

# Course Overview

Computer Systems  
1<sup>st</sup> Lecture, Sep. 6, 2021

**Lecture:**

Troels Henriksen

**Based on slides by:**

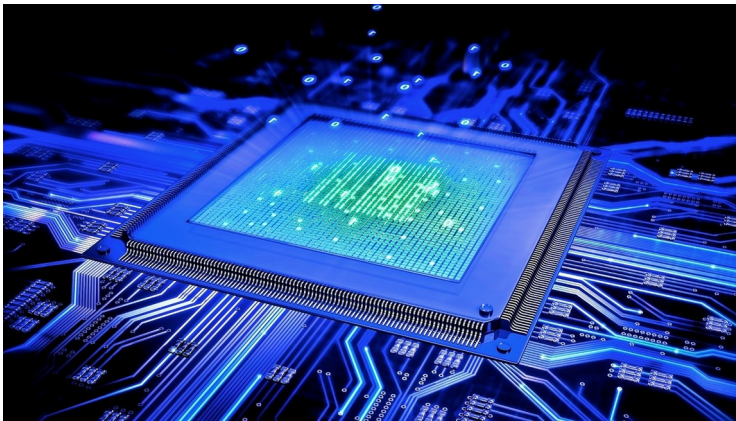
Randal E. Bryant and David R. O'Hallaron

# Overview

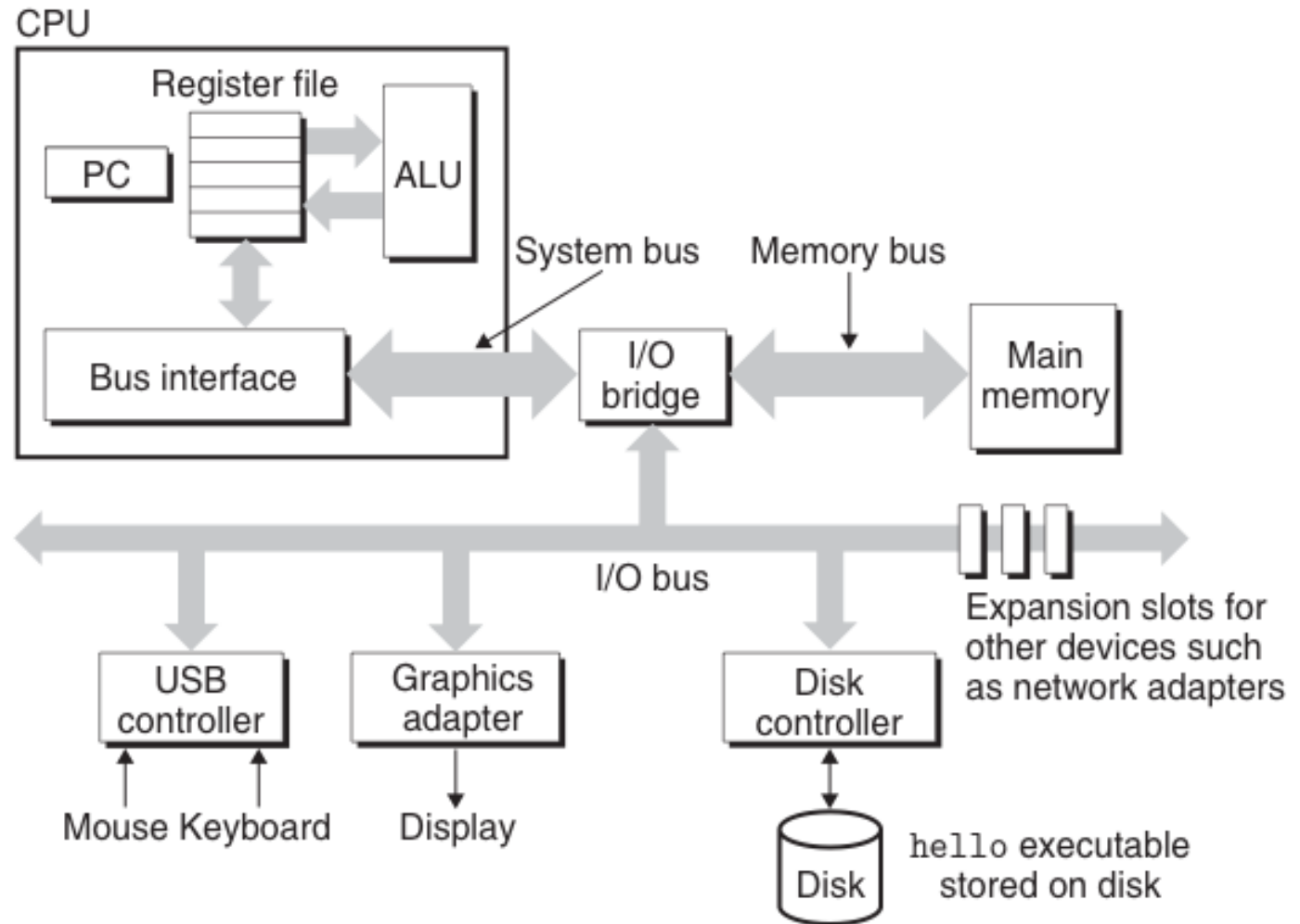
- Course introduction
- Overall theme
- What is part of the course

