ECE 551
C++ Programming, Data structures, and Algorithms

Introduction

Course Overview

Programming Ability Now

Admin

- Professor: Andrew ("Drew") Hilton
  - E-mail: adhilton@ee.duke.edu
  - Office Hours: Hudson 211, Thursdays 2:00-3:30
    - Or by appointment (e-mail me, we'll setup a time)
- Co-Instructor: Genevieve Lipp
  - E-mail: genevieve.lipp@duke.edu
  - Office Hours:
- TAs (OH to be posted on Piazza):
  - Lidea Shahidi
  - Georgios Mappouras
  - Runren Zhang
  - Ian Kaszubski
  - Bingyu Lan
  - Wanxin Yuan
  - Yuxiang Huang
  - Xi Chen

A bit about me

- Please, feel free to call me "Drew"
  - Actually, I **strongly** prefer that to "Professor Hilton"
- Who knows what my job is?
  - To teach you to be industry-ready programmers
  - Everything else I do is secondary to that
    - Research, reviewing papers, advising undergrads, sleeping…
- If you ask, I **will help you**
  - Office hours: for you
  - OH not enough? Set up 1:1 meeting
  - Chronically struggling? We can have a weekly 1:1 meeting

Learning your names

- I'd like to learn your names...
  - There are ~130 of you, so it takes a while
  - Please help me learn your names:
    - Feel free to remind me of it
  - Don't hesitate to correct my pronunciation of your name until I get it right

Topic Overview

Professional Tools	**Linux Emacs git gcc make valgrind**
Programming (in C)	**Reading Code Algorithms -> Code Strings Pointers Arrays Big O Dynamic Recursion**
C++	**Allocation Classes References Templates Inheritance Polymorphism Dynamic Dispatch**
Data Structures	**Stacks Queues ADTs Linked Lists Maps Sets BSTs Graphs Hash Tables Priority Queues Heaps**
Other Topics	**Sorting Object Layout Concurrency Multiple/Virtual Inheritance**

Professional Tools

- Why Emacs, Linux,...
  - "I want to use…"
  - NO(*): you need to be ready for professional world!

- Former students report VERY IMPORTANT to know
  - In interviews-> conveys message of "serious programmer"
    - I know many ACE programmers. All of them use Emacs or vim
  - On the job -> its what they use

(*) vim is acceptable, but I won't help you with it

From a Former Student

I was really mad at command line at first. But when I begin to look for jobs, these are the tools currently used by real companies. ... .These skills learnt on your course open lots of doors for me. Plus the programming experience and problem solving skills , getting an offer becomes fairly easy.

Programming: MUCH TO LEARN

- Not going to lie: you have a lot of hard work to do
  - Don't think this will be easy.
- Become competent in a thing? 10,000 hours of work
  - Get cracking!
- How could you hope to succeed?
  - Carefully designed pedagogy: teach you strong fundamentals
  - Requires SIGNIFICANT hard work on your part!
    - I'd suggest about 20 hours/week
  - Read and understand
  - Work on assignments, master concepts

20 Hours per Week???

Read Chapter [first time] (1 hour)

Re-read Chapter **Deeply** (2 hours)

??? ??? Work Practice Problems (~2 hours)

??? ??? Attempt Reading Quiz (30 minutes)

??? ??? **Class-time:** Do Problems (75 minutes)

??? ??? Catch up if behind on problems (??? minutes) Lets say 105 min

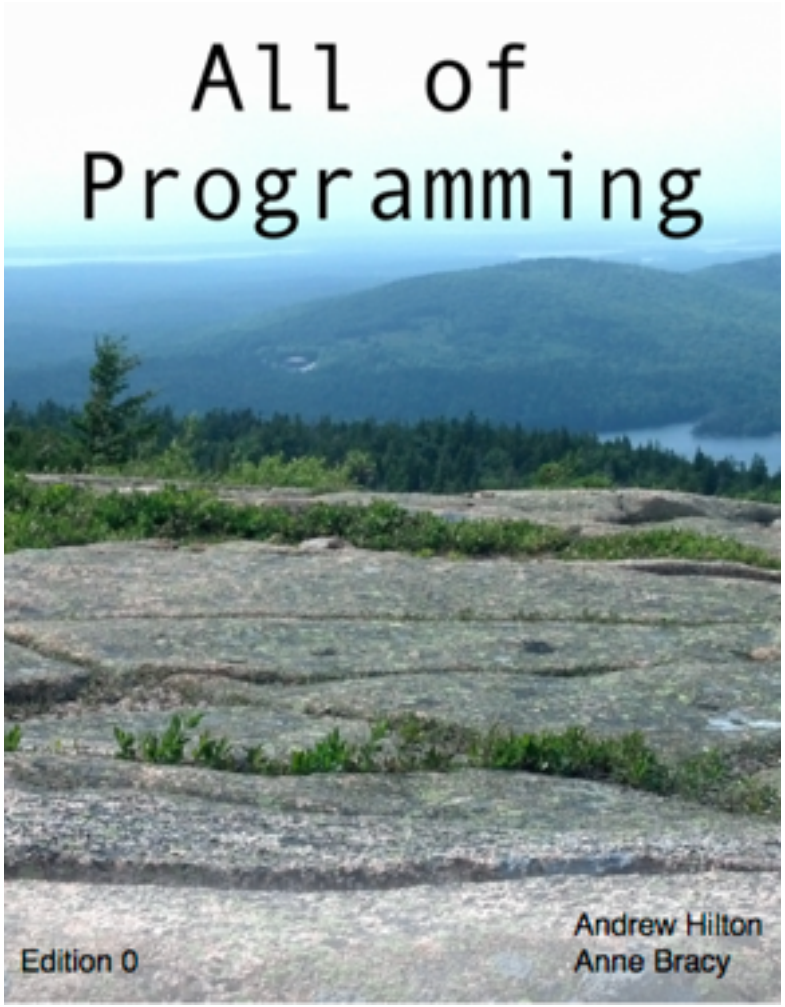
For Monday (~8.5 hours)

Wed: Repeat (~8.5 hours)

Fri Recitation 75 min

All of Programming

- Textbook: All of Programming
  - Hilton and Bracy, 2015
  - Edition 0
  - http://aop.cs.cornell.edu/
  - Reading schedule
    - On course webpage



