



# Course Overview

These slides adapted from materials provided by the textbook

# Overview

- Course theme
- Five realities

# Course Theme:

## Abstraction Is Good But Don't Forget Reality

- **Most CS and CE courses emphasize abstraction**
  - Abstract data types
  - Asymptotic analysis
- **These abstractions have limits**
  - Especially in the presence of bugs
  - Need to understand details of underlying implementations
- **Useful outcomes**
  - 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 & ECE
    - Compilers, Operating Systems, Networks, Computer Architecture, Embedded Systems, Storage Systems, etc.

# Great Reality #1:

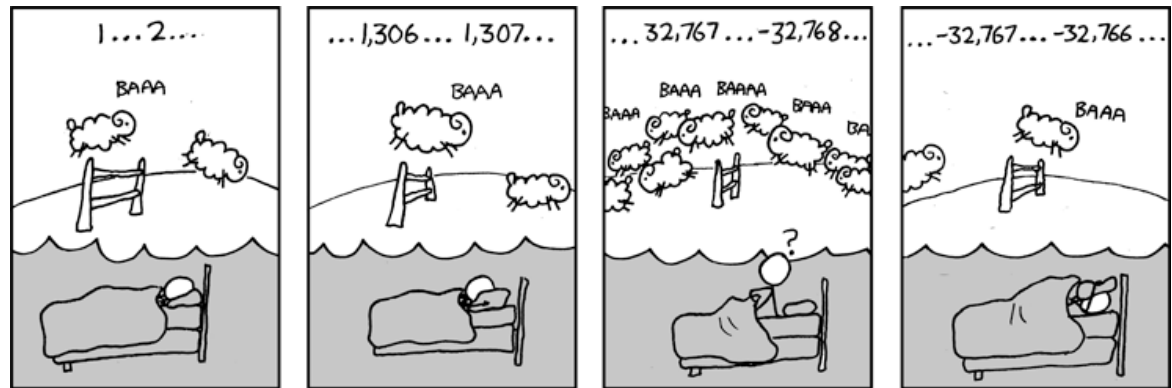
`int` is not Integers, `float` is not Reals

## ■ Example 1: Is $x^2 \geq 0$ ?

■ Float's: Yes!

■ Int's:

- $40000 * 40000 \Rightarrow 1600000000$
- $50000 * 50000 \Rightarrow ??$



## ■ Example 2: Is $(x + y) + z = x + (y + z)$ ?

■ Unsigned & Signed Int's: Yes!

■ Float's:

- $(1e20 + -1e20) + 3.14 \Rightarrow 3.14$
- $1e20 + (-1e20 + 3.14) \Rightarrow ??$

# Computer Arithmetic

- **Does not generate random values**

- Arithmetic operations have important mathematical properties

- **Cannot assume all “usual” mathematical properties**

- Due to finiteness of representations
  - Integer operations satisfy “ring” properties
    - Commutativity, associativity, distributivity
  - Floating point operations satisfy “ordering” properties
    - Monotonicity, values of signs

- **Observation**

- Need to understand which abstractions apply in which contexts
  - Important issues for compiler writers and serious application programmers

# Great Reality #2:

## You've Got to Know 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 machine-level execution model**
  - Behavior of programs in 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 target
    - Operating systems must manage process state
  - Creating / fighting malware

# 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;  
}
```

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

- Result is system specific

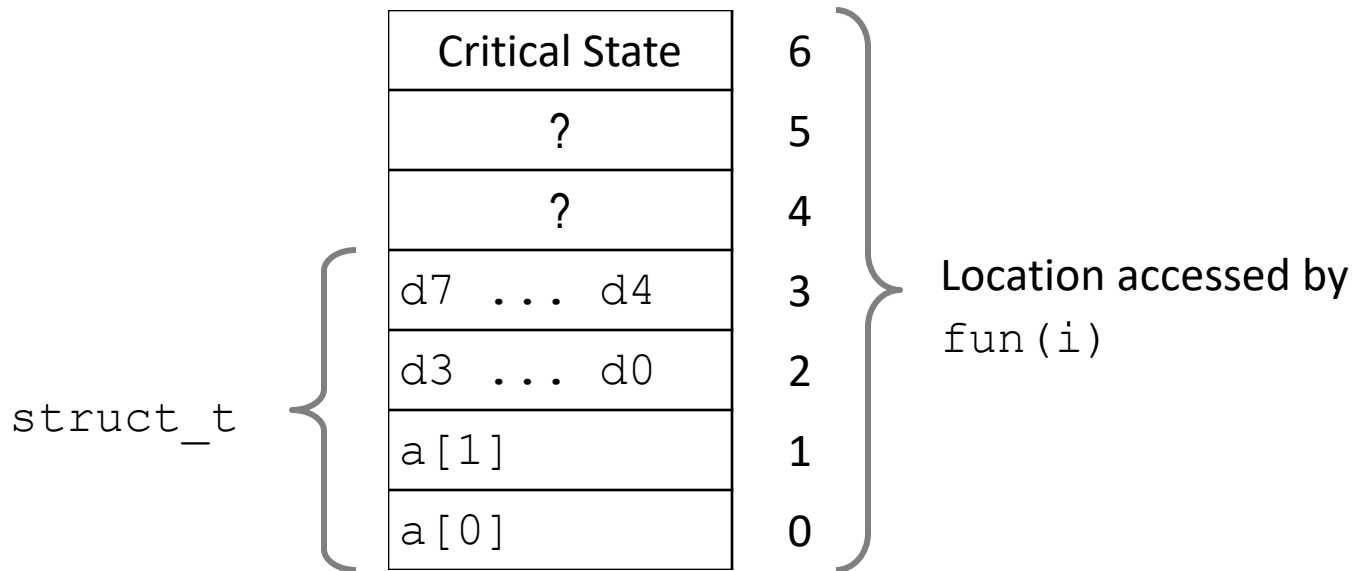


# Memory Referencing Bug Example

