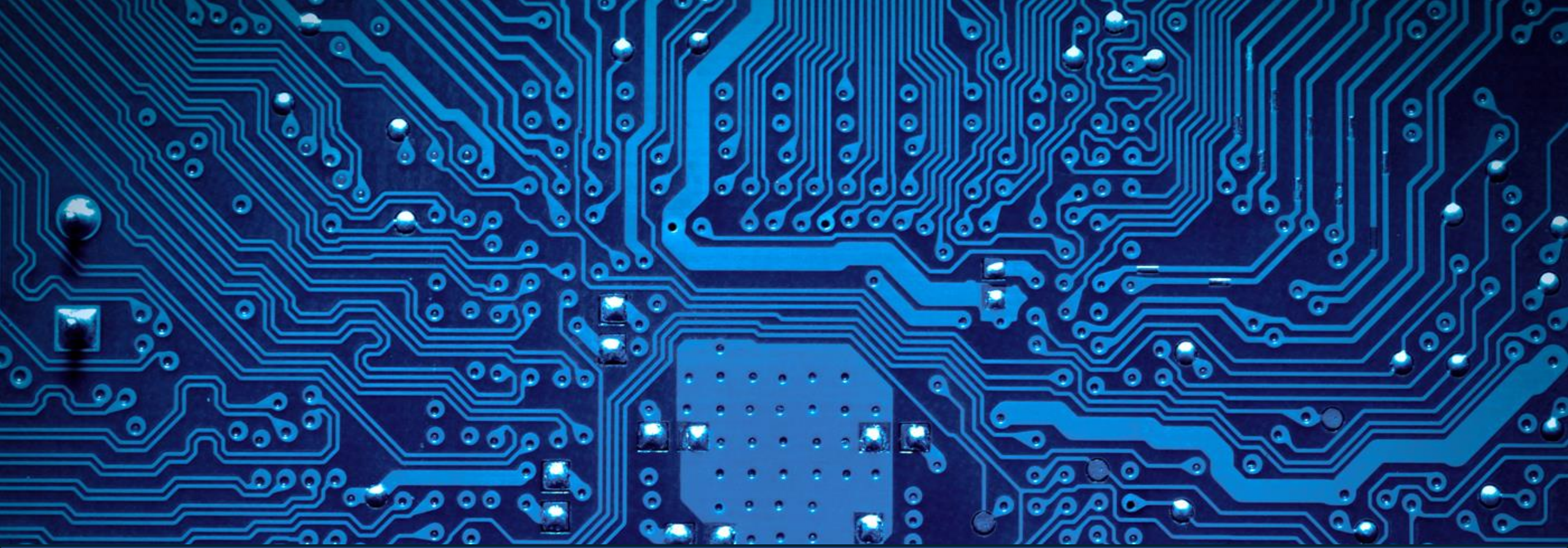


ECE 568: Computer Security

Part 1: Course Overview and Review

Courtney Gibson, P.Eng.

UNIVERSITY OF TORONTO

Part 1

Course Overview and Review

- Introduction
- Why should you take this course?
- Course Overview
- Security Fundamentals
- Review (Comp Org and Operating Systems)

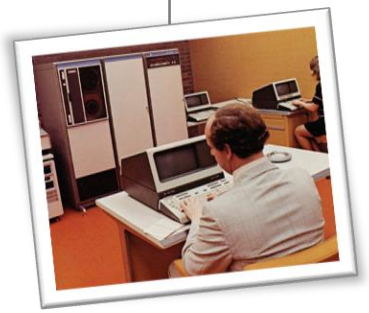
Introduction

UNIVERSITY OF TORONTO

Courtney Gibson

courtney.gibson@utoronto.ca

1989



Early Mainframe Systems
(“Time Share”)

2000

Early Cellular
Data/Voice Systems
(3G)

2003

Financial Systems
(High-Frequency Trading)

2018

Digital-Physical
Converged Systems
(IoT)

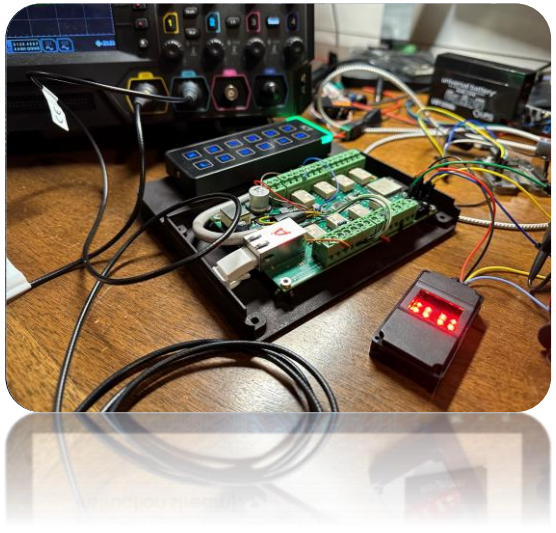
Courtney Gibson

courtney.gibson@utoronto.ca

bioconnect.

Chief Technology Officer
Chief Information Security Officer

IoT Design



Biometrics, RFID and Access Control

Compliance

AICPA SOC 2 Formerly SAS 70 Reports
GDPR
Personal Information Protection and Electronic Documents
HIPAA
PCI

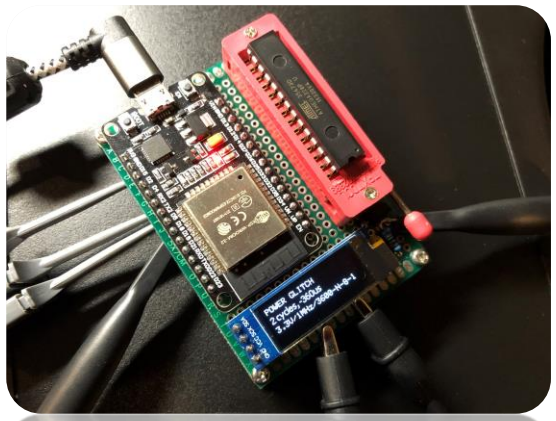
Courtney Gibson

courtney.gibson@utoronto.ca

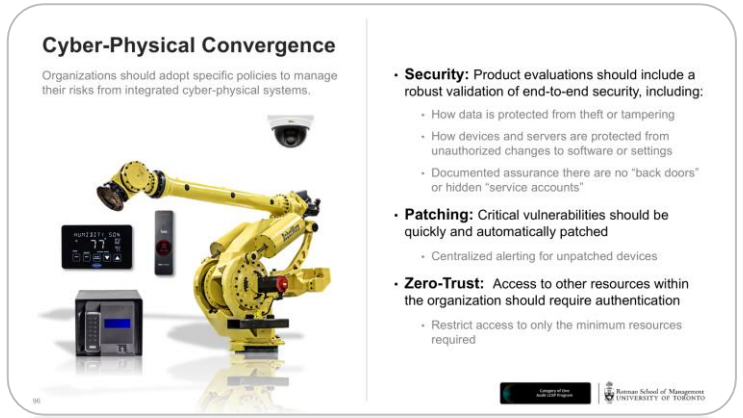
UNIVERSITY OF TORONTO

Adjunct Lecturer, Electrical and Computer Engineering
Instructor, Rotman School of Management

Cybersecurity



Cyber-Risk Management

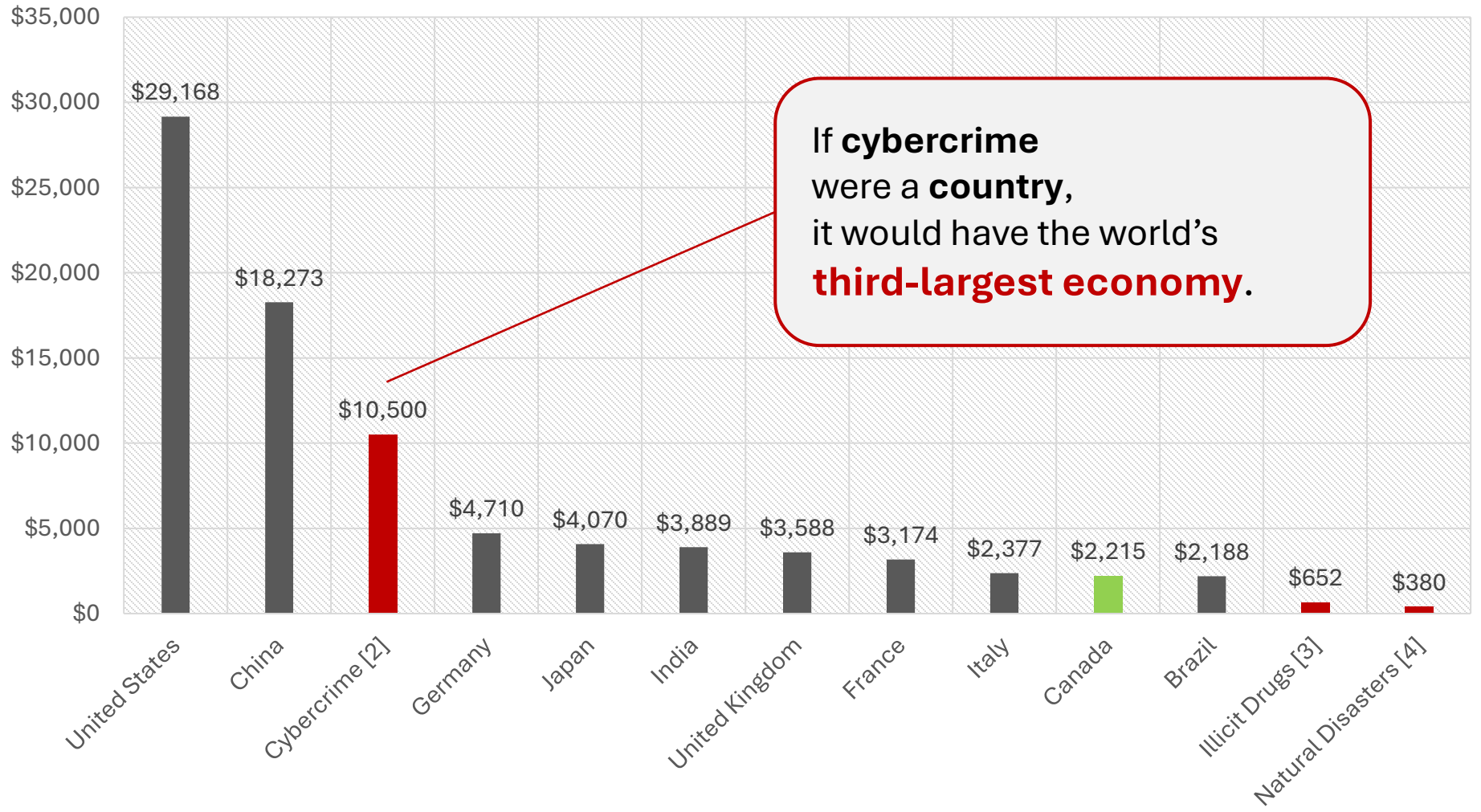


Please Ask Questions!

Why Should You Take This Course?

UNIVERSITY OF TORONTO

Top-10 Countries by GDP (2024[1], $B USD)

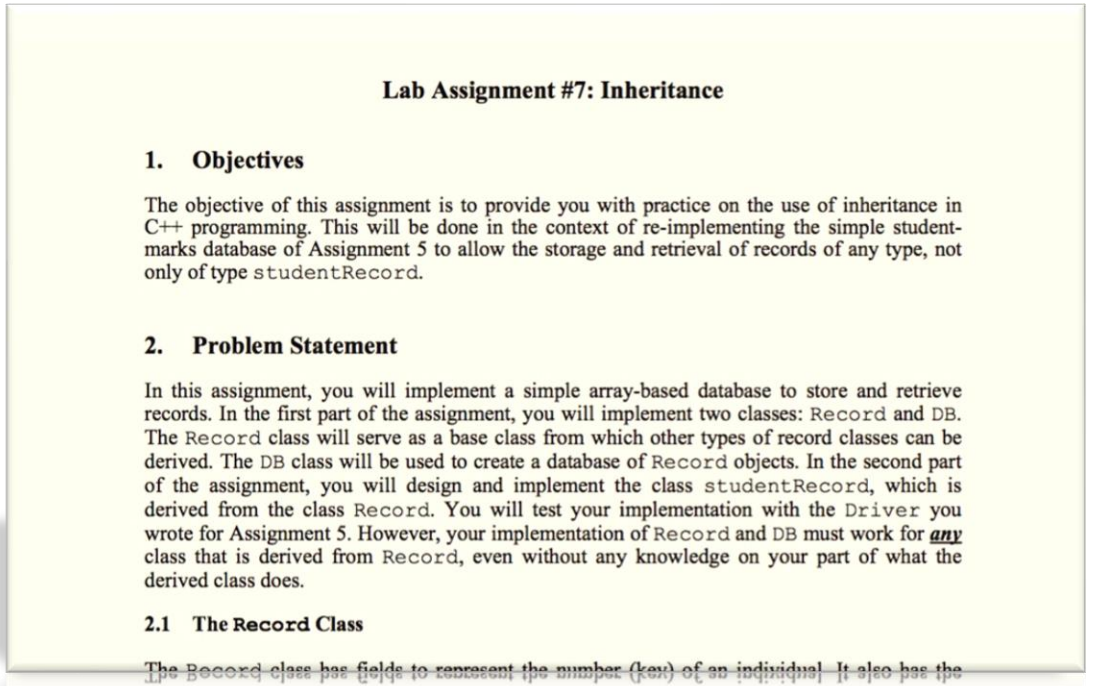


[1] IMF, *World Economic Outlook Database (October 2024)*
[2] Cybersecurity Ventures, *Boardroom Cybersecurity Report 2024*
[3] GFI, *Transnational Crime and the Developing World (2017)*
[4] AON, *Climate and Catastrophe Insight 2024*

You have spent a **lot of time** over the past few years learning the skills you need to analyze problems and solve them efficiently.

"Write a program to sort of a list of up to 100 numbers, as efficiently as possible."

You have spent **very little time** (if any) thinking about how an attacker might **break your work**... and **what might happen** if they succeed.



Lab Assignment #7: Inheritance

1. Objectives

The objective of this assignment is to provide you with practice on the use of inheritance in C++ programming. This will be done in the context of re-implementing the simple student-marks database of Assignment 5 to allow the storage and retrieval of records of any type, not only of type `studentRecord`.

2. Problem Statement

In this assignment, you will implement a simple array-based database to store and retrieve records. In the first part of the assignment, you will implement two classes: `Record` and `DB`. The `Record` class will serve as a base class from which other types of record classes can be derived. The `DB` class will be used to create a database of `Record` objects. In the second part of the assignment, you will design and implement the class `studentRecord`, which is derived from the class `Record`. You will test your implementation with the `Driver` you wrote for Assignment 5. However, your implementation of `Record` and `DB` must work for ***any*** class that is derived from `Record`, even without any knowledge on your part of what the derived class does.

2.1 The Record Class

The Record class has fields to represent the number (key) of an individual. It also has the

Motivation: Users of Technology

Software systems (and their associated cybersecurity risks) are ubiquitous in our daily lives.

- Protect your system
- Protect your data
- Protect your identity

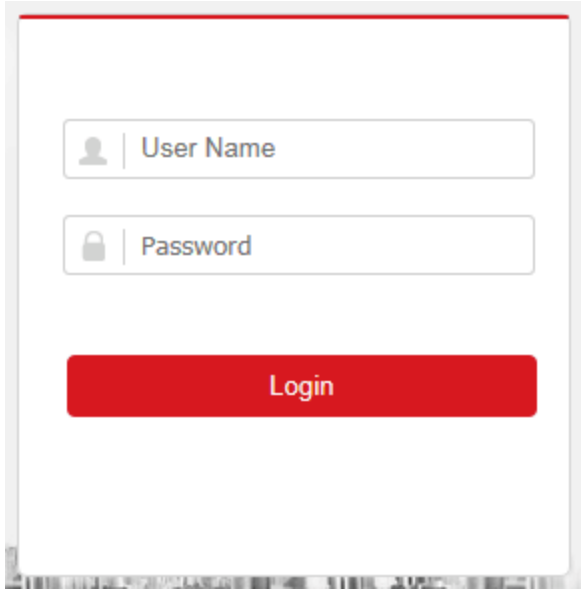
The New York Times

Somebody's Watching: Hackers Breach Ring Home Security Cameras

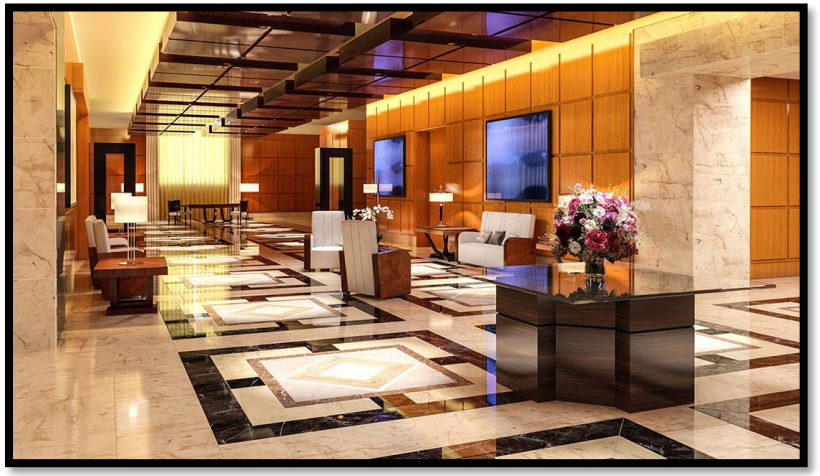
Unnerved owners of the devices reported recent hacks in four states. The company reminded customers not to recycle passwords and user names.

A family in Mississippi said a man hacked into a Ring home security camera in a bedroom shared by their daughters. Ashley LeMay

HTTP



RTSP

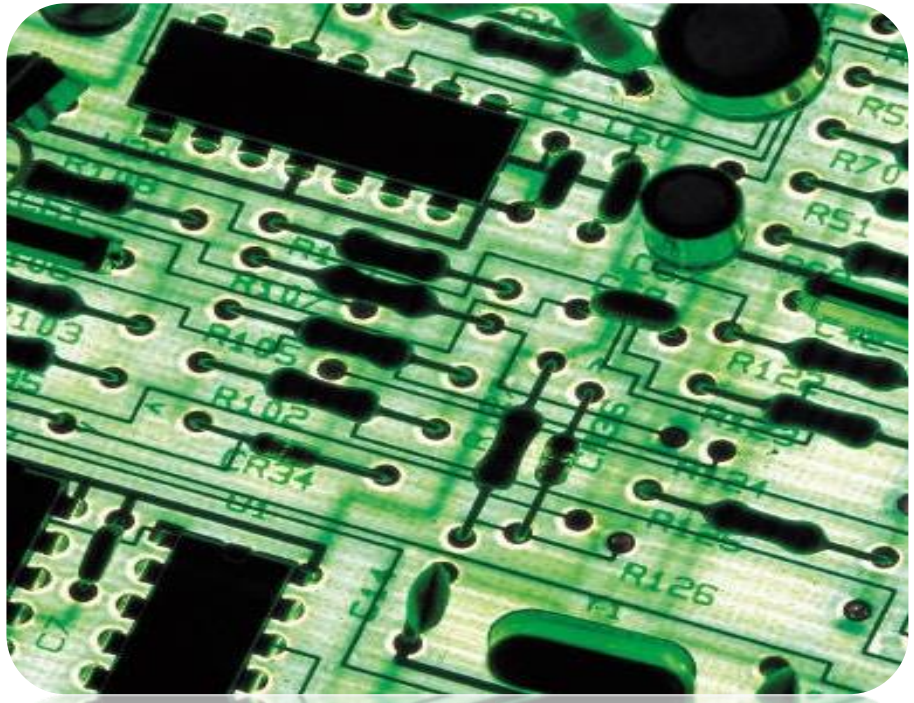


CitizenLab.ca

Motivation: Engineer

Security should be a daily consideration in your work.

