Introduction

Hi, I'm Randy

- Clinical Assistant Professor, Courant Institute of Mathematical Sciences, New York University
- Technical Consultant @ Giphy
- Former Director of Engineering @ Tapad
- Co-Organizer of the NY-Scala Meetup
- 15 years of professional software engineering experience.

Why I am here

- Second semester as full-time faculty.
- I love to teach! Especially excited for this class.
- I am also here as a resource to you as you navigate your education.
- Thinking about an internship? Thinking about a career in CS? Doing a a technical interview? I can give you some advice.

How to reach me

- **Email**: rjs471@nyu.edu
- **Skype**: randy.j.shepherd
- **Office Hours**: Monday & Wednesday, 4:45PM-6:15PM @ WWH 425
  - If you want to meet outside these office hours, there is a protocol for this posted on the website under “Protocols & Policies”

+

Course Content

Abstraction Is Good But Don't Forget Reality

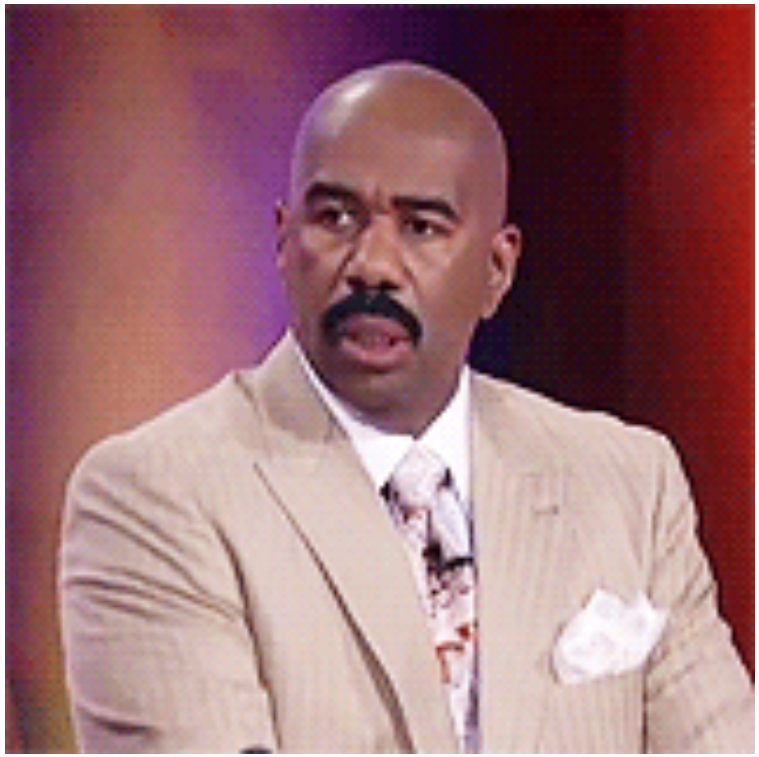
- Most CS course emphasize abstraction
  - Abstract data types
  - Asymptotic analysis
- These abstractions have limits
  - Need to understand details of underlying system.
- You will depend on this knowledge to write real-world, high-performance systems.
- **Reality** can sneak up on you in unexpected ways if you are not aware.

Reality #1: Integral types are not Integers, Floating point types are not Reals

- **Example**: Is $x^2 \geq 0$?
  - for integral types?
    - 40000 * 40000 → 1600000000
    - 50000 * 50000 → ??
  - for floating point types, yes!

Reality #1: Integral types are not Integers, Floating point types are not Reals

- **Example**: Is (x + y) + z = x + (y + z)?
  - for floating point types?
    - (1e20 + -1e20) + 3.14 --> 3.14
    - 1e20 + (-1e20 + 3.14) --> ??
  - for integral types? yes!



Reality #2: You've Got to Know Assembly

- We won't be writing assembly. (probably)
- However, knowledge of assembly helps one understand machine-level execution and allows for..
  - Debugging
  - Performance tuning
  - Writing system software (e.g. compilers , OS)
  - Concurrency and parallelism
- x86 assembly is the language of choice!

