CY 2550 Foundations of Cybersecurity

Systems Security

Alina Oprea
Associate Professor, Khoury College
Northeastern University
March 23 2020

Outline

- Mandatory Access Control (MAC)
  - Bell LaPadula Multi-Level Security
  - Biba Integrity Model
- System Security
  - Threat Model
  - Intro to Computer Architecture
  - Hardware Support for Isolation
  - Security Technologies
  - Design Principles

Access Control Models

- Discretionary Access Control (DAC)
  - The kind of access control you are familiar with
  - Most widely deployed (Windows, Unix)
  - Access rights propagate and may be changed at subject's discretion
  - Owner of resource controls the access rights for the resource

- Mandatory Access Control (MAC)
  - Access of subjects to objects is based on a system-wide policy
  - Global policy controlled by system administrator
  - Might deny users full control over resources they create

Mandatory Access Control

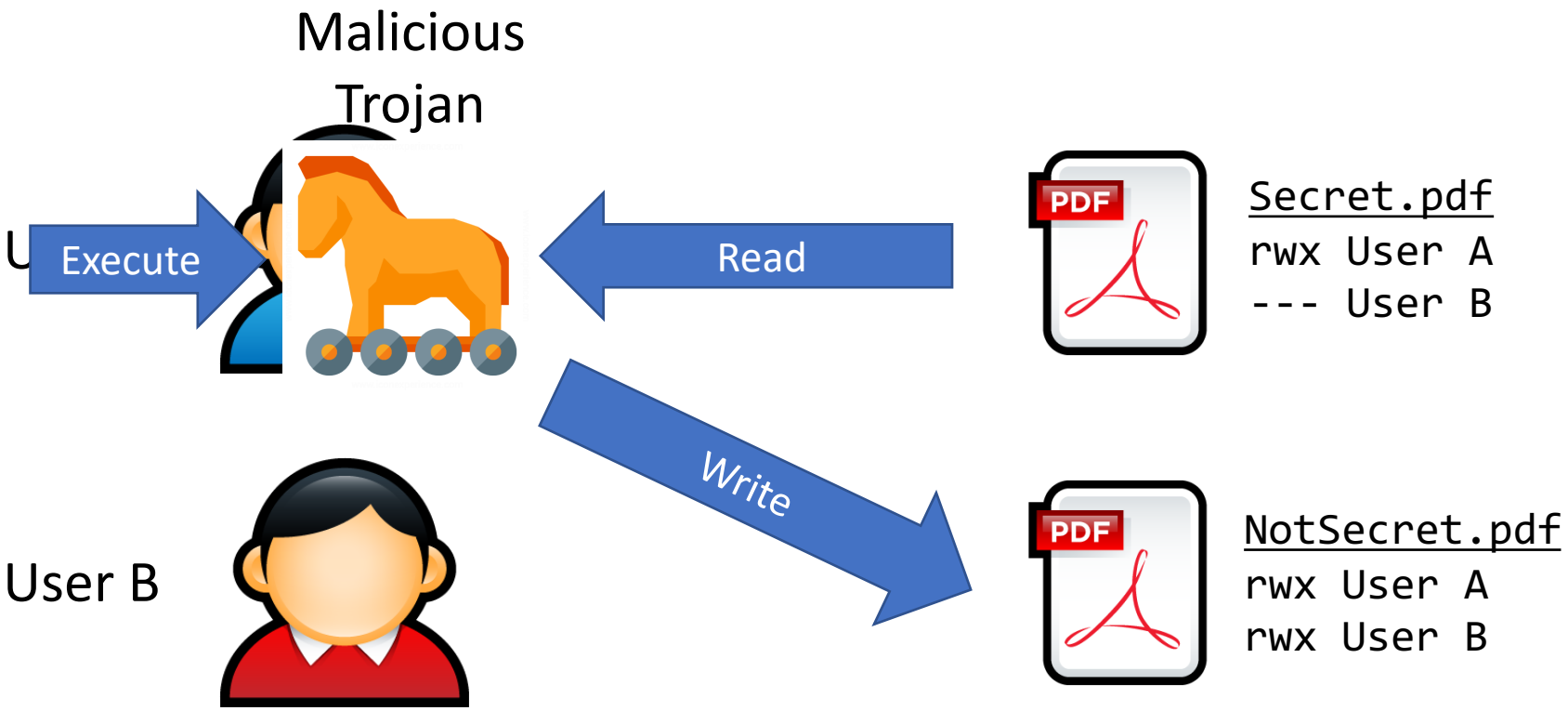
Multi-level Security

Bell-LaPadula Model

Biba Model

Failure of DAC

- DAC cannot prevent the leaking of secrets



Why is DAC Vulnerable?

- Implicit assumptions
  - Software is benign
  - Software is bug free
  - Users are well behaved
- Reality
  - Software is full of bugs (e.g., confused deputies)
  - Malware is widely available
  - Users may be malicious (insider threats)

Towards Mandatory Access Control (MAC)

- Mandatory access controls (MAC) restrict the access of subjects to objects based on a system-wide policy
  - System security policy (as set by the administrator) entirely determines access rights
  - Denying users full control over to resources that they create
- Often used in systems that must support Multi-level Security (MLS)
  - Define security labels on subjects and objects
  - System-wide policy uses security labels
- Implemented in SELinux and AppArmor for Linux

Multi-level Security (MLS)

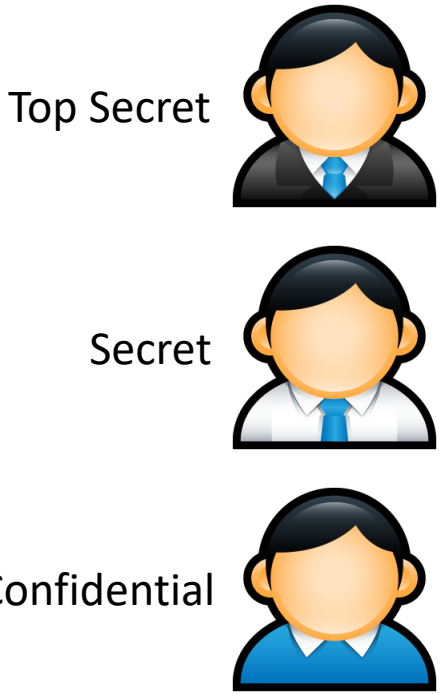
- The capability of a computer system to carry information with different sensitivities
  - Permit simultaneous access by users with different security clearances
  - Prevent users from obtaining access to information for which they lack authorization
- Examples of security levels
  - Top Secret > Secret > Confidential > Unclassified
- Overall goal is confidentiality
  - Ensure that information does not flow to those not cleared for that level

