Introduction to Computer Systems

CSCI3240: Lecture 1

Dr. Arpan Man Sainju

Middle Tennessee State University

Course Theme

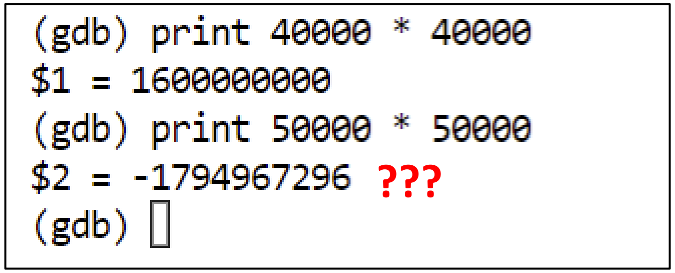
- **Abstraction Is Good But Don't Forget Reality**

- **Abstractions have limits**
  - Especially in the presence of bugs
  - Need to understand details of underlying implementations

- **Useful outcomes from taking CSCI3240**
  - Become more effective programmers
    - Able to find and eliminate bugs efficiently
    - Able to understand and tune for program performance
  - Prepare for later "systems" classes in CS
    - Compilers, Operating Systems, Networks, etc.

Great Reality #1:
Ints are not Integers, Floats are not Reals

- **Example 1: Is $x^2 \geq 0$?**
  - Float's:
    - Yes!   (always true for Float)
  - Int's:
    - 40000 * 40000 → 1600000000
    - 50000 * 50000 → ??

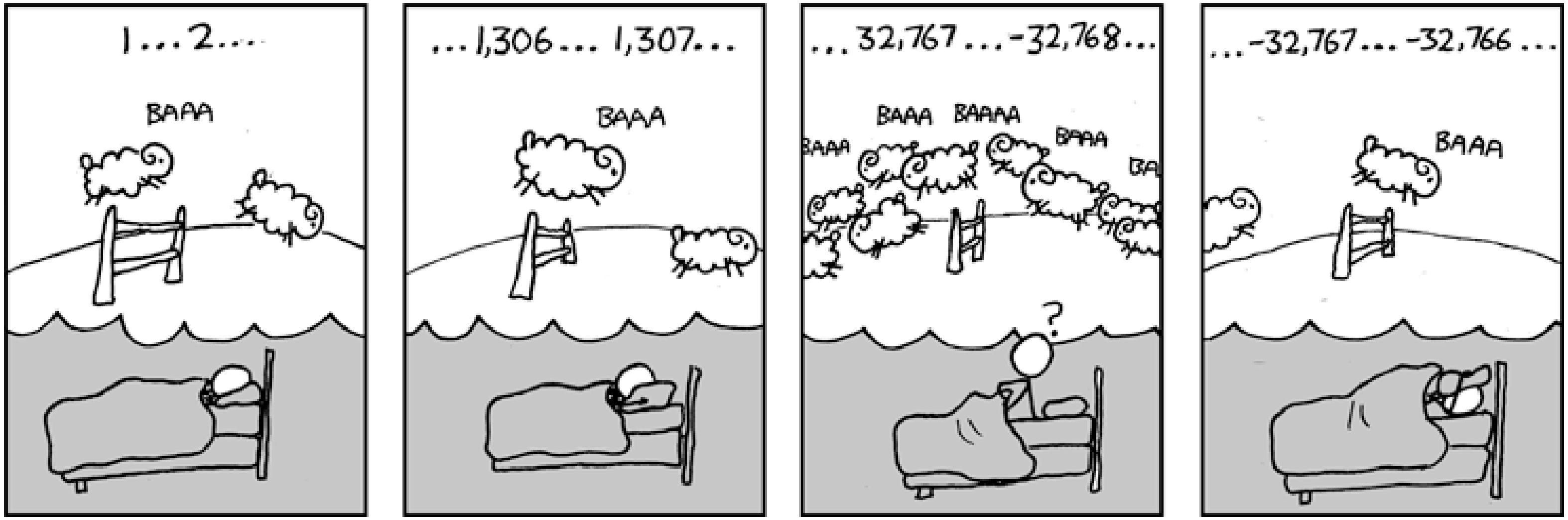


```
(gdb) print 40000 * 40000
$1 = 1600000000
(gdb) print 50000 * 50000
$2 = -1794967296 ???
(gdb) 
```

Is addition Associative?

- **Example 2: Is (x + y) + z = x + (y + z)?**
  - Unsigned & Signed Int's: Yes!
  - Float's:
    - (1e20 + -1e20) + 3.14 --> 3.14
    - 1e20 + (-1e20 + 3.14) --> ??

