CY 2550 Foundations of Cybersecurity

Systems Security

Alina Oprea
Associate Professor, Khoury College
Northeastern University
March 26 2020

Announcements

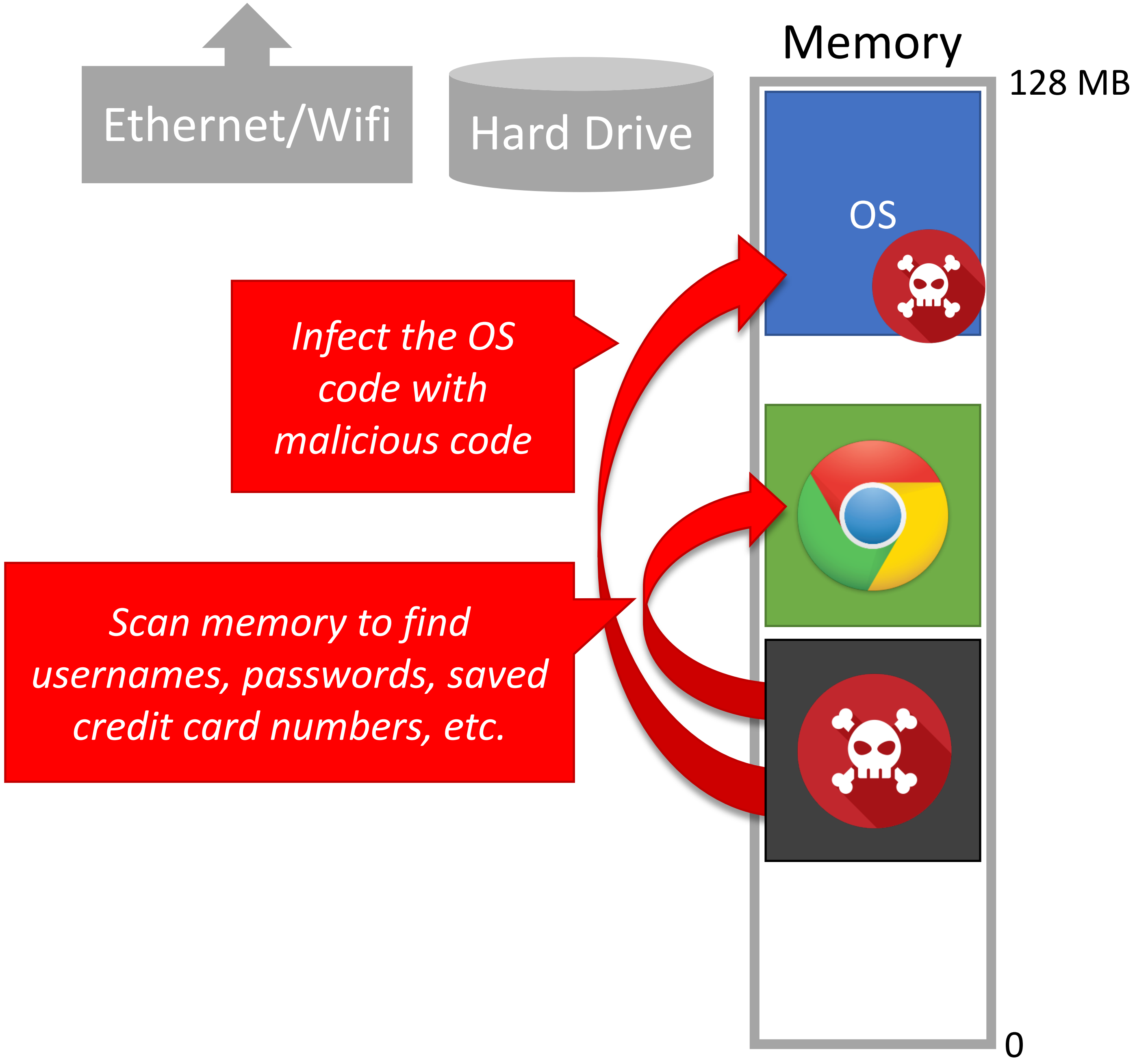
- Social engineering and ethics projects are due today
- Forensics project will be released today, due on April 4
- Exploit project is the last one, due on April 17
- Final exam
  - Take home
  - Released on April 13 at 11:45am EST, due on April 14 at noon
  - Submitted through Gradescope
  - Questions on the material to test general understanding
  - Might include questions from the “Countdown to Zero Day” book

Systems Security

Threat Model
Intro to Computer Architecture
Hardware Support for Isolation
Security Technologies
Principles

Memory Unsafety

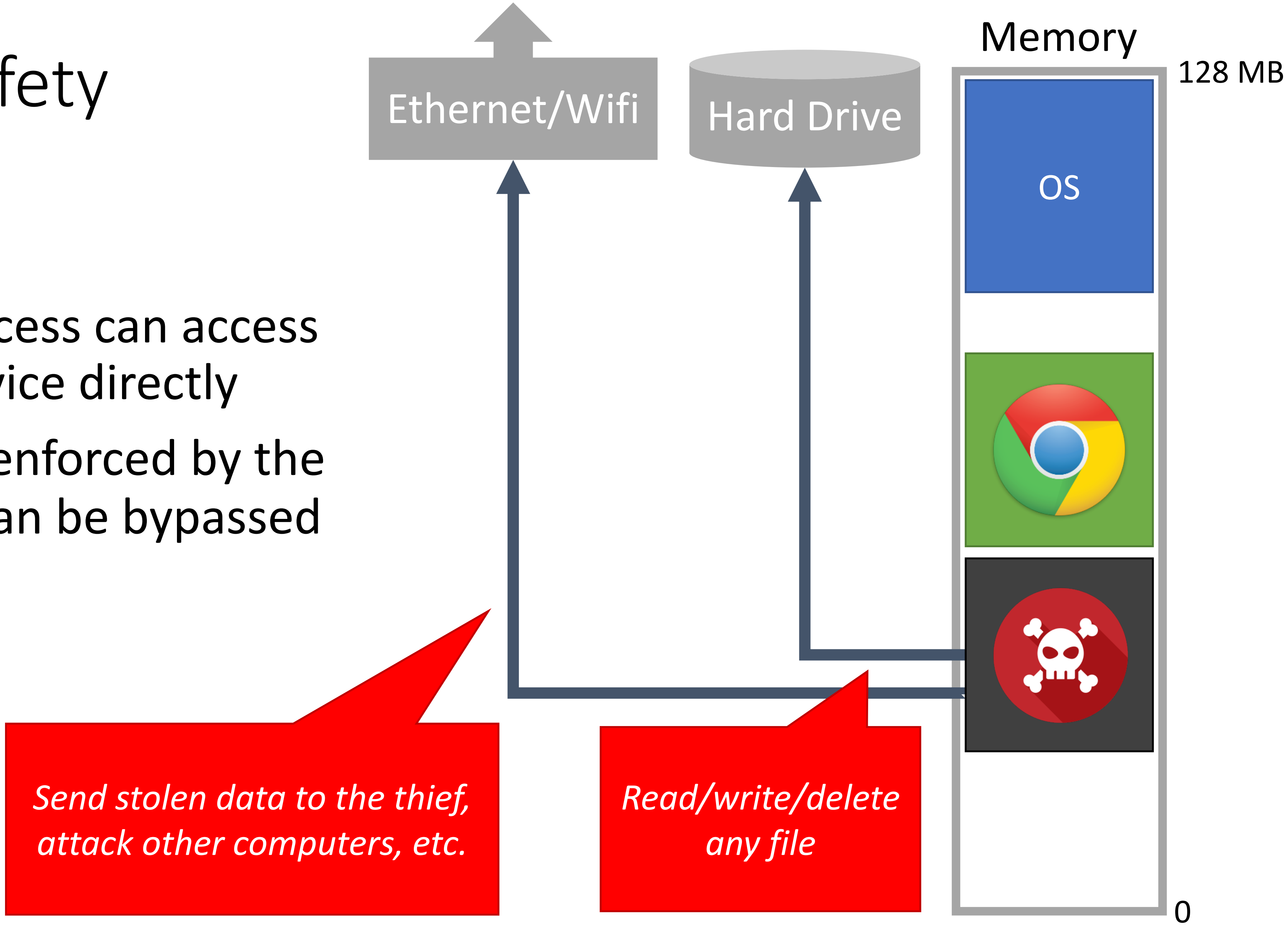
Problem: any process can read/write any memory