Bell-LaPadula: A MAC Model for MLS

- Introduced in 1973
  - Extremely influential document
  - Introduced fundamental ideas for formally modeling security
- Air Force was concerned about data confidentiality in time-sharing systems
  - Old OS with many bugs
  - Accidental misuse by operators
  - Insider threats
- Goal: formally show that a computer system can securely process classified information

Elements of the Bell-LaPadula Model

Subjects
L(s) : level



A system is secure iff it obeys:

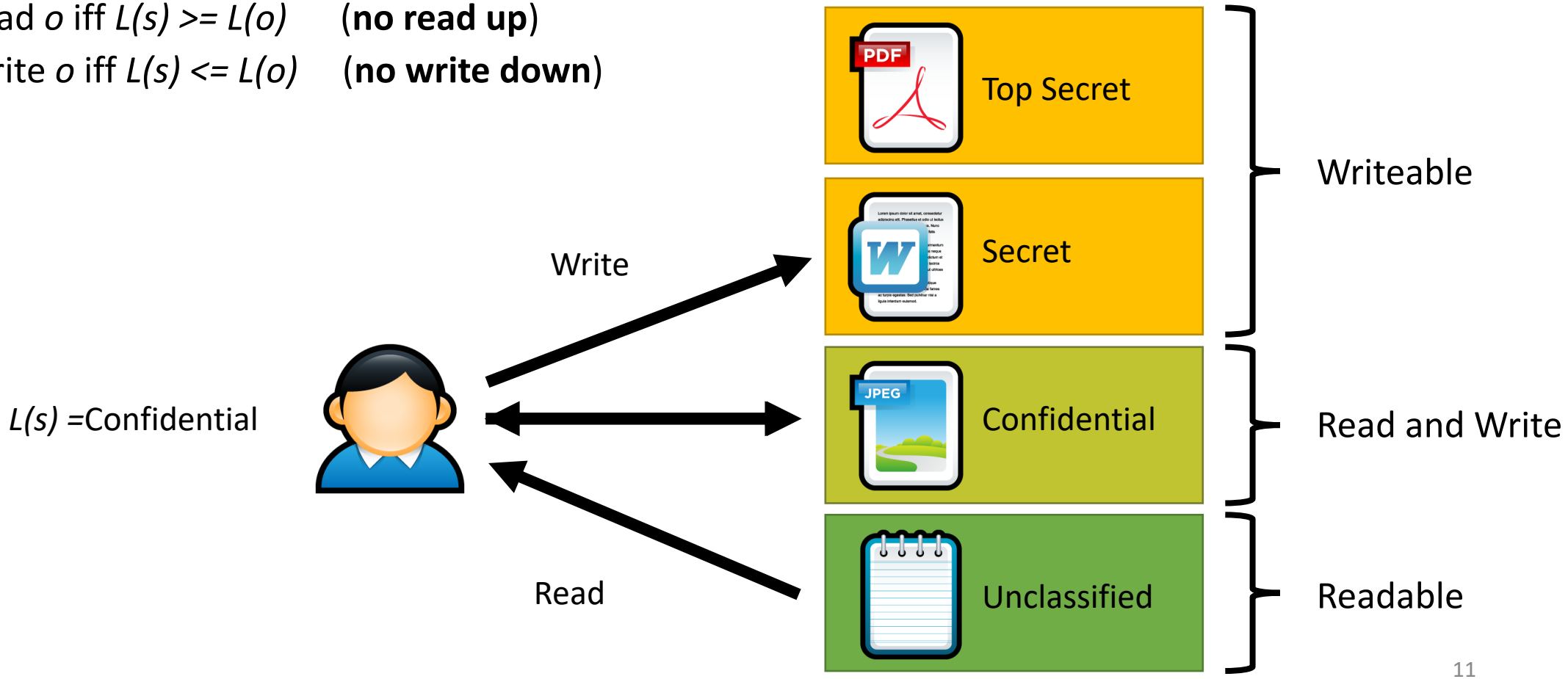
1. **No read up**
   s can read *o* iff *L(s) >= L(o)*

2. **No write down**
   s can write *o* iff *L(s) <= L(o)*

Objects
L(o) : level

Bell-LaPadula Example

- Properties
  - *s* can read *o* iff *L(s) >= L(o)* (**no read up**)
  - *s* can write *o* iff *L(s) <= L(o)* (**no write down**)