# What is a computer system for you?

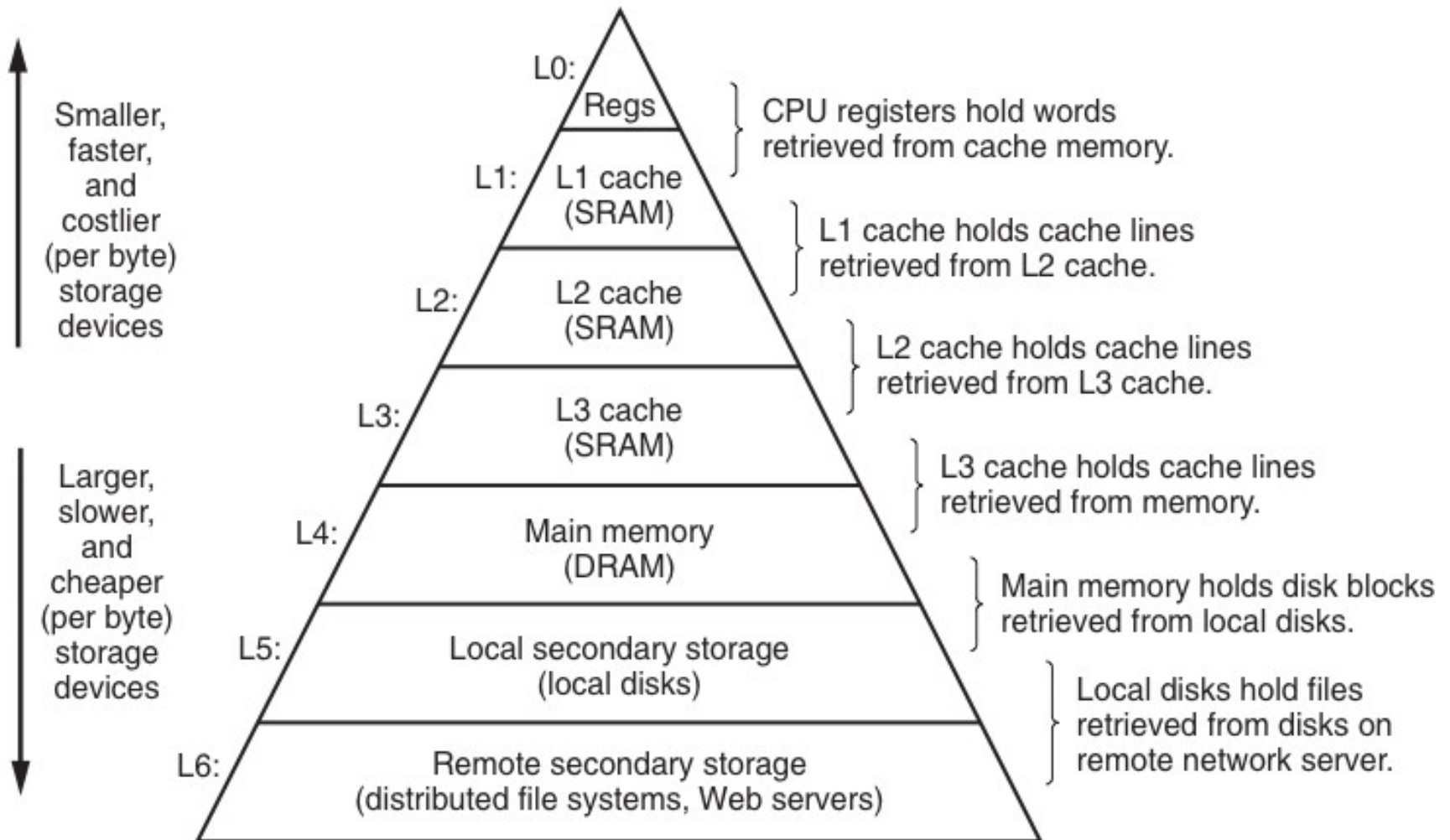
- 5 minutes! What to you is a computer system? What does it effect?
  - For inspiration (and more):
  - CPU, logic gate, transistor, RAM, memory hierarchy, virtual memory, process, thread, network, disk, I/O, http, TCP/IP, RSA, bus, cache, WiFi, switch, internet, synchronization, pipeli