- Challenge your design assumptions
- Understand attackers and identify risks
- Defensive architecture and implementation



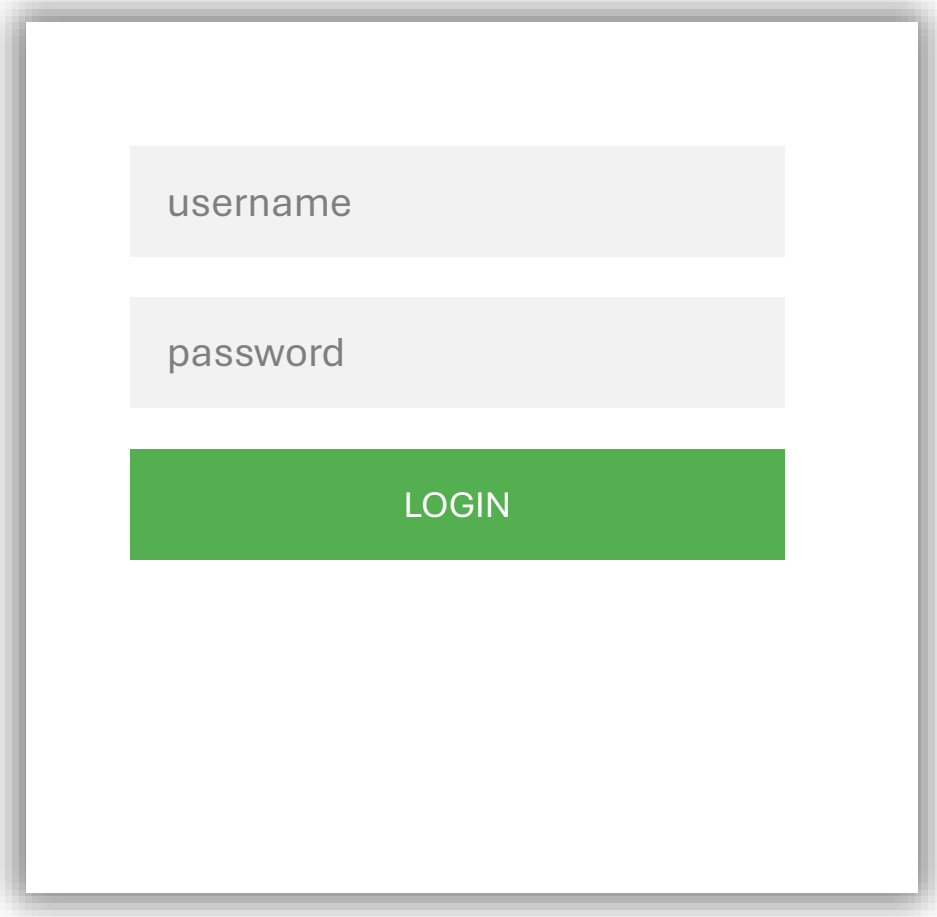
Case Study: Overly-Helpful Login

Background

- New hire at a financial services company was given the task of developing a web form that would allow customers to download their annual tax forms.

Analysis

- System should not have been storing passwords in plaintext!
- Error message leaked too much information to potential attackers.



Case Study: Las Vegas Casino

Background

- In early 2018 a large Las Vegas casino installed a new saltwater fish tank in their lobby; this involved installing an IoT device to permit remote monitoring of the water conditions in the tank

Issue

- The IoT tank monitor had a vulnerability: attackers remotely accessed the device, and then used it as an entry-point into the casino's internal network

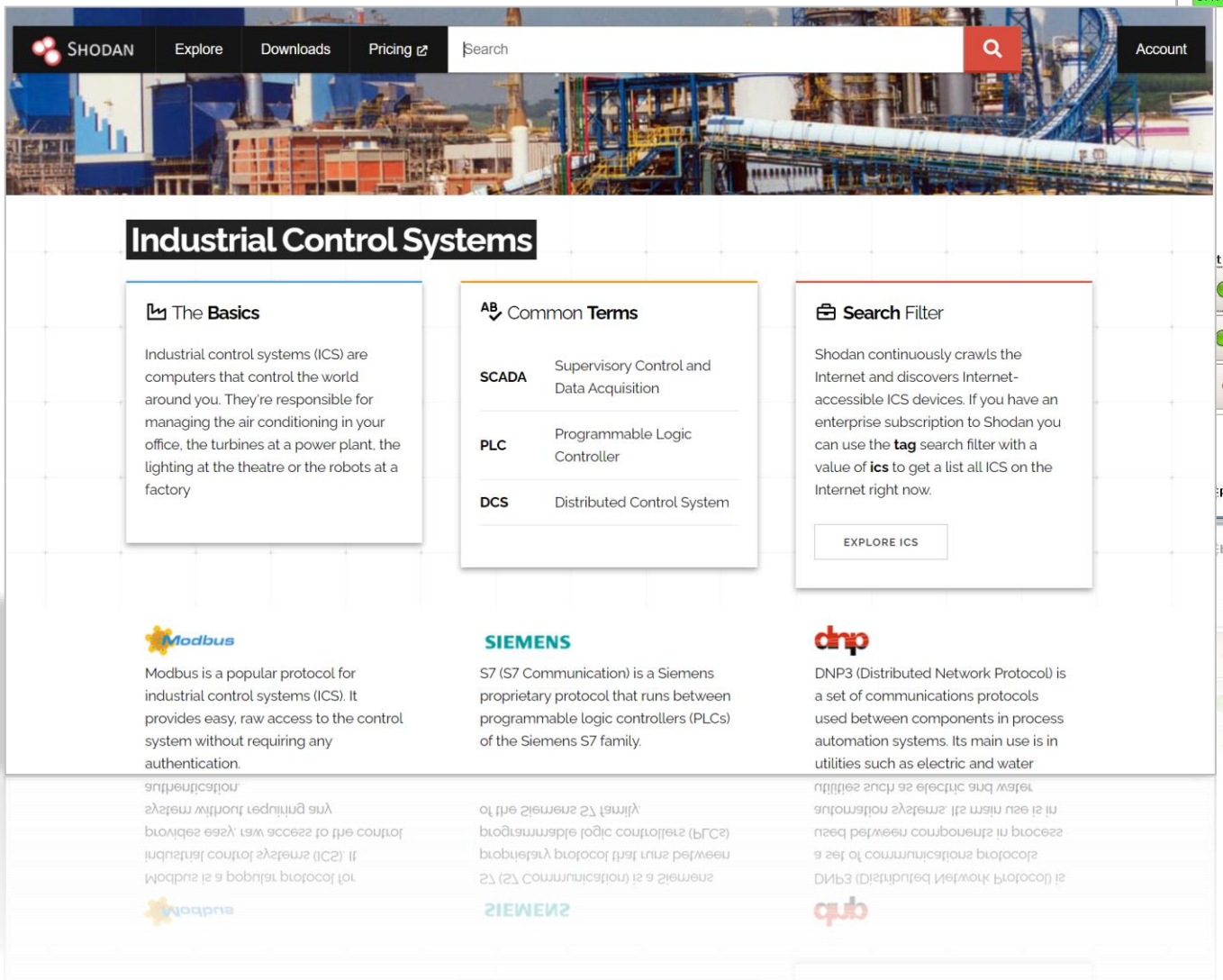
Result

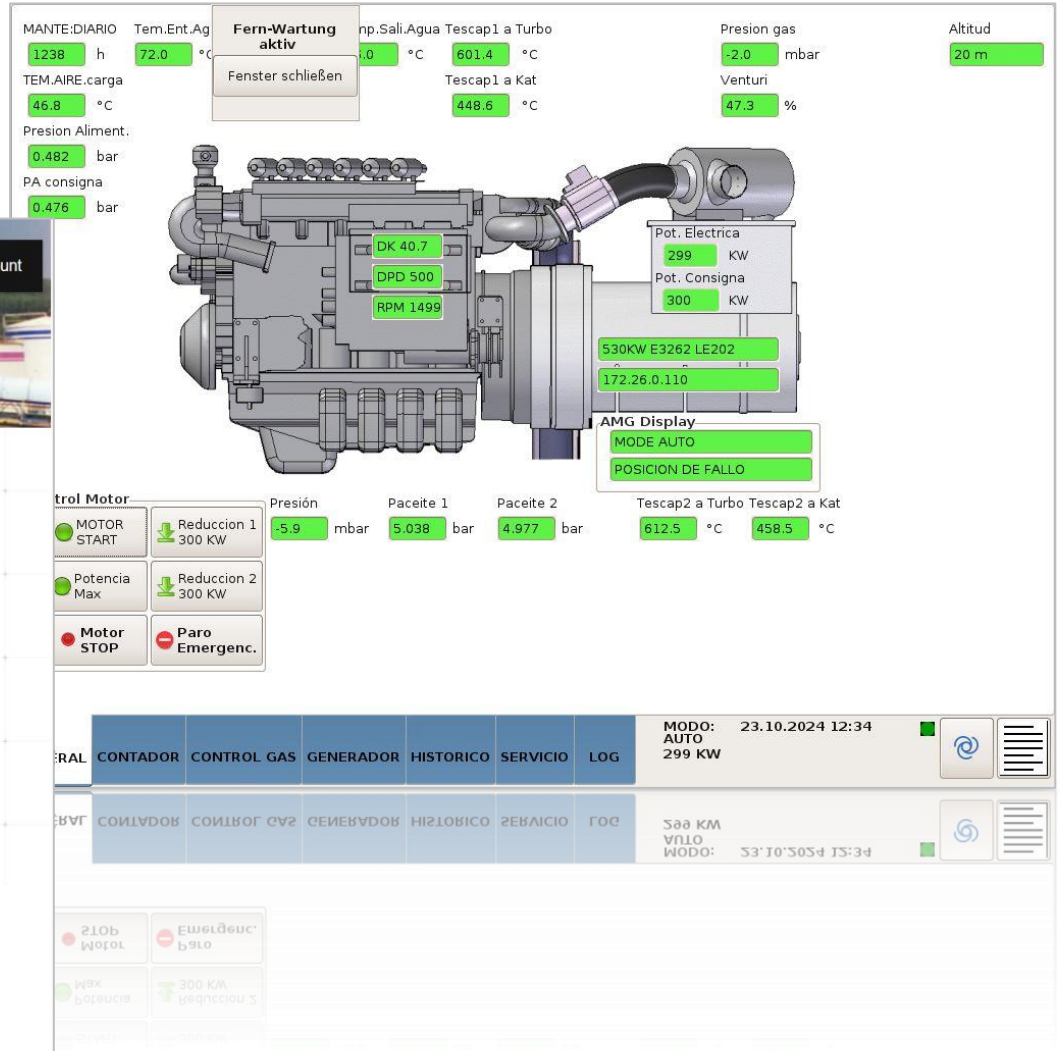
- The contents of the casino's "high-roller" database (with PII on thousands of VIP clients) was sent to a cloud server hosted in Finland (~10GB of data)
- Several million dollars in ransom was paid to prevent the release of the data

Analysis

- IoT devices are often overlooked, and avoid the same level of review that other new computers receive

Source: Nicole Eagan (CEO, Darktrace), *Wall Street Journal CEO Council Conference (2018)*


SHODAN
Explore
Downloads
Pricing
Search
Account
Industrial Control Systems
The Basics
Industrial control systems (ICS) are computers that control the world around you. They're responsible for managing the air conditioning in your office, the turbines at a power plant, the lighting at the theatre or the robots at a factory
Common Terms
SCADA
Supervisory Control and Data Acquisition
PLC
Programmable Logic Controller
DCS
Distributed Control System
Search Filter
Shodan continuously crawls the Internet and discovers Internet-accessible ICS devices. If you have an enterprise subscription to Shodan you can use the tag search filter with a value of ics to get a list all ICS on the Internet right now.
EXPLORE ICS
Modbus
Modbus is a popular protocol for industrial control systems (ICS). It provides easy, raw access to the control system without requiring any authentication.
SIEMENS
S7 (S7 Communication) is a Siemens proprietary protocol that runs between programmable logic controllers (PLCs) of the Siemens S7 family.
dnp
DNP3 (Distributed Network Protocol) is a set of communications protocols used between components in process automation systems. Its main use is in utilities such as electric and water


MANTE:DIARIO
1238 h
Tem.Ent.Ag
72.0 °C
Fern-Wartung aktiv
Fenster schließen
601.4 °C
Tescap1 a Turbo
Presion gas
-2.0 mbar
Altitud
20 m
TEM.AIRE.carga
46.8 °C
448.6 °C
Tescap1 a Kat
Venturi
47.3 %
Presion Aliment.
0.482 bar
PA consigna
0.476 bar
DK 40.7
DPD 500
RPM 1499
Pot. Electrica
299 KW
Pot. Consigna
300 KW
530KW E3262 LE202
172.26.0.110
AMG Display
MODE AUTO
POSICION DE FALLO
MOTOR START
Reduccion 1 300 KW
Potencia Max
Reduccion 2 300 KW
Motor STOP
Paro Emergenc.
Presión
-5.9 mbar
Paceite 1
5.038 bar
Paceite 2
4.977 bar
Tescap2 a Turbo
612.5 °C
Tescap2 a Kat
458.5 °C
CONTADOR
CONTROL GAS
GENERADOR
HISTORICO
SERVICIO
LOG
MODO: AUTO
299 KW
23.10.2024 12:34

Motivation: Employee

Laws increasingly hold companies accountable for poor security practice.

- Protect your company
- Protection of confidential data
- Regulatory requirements

Case Study: Equifax

Background

- Multinational consumer credit reporting agency: processes data on over 800M individuals and 88M businesses worldwide. In 2017 they experienced a substantial cyber breach.

Issue

- 2015 internal audit showed significant deficiencies with the patching of critical systems:
  - IT staff did not have a comprehensive asset inventory
  - The criticality of an IT asset was not considered when prioritizing patches
  - The patching process worked on an 'Honour system'
  - The report identified remediation actions; two years later many of the steps had not been completed
- A critical web server security patch was released in March 2017
  - Equifax was aware of the issue; had not updated their Credit Dispute website
  - Hacked over two months after the public was notified and urged to install the patch
- Systems designed to detect data exfiltration were offline: a certificate had expired 9 months prior

Result

- Breach lasted for 76 days
- Private records of over 163M American, British and Canadian citizens were compromised
- Multiple class-action lawsuits and regulatory actions

Source: U.S. Senate Permanent Subcommittee on Investigations, Committee on Homeland Security and Governmental Affairs, *How Equifax Neglected Cybersecurity and Suffered a Devastating Data Breach* (2019).

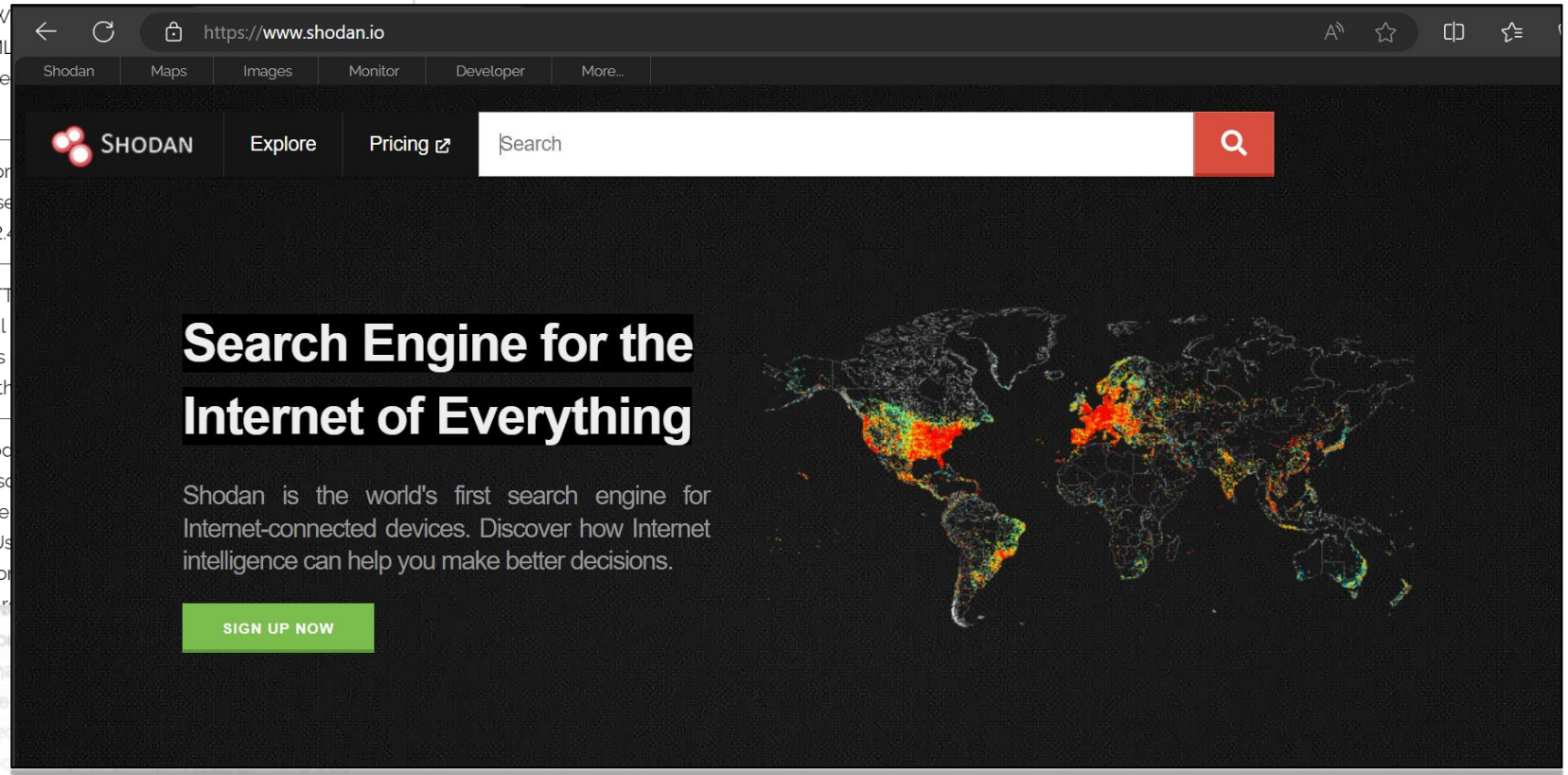
Vulnerabilities

All ports | Latest

Note: the device may not be impacted by all of these issues. The vulnerabilities are implied based on the software and version.