Device Unsafety

Problem: any process can access any hardware device directly

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



Review

Old systems did not protect memory or devices

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

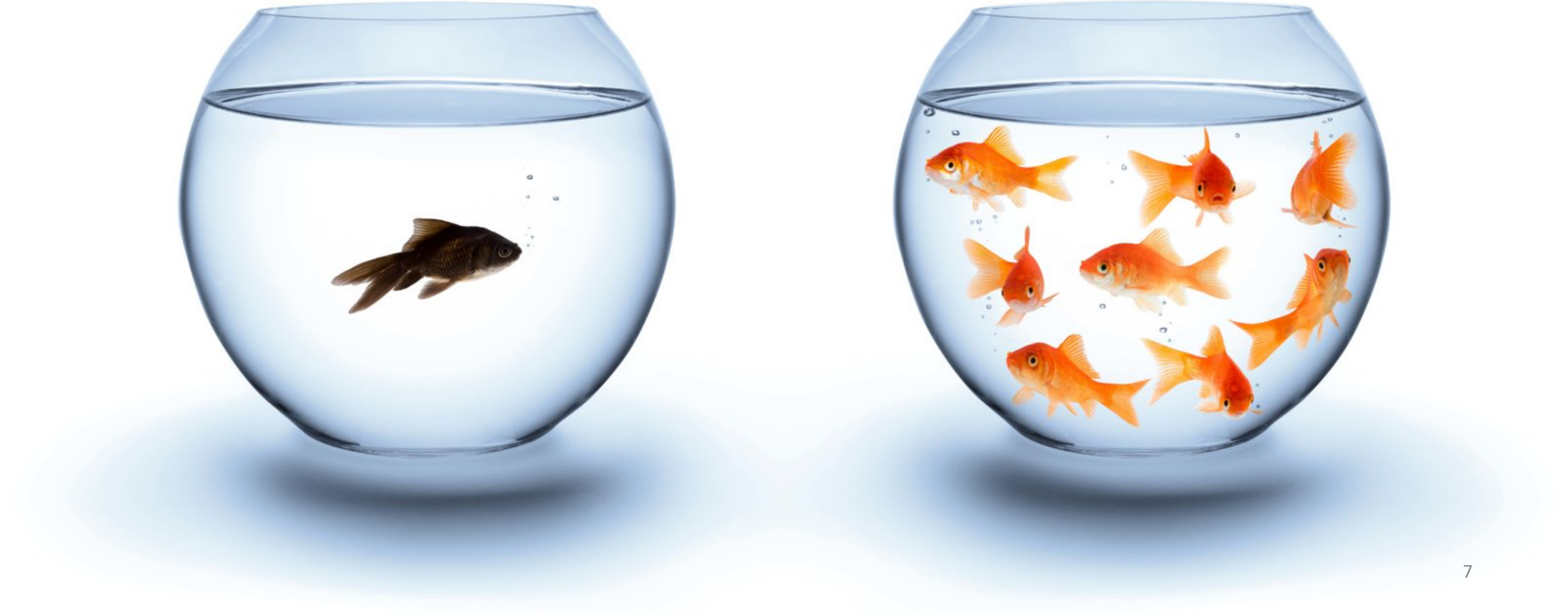
Problems

- 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

How do we fix these in modern architectures?

ISOLATION



Systems Security

Threat Model
Intro to Computer Architecture
Hardware Support for Isolation
Security Technologies
Principles

Modern Architecture

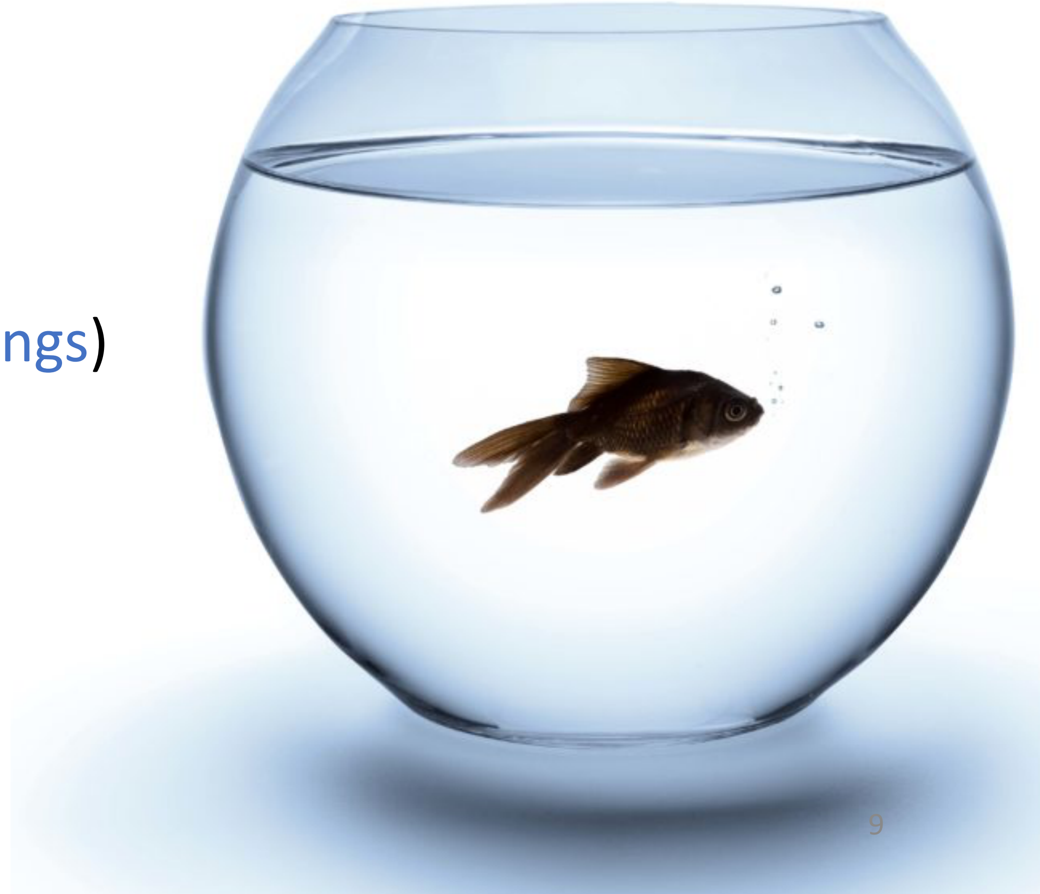
To achieve systems security, we need process isolation

