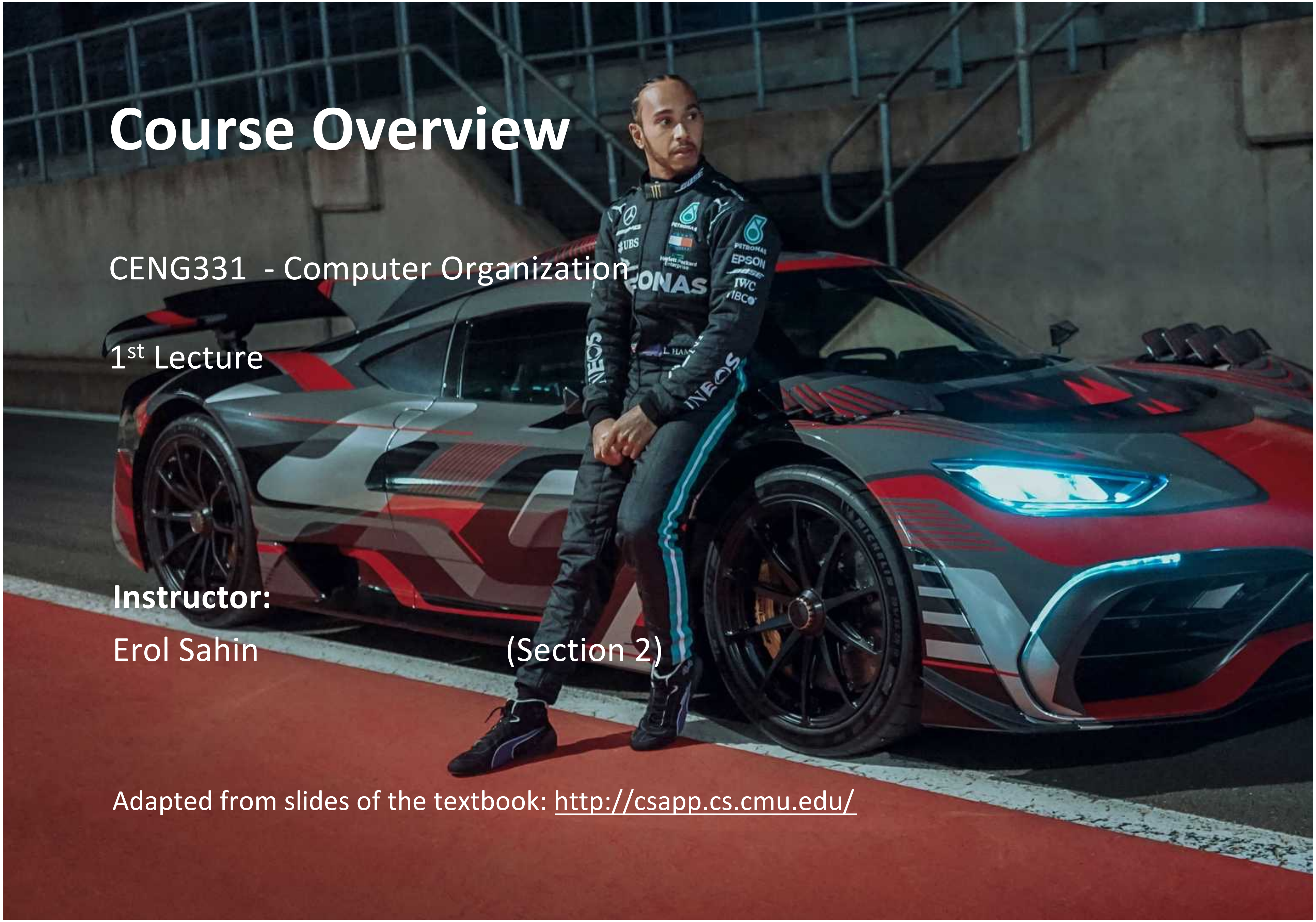


Course Overview

CENG331 - Computer Organization

