Getting Help

Main Instructor: **Azzam Mourad <azzam.mourad@nyu.edu>**

- Office hours: online and in-person, MW: 11:15AM-12:30 pm or by appointment
- Zoom Link is provided in the syllabus

Teaching Assistant: **Khalid Mengal <kqm1@nyu.edu>**

- Office Hours: W: 2:00PM-3:00PM

Course Overview

Computer Systems Organization

Course Perspective

Most Systems Courses are Builder-Centric

- Computer Architecture
  - Design pipelined processor in Verilog
- Operating Systems
  - Implement large portions of operating system
- Compilers
  - Write compiler for simple language
- Networking
  - Implement and simulate network protocols

Course Perspective (Cont.)

This course is **programmer-centric**

- Understanding of underlying system makes a more effective programmer
- Bring out the hidden hacker in everyone

Textbooks

Randal E. Bryant and David R. O’Hallaron,

“Computer Systems: A Programmer’s Perspective, 3rd Edition”, Prentice Hall

Brian Kernighan and Dennis Ritchie,

“The C Programming Language, 2nd Edition”, Prentice Hall

Course Components

Lectures

- Higher level concepts

Programming Assignments/Labs (4)

- The heart of the course
- Provide in-depth understanding of some aspect of systems

In-class Quizzes (2)

One midterm

One final exam

Course Syllabus

- C Programming
- Data representation and manipulation (bit, int, float …)
- Assembly and Program Representation
- Memory hierarchy
- Program optimization and parallelism
- Virtual memory
- Linking

Abstraction Is Good But Don’t Forget Reality

Most CS Courses emphasize abstraction

Goal of CSO:

- Help you understand how computers work and build bug-free/efficient software and system programs

Why is it important?

- Fundamental computer science
- Become better programmer
  - Avoid bugs
  - Write fast code
  - Write secure code
  - Write low-level/system code
- Prepare for later “systems” classes in CS
  - Operating Systems, Networking, Computer Architecture, Distributed Systems, Compilers

So, why CSO?

Reason #1: Understanding Internal Representations is Important

$x^2 \geq 0$?