- 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

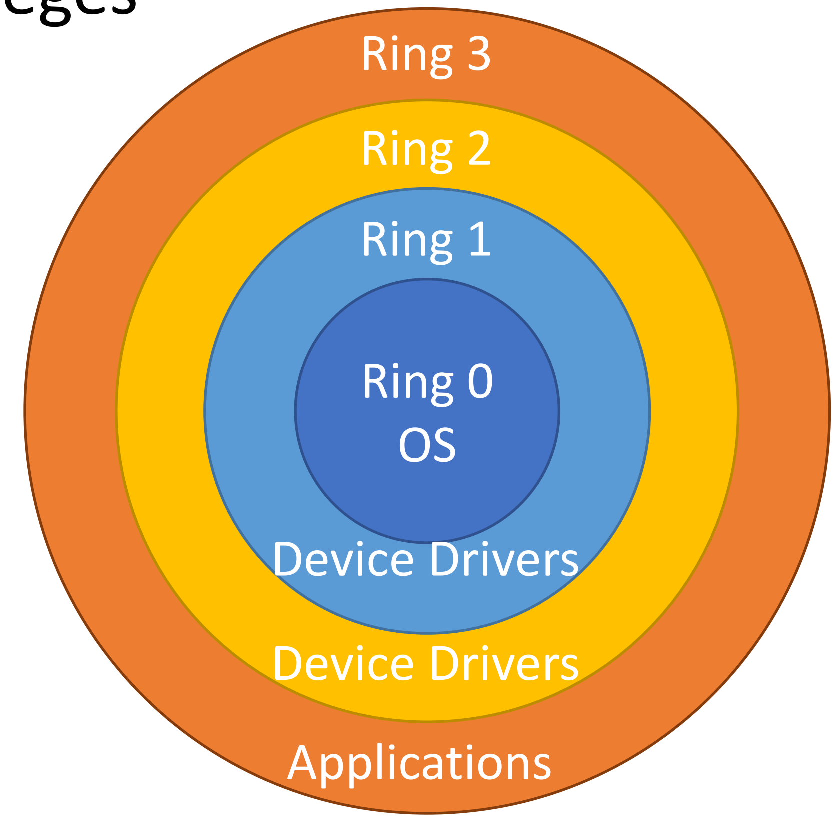


Protected Mode

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

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

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

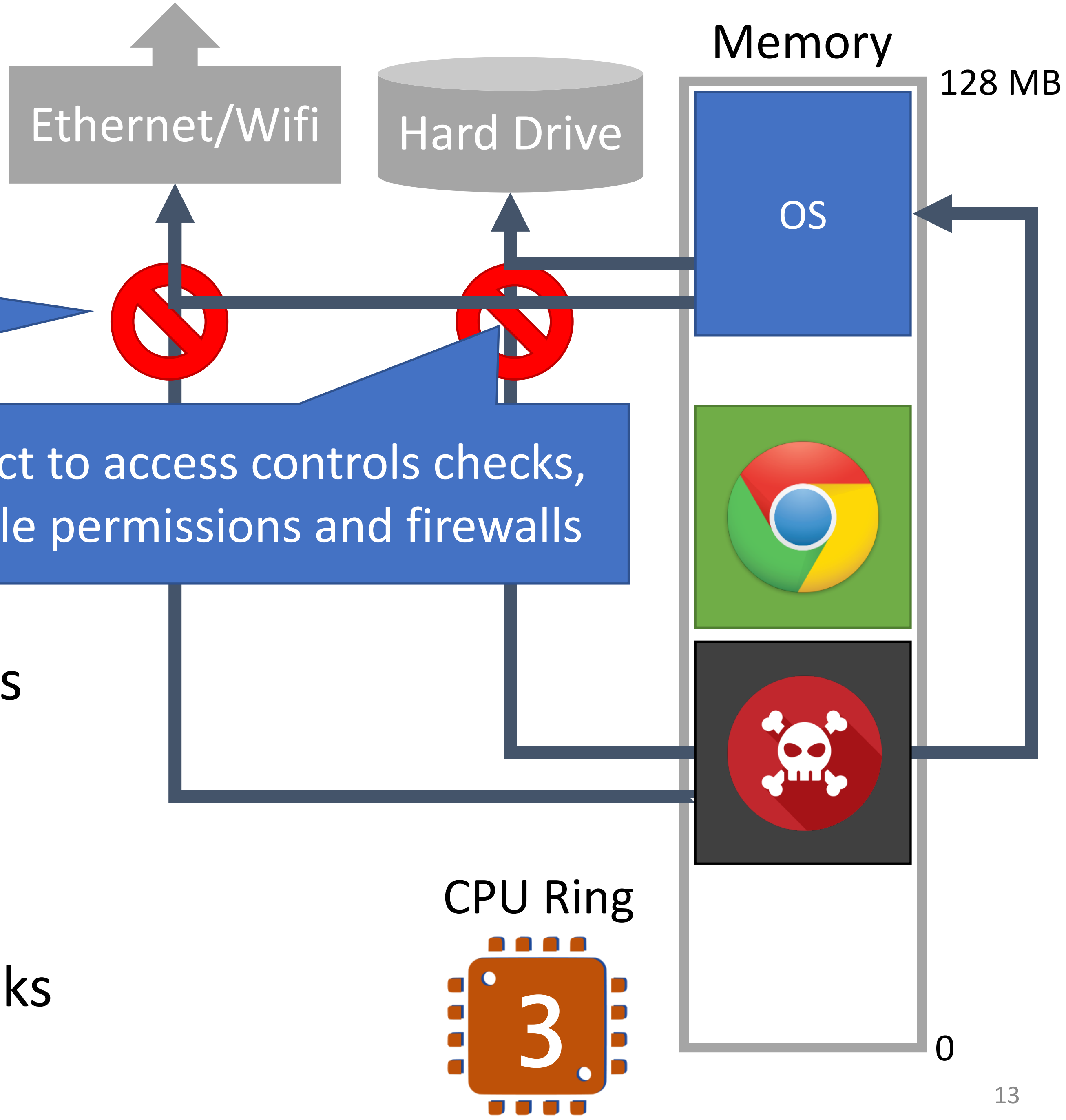
How do processes get access to the OS?

- 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

Protection in Action

Ring 3 = protected mode.
No direct device access

Subject to access controls checks,
e.g. file permissions and firewalls



Protected mode stops direct access to devices

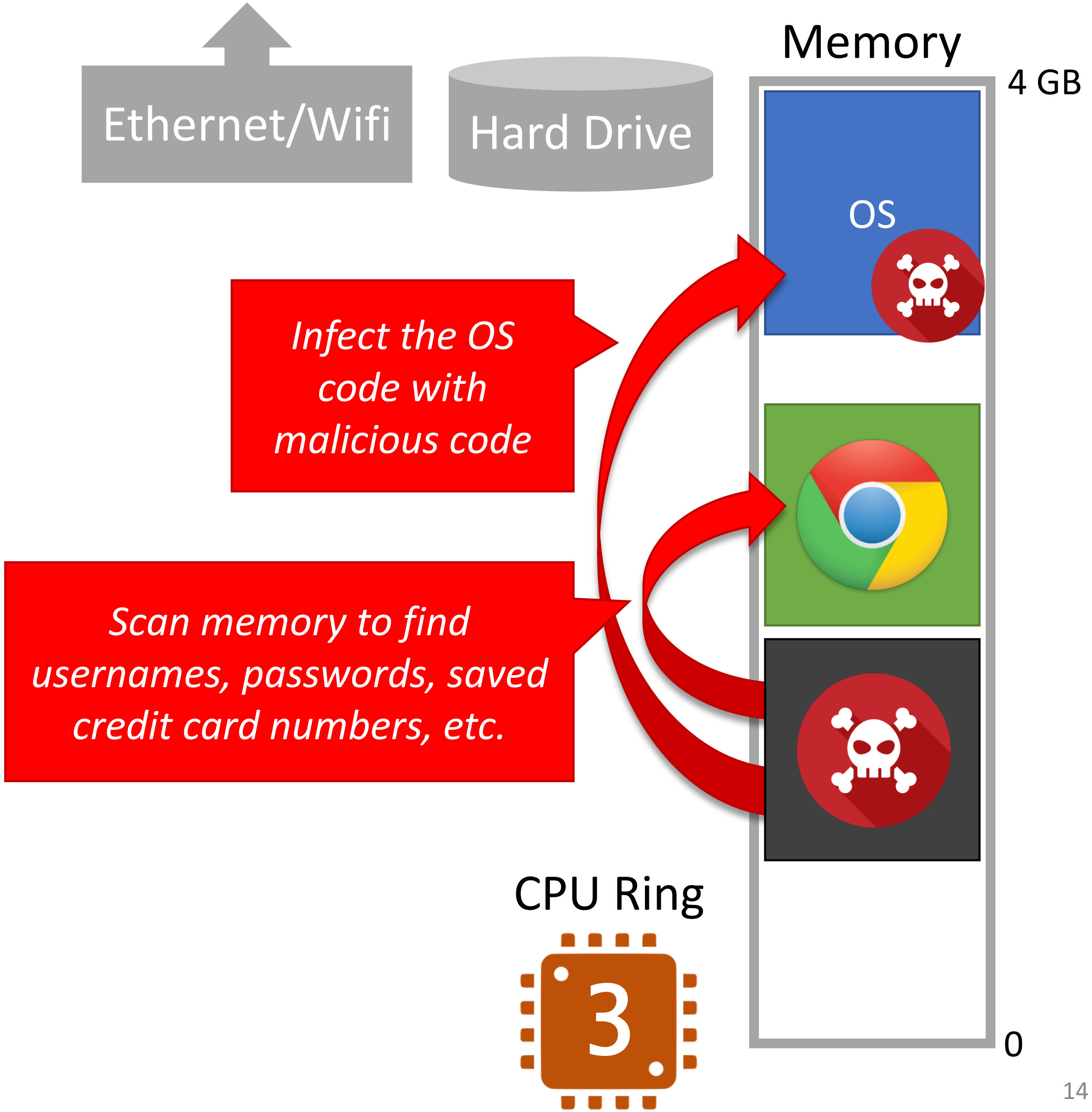
All device access must go through the OS

OS will impose access control checks

Status Check

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

… But we have not protected memory



Virtual Memory

Modern CPUs support virtual memory

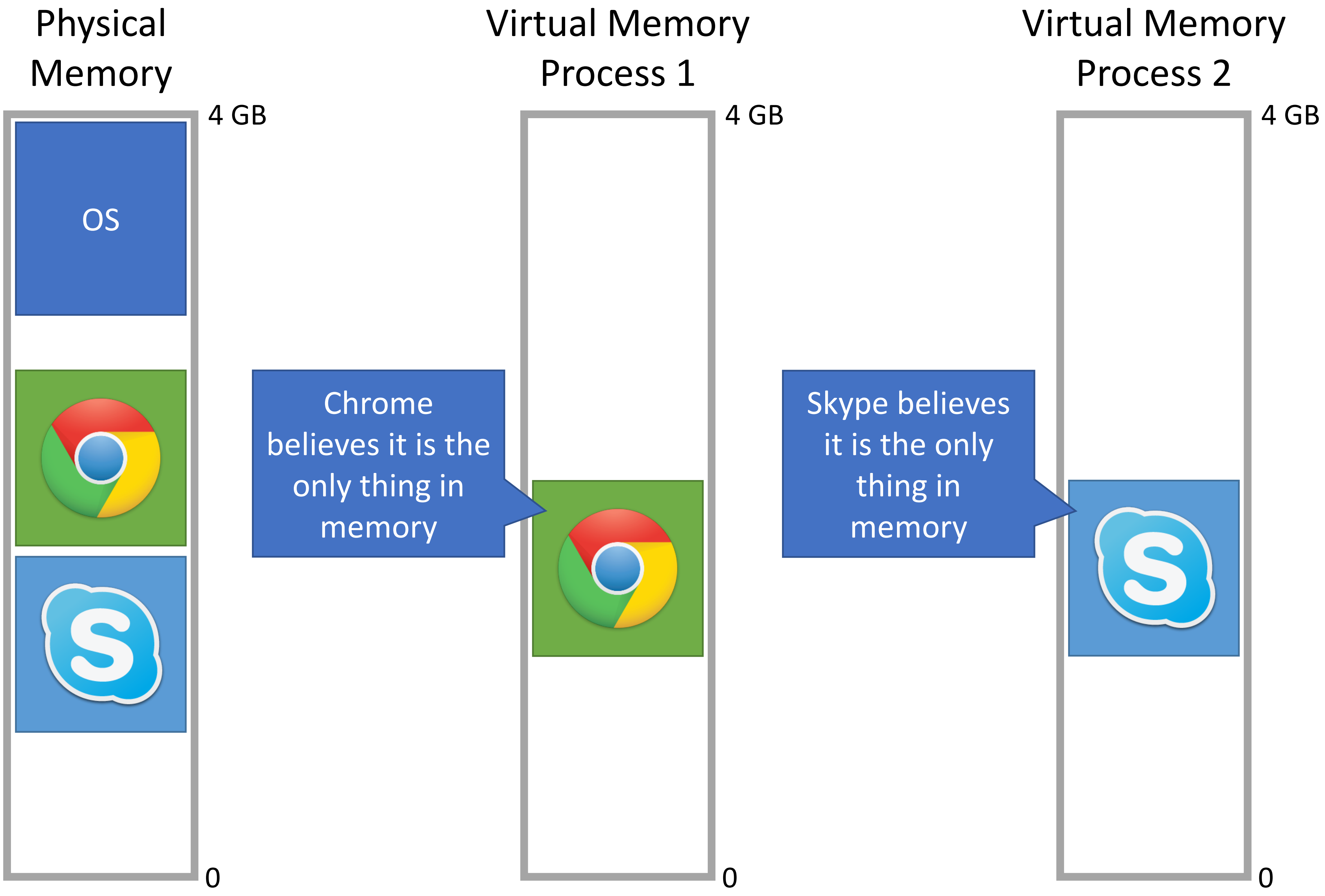
Creates the illusion that each process runs in its own, empty memory space