1st Lecture

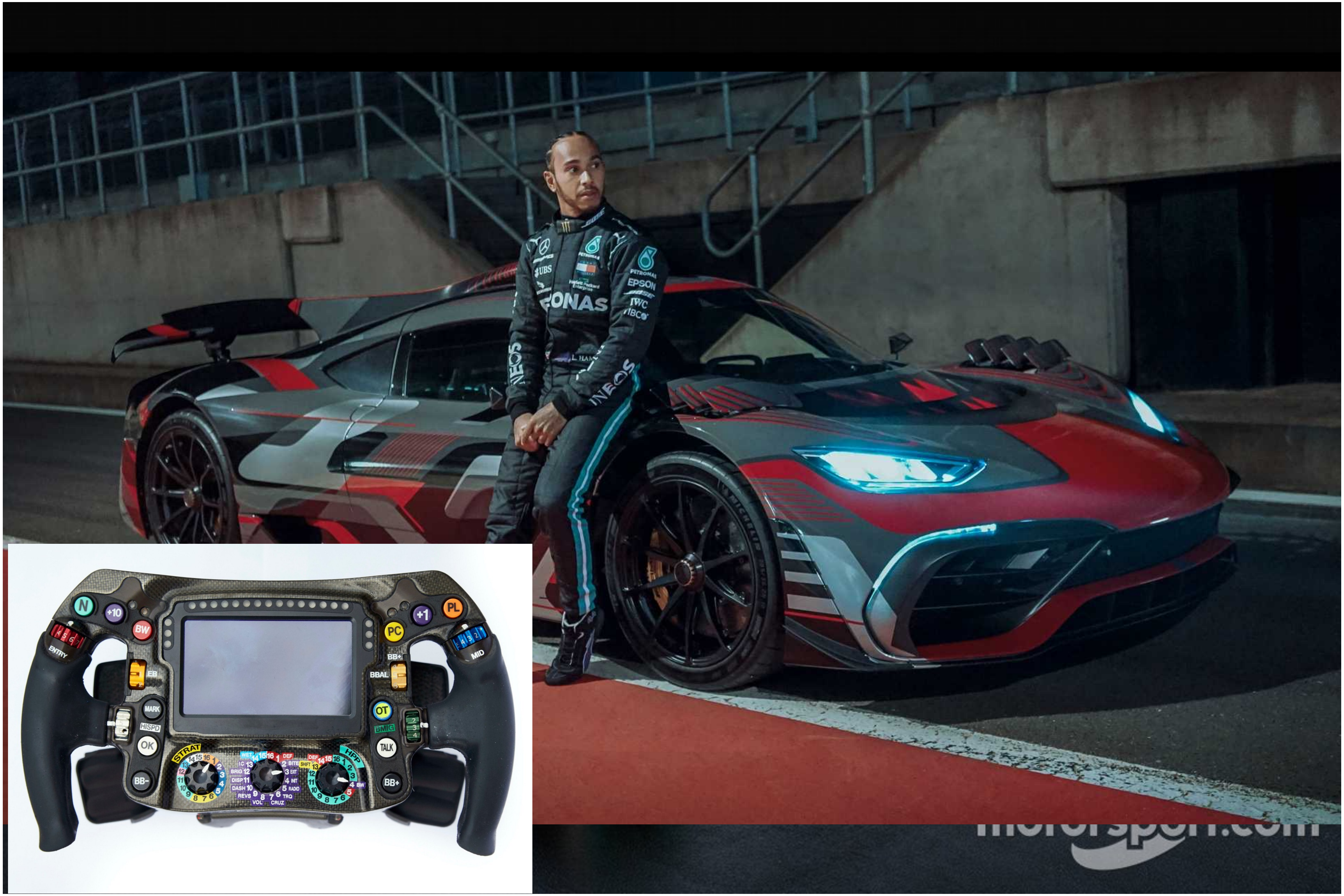
Instructor:

Erol Sahin (Section 2)

