. DOS (before hw isolation)
- Typically places itself in high memory

What is the role of the OS?

- Allow the user to run processes
- Often comes with a shell
  - Text shell like bash
  - Graphical shell like the Windows desktop
- Provides APIs to access devices
  - Offered as a convenience to application developers



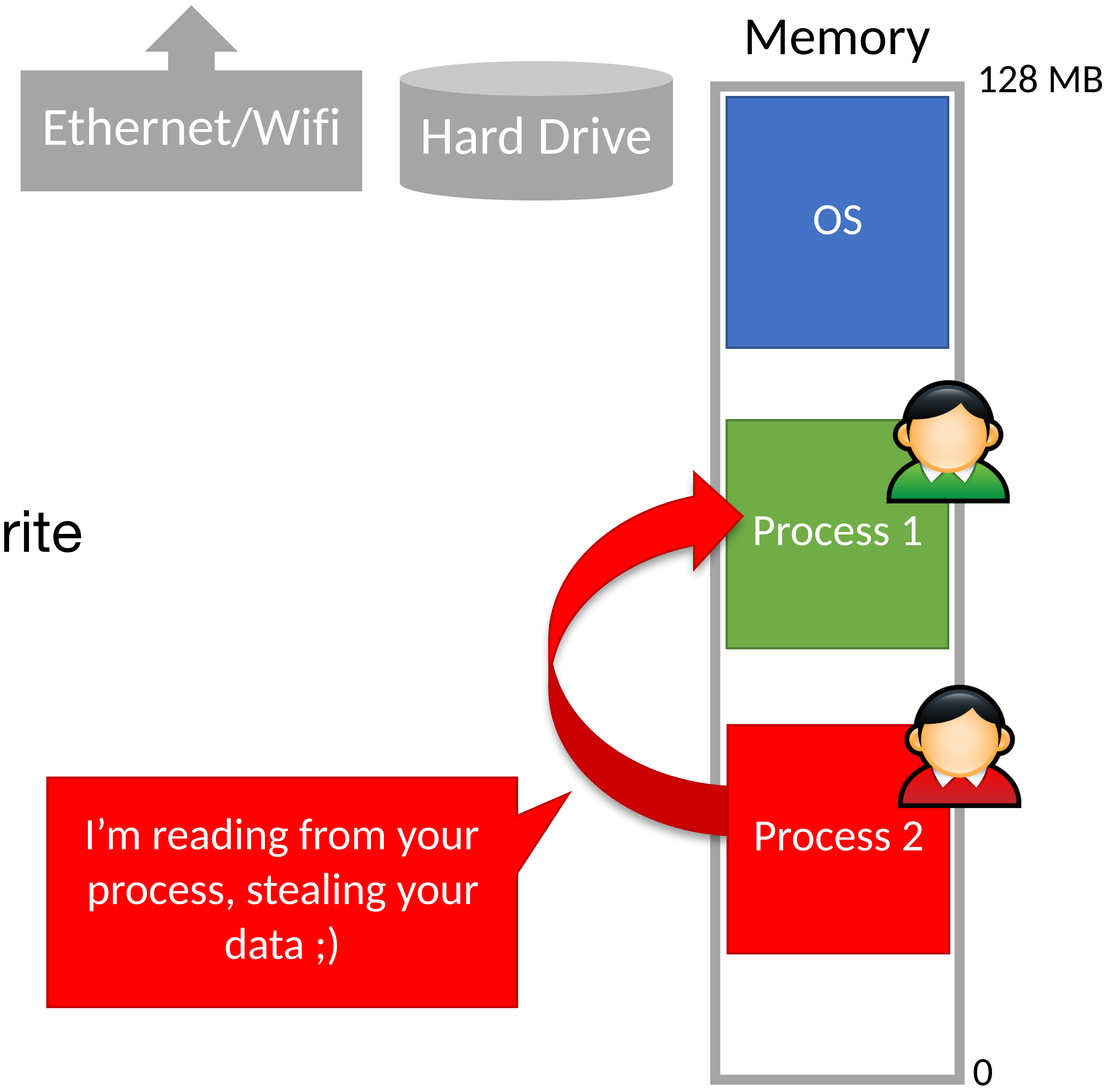
Memory Unsafety



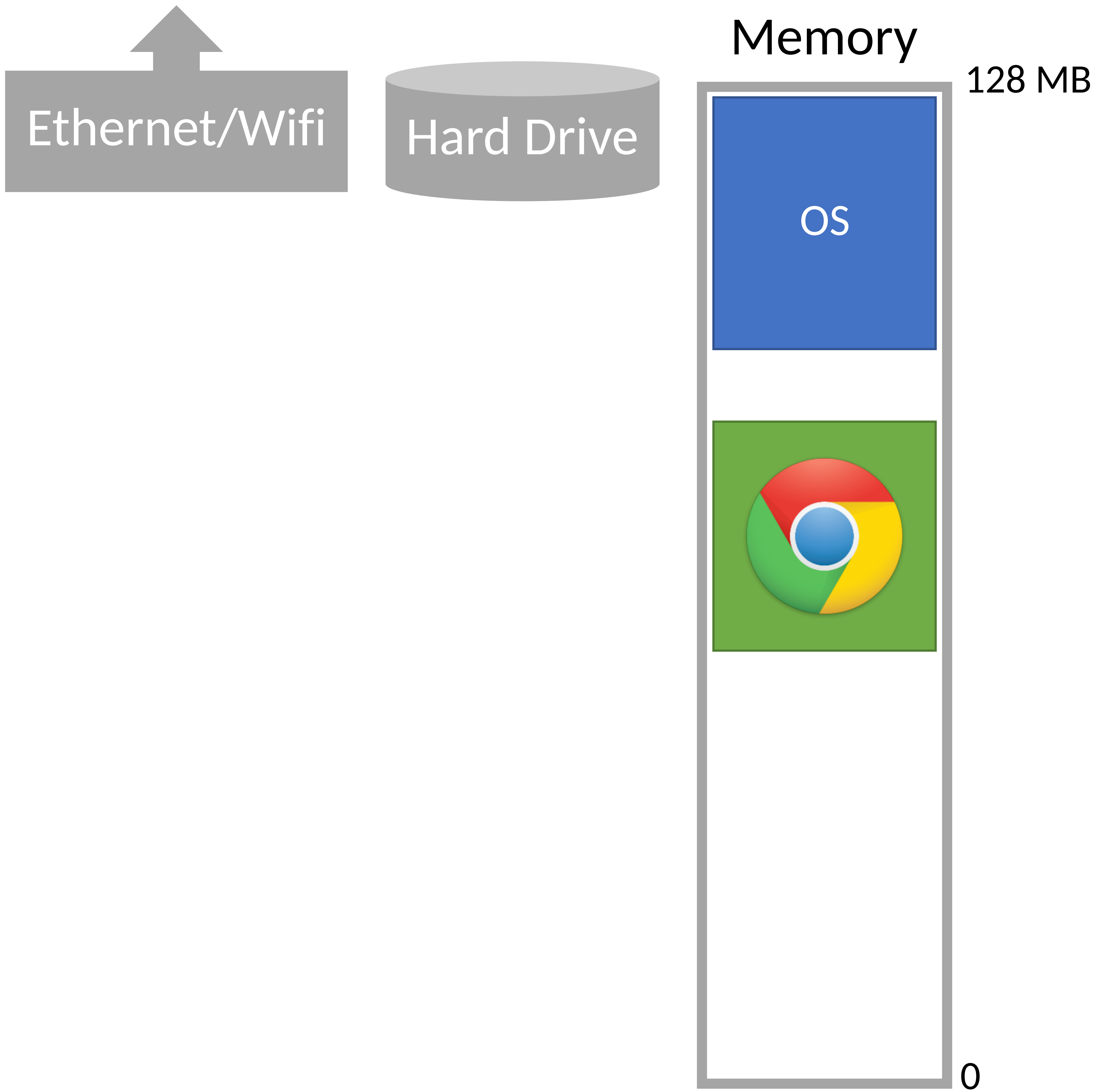
Problem: any process can read/write any memory

Memory Unsafety

Problem: any process can read/write any memory

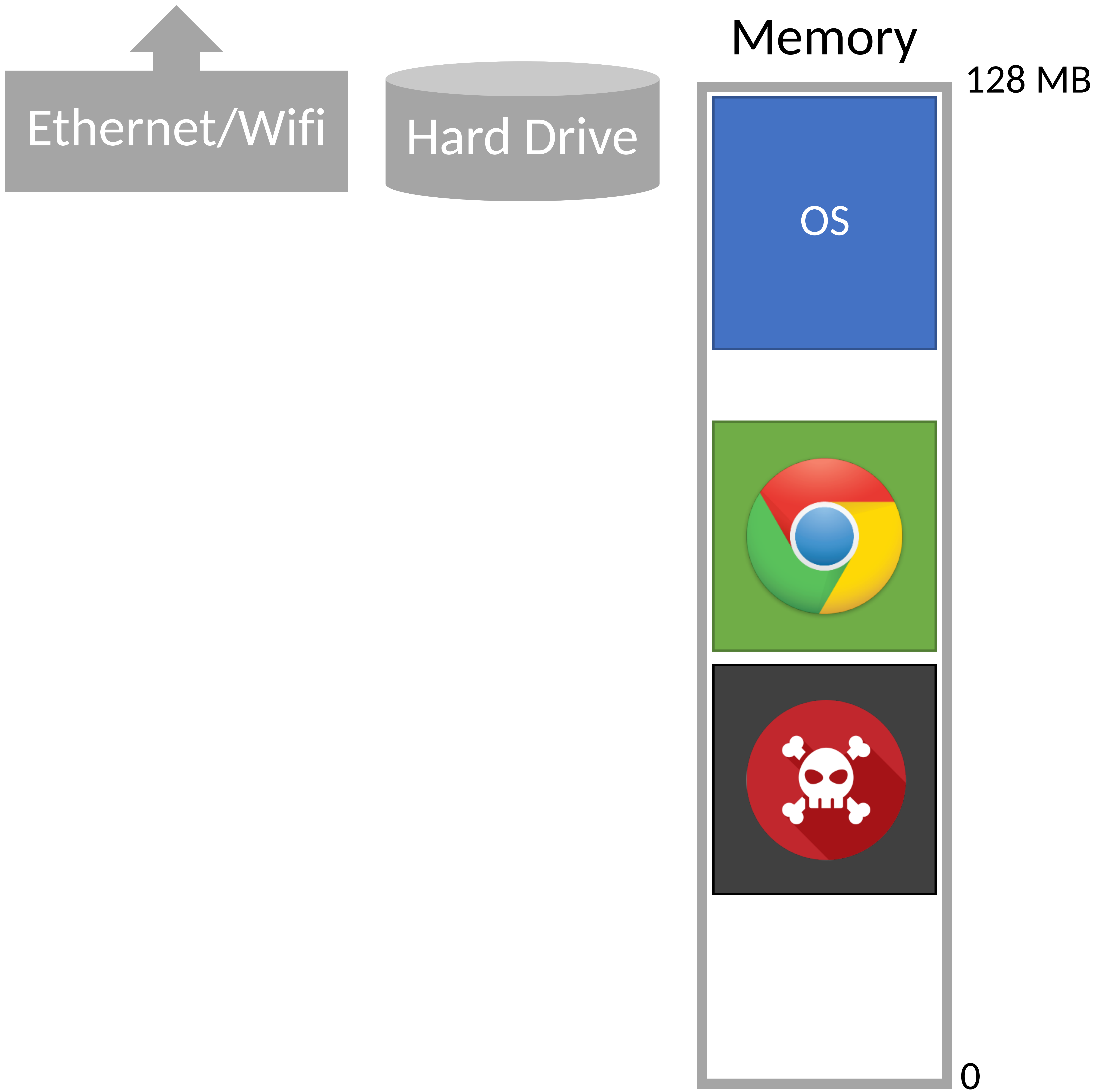


Memory Unsafety



Problem: any process can read/write any memory

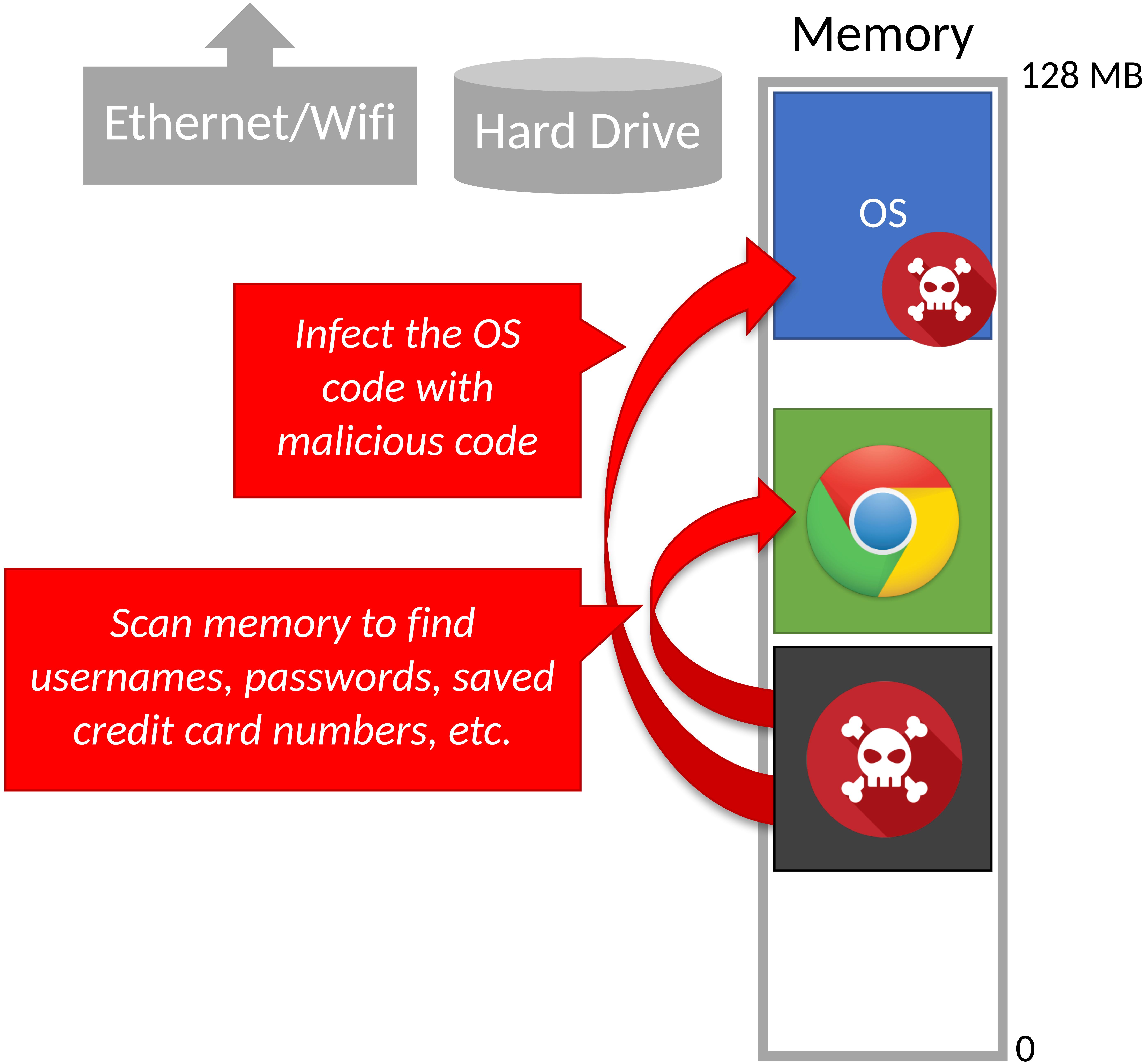
Memory Unsafety



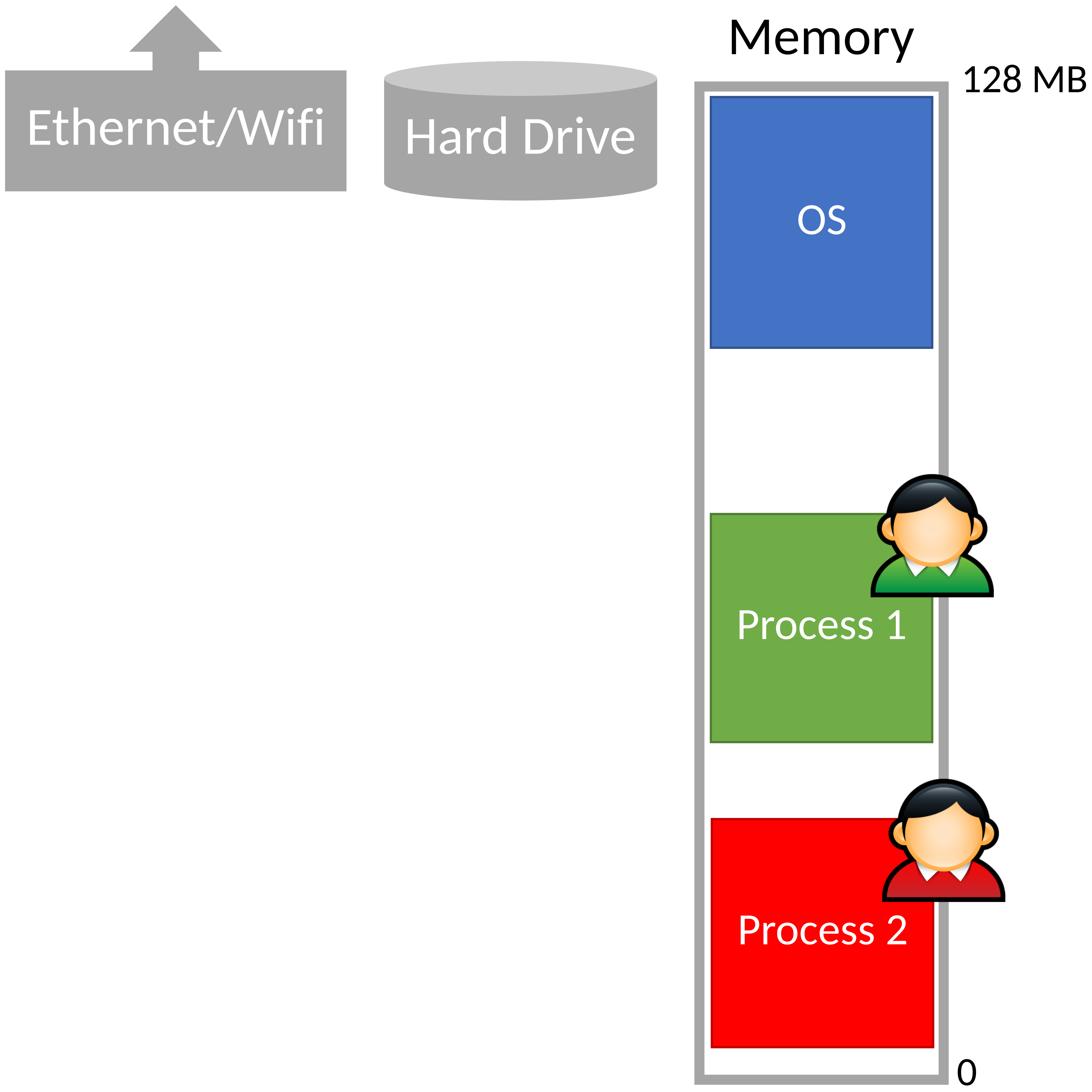
Problem: any process can read/write any memory

Memory Unsafety

Problem: any process can read/write any memory



Device Unsafety



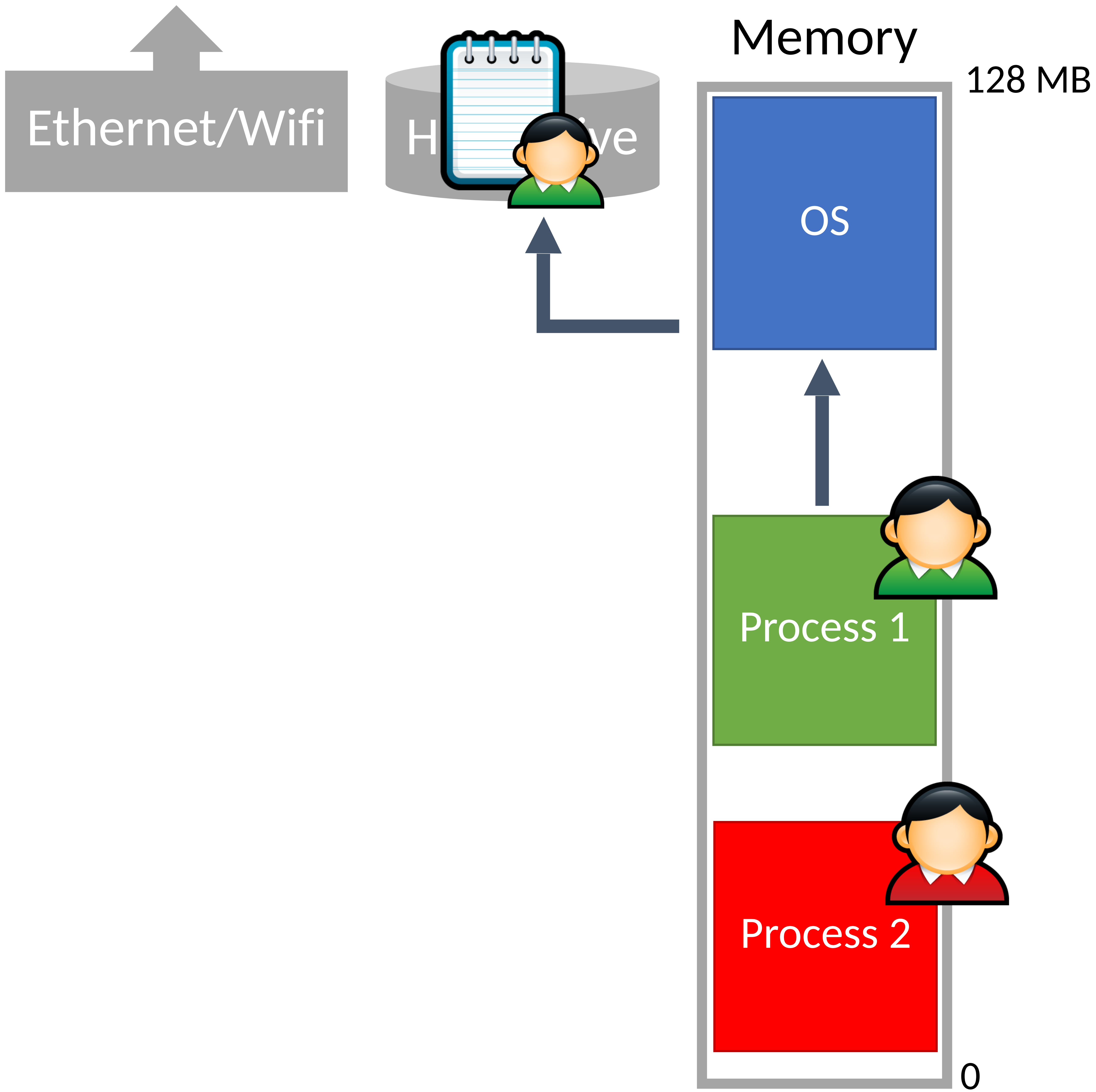
Problem: any process can access any hardware device directly

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

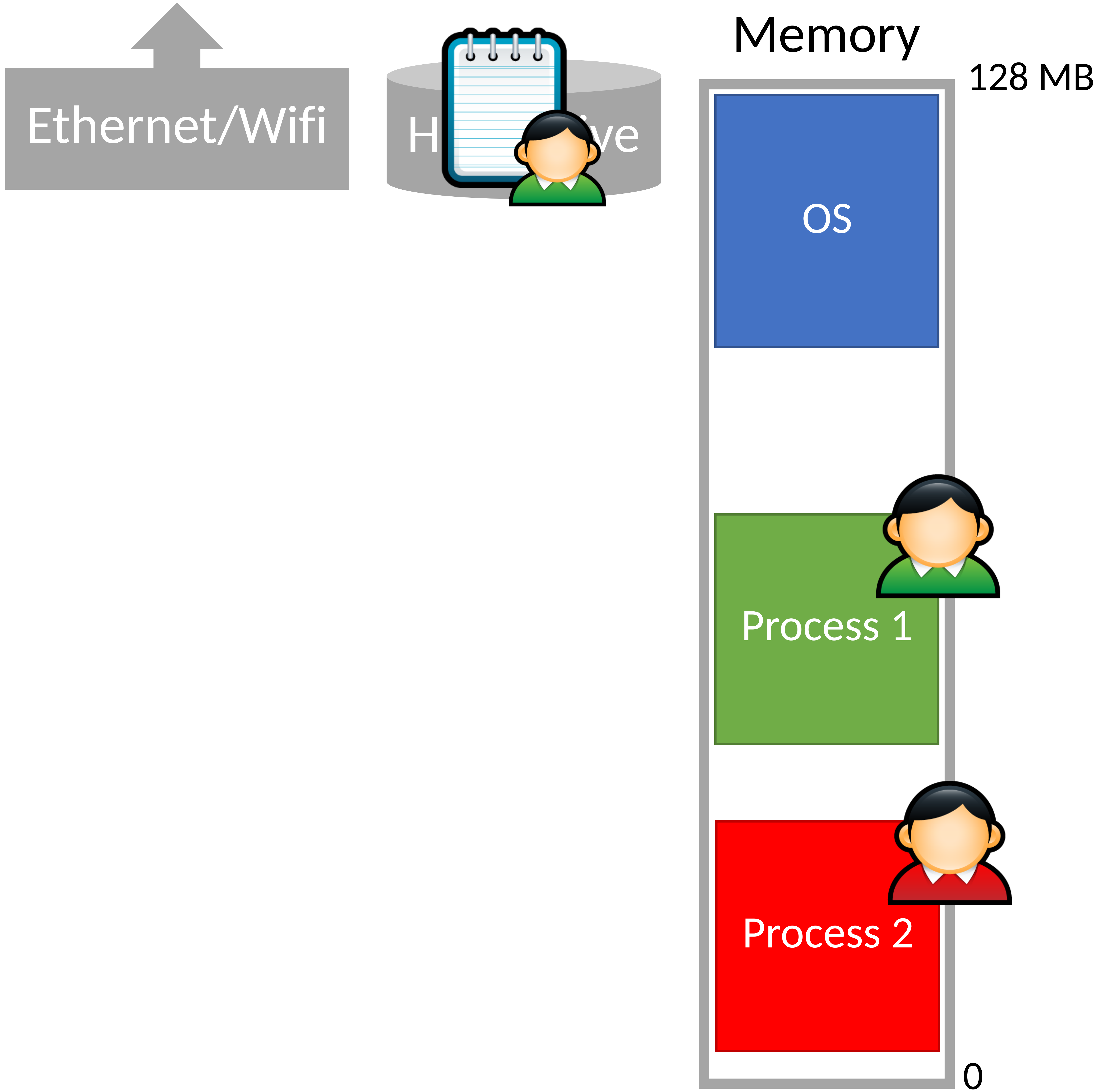
Device Unsafety

Problem: any process can access any hardware device directly

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



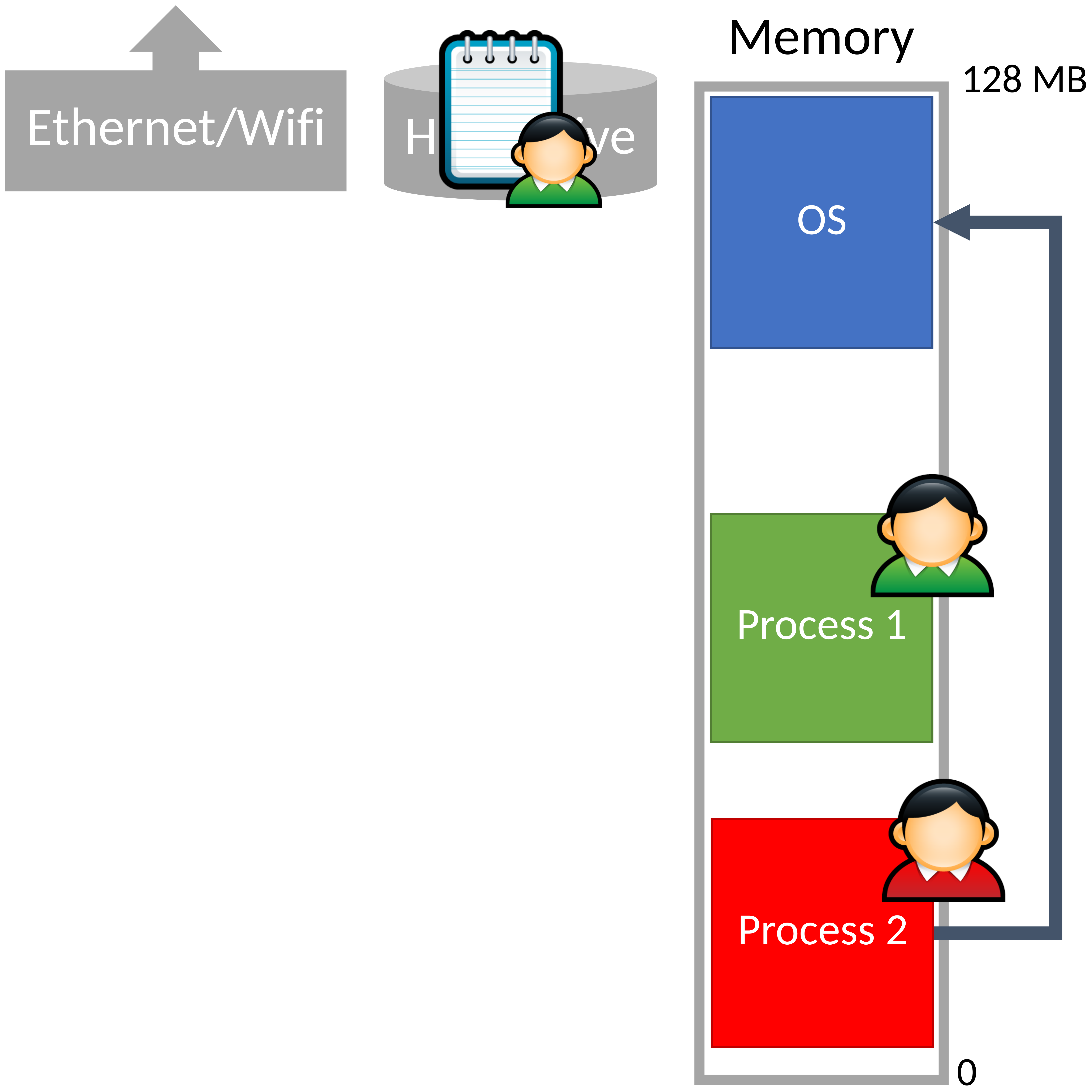
Device Unsafety



Problem: any process can access any hardware device directly

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

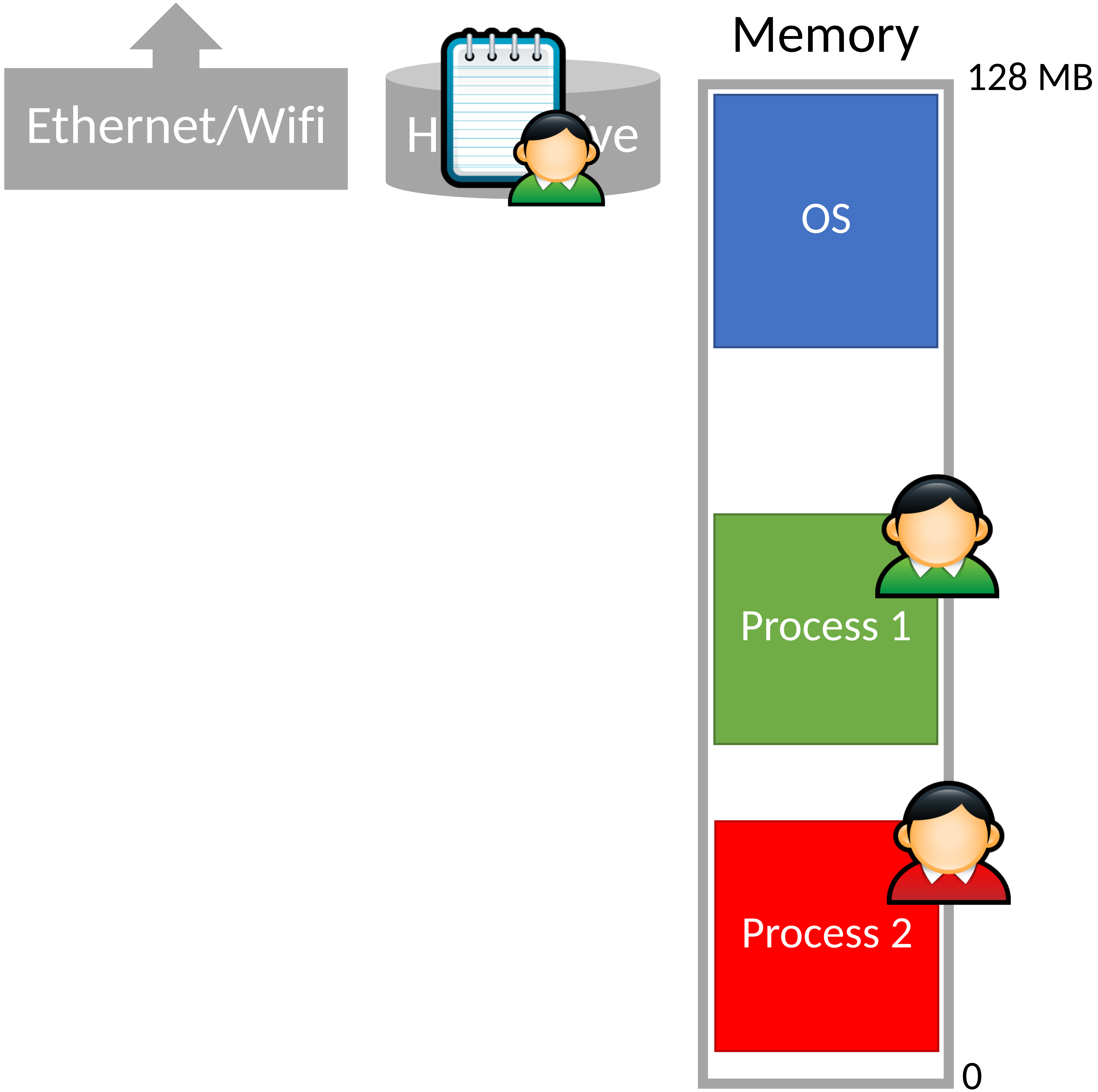
Device Unsafety



Problem: any process can access any hardware device directly

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

Device Unsafety



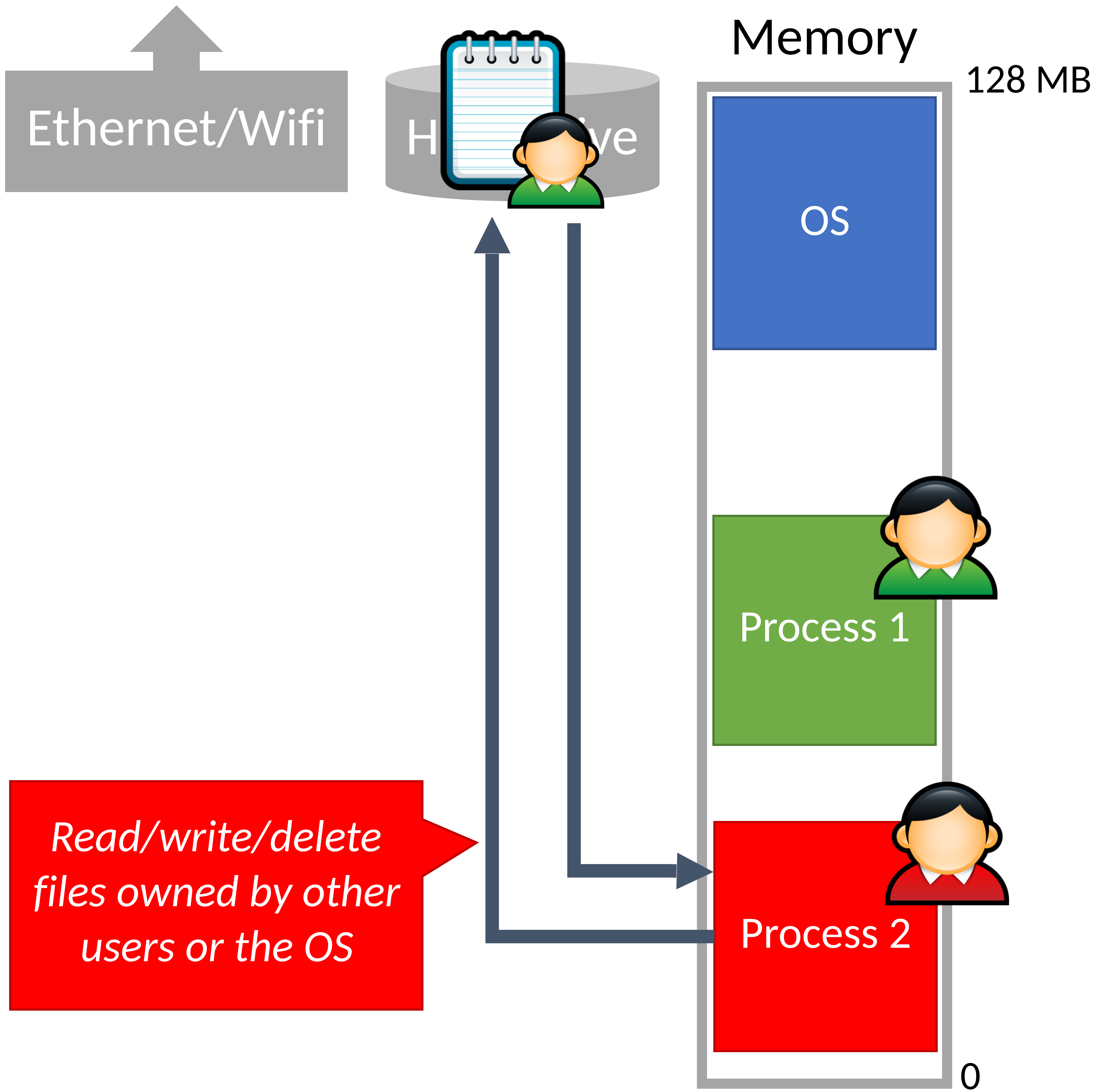
Problem: any process can access any hardware device directly