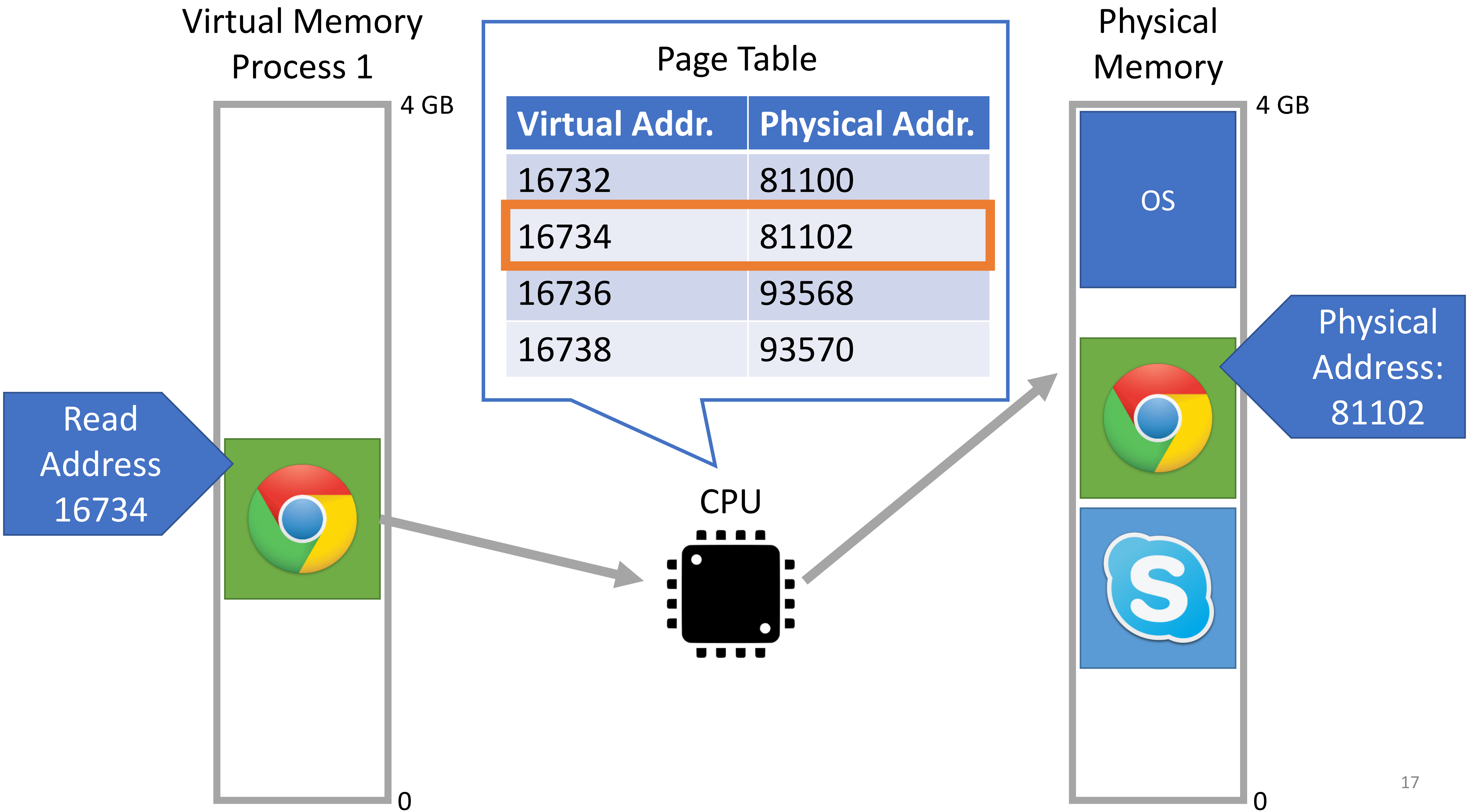
- Processes can not read/write memory used by other processes
- Processes can not read/write memory used by the OS

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



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

Virtual Memory Implementation

Each process has its own virtual memory space

- Each process has a page table that maps in 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

Systems Security

Threat Model
Intro to Computer Architecture
Hardware Support for Isolation
Security Technologies
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