Adapted from slides of the textbook: http://csapp.cs.cmu.edu/


RADIO
MEDIA
TEL
MY CAR


UBS
EPSON
IWC
INEOS


N
+10
BW
PC
+1
PL
ENTRY
MID
EB
BBAL
BB+
MARK
HISPD
OK
OT
BMIG
TALK
STRAT
HPP
BB-
BB+


N
+10
BW
PC
+1
PL
ENTRY
MID
EB
BBAL
MARK
HISPD
OK
OT
TALK
BB-
BB+
STRAT


PSU
CD-ROM
RAM
CPU
VIDEO CARD
HARD DRIVE
FAN
MOTHERBOARD

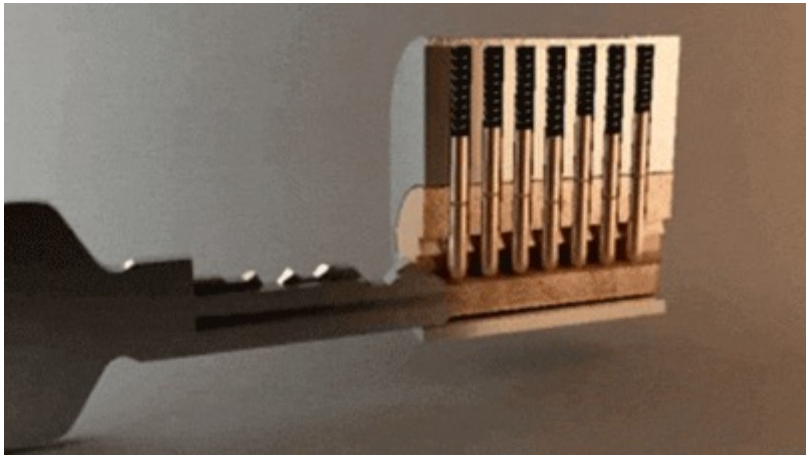
From Noob to Pro in just 16 weeks!

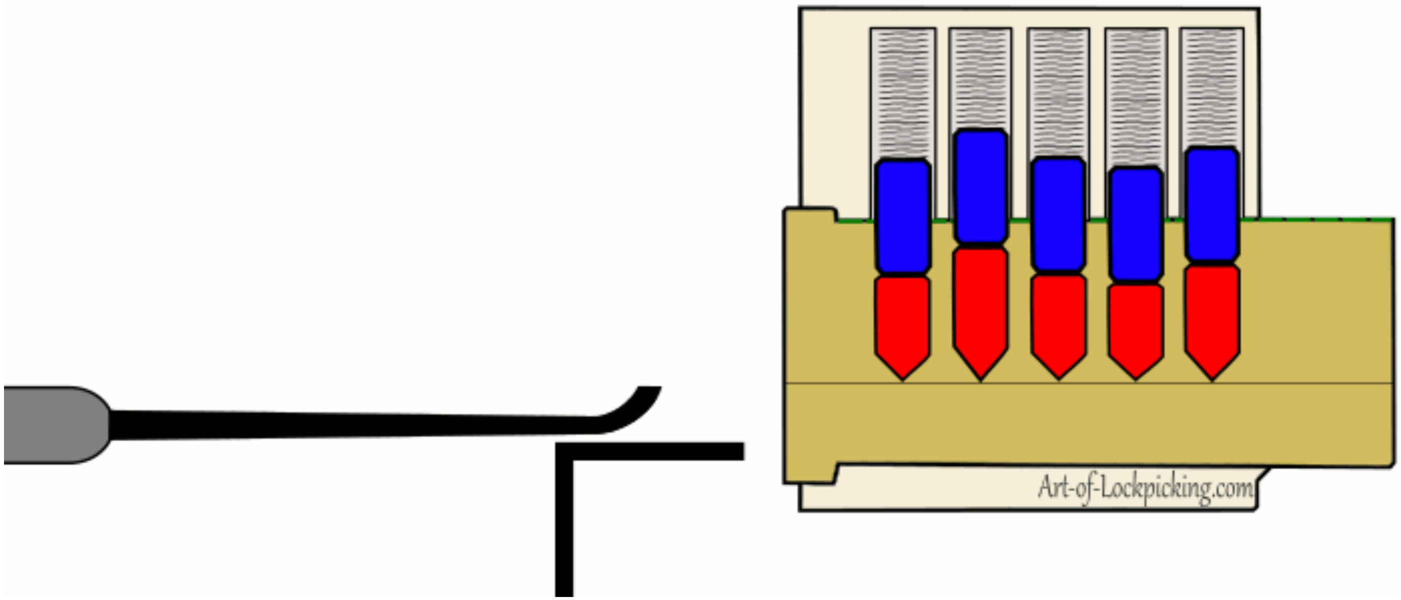
Overview

- Course theme
- Five realities
- How the course fits into the CENG curriculum
- The specifics of the course
- Academic integrity