Read Book, No Really

- Flipped Classroom
  - No lectures
  - Read your book!
    - Anne and I already wrote down everything you need to know…
  - Programming activities in class
    - Help available: don't stay stuck!
- Book: custom crafted for this course!
  - Everything you need to know
  - Nothing you don't.
- Can you understand it all first time through?
  - Maybe not, but…
  - Try to understand as much as you can
  - Work back through later if you want deeper mastery…

Reading, Taking Notes

- Keep notebook
  - Can use on exams
  - Everything must be **handwritten by you**
- Suggestions
  - Reading notes
  - Work on each classwork problem
  - Library reference
  - Tool reference

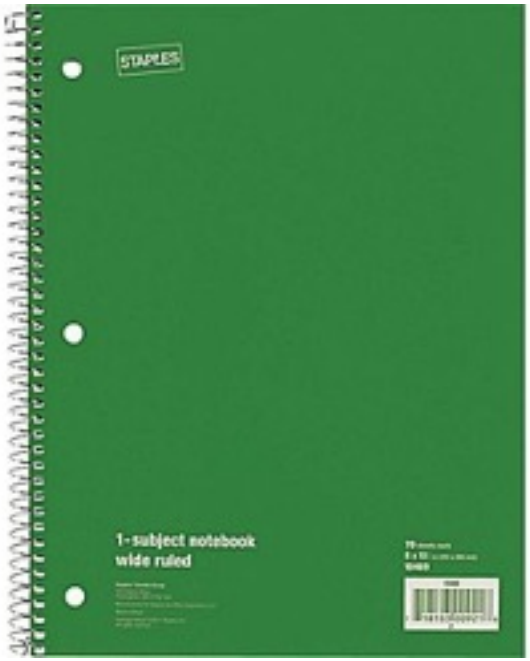
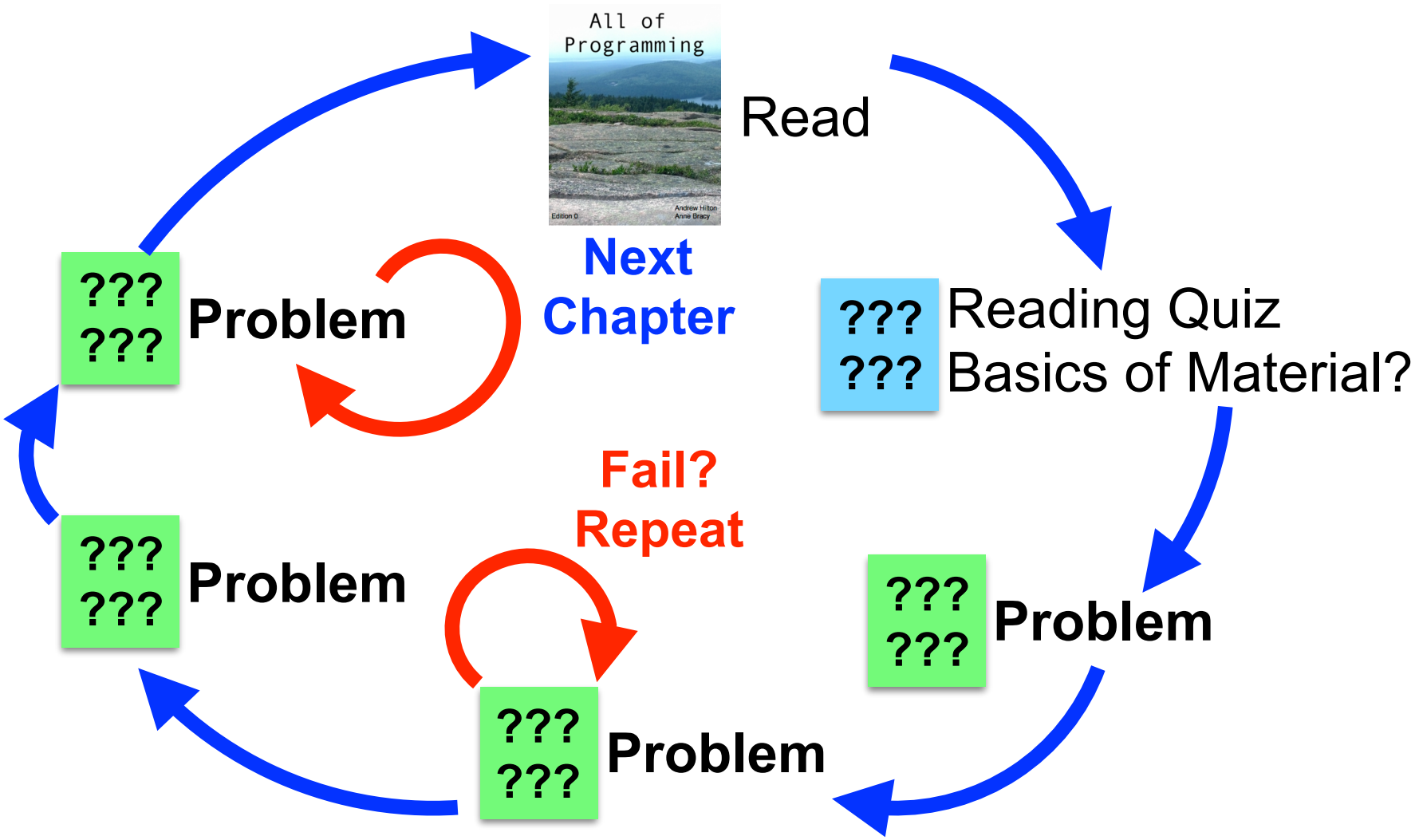


Image from staples.com

Read, Ask, Re-read, Understand

- Read your book
  - No seriously
  - I know, you don't for a lot of classes…
- But there is NO lecture in this class
  - You can't just magically learn to program!
- But, its so much work!
  - Yeah, I told you this wasn't going to be easy…