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

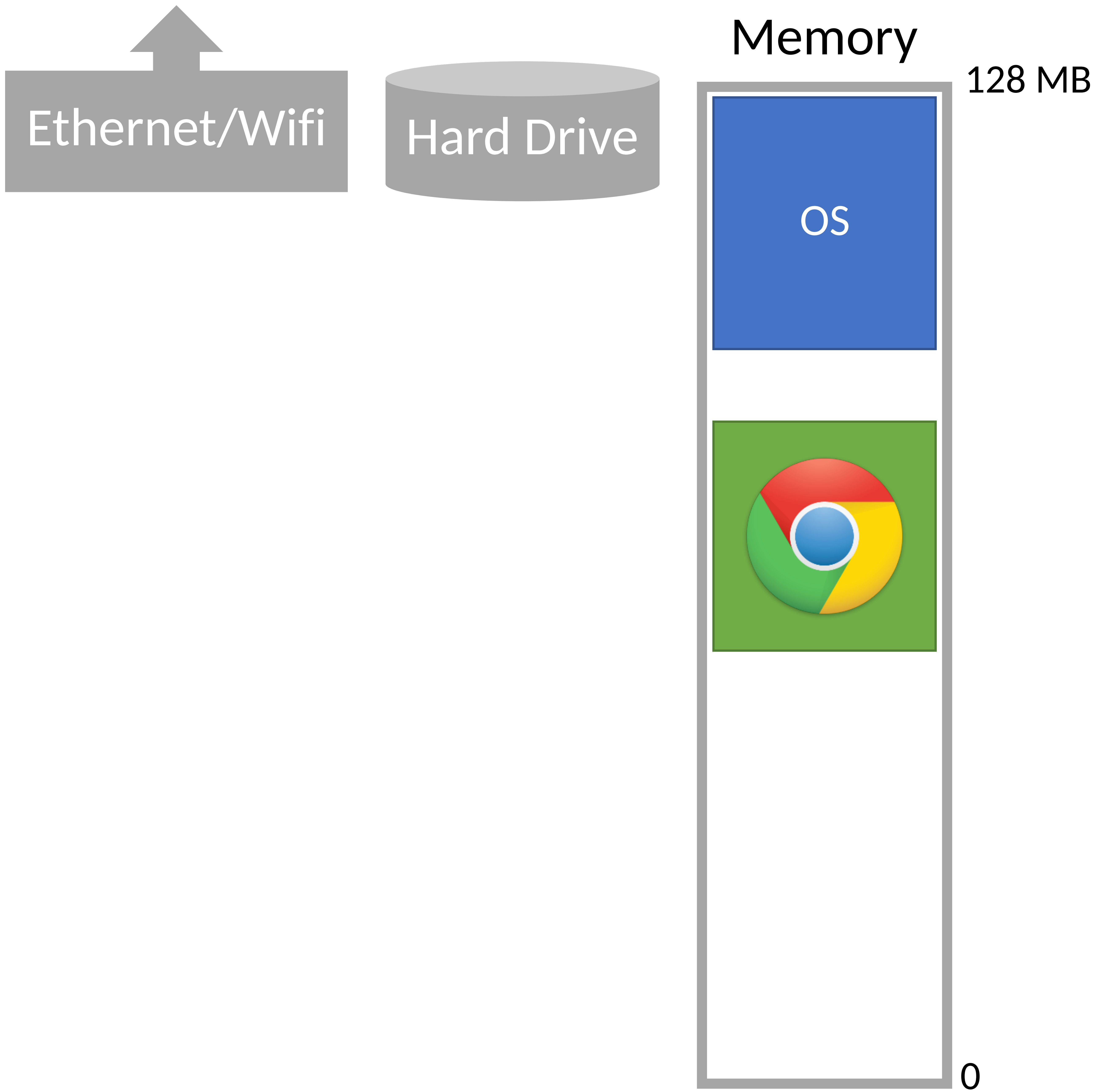
Device Unsafety

Problem: any process can access any hardware device directly

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



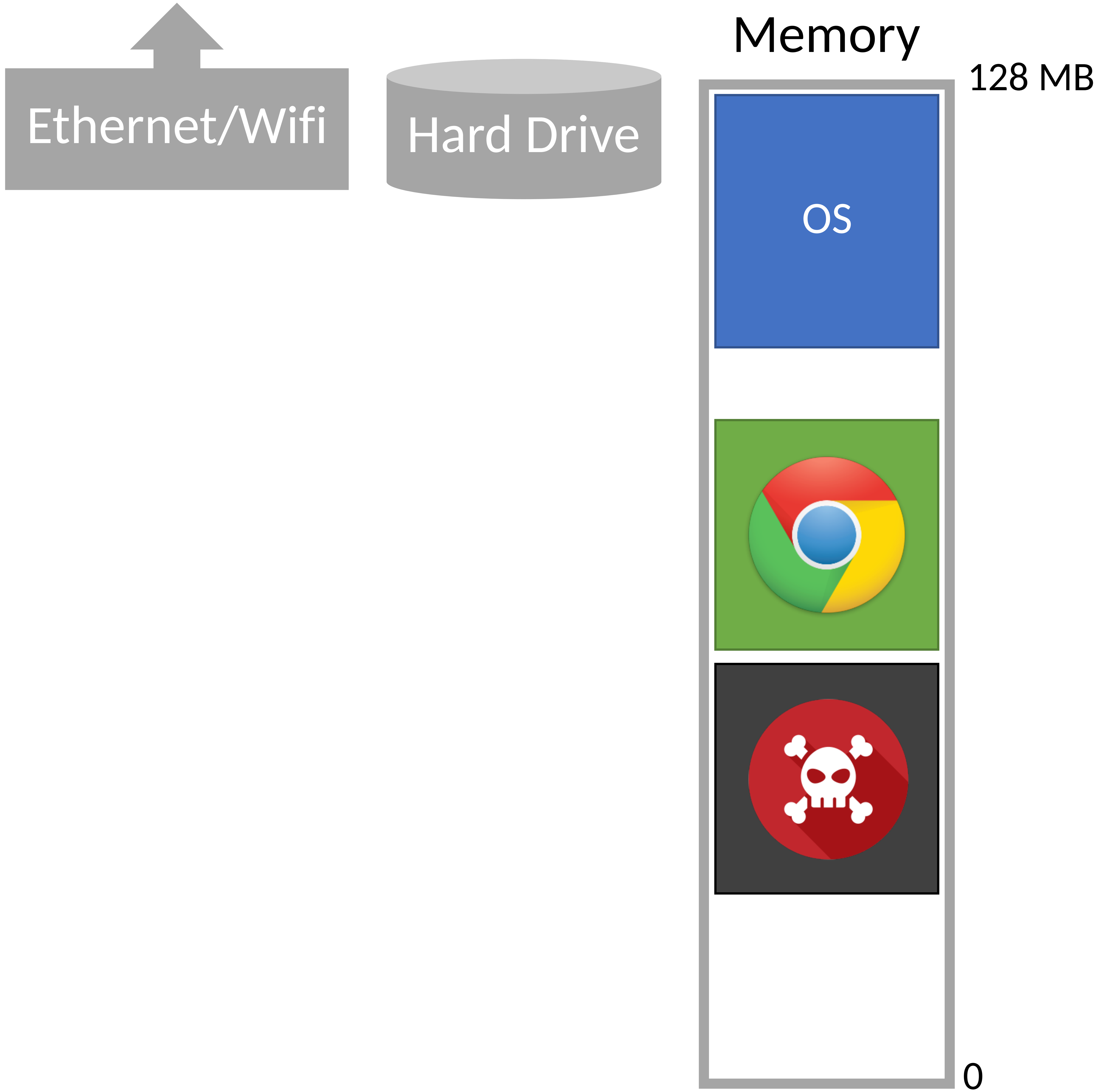
Device Unsafety



Problem: any process can access any hardware device directly

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

Device Unsafety



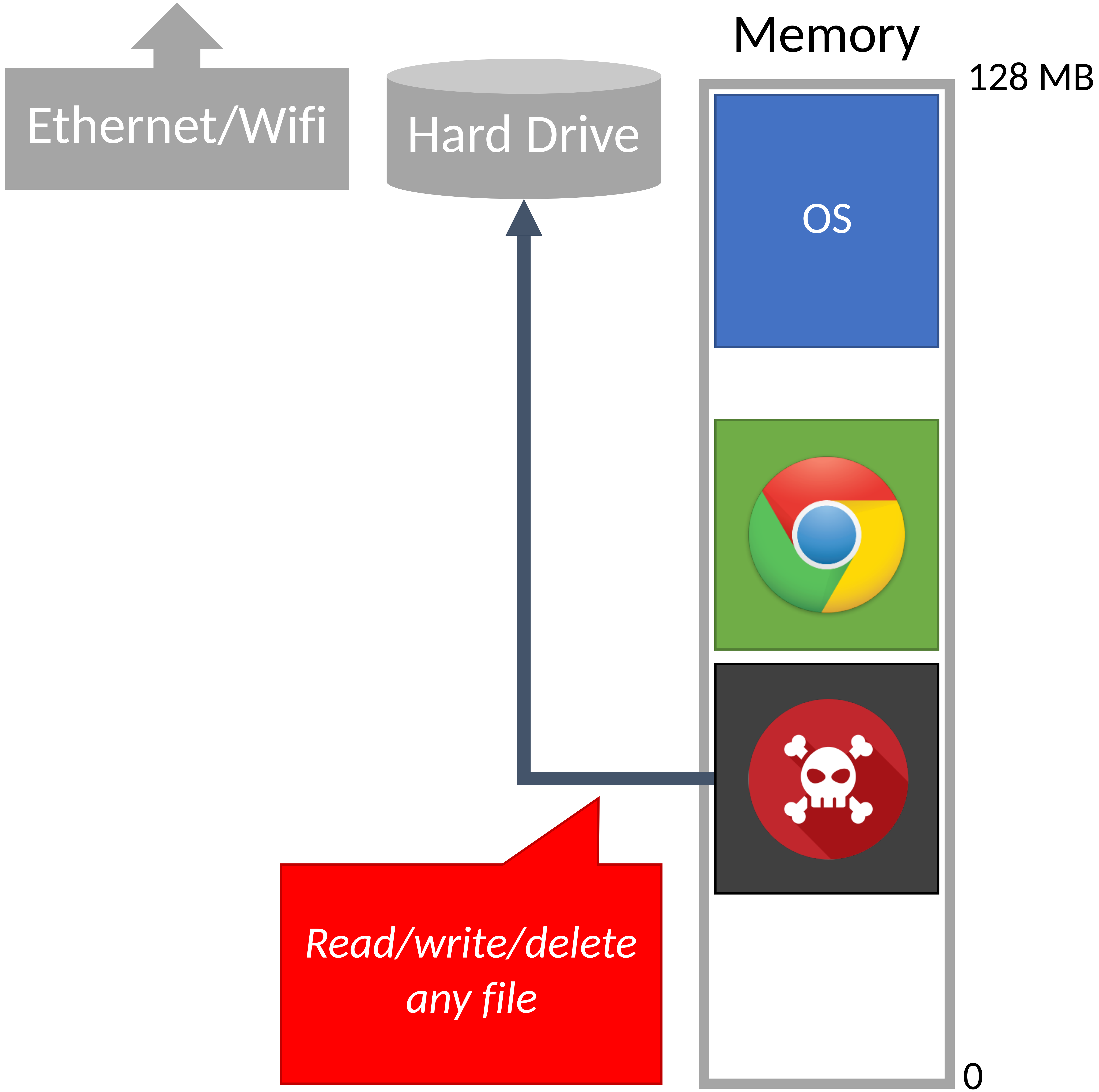
Problem: any process can access any hardware device directly

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

Device Unsafety

Problem: any process can access any hardware device directly

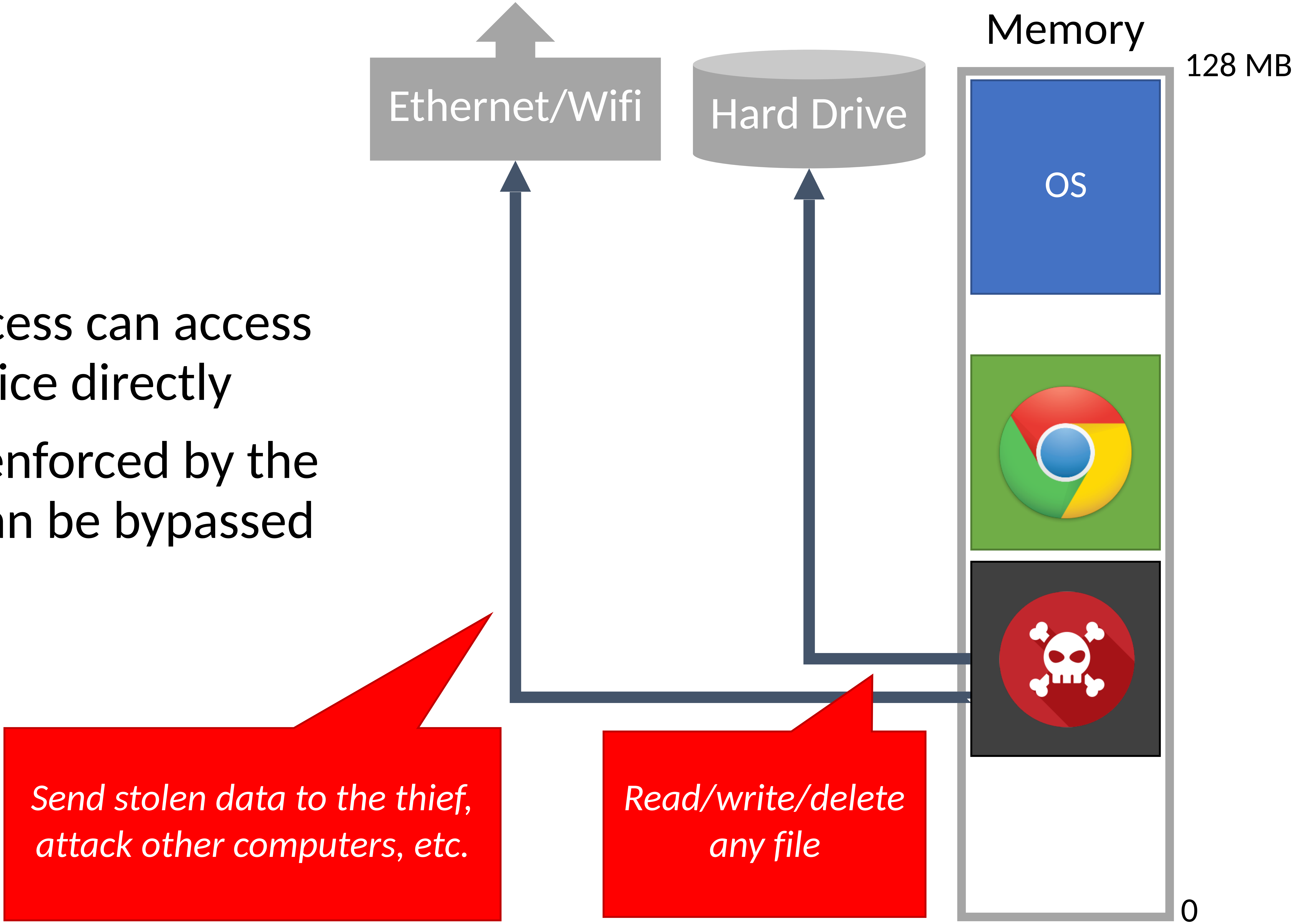
Access control is enforced by the OS, but OS APIs can be bypassed



Device Unsafety

Problem: any process can access any hardware device directly

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



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


ASUS

Towards Modern Architecture

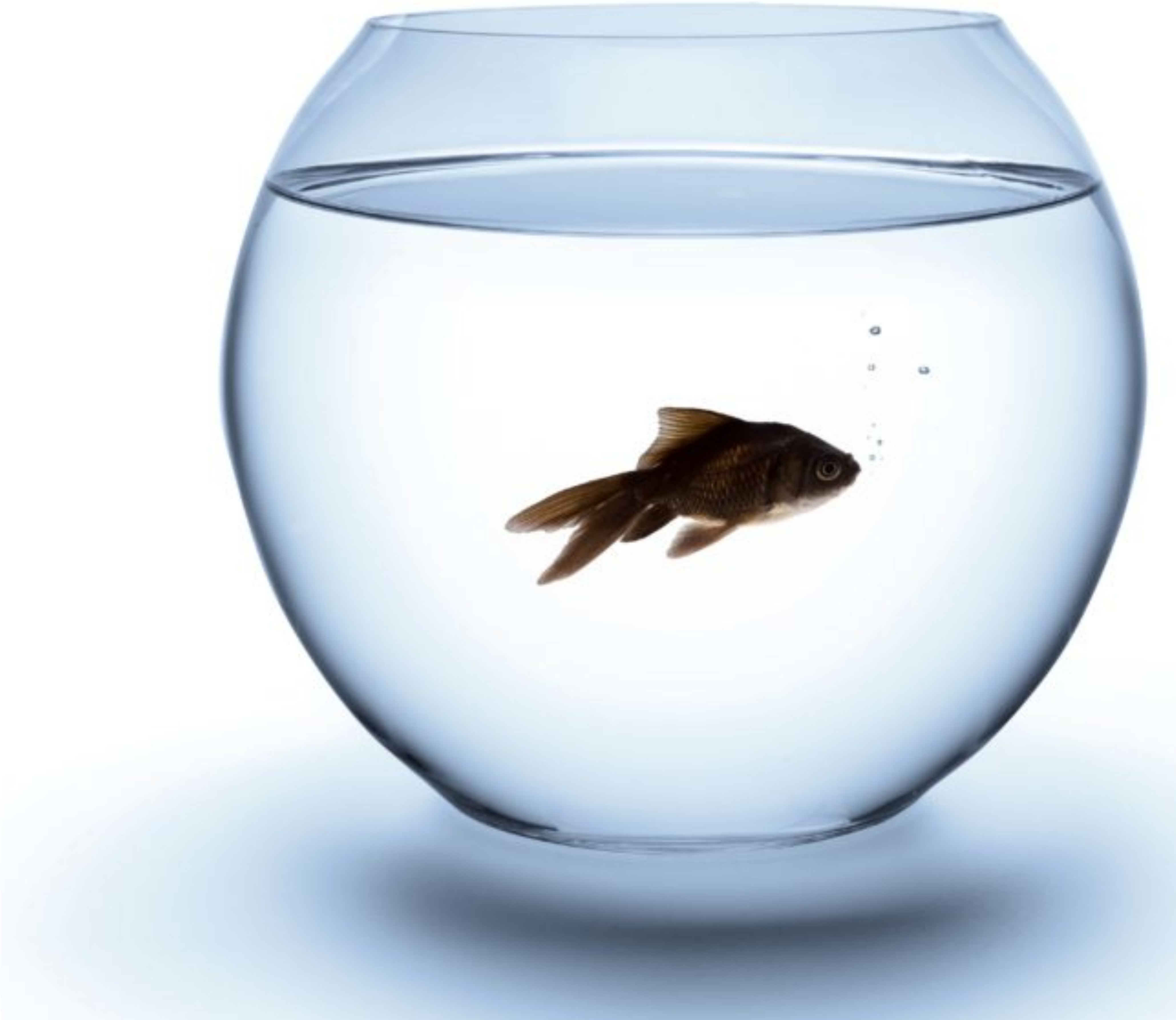
To achieve systems security, we need process isolation

- 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



Protected Mode

Protected Mode

Most modern CPUs support protected mode

x86 CPUs support three rings with different privileges

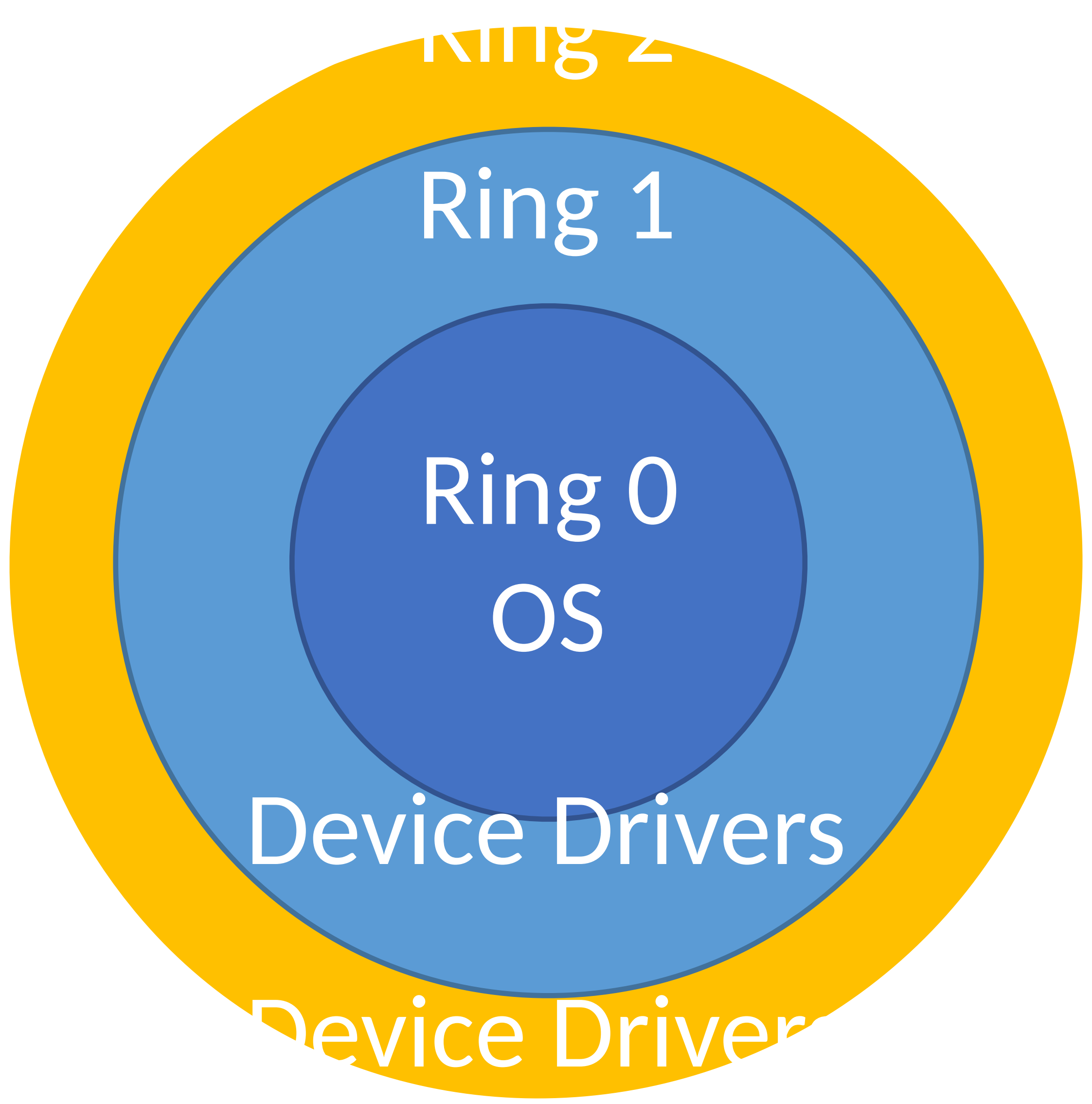
- Ring 0: Operating System
  - Code in this ring may directly access any device

Protected Mode

Most modern CPUs support protected mode

x86 CPUs support three rings with different privileges

- Ring 0: Operating System
  - Code in this ring may directly access any device
- Ring 1, 2: device drivers
  - Code in these rings may directly access some devices
  - May not change the protection level of the CPU

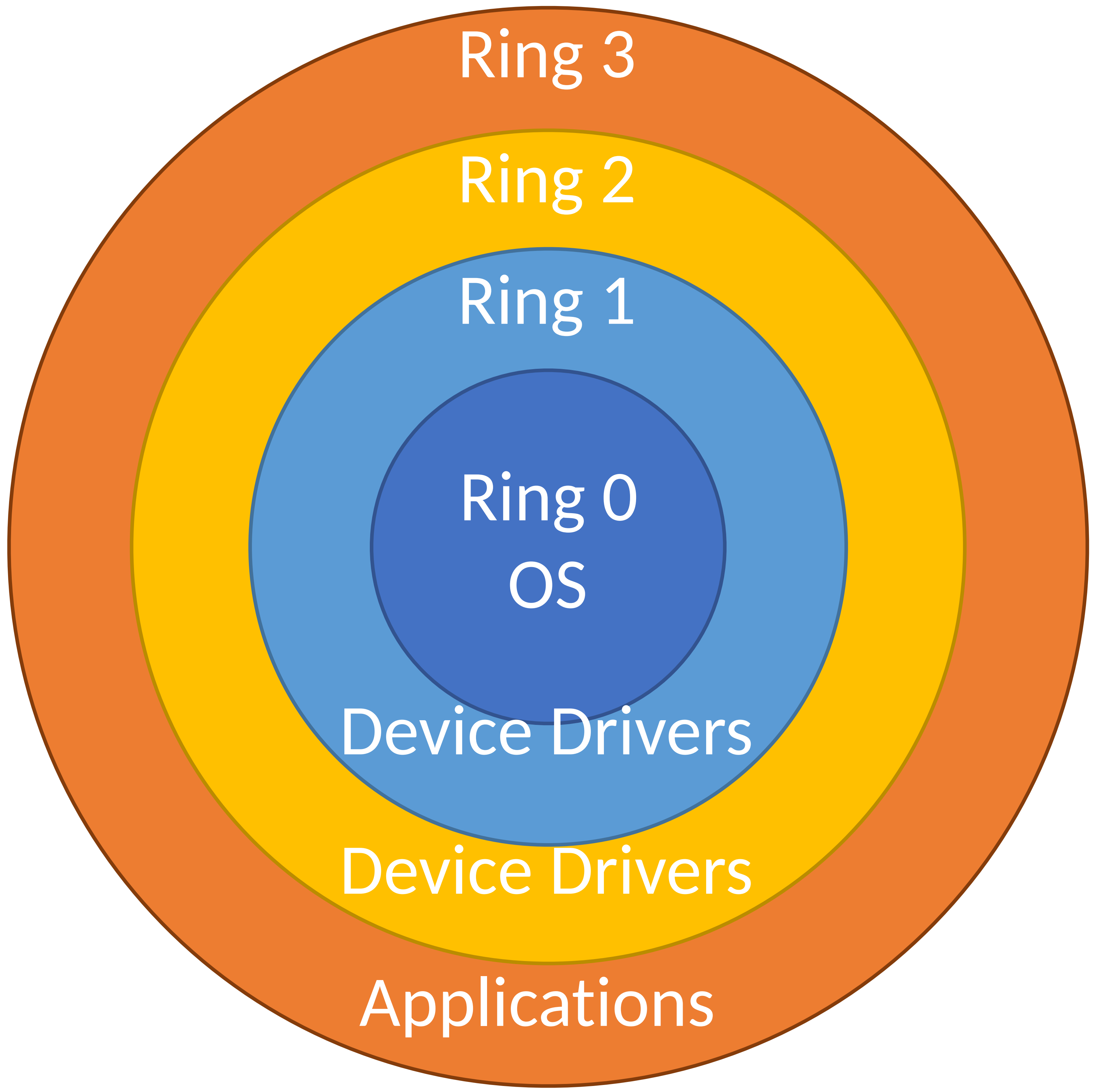


Protected Mode

Most modern CPUs support protected mode

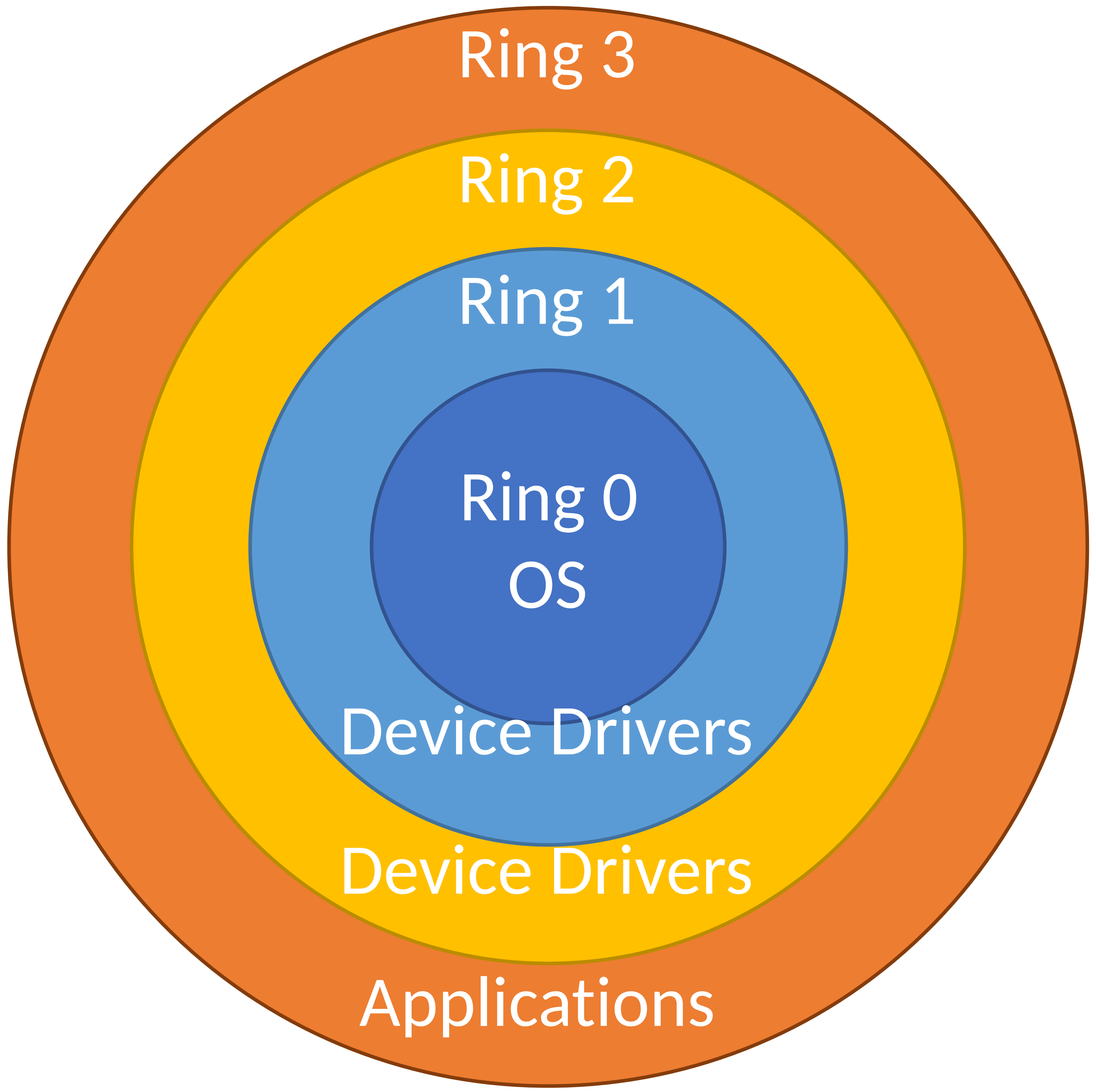
x86 CPUs support three rings with different privileges

- Ring 0: Operating System
  - Code in this ring may directly access any device
- Ring 1, 2: device drivers
  - Code in these rings may directly access some devices
  - May not change the protection level of the CPU
- Ring 3: userland
  - Code in this ring may not directly access devices
  - All device access must be via OS APIs
  - May not change the protection level of the CPU



Protected Mode

- Most modern CPUs support protected mode
- x86 CPUs support three rings with different privileges
  - Ring 0: Operating System
    - Code in this ring may directly access any device
  - Ring 1, 2: device drivers
    - Code in these rings may directly access some devices
    - May not change the protection level of the CPU
  - Ring 3: userland
    - Code in this ring may not directly access devices
    - All device access must be via OS APIs
    - May not change the protection level of the CPU
- Most OSes only use rings 0 and 3



Ring -1,-2,-3

“Google cited worries that the Intel ME (actually MINIX) code runs on their CPU's deepest access level — Ring "-3" — and also runs a web server component that allows anyone to remotely connect to remote computers, even when the main OS is turned off.”

System Boot Sequence

1. On startup, the CPU starts in 16-bit real mode
   - Protected mode is disabled
   - Any process can access any device

System Boot Sequence

1. On startup, the CPU starts in 16-bit real mode
   - Protected mode is disabled
   - Any process can access any device
2. BIOS executes, finds and loads the OS

System Boot Sequence

1. On startup, the CPU starts in 16-bit real mode
   - Protected mode is disabled
   - Any process can access any device
2. BIOS executes, finds and loads the OS
3. OS switches CPU to 32-bit protected mode
   - OS code is now running in Ring 0
   - OS decides what Ring to place other processes in

System Boot Sequence

1. On startup, the CPU starts in 16-bit real mode
    - Protected mode is disabled
    - Any process can access any device
2. BIOS executes, finds and loads the OS
3. OS switches CPU to 32-bit protected mode
    - OS code is now running in Ring 0
    - OS decides what Ring to place other processes in
4. Shell gets executed, user may run programs
    - User processes are placed in Ring 3

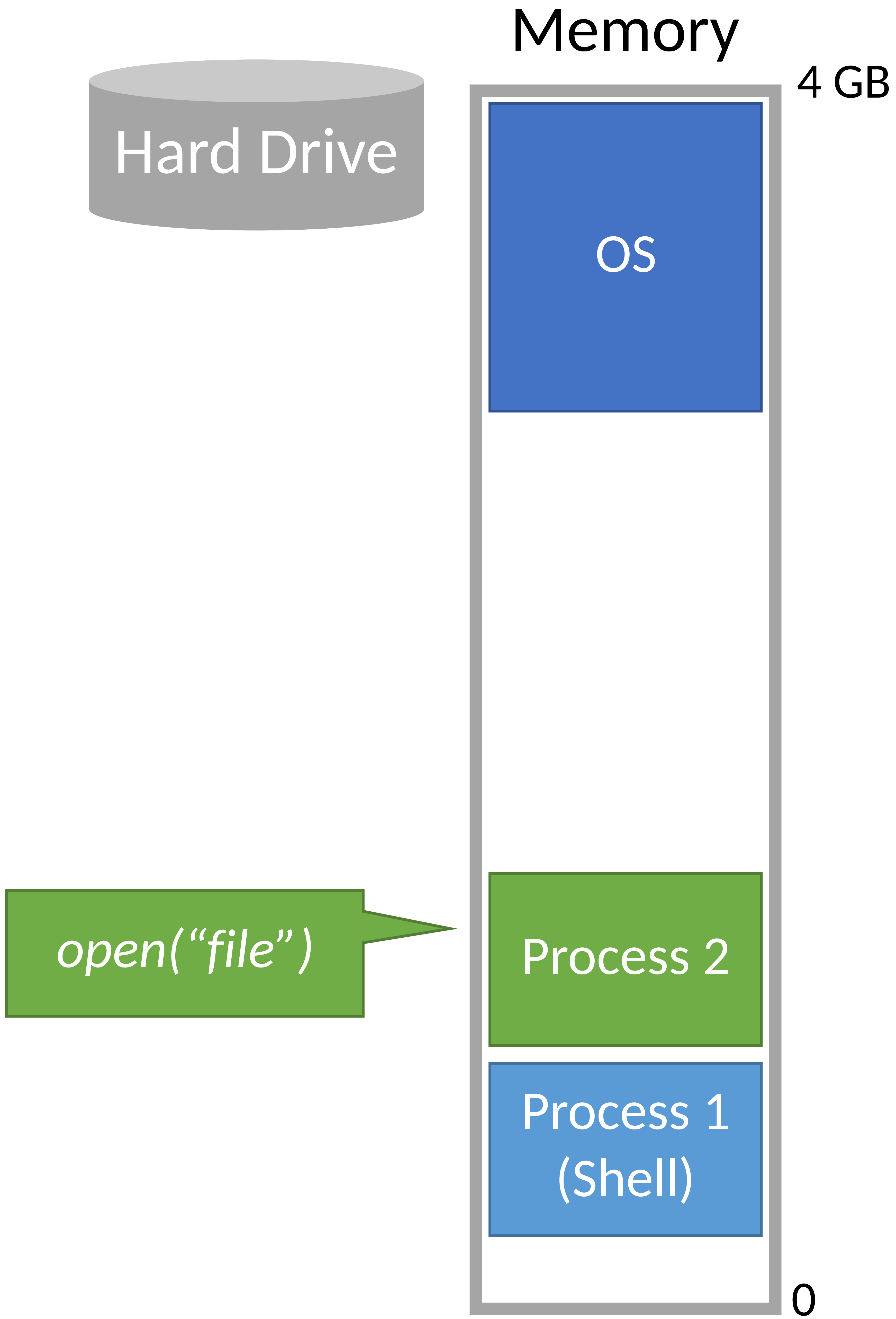
Restriction on Privileged Instructions

What CPU instructions are restricted in protected mode?