Course Theme:
Abstraction Is Good But Don’t Forget Reality

Lock/key abstraction/hacking





Abstraction Is Good But Don't Forget Reality

- **Most CENG 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

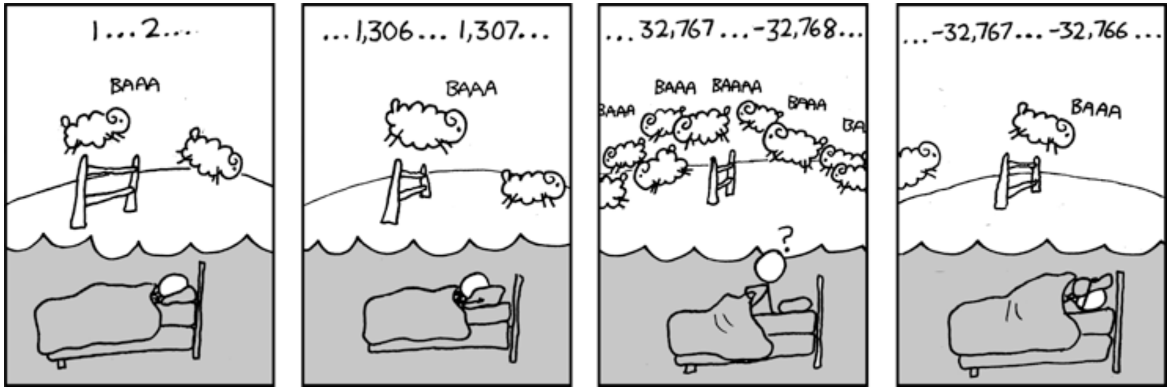
Computer Systems: A Programmer's Perspective (CS:APP)

- **Also known as Computer Architecture**
  - Cover topics about how one can build a computer from scratch
  - Older/alternative coverage was a continuation of CENG232
  - Designing circuits for CPU, cache, memory etc.

- **CS:APP: Useful outcomes from taking CENG331**
  - 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/CENG
    - Compilers, Operating Systems, Networks, Embedded Systems, etc.

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

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
    - 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;
}
```

```
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

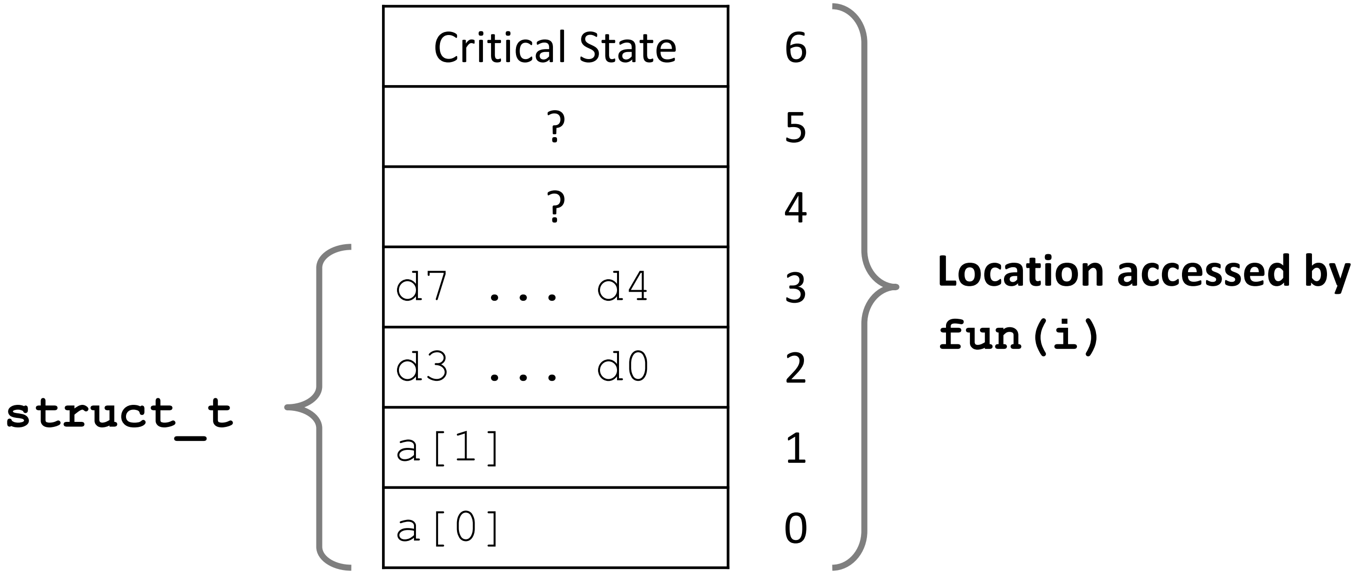
volatile keyword indicates that a value may change between different accesses, even if it does not appear to be modified. This keyword prevents an optimizing compiler from optimizing away subsequent reads or writes and thus incorrectly reusing a stale value or omitting writes