# Simple Computer System (hardware)

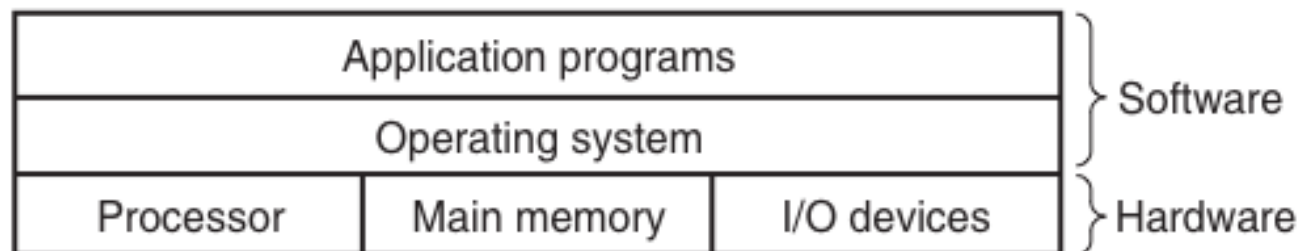


# The Memory Hierarchy

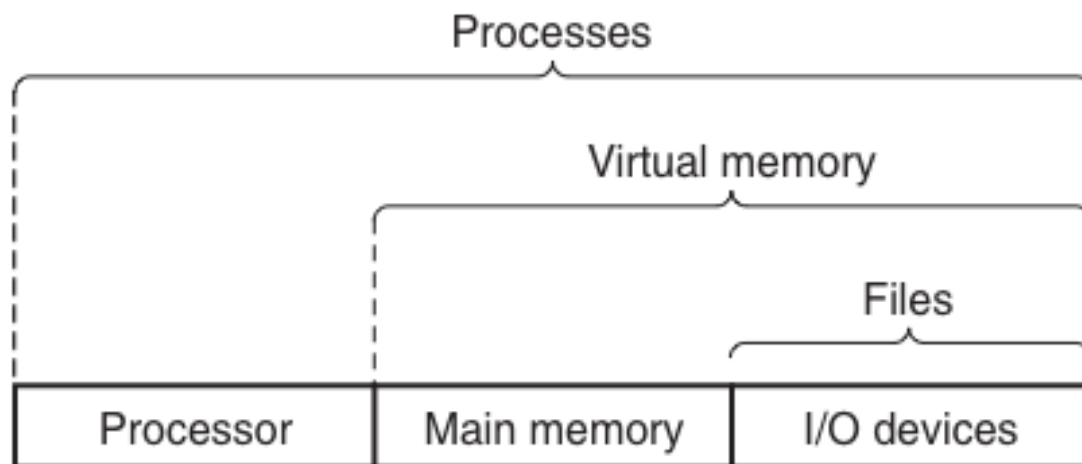


# The Memory Hierarchy

- Layered view of computer system

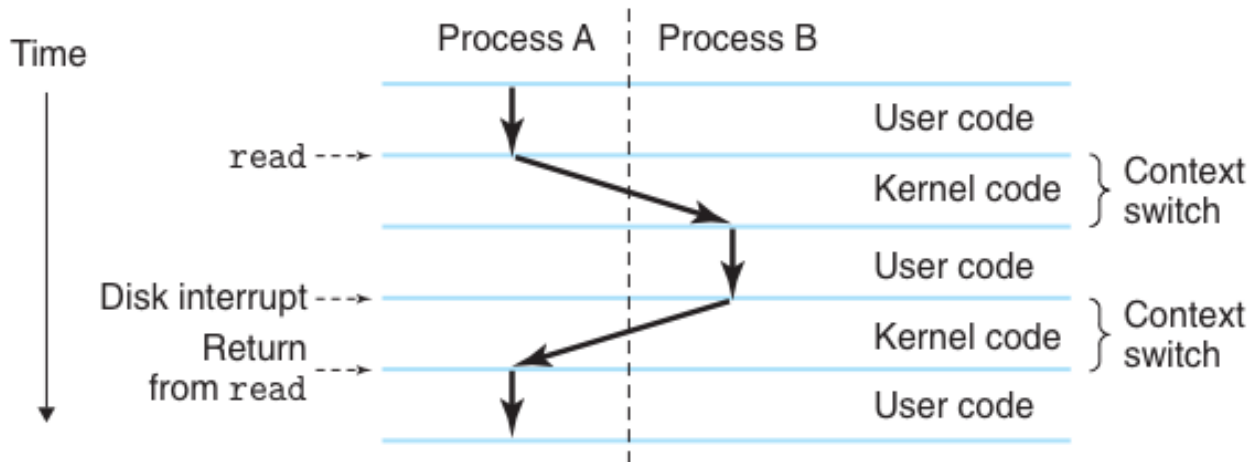


- Abstractions provided by OS



# The Processes

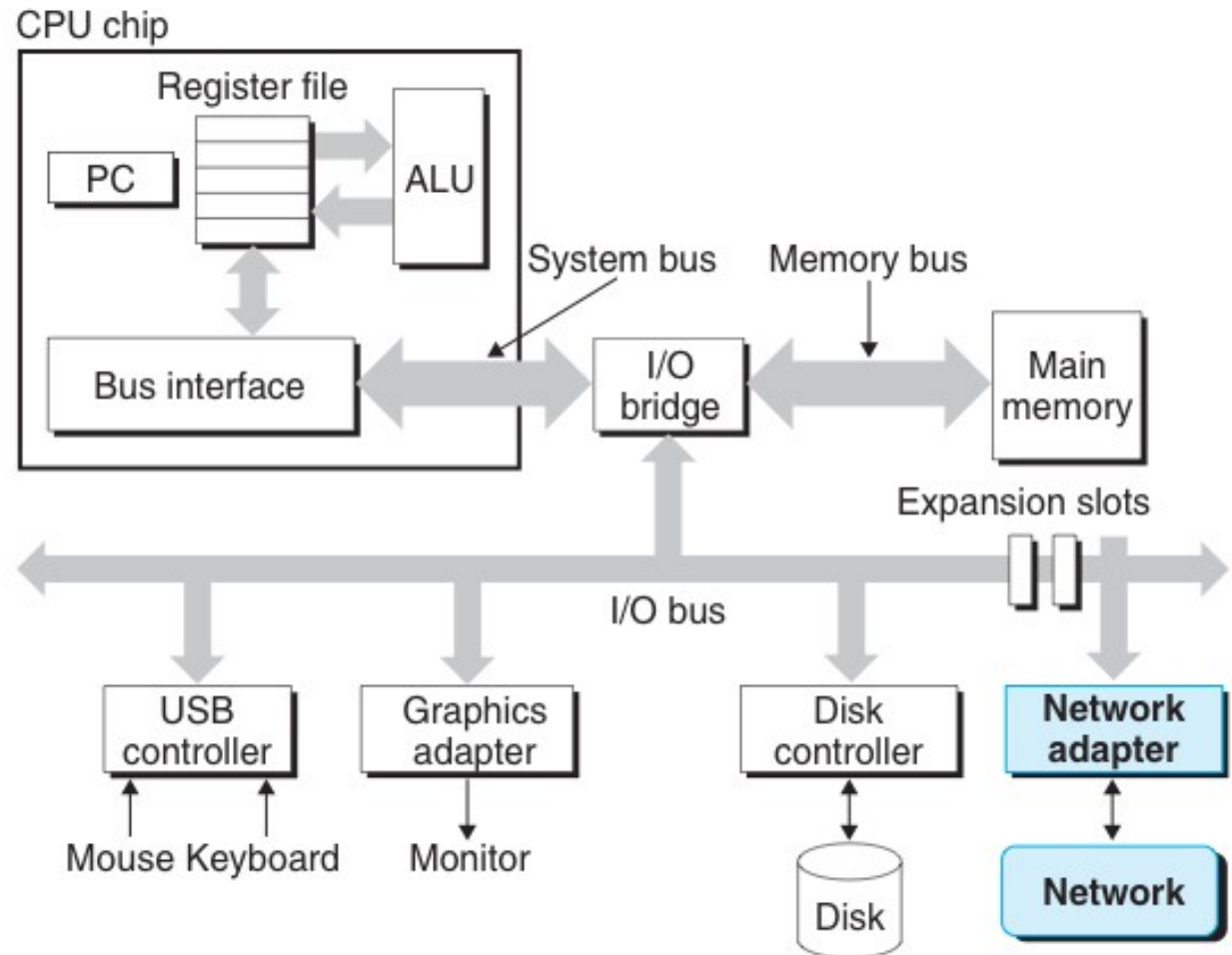
## ■ Multi-process system



- Multi-process on sequential processor
- Multi-process on multi-core processor

# The Network – system side

- How to interface with computer system
- Learn the different abstractions and how they are implemented
- Understand how security can be improved