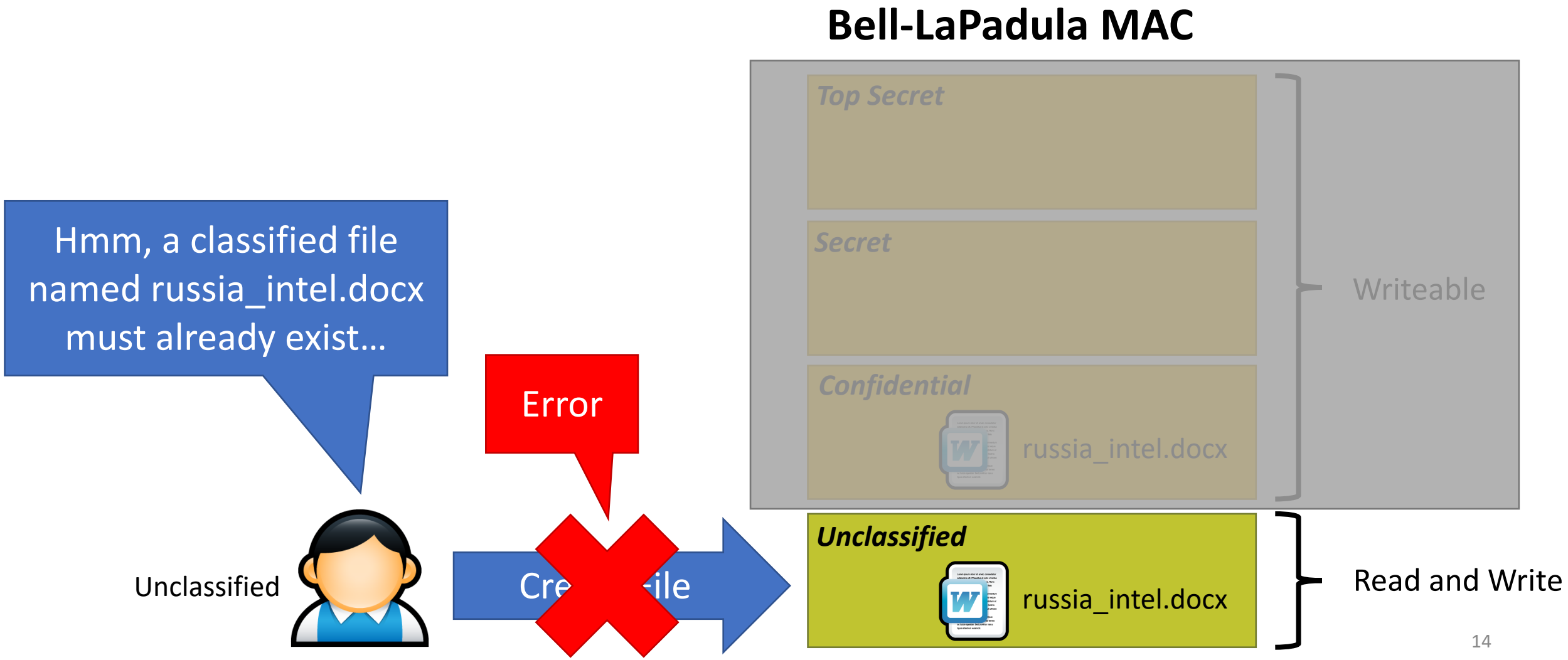
Caveats

- Security properties apply to subjects (programs), not principals (users)
  - Assume users won't disclose secrets outside of the computer system
- Security property does not address covert or side channels
  - Mechanisms that are not intended for communication
  - Information leakage channels
  - No evidence is left behind, so very difficult to detect
- Bell-LaPadula only addresses confidentiality
  - No integrity guarantees

Covert Channels

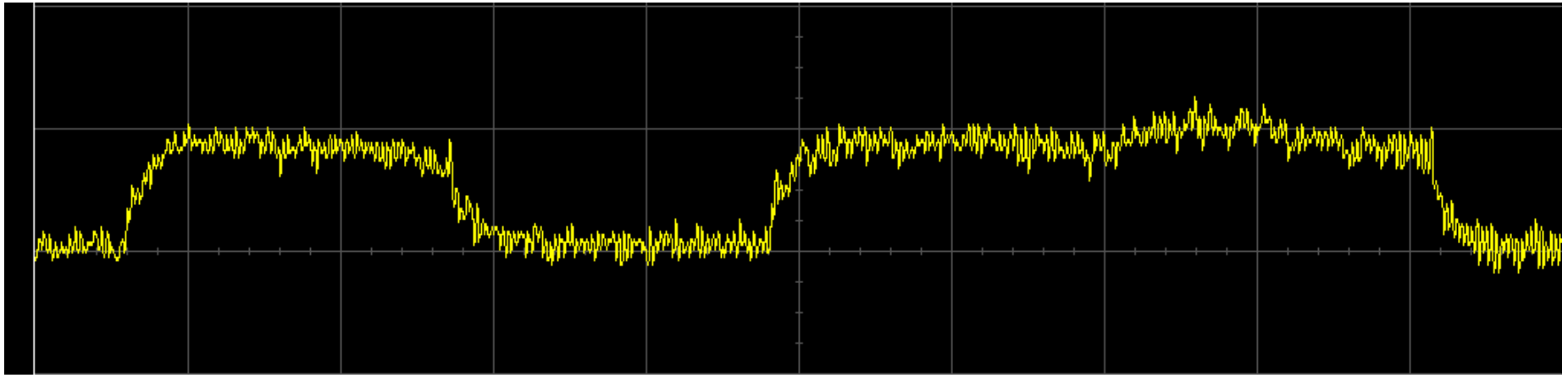
- Access control is defined over “legitimate” channels
  - Read/write an object
  - Send/receive a packet from the network
  - Read/write shared memory
- However, isolation in real systems is imperfect
  - Actions have observable side-effects
- External observations can create covert channels
  - Communication via unintentional channels
  - Examples:
    - Existence of file(s) or locks on file(s)
    - Measure the timing of events

Simple Example Covert Channel



Side Channel Attack Example

- Victim is decrypting RSA data
  - Key is not known to the attacker
  - Encryption process is not directly accessible to the attacker
- Attacker is logged on to the same machine as the victim
  - In cloud settings, resources such as the servers are shared
  - Secret key can be deciphered by observing the CPU voltage
  - Short peaks = no multiplication (0 bit), long peaks = multiplication (1 bit)