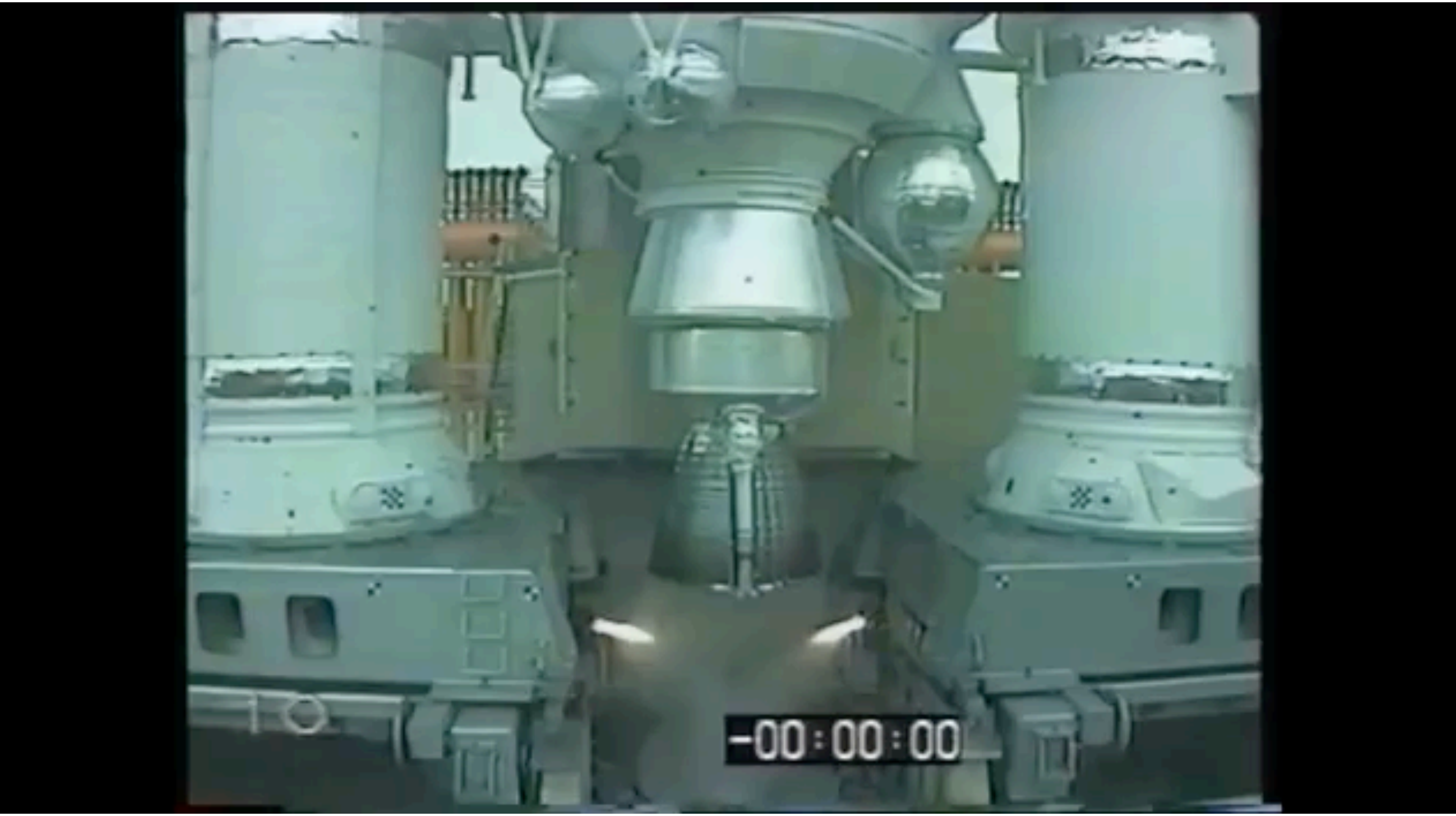
- 40000 * 40000 =
- 50000 * 50000 = ?

(x + y) + z = x + (y + z)?

- $\boxed{(1e20 - 1e20)} + 3.14 = 3.14$
- 1e20 - (1e20 + 3.14) = ?

Demo!

Ariane 5 Rocket



Reason #2: Knowing Assembly is Important

No need to program in assembly

Knowledge of assembly helps one understand machine-level execution

- Creating/fighting malware
- Debugging
- Writing system software (e.g. compilers , OS)

Reason #3: Speed is Very Important

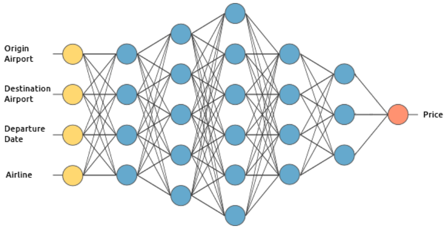
Compute Intensive Applications are Everywhere