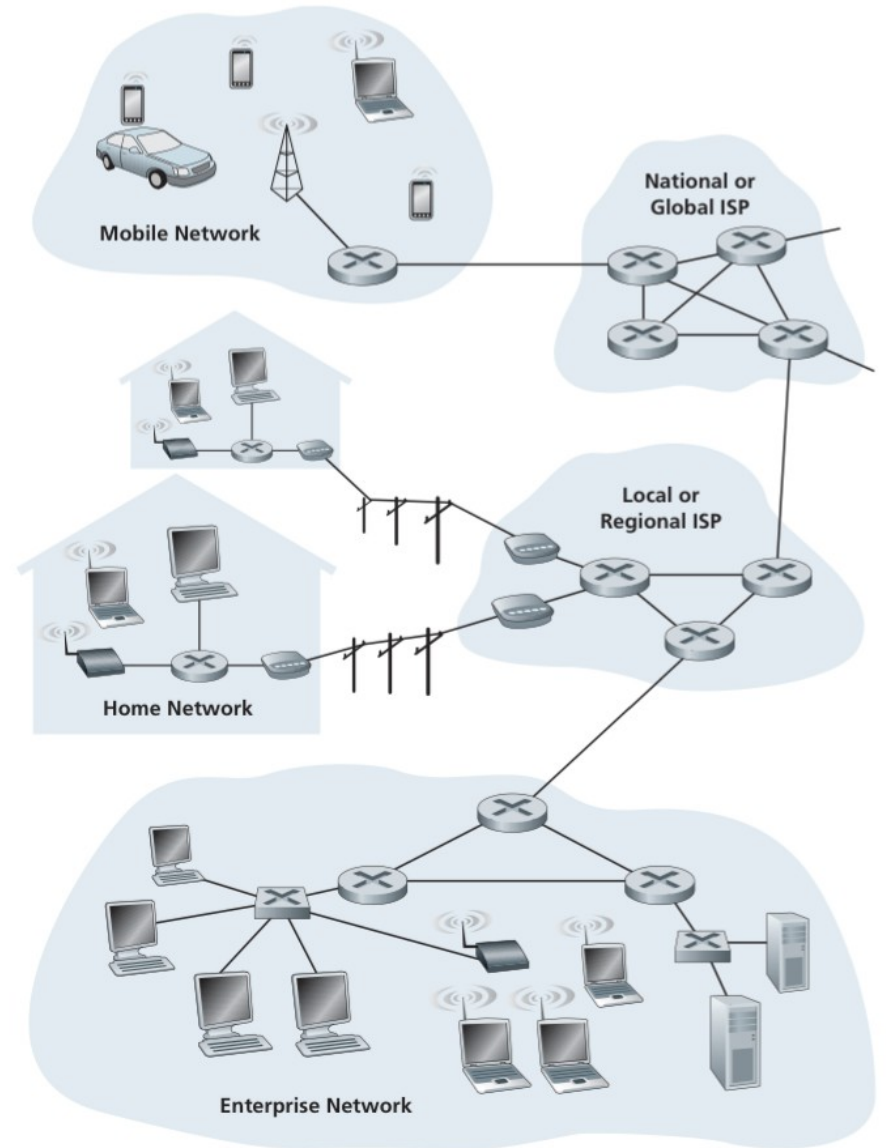




# The Network – network side



a. Five-layer Internet protocol stack



Key:



# Course Theme:

## Abstraction Is Good But Don't Forget Reality

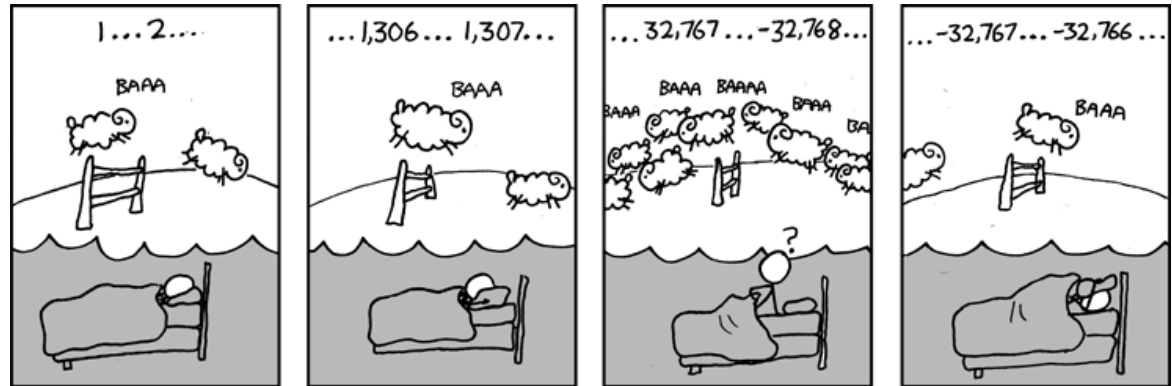
- **Most CS courses emphasize high-level abstraction**
  - Abstract data types
  - Asymptotic analysis
- **These abstractions have limits**
  - Especially in the presence of bugs
  - Need to understand details of underlying implementations
- **Useful outcomes from taking CompSys**
  - Knowledge about concepts of (low-level abstractions)
    - Machine Architecture, Memory hierarchy, Operating Systems, Computer Networks, and Encryption.
  - Become more effective programmers
    - Able to find and eliminate bugs efficiently
    - Able to understand and tune for program performance

# Great Reality #1:

## Ints are not Integers, Floats are not Reals

### ■ Example 1: Is $x^2 \geq 0$ ?

- Floats: Yes!