2024

CVE-2024-40898	falseSSRF in Apache HTTP Server on W… context, allows to potentially leak NTML… malicious requests. Users are recomme… fixes this issue.
CVE-2024-38477	falsenull pointer dereference in mod_pr… earlier allows an attacker to crash the se… recommended to upgrade to version 2.4…
CVE-2024-38476	falseVulnerability in core of Apache HTT… to information disclosure, SSRF or local … whose response headers are malicious … upgrade to version 2.4.60, which fixes th…
CVE-2024-38474	falseSubstitution encoding issue in mod… and earlier allows attacker to execute sc… configuration but not directly reachable… meant to only to be executed as CGI. Us… version 2.4.60, which fixes this issue. So… substitute unsafely will now fail unless r…



Case Study: Colonial Pipeline

Background

- Largest fuel operator in the United States
- Transports refined gasoline and jet fuel from Houston, across the southeastern United States
    - Approximately 45% of all fuel consumed on the US east coast arrives via their pipeline system

Issue

- May 2021: Ransomware attack took down their billing system. Unable to bill customers, Colonial decided to shut down their pipeline.
    - Fuel shortages led to panic buying: over 11,000 gas stations ran out of fuel in US east coast; impact to airlines at several airports
    - Federal government declared a state of emergency

Result

- Paid a ransom of $4.4M in Bitcoin
    - Attackers stole ~100GB of data day before malware attack; threatened to release it if the ransom wasn't paid
    - FBI recovered 84% of payment – but drop in Bitcoin meant the recovered funds had lost half their value
- A data-recovery tool was provided by the attackers – however, the tool was so slow that Colonial's own BCP procedures were more-effective in restoring their systems. Led to a six-day shutdown and significant operating losses.

My two goals for you in this course are to help you:

1. Get better at **questioning your assumptions**.
2. Better-understand **what tools you can trust, and when.**

We will be spending this term looking closely at the this limits of trust you can put in your tools:

- Languages, Compilers
- Hardware
- Networking
- Storage

KELSER

Microsoft Partner
Silver Cloud Productivity
Silver Devices and Deployment
Silver Midmarket Solution Provider

Ten Immutable Laws of Security (Version 2.0)

Here at the Microsoft Security Response Center, we investigate thousands of security reports every year. In some cases, we find that a report describes a bona fide security vulnerability resulting from an issue in one of our products; when this happens, we develop a corrective update as quickly as possible. In other cases, the reported problems simply result from a mistake someone made in using the product, or our investigation finds a problem with the product that, while troublesome for users, does not expose them to a security vulnerability. But many fall in between. They are genuine security problems, but the problems don't result from product flaws. Over the years, we've developed a list of issues like these that we call the 10 Immutable Laws of Security.

Don't hold your breath waiting for an update that will protect you from the issues we'll discuss below. It isn't possible for Microsoft-or any software vendor-to "fix" them, because they result from the way computers work. But don't abandon all hope yet. Sound judgment is the key to protecting yourself against these pitfalls, and if you keep them in mind, you can significantly improve the security of your computers, whether they sit on your desk, travel in your pocket, or exist in a virtual cloud. (Throughout this list we'll use "computer" to mean all of those objects, by the way.)

The 10 Immutable Laws

Law #1: If a bad guy can persuade you to run his program on your computer, it's not solely your computer anymore.
Law #2: If a bad guy can alter the operating system on your computer, it's not your computer anymore.
Law #3: If a bad guy has unrestricted physical access to your computer, it's not your computer anymore.
Law #4: If you allow a bad guy to run active content in your website, it's not your website any more.
Law #5: Weak passwords trump strong security.
Law #6: A computer is only as secure as the administrator is trustworthy.
Law #7: Encrypted data is only as secure as its decryption key.
Law #8: An out-of-date antimalware scanner is only marginally better than no scanner at all.
Law #9: Absolute anonymity isn't practically achievable, online or offline.
Law #10: Technology is not a panacea.

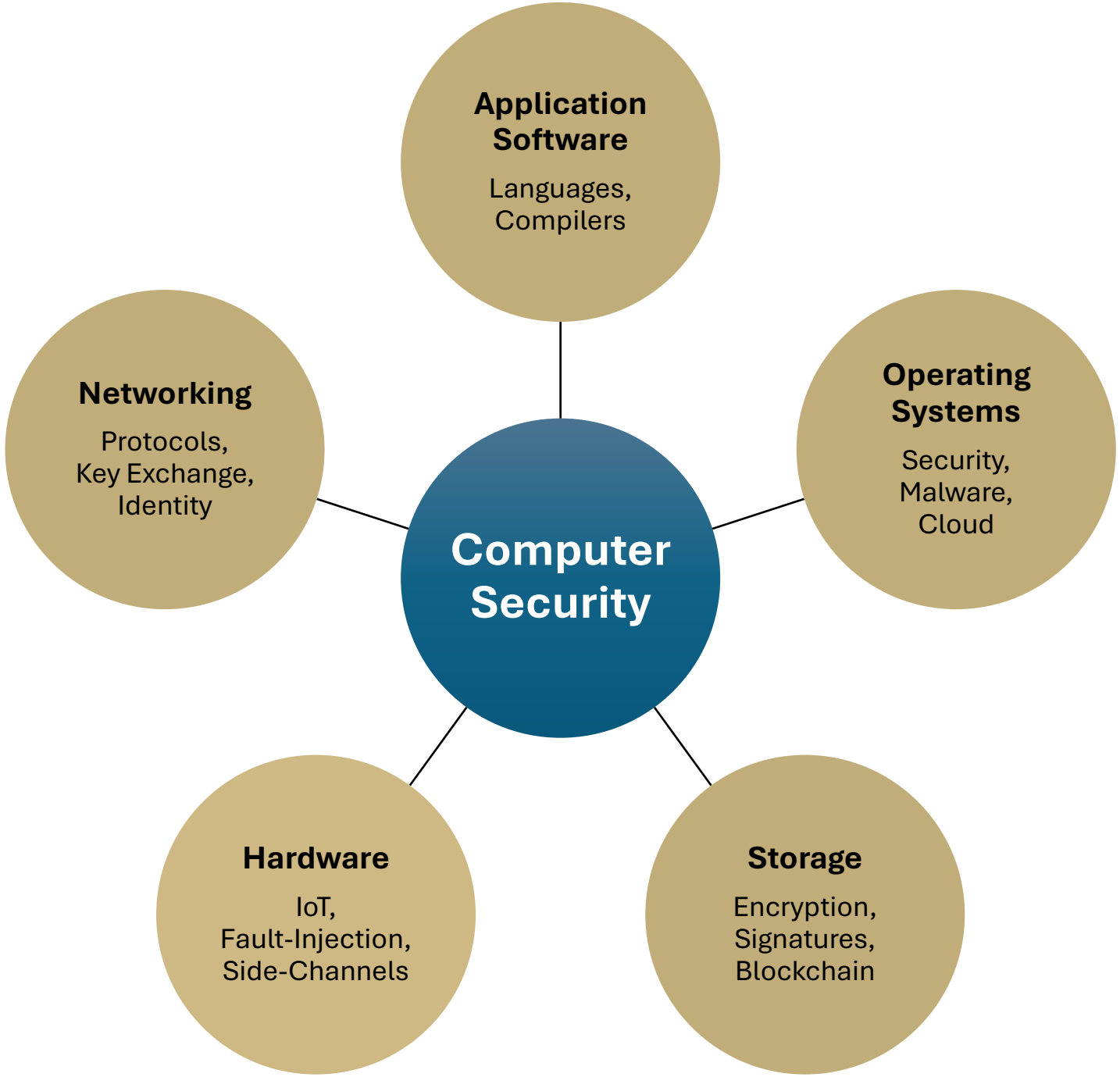
Law #1: If a bad guy can persuade you to run his program on your computer, it's not solely your computer anymore

It's an unfortunate fact of computer science: when a computer program runs, it will do what it's programmed to do, even if it's programmed to be harmful. When you choose to run a program, you are making a decision to turn over a certain level of control of your computer to it - often anything up to the limits of what you yourself can do on the computer (and sometimes beyond). It could monitor your keystrokes and send them to criminals eager for the information. It could open every document on the computer, and change the word "will" to "won't" in all of them. It could send rude emails to all your friends. It could install a virus. It could create a "back door" that lets someone remotely control your computer. It could relay a bad guy's attack on someone else's computers. Or it could just reformat your hard drive.

That's why it's important never to run a program from an untrusted source, and to limit the ability of others to make that decision for you on your computer. There's a nice analogy between running a

Microsoft's

Ten Immutable Laws of Security


Application Software
Languages, Compilers
Networking
Protocols, Key Exchange, Identity
Operating Systems
Security, Malware, Cloud
Computer Security
Hardware
IoT, Fault-Injection, Side-Channels
Storage
Encryption, Signatures, Blockchain

Course Overview

UNIVERSITY OF TORONTO

ECE568

- We will be focusing on the technology used to **build software and systems**.

- We will **not** be focusing on configuration settings, administrative policies or physical controls:
    - Wifi security standards
    - Data retention policies, data-destruction procedures
    - Cooling / fire-suppression, guards, fences, cameras, etc..

- We will look at cryptography from the standpoint of how to build secure systems (rather than focusing on formal proofs).

- You have a responsibility to use what we learn in this course to **create better security, rather than break it.**

Quercus

https://q.utoronto.ca/

Syllabus
Lecture Slides
Assignments
Past Tests
Announcements

Course Outline

Part 1: Course Overview and Review

- Course introduction
- Quick review: O/S, assembly language, software tools
- Basic principles of computer security
- Confidentiality, Integrity, Availability

Primary Learning Goals:

1. Ability to explain the fundamental goals of Confidentiality, Integrity and Availability, and how they each can be applied to creating secure product designs.
2. Ability to use basic software debugging tools (*e.g.*, gdb) to analyze program behaviour.

Course Outline

Part 2: Software Code Vulnerabilities

- Injection-type attacks: buffer overflows, format strings, double-free, ROP, etc.
- Library attacks: encapsulation, etc.
- Defenses and good programming practice

Primary Learning Goals:

1. Ability to explain and execute the most common attacks against software vulnerabilities, in order to locate design and implementation flaws.
2. Ability to use appropriate tools and best-practice coding techniques to protect against these types of attacks in your own work.

Course Outline

Part 3: Cryptography

- Encryption: Symmetric (block/stream ciphers) and Public Key
- Secure key-exchange
- Hashes and Signatures: MAC, HMAC, blockchain, etc.
- Secure Communication Protocols: SSL/TLS, VPN

Primary Learning Goals:

1. Ability to select appropriate encryption techniques to meet both the Data Confidentiality requirements and the performance requirements of your software.
2. Ability to securely exchange cryptographic keys over insecure network channels.
3. Ability to use hash-based tools to provide Data Integrity and Data Authentication, through the application of MACs, HMACs and digital signatures.

Course Outline

Part 4: Web Security

- Web authentication, cookies
- Web attacks: Cross-Site Scripting (XSS), CSRF, etc.
- SQL Injection
- Cloud security

Primary Learning Goals:

1. Ability to safely use web cookies to secure secure sessions and end-user data.
2. Ability to execute the most common web-based attacks, in order to locate design and implementation flaws.
3. Ability to use appopriate tools and best-practice coding techniques to protect against these types of attacks in your own work.

Course Outline

Part 5: Access Control

- Access control models
- Hardware Security Modules (HSMs)
- Side-Channels and Covert Channels
- Attacks on IoT/embedded systems

Primary Learning Goals:

1. Ability to identify potential sources of side- and covert-channels that could be used to leak information out of both hardware- and software-based systems.
2. Ability to identify and describe potential injection vulnerabilities, and how they might compromise the security of IoT/embedded systems.

Course Outline

Part 6: Network Security

- Network-layer security risks: BGP, ARP, DNS, TCP/IP, etc.

Primary Learning Goals:

1. Ability to identify how vulnerabilities in many standard networking protocols could be exploited by a third-party to pose risks to confidentiality, integrity and availability of your software systems.
2. Ability to use appropriate tools and best-practice techniques to protect your own systems and designs.

Course Outline

Part 7: Malware

- Viruses and worms
- Rootkits and bootkits

Primary Learning Goals:

1. Ability to identify the mechanisms though which computer viruses, worms, rootkits and bootkits functions and propagate, and the risks these may pose to your systems.

Course Outline

Section 8: Physical Security

- Physical security system design and vulnerabilities

Primary Learning Goal:

1. Ability to explain the most common vulnerabilities and defenses in physical security systems, and relate this to design challenges in digital security designs.

Optional Textbooks

Security in Computing

- Pfleeger and Pfleeger

Computer Security: Principles & Practice

- Stallings and Brown

Applied Cryptography

- Bruce Schneier

Marking Scheme

- **Labs:** 25%
- **Midterm:** 35%
- **Final Exam:** 40%

The midterm and final are both “Type C” (single reference sheet), no calculator.

Plagiarism

All labs, assignments and tests are to be completed with your own original work.

Anything submitted for credit must be something that you (and your lab partner) produced.

What to Expect

Course Covers a Lot of Material

- Memory, CPUs
- Operating Systems, Networking
- Cryptography

Where You'll Spend Your Time

- Four lab assignments
- Most of the work will be in the labs: course focuses on practical aspects of security

Lab Assignments

Lab 1: Identification of Vulnerabilities, Construction of Attacks

- You will be given some vulnerable programs.
- Your job is to construct attacks that will let you hijack the programs and spawn a command shell.

Lab 2: Cryptography

- You will use X509 certificates and digital signatures to create a basic blockchain.

Lab 3: Web Security

- You will be given a web application with a number of security vulnerabilities.
- You will craft a series of attacks that exploit the vulnerabilities in the application's design.

Lab 4: Network Security

- TBD (Under Revision)

Computer Security

Rules and Assumptions

UNIVERSITY OF TORONTO

Rules

Like the programs that enforce them, computer systems have rules.

Some rules are **explicit** and well thought out; others are **implicit** and based on assumptions.

Implicit rules are often a source of security vulnerabilities.

Assumptions

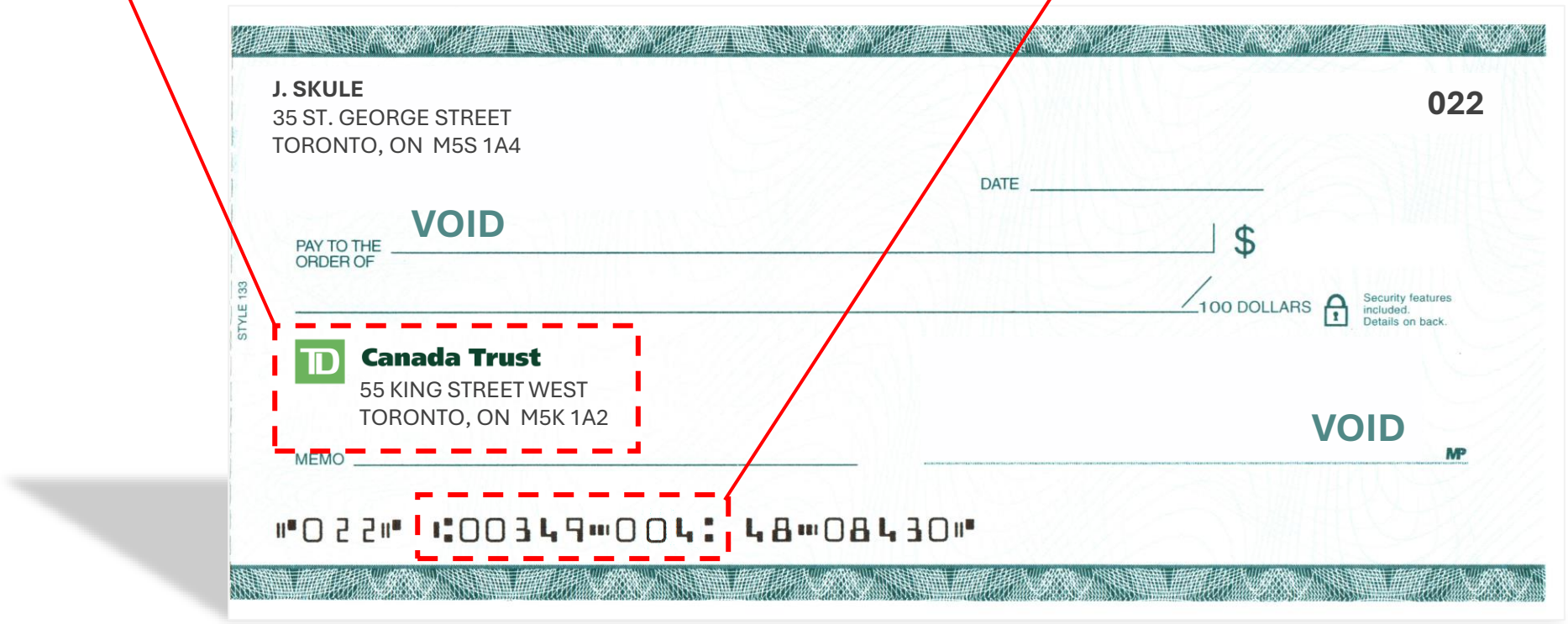
Security risks occur when our assumptions turn out to be false:

- Data
- Input
- User behaviour

Case Study: “Cheque Kiting” Scam (1980’s)

Human-Readable:
“Toronto”

Machine-Readable:
“Newfoundland”



Computer Security

What is it? What makes it difficult?