Scalable Web Applications

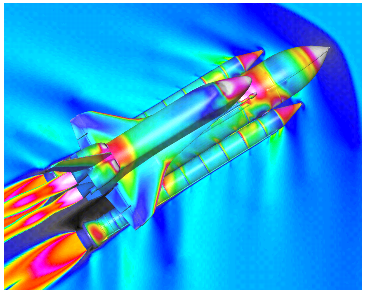
Scalable Mobile Applications

Image Processing

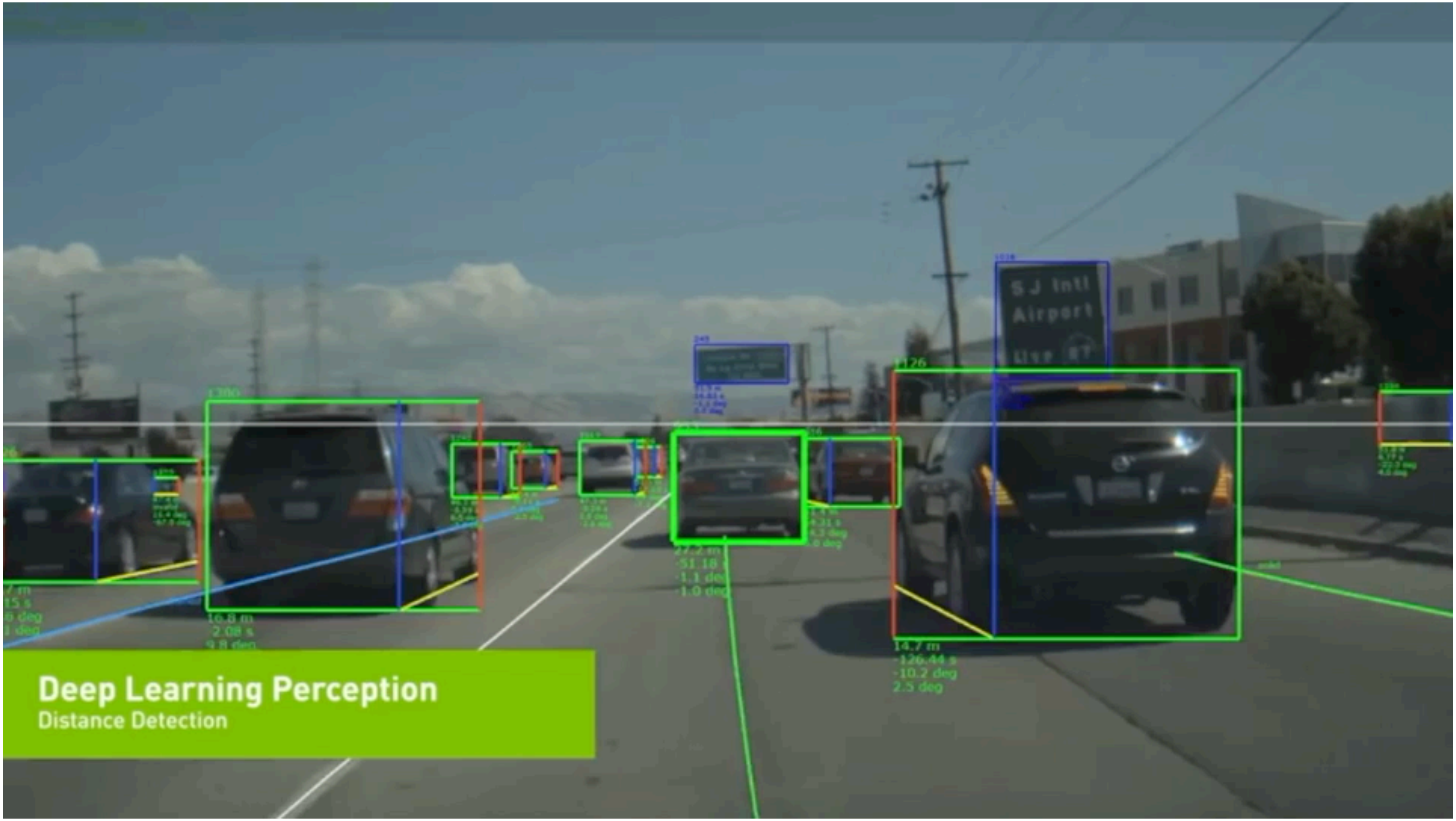
Bioinformatics



Deep Learning

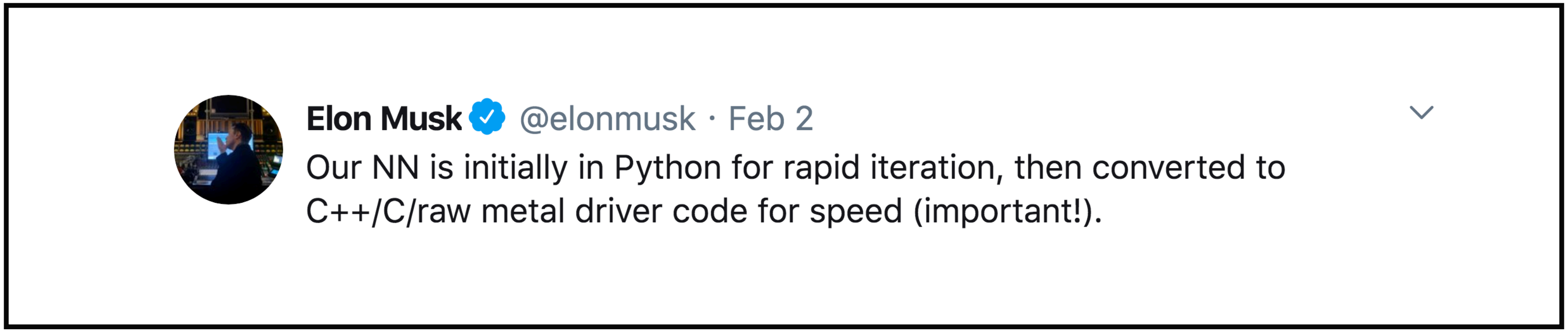


Scientific Computing