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

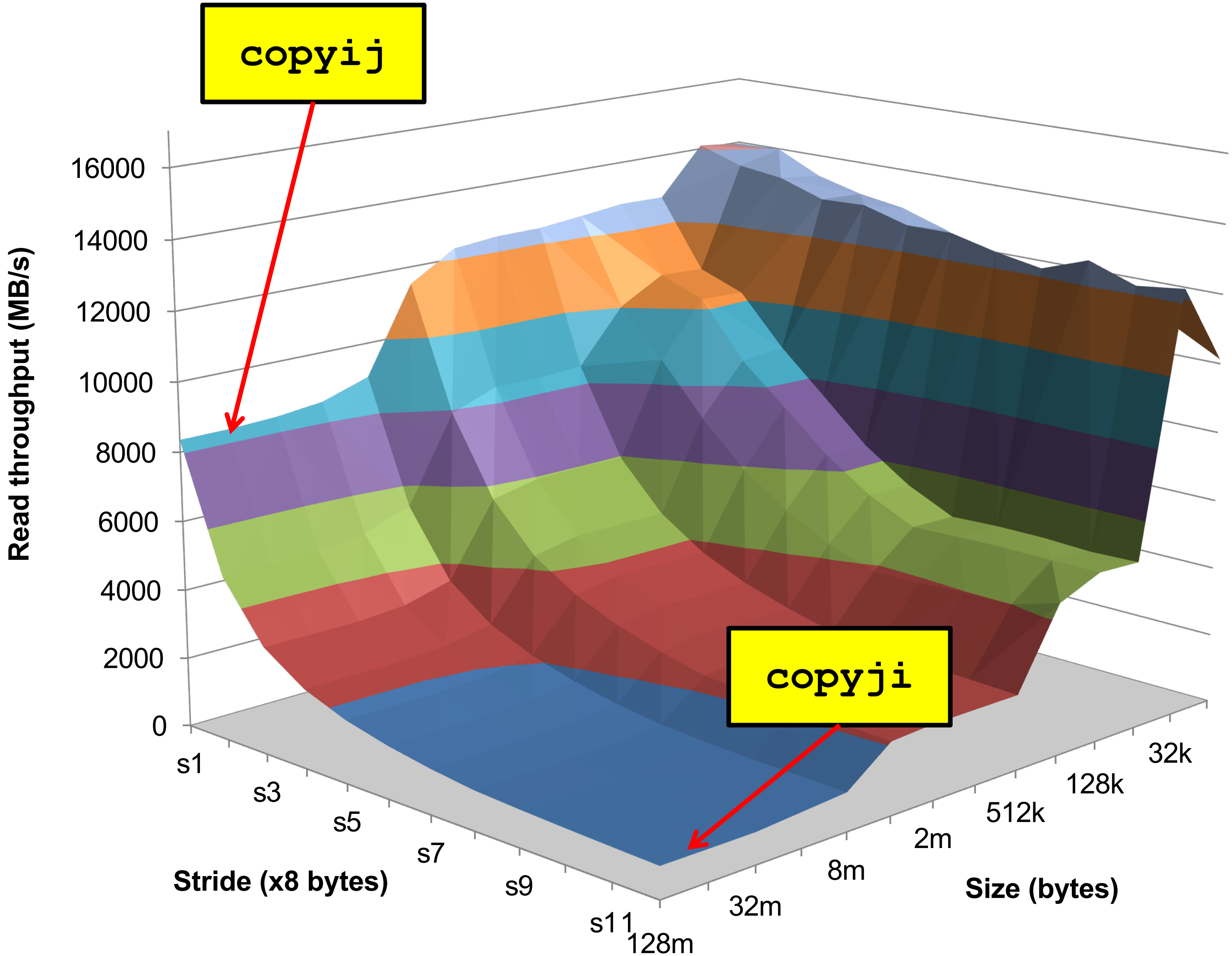
```
void copyij(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (i = 0; i < 2048; i++)
    for (j = 0; j < 2048; j++)
      dst[i][j] = src[i][j];
}
```

```
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

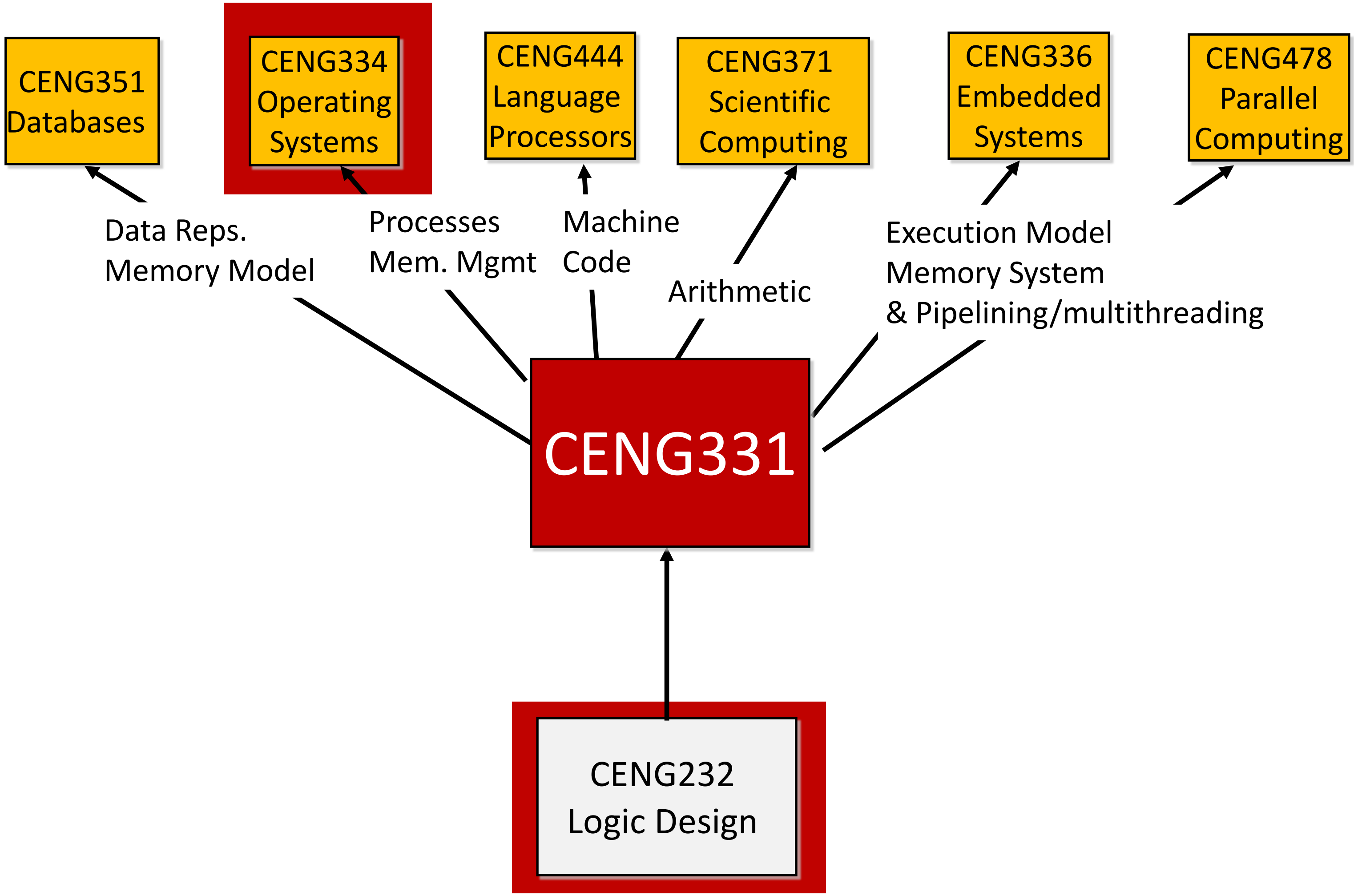
Course Perspective

- Computer Architecture
  - Older versions of the course had focus from a designers perspective
- Most Systems Courses are Builder-Centric
  - Operating Systems
    - Implement sample portions of operating system
  - Compilers
    - Write compiler for simple language
  - Networking
    - Implement and simulate network protocols

Course Perspective (Cont.)

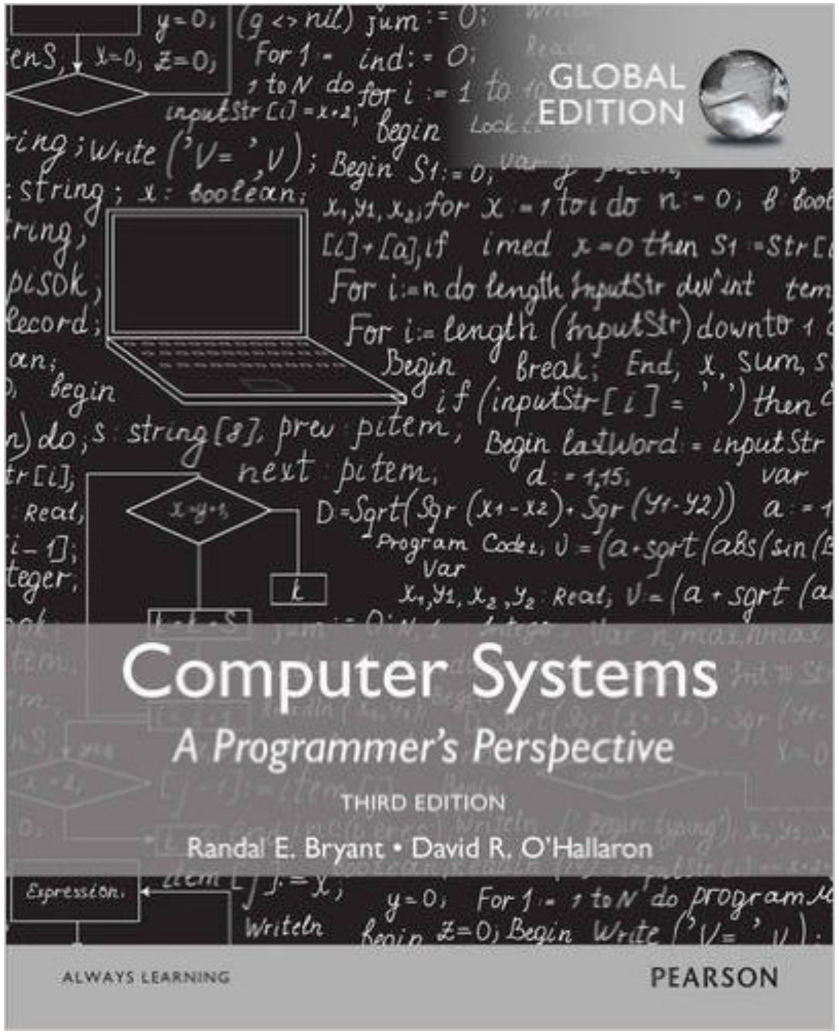
- Our Course is Programmer-Centric
  - Purpose is to show that by knowing more about the underlying system, one can be more effective as a programmer
  - Enable you to
    - Write programs that are more reliable and efficient
    - Incorporate features that require hooks into OS
      - E.g., concurrency, 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!**

Role within CENG Curriculum



Textbook

- 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 labs
    - Practice problems typical of exam problems



Course Components

- Lectures (online)
  - Higher level concepts
  - Lecture videos of Randal Bryant from CMU
  - https://scs.hosted.panopto.com/Panopto/Pages/Sessions/List.aspx#folderID=%22b96d90ae-9871-4fae-91e2-b1627b43e25e%22&view=0&maxResults=250&sortColumn=0&sortAscending=true
- Take-home labs (4)
  - The heart of the course
  - Provide in-depth understanding of an aspect of systems
  - Programming and measurement
- Exams (midterm + final)