UNIVERSITY OF TORONTO

Computer Security

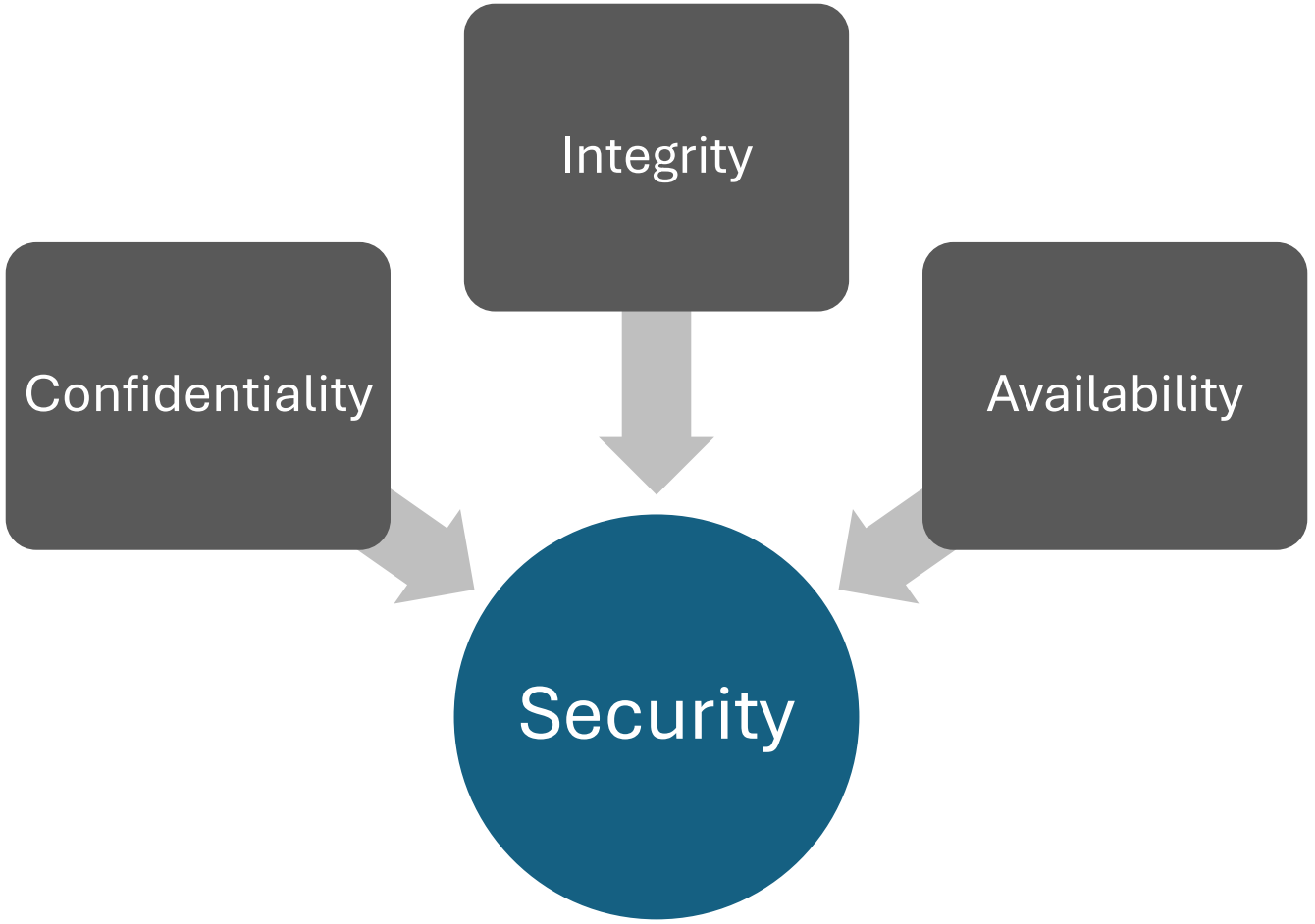
Cybersecurity is about **understanding** a system really well, and **questioning** the implicit rules.

- A **reliable** system does what it is supposed to do.
- A **secure** system does what it is supposed to do, and nothing else.

Cybersecurity is…

1. …the **protection** of information and information-systems…
2. …against **unauthorized** access, use, disclosure, modification or destruction…
3. …in order to provide **confidentiality**, **integrity** and **availability**.

Security Fundamentals



Confidentiality

Confidentiality is the protection of data or resources from exposure.

There are two aspects of information or resources that are often important to conceal:

1. The **content** of the data or resource (*e.g.*, passwords, etc.).
2. The **existence** of the data or resource (as merely knowing that something exists may be damaging).

- **Organizational Controls**
  - Acceptable-Use Policies
  - Security Training
  - Privacy Policies
  - Retention Policies
- **Access Rules**
  - Software
  - Hardware
- **Cryptography**
  - Authentication / authorization
  - Encryption

Integrity

Integrity is establishing and maintaining the trustworthiness of the data or resource.

There are two important aspects of a resource or piece of data:

1. Correctness of its **contents**: can we tell if it's been altered?
2. Correctness of its **origin**: can we authenticate who this really came from?

- **Tools**
  - Monitoring (anti-malware, etc.)
  - Backups
- **Access Rules**
  - Policies
  - Controls (SQL Injection, etc.)
- **Cryptography**
  - Hashing
  - Signing
  - Cloud-specific tools/techniques

Availability

Availability ensures the data or resource can be used as/when desired.

- A **resource** is available if it is responding to requests
- **Data** are available if the service that stores the information is up and running

Availability is generally harder to ensure.

- A resource could be temporarily unavailable, but at what time does it really become unavailable?

- **Disaster Recovery**
  - Files (data/software)
  - Servers
  - Networking
- **Backups**
- **Mitigation Tools**
  - Firewalls
  - Denial-of-Service prevention

Security: **CIA**

- All secure systems provide **Confidentiality**, **Integrity** and **Availability** to some degree
- However, no system is absolutely secure, and no system can provide all (any?) of the three absolutely
- Generally, a system is considered secure if the security requirements are met **for a given time**
  - *e.g.*, confidential data is not leaked until it is declassified

Vulnerabilities

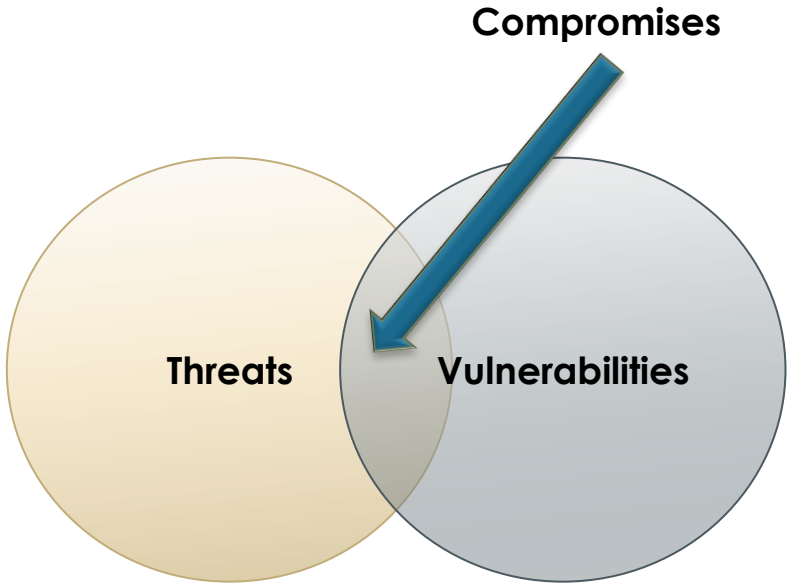
A **vulnerability** is a flaw that weakens the security of the system. Vulnerabilities are very difficult to identify even after significant amount of testing

- They can be serious:
  - An unchecked string copy allows an attacker to overflow a buffer and execute arbitrary code in a privileged program
  - During configuration, a system administrator forgets to disable debug mode on a program, allowing an attacker to gain privileged information
  - A naïve user does not change the default password on their router from the factory default

Threats

A **threat** is any method that can be used to potentially breach security and cause harm.

- When an attacker exercises a threat, it becomes an **attack** (also called an “exploit”)
- If the attack is successful, the system is **compromised**



Security is a **Negative Goal**

Ensuring that something **cannot** happen is much harder to design, measure and verify than traditional software quality-assurance goals.

- **Positive goal:** Alice can read the file
- **Negative goal:** Bob cannot read the file

Problem: In what ways might an attacker try to access Alice's file? (Not an easy question to answer!) Start with designing **rules**, and then check the assumptions those rules rely on.

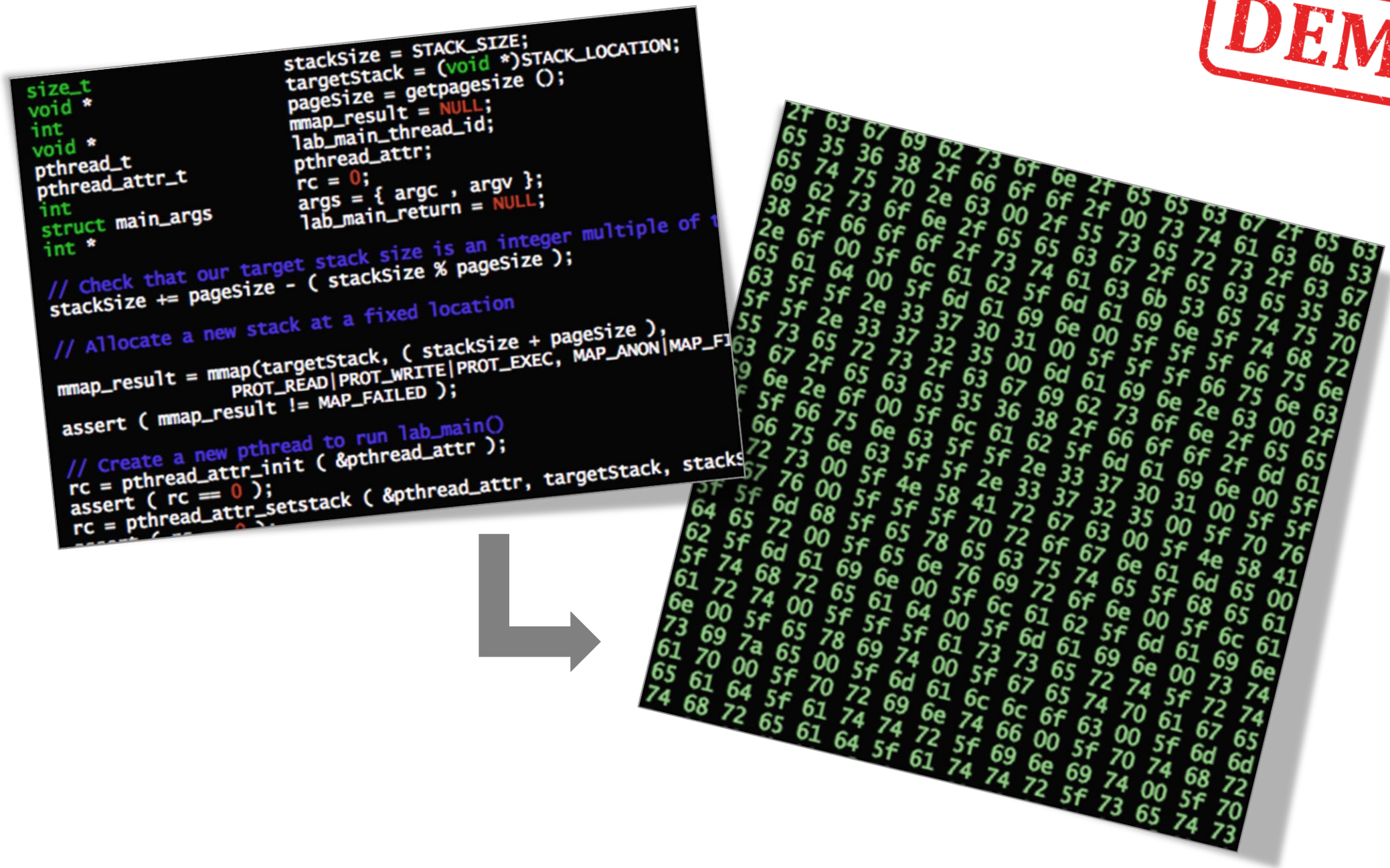
Review

Memory: Data Representation

UNIVERSITY OF TORONTO

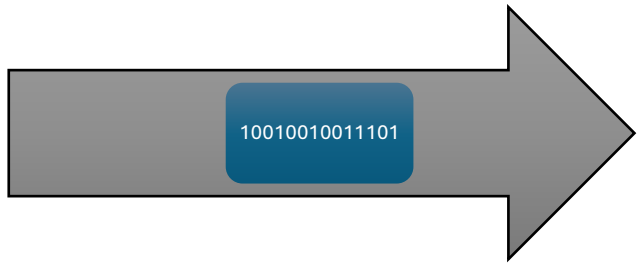
Data Representation

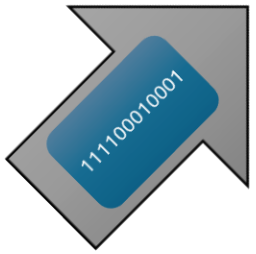
As developers, our primary job is to manipulate numbers.

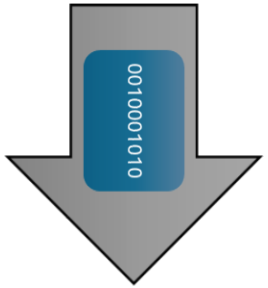

size_t stacksize = STACK_SIZE;
void * targetStack = (void *)STACK_LOCATION;
int pageSize = getpagesize ();
void * mmap_result = NULL;
pthread_t lab_main_thread_id;
pthread_attr_t pthread_attr;
int rc = 0;
struct main_args args = { argc , argv };
int * lab_main_return = NULL;
// Check that our target stack size is an integer multiple of t
stacksize += pageSize - (stackSize % pageSize);
// Allocate a new stack at a fixed location
mmap_result = mmap(targetStack, (stackSize + pageSize),
PROT_READ|PROT_WRITE|PROT_EXEC, MAP_ANON|MAP_FI
assert (mmap_result != MAP_FAILED);
// Create a new pthread to run lab_main()
rc = pthread_attr_init (&pthread_attr);
assert (rc == 0);
rc = pthread_attr_setstack (&pthread_attr, targetStack, stacks

DEMO

```
size_t          stackSize = STACK_SIZE;
void *          targetStack = (void *)STACK_LOCATION;
int             pageSize = getpagesize ();
void *          mmap_result = NULL;
pthread_t       lab_main_thread_id;
pthread_attr_t  pthread_attr;
int             rc = 0;
struct main_args args = { argc , argv };
int *           lab_main_return = NULL;

// Check that our target stack size is an integer multiple of
stackSize += pageSize - ( stackSize % pageSize );

// Allocate a new stack at a fixed location

mmap_result = mmap(targetStack, ( stackSize + pageSize ),
                   PROT_READ|PROT_WRITE|PROT_EXEC, MAP_ANON|MAP_F
assert ( mmap_result != MAP_FAILED );

// Create a new pthread to run lab_main()
rc = pthread_attr_init ( &pthread_attr );
assert ( rc == 0 );
rc = pthread_attr_setstack ( &pthread_attr, targetStack, stacks
```







Data Representation

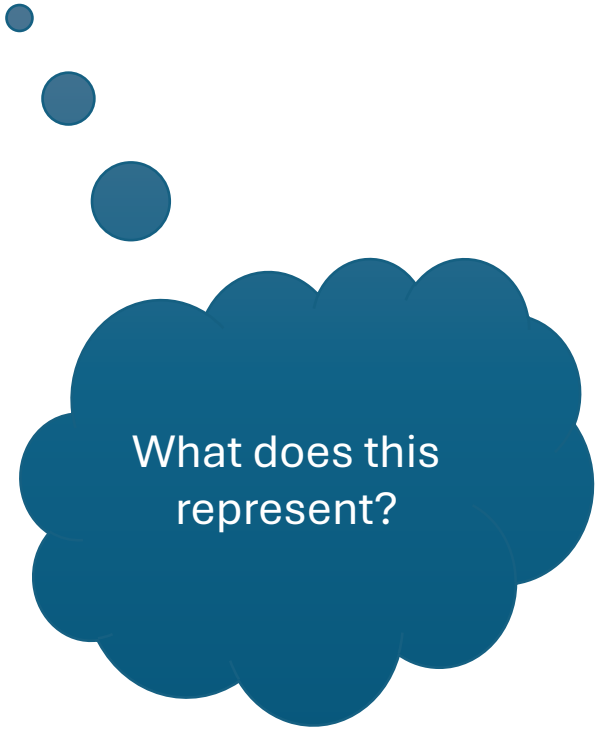
Because everything (instructions, input and output) is represented as numbers, we need to be careful.

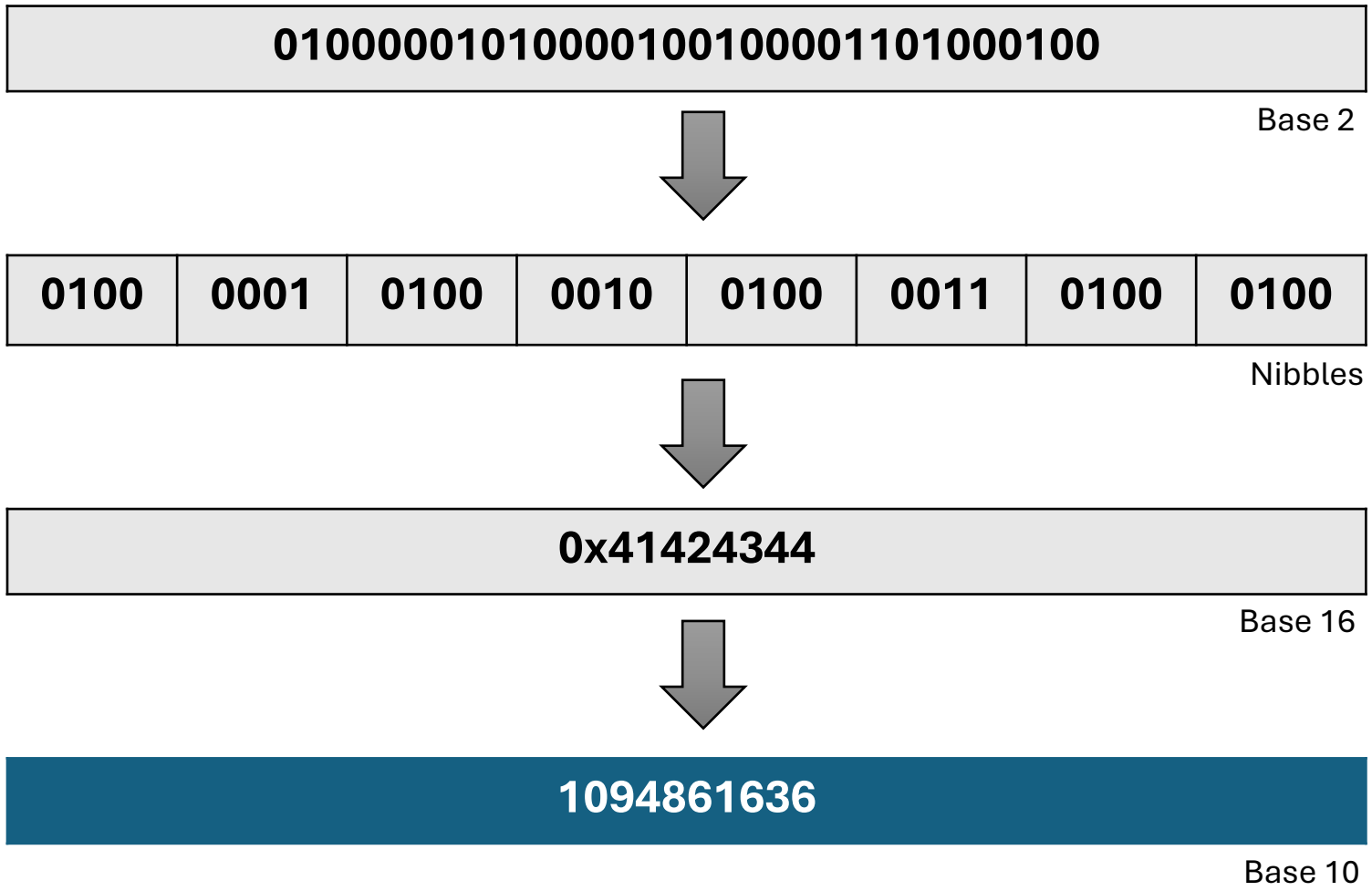
Try to avoid thinking of memory only in terms of how you define your variables:

```
int x = 1145258561;
```