Driverless cars

Code Optimization in C



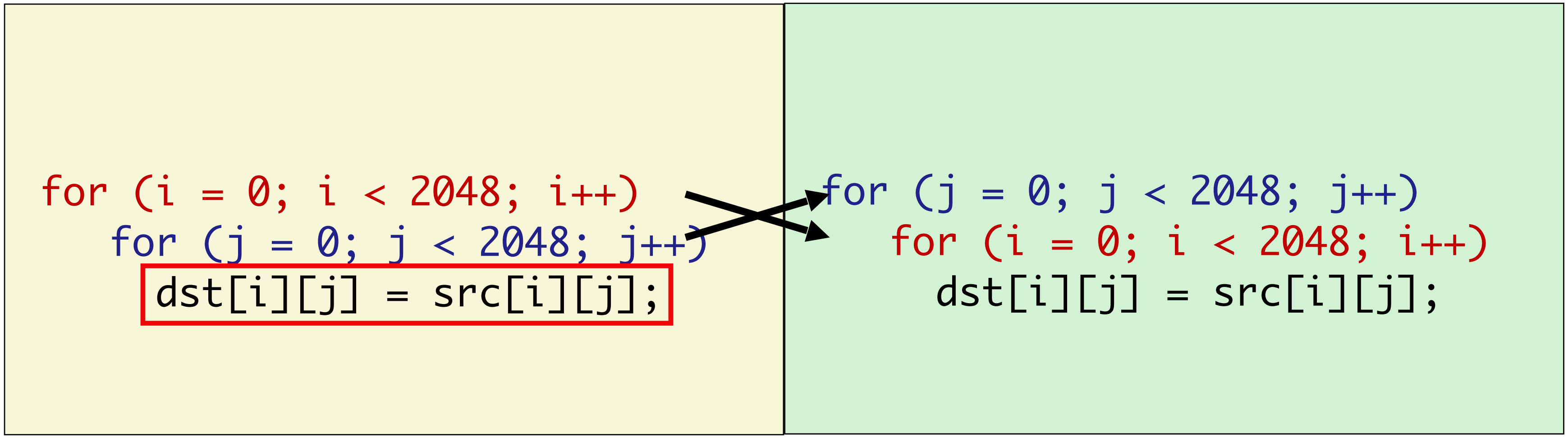
Elon Musk ✓ @elonmusk · Feb 2

Our NN is initially in Python for rapid iteration, then converted to C++/C/raw metal driver code for speed (important!).

