Computer Systems

Computer Systems
Pitch lecture, June 2 2023

Lecture:

Michael Kirkedal Thomsen

Other on the course

Finn Schiermer

David Marchant

With some slides by:

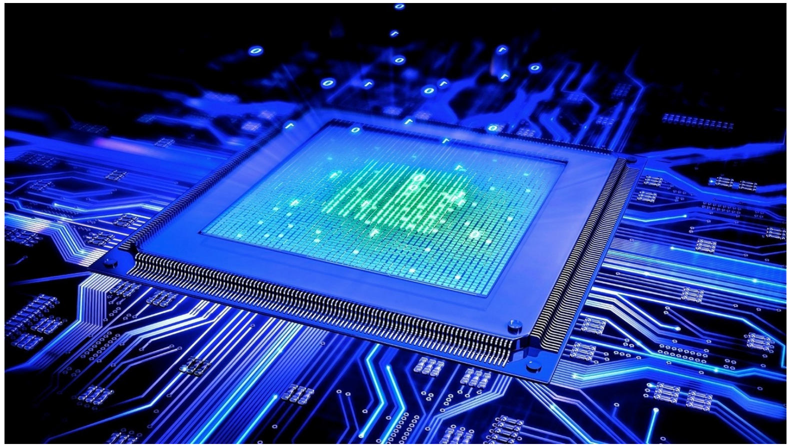
Randal E. Bryant and David R. O'Hallaron

CompSys

- På første år har I set programmer som abstrakte modeller
  - Abstrakte datatyper
  - Asymptotisk analyse
- Programmer skal dog afvikles på en fysisk maskine
- CompSys: Vi skal forstå hvad der virkeligt foregår i en computer
  - Forstå low-level abstraktioner
    - Machine Architecture, Memory hierarchy, Operating Systems, Computer Networks, and Encryption.
  - Blive bedre til at skrive gode programmer
    - Kan finde og fjerne bugs
    - Kan forstå og forbedre performance

What is a computer system?

- **What to you is a computer system? What does it effect?**
  - 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, pipeline,..



Simple example

- What happens here?

- Indexes are switched

- Data representation
- Memory hierarchy

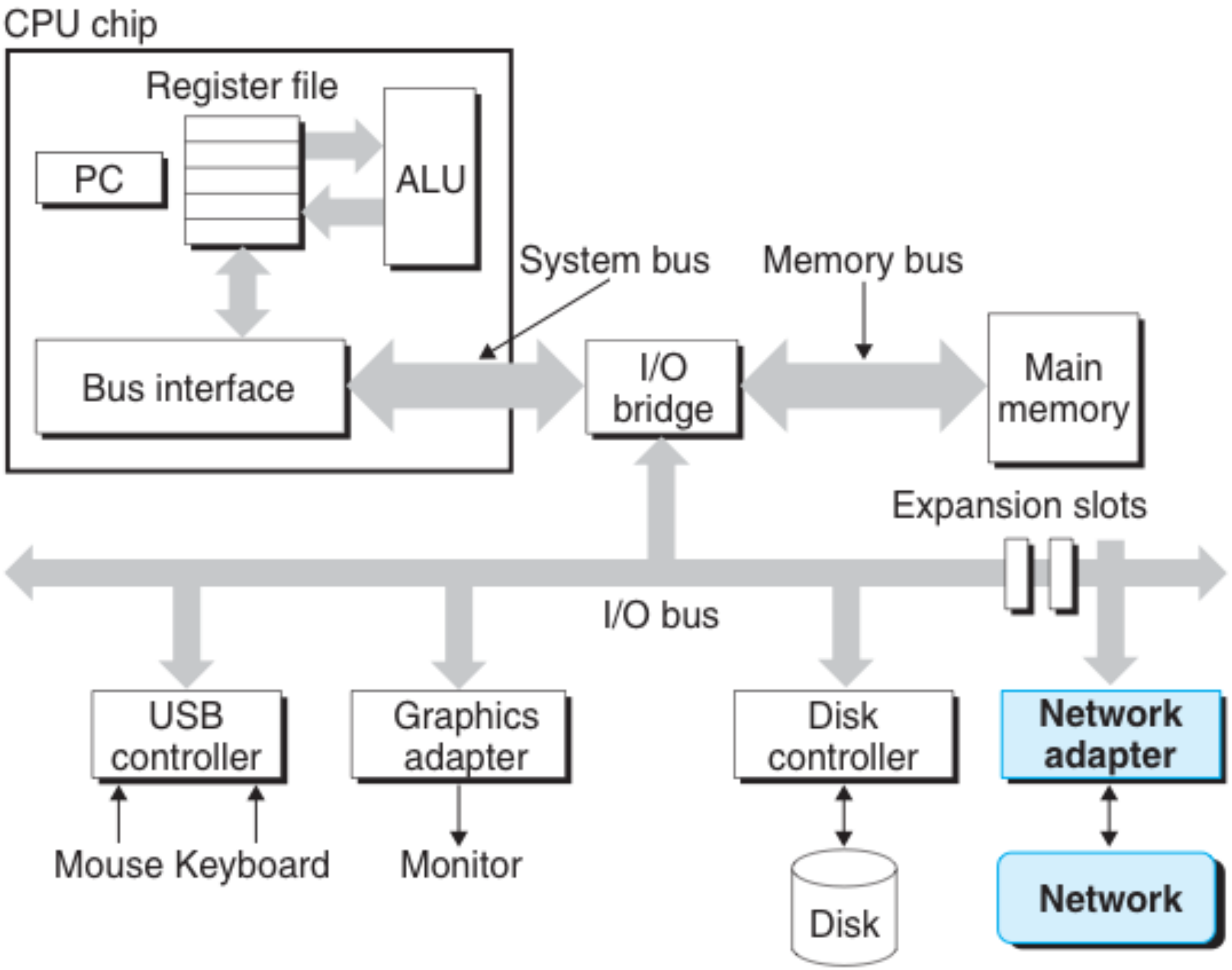
- Hidden abstractions – The real world is messy