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

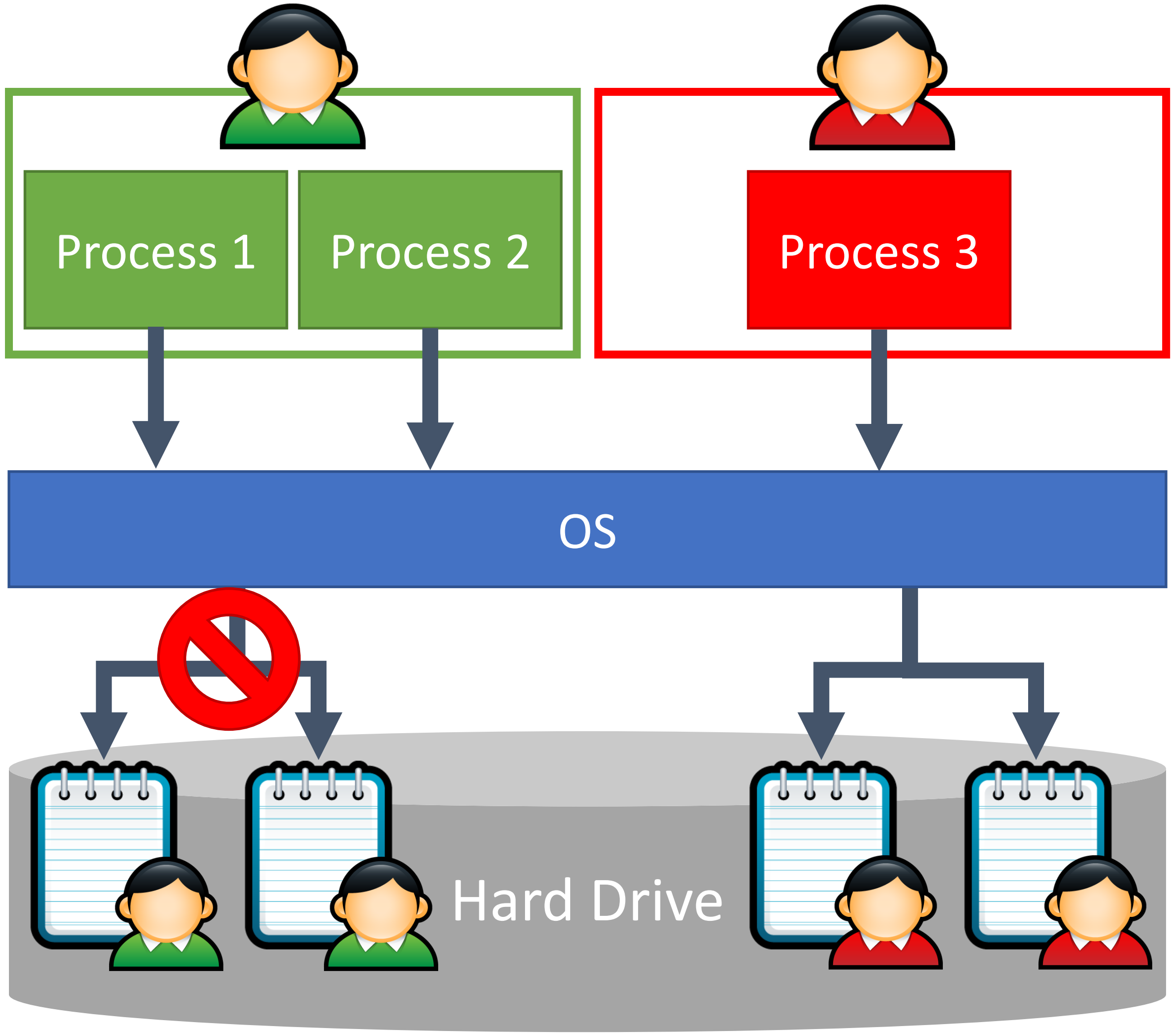

Firewall


Secure Logging

File Access Control

All disk access is mediated by the OS

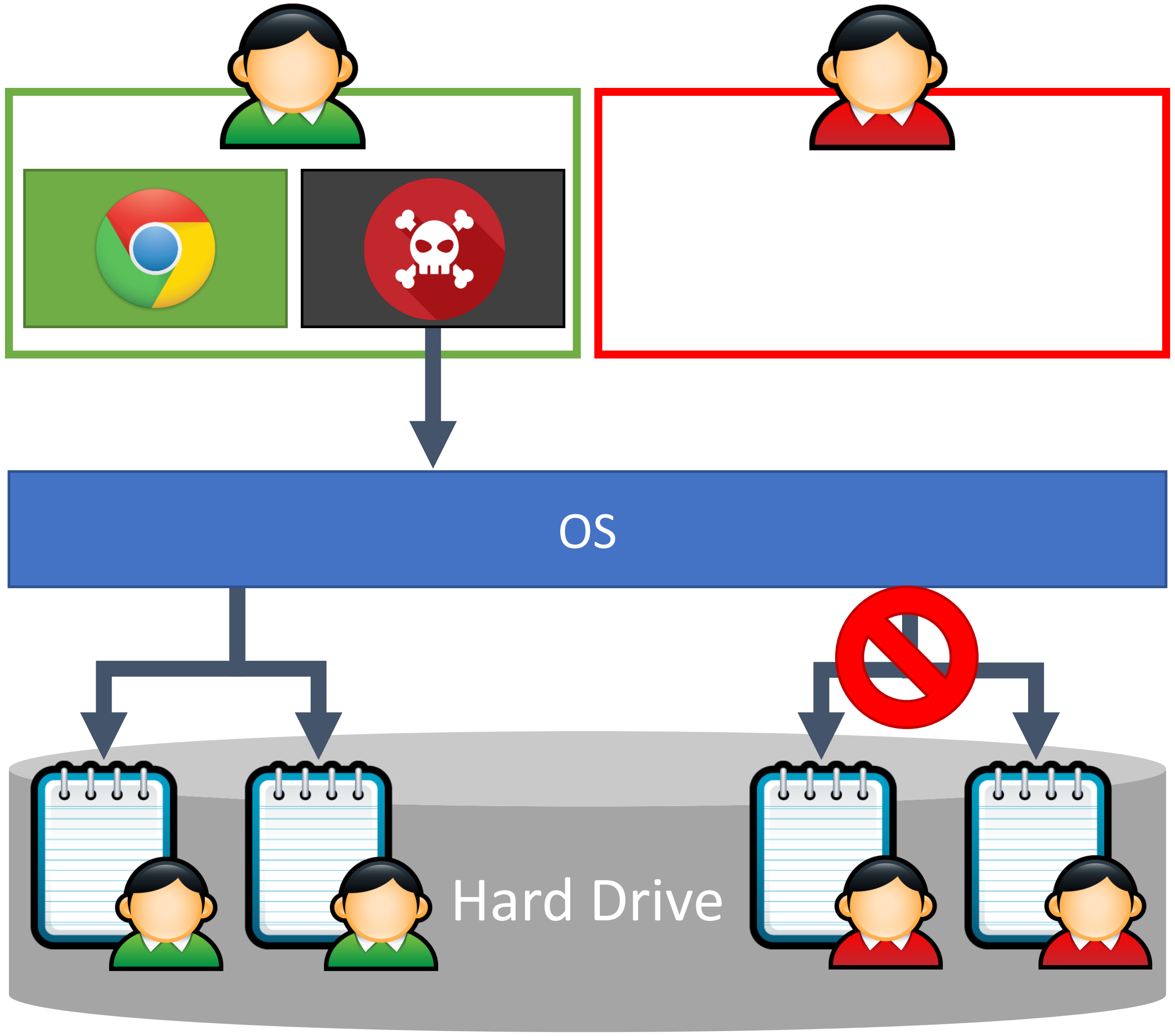
OS enforces access controls



Limitations

Malware can still cause damage

Discretionary access control means that isolation is incomplete



Anti-virus

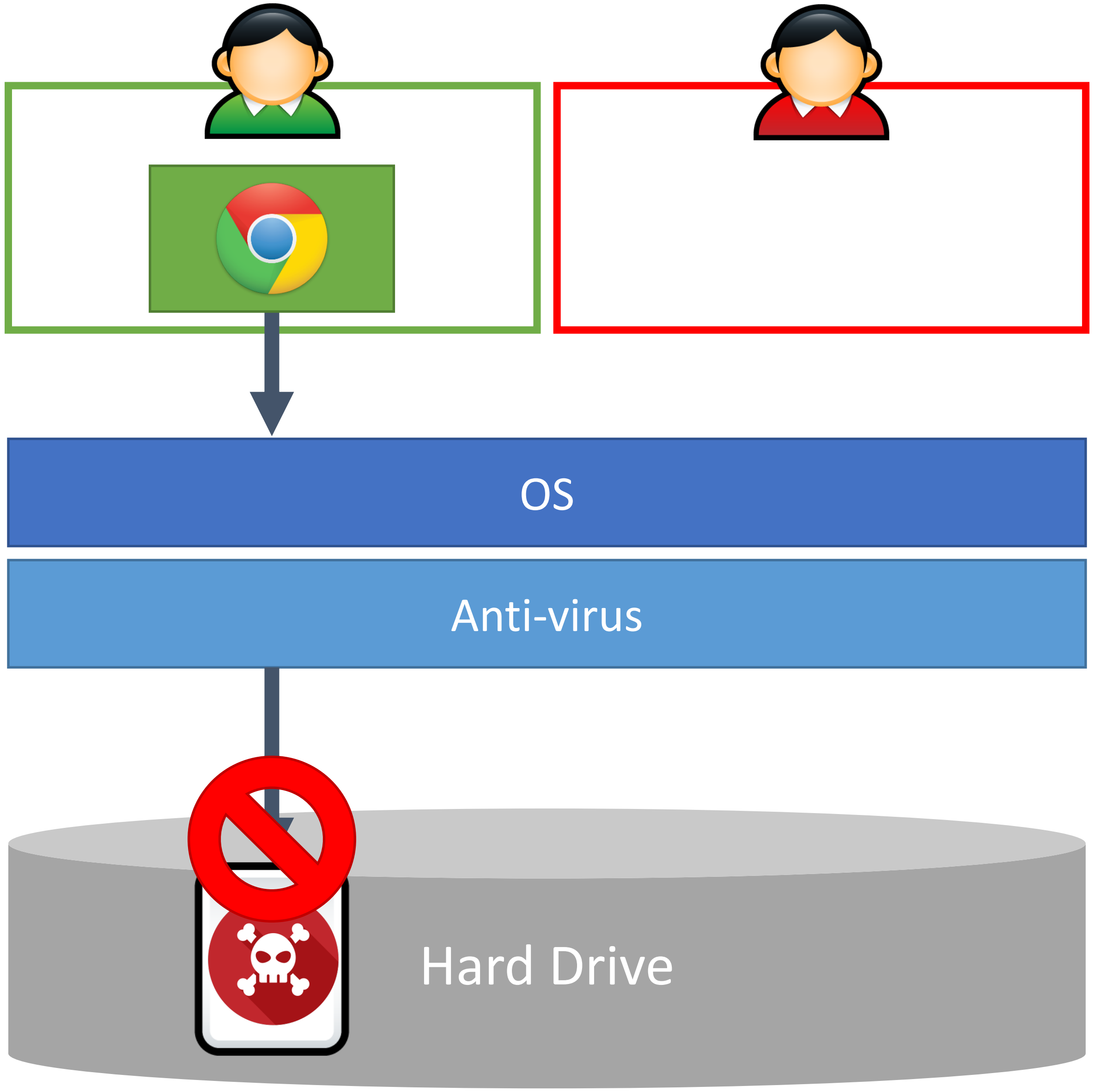
Anti-virus process is privileged

- 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

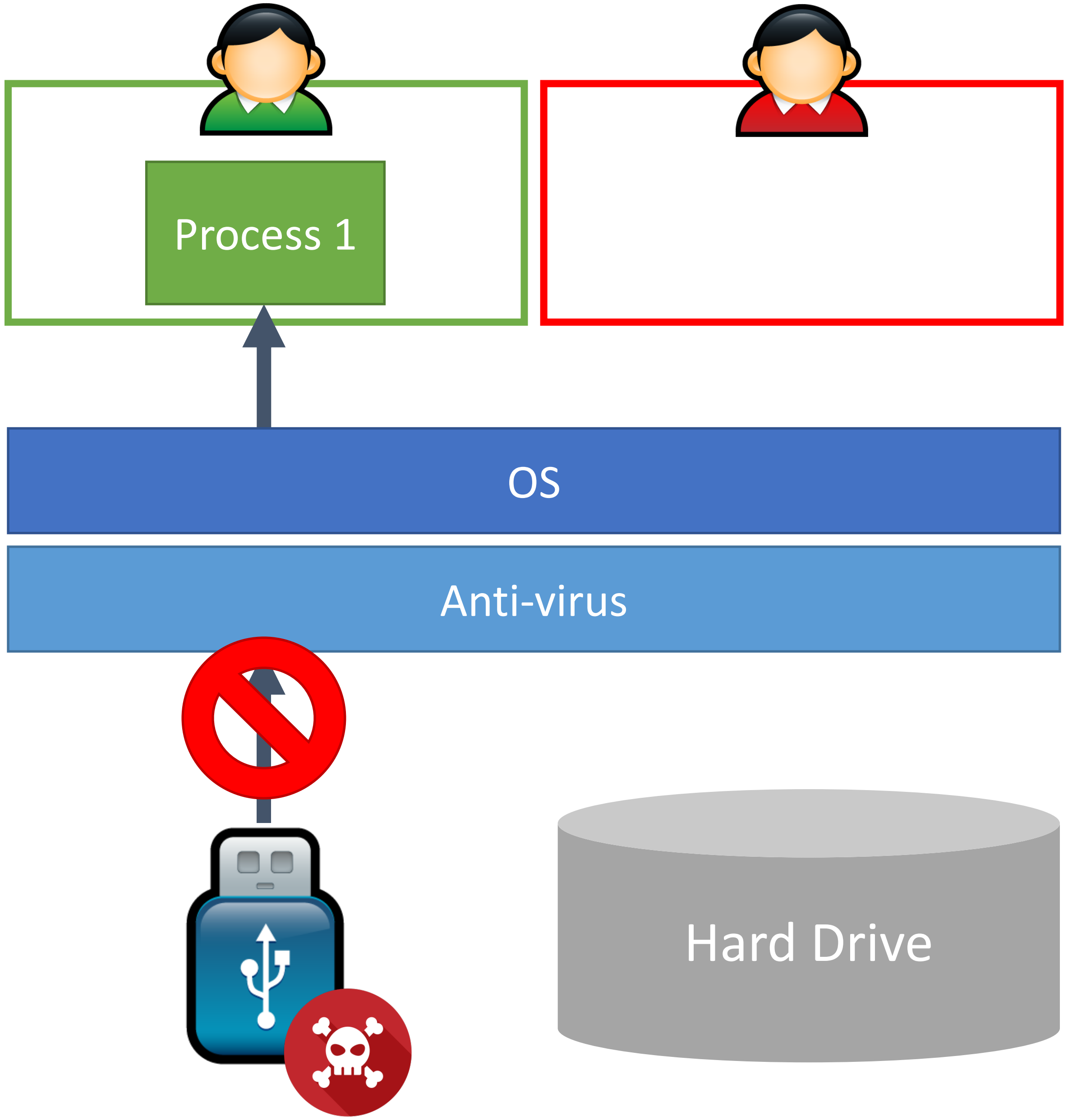
Anti-virus process is privileged

- 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



Example: Zeus Yara signature

```
rule Windows_Malware_Zeus : Zeus_1134
    {
            meta:
                    author = "Xylitol xylitol@malwareint.com"
                    date = "2014-03-03"
                    description = "Match first two bytes, protocol and string present in Zeus 1.1.3.4"
                    reference = "http://www.xylibox.com/2014/03/zeus-1134.html"

            strings:
                    $mz = {4D 5A}
                    $protocol1 = "X_ID: "
                    $protocol2 = "X_OS: "
                    $protocol3 = "X_BV: "
                    $stringR1 = "InitializeSecurityDescriptor"
                    $stringR2 = "Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; SV1)"
            condition:
                    ($mz at 0 and all of ($protocol*) and ($stringR1 or $stringR2))
    }
```

Example: Cryptolocker Yara signature

```
rule CryptoLocker_set1
{
meta:
        author = "Christiaan Beek, Christiaan_Beek@McAfee.com"
        date = "2014-04-13"
        description = "Detection of Cryptolocker Samples"

strings:
        $string0 = "static"
        $string1 = " kscdS"
        $string2 = "Romantic"
        $string3 = "CompanyName" wide
        $string4 = "ProductVersion" wide
        $string5 = "9%9R9f9q9"
        $string6 = "IDR_VERSION1" wide
