Course Overview

Introduction to Computer Systems

Overview

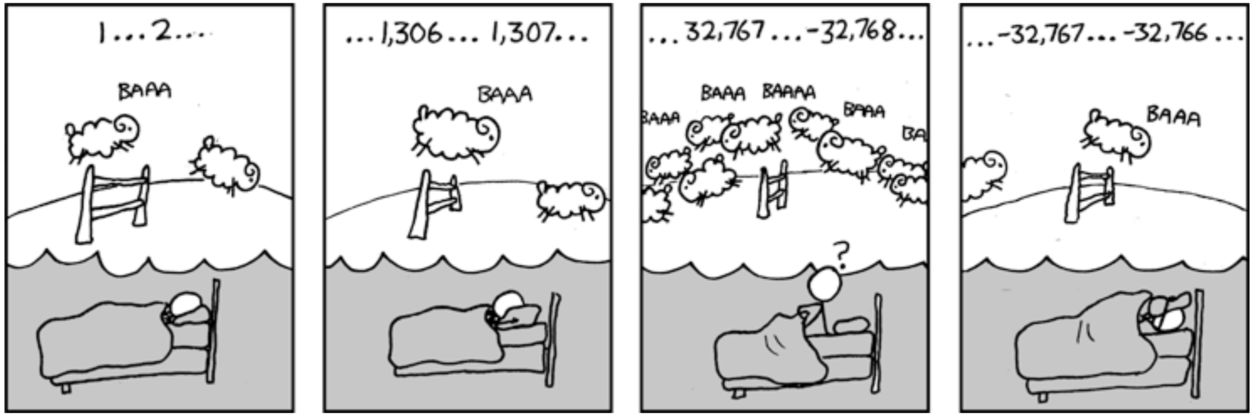
- Course theme
- Five realities
- How the course fits into the CS curriculum
- Academic integrity