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

Computer system

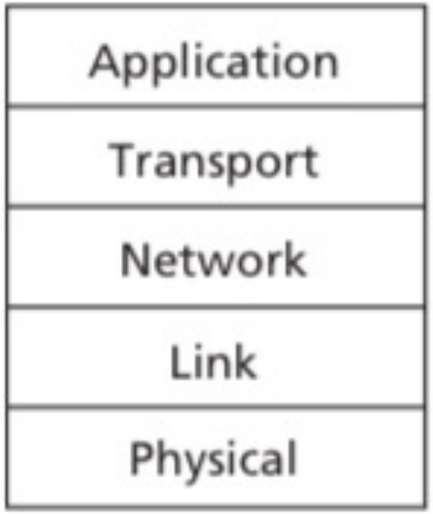
- How to interface with computer system
- Learn the differen abstractions and how they are implemented
- Understand how security can be improved



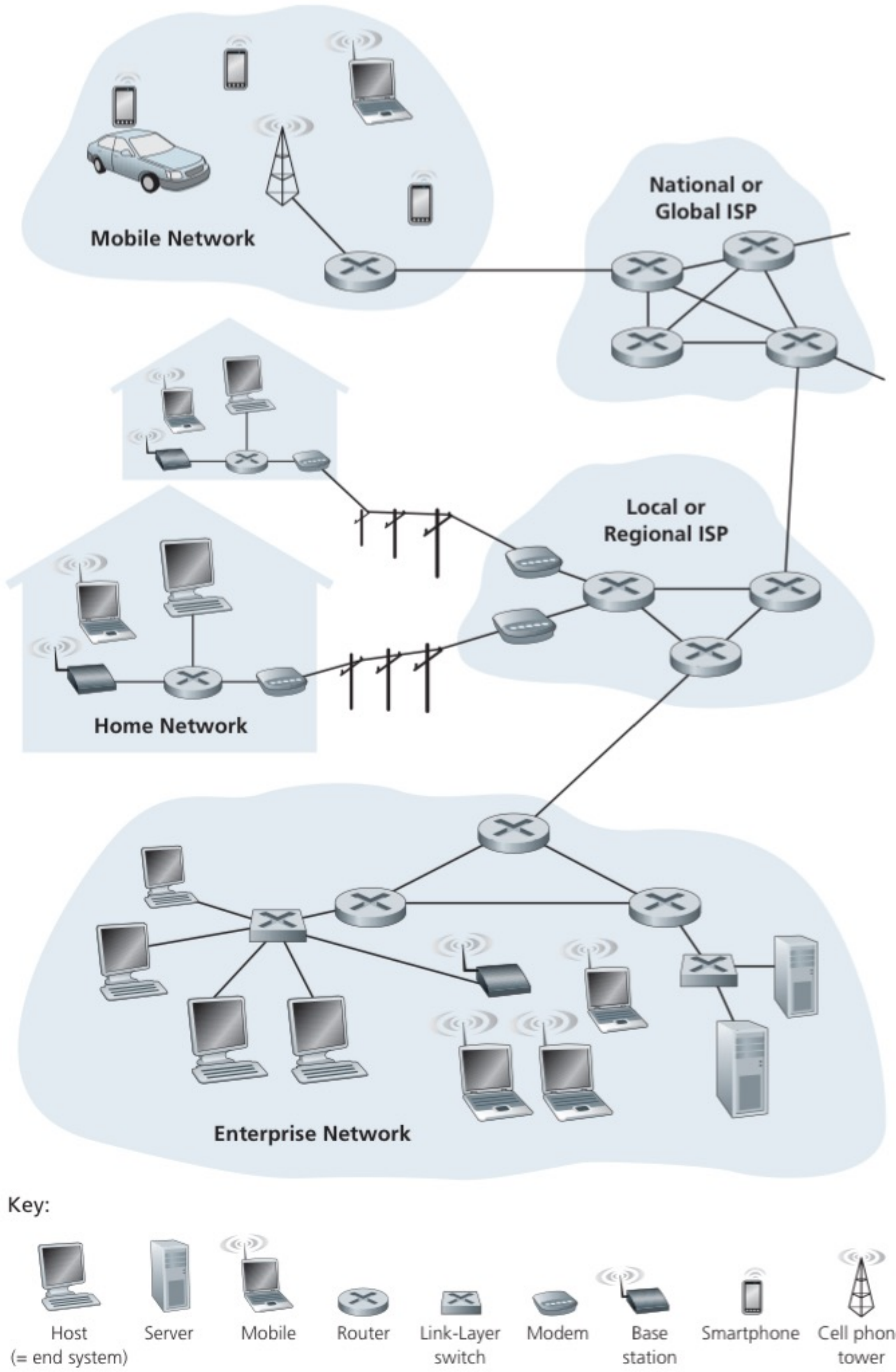
Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

The Network

-

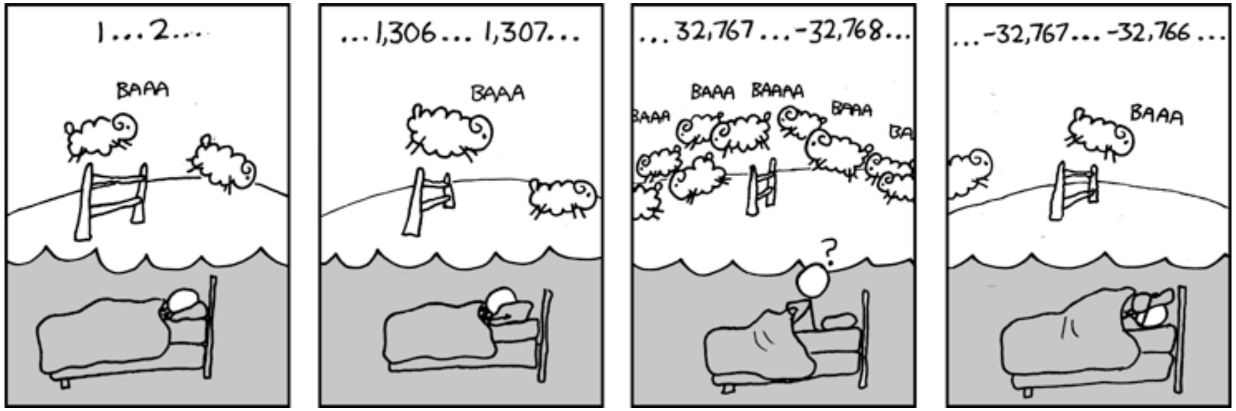


a. Five-layer Internet protocol stack



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

- **Example 1: Is $x^2 \geq 0$?**
  - Float's: Yes!