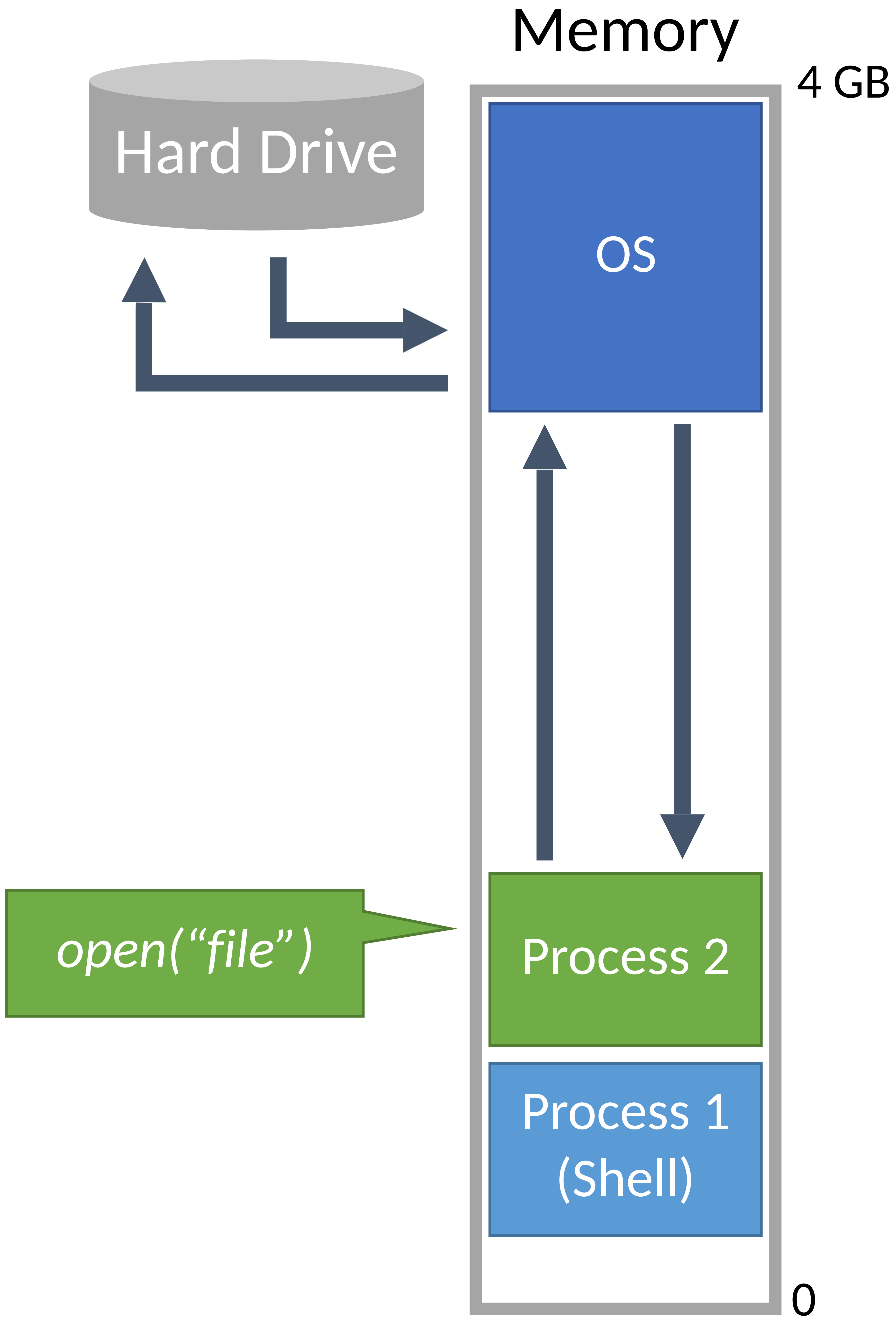
- Any instruction that modifies the CR0 register
  - Controls whether protected mode is enabled
- Any instruction that modifies the CR3 register
  - Controls the virtual memory configuration
  - More on this later…
- hlt – Halts the CPU
- sti/cli – enable and disable interrupts
- in/out – directly access hardware devices

If a Ring 3 process tries any of these things, it immediately crashes

How to change modes



How to change modes



Changing Modes

Applications often need to access the OS APIs

- Writing files
- Displaying things on the screen
- Receiving data from the network
- etc...

But the OS is Ring 0, and processes are Ring 3

How do processes get access to the OS?

Changing Modes

Applications often need to access the OS APIs

- Writing files
- Displaying things on the screen
- Receiving data from the network
- etc...

But the OS is Ring 0, and processes are Ring 3

How do processes get access to the OS?

- Invoke OS APIs with special assembly instructions
  - Interrupt: `int 0x80`
  - System call: `sysenter` or `syscall`
- `int`/`sysenter`/`syscall` cause a mode transfer from Ring 3 to Ring 0

Mode Transfer

Userland

1. Application executes trap (int) instruction
   - EIP, CS, and EFLAGS get pushed onto the stack
   - Mode switches from ring 3 to ring 0

Kernel Mode

Mode Transfer

Userland

1. Application executes trap (int) instruction
   - EIP, CS, and EFLAGS get pushed onto the stack
   - Mode switches from ring 3 to ring 0

Kernel Mode

2. Save the state of the current process
   - Push EAX, EBX, …, etc. onto the stack

Mode Transfer

Userland

1. Application executes trap (int) instruction
   - EIP, CS, and EFLAGS get pushed onto the stack
   - Mode switches from ring 3 to ring 0

Kernel Mode

2. Save the state of the current process
   - Push EAX, EBX, …, etc. onto the stack
3. Locate and execute the correct syscall handler

Mode Transfer

Userland

1. Application executes trap (int) instruction
   - EIP, CS, and EFLAGS get pushed onto the stack
   - Mode switches from ring 3 to ring 0

Kernel Mode

2. Save the state of the current process
   - Push EAX, EBX, …, etc. onto the stack
3. Locate and execute the correct syscall handler
4. Restore the state of process
   - Pop EAX, EBX, … etc.

Mode Transfer

Userland

1. Application executes trap (int) instruction
   - EIP, CS, and EFLAGS get pushed onto the stack
   - Mode switches from ring 3 to ring 0

Kernel Mode

2. Save the state of the current process
   - Push EAX, EBX, …, etc. onto the stack
3. Locate and execute the correct syscall handler
4. Restore the state of process
   - Pop EAX, EBX, … etc.
5. Place the return value in EAX

Mode Transfer

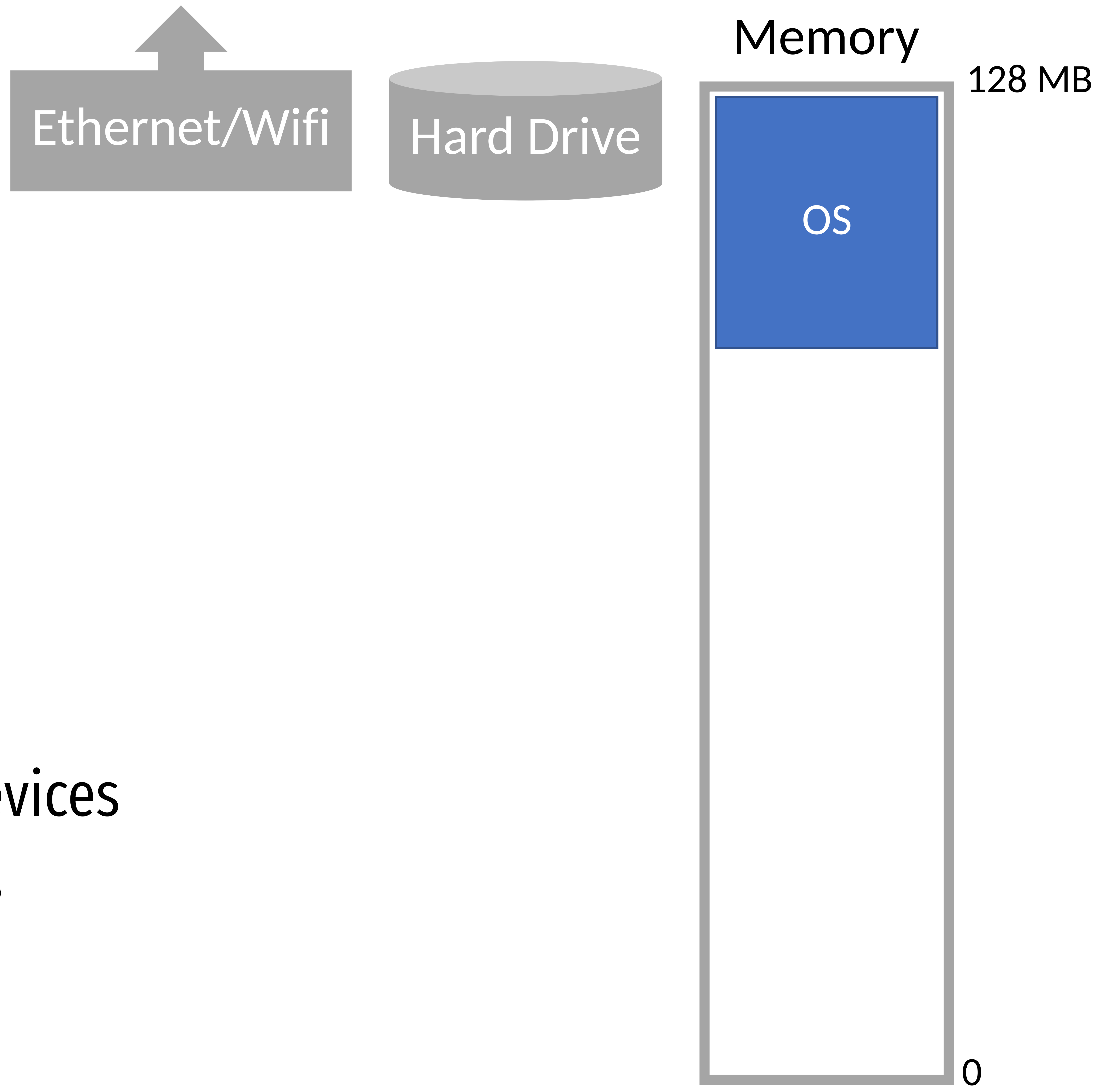
Userland

1. Application executes trap (int) instruction
   - EIP, CS, and EFLAGS get pushed onto the stack
   - Mode switches from ring 3 to ring 0

Kernel Mode

2. Save the state of the current process
   - Push EAX, EBX, …, etc. onto the stack
3. Locate and execute the correct syscall handler
4. Restore the state of process
   - Pop EAX, EBX, … etc.
5. Place the return value in EAX
6. Use iret to return to the process
   - Switches back to the original mode (typically 3)

Protection in Action

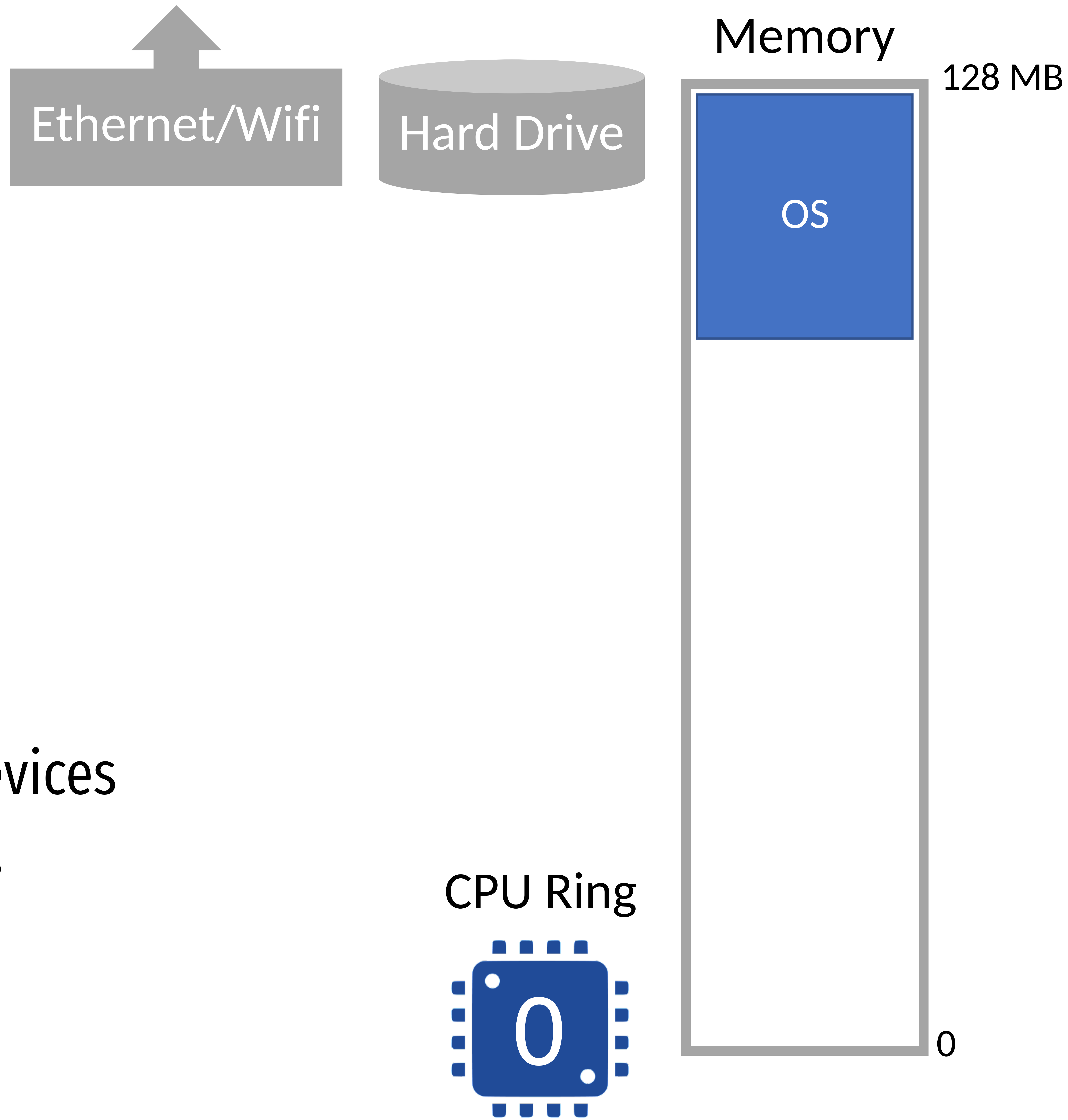


Protected mode stops direct access to devices

All device access must go through the OS

OS will impose access control checks

Protection in Action

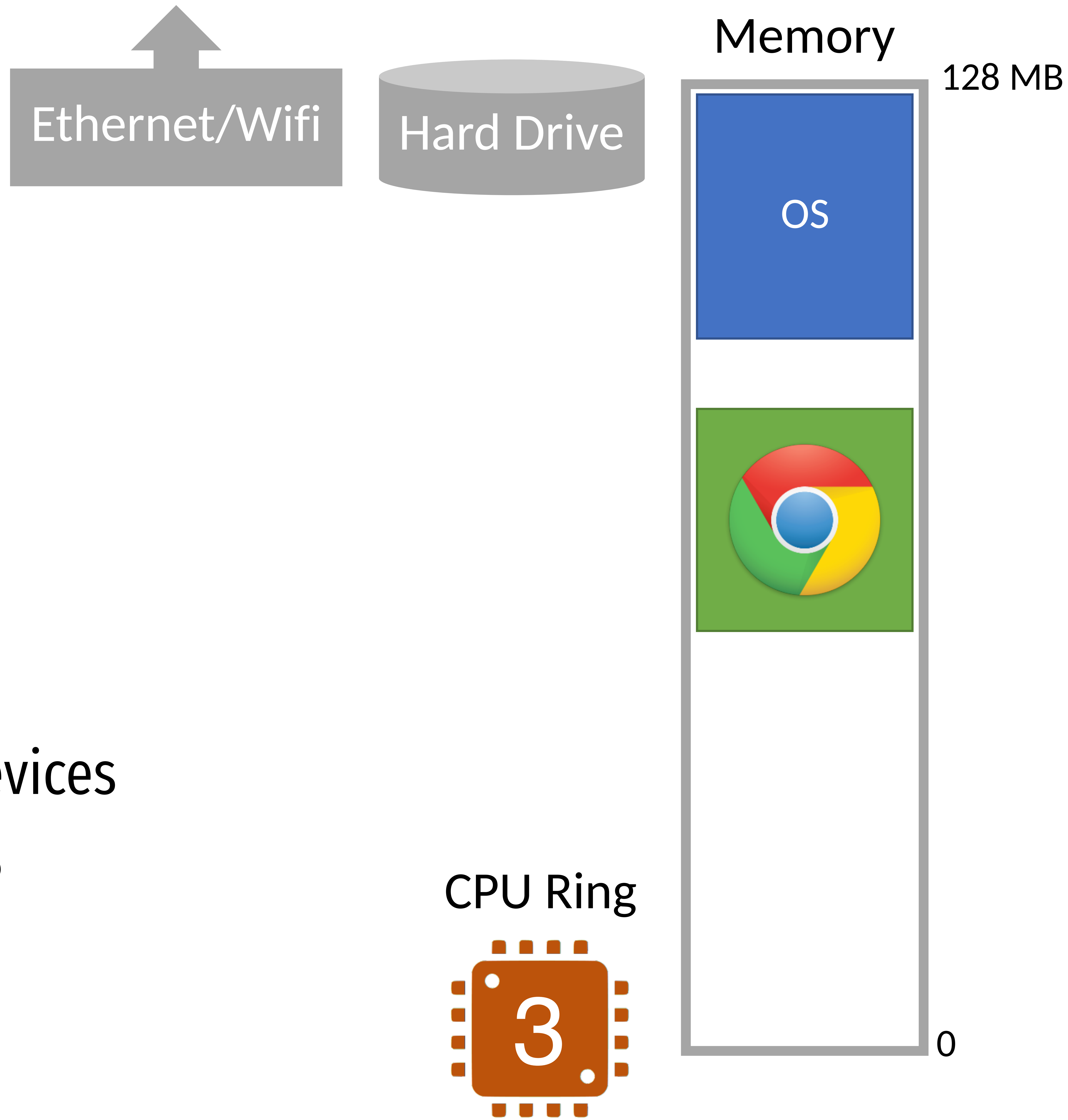


Protected mode stops direct access to devices

All device access must go through the OS

OS will impose access control checks

Protection in Action

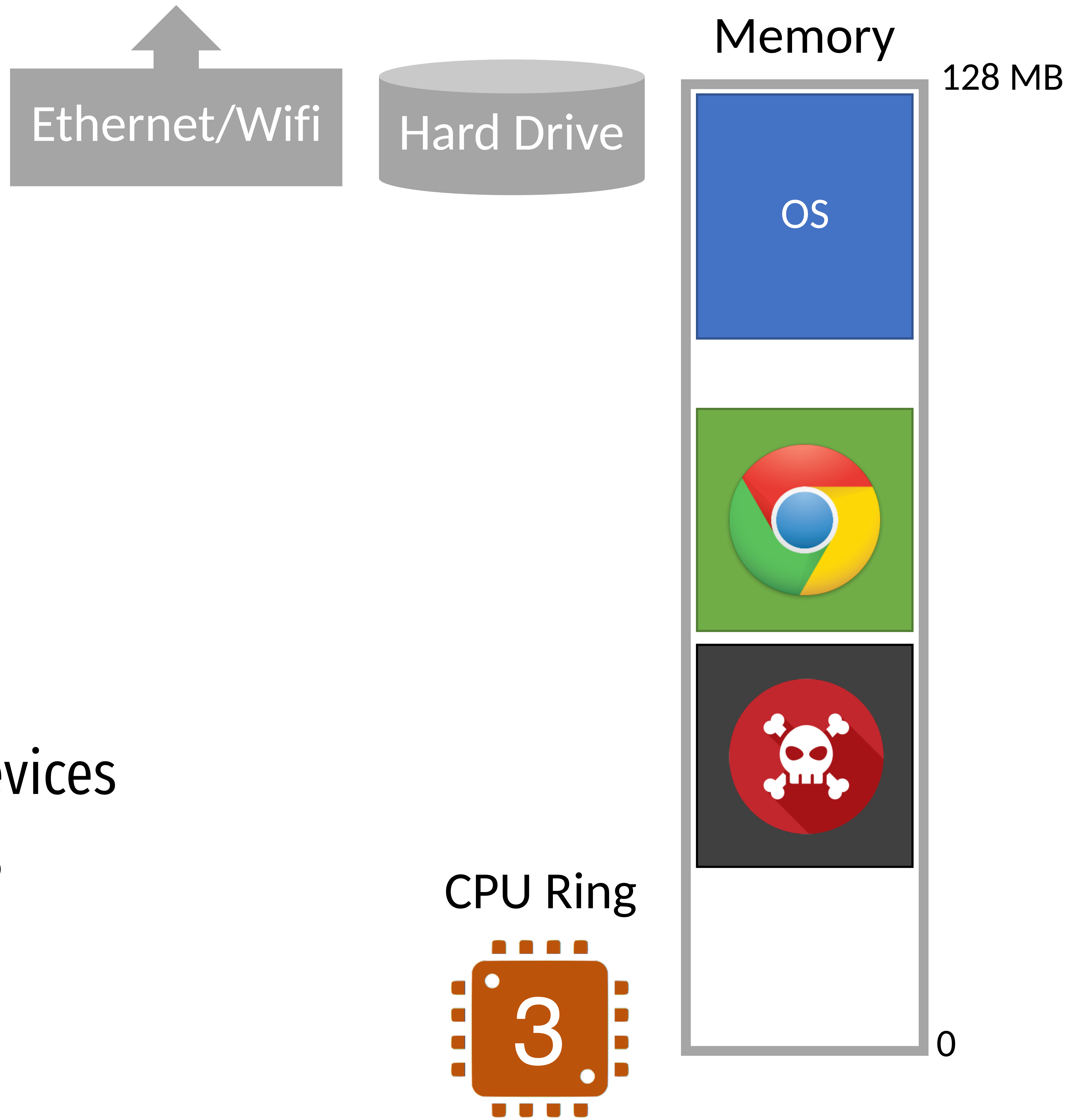


Protected mode stops direct access to devices

All device access must go through the OS

OS will impose access control checks

Protection in Action

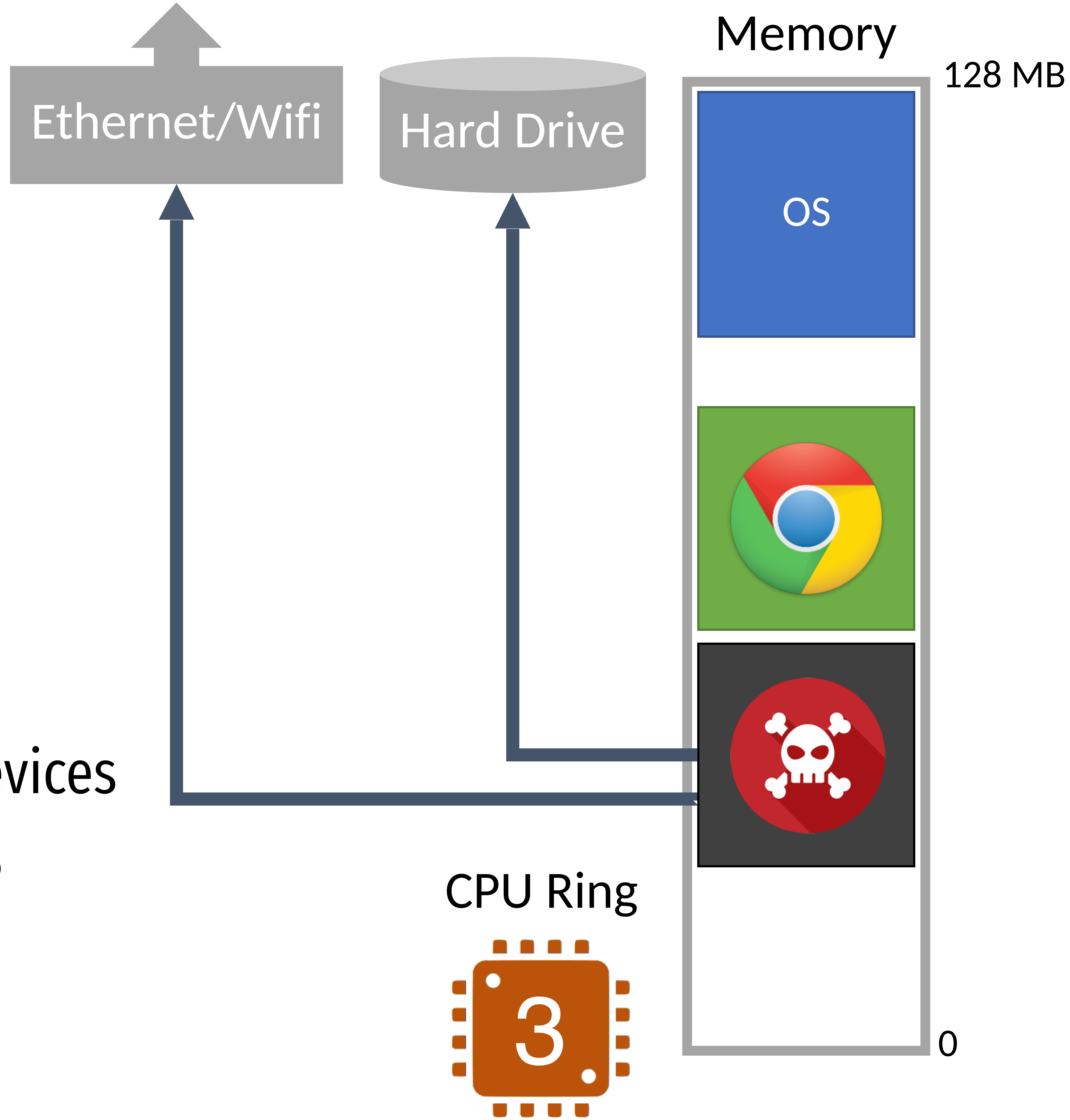


Protected mode stops direct access to devices

All device access must go through the OS

OS will impose access control checks

Protection in Action



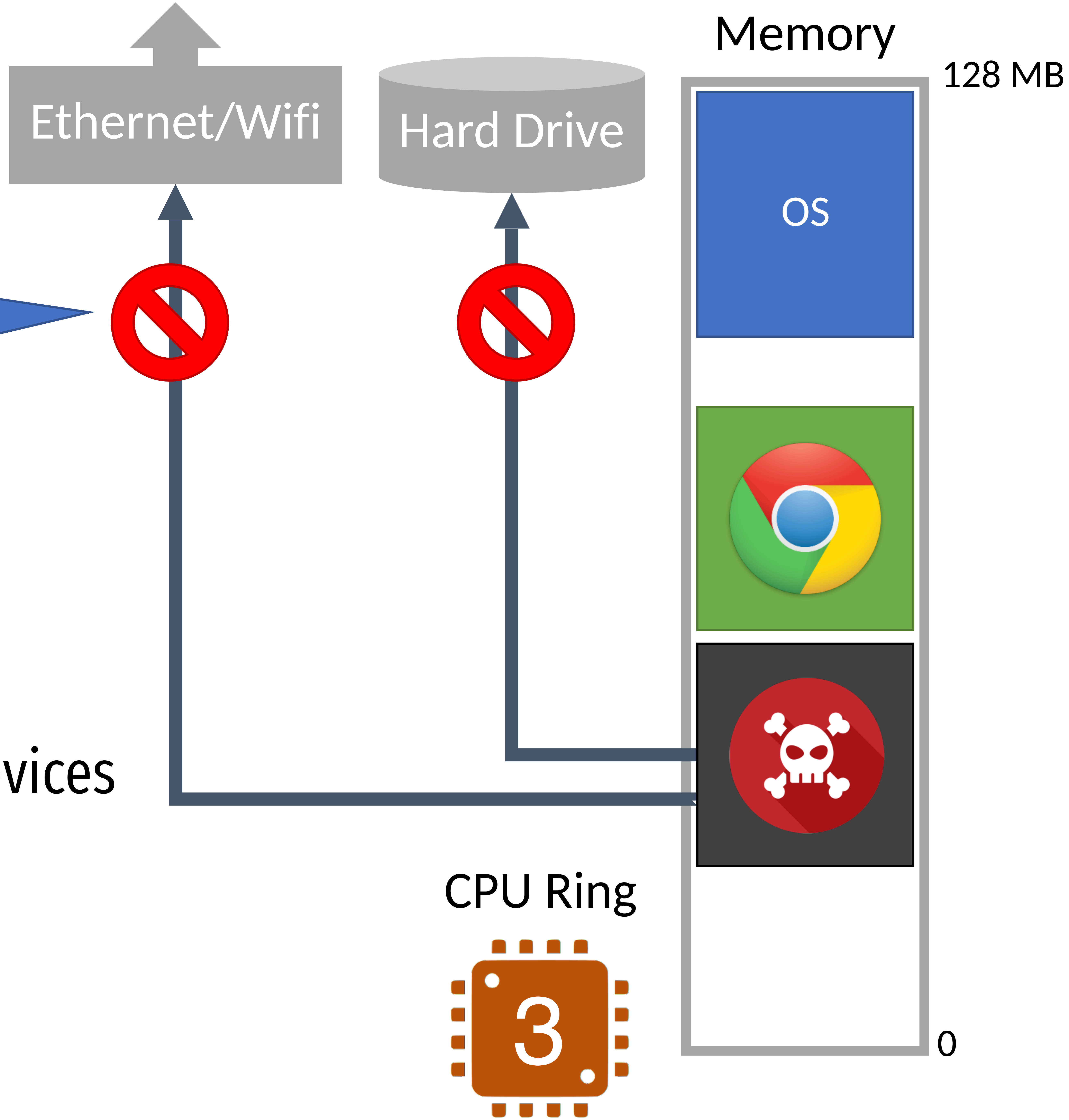
Protected mode stops direct access to devices