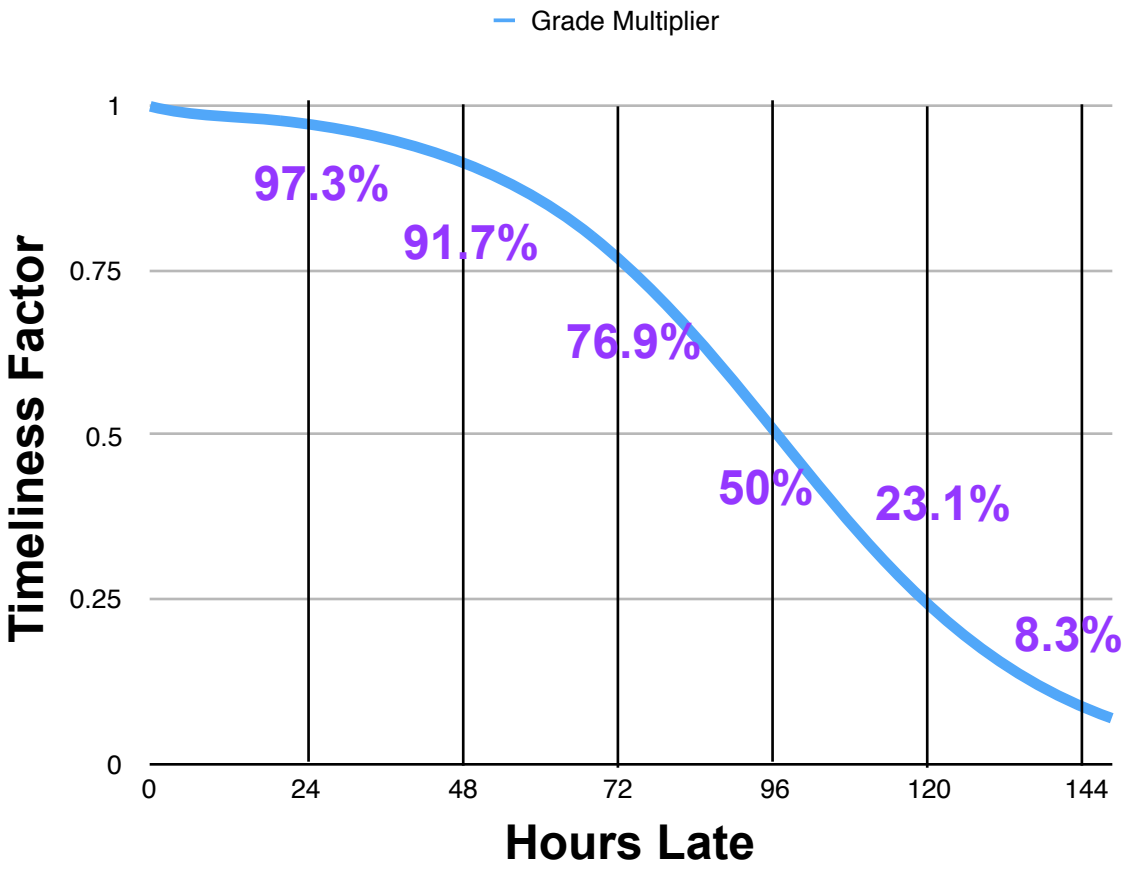
Flipped Class, Mastery Learning



Formative Assignments

- Most assignments: "formative"
  - For learning!
- Repeat until passed!
  - Can be resubmitted/regraded as much as you want (*)
- Collaboration?
  - Whatever it takes for you to **learn**
  - Ask friends for help? Fine, but be sure you **learn**
- Grading/re-grading
  - Run "grade" to grade assignment
    - After git commit/push
  - Tokens: limit rate of regrading
    - Periodic refresh
    - Cost increases when past due date

Formative Assignments: Timeliness



$$\text{Grade} = \%\text{correct} * 1/(1+e^{0.05*(\text{Hours\_Late} - 96)})$$

Evaluative Assignments

- Evaluative Assignment
  - Show me what you know!
- Think of these as out of class programming **exams**
  - Must work on by yourself! No help from classmates
  - Limited outside resources: your notebook, and AoP
  - Graded **ONCE** after **strict** deadline
- 4 of these:
  - Assignment 29: 9/15 to 9/23
  - Assignment 43: 9/22 to 10/02
  - Assignment 60: 10/11 to 10/20
  - Miniproject: 11/09 to 12/03
- Depend on previous formative assignments
  - **Must pass prior assignments to get the assignment**

Academic Integrity

- Academic Integrity Expectations
  - I take academic integrity **VERY** seriously, and you should too
- **Students have been expelled from Duke for cheating**
  - Fall 2015: multiple students expelled for cheating in 551
- More important than your grade: what you learn
  - Interviews? Jobs?
- Think you are "helping your friend out"?
  - Will you be able to do their interview for them?
  - What happens if someone gets A, is clueless in interview?
    - Will you (or others) be able to get interviews?
- Ethics: very important!
  - Industry, as well as academia
- I have about a decade of experience in software forensics…

Academic Integrity

- Question?
  - Ask me
  - Afraid to ask (maybe he'll say no…): probably bad
- Someone else cheats?
  - Please report it
- Evaluative assignments:
  - Sign honor statement before receiving assignment

Some questions

- What makes a good programmer?
  - What makes a great programmer?
- Where does a good programmer spend most of her time?
  - Where does a great programmer spend most of her time?
- What skill/knowledge would you most like from 551?

What Makes a Terrible Programmer

- What Makes A Terrible Programmer?
  - Copy/paste code from Stack Overflow
  - No understanding of what it does
  - "Frankencoding"
- Other bad things:
  - Writes a bunch of stuff with no plan
  - Debug by randomly changing stuff…

What Makes a Good Programmer

- Good Programmer
  - Careful planning before coding
  - Deep understanding of what code does
    - Semantics of language (Ch 2)
  - Debugging by scientific method
  - Careful testing of code
  - Incremental development
    - Write small piece
    - Test carefully
    - Build on it

Story Time

- Before we talk about what makes a great programmer..
  - Story time…..
- Now, you all tell me, what makes a great programmer?
  - Discipline under pressure
  - Thinks of problematic cases in advance
  - Defensive coding
  - Expansive testing
  - Deep understanding of tools
  - Intimate knowledge of language semantics