```
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:



# 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 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

```
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];
}
```

4.3ms

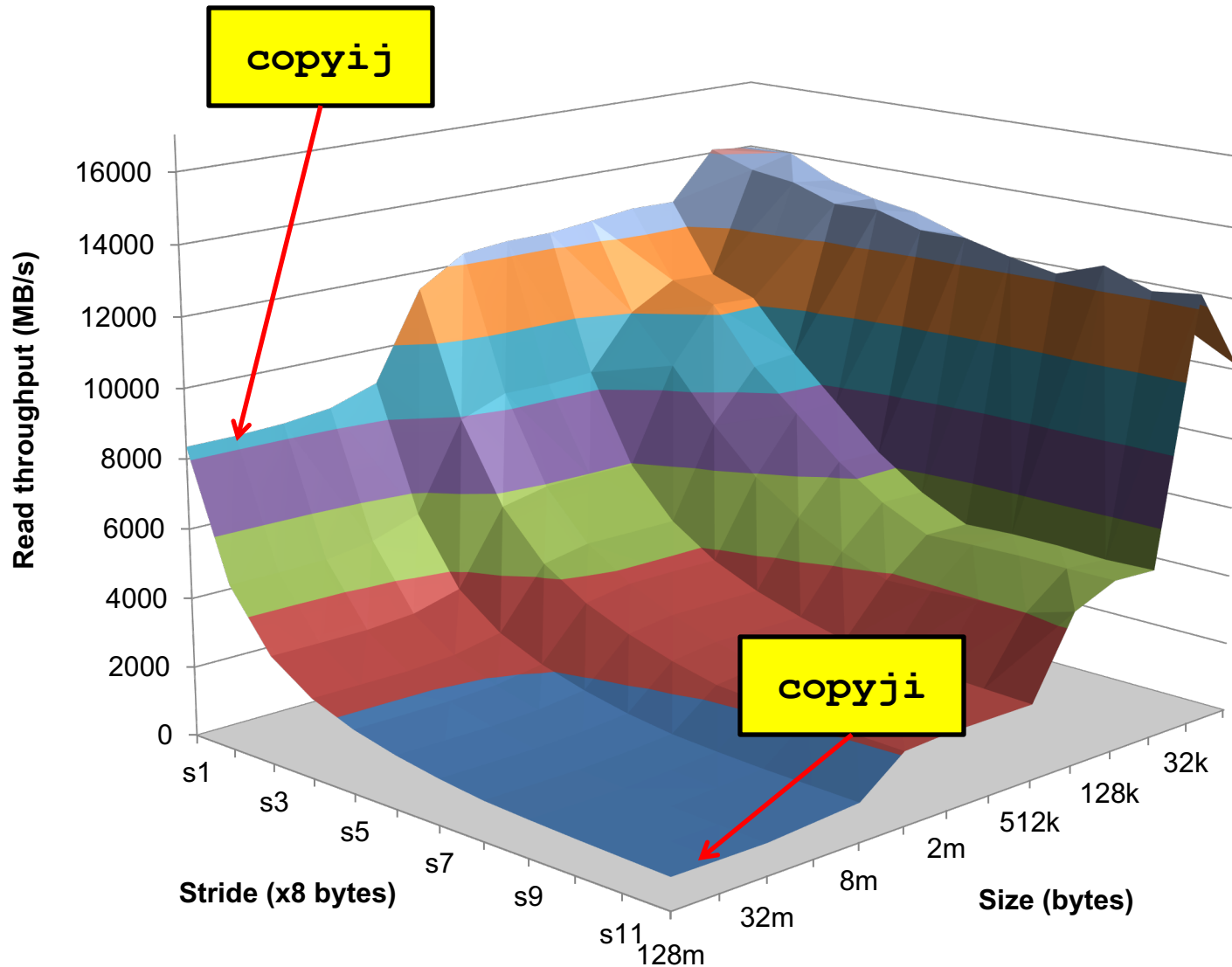
2.0 GHz Intel Core i7 Haswell

```
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];
}
```

81.8ms

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

# Why The Performance Differs



# 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

**Welcome  
and  
Enjoy!**