        $string7 = "  </trustInfo>"
        $string8 = "LookFor" wide
        $string9 = ":n;t;y;"
        $string10 = "        <requestedExecutionLevel level"
        $string11 = "VS_VERSION_INFO" wide
        $string12 = "2.0.1.0" wide
        $string13 = "<assembly xmlns"
        $string14 = "  <trustInfo xmlns"
        $string15 = "srtWd@@"
        $string16 = "515]5z5"
        $string17 = "C:\\lZbvnoVe.exe" wide
condition:
        12 of ($string*)
}
```

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

- Zero-day malware: malware for which signatures are not available (not yet known and analyzed)

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.

Evasion: 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

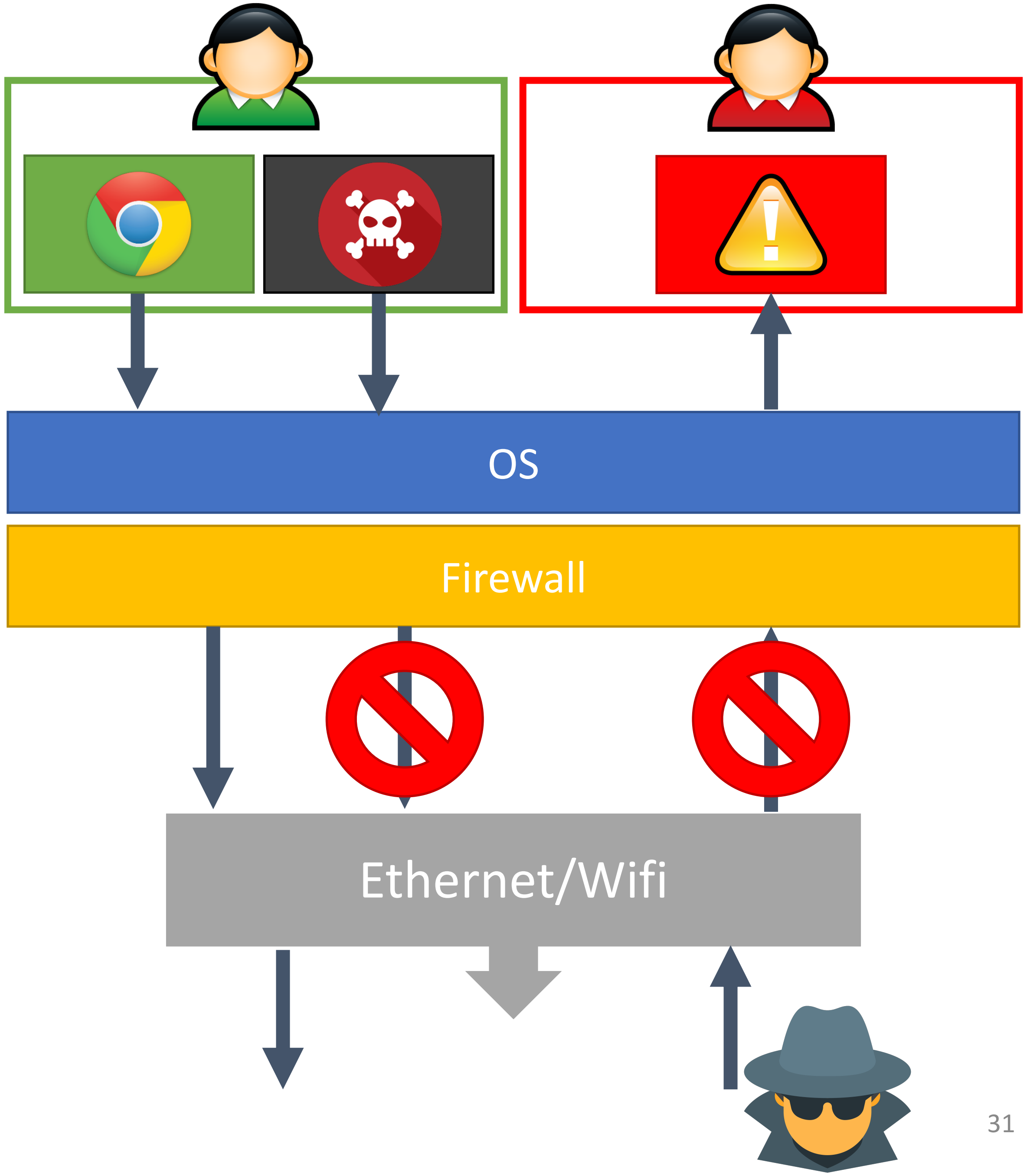
Firewall process is privileged

- 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



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 / Zeek

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

Systems Security

Threat Model
Intro to Computer Architecture
Hardware Support for Isolation
Security Technologies
Principles

Security Principles

At this point, we've explored the basics of secure systems architecture

- Device and memory isolation
- Basis for all higher-level functionality

But, designing secure systems (and breaking them) remains an art

Security principles help bridge the gap between art and science

- Developed by Saltzer and Schroeder
- "The Protection of Information in Computer Systems", 1975

Example

Built-in security features of Windows 10

- Secure boot: cryptographically verified bootup process
- Bitlocker full-drive encryption
- Kernel protections, e.g. Address Space Layout Randomization (ASLR)
- Cryptographic signing for device drivers
- User authentication
- User Account Control: permission check for privileged operations
- Anti-virus and anti-malware
- Firewall
- Automated patching
- System logs