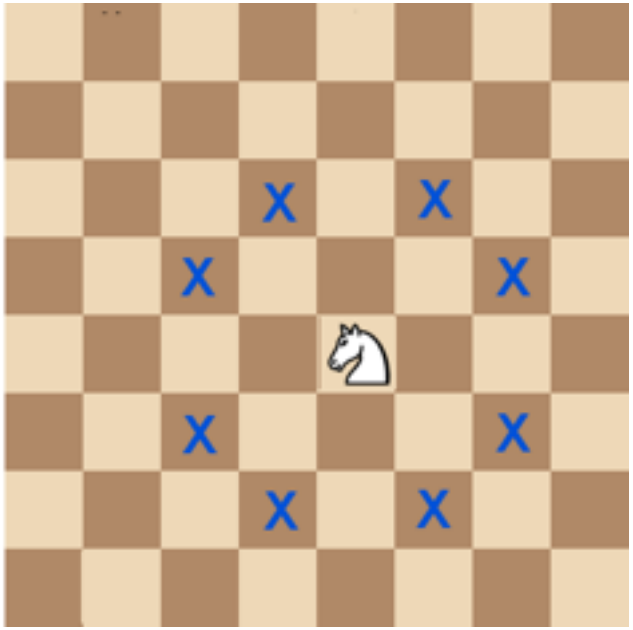
Let's highlight a couple of those points

- Careful planning before coding
  - Chapter 1: How to plan before you code
  - VERY IMPORTANT
- Deep understanding of what code does
  - Chapter 2: Syntax + Semantics
  - Build on these ideas in later chapters!
- Discipline under pressure
  - Stick to right way to do things
  - Even **(Especially)** under time pressure!