Reality #3: Memory Matters

- Memory is not unbounded. It must be allocated and managed
- Memory referencing bugs are especially challenging to debug.
- Not all memory is created equal!
  - There are a number of types of memory with different characteristics.
  - Memory performance is not uniform across these types
  - **Cache** and **virtual memory** effects can greatly affect performance

Memory Referencing Bug Example

```
typedef struct {
  int a[2];
  double d;
} struct_t;

double fun(int i) {
  volatile struct_t s;
  s.d = 3.14;
  s.a[i] = 1073741824; /* Possibly out of bounds */
  return s.d;
}
```

Memory Referencing Bug Example

```
typedef struct {
  int a[2];
  double d;
} struct_t;

double fun(int i) {
  volatile struct_t s;
  s.d = 3.14;
  s.a[i] = 1073741824; /* Possibly out of bounds */
  return s.d;
}
```

```
fun(0)  →  3.14
fun(1)  →  3.14
fun(2)  →  3.1399998664856
fun(3)  →  2.00000061035156
fun(4)  →  3.14
fun(6)  →  Segmentation fault
```

* Results are system dependent

Memory Referencing Bug Example

```
typedef struct {
  int a[2];
  double d;
} struct_t;

double fun(int i) {
  volatile struct_t s;
  s.d = 3.14;
  s.a[i] = 1073741824; /* Possibly out of bounds */
  return s.d;
}
```

```
fun(0)  →  3.14
fun(1)  →  3.14
fun(2)  →  3.1399998664856
fun(3)  →  2.00000061035156
fun(4)  →  3.14
fun(6)  →  Segmentation fault
```

* Results are system dependent

Memory Referencing Bug Example

```
typedef struct {
  int a[2];
  double d;
} struct_t;
```

```
fun(0)  →  3.14
fun(1)  →  3.14
fun(2)  →  3.1399998664856
fun(3)  →  2.0000000610351 56
fun(4)  →  3.14
fun(6)  →  Segmentation fault
```

Explanation:

`struct_t`

Critical State	6
?	5
?	4
d7 ... d4	3
d3 ... d0	2
a[1]	1
a[0]	0

Location accessed by `fun(i)`

Memory Referencing Errors

- C/C++ let programmers make memory errors
  - Out of bounds array references
  - Invalid pointer values
  - Double delete, usage after deallocation
- Errors can lead to nasty bugs
  - Effect of bug observed long after the corruption
  - Effect of bug may be first observed long after it is generated
- How can I deal with this?
  - Program in Java, Ruby, Python, ML, …
  - Understand what possible interactions may occur
  - Use or develop tools to detect referencing errors (e.g. Valgrind)

Reality #4: Asymptotic performance is not always sufficient

- Constant factors matter
- Even operation count might not predict performance
- Must understand system to optimize performance
  - How programs compiled and executed
  - How to measure performance and identify bottlenecks
  - How to improve performance without destroying code modularity and generality

Memory System Performance Example

```
void copyij(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (i = 0; i < 2048; i++)
    for (j = 0; j < 2048; j++)
      dst[i][j] = src[i][j];
}
```