All device access must go through the OS

OS will impose access control checks

Protection in Action

Ring 3 = protected mode.
No direct device access

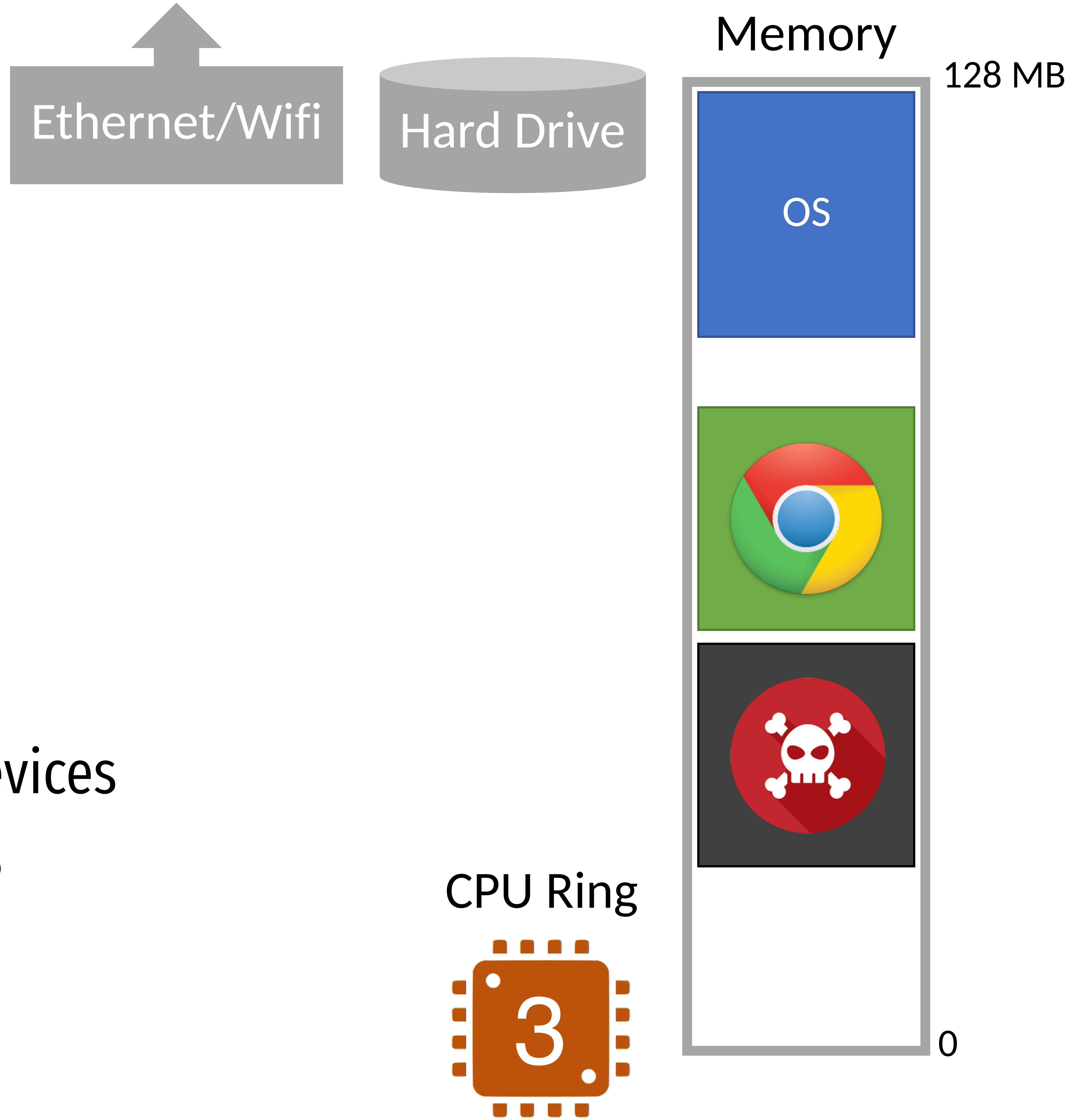


Protected mode stops direct access to devices

All device access must go through the OS

OS will impose access control checks

Protection in Action

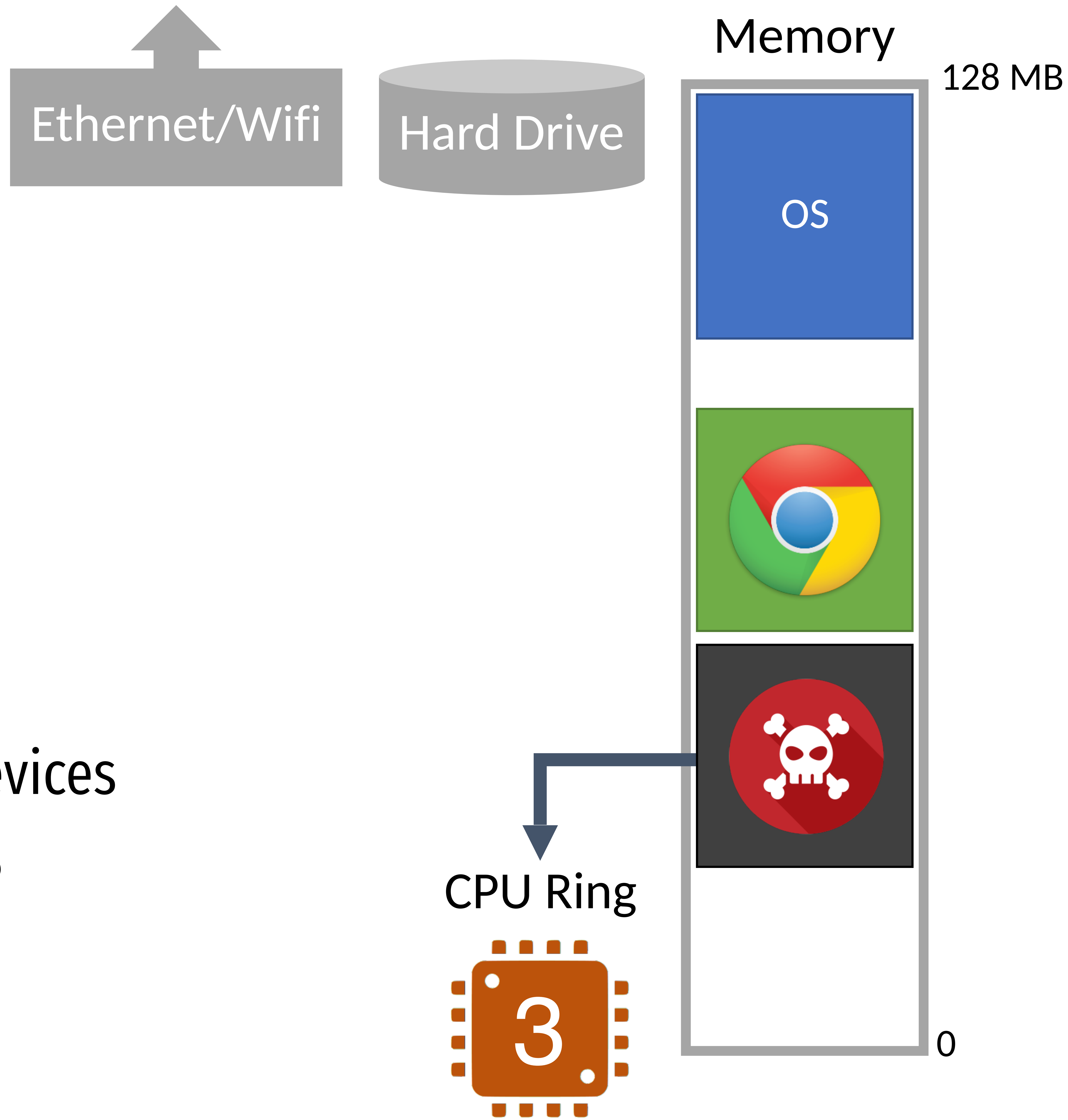


Protected mode stops direct access to devices

All device access must go through the OS

OS will impose access control checks

Protection in Action

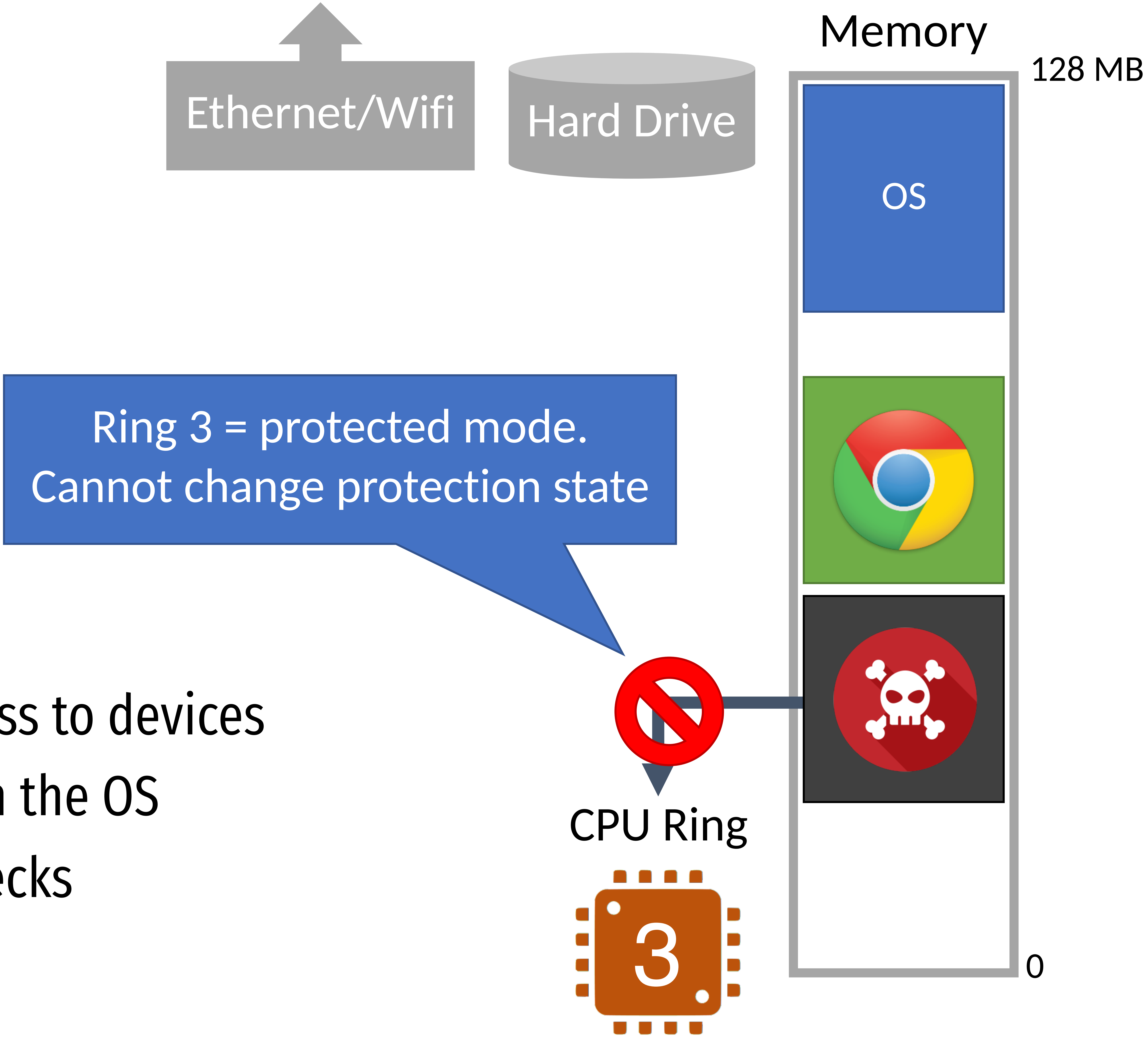


Protected mode stops direct access to devices

All device access must go through the OS

OS will impose access control checks

Protection in Action

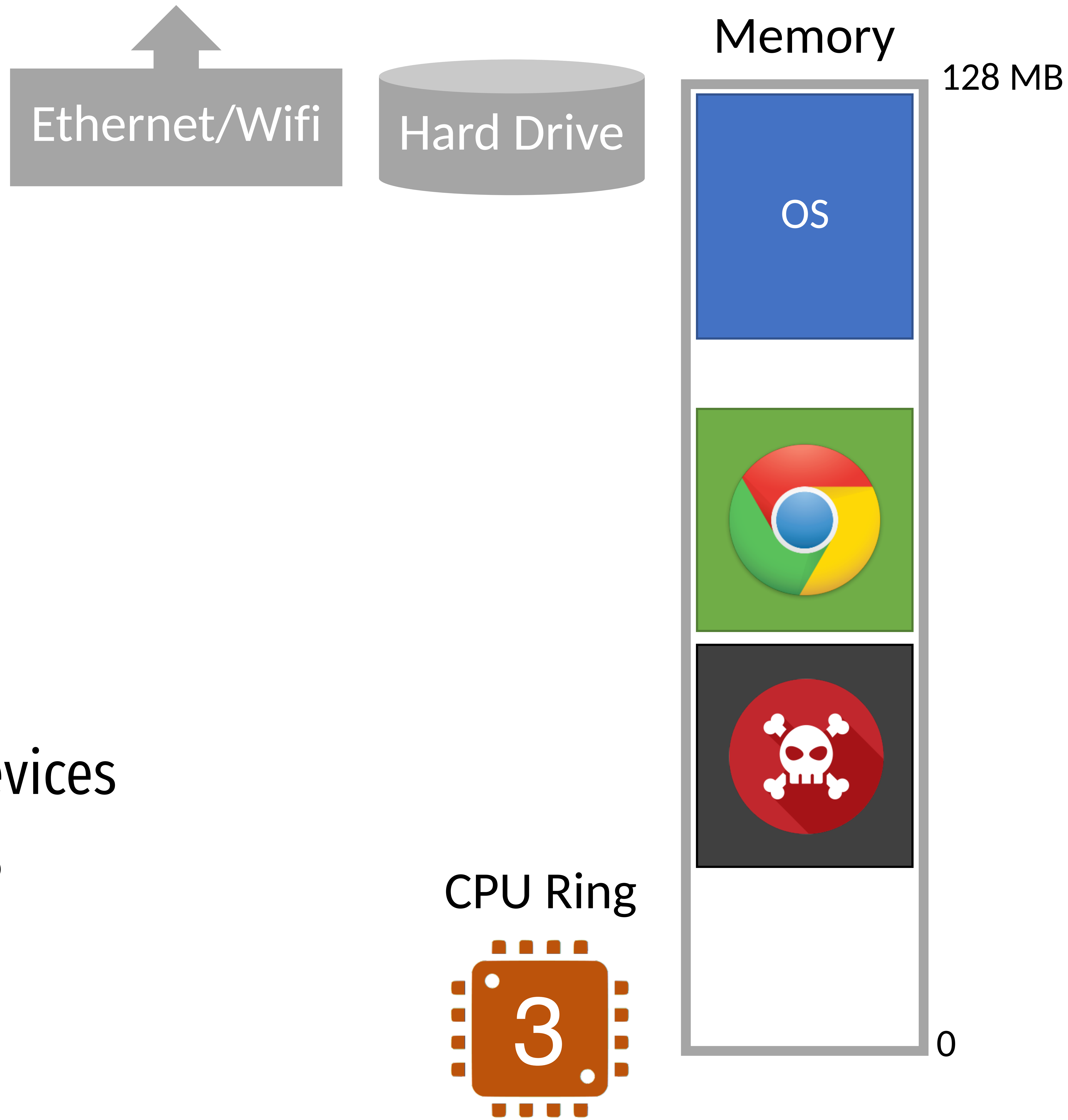


Protected mode stops direct access to devices

All device access must go through the OS

OS will impose access control checks

Protection in Action

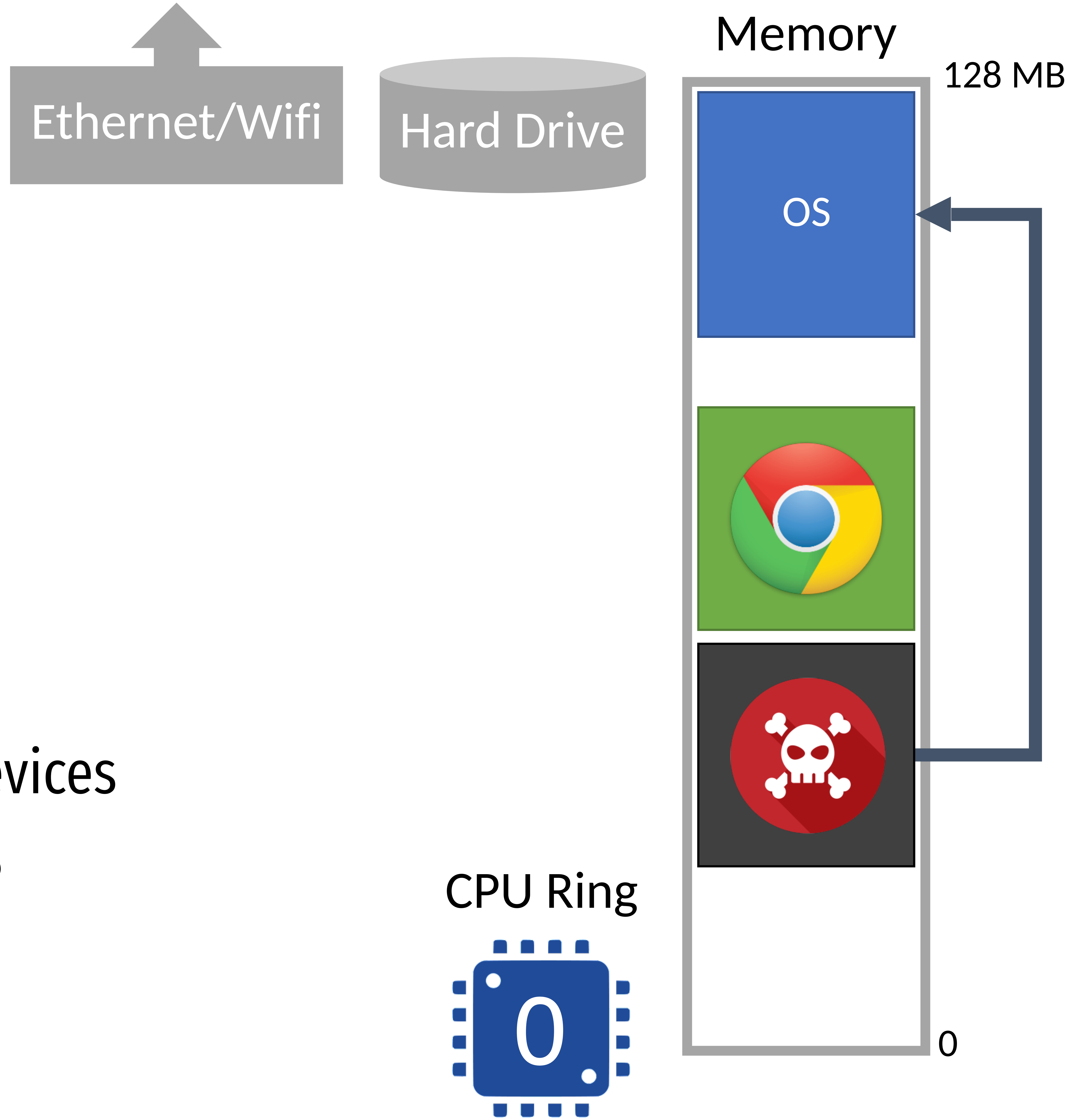


Protected mode stops direct access to devices

All device access must go through the OS

OS will impose access control checks

Protection in Action

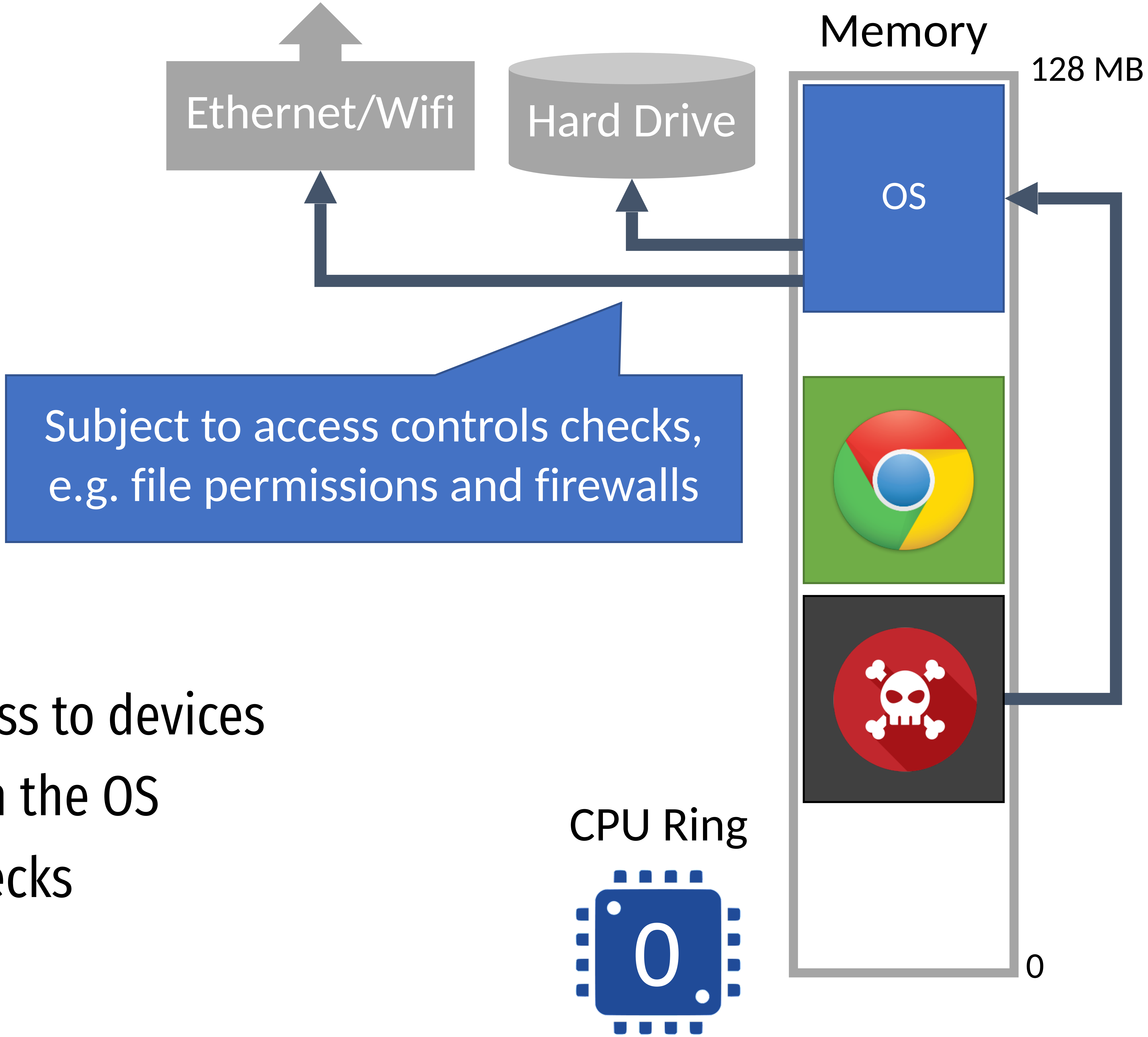


Protected mode stops direct access to devices

All device access must go through the OS

OS will impose access control checks

Protection in Action



Protected mode stops direct access to devices

All device access must go through the OS

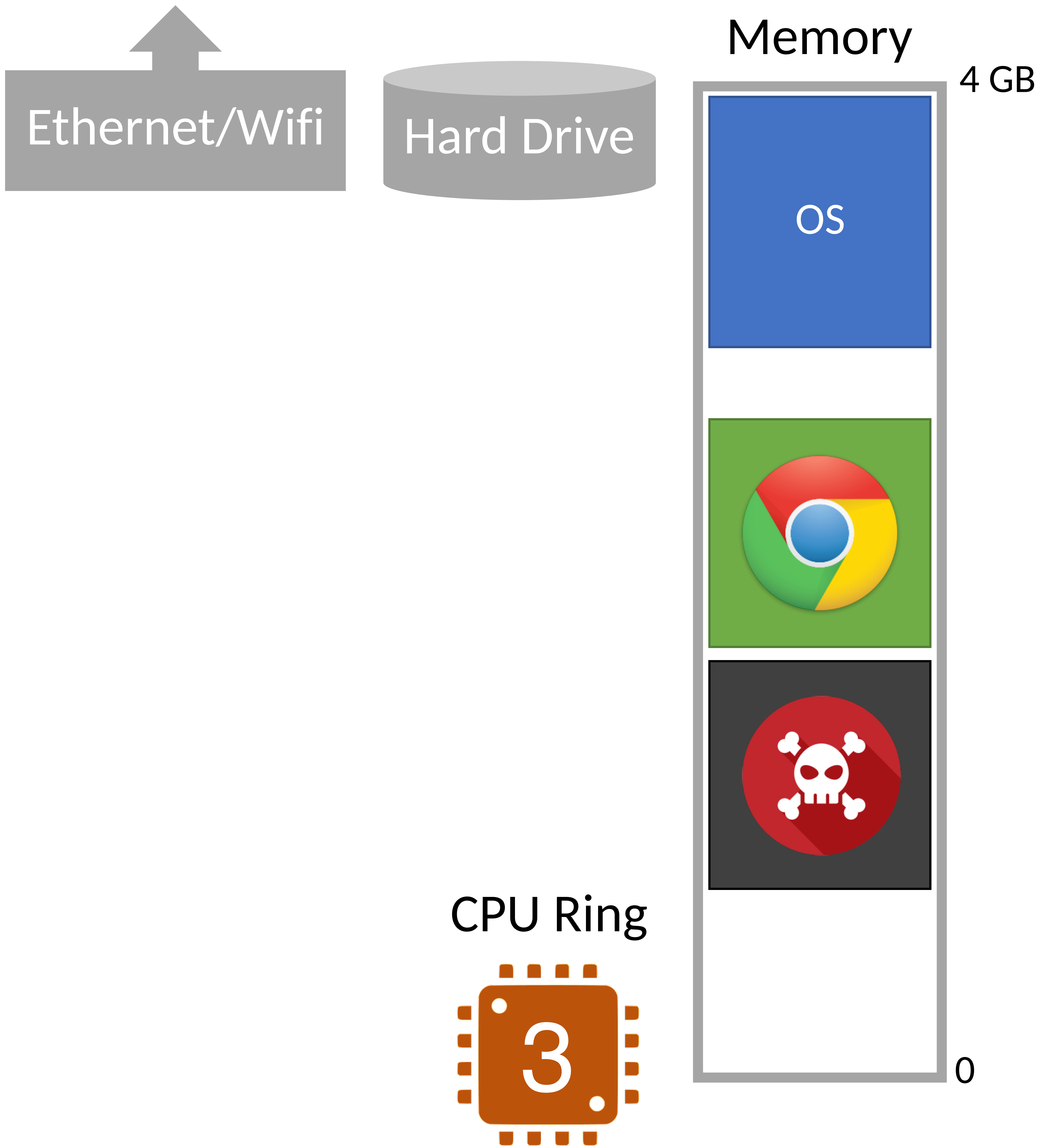
OS will impose access control checks

Virtual Memory

Status Check

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

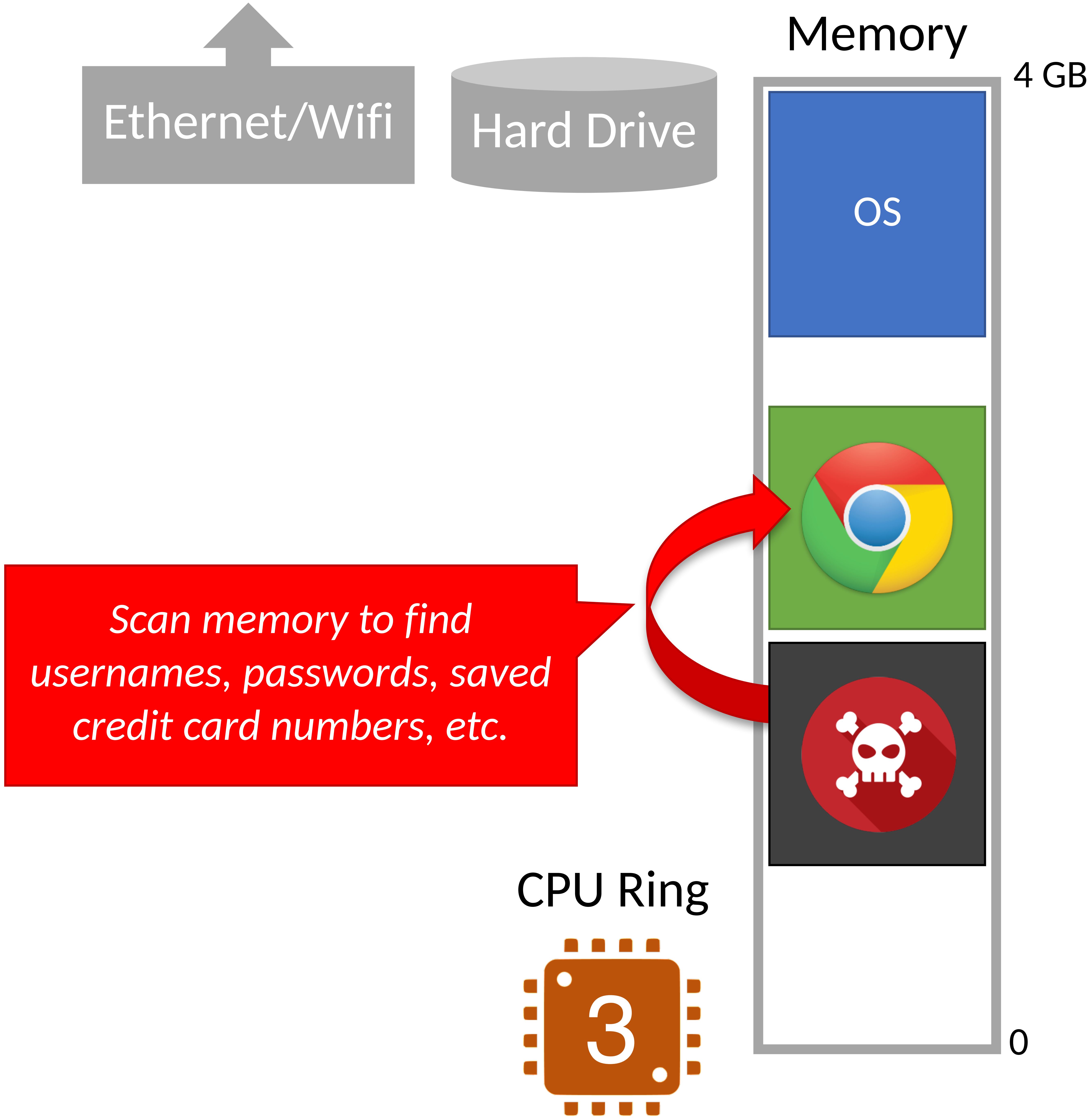
… But we have not protected memory



Status Check

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

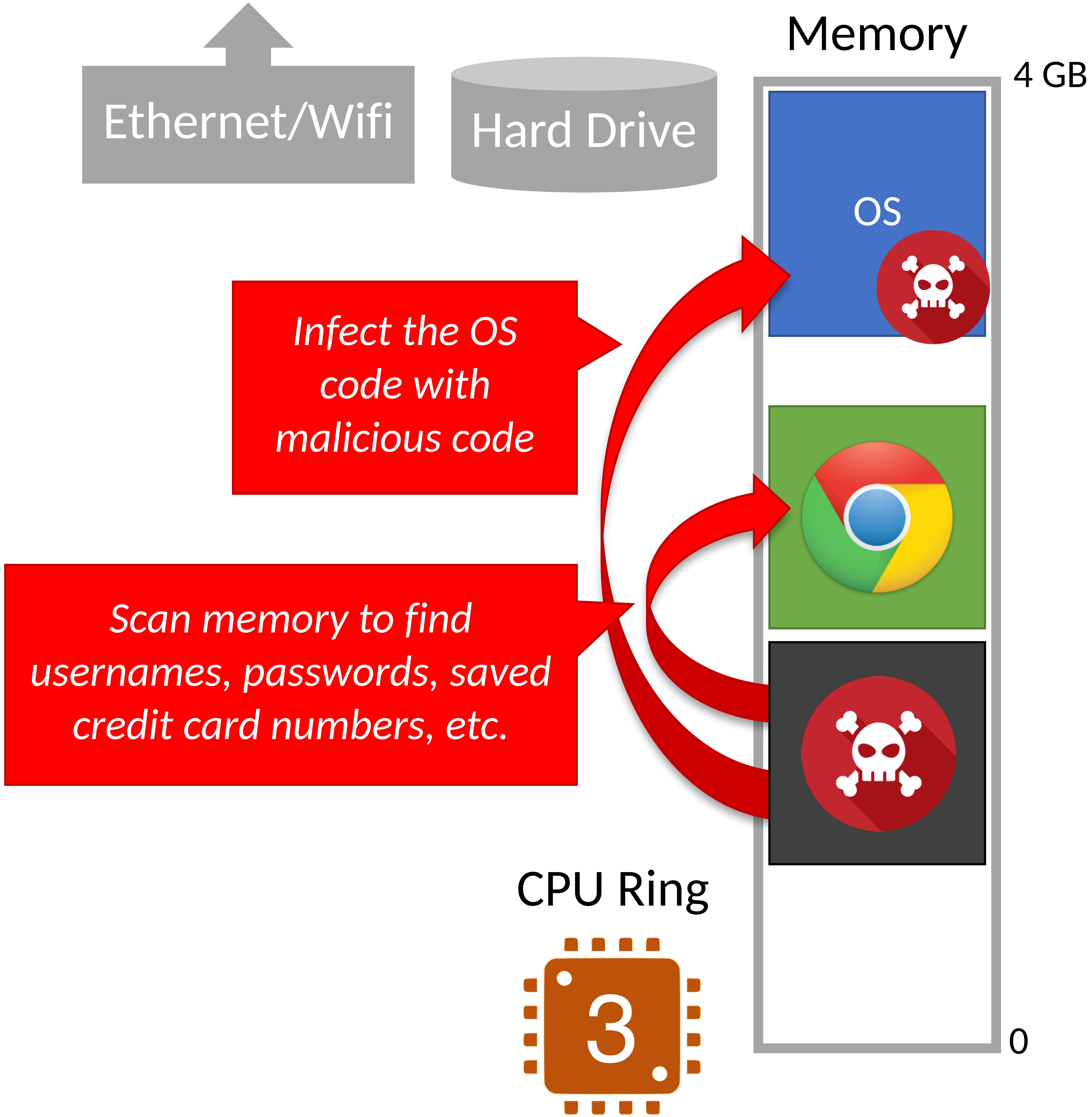
… But we have not protected memory



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

Memory Isolation and Virtual Memory

Modern CPUs support virtual memory

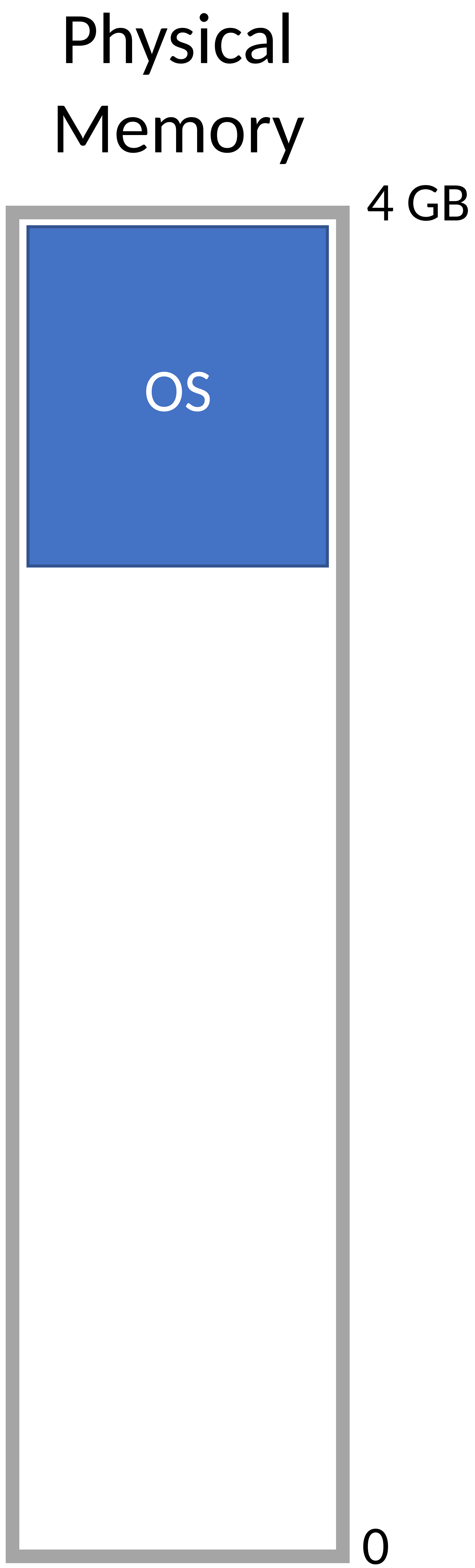
Creates the illusion that each process runs in its own, empty memory space

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

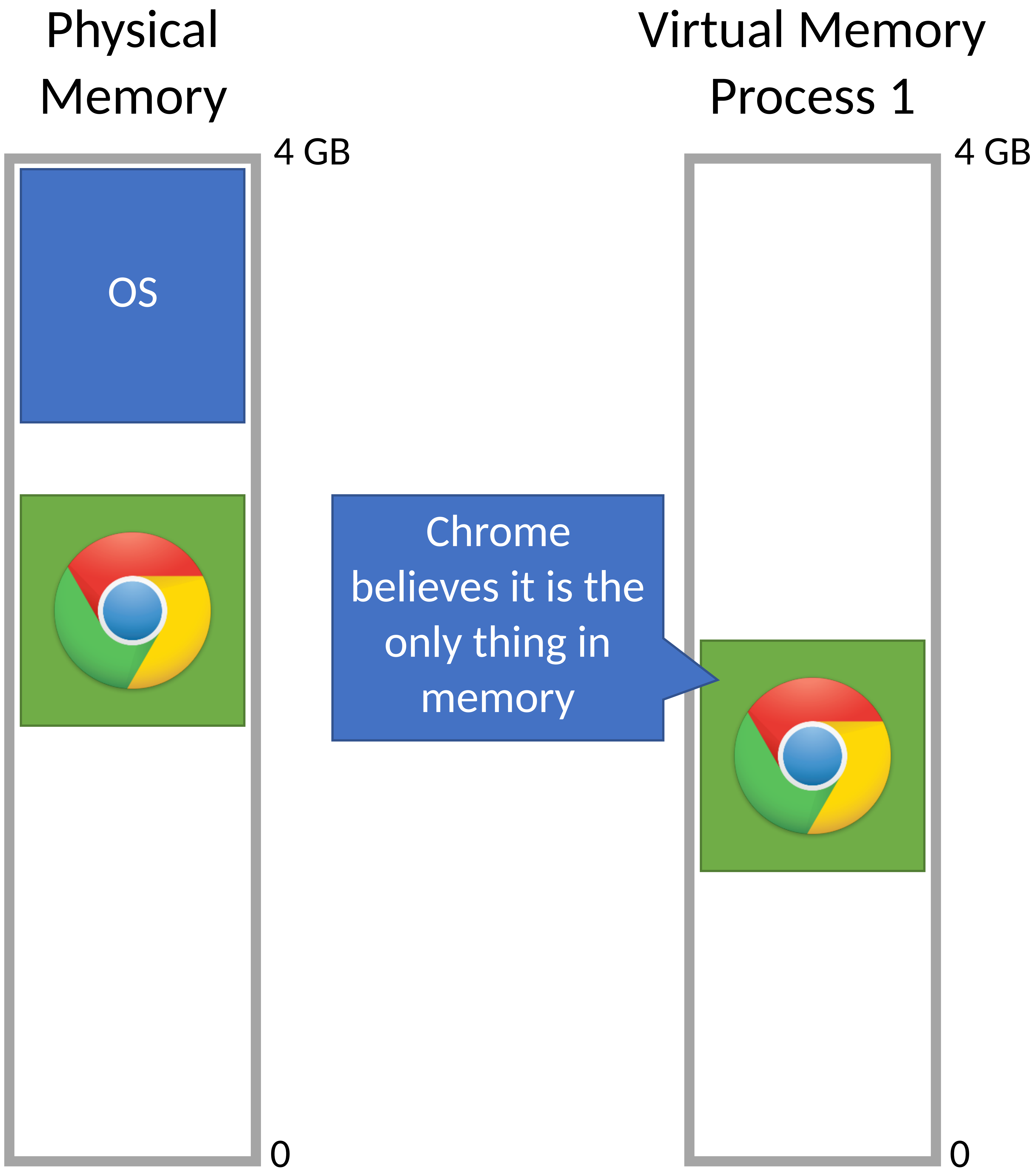
In later courses, you will learn how virtual memory is implemented

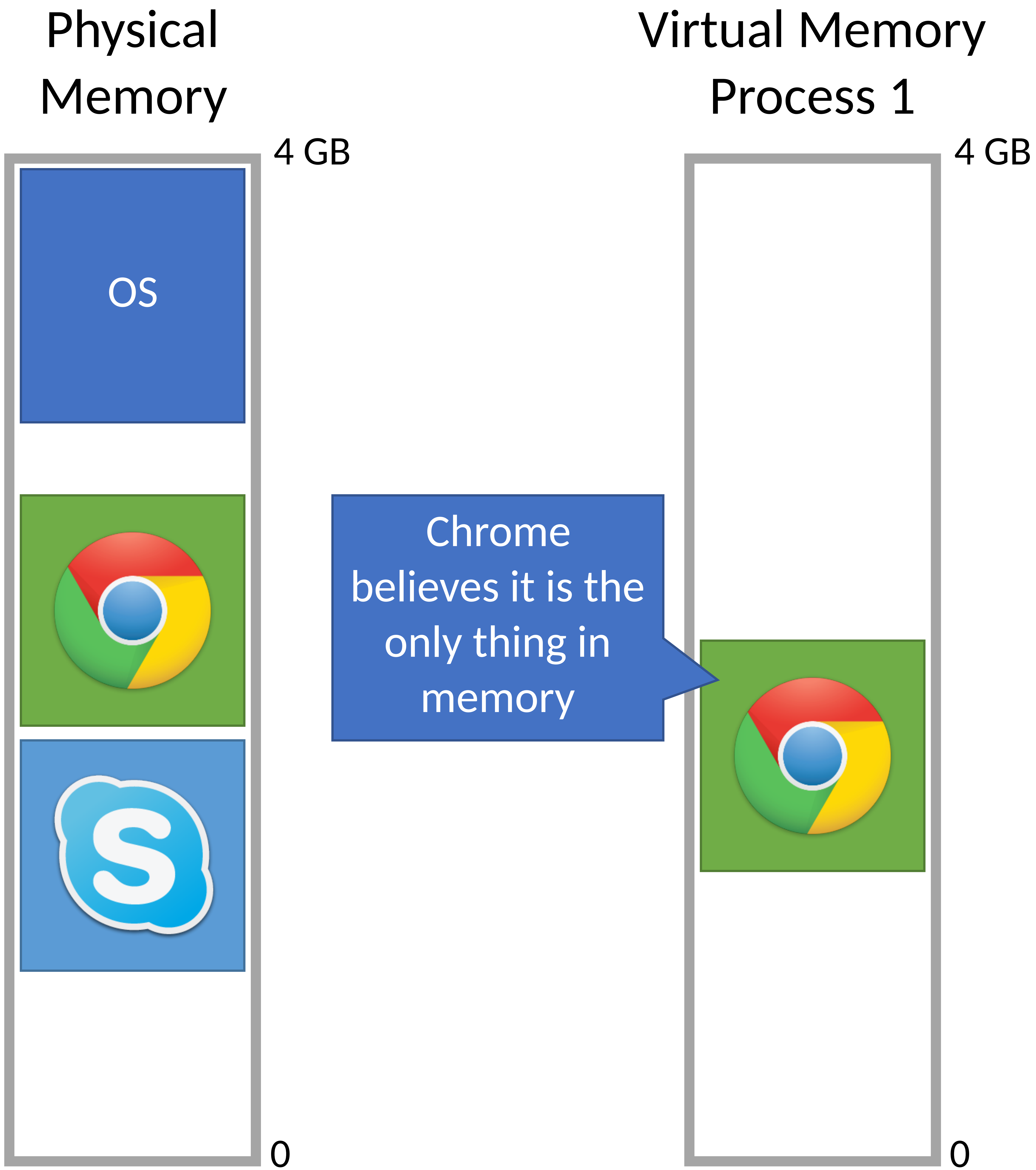
- Base and bound registers
- Segmentation
- Page tables

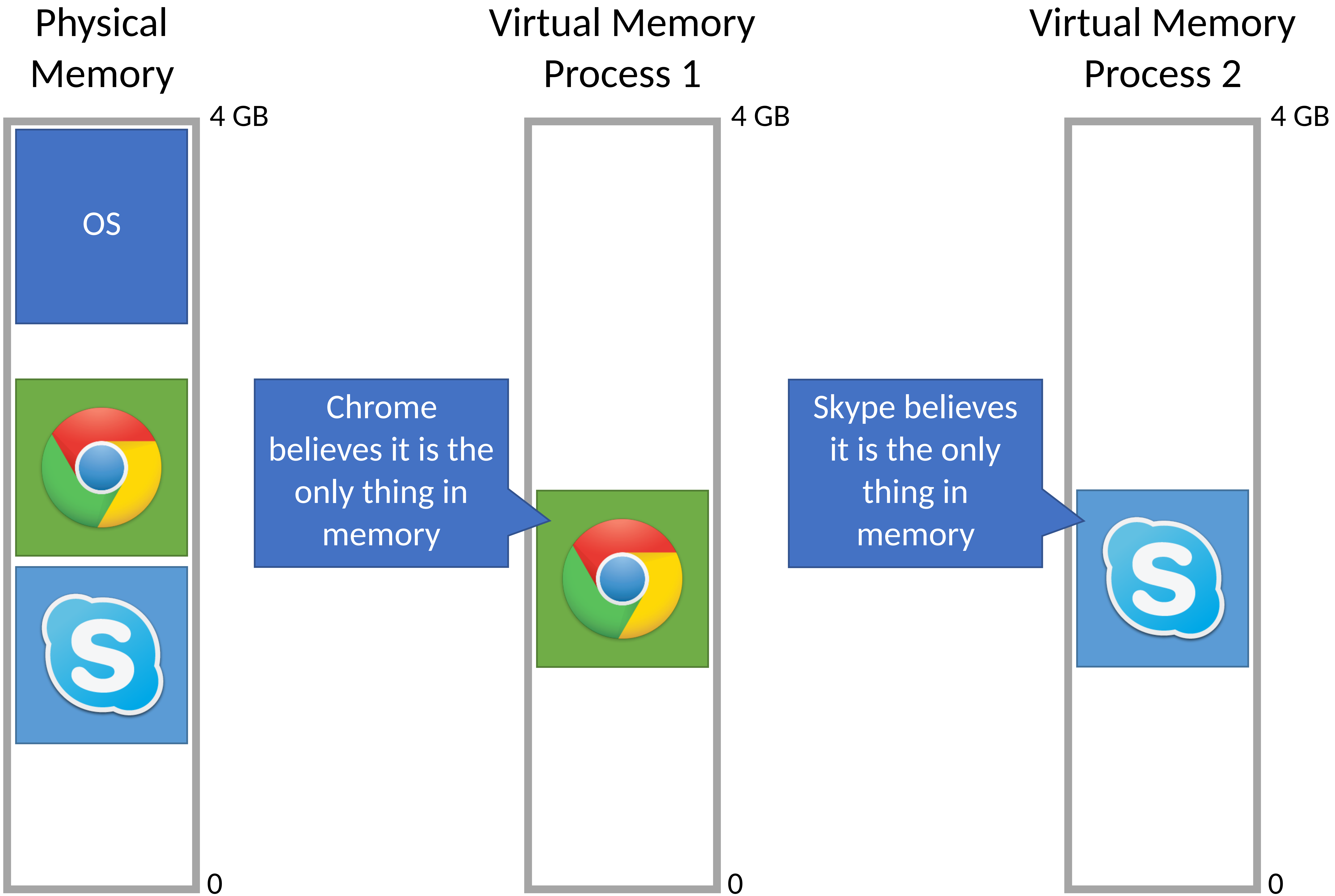
Today, we will do the cliffnotes version…