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

- **Most CS 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 from taking this course**
  - 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
    - Computer Organization, Compilers, Operating Systems, Networks, Computer Architecture, Embedded Systems, Storage 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 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
```

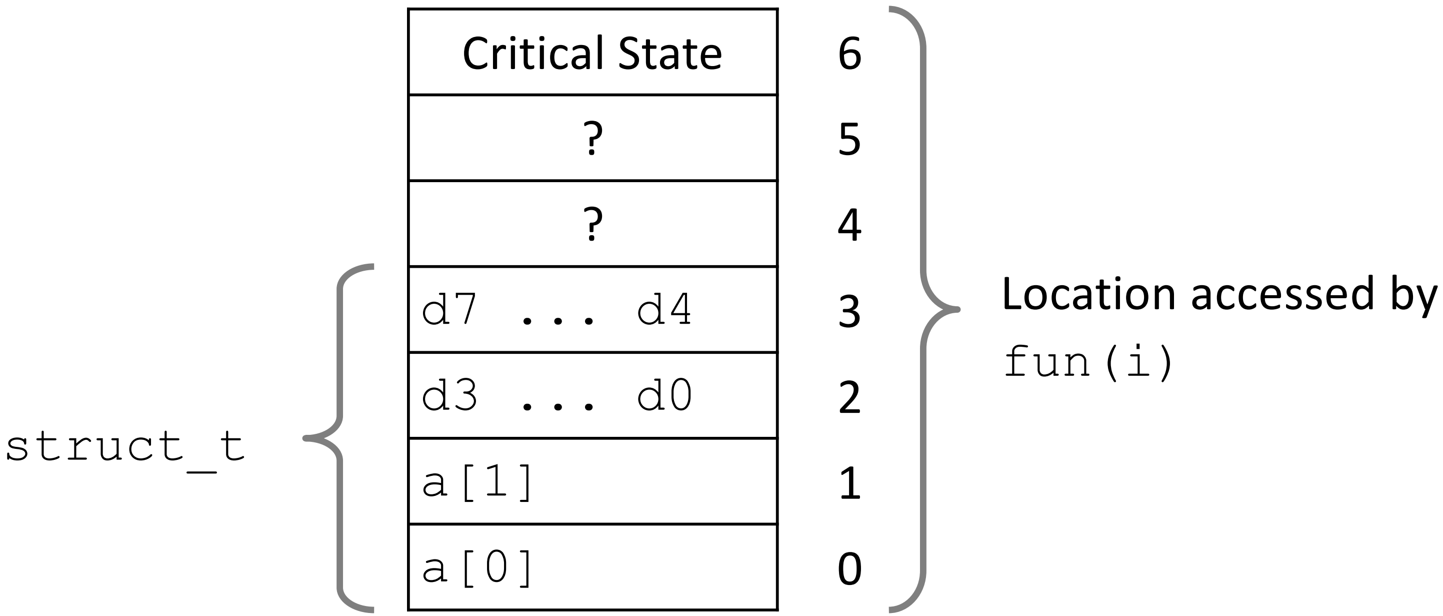
- Result is system specific

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, Python, Matlab, …
  - Understand what possible interactions may occur
  - Use or develop tools to detect referencing errors (e.g. Valgrind)

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

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

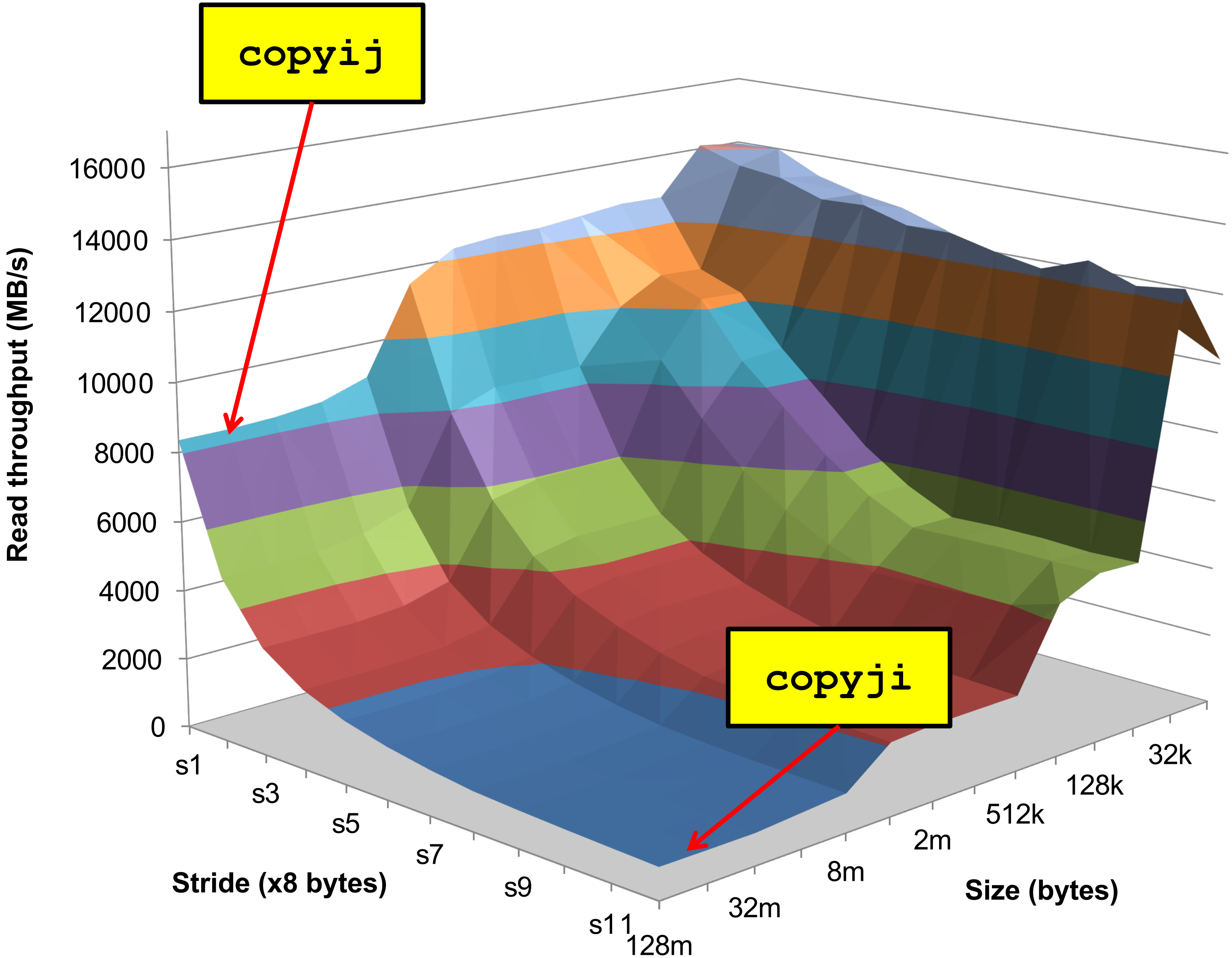
```
void copyij(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (i = 0; i < 2048; i++)
    for (j = 0; j < 2048; j++)
      dst[i][j] = src[i][j];
}
```

```
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

- Most Systems Courses are Builder-Centric
  - Computer Organization and Architecture
    - Design pipelined processor in Verilog
  - 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!**