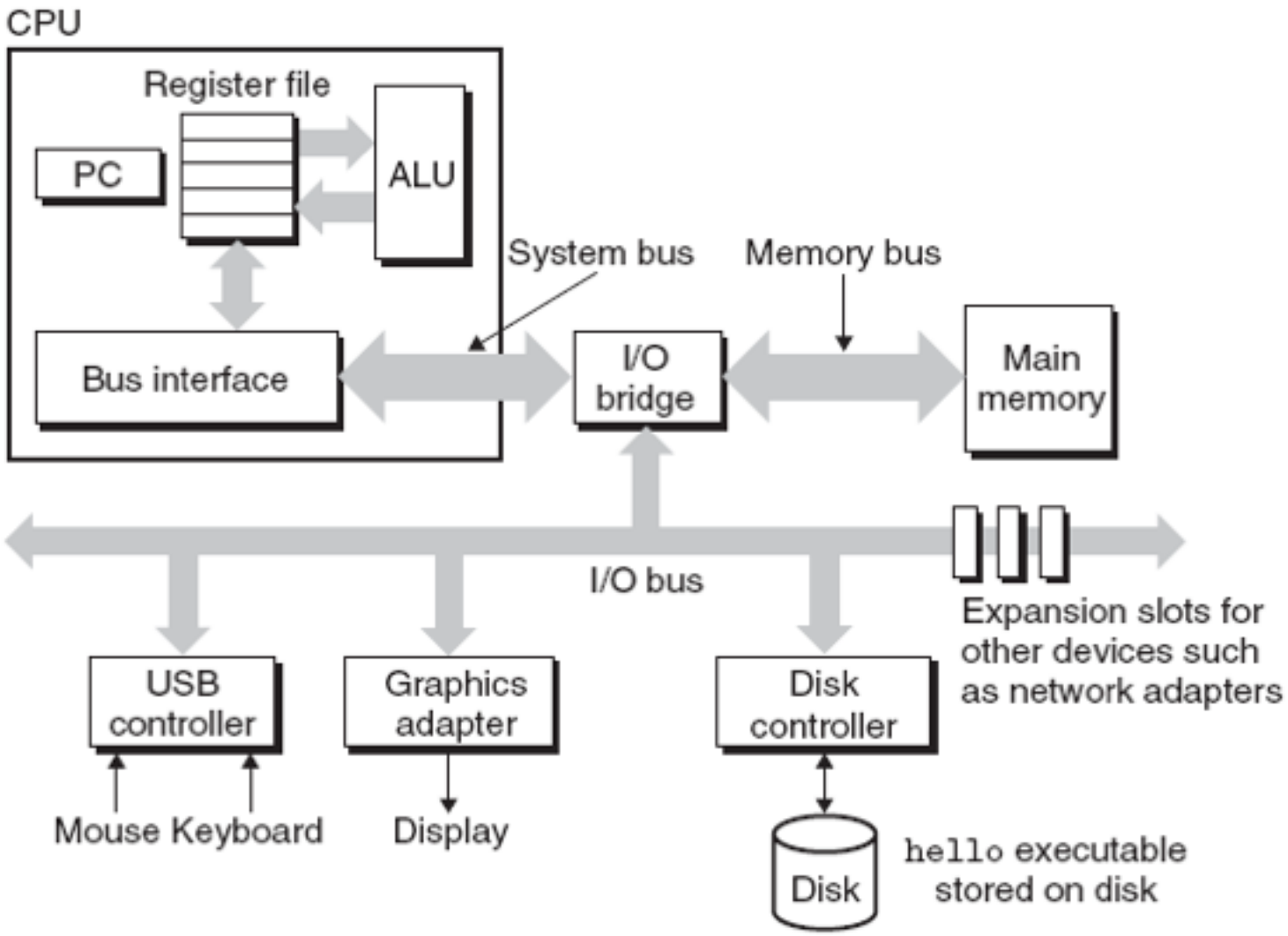
```
void copyji(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

- In one benchmark, right side is **20 times slower**
- Performance can depend on access patterns

Reality #5: Computer is more than the CPU

- They need to do I/O (get data in and out)
  - They communicate with each other over networks
- Concurrent operations by autonomous processes
- Reading from and writing to unreliable media
- Many components collaborating
- This all leads to complex performance issues

Reality #5: Computer is more than the CPU



Reality #5: Computer is more than the CPU

+

Course Perspective

Goals

- To understand…
  - what happens ‘under the hood’ when a computer boots and runs programs
  - how software and hardware interact
  - how to write performant code
  - how to inform your debugging with systems-level understanding
  - how to program in C
  - how to do *nix command-line kung fu

Perspective

- Most systems courses are *builder-centric*, ex.
  - Computer Architecture
    - Learn how systems are physically engineered
  - Operating Systems
    - Implement sample portions of operating system
  - Compilers
    - Write compiler for simple language
- This course is *programmer-centric*

Motivation

- Understanding of underlying system makes a more effective programmer, allowing you to…
  - Write programs that are more reliable and efficient
  - Incorporate features that interface with the OS (e.g. concurrency)
  - Become not just a programmer but an engineer and a hacker.
- And if you choose, it will prepare you well for later "systems" classes
  - Compilers, Operating Systems, Networks, etc.

Operational Details

To the website!

- I obsessively-compulsively update the website with course details
- So rather than repeating everything I have put there, lets just look it together.

Expectations

Come to class & recitation

- There is a *strong* correlation between those that attend class and their grade.
- What will appear on the test is what I think is important. What I think is important, I will tell you in class.
- This class is considered difficult by many students.
- I will have attendance taken at recitations. I will in lecture if I feel attendance becomes an issue.

Participate

- I am going to say things you do not understand. If you do not understand it, others do not also.
- It is a lot more fun when we talk and interact.
- Plus, it gets you the benefit of the doubt. If you are a half-point away from an A, and you show engagement in class, guess what happens when I report your grade?

A psychic reading & a look into the future

- I sense….
  - some of you have written C before
  - or programmed your first text-based adventure game at the age of 2
- I predict….
  - some of those people will not come to class and be very surprised come test time.
- Once you start running late, it's really hard to catch up. REALLY HARD.

Algorithm for success in CSO

- Come to class.
- Download materials from class from website.
- Study and experiment with those materials
- Read book to fill in gaps in understanding.
- *ABCW* - Always Be Checking the Website
- Come to recitation. And do the exercises.
- Use the TA.
- Come to office hours.

+

Homework

Tonight!

- Read every post on Piazza!!
- Follow the instructions posted here by **tonight**
  https://github.com/nyu-cso-s16/github-setup
  - Then accept invite to join 'organization' from Github.
  - (You must have a Github account in order to participate in tomorrow's very important recitation.)
- Bring your laptop to recitation!!
- Start reading KR ch. 1,2,3,4