- Ints:

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

### ■ 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
  - Well, *almost*...

## ■ 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 and 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
    - 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;  
}
```

```
fun(0)    → 3.14  
fun(1)    → 3.14  
fun(2)    → 3.13999998664856  
fun(3)    → 2.000000061035156  
fun(4)    → 3.14  
fun(6)    → Segmentation fault
```

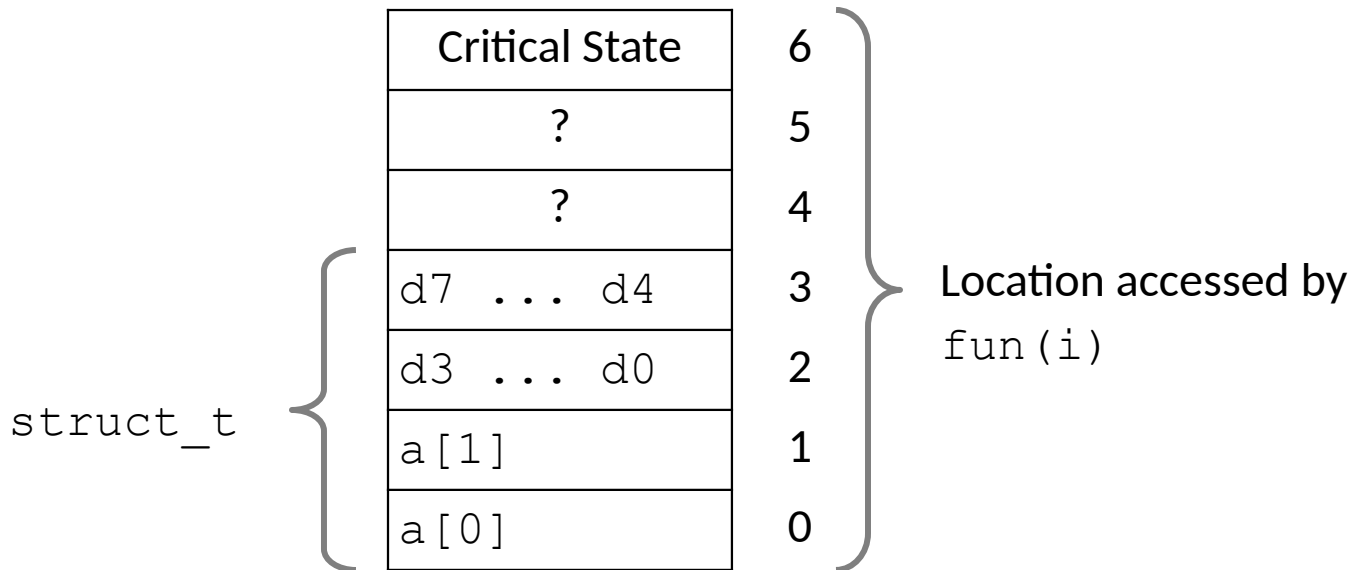
- Result is system specific

# Memory Referencing Bug Example

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

fun(0) → 3.14  
fun(1) → 3.14  
fun(2) → 3.13999998664856  
fun(3) → 2.000000061035156  