Covert channels vs side channels

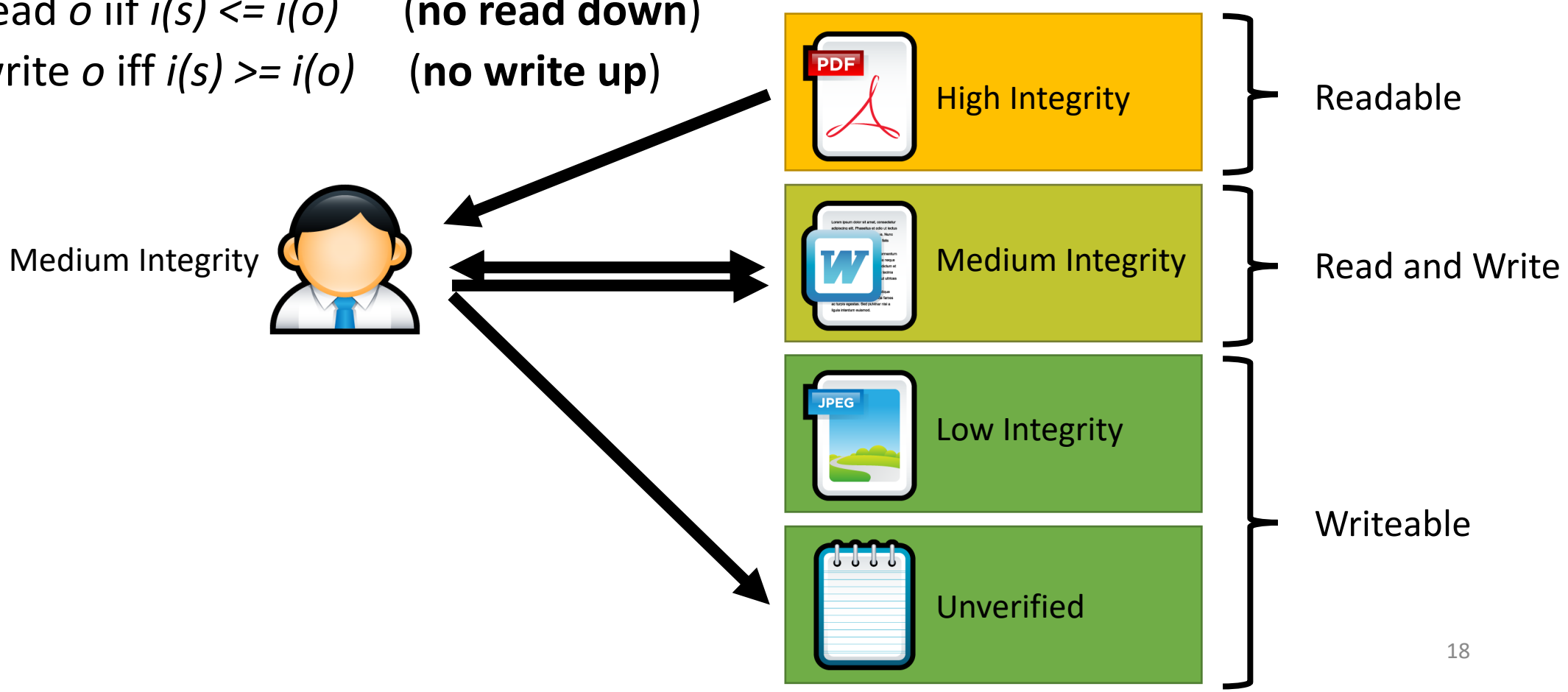
- Covert channels
  - Both parties / processes wish to communicate
  - Without the communication being observed / detected by others
  - Usually channel has low bandwidth
- Side channels
  - One process is victim, the other is attacker
  - Rise from systems implementation (timing of operation, etc.)
  - Very difficult to detect from victim's perspective
  - Shared resources (CPU caches) enable both types of channels
  - Famous recent examples: Spectre, Meltdown
    - Leverage use of speculative execution in modern processors
    - Can extract secrets from victim's memory

Biba Integrity Model

- Proposed in 1975
- Like Bell-LaPadula, security model with provable properties based on a state transition model
  - Each subject has an integrity level
  - Each object has an integrity level
  - Integrity levels are totally ordered (high → low)
- Integrity levels in Biba are not the same as security levels in Bell-LaPadula
  - Some high integrity data does not need confidentiality
  - Examples: stock prices, official statements from the president

Biba Strict Integrity Example

- Strict integrity
  - *s* can read *o* iif *i(s) <= i(o)* (**no read down**)
  - *s* can write *o* iff *i(s) >= i(o)* (**no write up**)



Practical Example of Biba Integrity

- Military chain of command
  - Generals may issue orders to majors and privates
  - Majors may issue orders to privates, but not generals
  - Privates may only take orders

Comparison

Bell-LaPadula

- Offers confidentiality
- “Read down, write up”
- Focuses on controlling reads
- Theoretically, no requirement that subjects be trusted
  - Even malicious programs can’t leak secrets they don’t know

Biba

- Offers integrity
- “Read up, write down”
- Focuses on controlling writes
- Subjects must be trusted
  - A malicious program can write bad information

Review Access Control

- Two main methods
  - DAC: ACL (Windows-style) or Linux style (3 levels of permissions per object)
  - MAC: Bell LaPadula (confidentiality), Biba (integrity)
- Main issues with DAC
  - Ambient authority (subjects inherit all permissions of principals)
  - Confused deputies (subject doesn't know which principal it serves)
  - Fixes: capability-based access control
    - Hardware and software implementations exist
    - Challenging to adopt in practice
- Main issues with MAC
  - Need to define security levels and implement a system-wide policy
  - Inflexible and complicated to manage
  - Does not prevent side channel attacks

Systems Security

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

Threat Modeling

Threat modeling is the process of systematically identifying the threats faced by a system

1. Identify things of value that you want to protect
2. Enumerate the attack surfaces
3. Hypothesize attackers and map them to
   - Things of value they want from (1)
   - Their ability to target vulnerable surfaces from (2)
4. Survey mitigations
5. Balance costs versus risks

Identify Things of Value

- Saved passwords
- Monetizable credentials (webmail, social networks)
- Access to bank accounts, paypal, venmo, credit cards, or other financial services
- Pics, messages, address book, browsing/search history (for blackmail)
- Sensitive business documents
- Access to sensors (camera, mic, GPS) or network traffic (for surveillance)
- The device itself
  - Steal it and sell it
  - Use the CPU and network for other criminal activity

Enumerate Attack Surfaces

- Steal the device and use it
- Social Engineering
  - Trick the user into installing malicious software
  - Spear phishing
- OS-level attacks
  - Backdoor the OS
  - Direct connection via USB
  - Exploit vulnerabilities in the OS or apps (e.g. email clients, web browsers)
- Network-level attacks
  - Passive eavesdropping on the network
  - Active network attacks (e.g. man-in-the-middle, SMS of death)

Cybercriminal

High-level goal: $$$ profit $$$

Immediate goal: running a process on a victim's computer

- Ransomware
- Botnet
- Spyware
- Adware

How to do this?