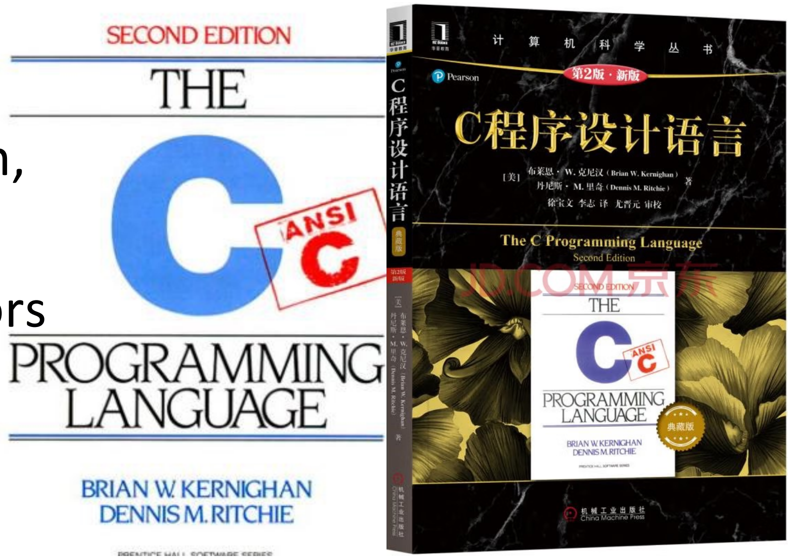
Cheating: Description

- 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
- What is NOT cheating?
  - Explaining how to use systems or tools
  - Helping others with high-level design issues

Textbooks

- Randal E. Bryant and David R. O’Hallaron,
  - *Computer Systems: A Programmer’s Perspective*, **Third Edition** (CS:APP3e), Pearson, 2016
  - This book really matters for the course!
    - How to solve labs
    - 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
- Labs (4-5)
  - The heart of the course
  - 1-3 weeks each
  - Provide in-depth understanding of an aspect of systems
  - Programming and measurement
- Flipped Classroom
- Final Exam
  - Test your understanding of concepts & mathematical principles

往届同学课程学习心得

可以让人对计算机体系有了一个总体的了解，这门课程不仅让后续课程的学习变得轻松了许多(其他班的同学学体系结构时看见汇编都被吓了大跳)，也让人学习了很多实际中有用的东西。比如说实验就要求安装Linux来使用。

感觉对具体实现有个初步概念，然后一步一步很有成就感，解开谜团的感觉。对将来的电脑学习充满了兴趣。

计算机系统基础是让我开始觉得计算机真的有意思的课程。做labs虽然耗费精力但是好玩。对计算机组成原理的过渡效果非常好，计组学习基本没有太大压力。尤其是优化程序性能的那一部分，对编程帮助也蛮大的。

别整虚的（比如相信书光看就能看懂，大佬除外），多动手做课后题和lab

计算机系统基础是CMU的镇校神课当然要上

Other Rules of the Lecture Hall

- Laptops: permitted

- Electronic communications: forbidden
  - No email, instant messaging, cell phone calls, etc

- Presence in lectures: voluntary, recommended by me, while requested by the school

Assistance

- 通知、答疑：QQ群/私信
- 作业与实验平台：计算机学院希冀平台
  - https://cslabcg.whu.edu.cn/
  - 账号已导入，务必登陆后补充email，方便找回密码
- 教辅TA：QQ群，批改作业，答疑
- 线下：教师办公室时间
  - 每周四中午12点～2点