C is a good balance between high speed and productivity

Memory System Performance Example

Is the right code faster?



```
for (i = 0; i < 2048; i++)
    for (j = 0; j < 2048; j++)
        dst[i][j] = src[i][j];
```

```
for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
        dst[i][j] = src[i][j];
```

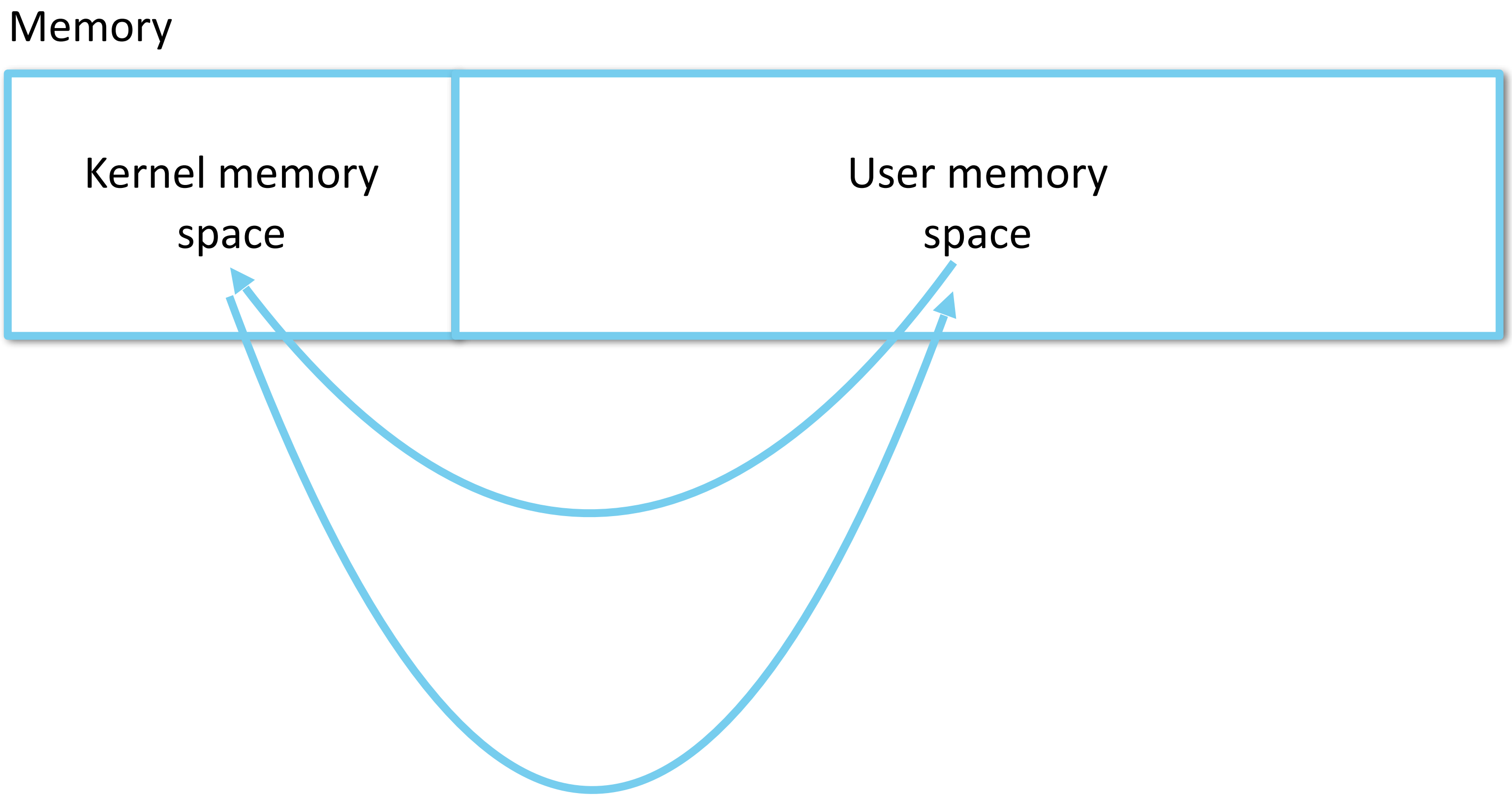
21 times slower

Performance depends on access patterns

Reason #4: Important for Security

Understanding how computers work is important for securing your code/hacking codes