- Infected storage media (e.g. USB keys)
- Malicious attachments or downloads
- Exploits targeting the OS or common apps
- Guess or crack passwords for remote desktop, etc.

Mitigations

Authentication

- Physical and remote access is restricted

Access control

- Processes cannot read/write any file
- Users may not read/write each other's files arbitrarily
- Modifying the OS and installing software requires elevated privileges

Firewall

- Unsolicited communications from the internet are blocked
- Only authorized processes may send/receive messages from the internet

Anti-virus

- All files are scanned to identify and quarantine known malicious code

Logging

- All changes to the system are recorded
- Sensitive applications may also log their activity in the secure system log

Question: how do you build these mitigations?

In other words, how do you build secure systems?


IBM
IBM
IBM

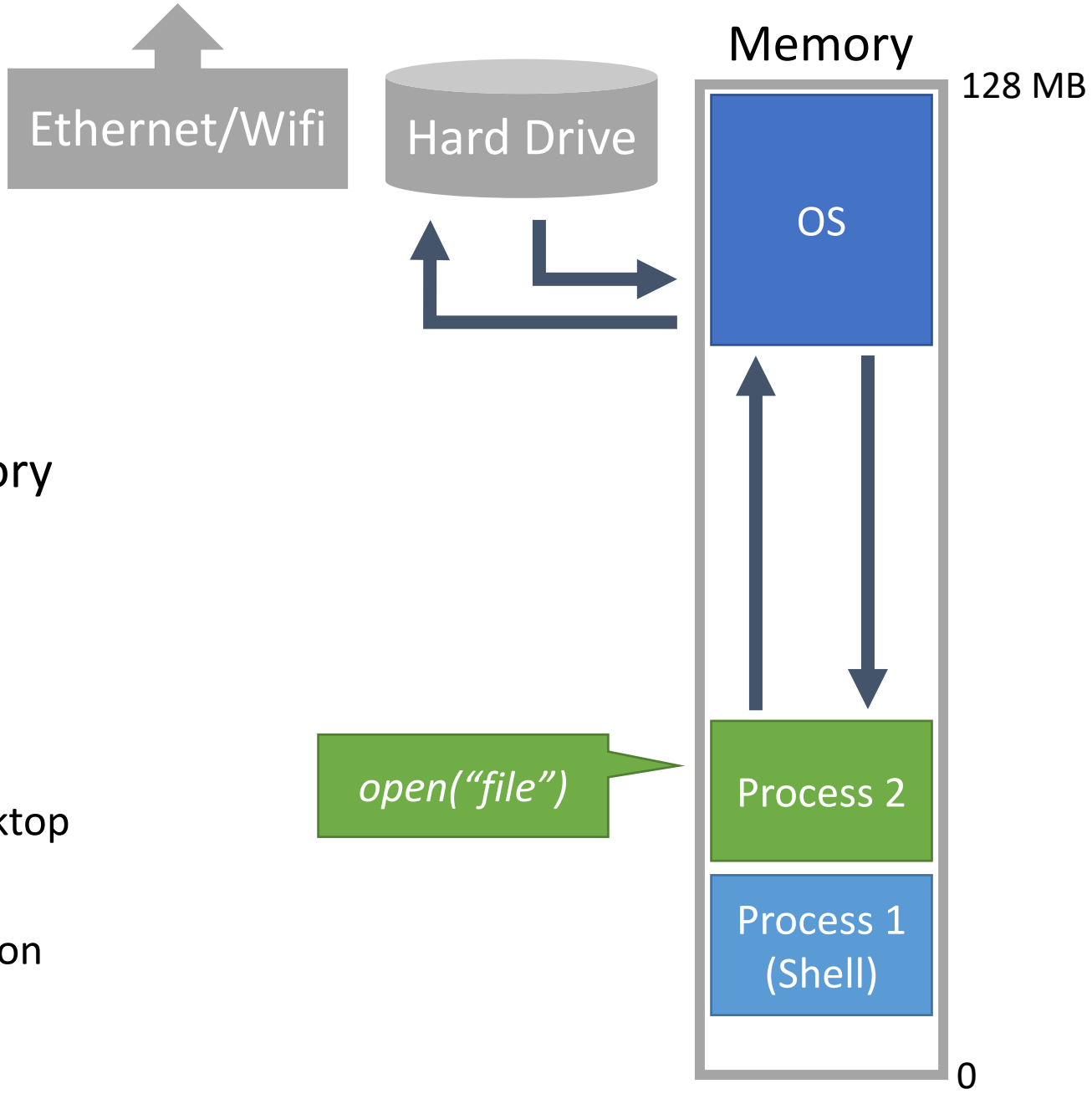
System Model

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

- DOS or Windows 3.1
- 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



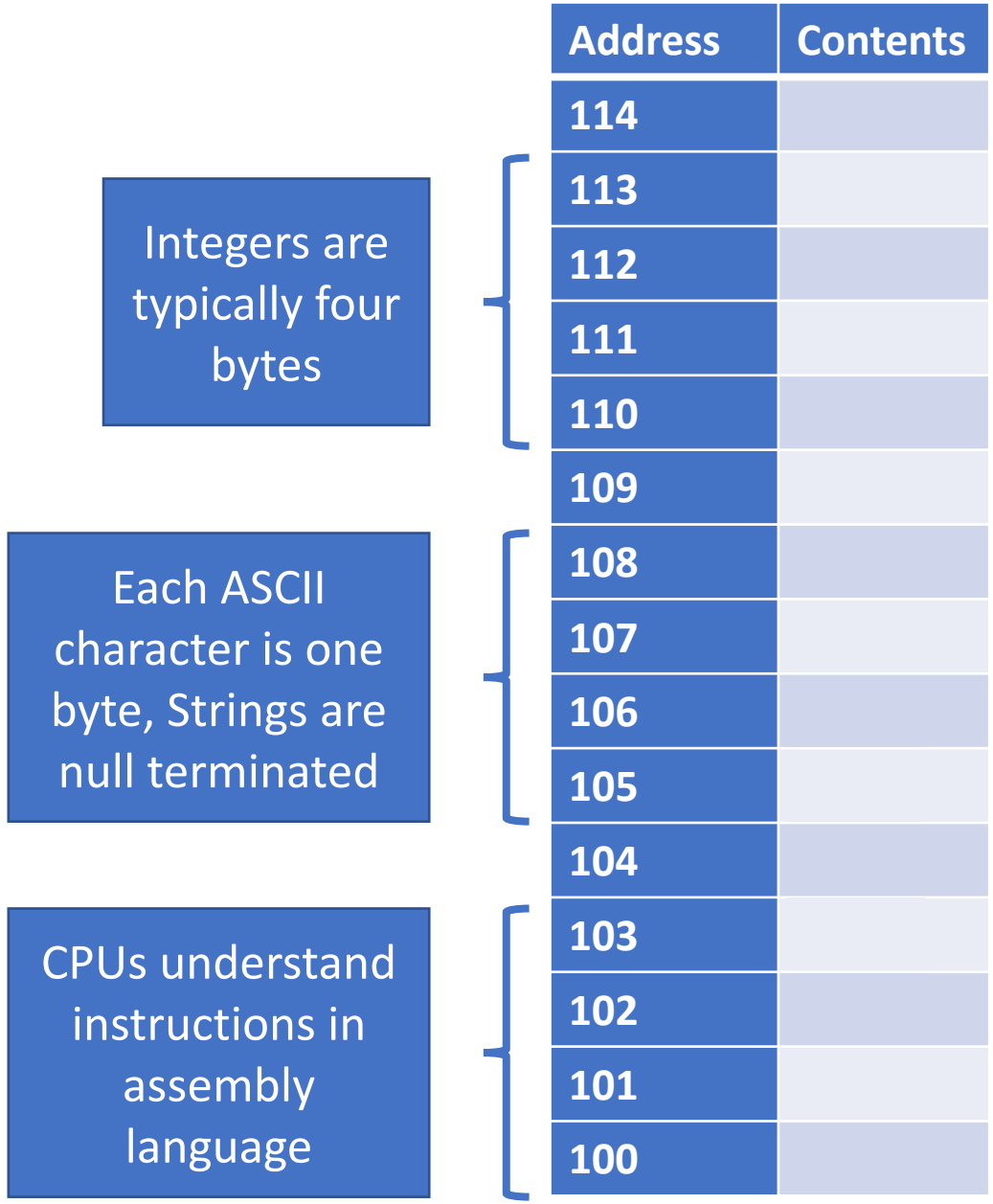
What is Memory?

Memory is essentially a spreadsheet with a single column

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

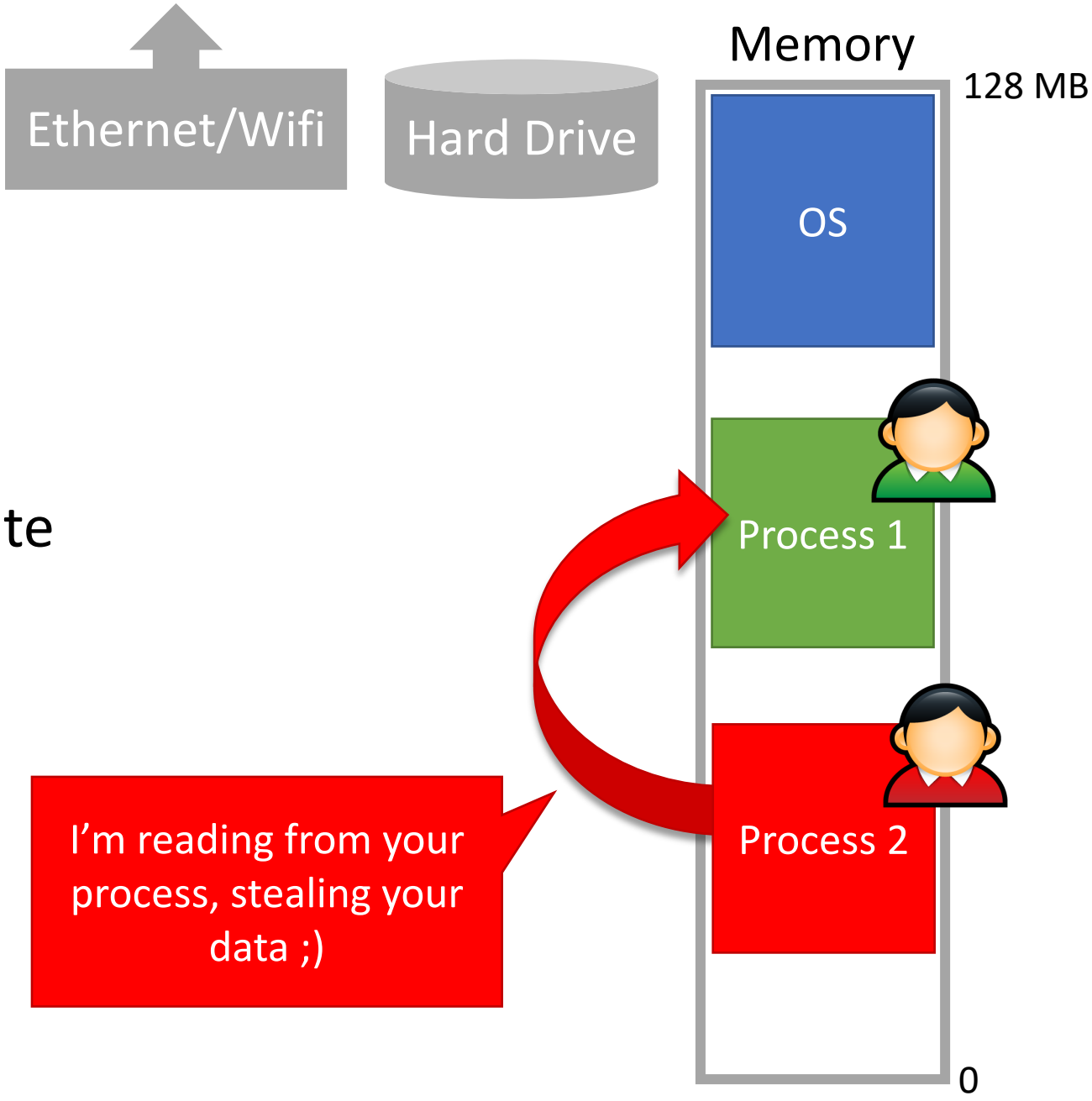
All data and running code are held in memory

```
int my_num = 8;
String my_str = "ABC";
while (my_num > 0) my_num--;
```