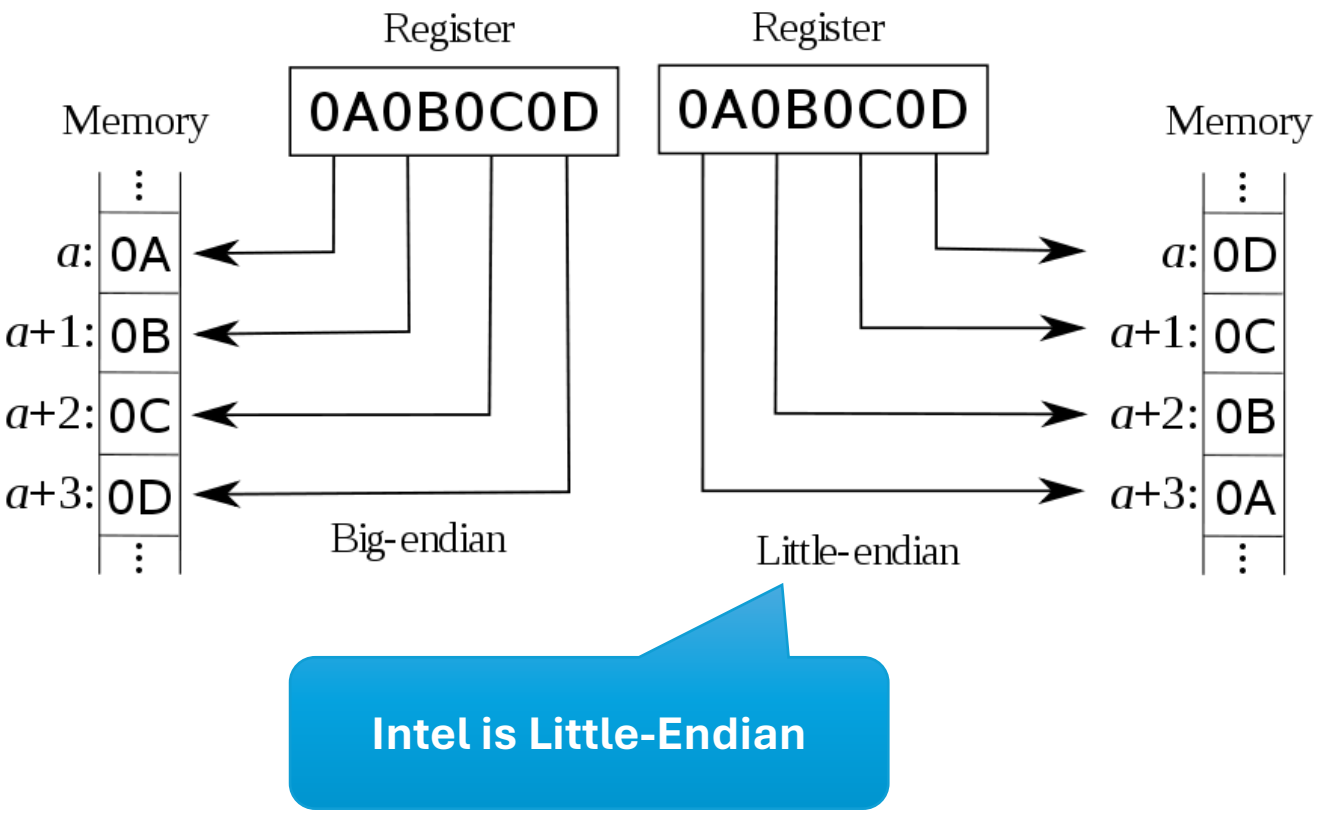
01000001010000100100001101000100

Base 2




01000001010000100100001101000100
Base 2
0100
0001
0100
0010
0100
0011
0100
0100
Nibbles
0x41424344
Base 16
1094861636
Base 10

Big-Endian (MSB) vs. Little-Endian (LSB)

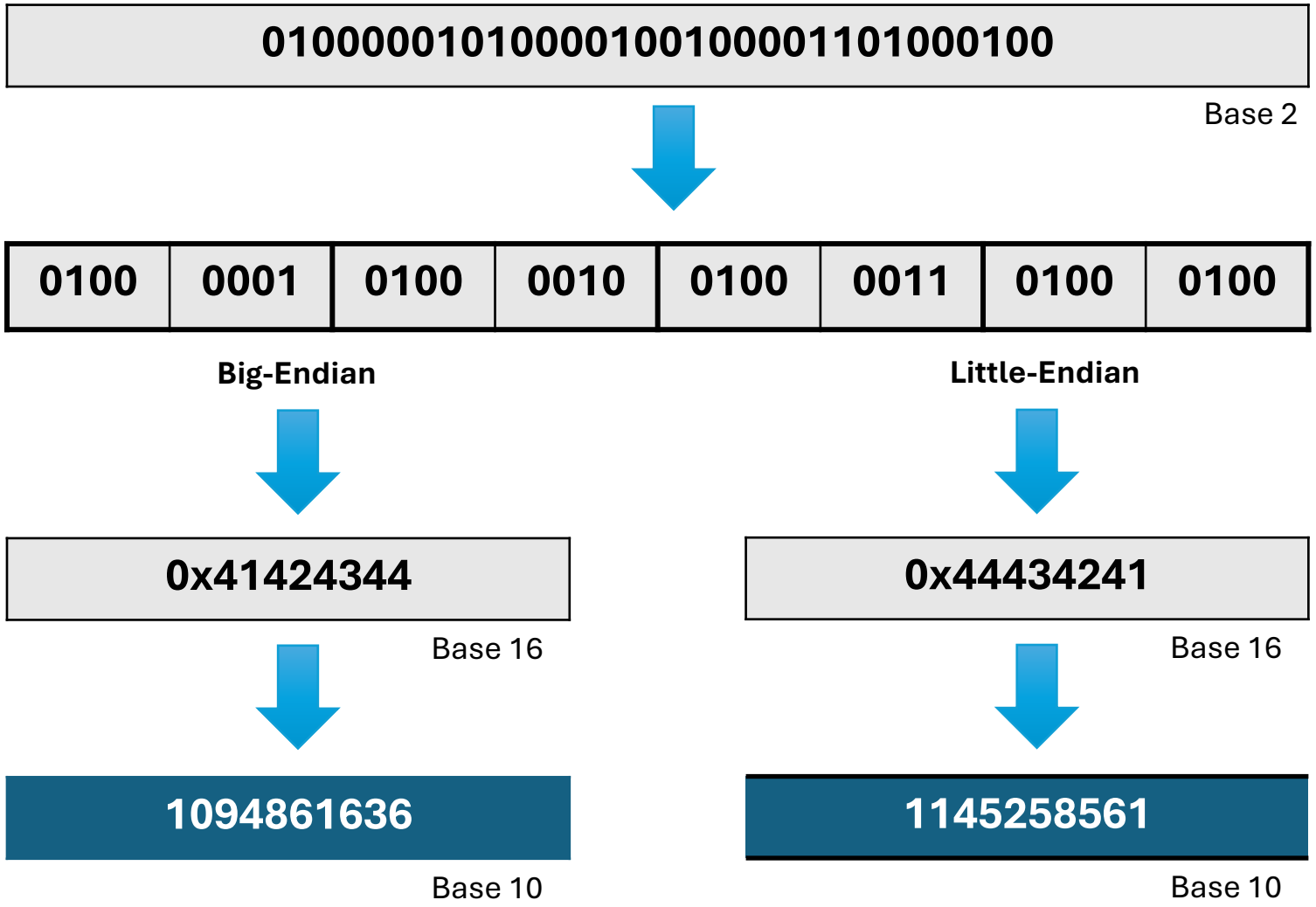


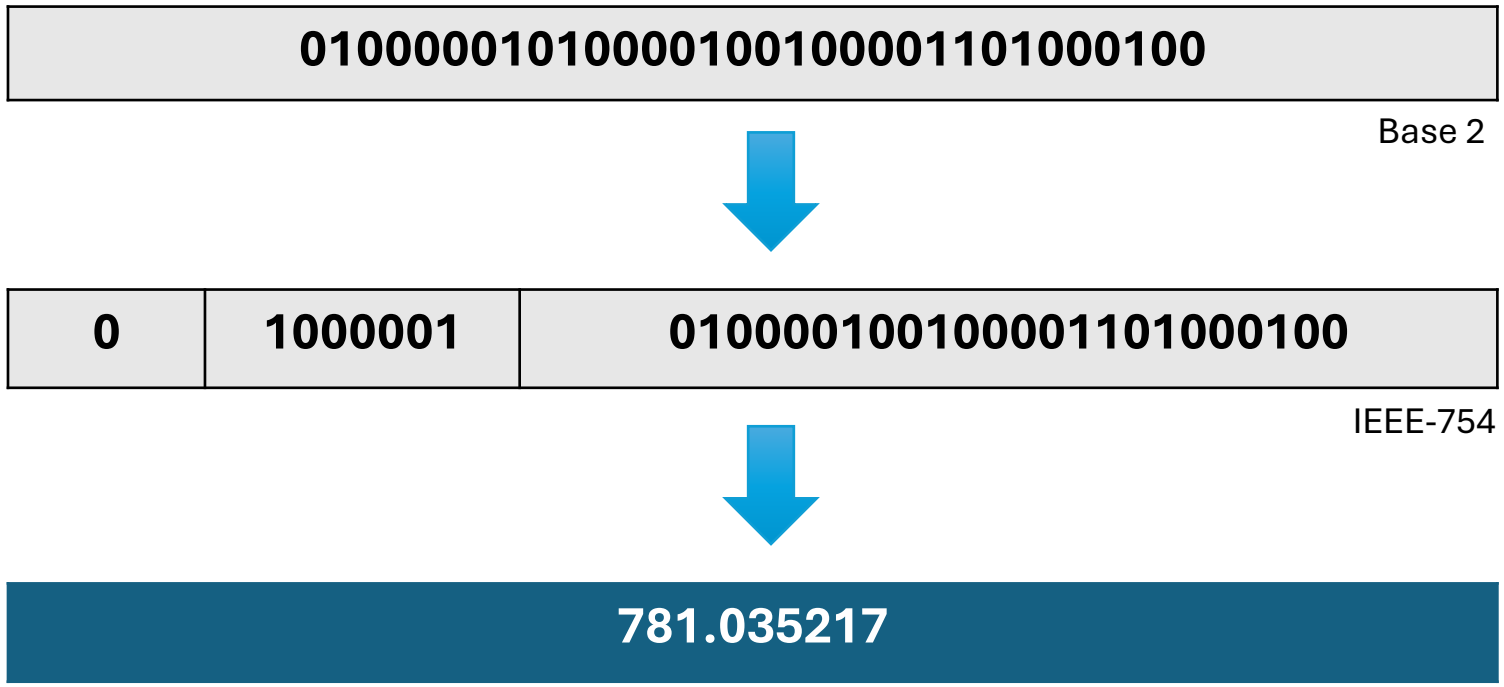
C

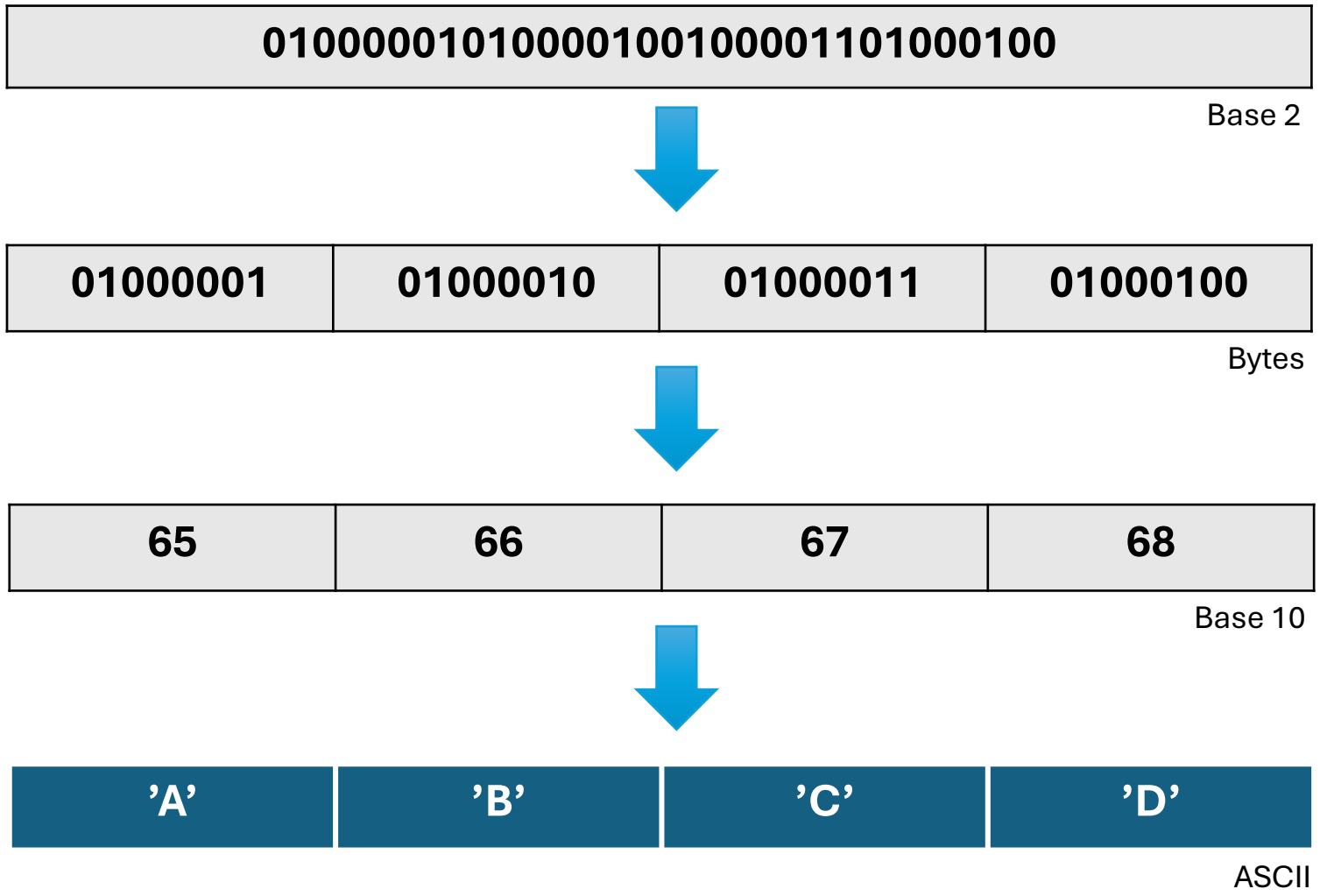
```
#include <arpa/inet.h>

int MSB = htonl(0x0A0B0C0D)
int LSB =  ntohl(x)
```