Bridges And Chess…

- Two analogies I want you to keep in mind:
  - Bridges and chess

Syntax vs…



=

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {
  printf("Hello World\n");
  return EXIT_SUCCESS;
}
```

- Syntax like teaching someone how how the pieces move
  - Important, core building block of "what can I do"
  - No notion of how to choose which "move"

Strategy, Planning, Problem Solving

- Being good at programming/chess much more than syntax

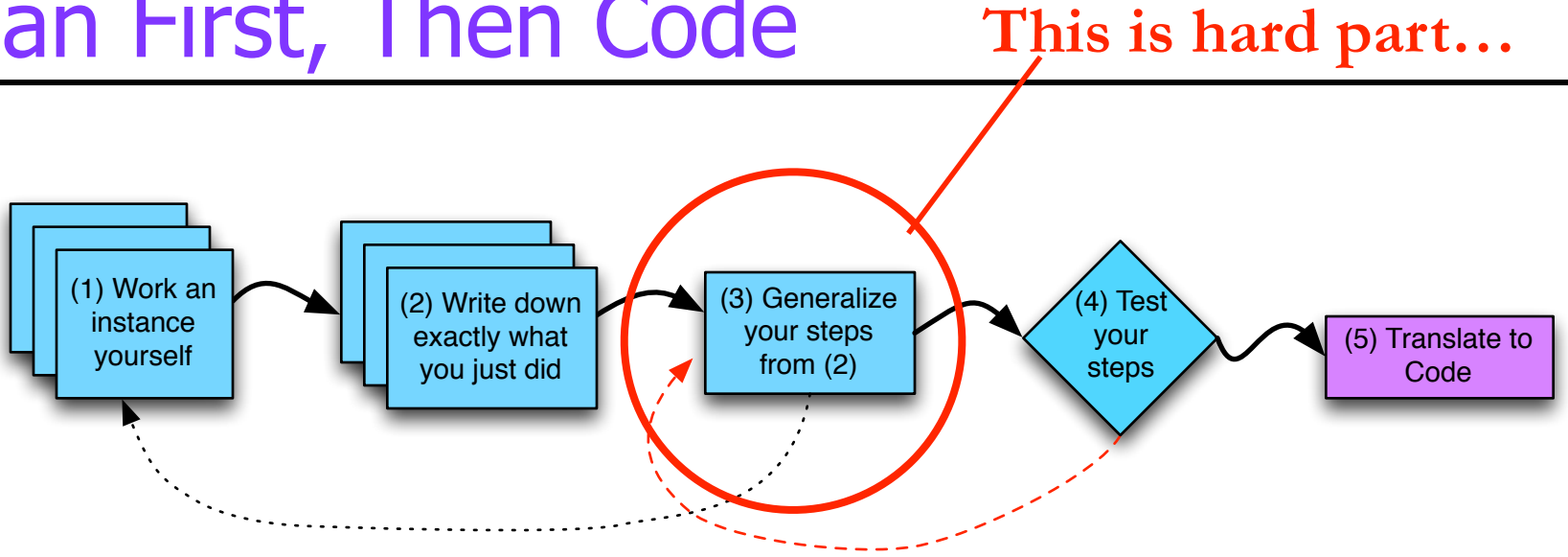
Bridges?

- Why are bridges a good analogy?

Bridges?

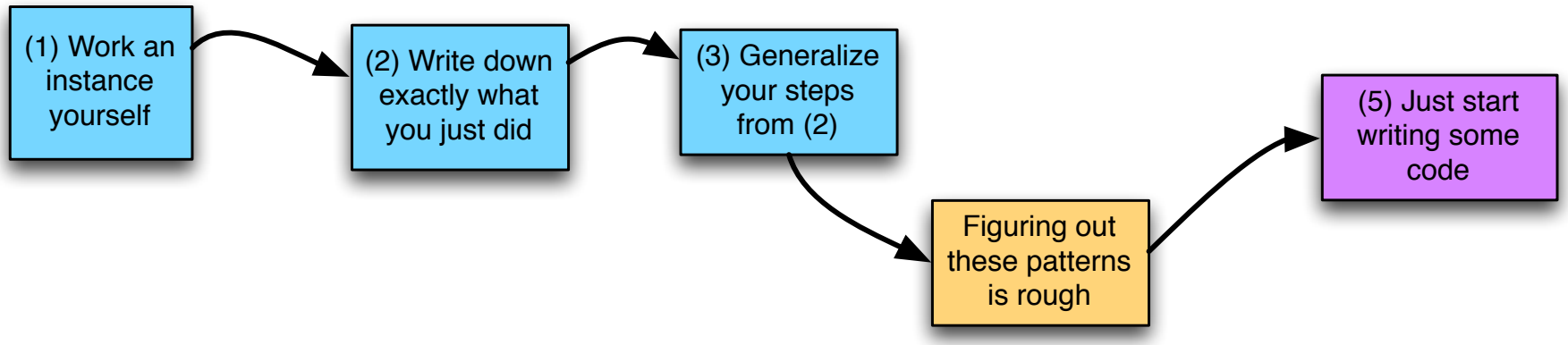
- Plan first, then build
  - Hard part
- Want to learn process, not memorize end results
- **Novices cannot learn from seeing a finished example**

Plan First, Then Code



- Important key: plan first, then code
- Step 3 is the hard part
  - Building a bridge: which is hard physics our pouring concrete?

Student Temptation



- This takes work, and thought, and…
  - Screw it, lets write some code
- Designing this bridge is hard…
  - Screw it, lets just start pouring concrete…

Failure to plan

- Result: code that looks like this…

Kludgy Messes

- Hacked together, kind of sort of works..
  - It passed a test case (I walked across it without dying…)

Disaster Lurks

- …but it keeps crashing on the harder tests

But, Nobody Will Ask In Interview…

- "Not worth my time, nobody will ask me in an interview"
- Who plays a musical instrument?
  - What did you start with?
  - Is that what you do in a recital?
  - Are those basics/fundamentals important?
    - Absolutely: form foundation for complex things
  - Nobody asks you to show them off
    - Expected
    - Ubiquitous in everything else you do
- Another analogy: learning to read
  - Who remembers first learning to read? What did you do?