Access to Kernel Memory



Memory

Kernel memory space

User memory space

Memory Referencing Errors

C/C++ let programmers make memory errors

- Out of bounds array references
- Invalid pointer values
- Double free, use after free

Errors can lead to nasty bugs

- Corrupt program objects
- Effect of bug observed long after the corruption

Memory Referencing Bug Example

```
double fun(int i)
{
  double d[1] = {3.14};
  int a[2];
  a[i] = 1073741824; /* Possibly out of bounds */
  return d[0];
}
```

```
fun(0)  ➙    3.14
fun(1)  ➙    3.14
fun(2)  ➙    ?
fun(3)  ➙    ?
fun(4)  ➙    ?
```

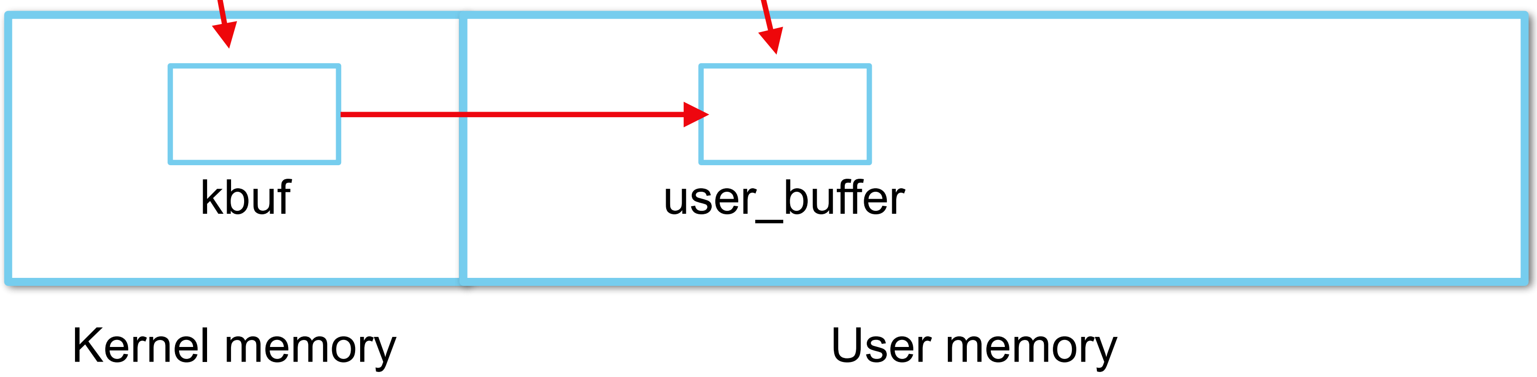
Code Security Example

```
/* Kernel memory region holding user-accessible data */
#define KSIZE 1024
char kbuf[KSIZE];

/* Copy data_amount bytes from kernel region to user buffer */
int copy_from_kernel(void *user_buffer, int data_amount)
{
    /* Byte count len is minimum of buffer size and maxlen */
    int len = data_amount > KSIZE ? KSIZE : data_amount;
    memcpy(user_buffer, kbuf, len);
    …
}
```

- Similar to code found in FreeBSD’s implementation of getpeername (get name of connected peer socket)
- There are legions of smart people trying to find weaknesses in programs

Code Security Example

```
/* Kernel memory region holding user-accessible data */
#define KSIZE 1024
char kbuf[KSIZE];

/* Copy data_amount bytes from kernel region to user buffer */
int copy_from_kernel(void *user_buffer, int data_amount)
{
    /* Byte count len is minimum of buffer size and maxlen */
    int len = data_amount > KSIZE ? KSIZE : data_amount;
    memcpy(user_buffer, kbuf, len);
    …
}
```