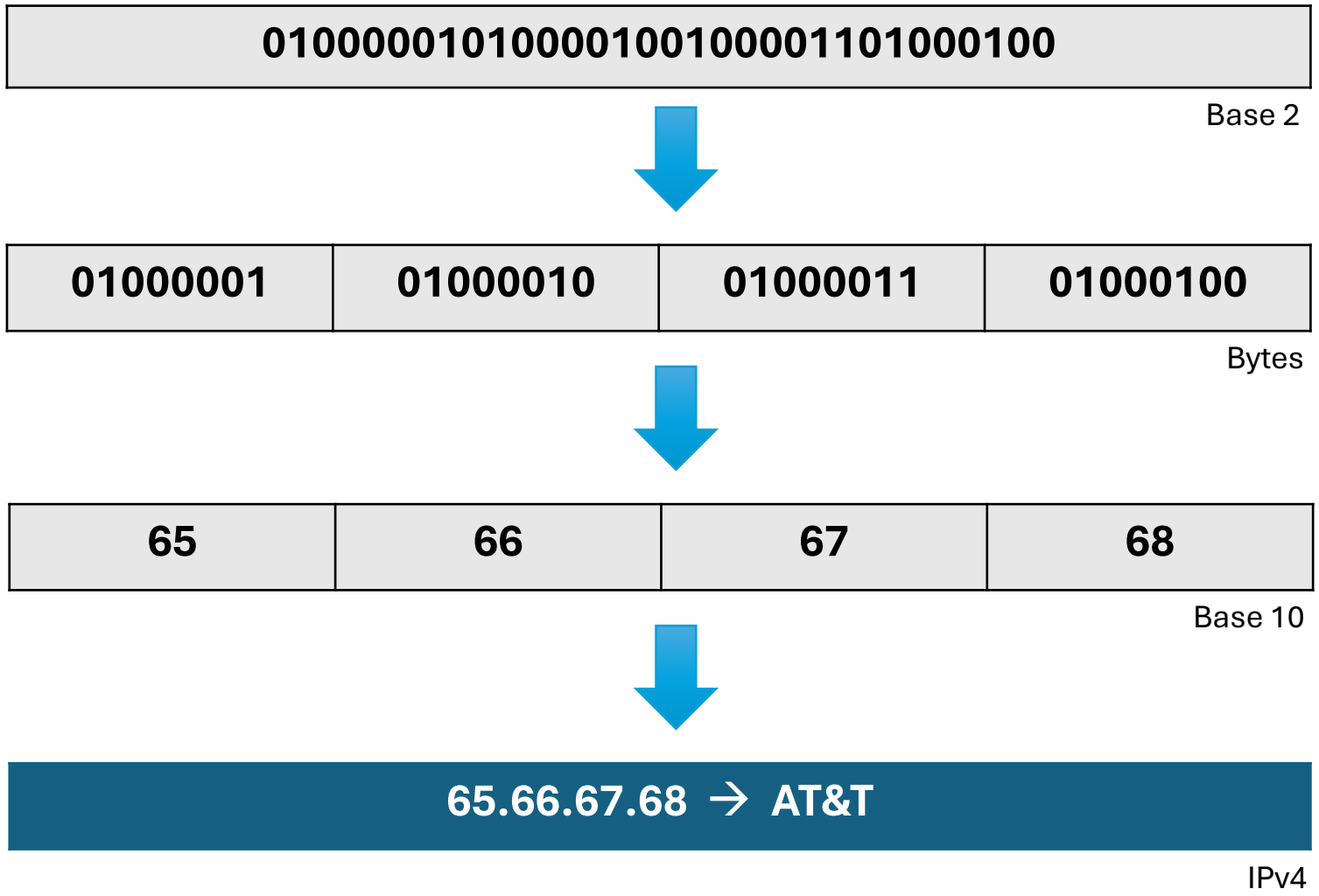
Python

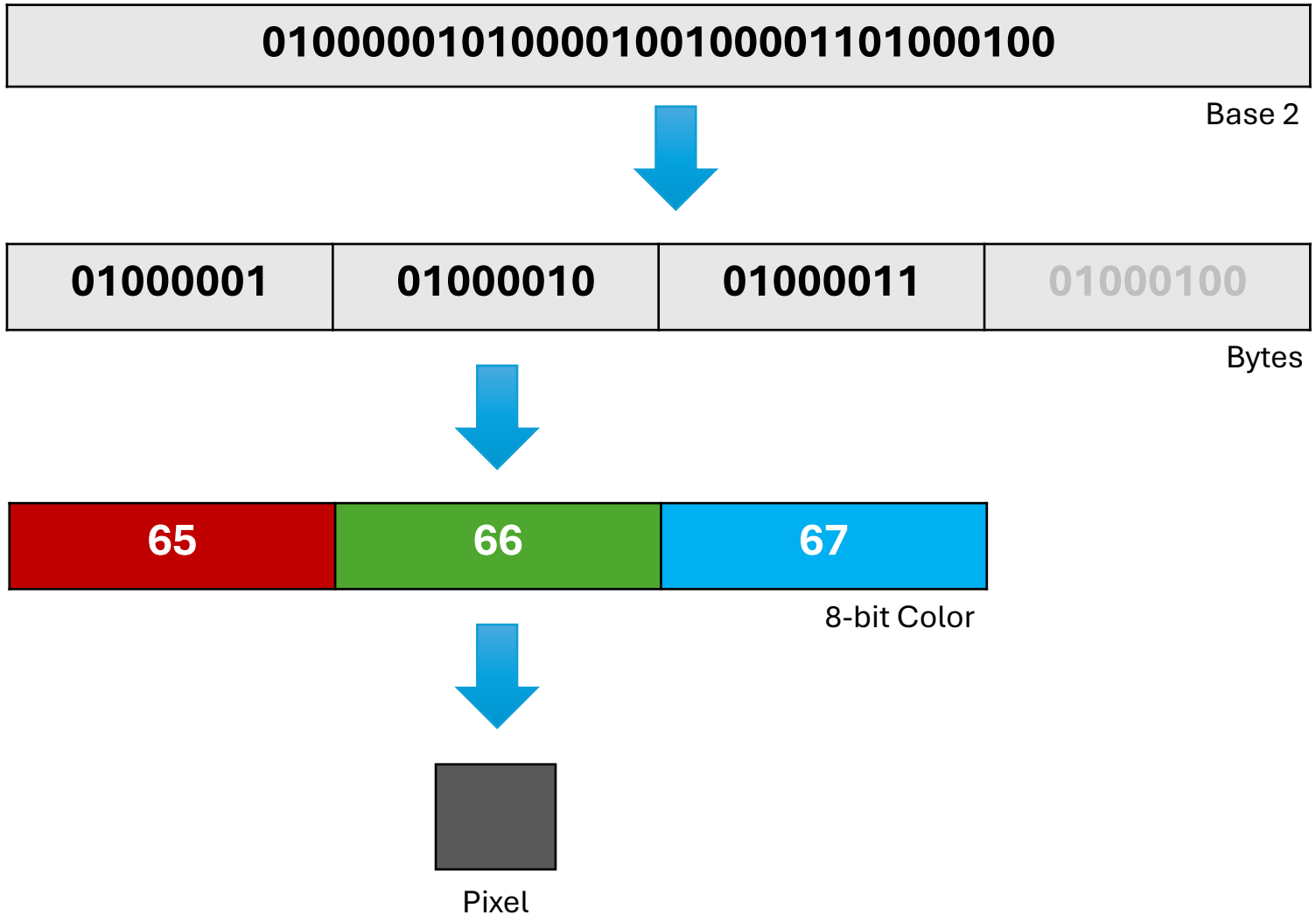
```
(0x0A0B0C0D).to_bytes(4,byteorder='big')
```

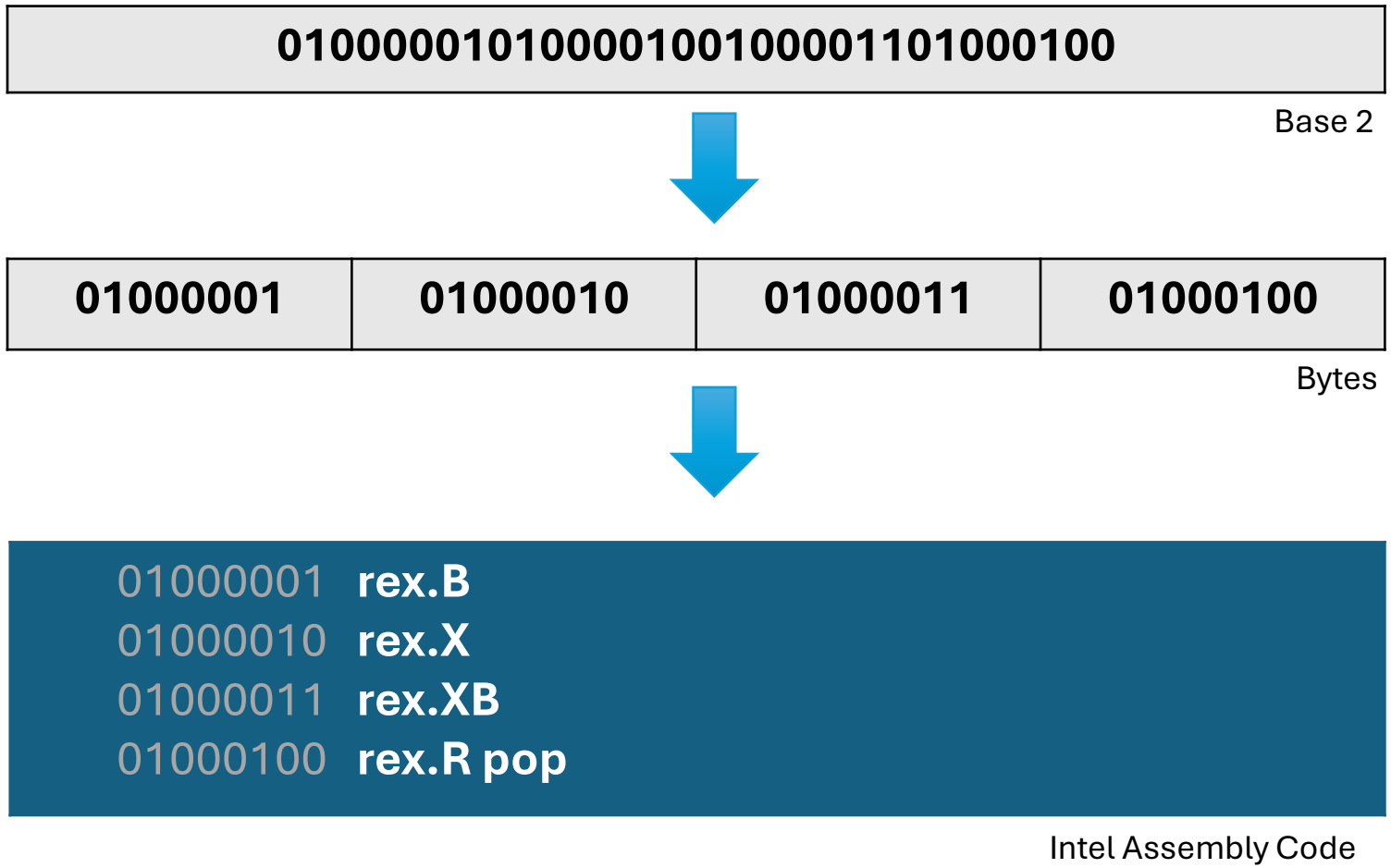

01000001010000100100001101000100
Base 2
0100
0001
0100
0010
0100
0011
0100
0100
Big-Endian
Little-Endian
0x41424344
0x44434241
Base 16
Base 16
1094861636
1145258561
Base 10
Base 10

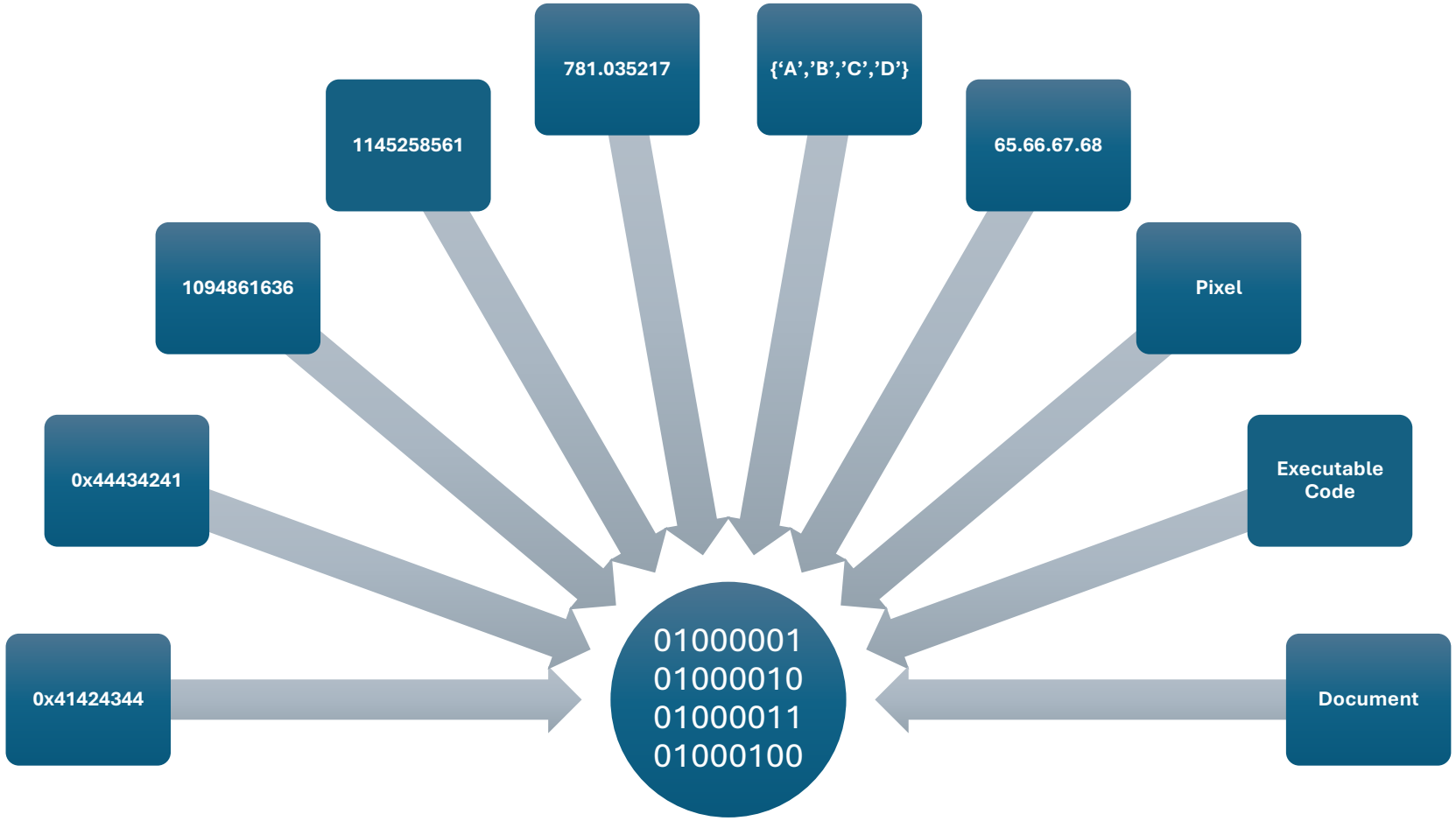

01000001010000100100001101000100
Base 2
0
1000001
01000010010000011010001 00
IEEE-754
781.035217
Base 10


01000001010000100100001101000100
Base 2
01000001
01000010
01000011
01000100
Bytes
65
66
67
68
Base 10
’A’
’B’
’C’
’D’
ASCII


01000001010000100100001101000100
Base 2
01000001
01000010
01000011
01000100
Bytes
65
66
67
68
Base 10
65.66.67.68 → AT&T
IPv4

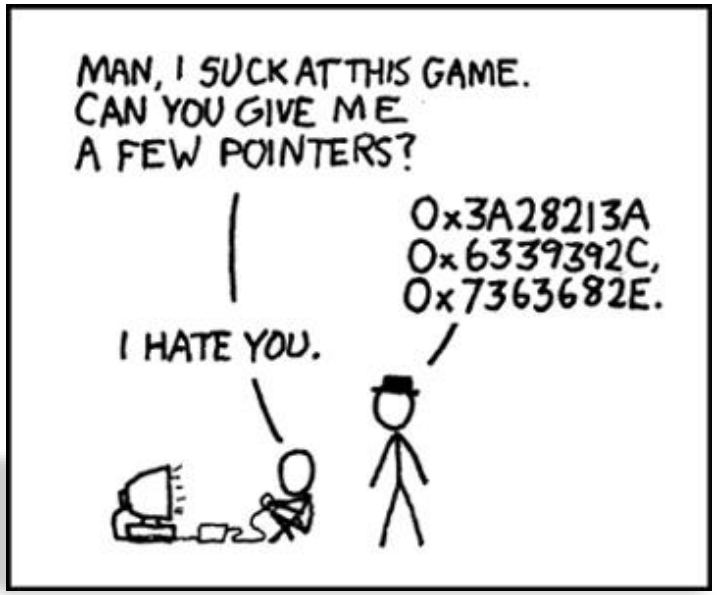

01000001010000100100001101000100
Base 2
01000001
01000010
01000011
01000100
Bytes
65
66
67
8-bit Color
Pixel

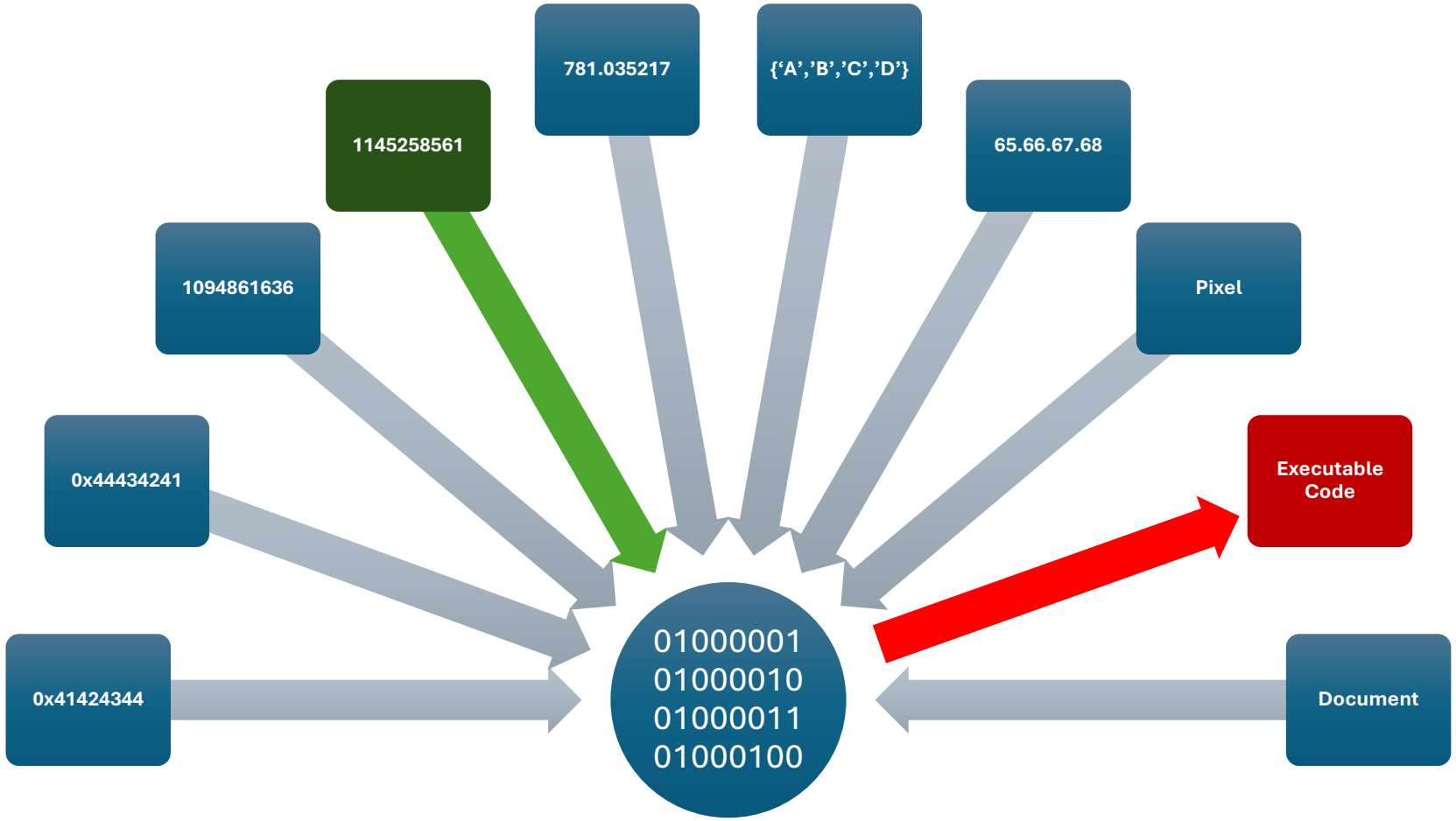

01000001010000100100001101000100
Base 2
01000001
01000010
01000011
01000100
Bytes
01000001 rex.B
01000010 rex.X
01000011 rex.XB
01000100 rex.R pop
Intel Assembly Code


781.035217
{‘A’,’B’,’C’,’D’}
1145258561
65.66.67.68
1094861636
Pixel
0x44434241
Executable Code
0x41424344
01000001
01000010
01000011
01000100
Document

Data Representation

What if our **assumptions** about how the data will be interpreted are **wrong**?





Many security exploits rely on data being interpreted in a **different way** than was originally intended.

```
unsigned long int i_hello = 0x6f6c6c6548;
unsigned long int i_world = 431316168567;

printf("%s %s”, (char *)&i_hello, (char *)&i_world);
```

ASCII	Hex	Dec	ASCII	Hex	Dec
A	41	65	a	61	97
B	42	66	b	62	98
C	43	67	c	63	99
D	44	68	d	64	100

Remember: Intel is **Little-Endian**; the bytes in an integer are stored in reverse order in memory.

Review

Intel Assembly Code

Intel Assembly Code

We will be making use of Intel assembly code, both to discuss and to create various security exploits.

You will not need to be an expert, but should be comfortable reading basic assembly code.