Great Reality #1: Ints are not Integers, Floats are not Reals



Computer Arithmetic

- **Does not generate random values**
  - Arithmetic operations have important mathematical properties

- **Cannot assume all “usual” mathematical properties**
  - Due to the finiteness of representations
  - Integer operations satisfy “ring” properties
    - Commutativity, associativity, distributivity
  - Floating point operations satisfy “ordering” properties
    - Monotonicity, values of signs

Great Reality #2: More Assembly

- **Chances are, you'll never write programs in assembly**
  - Compilers are much better & more patient than you are
- **But: Understanding assembly is key to the machine-level execution model**
  - Behavior of programs in the presence of bugs
    - High-level language models break down
  - Tuning program performance
    - Understand optimizations done / not done by the compiler
    - Understanding sources of program inefficiency
  - Implementing system software
    - Compiler has machine code as the target
    - Operating systems must manage process state
  - Creating/fighting malware
    - x86 assembly is the language of choice!

Great Reality #3: Memory Matters
Random Access Memory Is an Unphysical Abstraction

- **Memory is not unbounded**
  - It must be allocated and managed
  - Many applications are memory dominated

- **Memory referencing bugs especially pernicious**
  - Effects are distant in both time and space

- **Memory performance is not uniform**
  - Cache and virtual memory effects can greatly affect program performance
  - Adapting program to characteristics of memory system can lead to major speed improvements

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;
}
```

Console

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

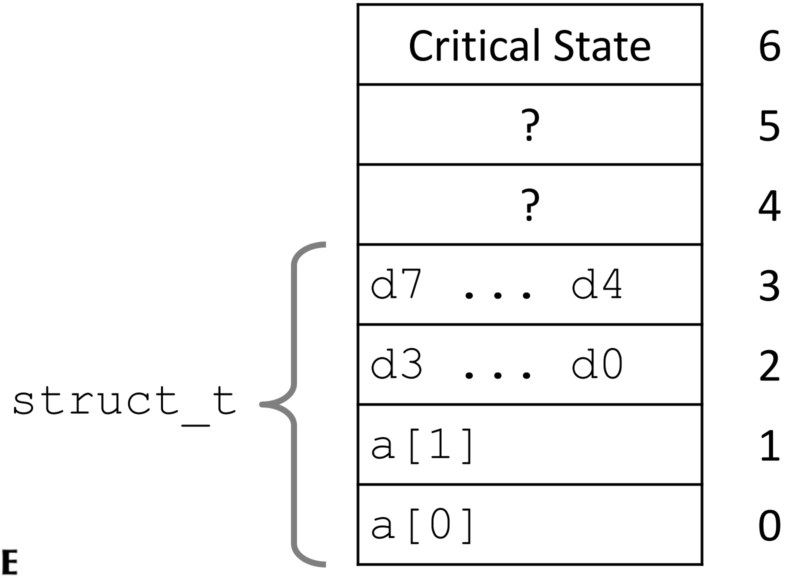
Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

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)   ↝ 2.00000061035156
fun(4)   ↝ 3.14
fun(6)   ↝ Segmentation fault
```

Explanation:



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

C/C++ will not do bound check for you. It is your responsibility as a programmer.

Location accessed by `fun(i)`

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Memory Referencing Errors

- **C and C++ do not provide any memory protection**
  - Out-of-bounds array references
  - Invalid pointer values
  - Abuses of malloc/free

- **Can lead to nasty bugs**
  - Whether or not bug has any effect depends on system and compiler
  - Action at a distance
    - Corrupted object logically unrelated to one being accessed
    - 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)

Great Reality #4: There's more to performance than asymptotic complexity

- **Constant factors matter too!**

- **And even exact op count does not predict performance**
  - Easily see 10:1 performance range depending on how code written
  - Must optimize at multiple levels: algorithm, data representations, procedures, and loops

- **Must understand system to optimize performance**
  - How programs are compiled and executed
  - How to measure program performance and identify bottlenecks
  - How to improve performance without destroying code modularity and generality

Memory System Performance Example

```c
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];
}
```

```c
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];
}
```