  - Test your understanding of concepts & mathematical principles
- Quizzes
  - Simple questions to test your understanding within that lecture

Communication

- ODTUClass [Also has a mobile app]
  - Announcements
  - Resource sharing
  - Discussions
  - E-mail: erol@metu
    - Subject Line: CENG331

QR code for Moodle App

- ODTUClass
  - https://odtuclass2022f.metu.edu.tr/course/view.php?id=3485
- (Google) Course Calendar - you can add it to your own calendar
  - Will be shared before next week

Take-home Labs

- **Bomb lab**
  - Followed by an online quiz after submission
  - Lab submission grade will be modulated with the quiz grade
- **Attack lab**
  - Followed by an online quiz after submission
  - Lab submission grade will be modulated with the quiz grade
- **Architecture lab**
  - The similarity will be evaluated using MOSS
  - Oral exams will be conducted in suspicious cases
- **Performance lab**
  - The similarity will be evaluated using MOSS
  - Oral exams will be conducted in suspicious cases

Teaching Assistants

- Çağrı Utku Akpak.
  - E-mail: cakpak@ceng
- Merve Taplı.
  - E-mail: mtapli@ceng
- Can Ünaldı
  - E-mail: cunaldı@ceng
- Cem Önem
  - E-mail: onem@ceng
- Office hours:
  - All: By appointment

Policies: Assignments And Exams

- Work groups
  - You must work alone on all assignments unless otherwise announced
- Exams
  - **Face-to-face exams in person**
  - Midterm → To-be-announced later
  - Final → will be announced by METU
- Quizzes (smart attendance)
  - One or two simple questions
  - Graded in ternary

Policies: Grading

- Midterm: 30%
- Take-home Labs: 24%
- Quizzes (a.k.a. smart attendance): 10%
- Final Examination: 36%

Lab Grading

Algorithm 1 Lab Grading