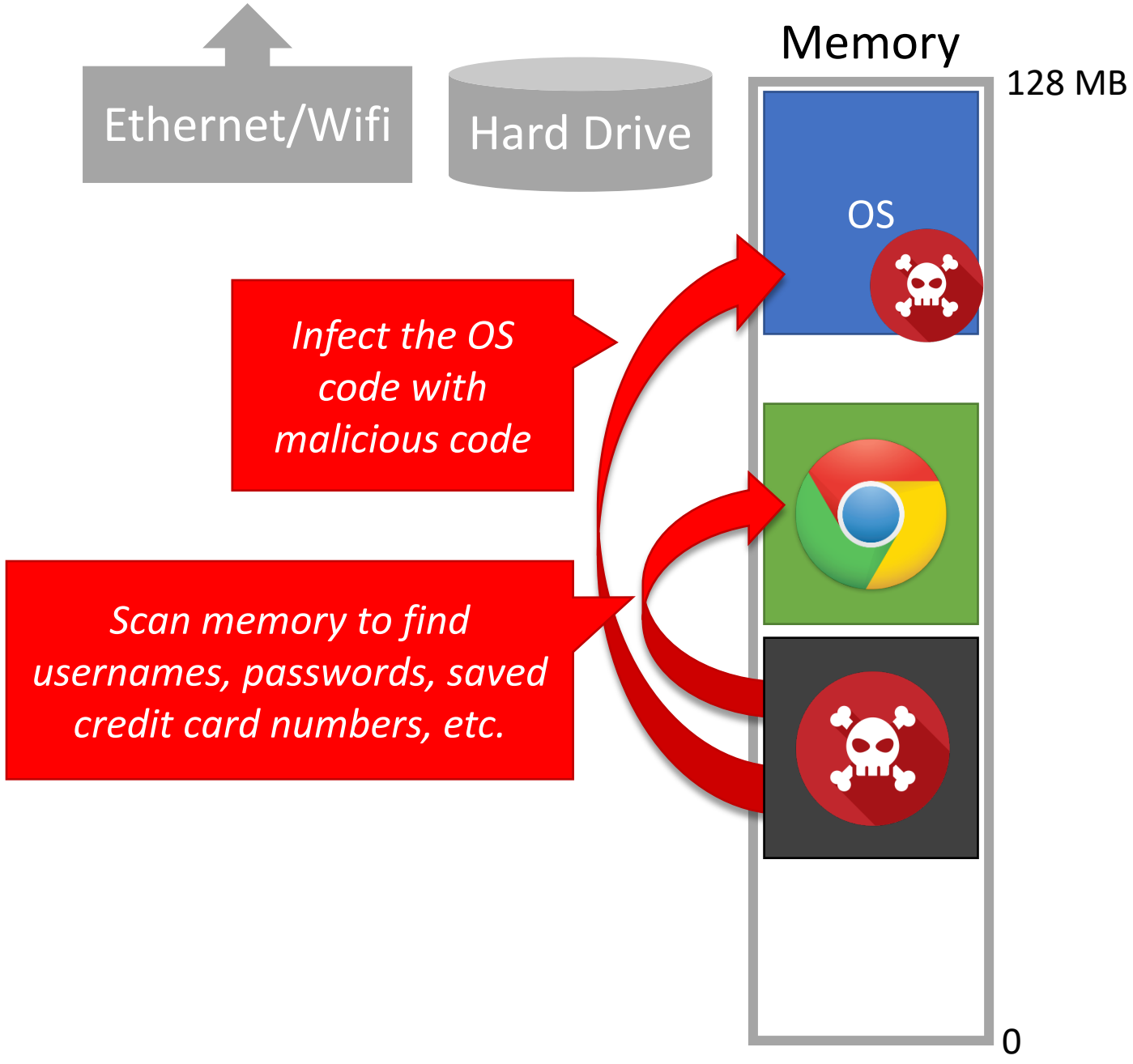
Memory Unsafety

Problem: any process can read/write any memory



Memory Unsafety

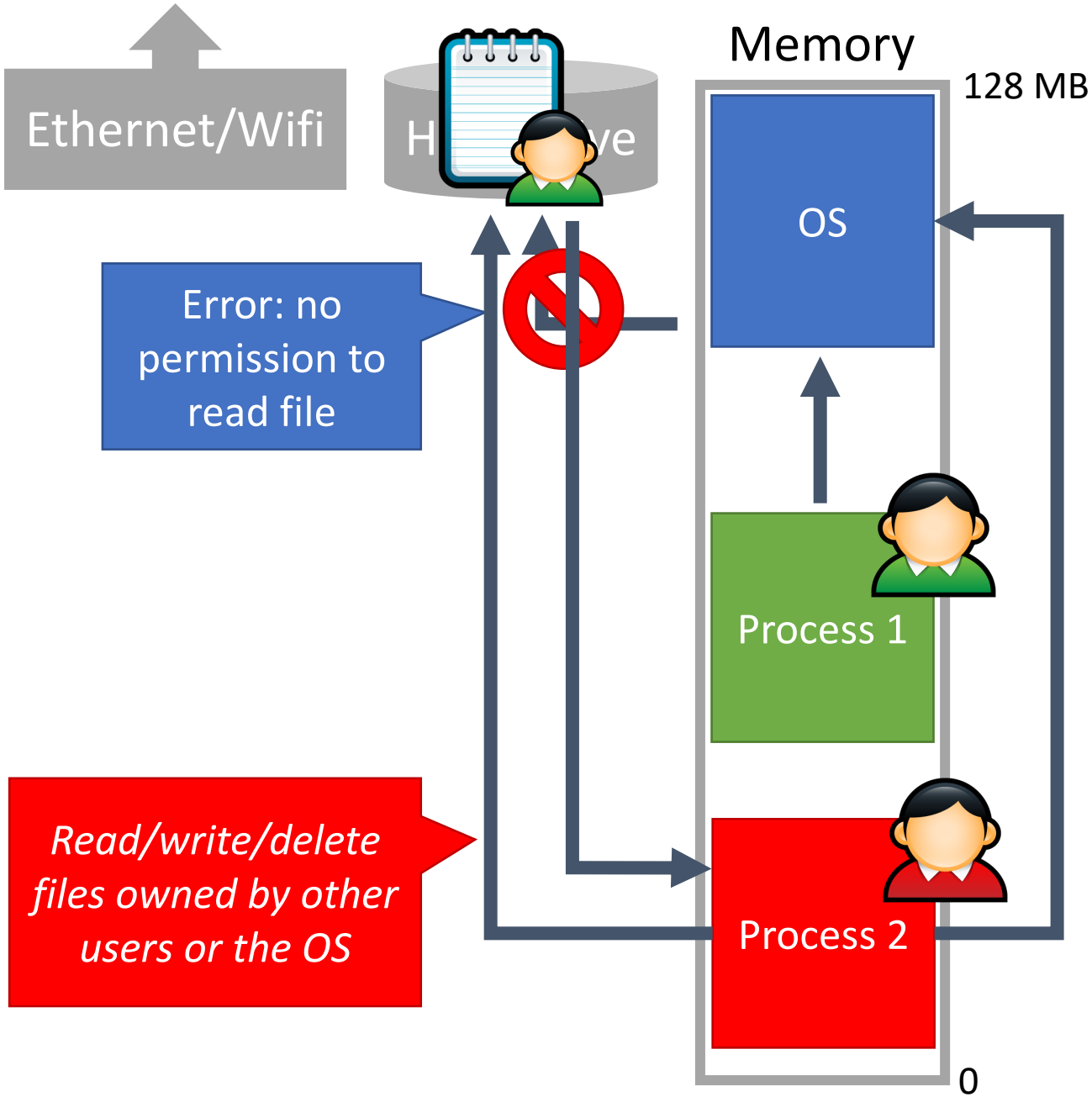
Problem: any process can read/write any memory



Device Unsafety

Problem: any process can access any hardware device directly