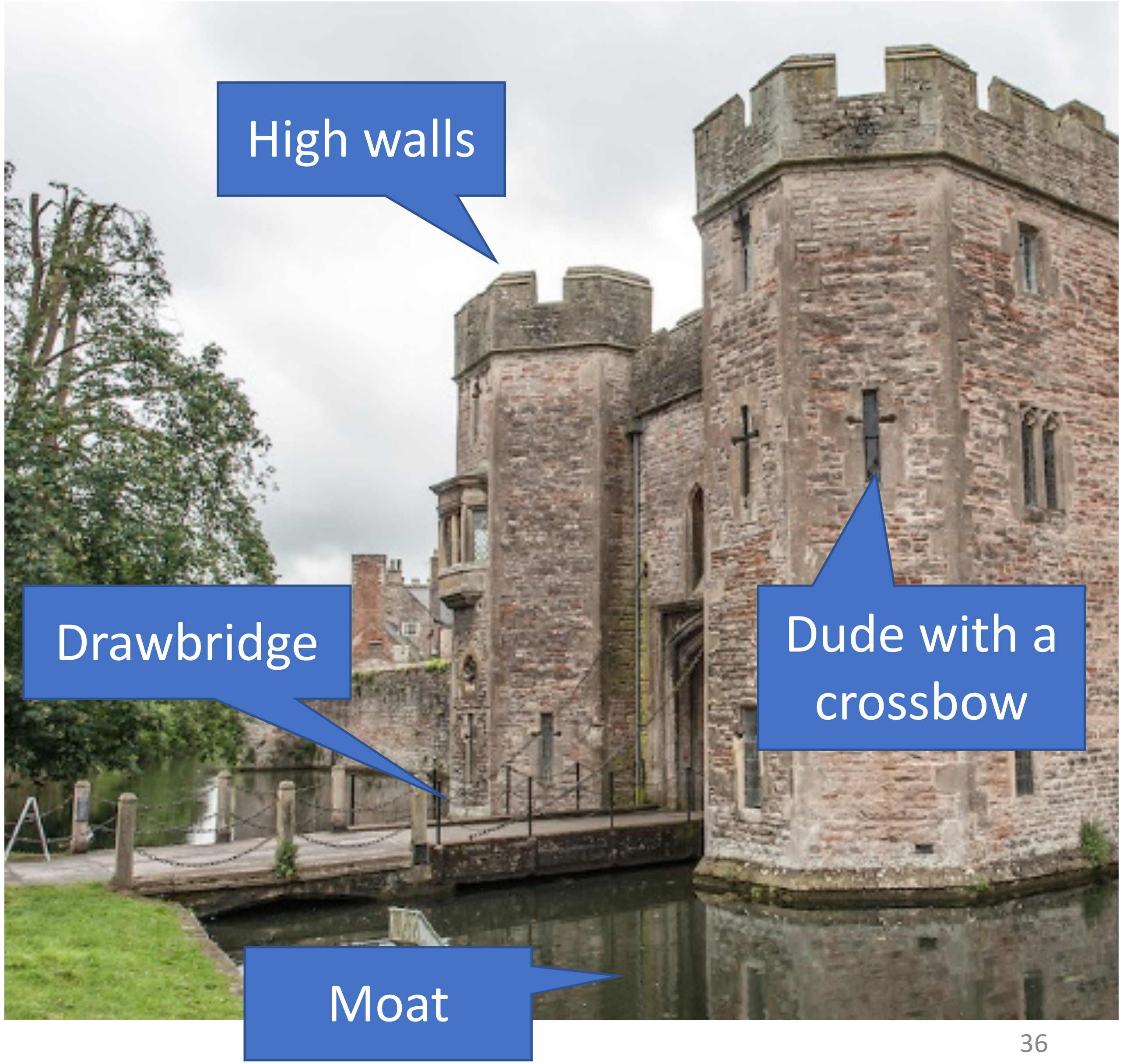
Defense in Depth

Don't depend on a single protection mechanism, since they are apt to fail

Even very simple or formally verified defenses fail

Layering defenses increases the difficulty for attackers

Defenses should be complementary!



Principles Overview

1. Fail-safe Defaults
2. Separation of Privilege
3. Least Privilege
4. Open Design
5. Economy of Mechanism
6. Complete Mediation
7. Compromise Recording
8. Work Factor

Principle 1: Fail-safe Defaults

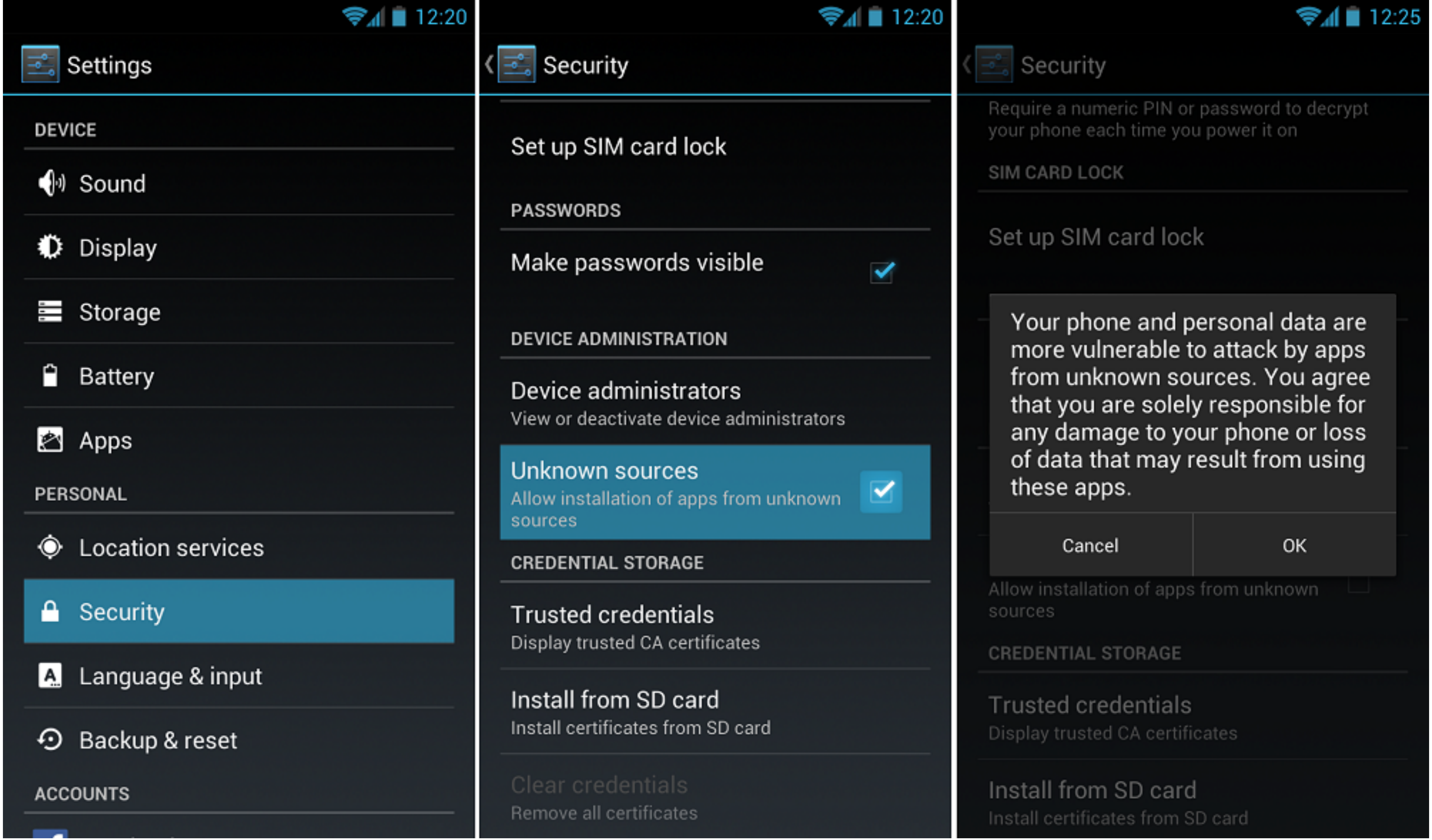
The absence of explicit permission is equivalent to no permission

Systems should be secure "out-of-the-box"

- Most users stick with defaults
- Users should "opt-in" to less-secure configurations

Examples. By default…

- New user accounts do not have admin or root privileges
- New apps cannot access sensitive devices
- Passwords must be >8 characters long
- Etc.



12:20
Settings
DEVICE
Sound
Display
Storage
Battery
Apps
PERSONAL
Location services
Security
Language & input
Backup & reset
ACCOUNTS
12:20
Security
Set up SIM card lock
PASSWORDS
Make passwords visible
DEVICE ADMINISTRATION
Device administrators
View or deactivate device administrators
Unknown sources
Allow installation of apps from unknown sources
CREDENTIAL STORAGE
Trusted credentials
Display trusted CA certificates
Install from SD card
Install certificates from SD card
Clear credentials
Remove all certificates
12:25
Security
Require a numeric PIN or password to decrypt your phone each time you power it on
SIM CARD LOCK
Set up SIM card lock
Your phone and personal data are more vulnerable to attack by apps from unknown sources. You agree that you are solely responsible for any damage to your phone or loss of data that may result from using these apps.
Cancel
OK
Allow installation of apps from unknown sources
CREDENTIAL STORAGE
Trusted credentials
Display trusted CA certificates
Install from SD card
Install certificates from SD card

Principle 2: Separation of Privilege

Privilege, or authority, should only be distributed to subjects that require it

Some components of a system should be less privileged than others

- Not every subject needs the ability to do everything
- Not every subject is deserving of full trust

- Examples
  - Not every user should have access to all enterprise machines
  - Should use a different admin account for every machine

Principle 3: Least Privilege

Subjects should possess only that authority that is required to operate successfully

Closely related to *separation of privilege*

Not only should privilege be separated, but subjects should have the least amount necessary to perform a task