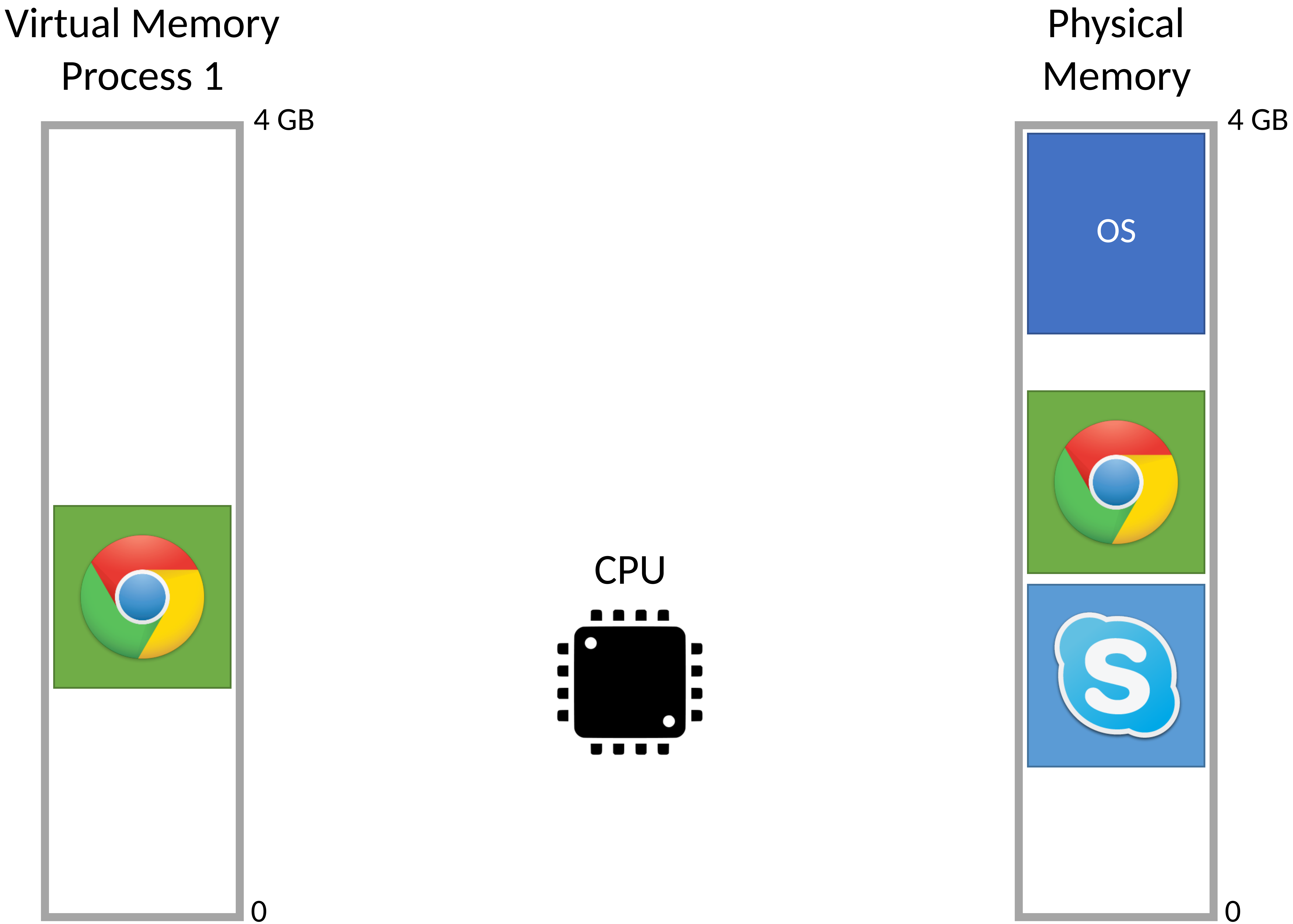

Physical
Memory
4 GB
OS
0

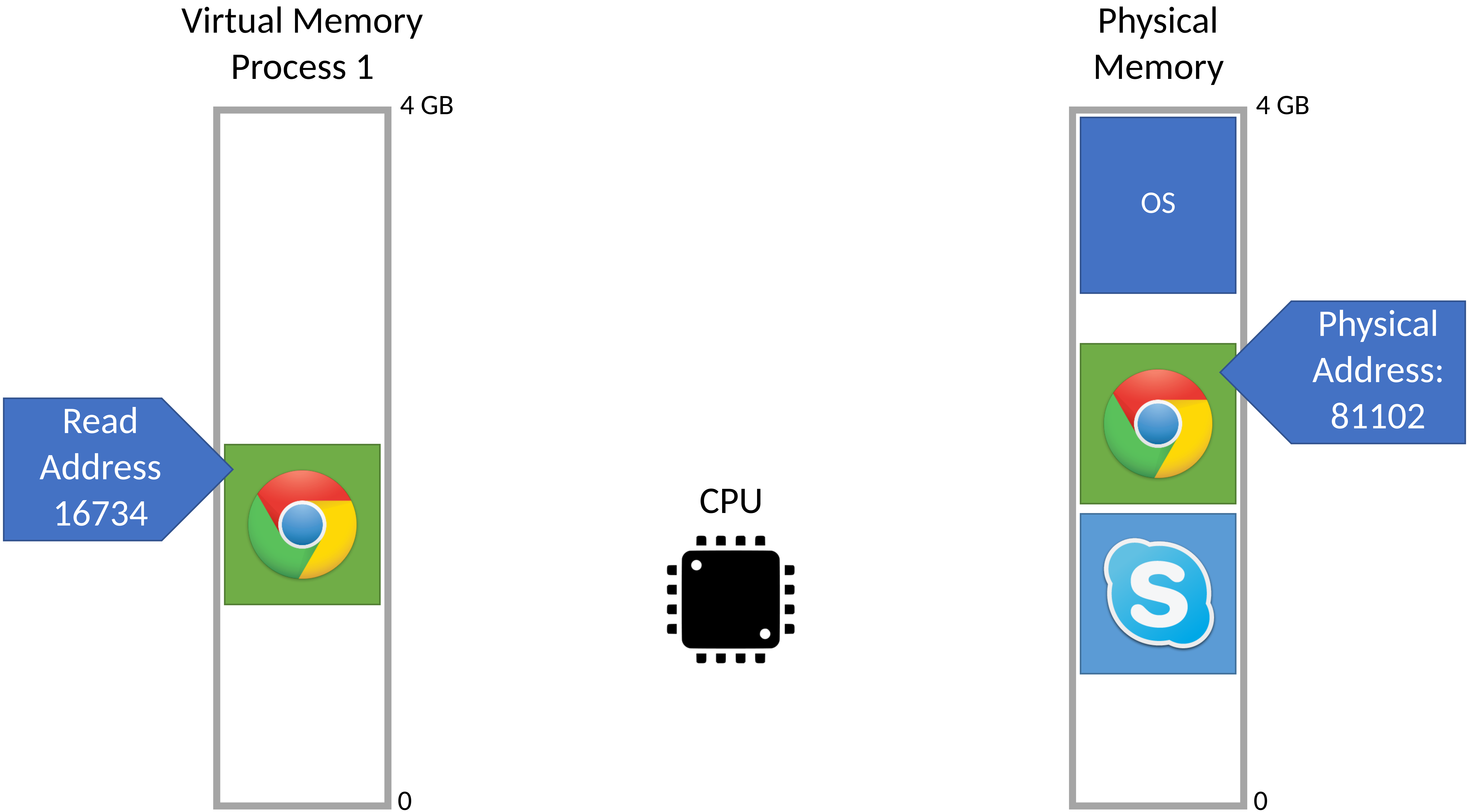

Physical
Memory
4 GB
OS
0

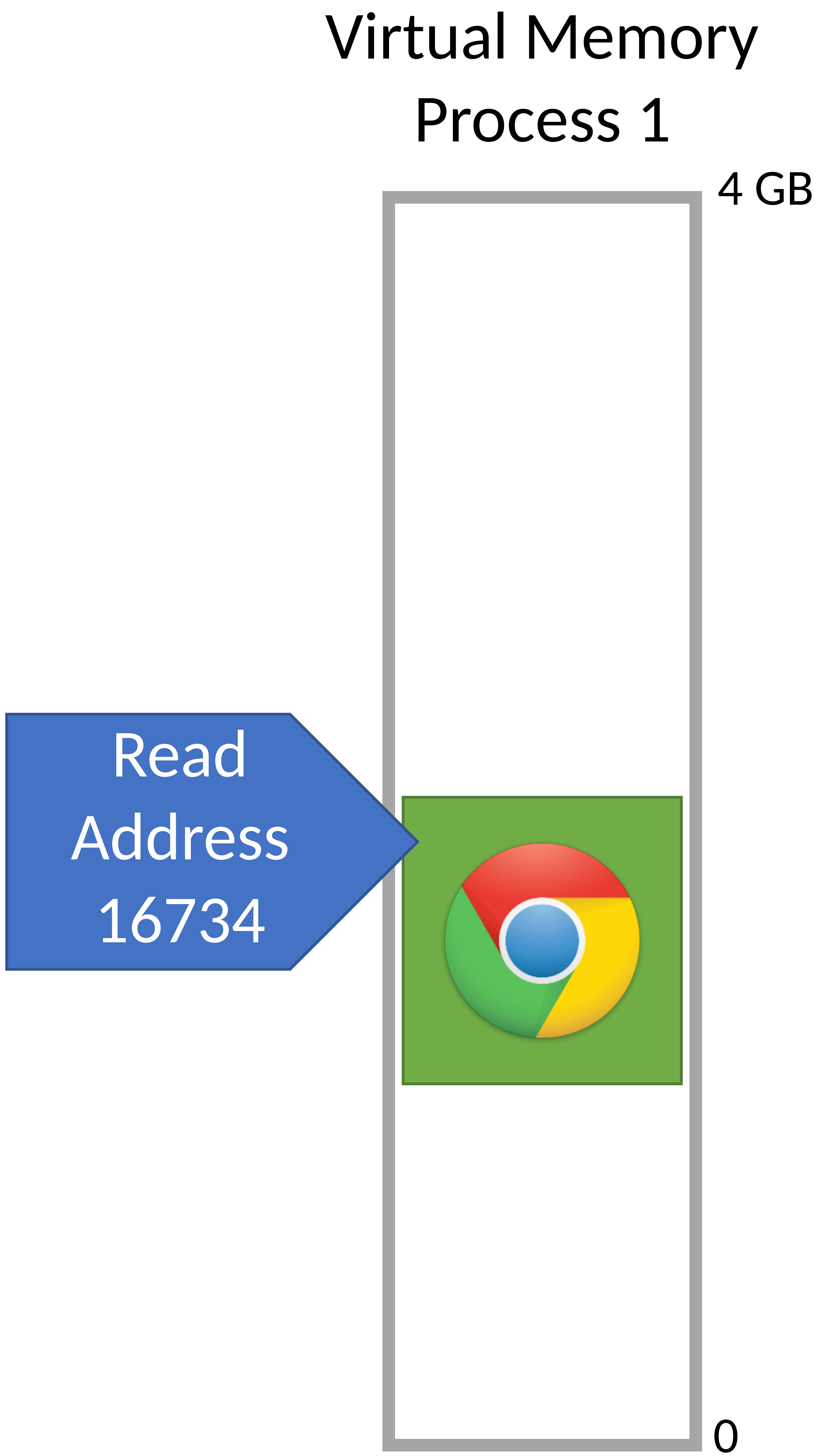

Physical
Memory
4 GB
OS
0
Chrome believes it is the only thing in memory
Virtual Memory
Process 1
4 GB
0


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


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

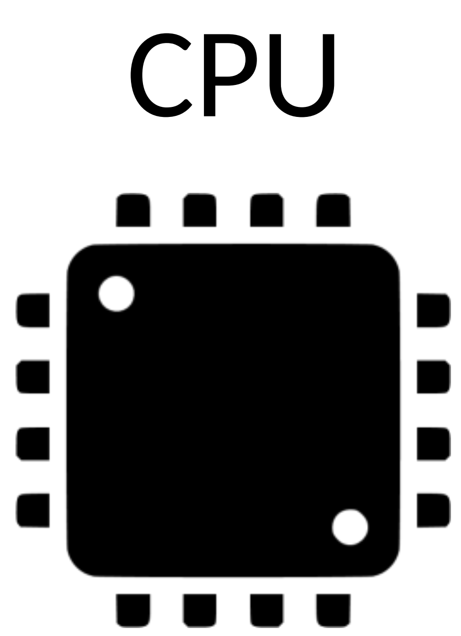

Virtual Memory
Process 1
4 GB
0
CPU
Physical
Memory
4 GB
OS
0

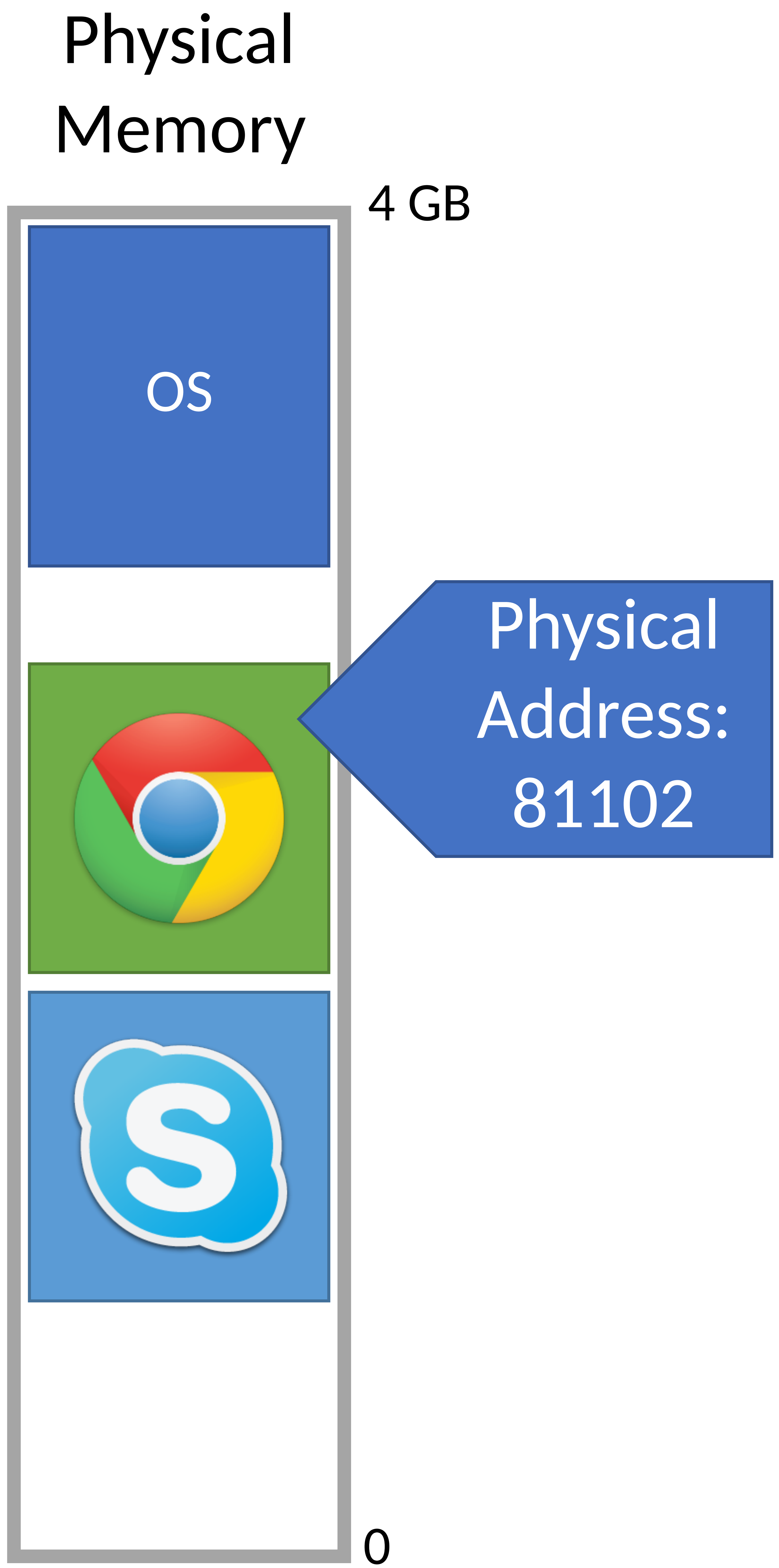

Virtual Memory
Process 1
4 GB
0
Read
Address
16734
CPU
Physical
Memory
4 GB
OS
Physical
Address:
81102
0



Page Table

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






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
0
Physical Address: 81102

Virtual Memory Implementation

Each process has its own virtual memory space

- Each process has a page table that maps is virtual space into physical space
- CPU translates virtual address to physical addresses on-the-fly

Virtual Memory Implementation

Each process has its own virtual memory space

- Each process has a page table that maps is virtual space into physical space
- CPU translates virtual address to physical addresses on-the-fly

OS creates the page table for each process

- Installing page tables in the CPU is a protected, Ring 0 instruction
- Processes cannot modify their page tables

Virtual Memory Implementation

Each process has its own virtual memory space

- Each process has a page table that maps is virtual space into physical space
- CPU translates virtual address to physical addresses on-the-fly

OS creates the page table for each process

- Installing page tables in the CPU is a protected, Ring 0 instruction
- Processes cannot modify their page tables

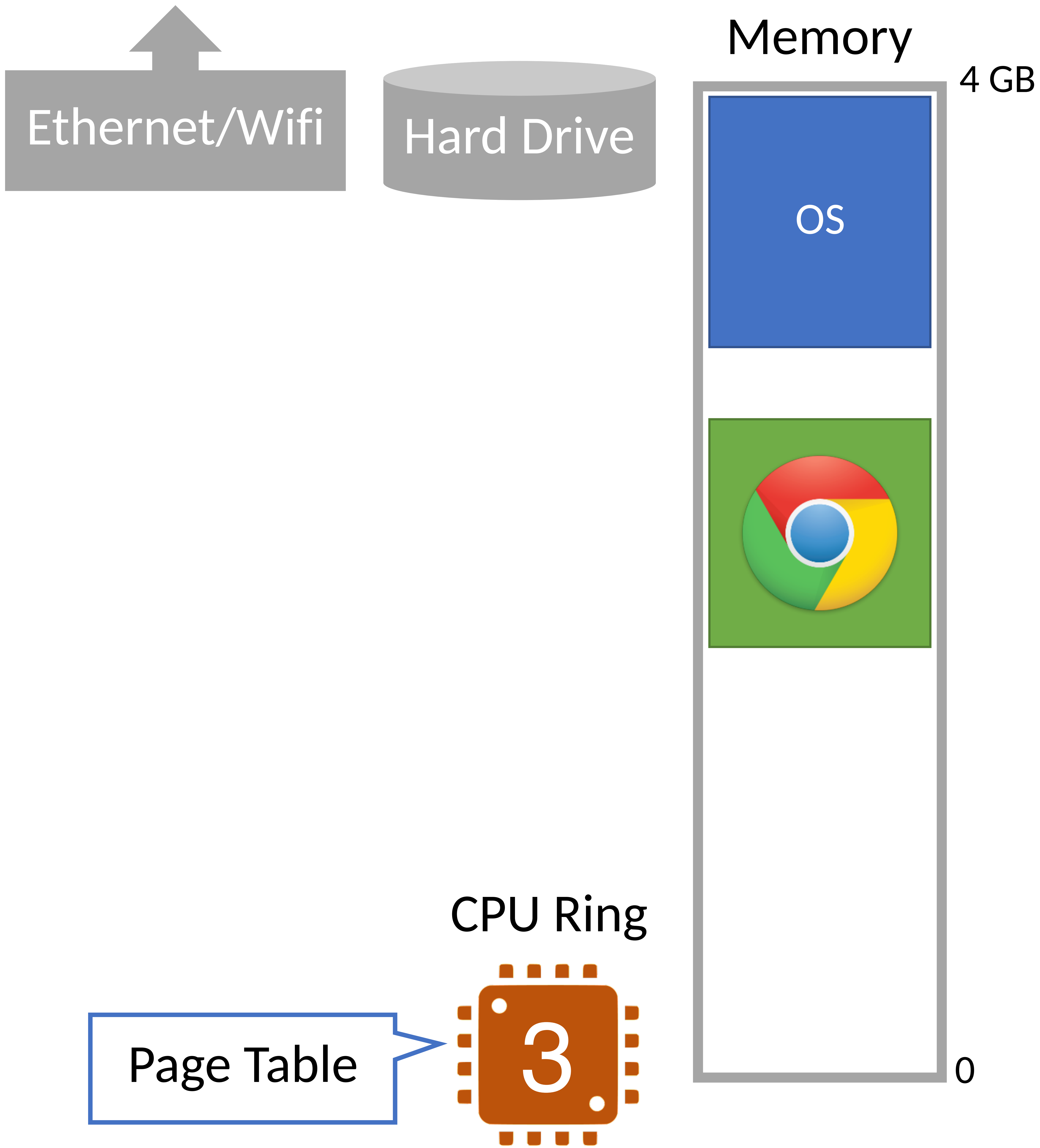
What happens if a process tries to read/write memory outside its page table?

- Segmentation Fault or Page Fault
- Process crashes
- In other words, no way to escape virtual memory

VM in Action

Processes can only read/ write within their own virtual memory

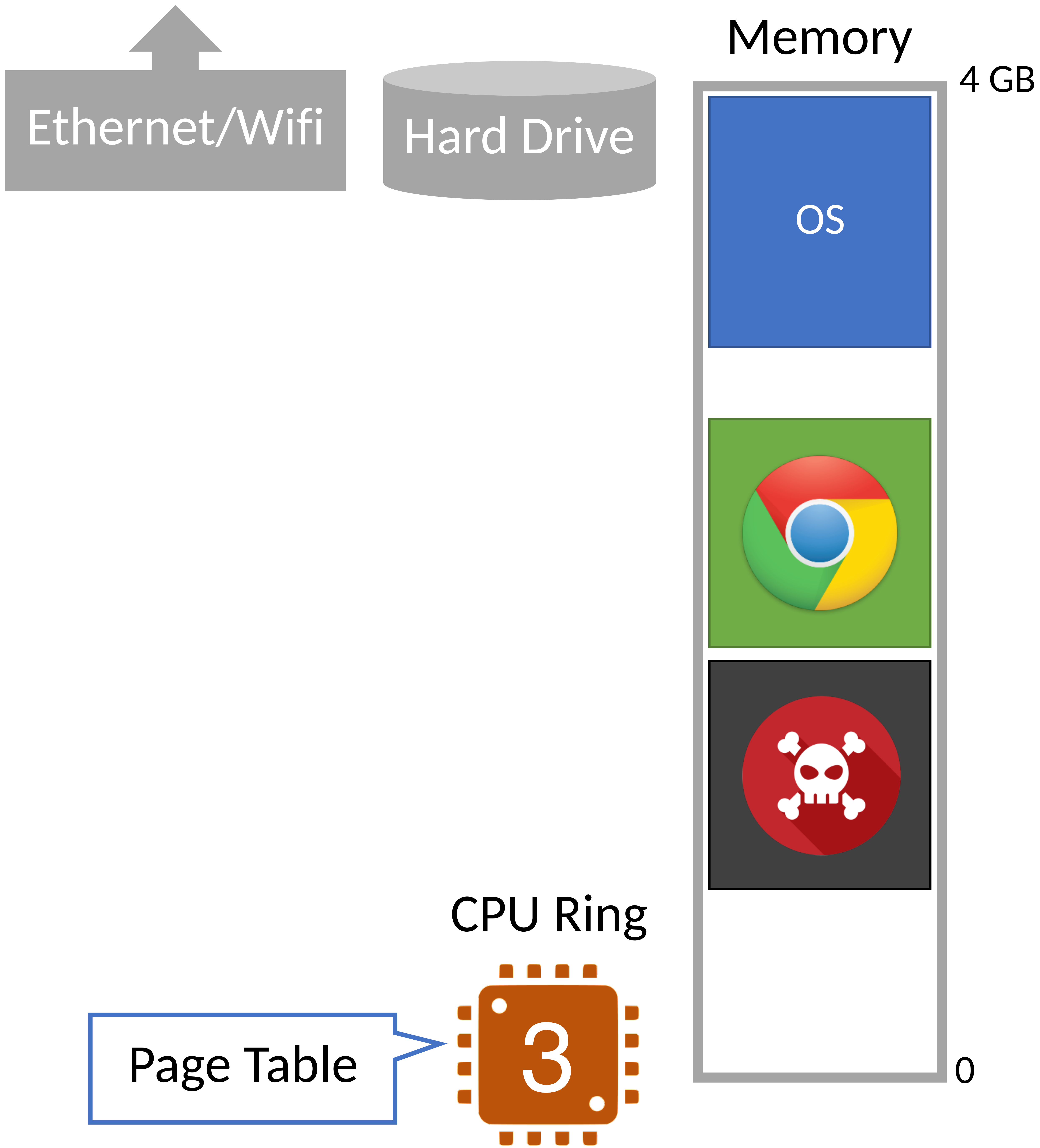
Processes cannot change their own page tables



VM in Action

Processes can only read/ write within their own virtual memory

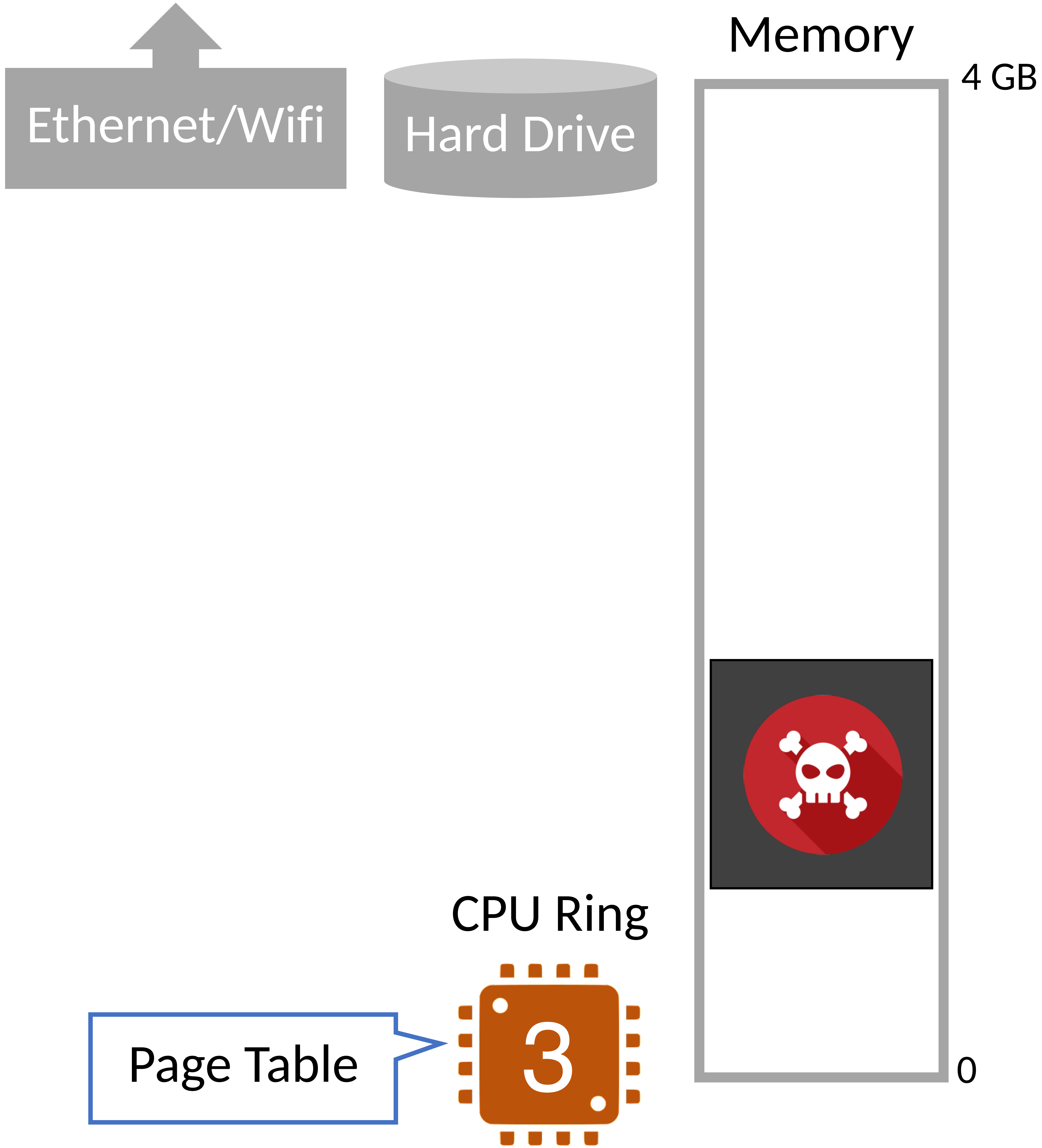
Processes cannot change their own page tables



VM in Action

Processes can only read/ write within their own virtual memory

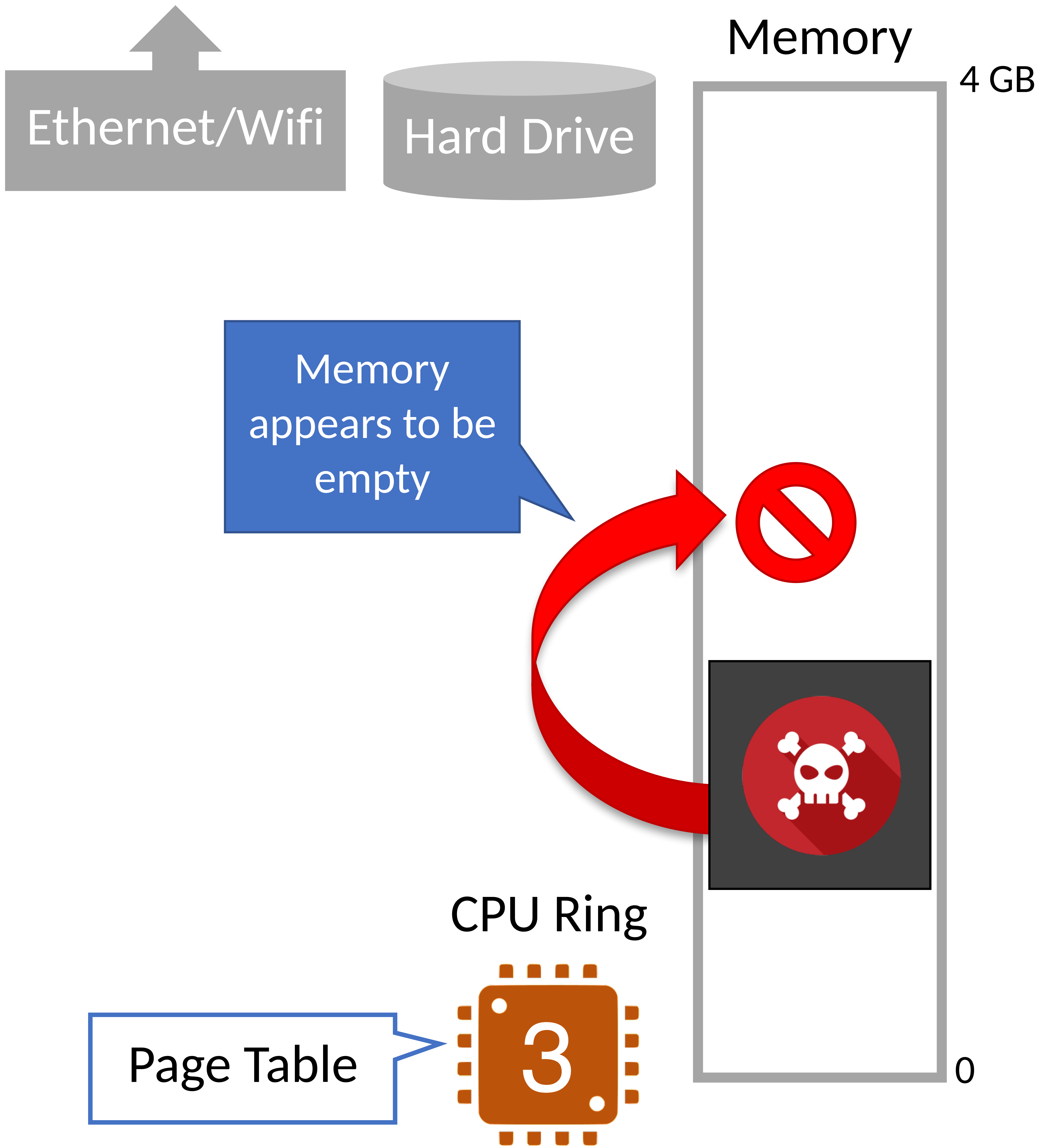
Processes cannot change their own page tables



VM in Action

Processes can only read/ write within their own virtual memory

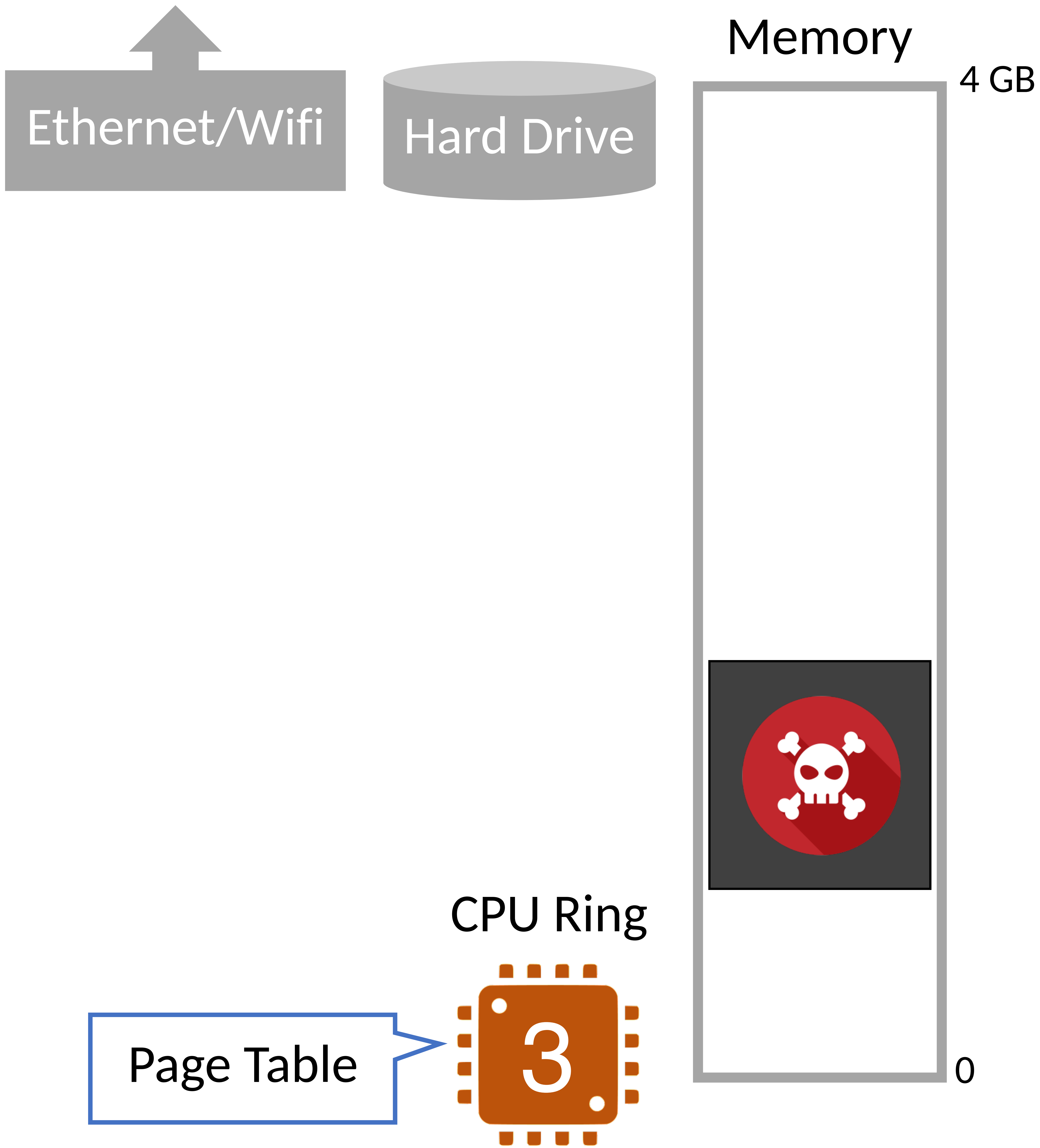
Processes cannot change their own page tables