  - Int's:
    - 40000 * 40000 ➞ 1600000000
    - 50000 * 50000 ➞ ??
- **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) --> ??

Source: xkcd.com/571

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
    - x86 assembly is the language of choice!

Great Reality #3: Memory Matters

Random Access Memory Is an Unphysical Abstraction

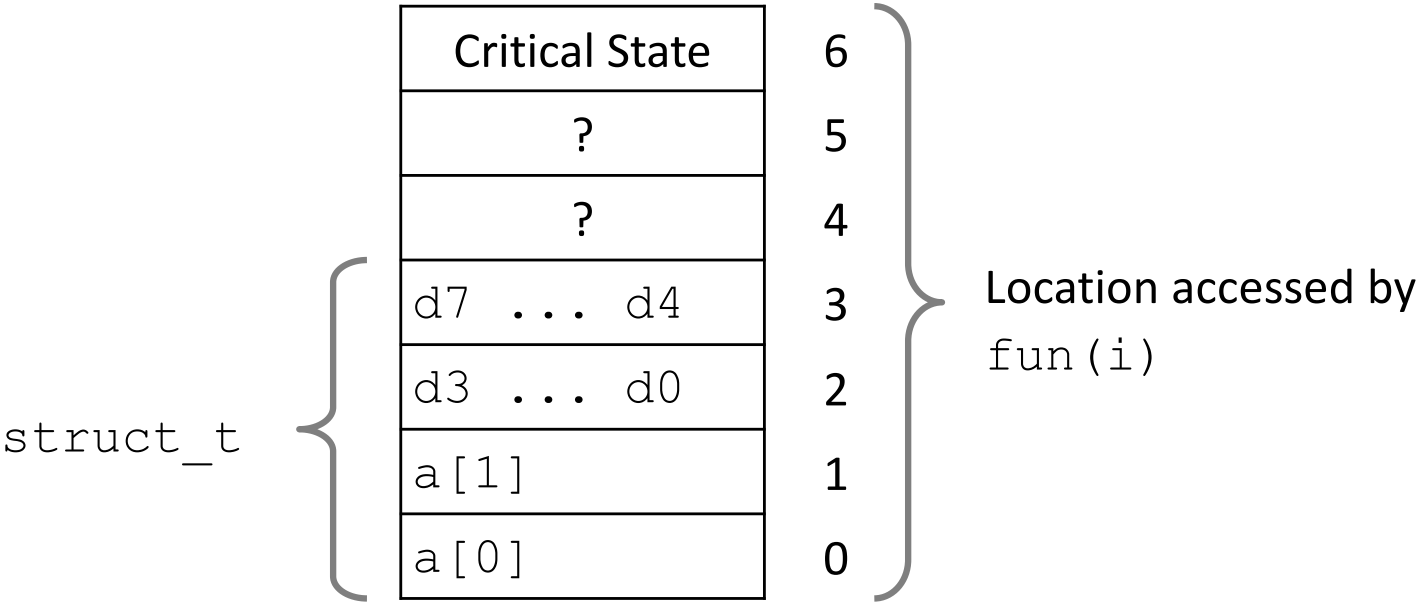
- **Memory is not unbounded**
  - It must be allocated and managed
  - Many applications are memory dominated
- **Memory referencing bugs especially pernicious**
  - Effects are distant in both time and space
- **Memory performance is not uniform**
  - Cache and virtual memory effects can greatly affect 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;
```

```
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

struct_t

Location accessed by `fun(i)`

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'
- Information on Absalon Course Page, and
- https://github.com/diku-compSys/compSys-e2023-pub

Preparation

- Programming is a craft
  - It is important that you practice it
  - Keeping your C# up-to-date

- **DIKU Summer of Programming 2021**
  - https://github.com/diku-summer-programming/DSoP21
- **Edabit**
  - https://edabit.com/challenges/csharp
- **Programmr**
  - http://www.programmr.com/exercises?lang=csharp