For reference, see:

http://www.cs.virginia.edu/~evans/cs216/guides/x86.html

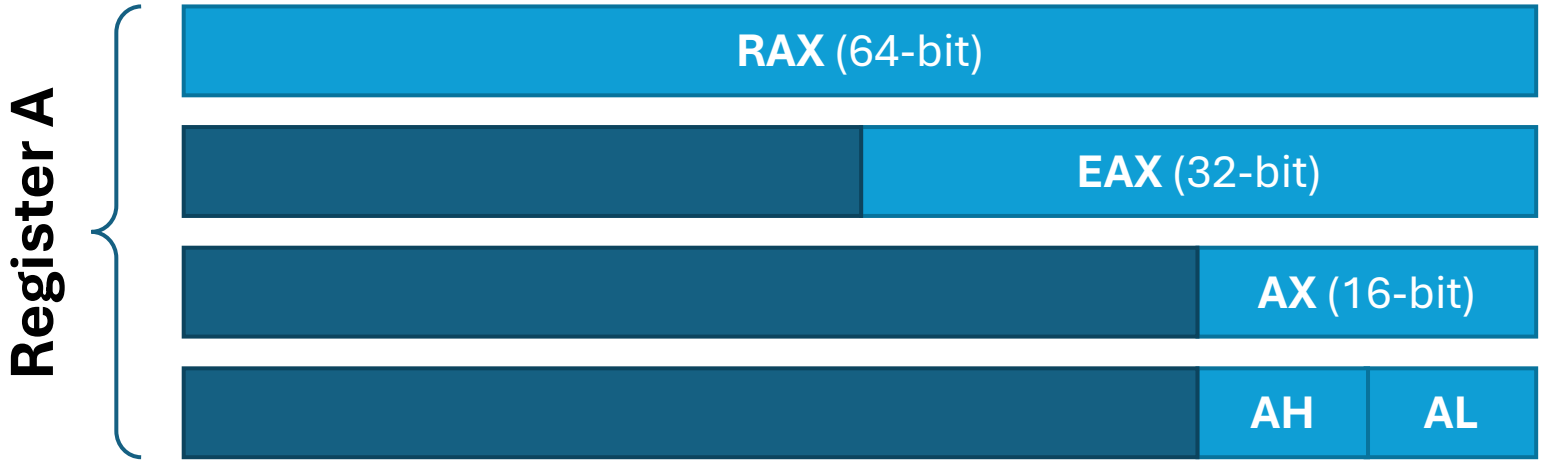
Intel Assembly Code

Intel assembly code looks very similar to most other assembly languages you may already be familiar with (e.g., Motorola 68k)

```
movl -4(%rbp), %eax
addl $1, %eax
movl %eax, -16(%rbp)
```

Intel Registers

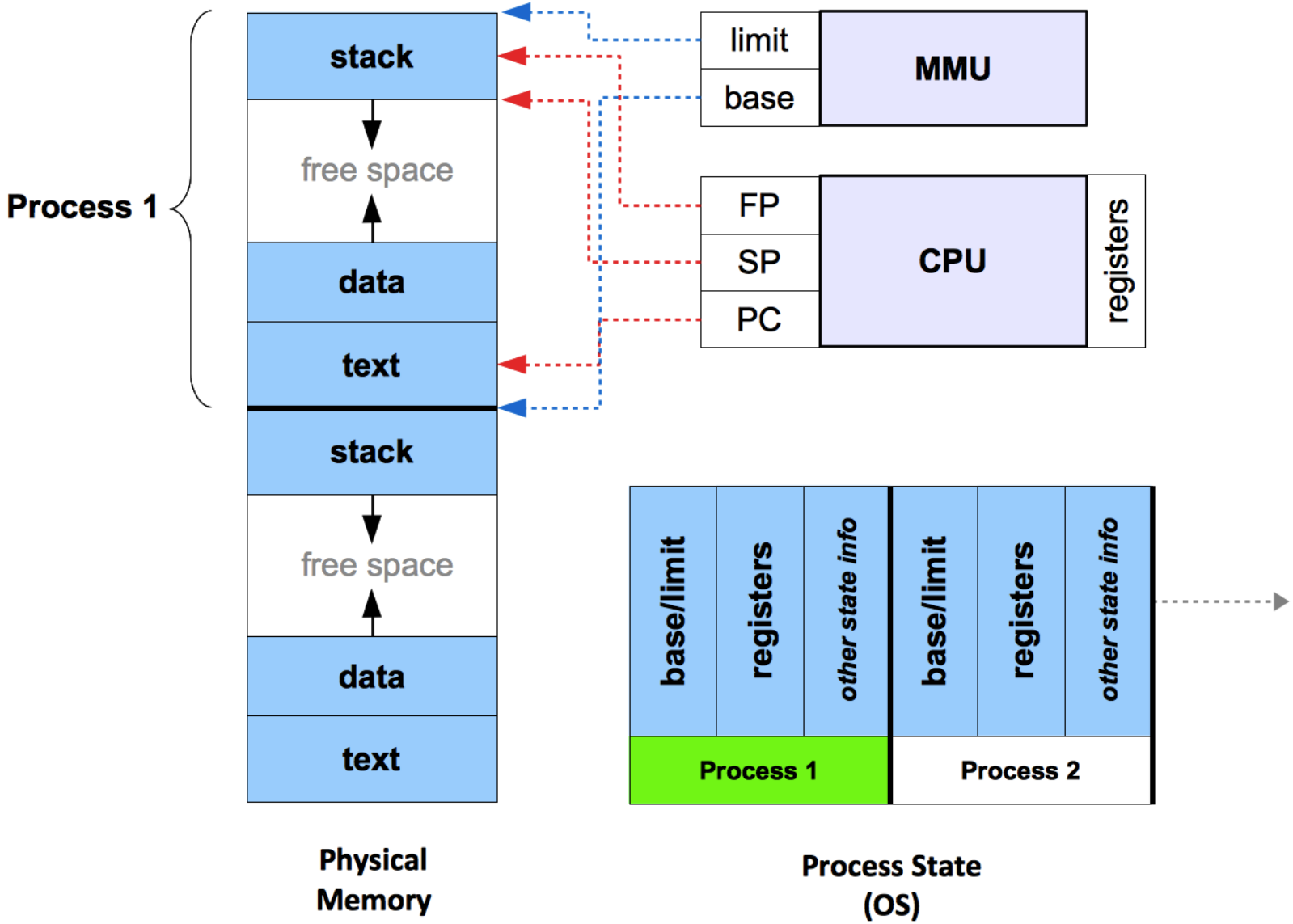
Current (64-bit) Intel CPUs have six general-purpose registers; the various bits in each individual register can be accessed in a number of ways:



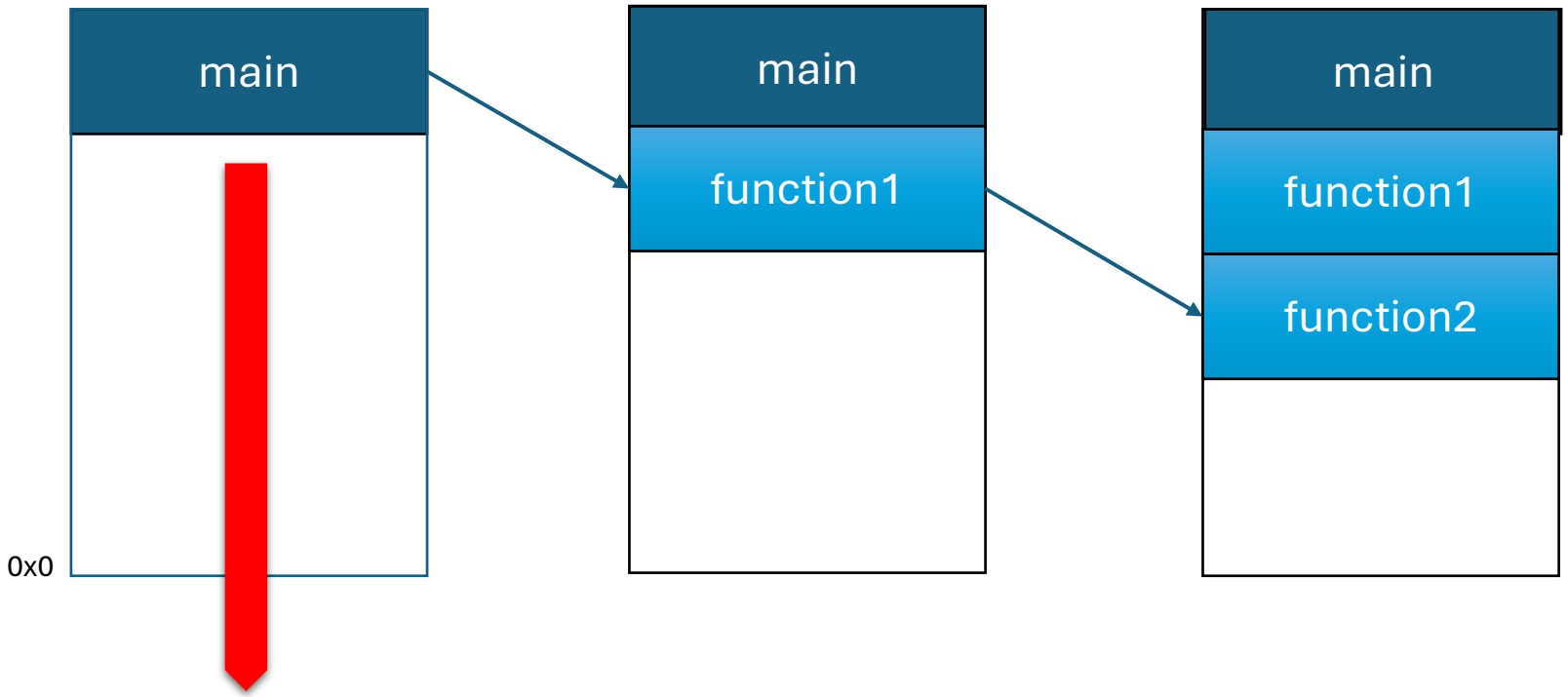
Review

Stack Frames and Function Calls

UNIVERSITY OF TORONTO


Process 1
stack
free space
data
text
stack
free space
data
text
limit
base
MMU
FP
SP
PC
CPU
registers
base/limit
registers
other state info
base/limit
registers
other state info
Process 1
Process 2
Physical Memory
Process State (OS)

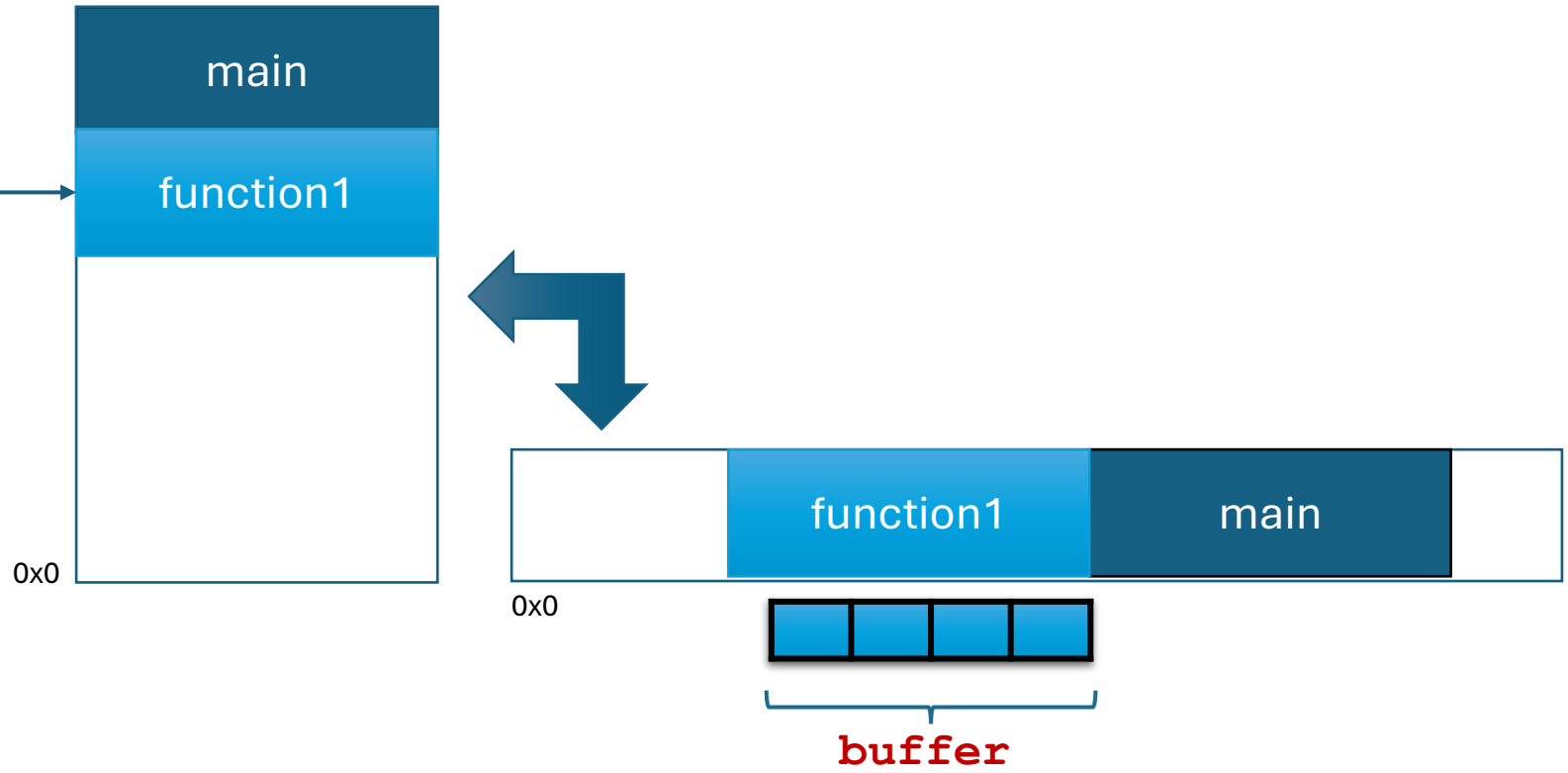
Process Stack



The process stack grows **downwards** on Intel.

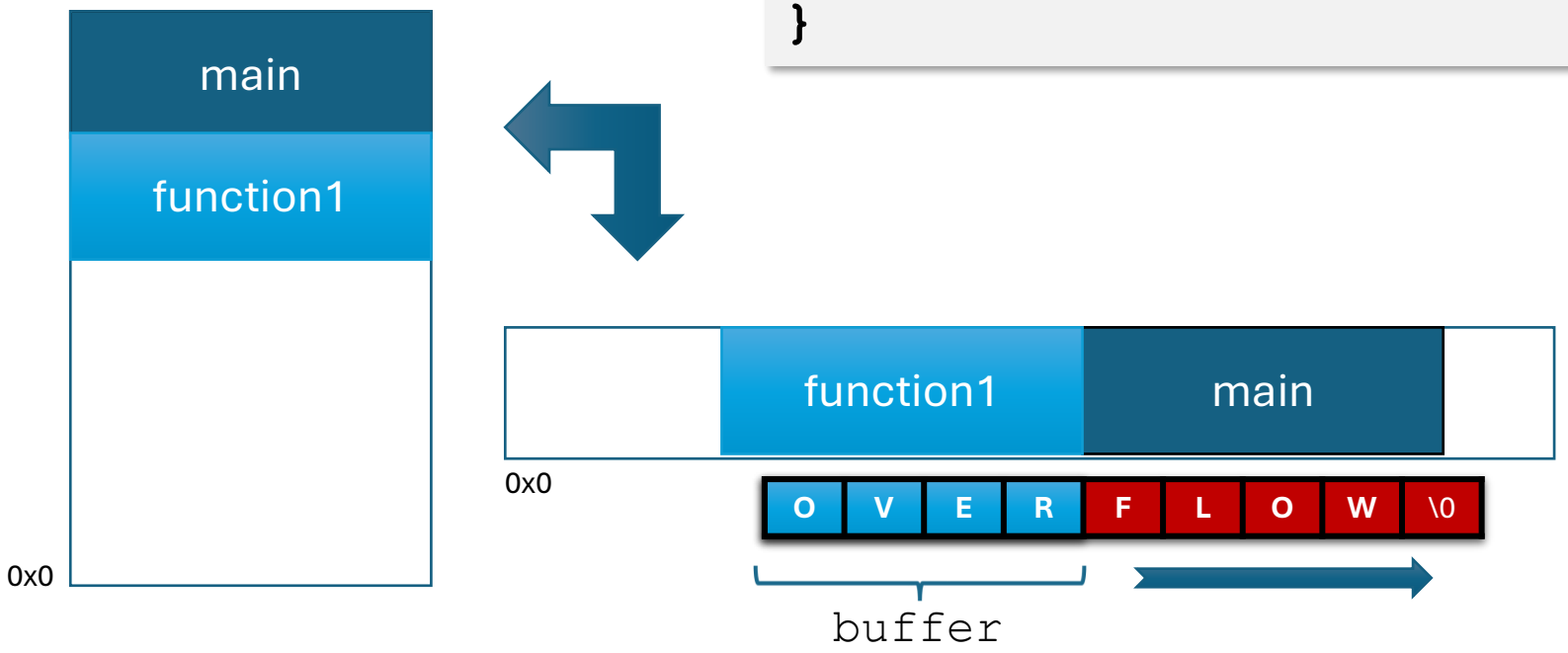
Process Stack

```
void function1()
{
    char buffer[4];
    …
}
```



Process Stack

```
void function1()
{
    char buffer[4];
    strcpy(buffer, "OVERFLOW");
}
```