$l : \{Bomb, Attack, Architecture, Performance\}$
procedure LAB(l)
  $H \leftarrow$ Your grade from lab homework, out of 100 + bonus
  if $l \equiv Bomb \vee l \equiv Attack$ **then** ▷ Bomb and Attack labs have quizzes
    if $H < 50$ **then**
      $Q \leftarrow 0$ ▷ Not allowed to take the quiz
    else
      $Q \leftarrow$ Your grade from lab quiz, out of 100
    $L \leftarrow 0.6 * H + 0.4 * Q$ ▷ Your final grade from the lab
  else ▷ Architecture and Performance labs have no quizzes
    $L \leftarrow H$ ▷ Your final grade from the lab

Course grading

Algorithm 2 Course Grading

procedure COURSEGRADING
    $MT \leftarrow$ Your grade from Midterm, out of 100
    $Att \leftarrow$ Your grade from attendance and online quizzes, out of 100
    $Labtotal \leftarrow Lab(Bomb) + Lab(Attack) +$
        $Lab(Architecture) + Lab(Performance)$
    if $0.06 * Labtotal < 10$ **then** ▷ Not allowed to take the final
        $LetterGrade \leftarrow NA$ ▷ Failure with no Resit exam option
    else
        $Final \leftarrow$ Your grade from Final, out of 100
        $Total \leftarrow 0.3 * MT + 0.36 * Final + 0.1 * Att + 0.06 * Labtotal$
        $LetterGrade \leftarrow$ Letter based on $Total$ ▷ Letter grades FF to AA

Programs and Data

- Topics
  - Bits operations, arithmetic, assembly language programs
  - Representation of C control and data structures
  - Includes aspects of architecture and compilers

- Assignments
  - L1 (bomblab): Defusing a binary bomb
  - L2 (attacklab): The basics of code injection attacks

Processor Architecture

- Topics
  - Y86-64 architecture
    - Pipelining and hazards
    - Control structures

- Assignments
  - L3 (architecturelab): performance improvement in a pipelined processor architecture