  - "Sound it out": basic principles that let you build up to complex
  - Reading now?

Wisdom From Prior Courses

- Some things I've learned
  - Cultural barriers to asking for help exist
- You can always ask me for help
  - Your success is important to me
  - I will make time for you
  - No question is too simple
  - I will not be offended if you did not learn something

Wisdom Continued

- Practice Exams
  - I post a practice midterm and a practice final... With solutions!
  - Use them!... and use them well...
  - I've heard the following:
    - "I tried the practice exam, and didn't do well... but just hoped the real exam would be easier"
    - "I skimmed the practice exam and figured I could do the questions if I tried"
    - "I started with the solutions, and they all made sense, so I figured I would do fine."
    - "I didn't have time to try the practice exam. I was too busy studying [for this class]."
  - Take the practice exam, like a real exam (time constraints too!)
  - Check your answers **after** you finish.

Wisdom Continued

- Learn by doing
  - Seeing finished result will not help (think bridge example)
  - Watching friend solve problem will not help
  - Finding solution on Stack Overflow will not help
- Sit down and work on the problem
  - May be tough
  - …but I give you a step-by-step approach to apply
    - (Use it!)
  - Working through it will help you learn
- Stuck? Ask me (or TAs)

Wisdom Continued

- Listen to me when I tell you how to approach problems
  - **S:** "I tried the programming problems, but was completely lost."
  - **D:** "What step did you get stuck on?"
  - **S:** "I didn't know what to do at all."
  - **D:** "So you were stuck on step 1: work an instance yourself?"
  - **S:** "Oh no, I didn't do it your way... I was just trying to write the program"
  - **D:** "How about you try it my way?"