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 F#, Python, Haskell, Futhark, ...
  - 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

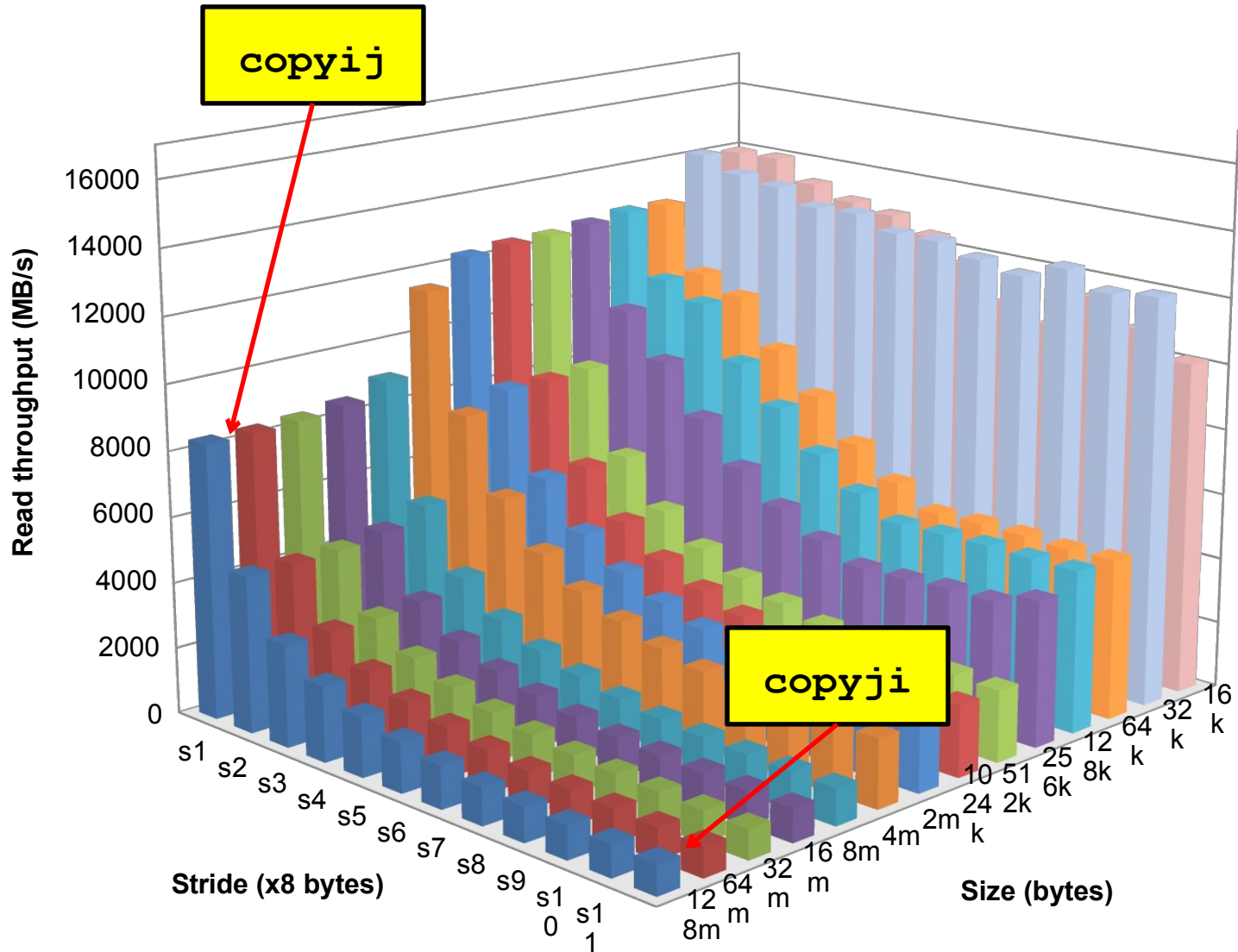
```
void copyij(int src[2048][2048],
            int dst[2048][2048])
{
    int i,j;
    for (i = 0; i < 2048; i++)
        for (j = 0; j < 2048; j++)
            dst[i][j] = src[i][j];
}
```

```
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 2.0 GHz Intel Core i7 Haswell 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

# Course Perspective

## ■ Course is 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
- Cover material in this course that you won't see elsewhere
- Not just a course for dedicated hackers
  - **We bring out the hidden hacker in everyone!**

Welcome  
and Enjoy!