Code optimization and Memory Hierarchy

- Topics
  - Code optimization
  - Memory technology, memory hierarchy, caches, disks, locality
  - Includes aspects of architecture and OS

- Assignments
  - L4 (performancelab): Improve the performance of a kernel which is a bottleneck in an application

Virtual Memory

- Topics
  - Virtual memory, address translation
  - Includes aspects of architecture and OS

Lab Rationale

- Each lab has a well-defined goal such as solving a puzzle or winning a contest
- Doing the lab should result in new skills and concepts

- We try to use competition in a fun and healthy way
  - Set a reasonable threshold for full credit
  - Post intermediate results (anonymized)

Cheating: Description

- What is cheating?
  - Sharing code: by copying, retyping, **looking at**, or supplying a file
  - Describing: verbal description of code from one person to another.
  - Coaching: helping your friend to write a lab, line by line
  - Searching the Web for solutions
  - Copying code from a previous course or online solution
    - You are only allowed to use code we supply, or from the CS:APP website
- What is NOT cheating?
  - Explaining how to use systems or tools
  - Helping others with high-level design issues

- See the course syllabus for details.
  - Ignorance is not an excuse

Cheating: Consequences

- Penalty for cheating:
  - Disciplinary action
- Detection of cheating:
  - We have sophisticated tools for detecting code plagiarism
  - And other forms of cheating
- Don’t do it!
  - Start early
  - Ask the staff for help when you get stuck


FAILURE IS
ALWAYS
AN OPTION

FAILURES

… are part of life.

If you don’t fail, you don’t learn.

If you don’t learn, you’ll never change.

Welcome
and Enjoy!