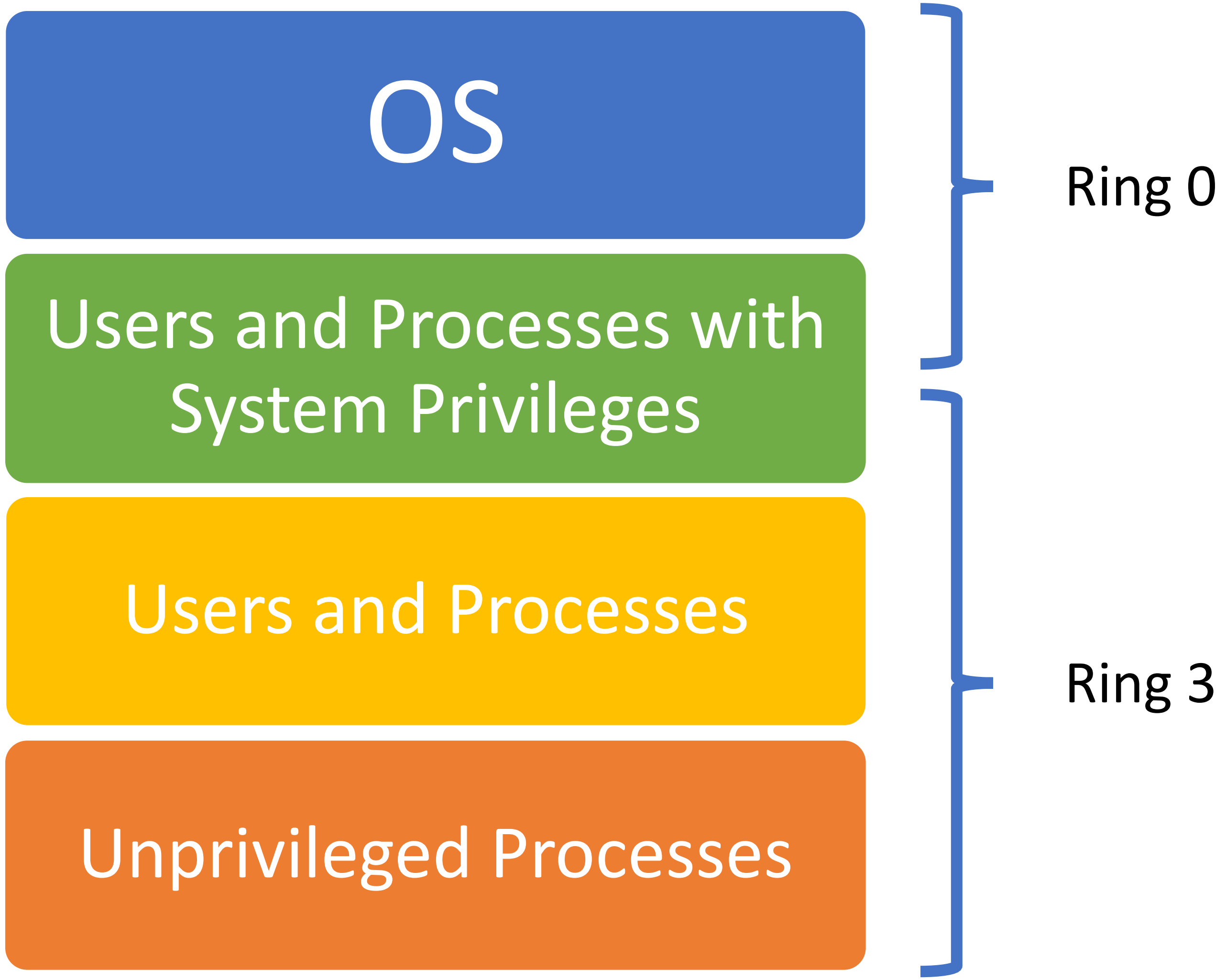
Examples

- Do not use sudo if command can be executed without
- Mobile apps should only have the permissions they need

Privilege Over Time

Privilege Hierarchy

- Device drivers, kernel modules, etc.
- sudo, “administrator” accounts, OS services
- Everything that is isolated and subject to access control
- chroot jails, containers



Principle 4: Open Design

Kerckhoff's Principle: A cryptosystem should be secure even if everything about the system, except the key, is public knowledge

Generalization: A system should be secure even if the adversary knows everything about its design

- Design does not include runtime parameters like secret keys

Contrast with “security through obscurity”

Examples:

- Crypto algorithm is known
- Authentication method is known
- Attacker knows network topology

Principle 5: Simplicity of Design

Also called “Economy of Mechanism”

Simplicity of design implies a smaller attack surface

Correctness of protection mechanisms is critical

- We need to be able to trust our security mechanisms
- Easier to verify and trust simpler design

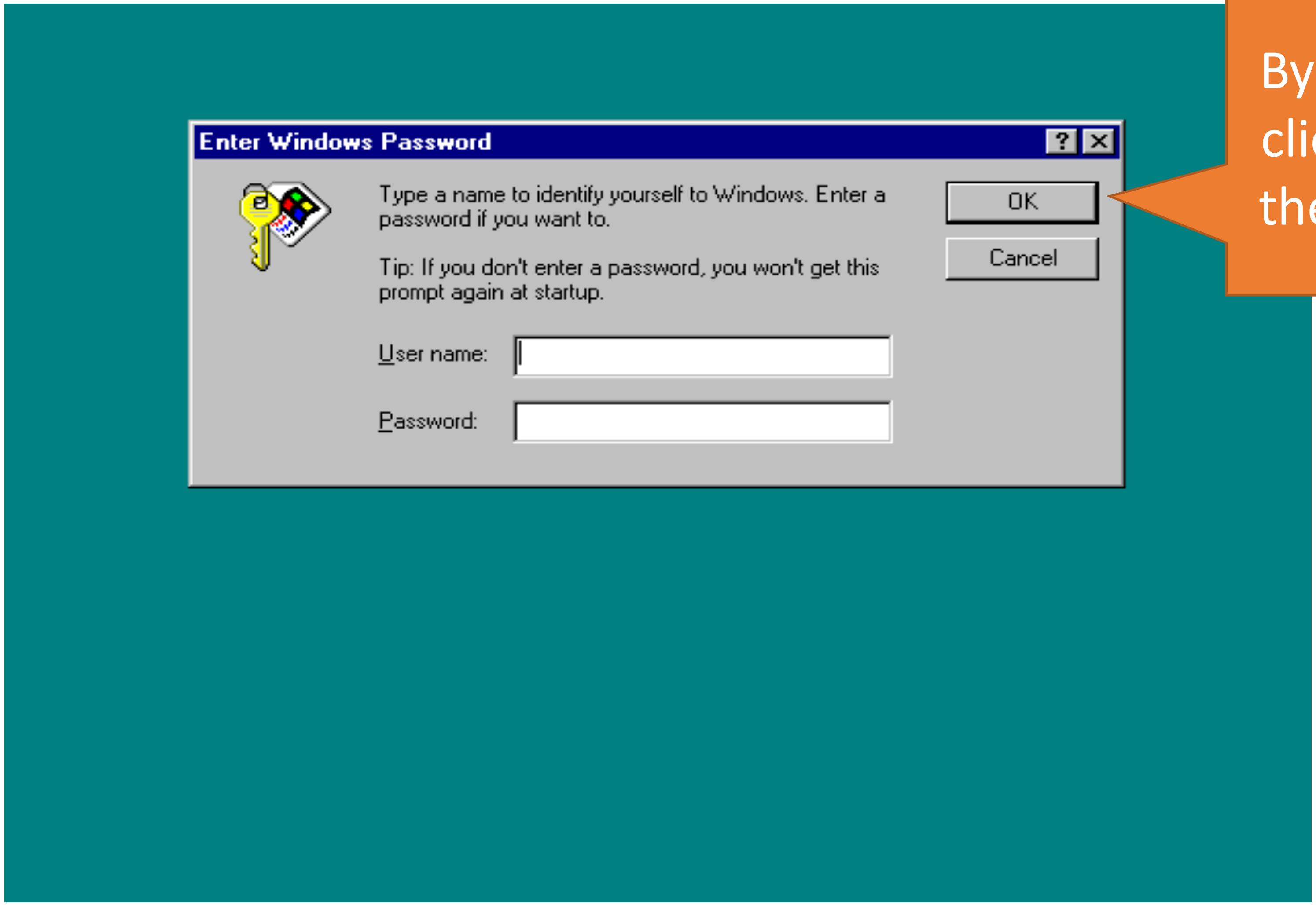
Principle 6: Complete Mediation

Every access to every object must be checked for authorization

Incomplete mediation implies that a path exists to bypass a security mechanism

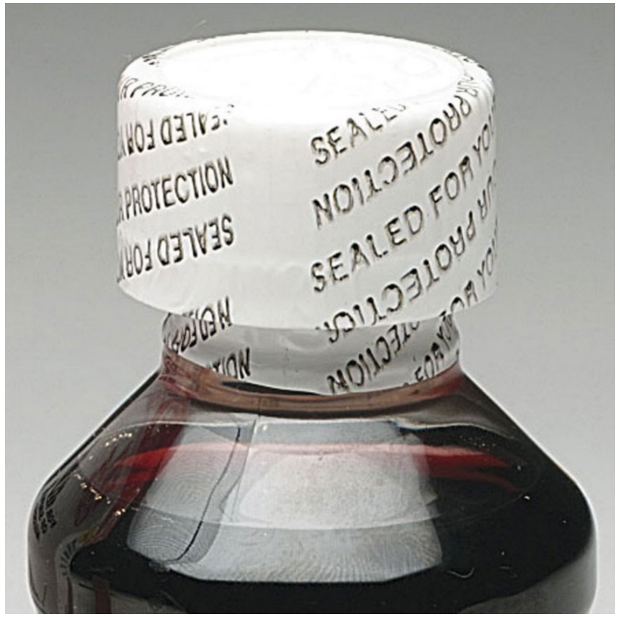
In other words, isolation is incomplete

Example



By default, user could click Cancel to bypass the password check :(

Principle 7: Compromise Recording



Concede that attacks will occur, but record the fact

Auditing approach to security

- Detection and recovery

"Tamper-evident" vs. "tamper-proof"

Principle 8: Work Factor

Increase the difficulty of mounting attacks

Sometimes utilizes non-determinism

- e.g. increasing entropy used in ASLR

Sometimes utilizes time

- Increase the lengths of keys
- Wait times after failed password attempts