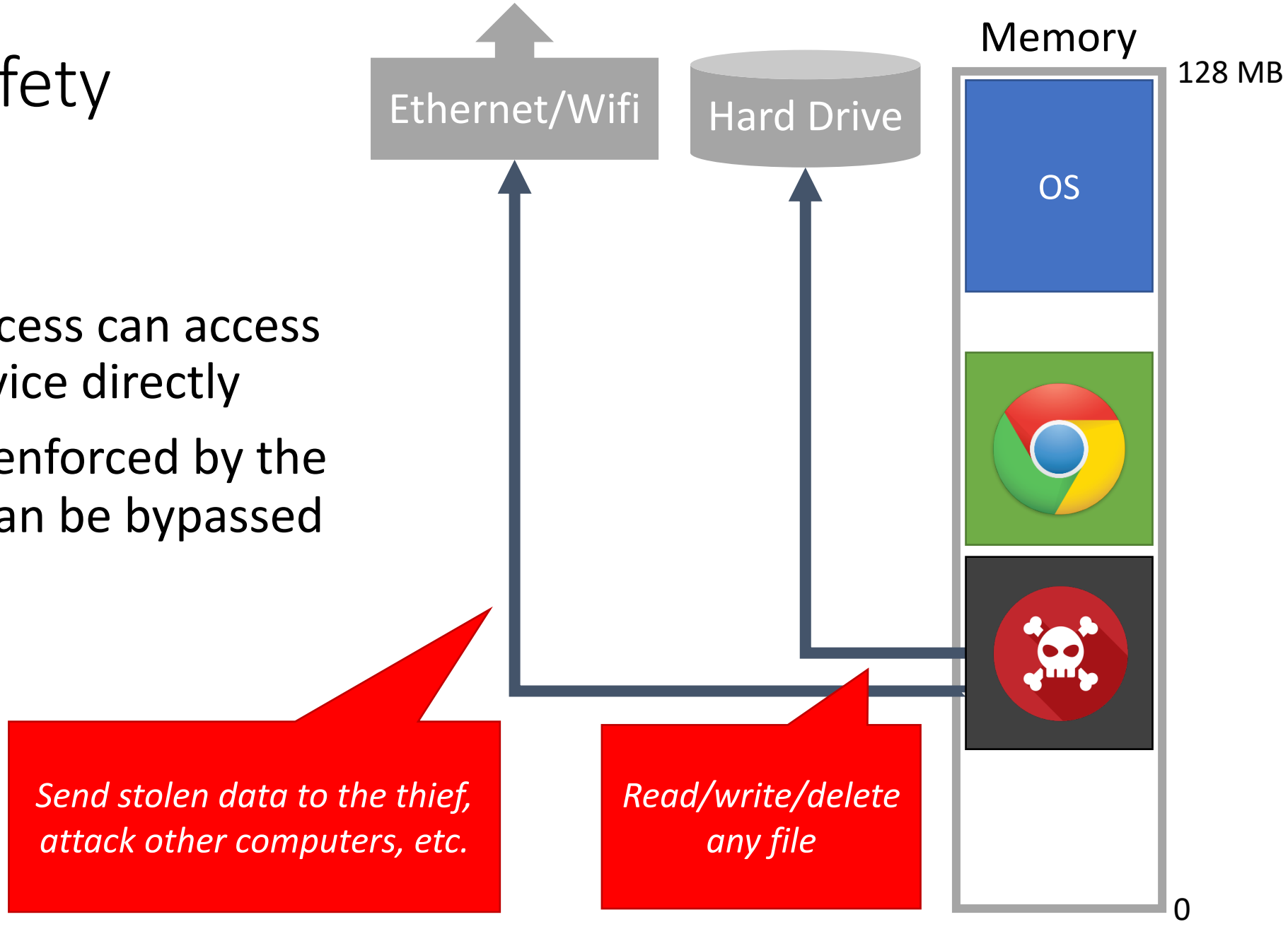
Access control is enforced by the OS, but OS APIs can be bypassed



Device Unsafety

Problem: any process can access any hardware device directly

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



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?