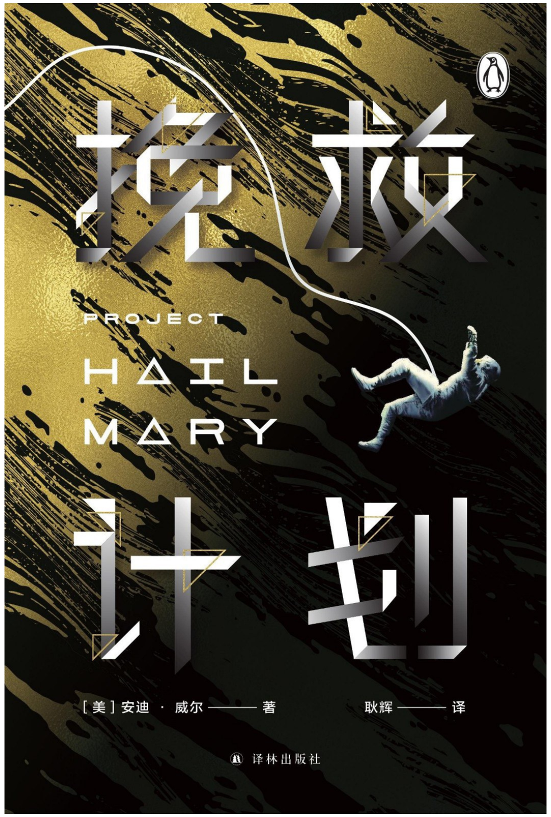
Policies: Grading

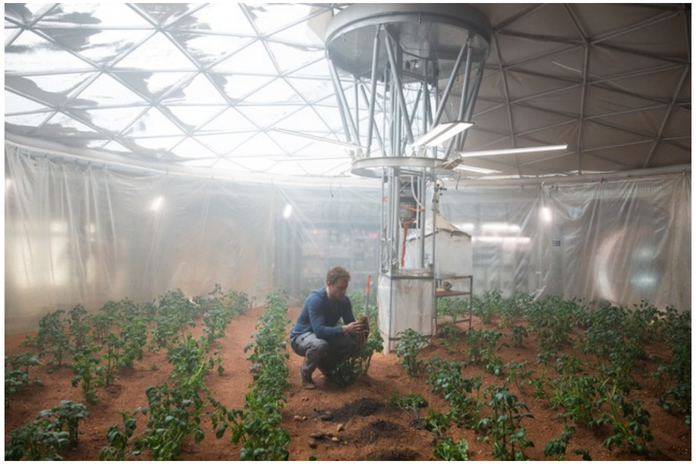
- Exams (60%)
- Labs (20%): weighted according to effort
- Classroom presentation (10%)
- Homework (10%)

Labs

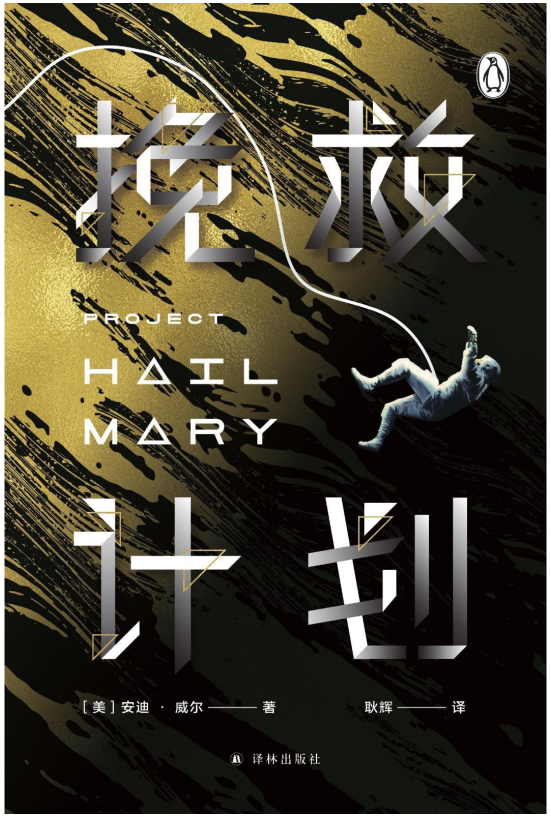
- L0 (clab): Basic C programming skills
- L1 (datalab): Manipulating bits
- L2 (bomblab): Defusing a binary bomb
- L3 (attacklab): The basics of code injection attacks
- L4 (shelllab): A simple Unix shell program with job control
- L5 (proxylab): A concurrent caching Web proxy that sits between the browser and the rest of the World Wide Web

如何学好一门系统课程？





如何学好一门系统课程?



- 在陌生的环境中醒来→想知道自己在哪里?
- 用卷尺测量桌子的高度（91cm），一支试管（高密度、可以忽略空气阻力），用秒表记录它掉落的时间（0.37s），重复实验20次，得到平均值0.348s。$d=1/2gt^2$，所以$g=15m/s^2$，是$9.8m/s^2$的1.5倍→我不在地球上！
- 我可能在一台离心机里?
- 一卷尼龙绳，卷尺，做一个单摆，无论幅度多大，摆动一个来回的时间——周期——是恒定的，只取决于单摆的长度和重力加速度。高处，346周，10分钟；低处，346周，10分钟！如果在离心机里，你离中心越远，向心力越大，结果没有差别，至少没有达到影响单摆的周期。
- 开始排除超大离心机的可能性（半径至少700m，转速88m/s，会有涡流、风噪）
- 在另一颗行星？太阳系内没有任何一颗行星、卫星或小行星有这么大引力，除非在木星的风暴里，否则不会体验到这么大的重力加速度....

Welcome

&

Enjoy!