VM in Action

Processes can only read/ write within their own virtual memory

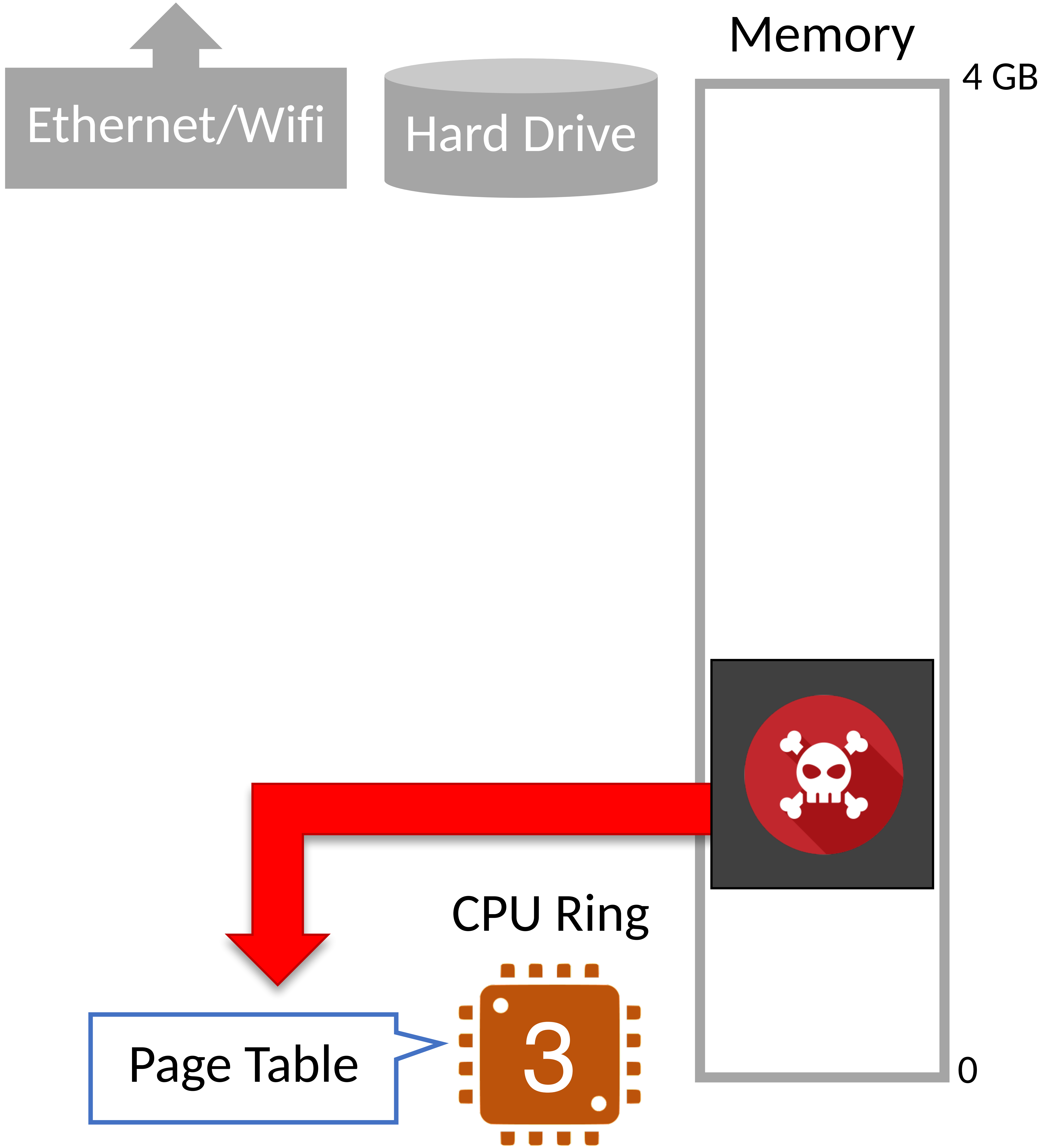
Processes cannot change their own page tables



VM in Action

Processes can only read/ write within their own virtual memory

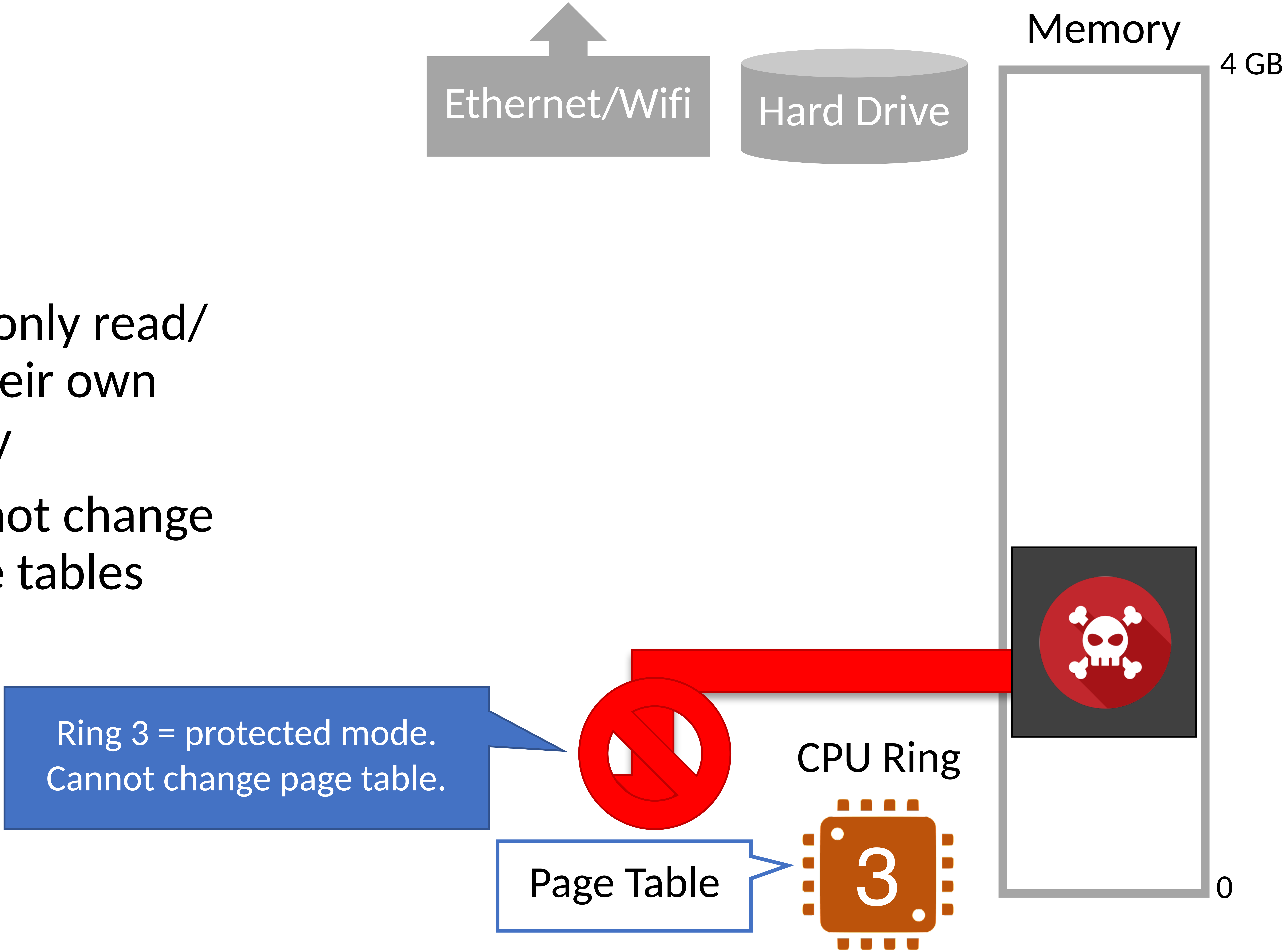
Processes cannot change their own page tables



VM in Action

Processes can only read/ write within their own virtual memory

Processes cannot change their own page tables



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

Review

At this point, we have achieved process isolation

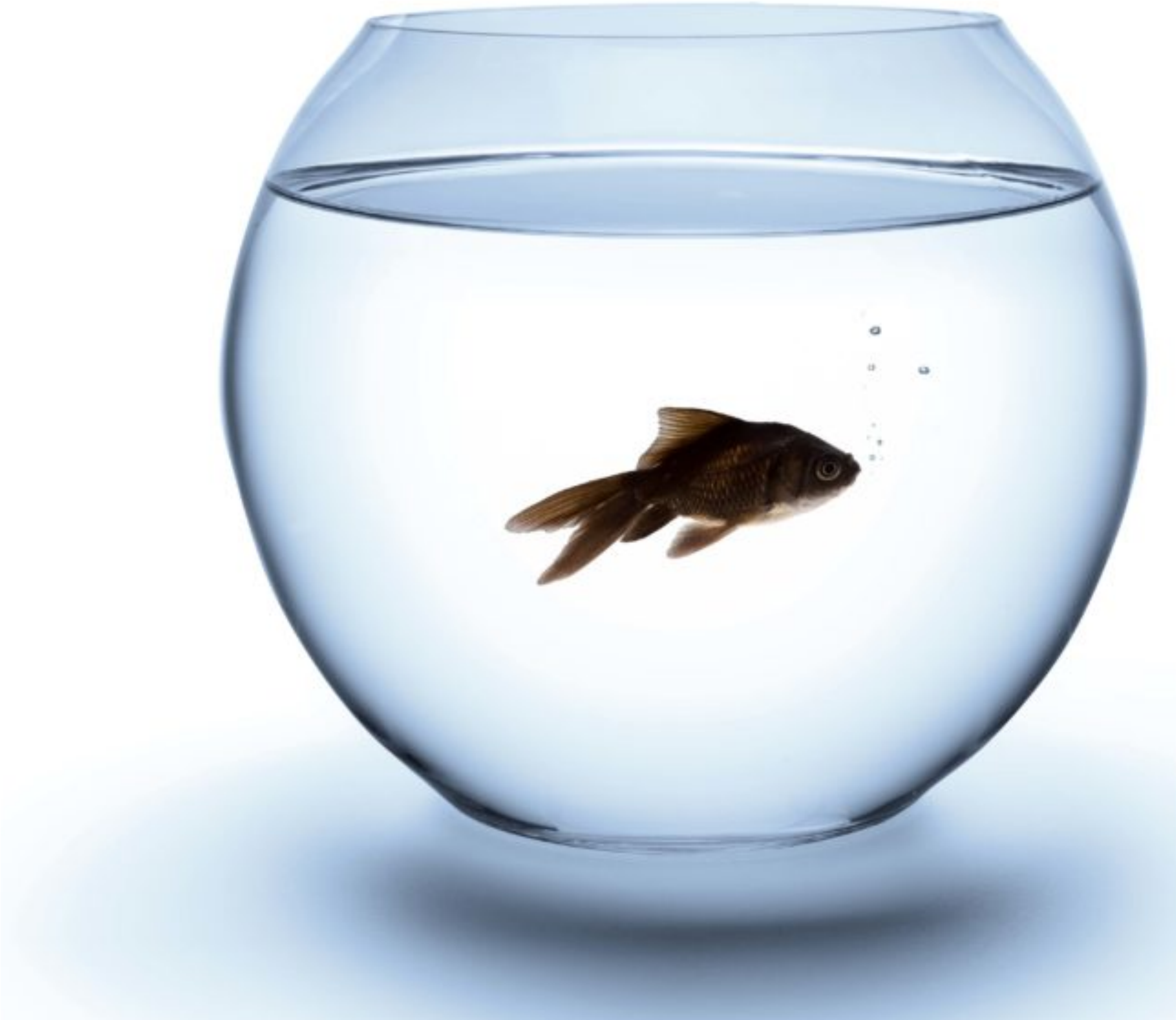
- Protected mode execution prevents direct device access
- Virtual memory prevents direct memory access

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

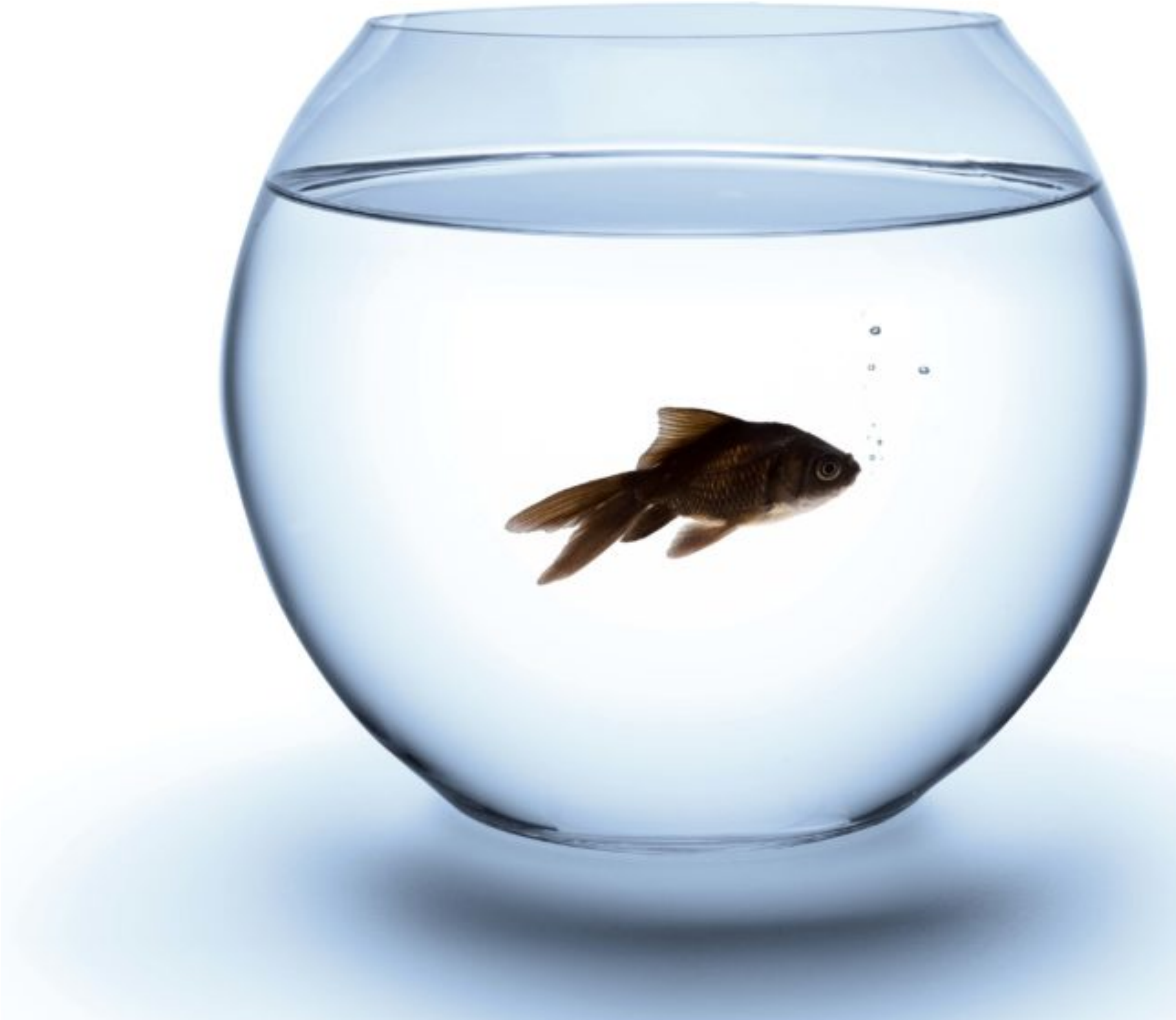
Requires CPU support

- 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



Towards Secure Systems

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

File Access Control

Anti-virus

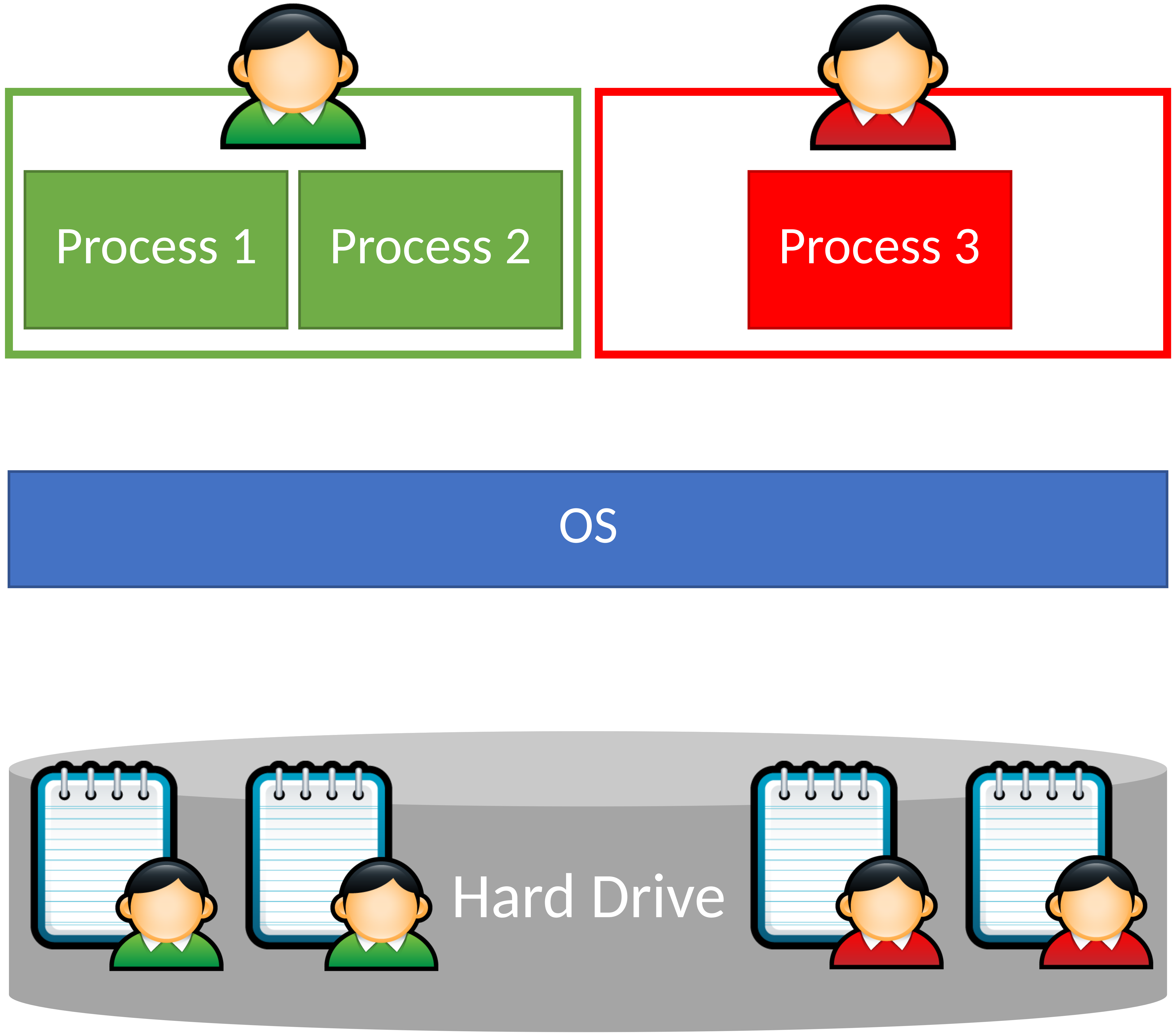
Firewall

Secure Logging

File Access Control

All disk access is mediated by the OS

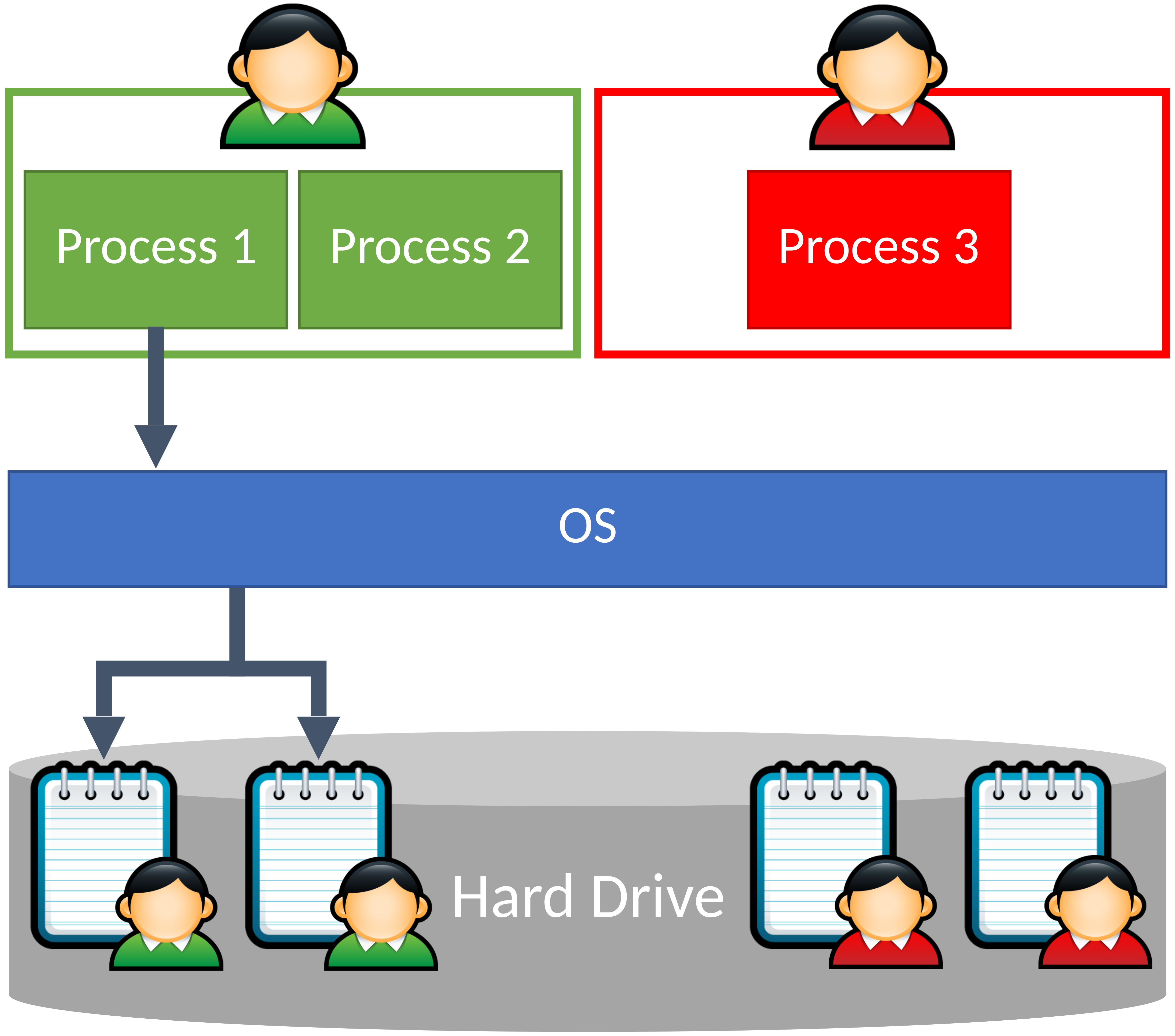
OS enforces access controls



File Access Control

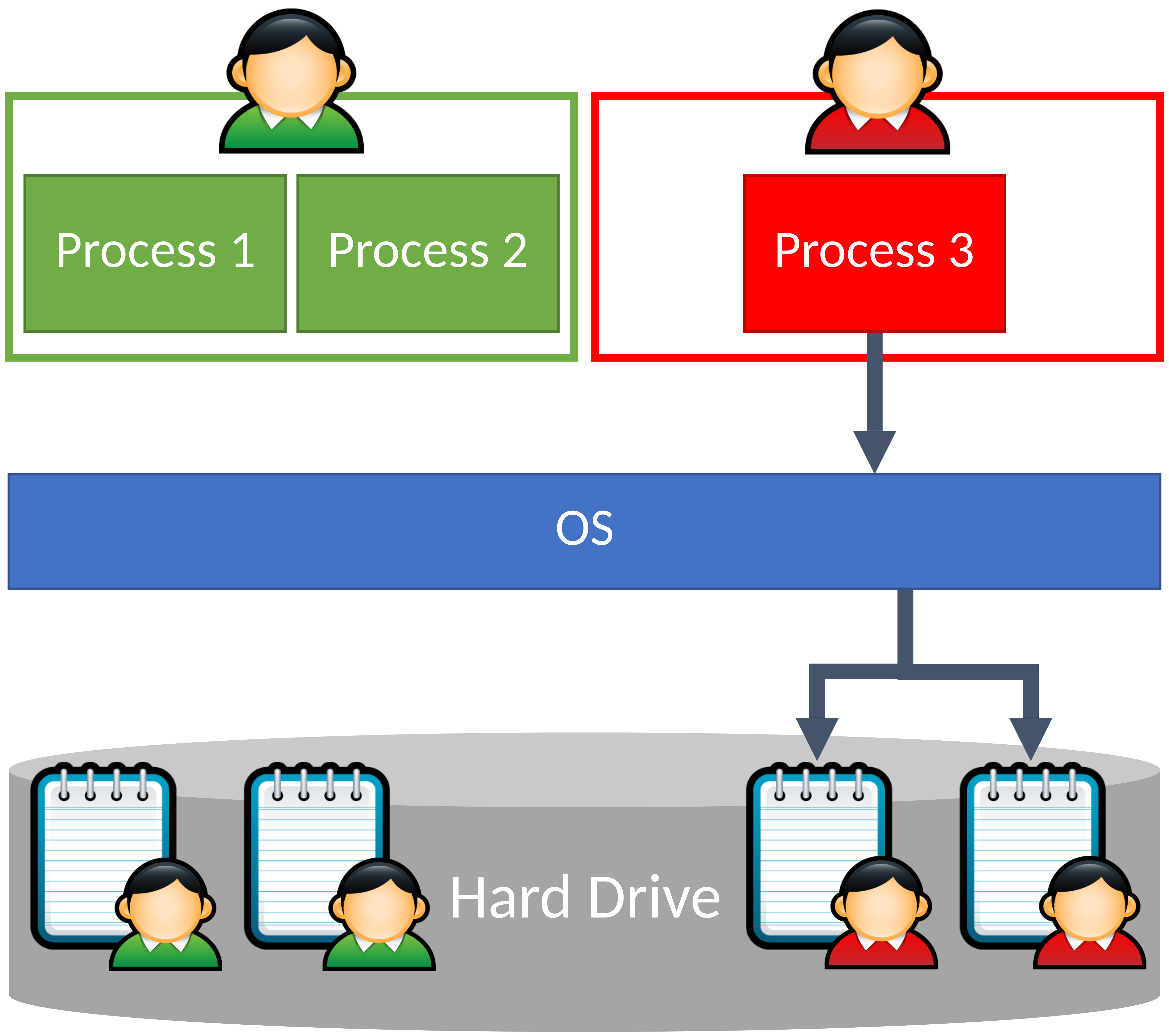
All disk access is mediated by the OS

OS enforces access controls



File Access Control

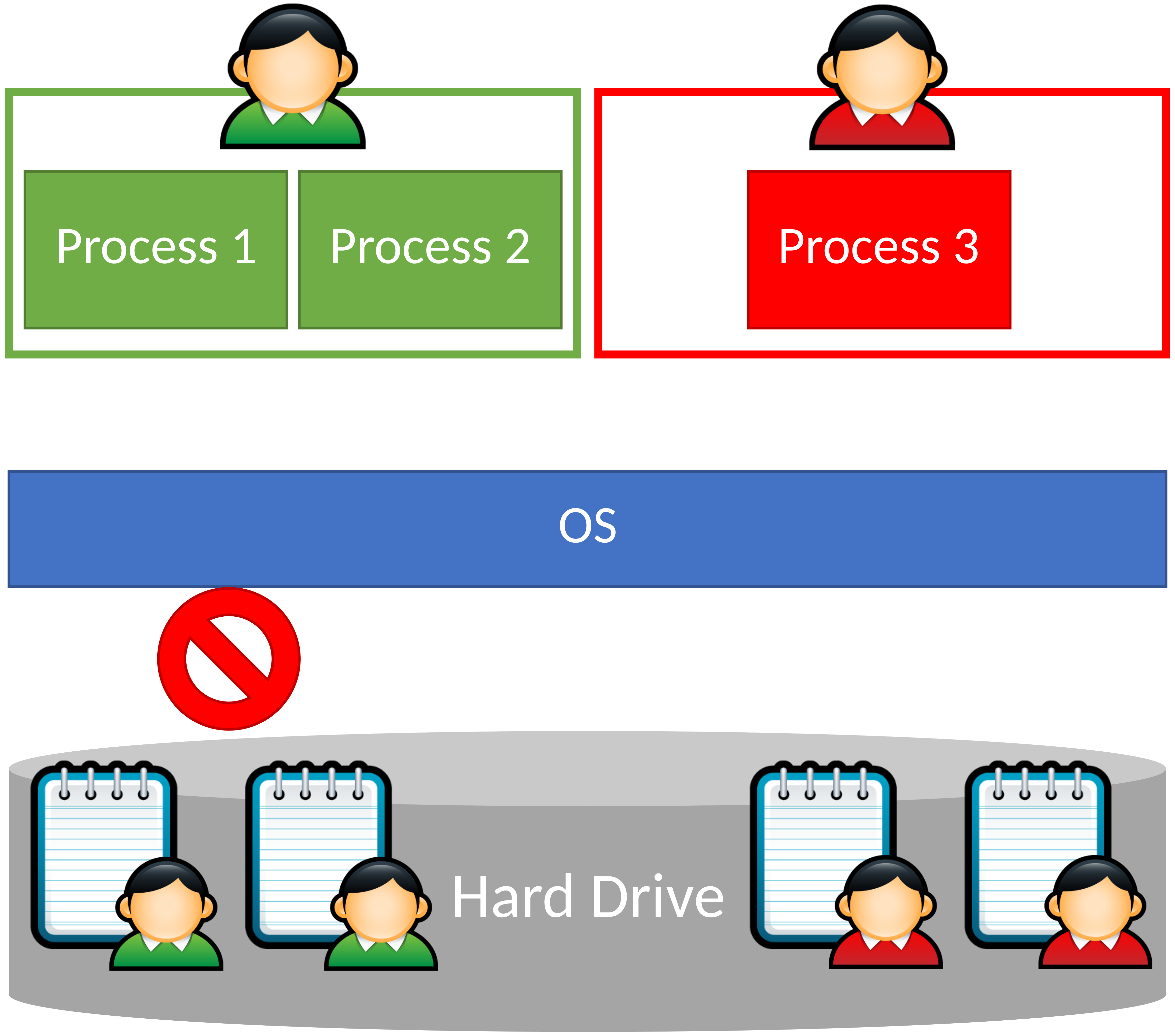
- All disk access is mediated by the OS
- OS enforces access controls



File Access Control

All disk access is mediated by the OS

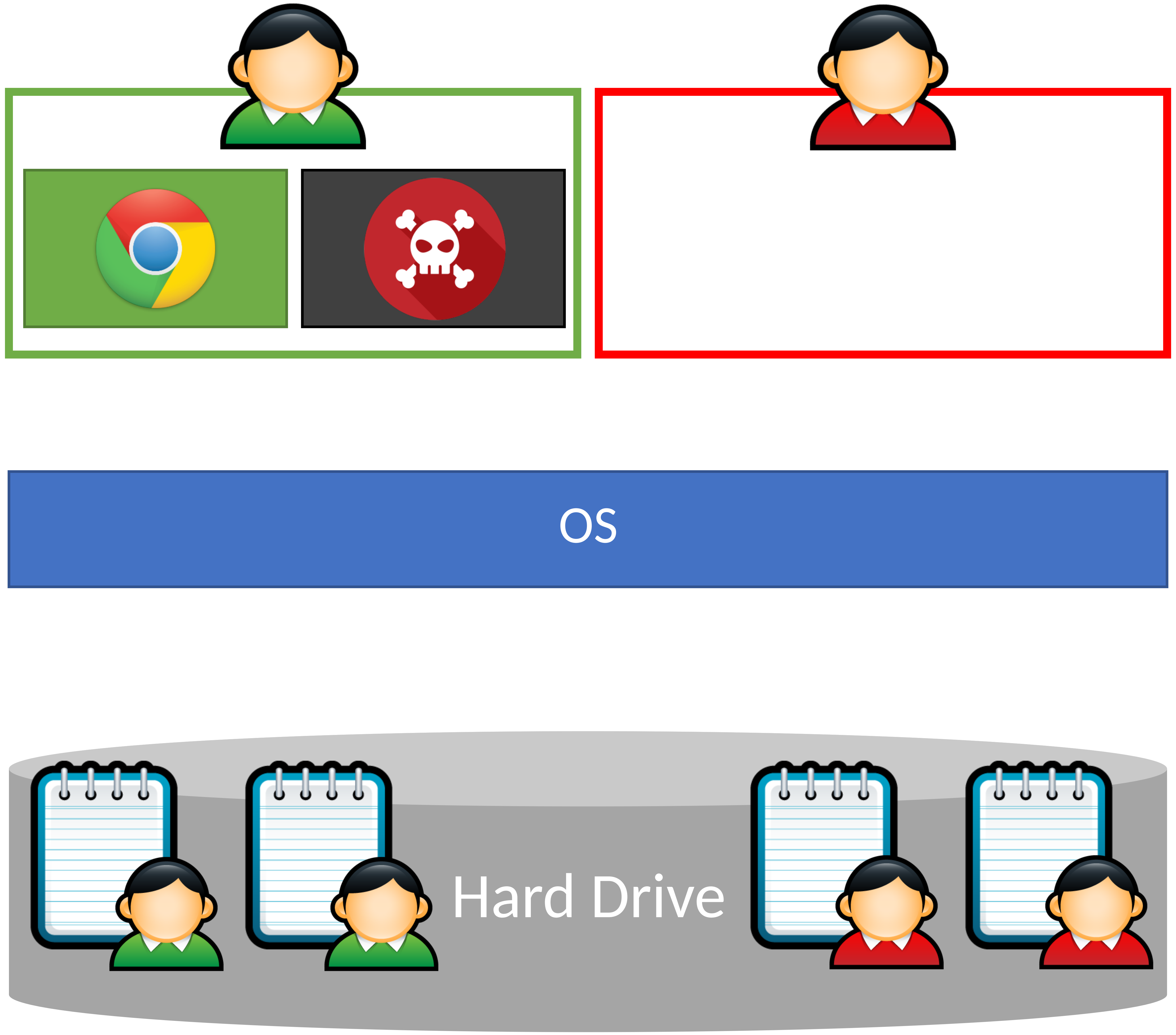
OS enforces access controls



Limitations

Malware can still cause damage

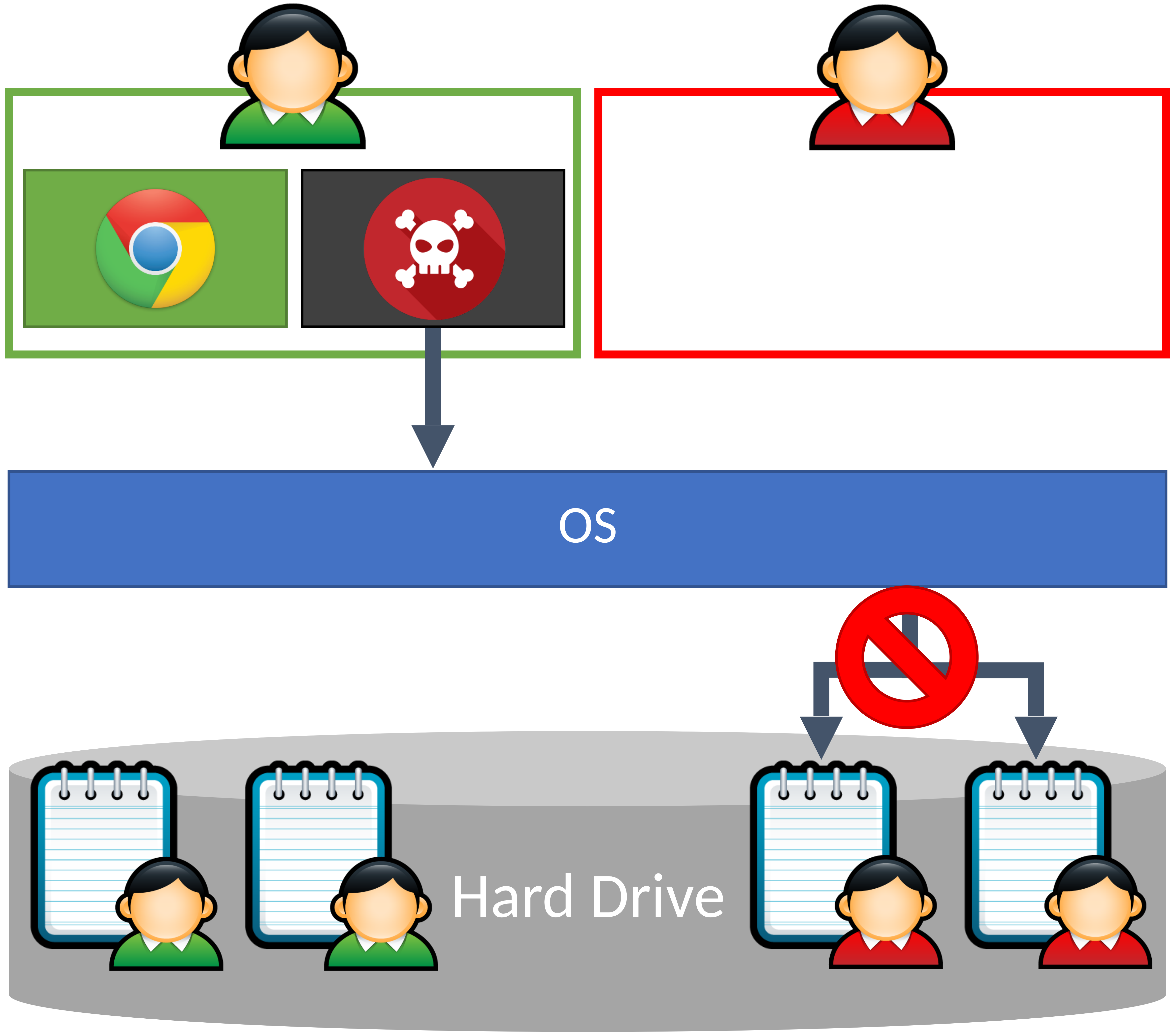
Discretionary access control means that isolation is incomplete