Typical Usage

```
/* Kernel memory region holding user-accessible data */
#define KSIZE 1024
char kbuf[KSIZE];

/* Copy data_amount bytes from kernel region to user buffer */
int copy_from_kernel(void *user_buffer, int data_amount)
{
    /* Byte count len is minimum of buffer size and maxlen */
    int len = data_amount > KSIZE ? KSIZE : data_amount;
    memcpy(user_buffer, kbuf, len);
    …
}
```

```
#define MSIZE 528

void getstuff() {
    char mybuf[MSIZE];
    copy_from_kernel(mybuf, MSIZE);
    printf(“%s\n”, mybuf);
}
```

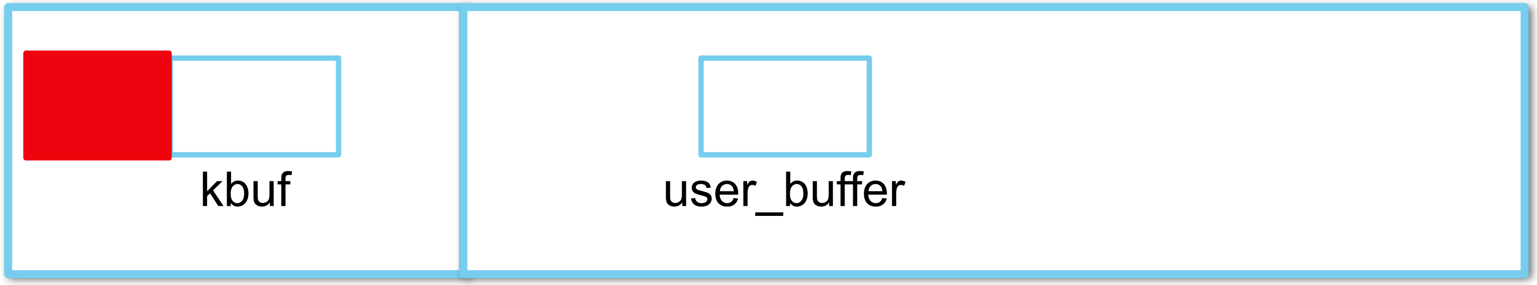
Malicious Usage

```
/* Kernel memory region holding user-accessible data */
#define KSIZE 1024
char kbuf[KSIZE];

/* Copy data_amount bytes from kernel region to user buffer */
int copy_from_kernel(void *user_buffer, int data_amount)
{
    /* Byte count len is minimum of buffer size and maxlen */
    int len = data_amount > KSIZE ? KSIZE : data_amount;
    memcpy(user_buffer, kbuf, len);
    …
}
```

```
#define MSIZE 528

void getstuff() {
    char mybuf[MSIZE];
    copy_from_kernel(mybuf, -MSIZE);
    . . .
}
```

Malicious Usage

```
/* Kernel memory region holding user-accessible data */
#define KSIZE 1024
char kbuf[KSIZE];

/* Copy data_amount bytes from kernel region to user buffer */
int copy_from_kernel(void *user_buffer, int data_amount)
{
    /* Byte count len is minimum of buffer size and maxlen */
    int len = data_amount > KSIZE ? KSIZE : data_amount;
    memcpy(user_buffer, kbuf, len);
    …
}
```



Kernel memory

User memory

Policies: Assignments (Labs)

You must work alone on all assignments

- Post all questions on the forum
- You are encouraged to answer others’ questions, but refrain from explicitly giving away solutions

Hand-ins

- Assignments due at 11:55pm on the due date
- Late submissions: 10% deducted each late day (maximum 3 days)
- Two grace days
- Zero score if a lab is handed in > 3 days late

UNIX Lab Environment

Use official class VM image

- Download (free) virtualbox for Windows/Linux
- Download VM appliance from course web page

Your assignments must work on this environment!

Cheating

What is cheating?

- Sharing code: by copying, looking at others’ files
- Coaching: helping your friend write a lab step by step
- Copying code from a previous course or from elsewhere

Penalty for cheating:

- Immediate removal from course with failing grade
- Permanent mark on your record