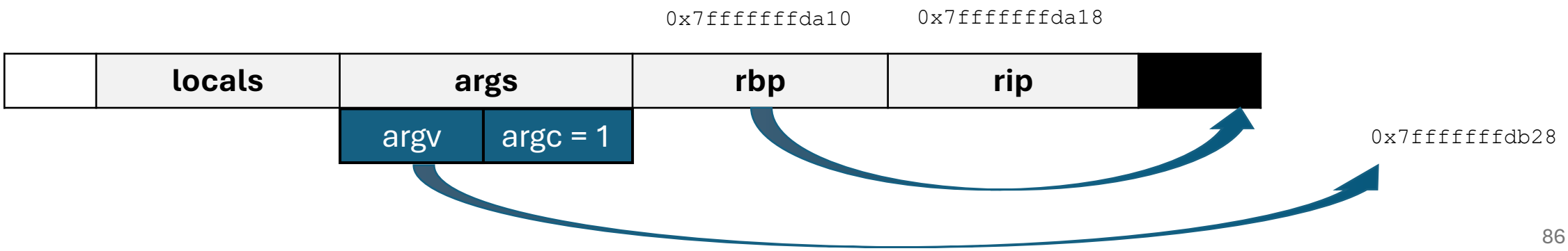


Using gdb

break	Defines a new breakpoint
run	Starts a new process
where	List of current stack frames
up / down	Move between frames
info frame	Display info on current frame
info args	List function arguments (“gcc –g”)
info locals	List all local variables (“gcc -g”)
print	Display a variable (“print x”)
x	Display the contents of memory (“x/10x array”)

gdb: info frame

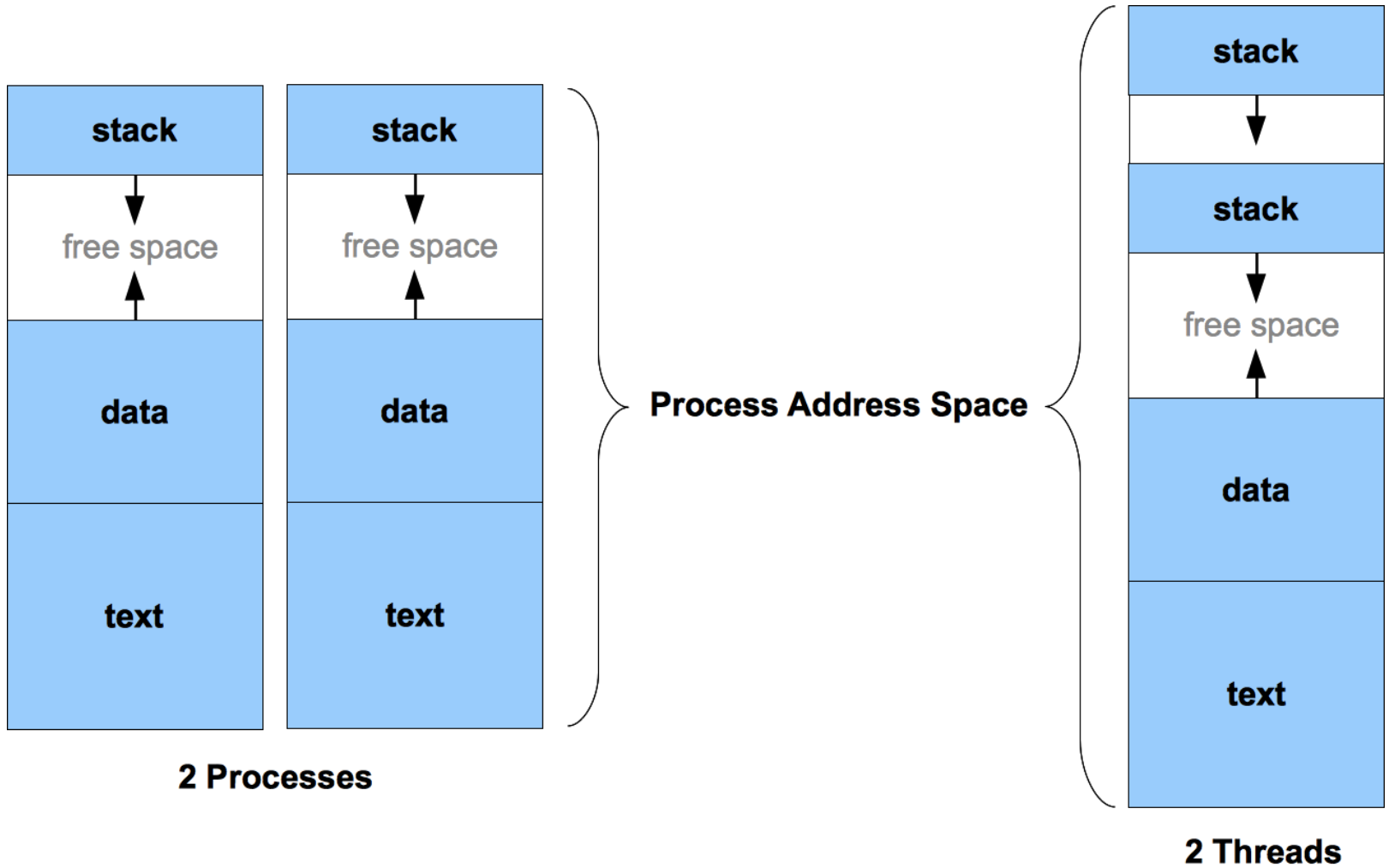
```
(gdb) info frame
Stack level 0, frame at 0x7fffffffda20:
 rip = 0x401844 in main (test.c:6); saved rip 0x401d14
 source language c.
 Arglist at 0x7fffffffda10, args: argc=1, argv=0x7fffffffdb28
 Locals at 0x7fffffffda10, Previous frame's sp is 0x7fffffffda20
 Saved registers:
  rbp at 0x7fffffffda10, rip at 0x7fffffffda18
(gdb)
```

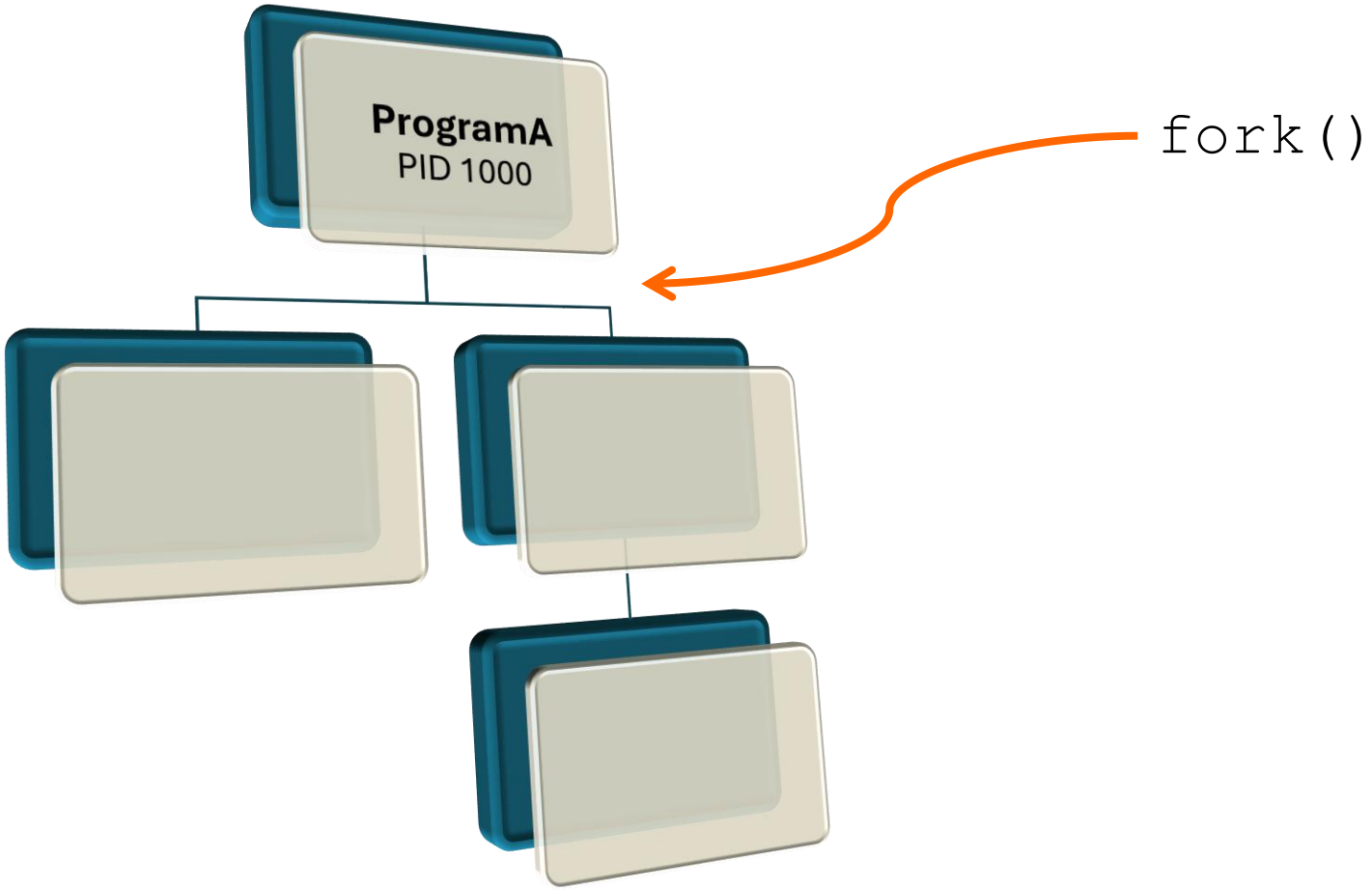


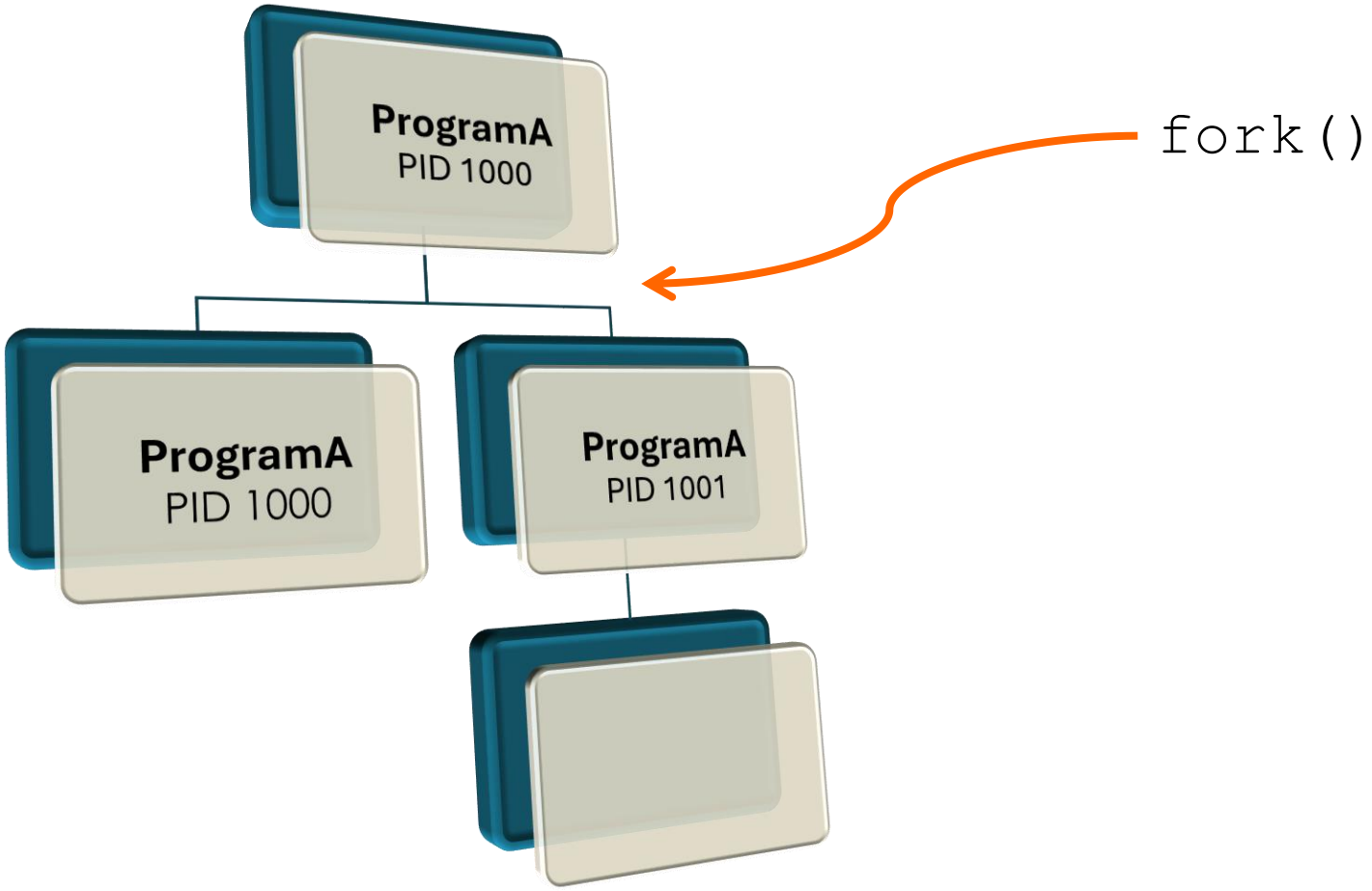
Review

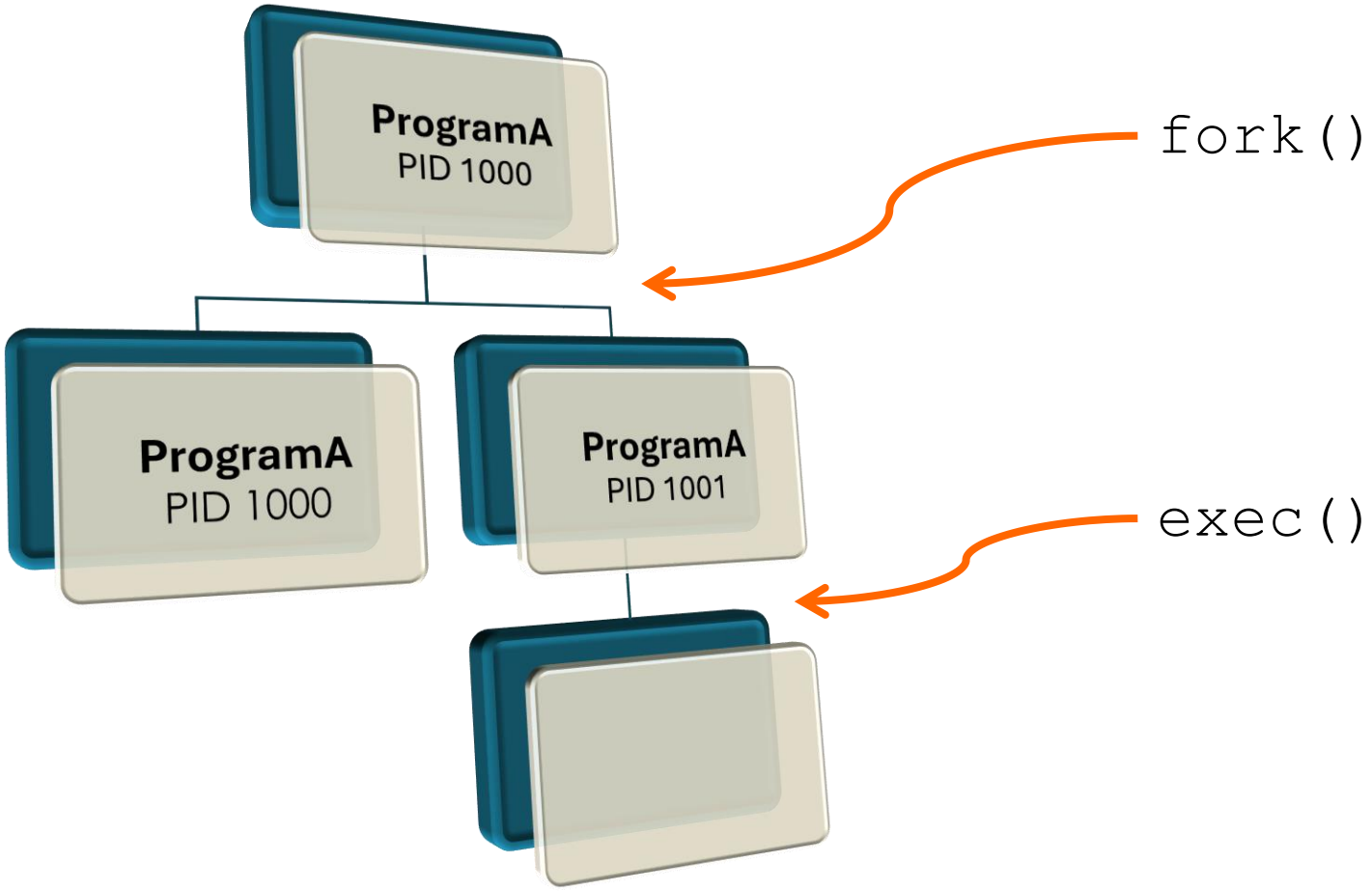
Processes, Threads, fork() and exec()

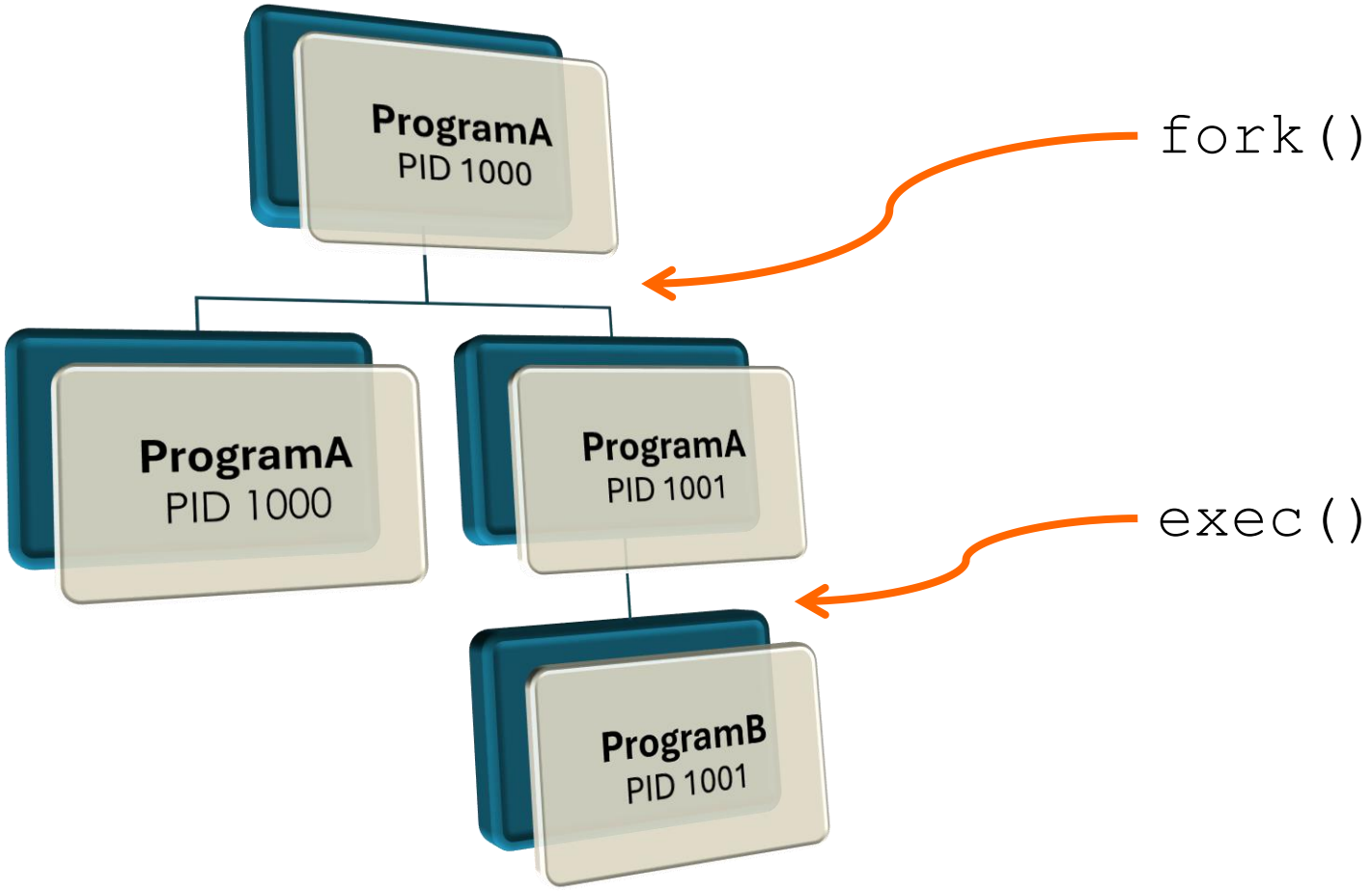
UNIVERSITY OF TORONTO


stack
free space
data
text
stack
free space
data
text
2 Processes
Process Address Space
stack
stack
free space
data
text
2 Threads


ProgramA
PID 1000
fork()


ProgramA
PID 1000
fork()
ProgramA
PID 1000
ProgramA
PID 1001


ProgramA
PID 1000
fork()
ProgramA
PID 1000
ProgramA
PID 1001
exec()


ProgramA
PID 1000
fork()
ProgramA
PID 1000
ProgramA
PID 1001
exec()
ProgramB
PID 1001

Questions?

UNIVERSITY OF TORONTO