Limitations

Malware can still cause damage

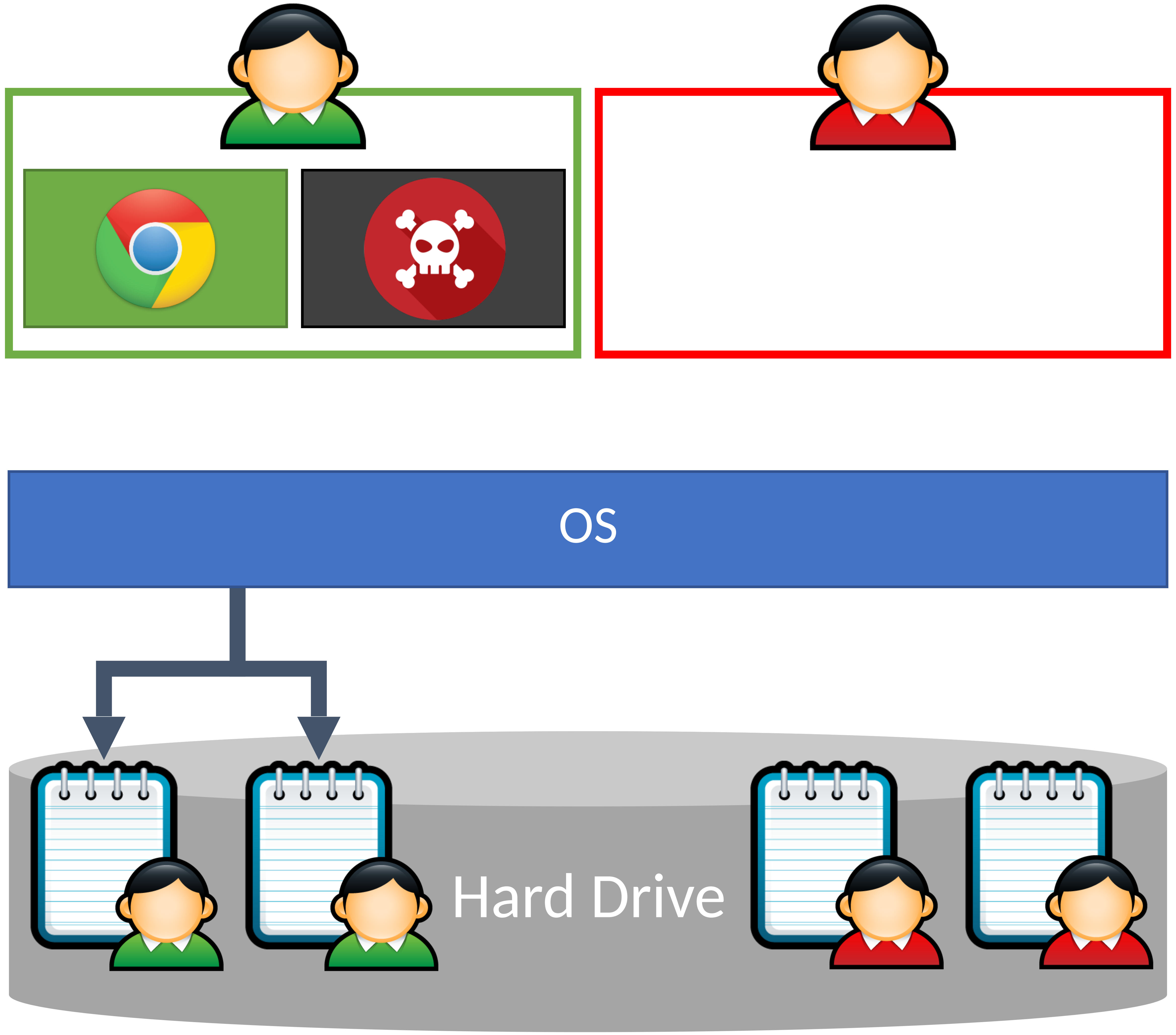
Discretionary access control means that isolation is incomplete



Limitations

Malware can still cause damage

Discretionary access control means that isolation is incomplete



Anti-virus

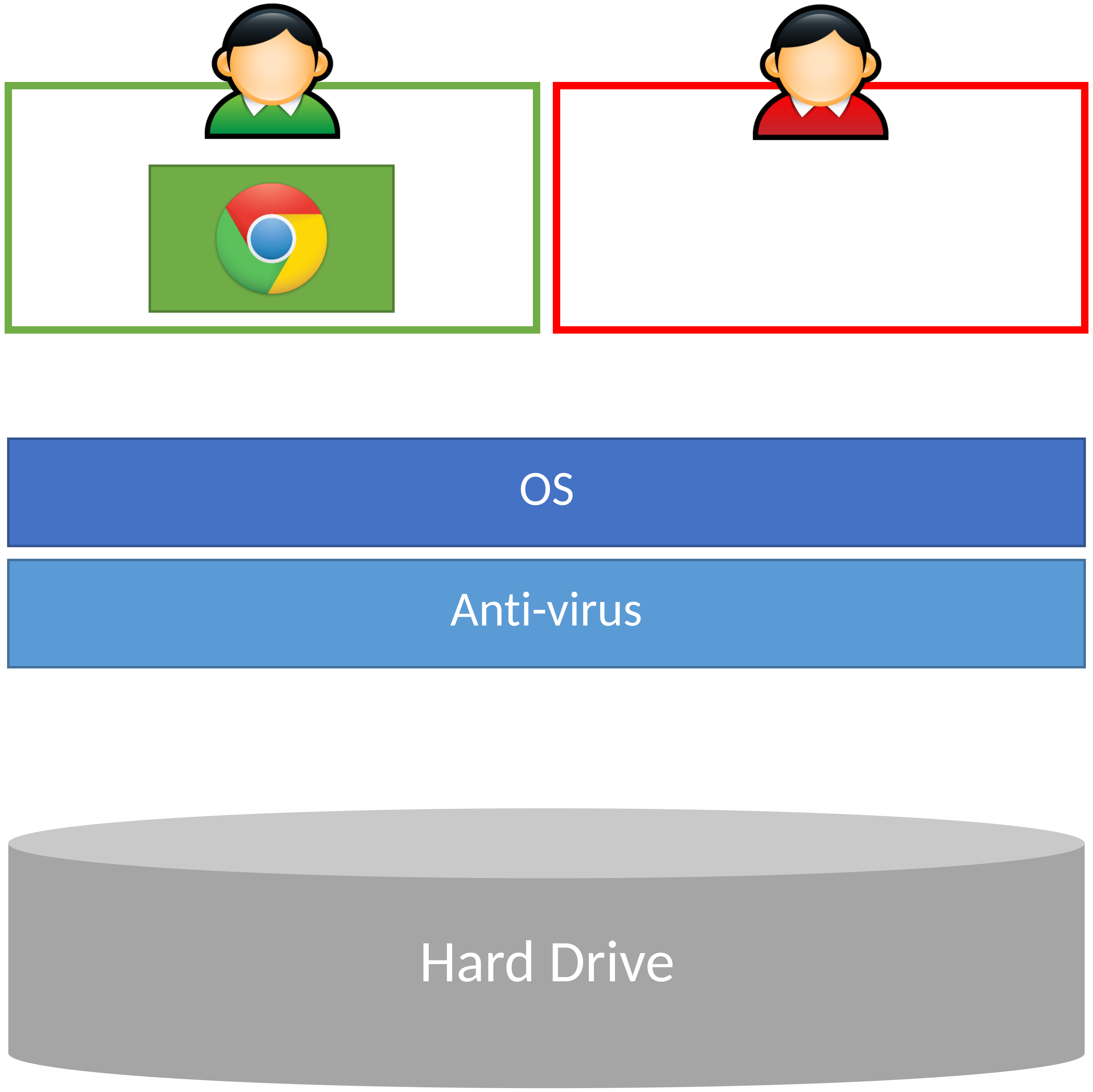
Anti-virus process is privileged

- Often 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

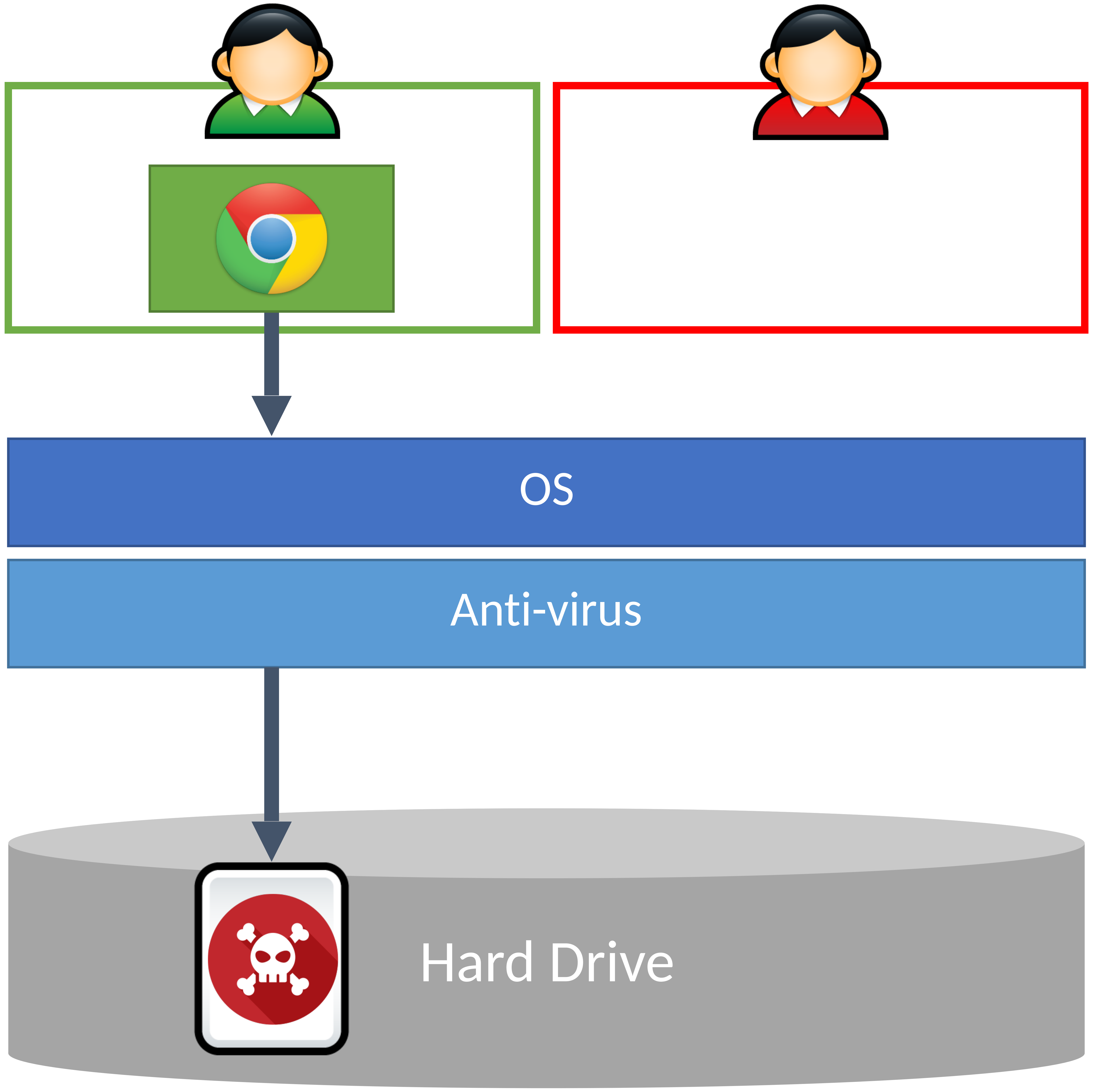
Anti-virus process is privileged

- Often 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

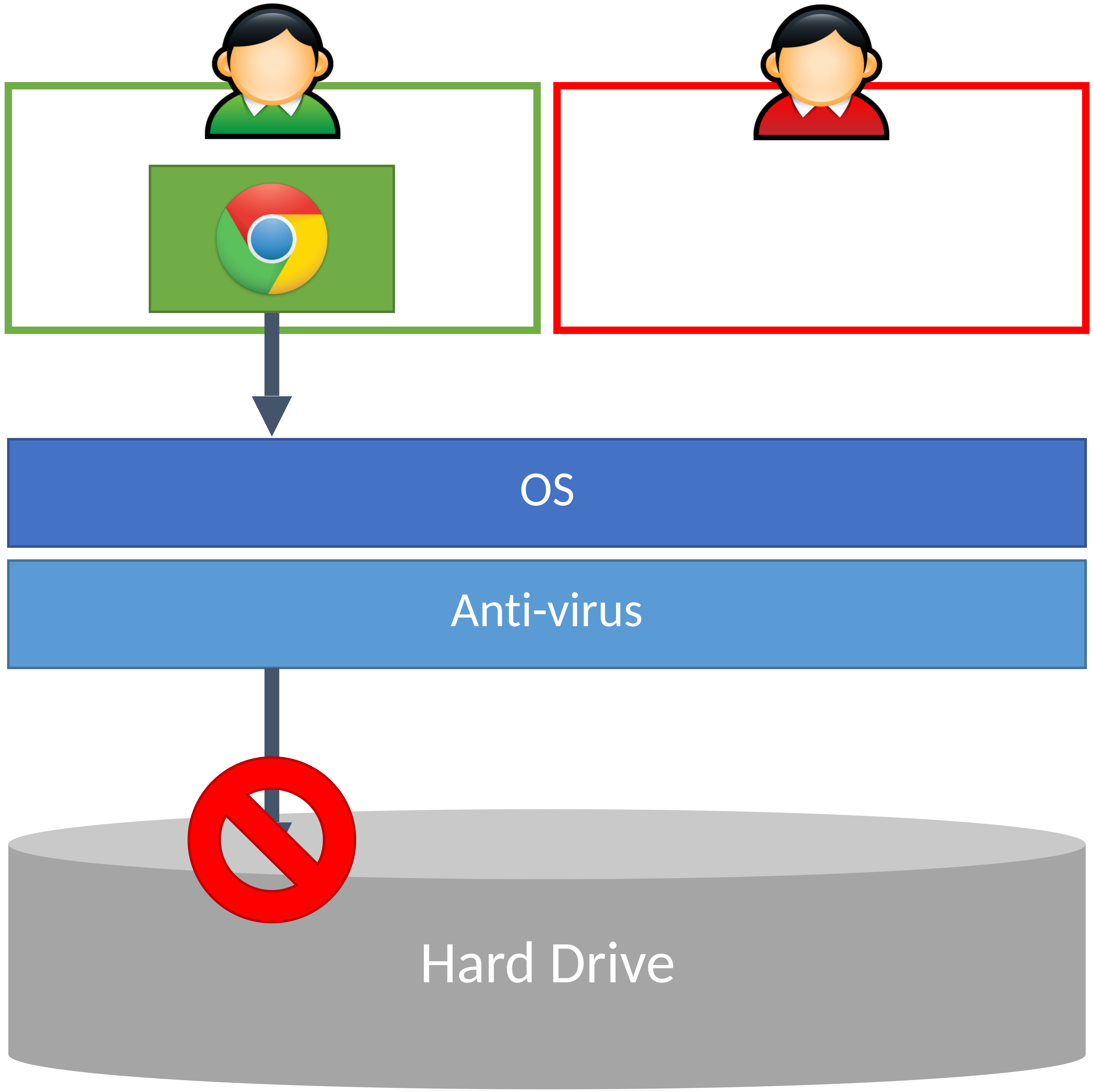
Anti-virus process is privileged

- Often 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

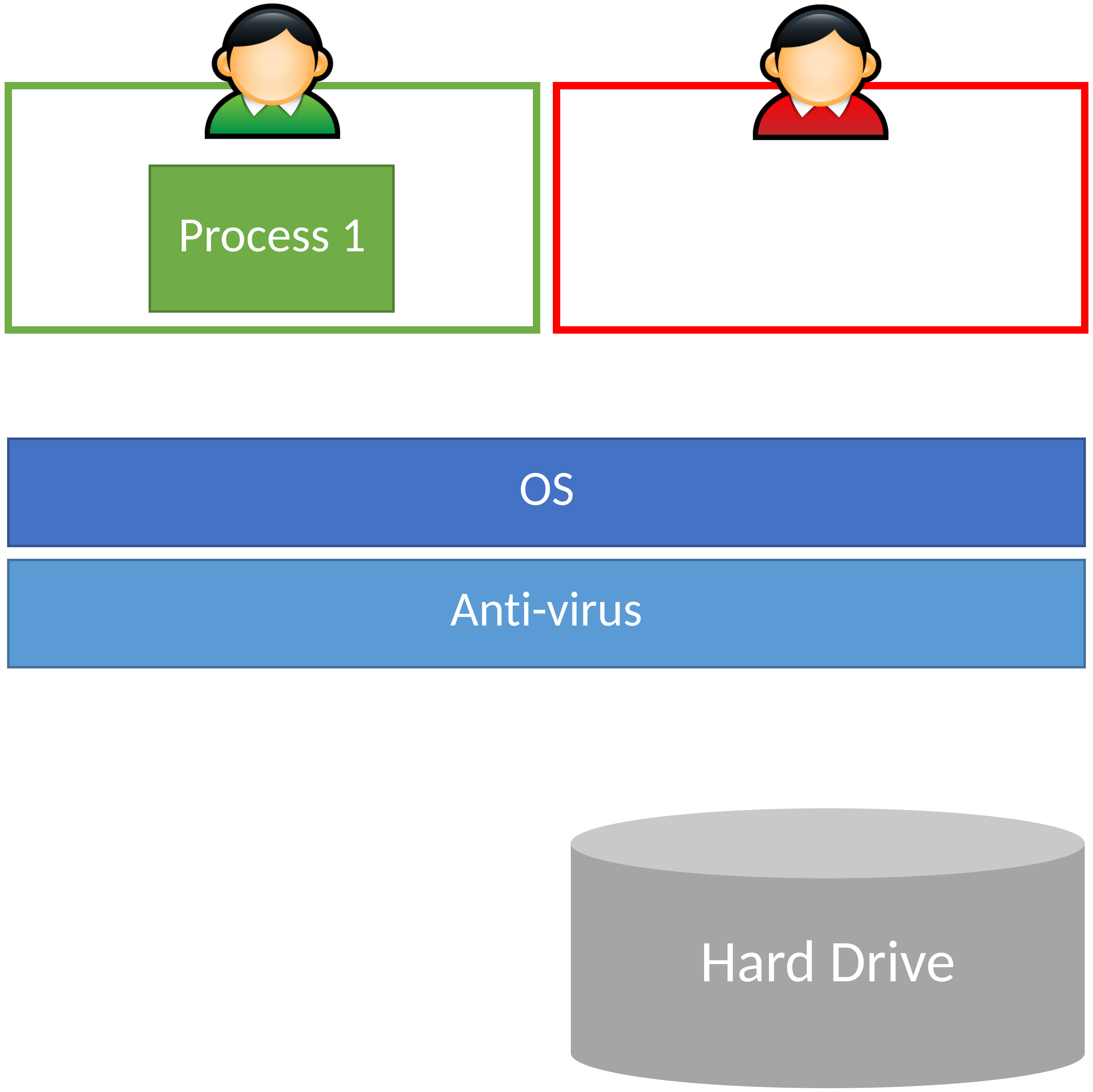
Anti-virus process is privileged

- 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

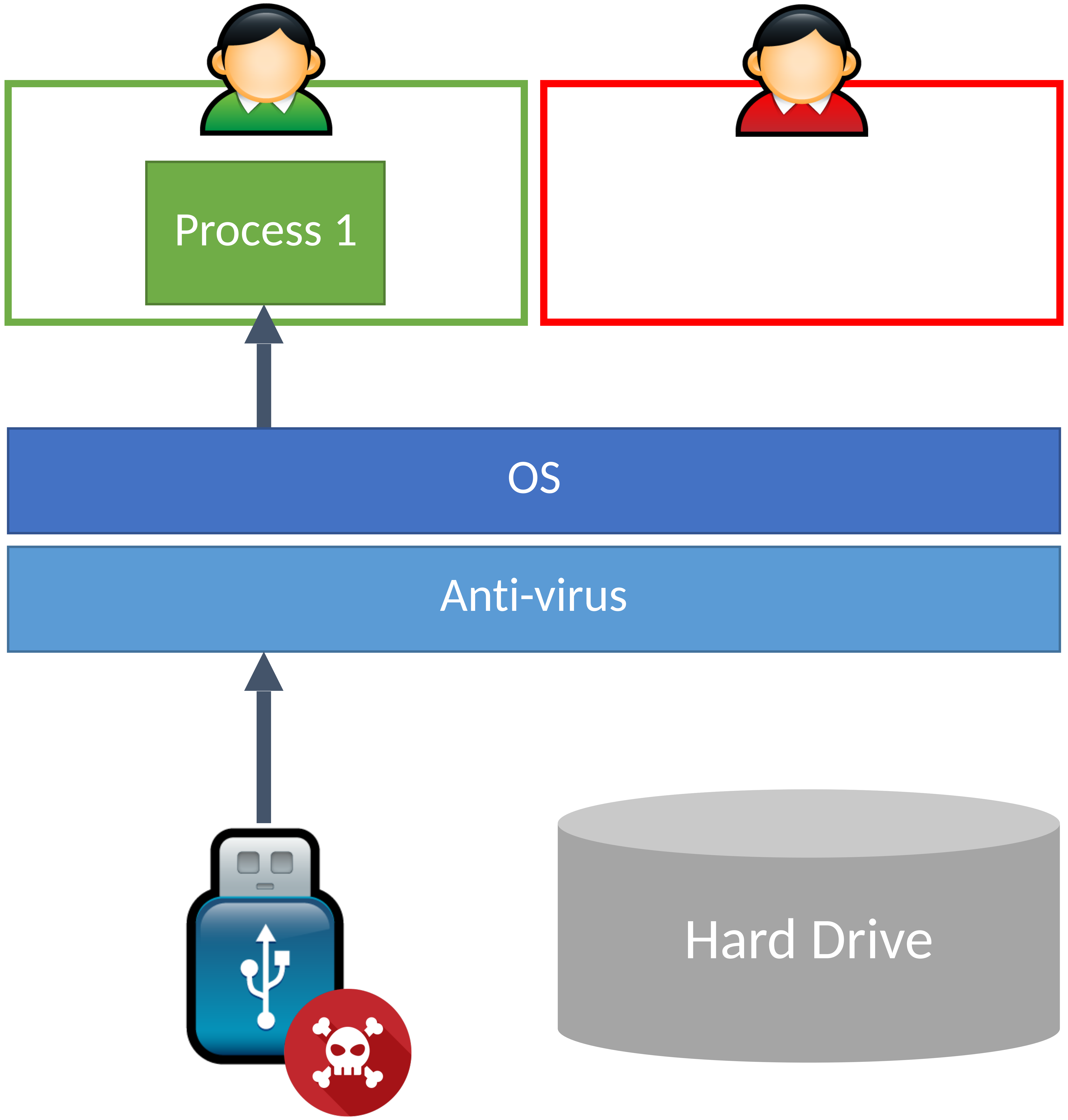
Anti-virus process is privileged

- 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



Anti-virus

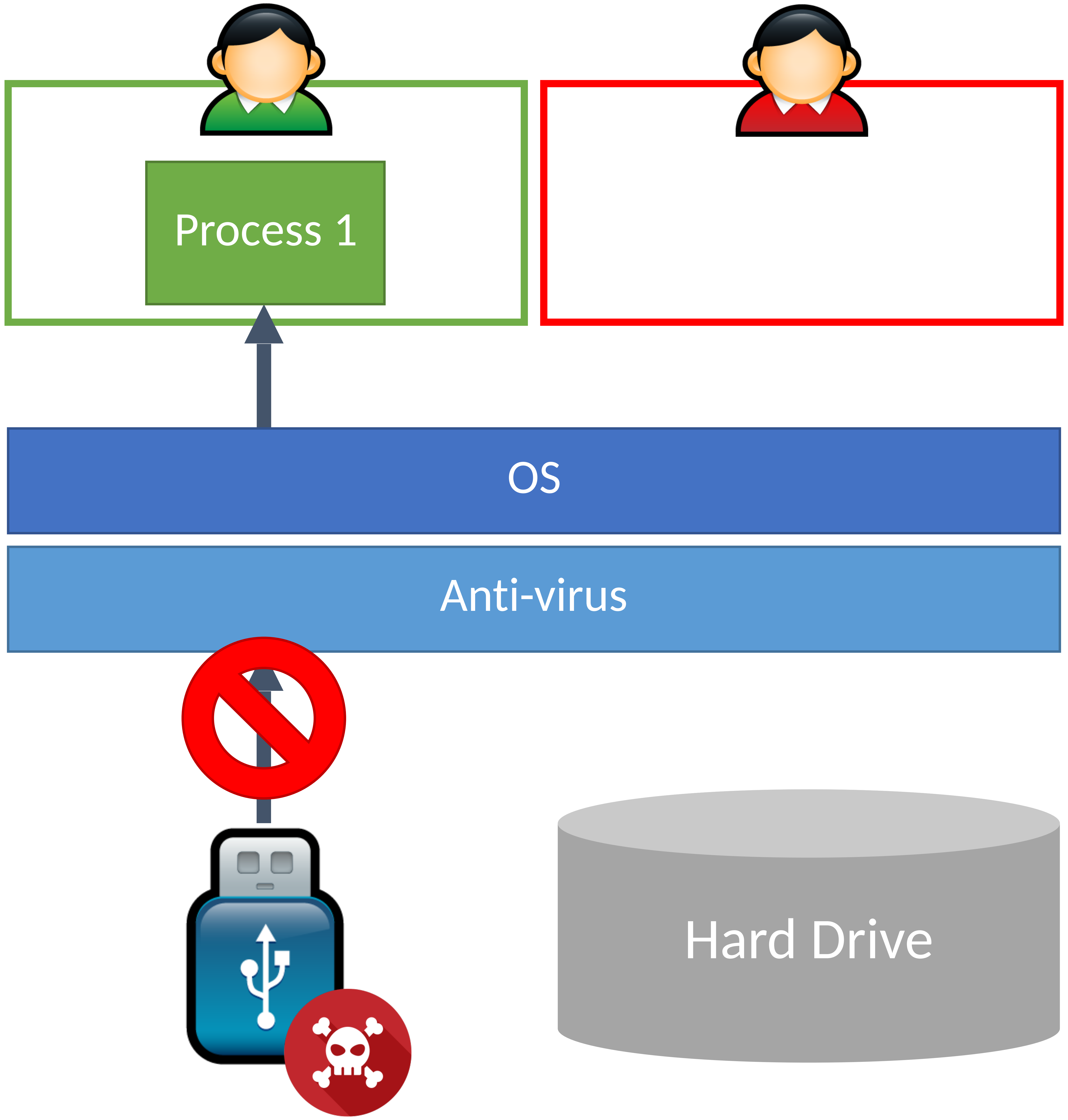
Anti-virus process is privileged

- 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.