4.3ms

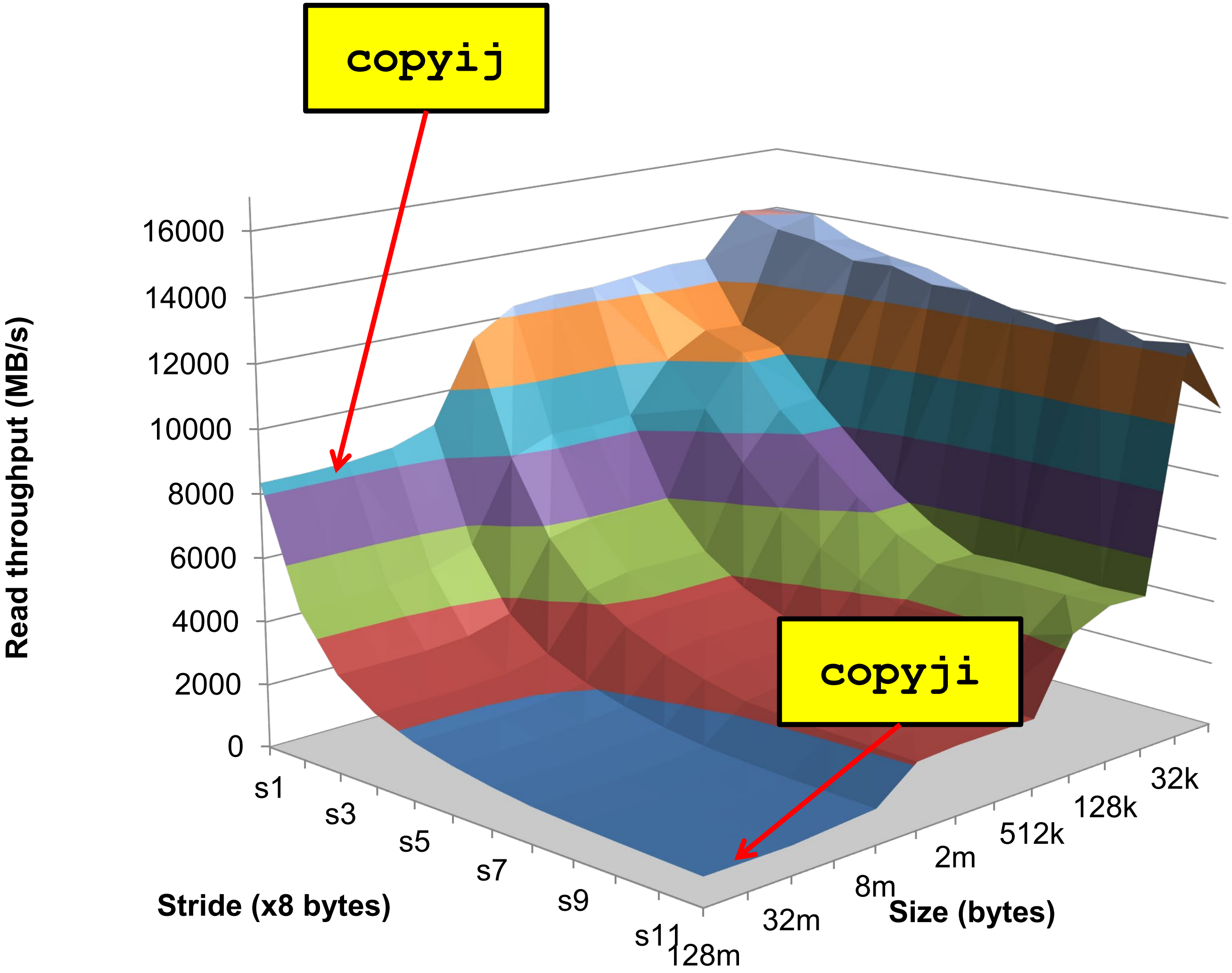
81.8ms

2.0 GHz Intel Core i7 Haswell

- Hierarchical memory organization
- Performance depends on access patterns
  - Including how step through multi-dimensional array

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

The Performance Differs



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Great Reality #5:
Computers do more than execute programs

- **They need to get data in and out**
  - I/O system critical to program reliability and performance

- **They communicate with each other over networks**
  - Many system-level issues arise in presence of network
    - Concurrent operations by autonomous processes
    - Coping with unreliable media
    - Cross platform compatibility
    - Complex performance issues

Course Perspective

- CSCI3130 Assembly and Computer Organization
  - Builder-Centric
  - We built a Relatively Simple Computer (RSC) in Logisim
- CSCI3240 Introduction to Computer Systems
  - Programmer-Centric
  - Purpose is to show that by knowing more about the underlying system, one can be more effective as a programmer
  - Enable you to
    - Write programs that are more reliable and efficient
    - Incorporate features that require hooks into OS
      - E.g., concurrency, signal handlers

Textbooks

- Randal E. Bryant and David R. O'Hallaron,
  - *Computer Systems: A Programmer's Perspective*, **Third Edition** (CS:APP3e), Pearson, 2016
  - http://csapp.cs.cmu.edu
  - This book really matters for the course!
    - How to solve projects
    - Practice problems typical of exam problems

- Brian Kernighan and Dennis Ritchie,
  - *The C Programming Language*, Second Edition, Prentice Hall, 1988
  - Still the best book about C, from the originators

Course Components

- Lectures
  - Higher level concepts

- Quizzes
  - 7 Quizzes
  - At max, three attempts are allowed.
  - Your final score will be based on your last attempt.
  - Total weight: 30%

- Projects
  - 4 Individual projects
  - Total weight: 40%

- Exams
  - 2 midterm exams.
  - Test your understanding of concepts & mathematical principles
  - Total weight: 30%

Class Discord

- https://discord.gg/tjQSEbVafu

Enjoy the process

THE TWO STATES OF EVERY PROGRAMMER
I AM A GOD.
I HAVE NO IDEA WHAT I'M DOING.