  - (later...)
  - **S:** "Hey, I tried it your way and it took me some time, but I got it!"

Wisdom Continued

- "Its all about the pictures" — student, mid-semester
  - Student basically discovered what I've been saying all along
- Listen to me, trust me, learn more, more quickly
  - But, _____?
- Pedagogy comes from years of experience
  - Hundreds of students taught, various ways of teaching
- If you had years, could develop intuitions over time
  - Figure out semantics, …
- Learn more quickly if you let me teach you!

A Few Big Picture Things to Start...

- My chance to get on my soapbox...
  - Before we plunge into technical content
- I also want you all to ask questions/discuss these as we go

Your Programs Will Run Our World

- You all are Master's students:
  - In 1--2 years you will be writing code which runs our world
  - Flies our airplanes...
  - Controls our nuclear reactors...
  - Drives our cars...
- This means I have to teach you to code, and to code right!
  - Different from undergrad intro courses
  - I expect you all to put in a lot of effort
  - ...and want you to know that I will do whatever I can to help you
  - But you have to ask for it if you need it!

Soapbox #1

Bad Code Kills People

Bad Code Kills People

- **Fatal Software Errors**: Therac 25, Patriot Missile Failure, Panama NIO radiation overdoes,…
- Also, non-fatal, but really bad consequences: $440 M trading error for Knight Capital Group, Ariane 5 explosion, phone system failure, Mariner 1, east coast blackout of 2003, 1998 Mars orbiter,…
- Will only become more true as more of our world is software dependent
- Therefore, I want you all to learn to write good software:
  - No sloppiness
  - Think about corner cases
  - **Code defensively**

Defensive Coding

- What is defensive coding?