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

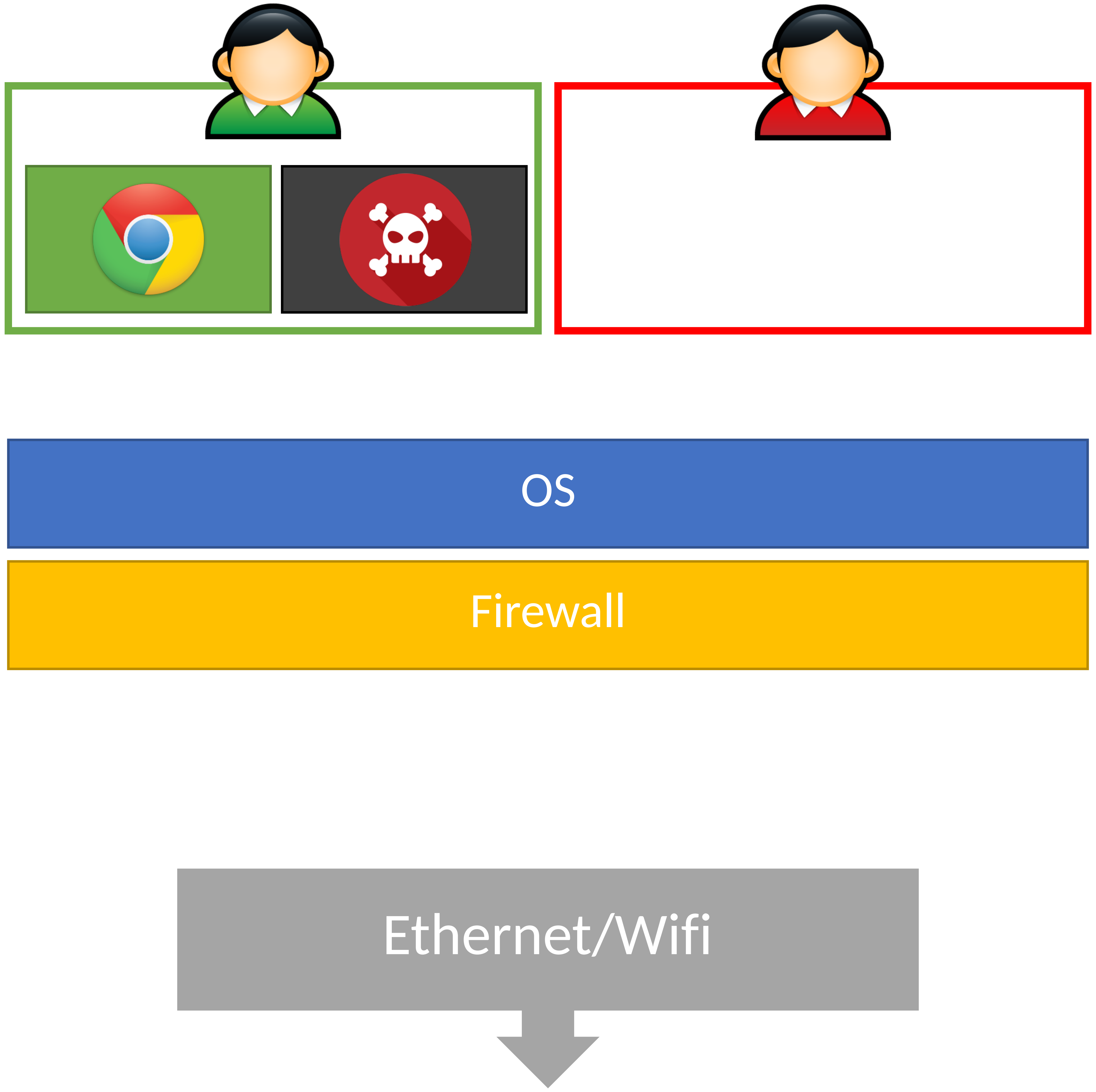
Firewall process is privileged

- 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

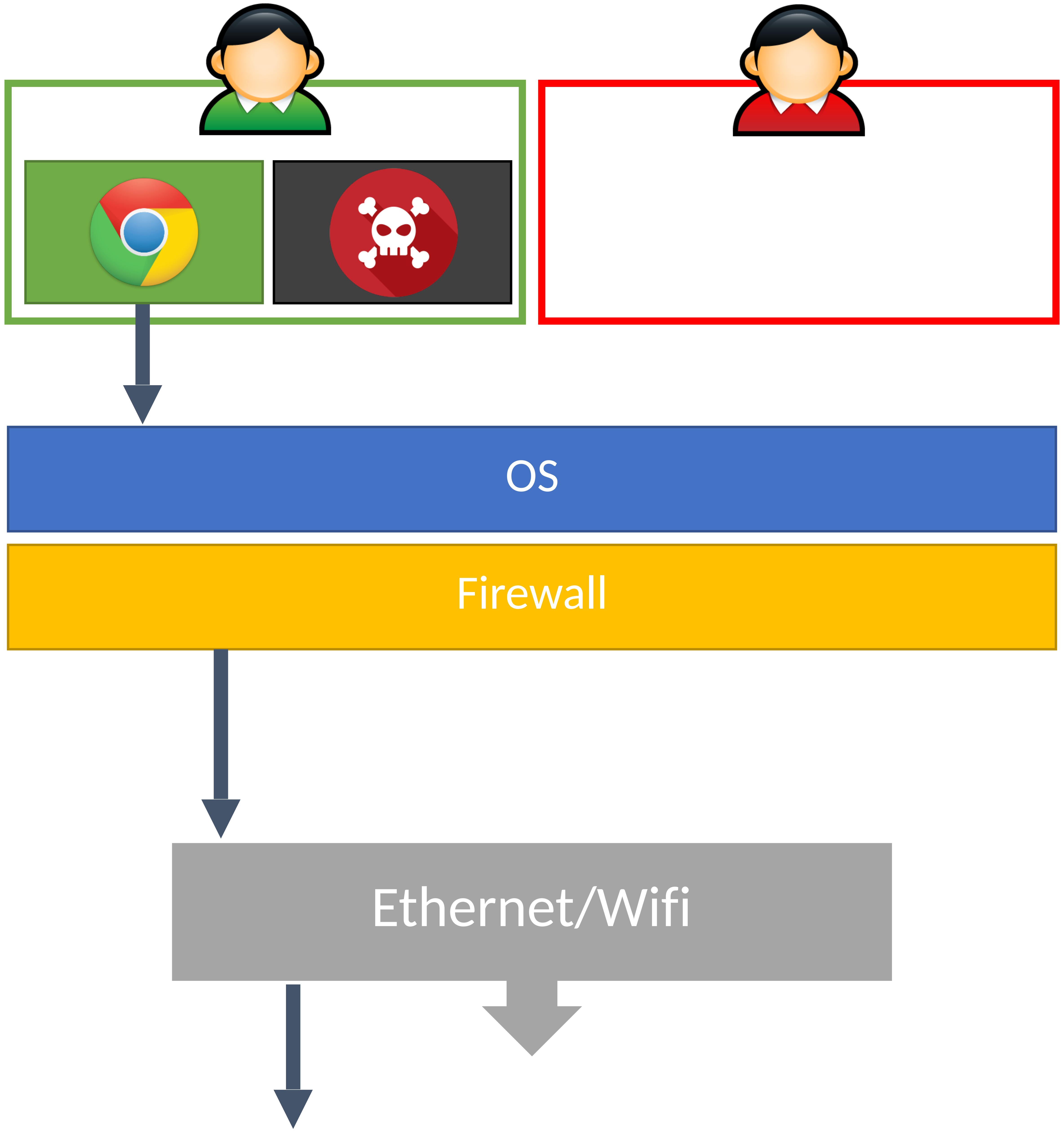
Firewall process is privileged

- 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

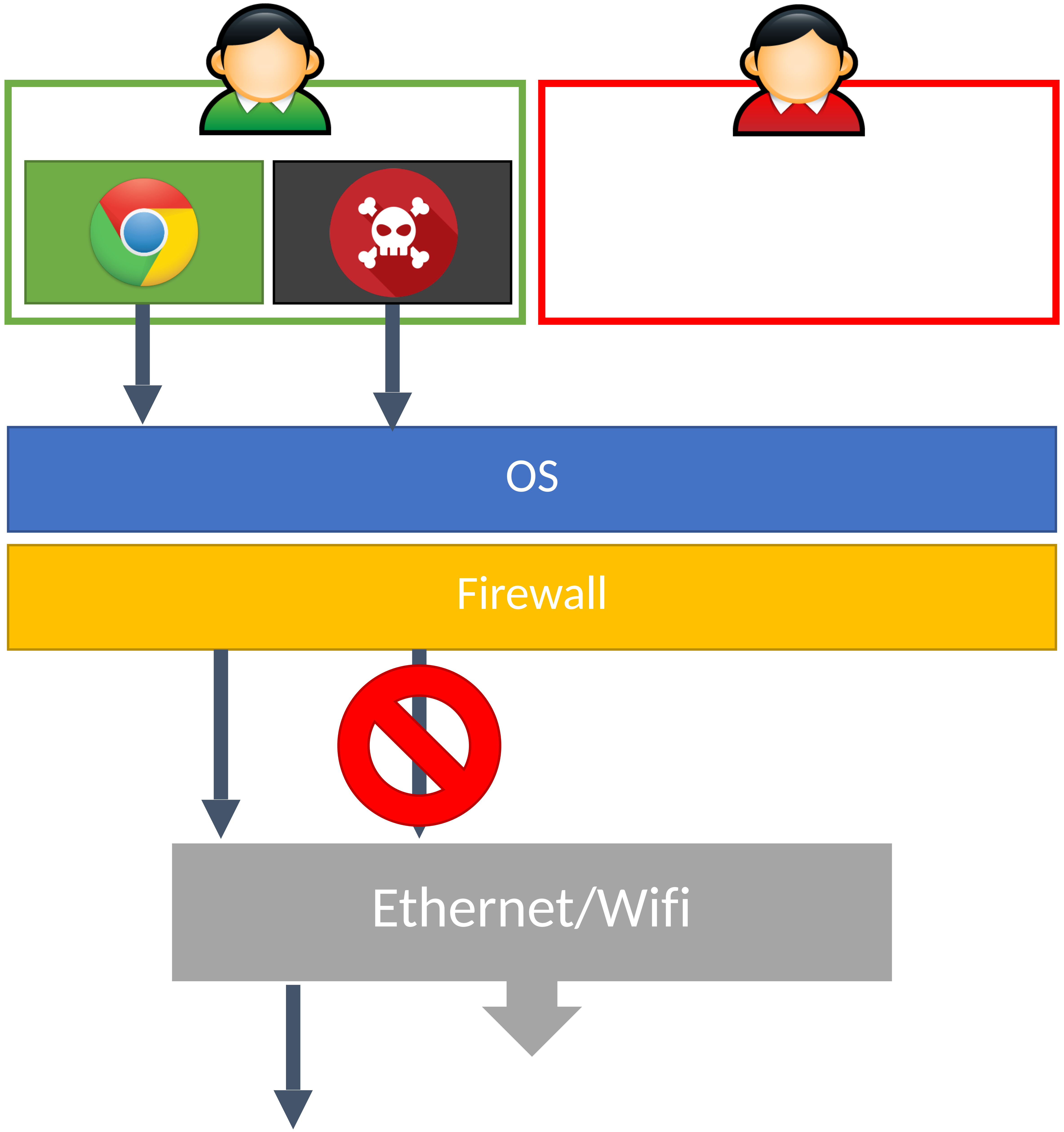
Firewall process is privileged

- 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

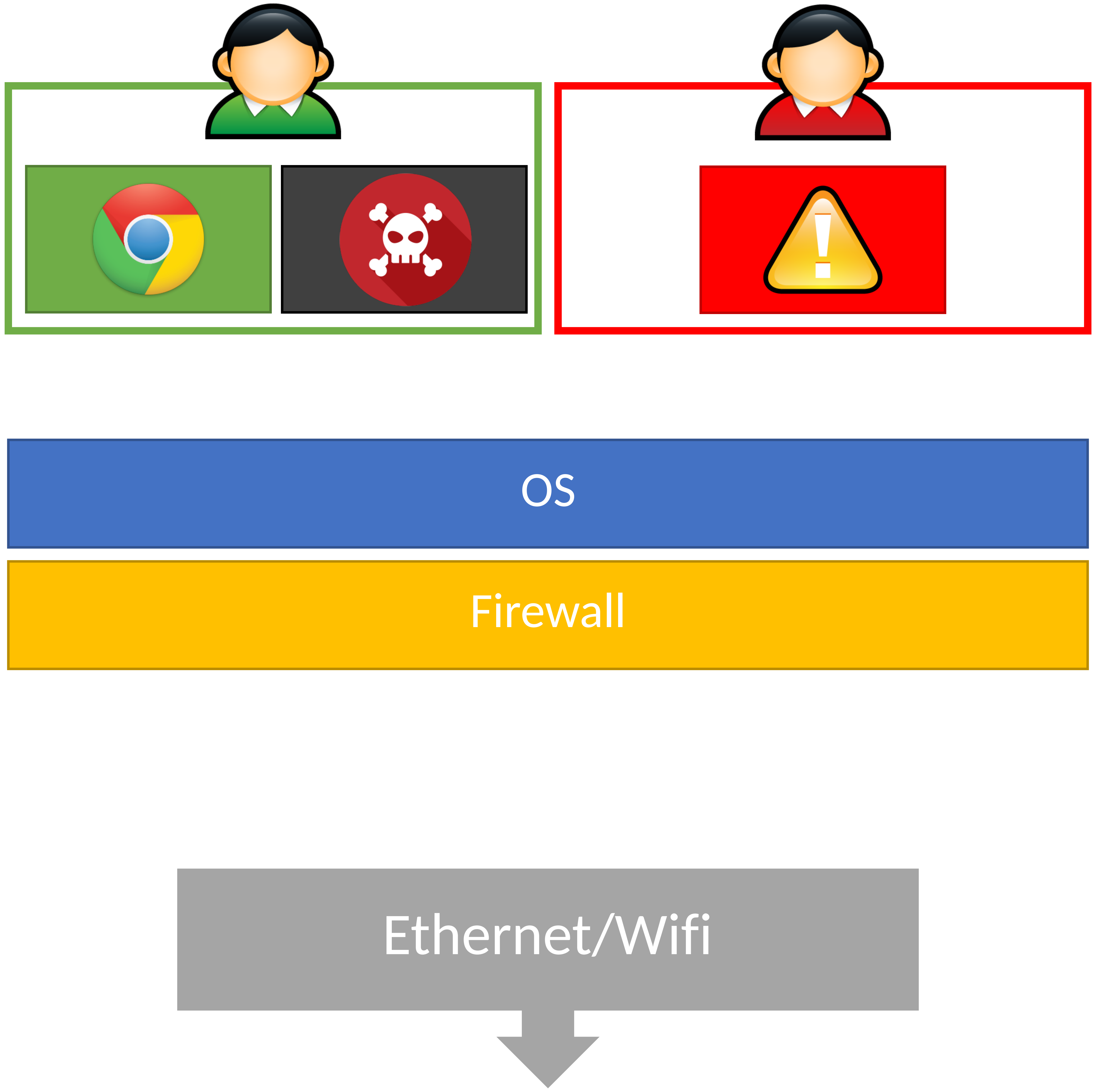
Firewall process is privileged

- 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

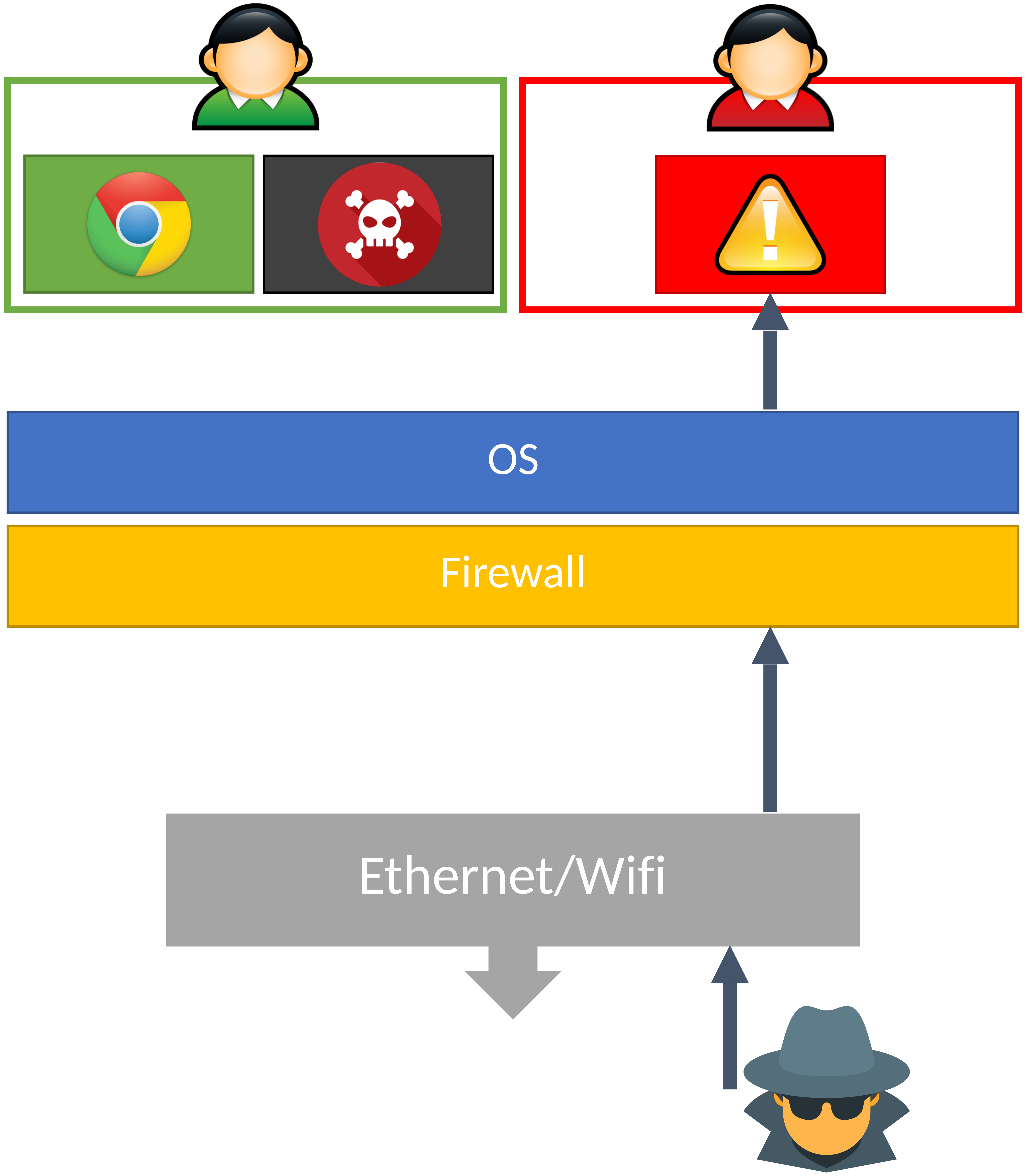
Firewall process is privileged

- 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

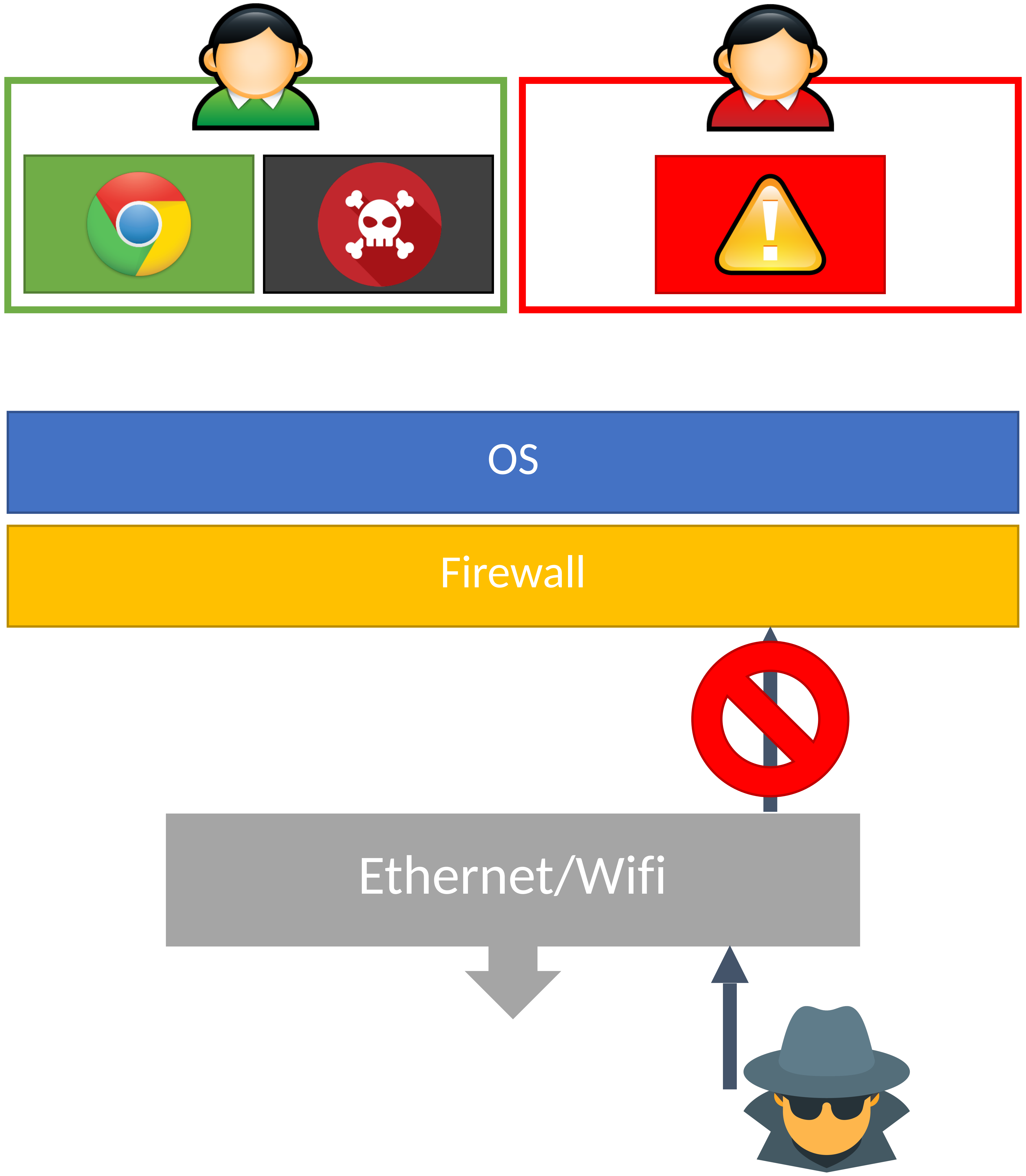
Firewall process is privileged

- 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

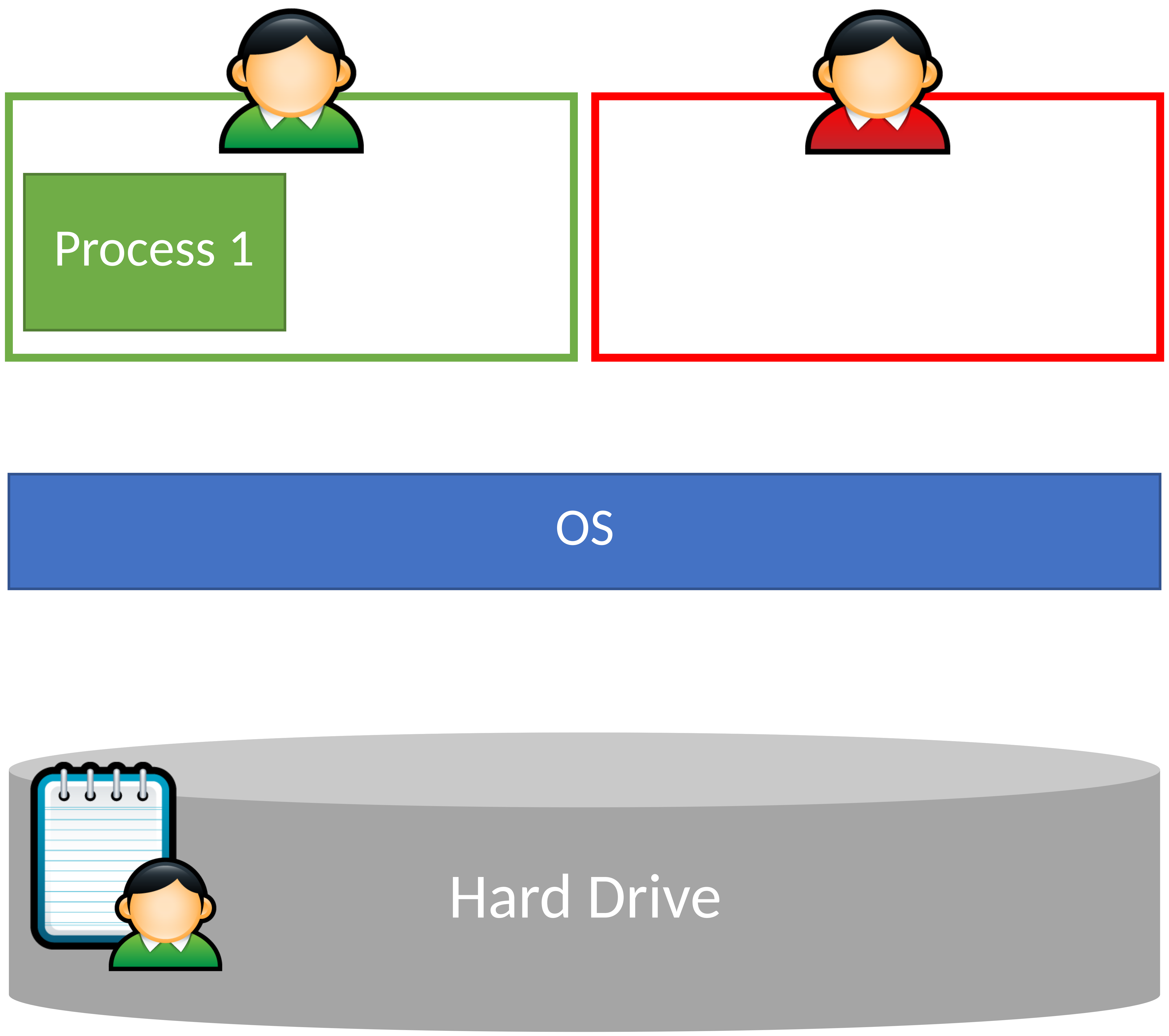
- Open source network monitoring, analysis, and logging framework
- Can be used to implement signature based detection
- Capable of more complex analysis

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

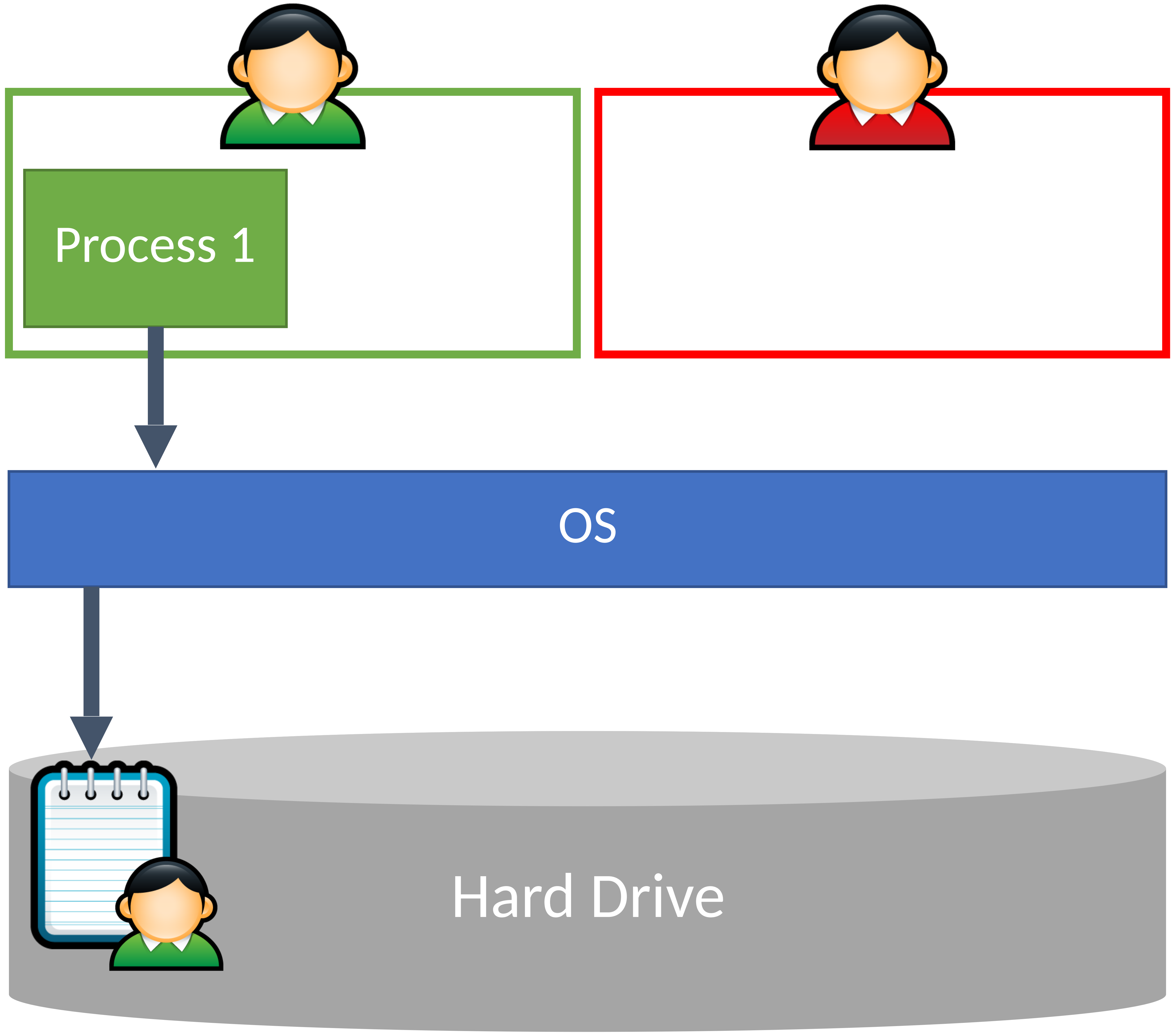


Insecure Logging

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

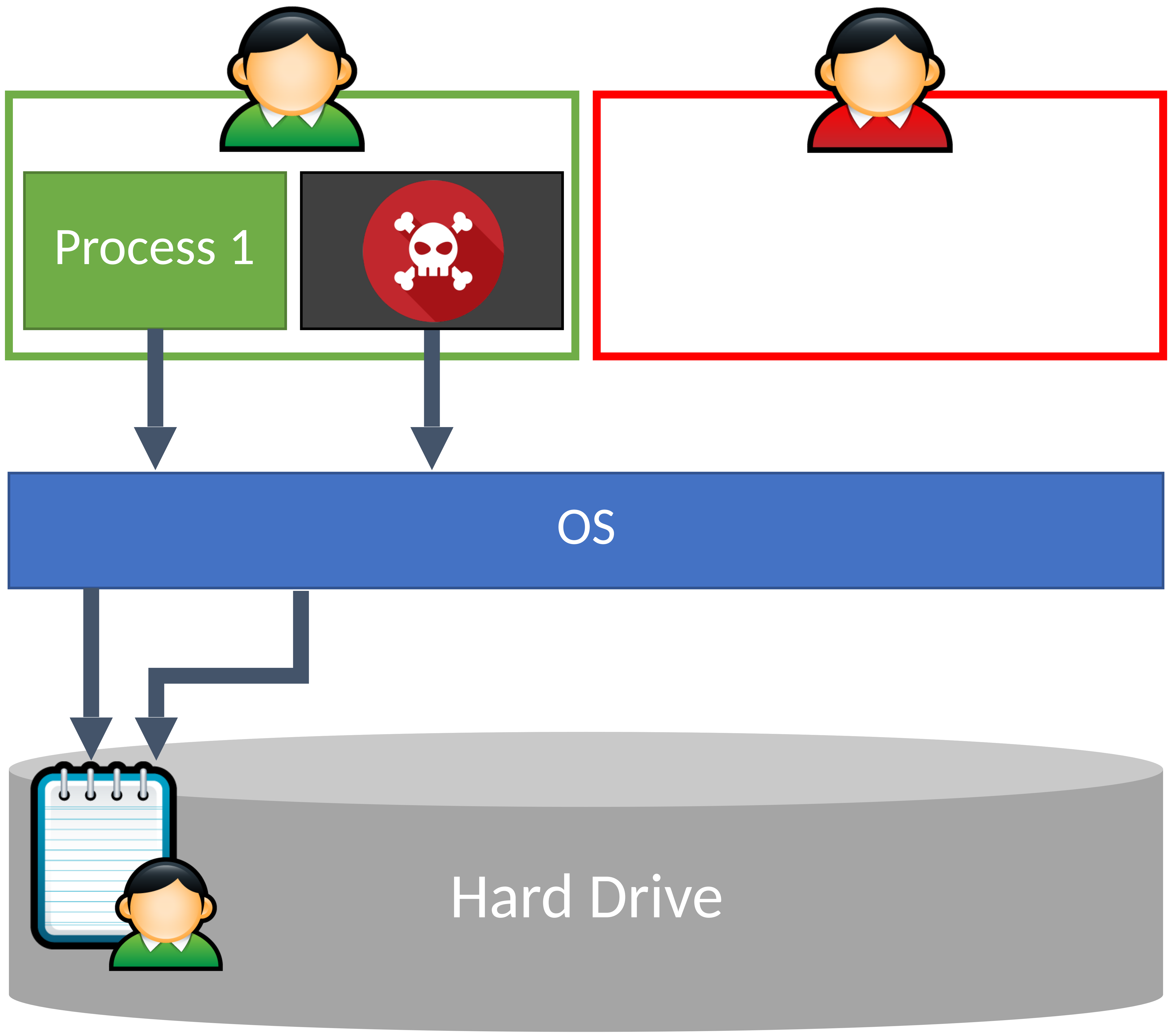


Insecure Logging

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

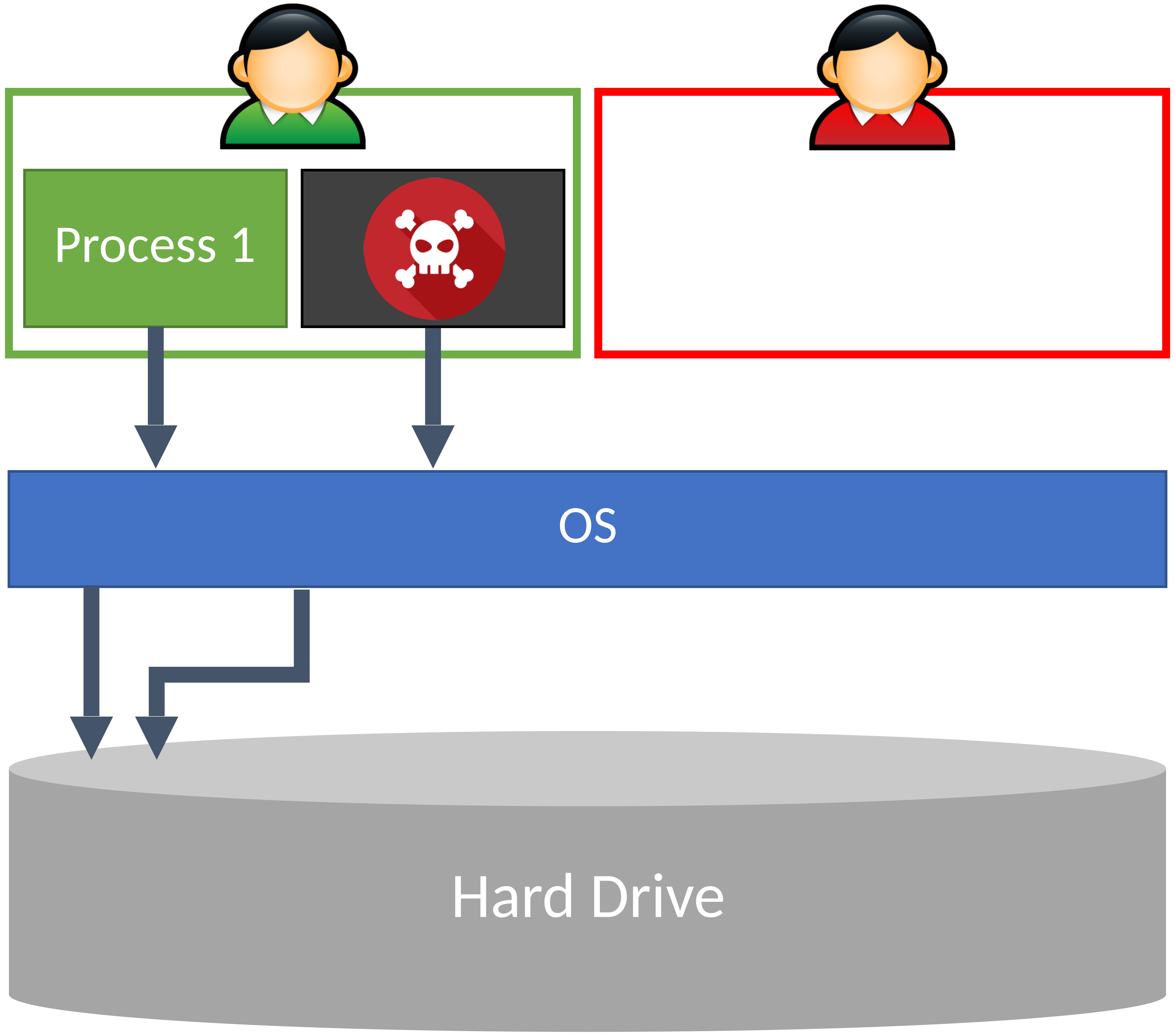


Insecure Logging

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

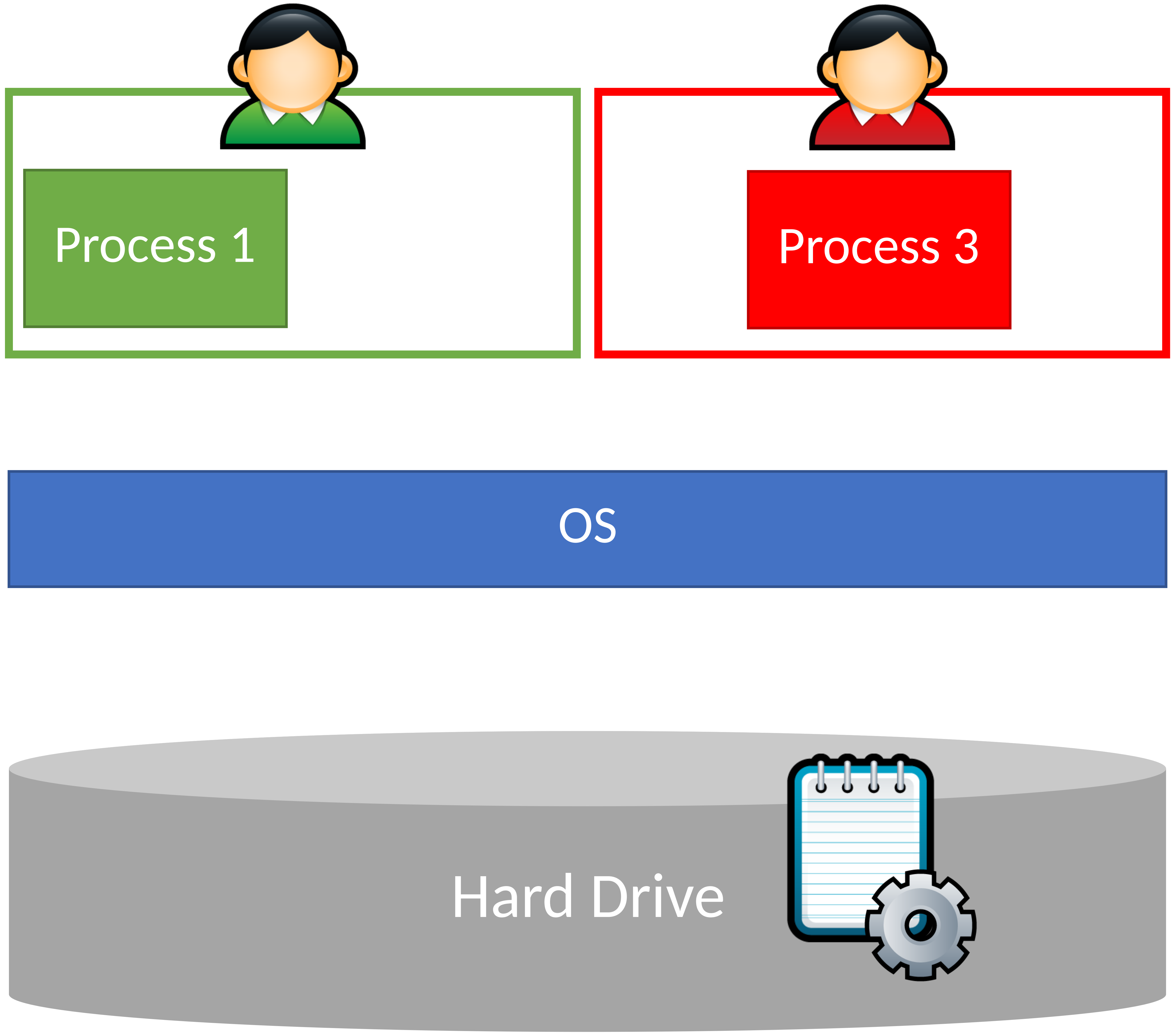


Secure Logging

OS maintains a system log

Processes may write entries to the log using an OS API

Processes may not delete entries or the log

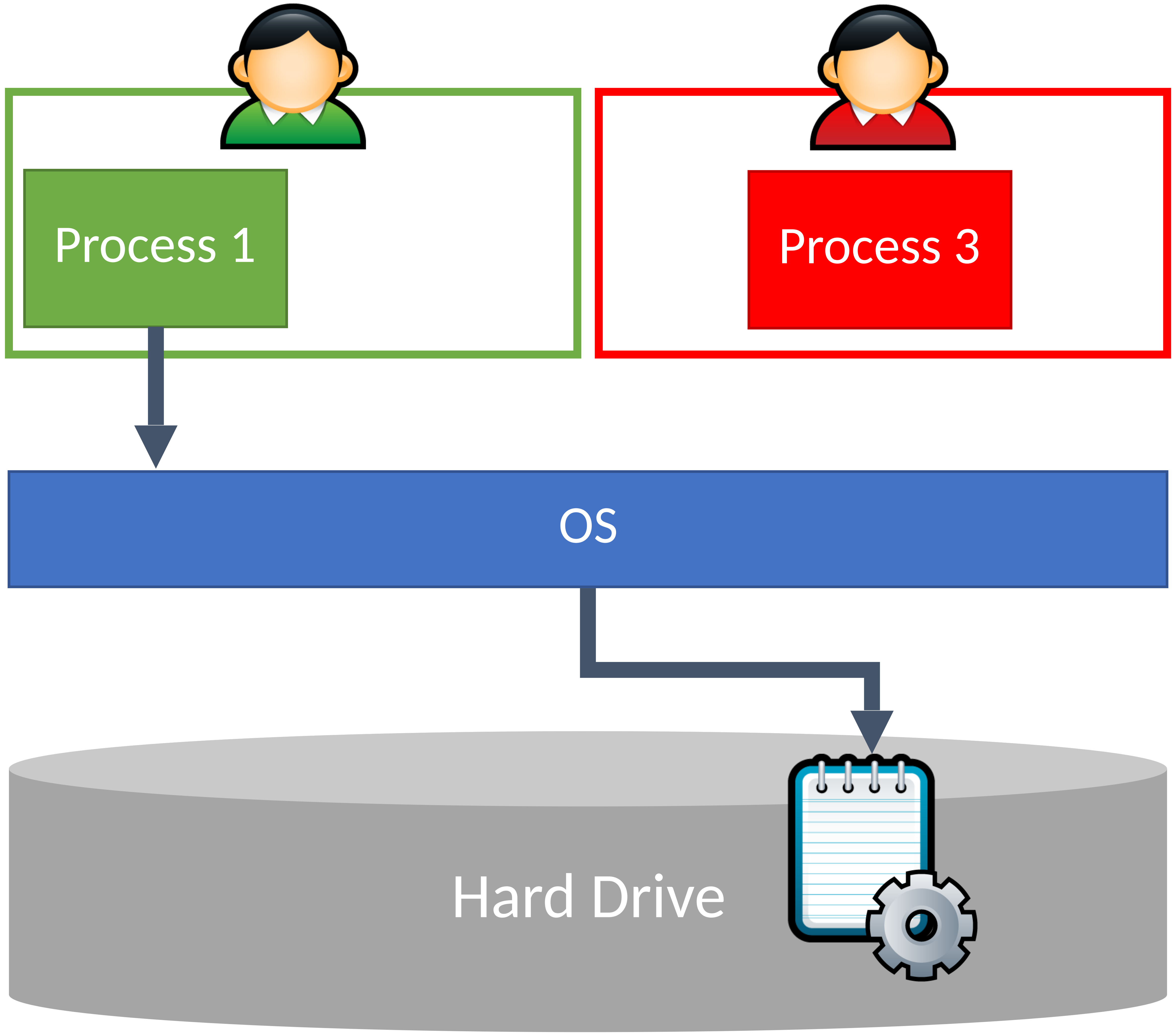


Secure Logging

OS maintains a system log

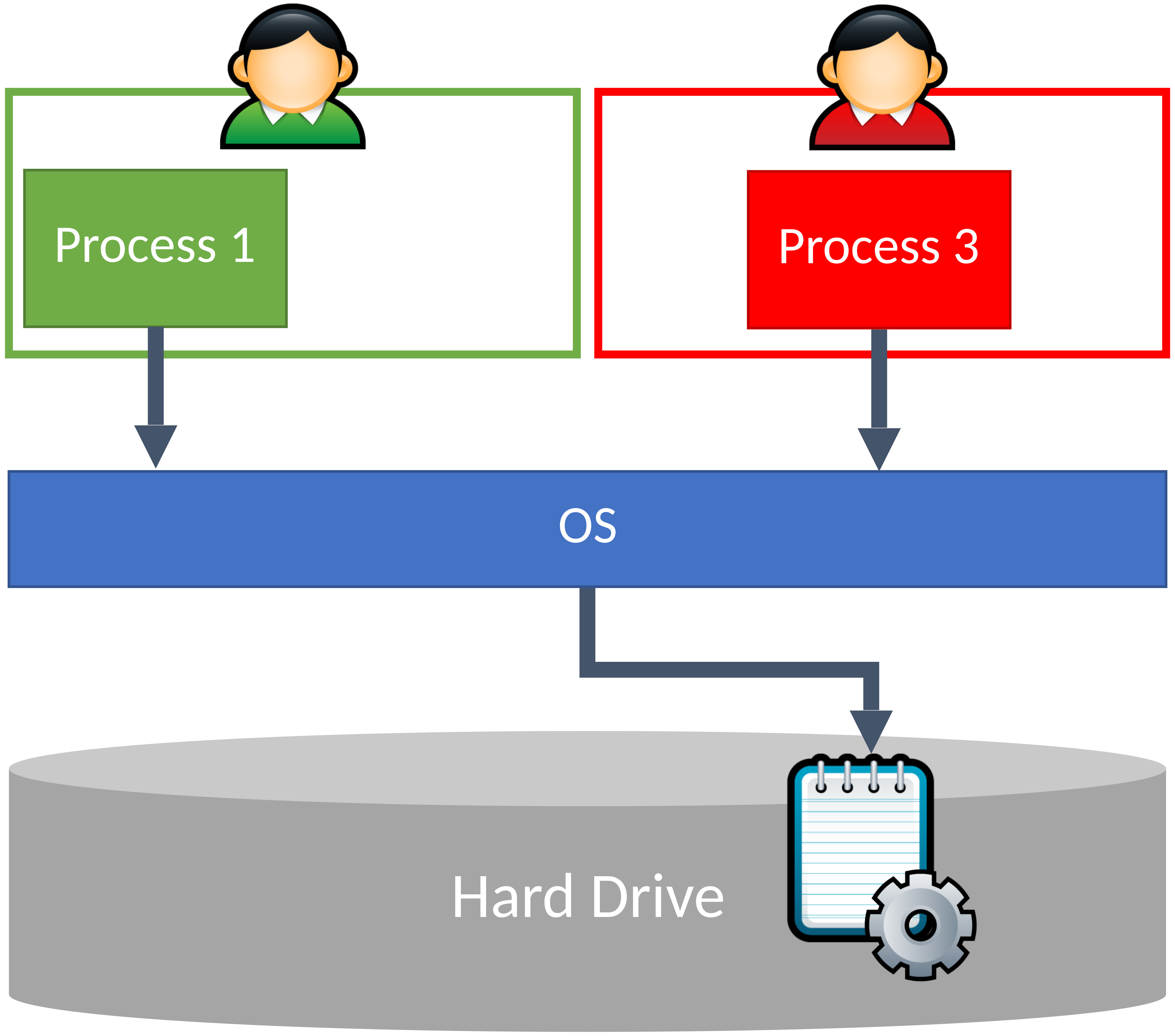
Processes may write entries to the log using an OS API

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