  - Let's start by thinking about "What is defensive driving"
- Defensive Coding
  - Anticipate worst case on user inputs
    - User may intentionally do the opposite of what you want
  - Check for errors
  - Do not use dangerous library functions
  - Do not assume
    - In this class: Answer to "Can I assume that...?" always **NO**
  - Write **portable code**
    - Do not rely on platform/compiler specific behavior
  - Be careful, not sloppy!

Don’t be sloppy!

- We will do a few things to prevent sloppiness
  - Enable compiler warnings (-Wall)
  - ...and treat as errors (-Werror)
  - Plan first, then code!
    - And test algorithm, thinking about corner cases
  - Understand what is wrong and why before trying to “fix”
    - Debugging as scientific method
- Later:
  - `valgrind` to check for memory access/allocation mistakes
  - May try some **code walk throughs** as class activities (TBD)

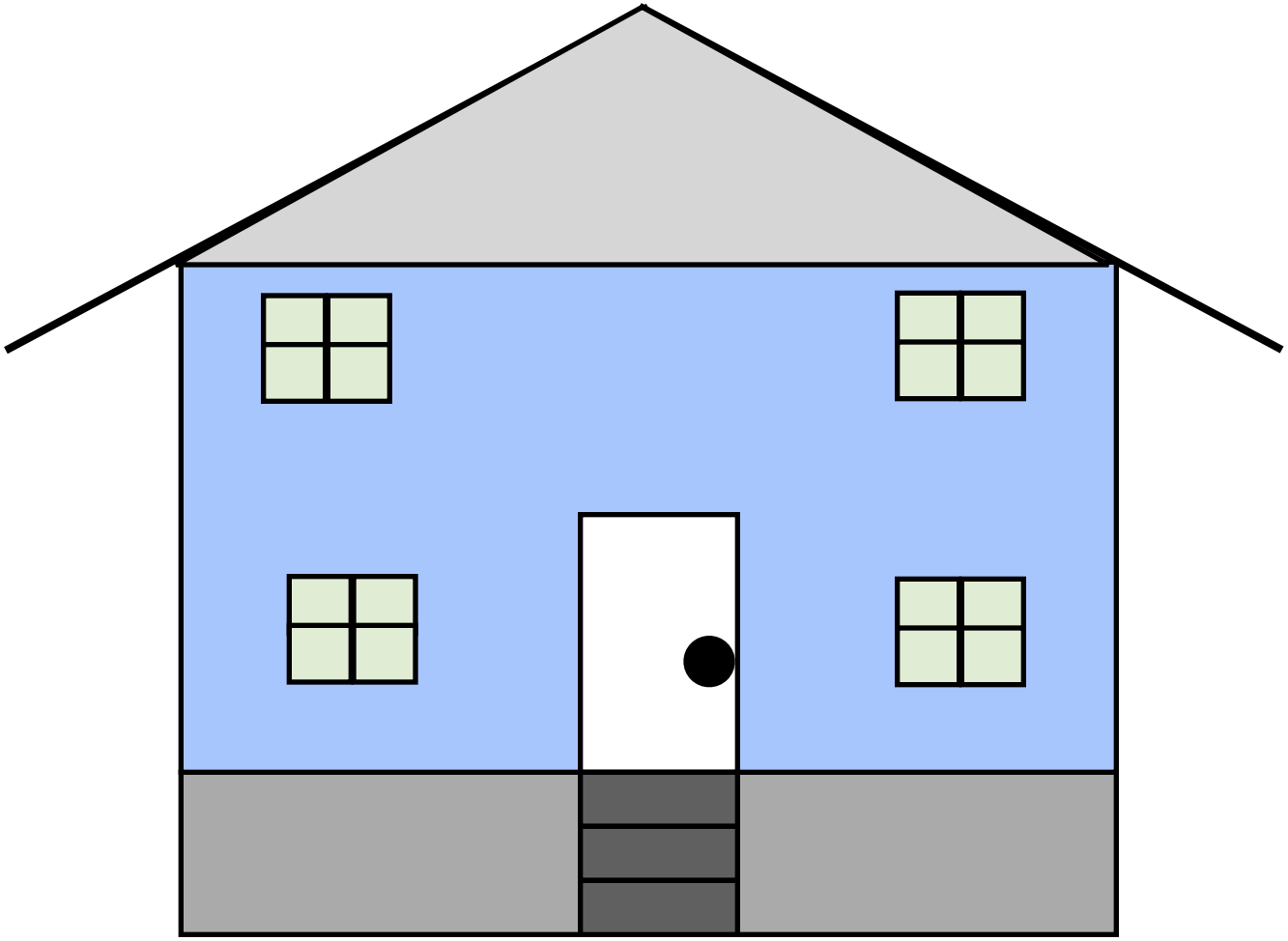
Soapbox #2

Coding = Basic Tool
Technical Depth = Job

Programming and Jobs

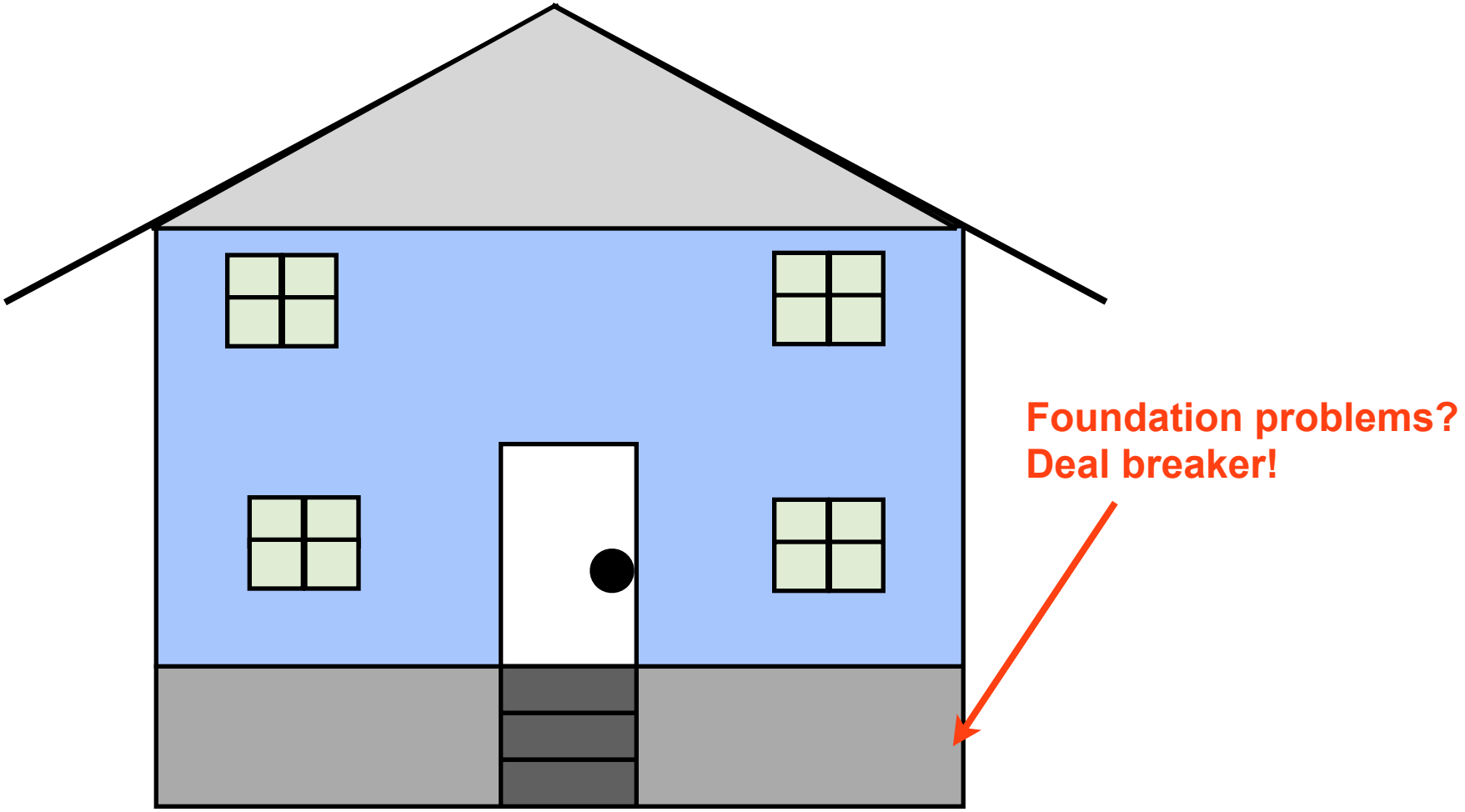
- Programming: good skill for good jobs...
- ...but its the **start**, not the end

House Analogy



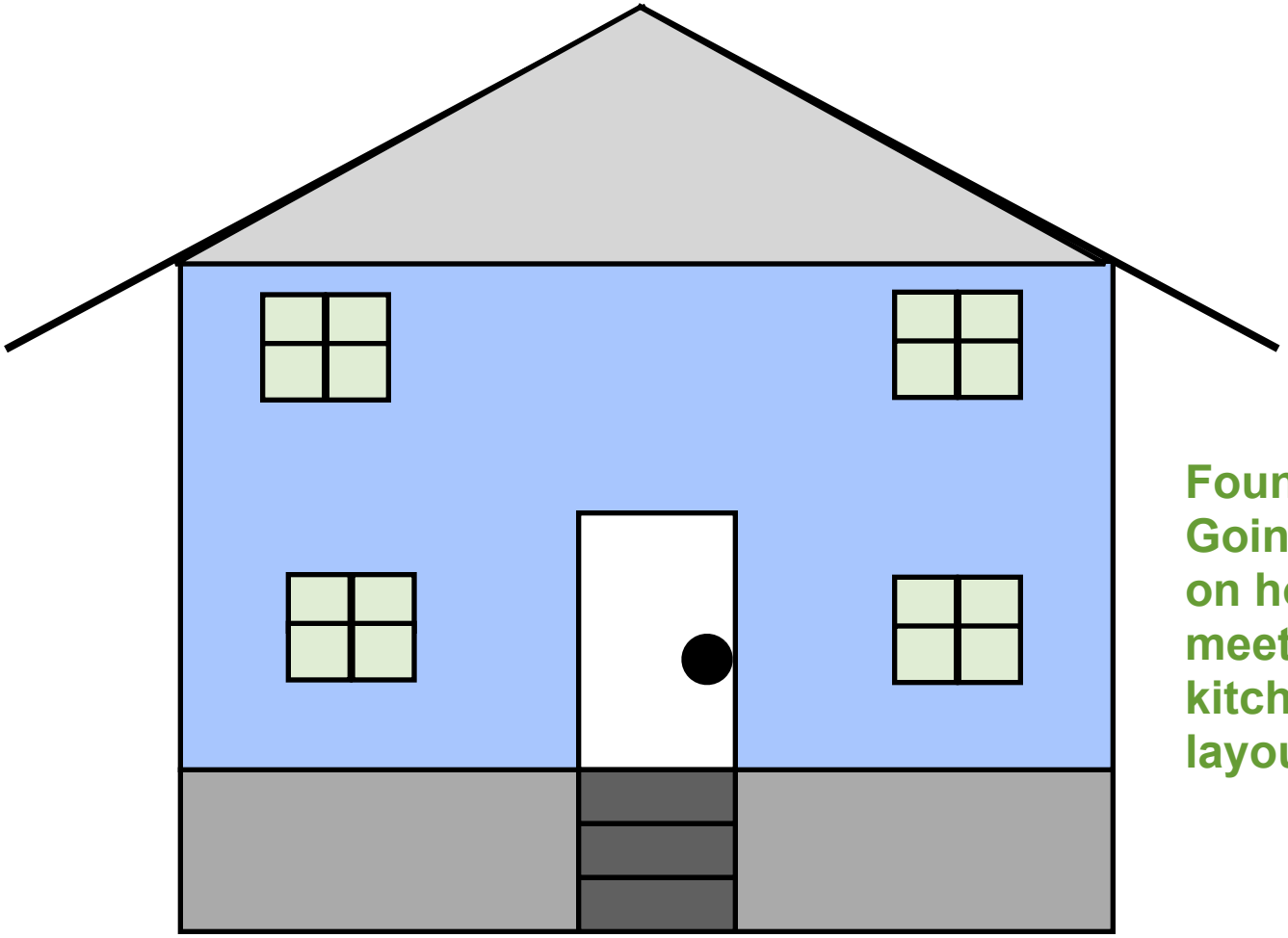
- Let us suppose you were buying a house....

House Analogy



- Let us suppose you were buying a house....

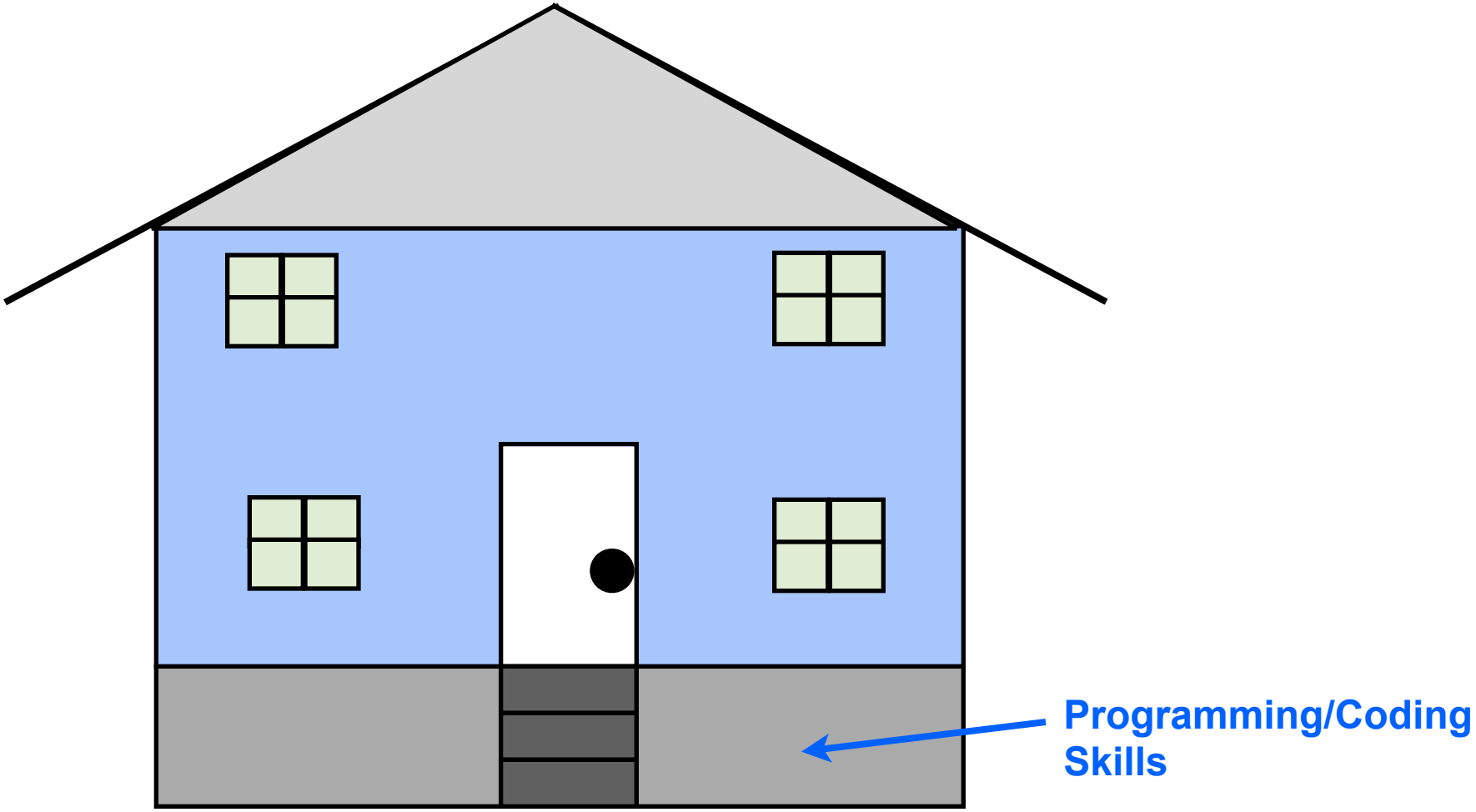
House Analogy



Foundation is good? Going to pick based on how well details meet your needs: kitchen, living room, layout...

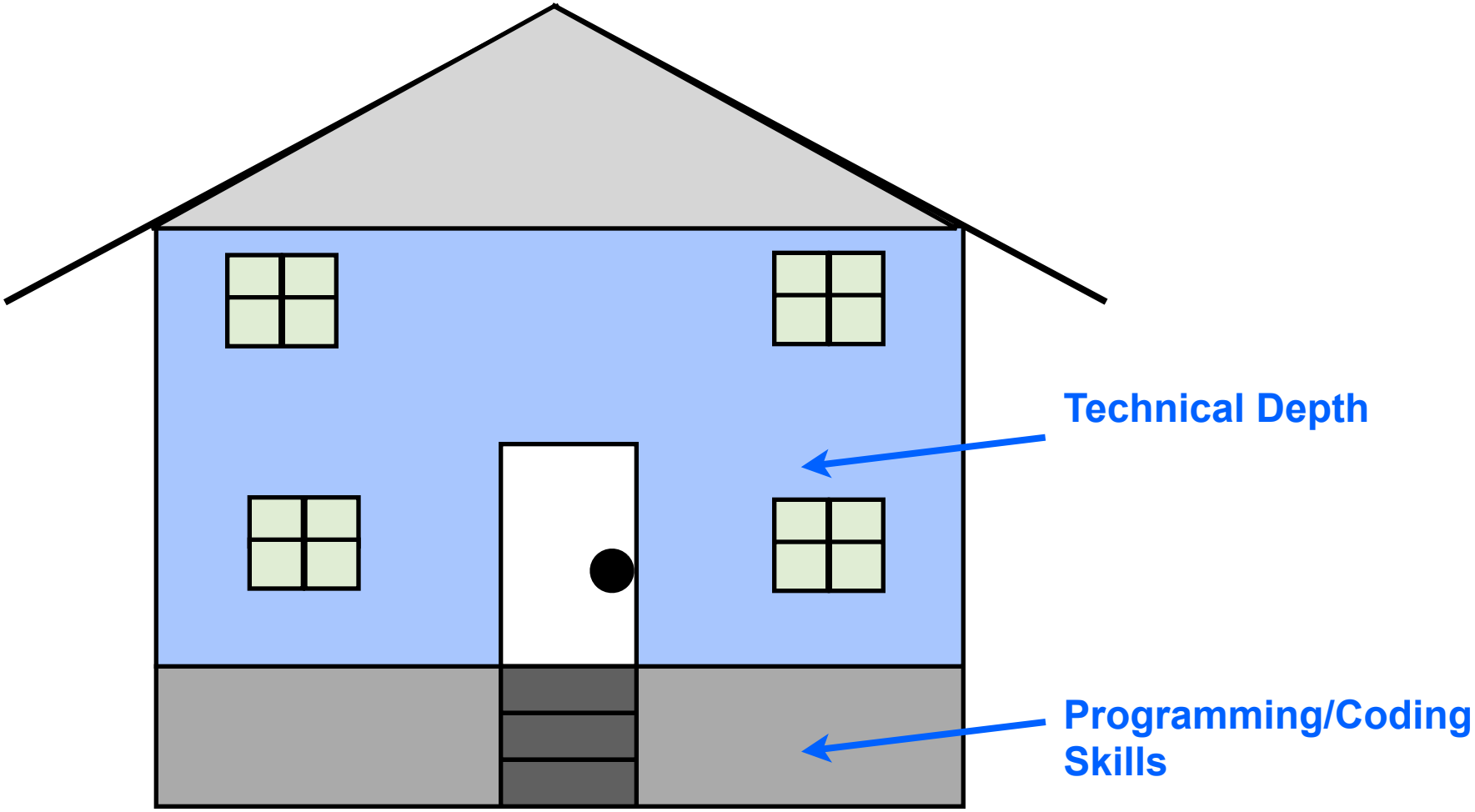
- Let us suppose you were buying a house....

House Analogy



- Programming Skills = Foundation

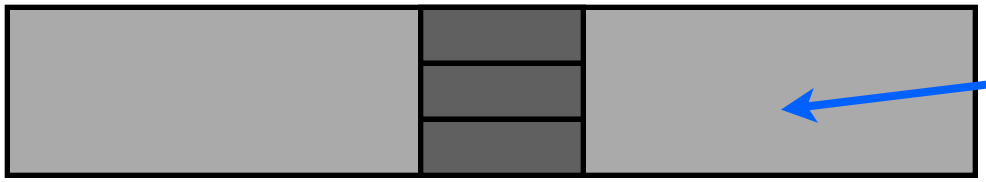
House Analogy



- Your "technical depth" is your main selling point

House Analogy

Hi, I'm a competent coder. Please Hire Me!



- Buy this foundation! There is nothing on top of it!

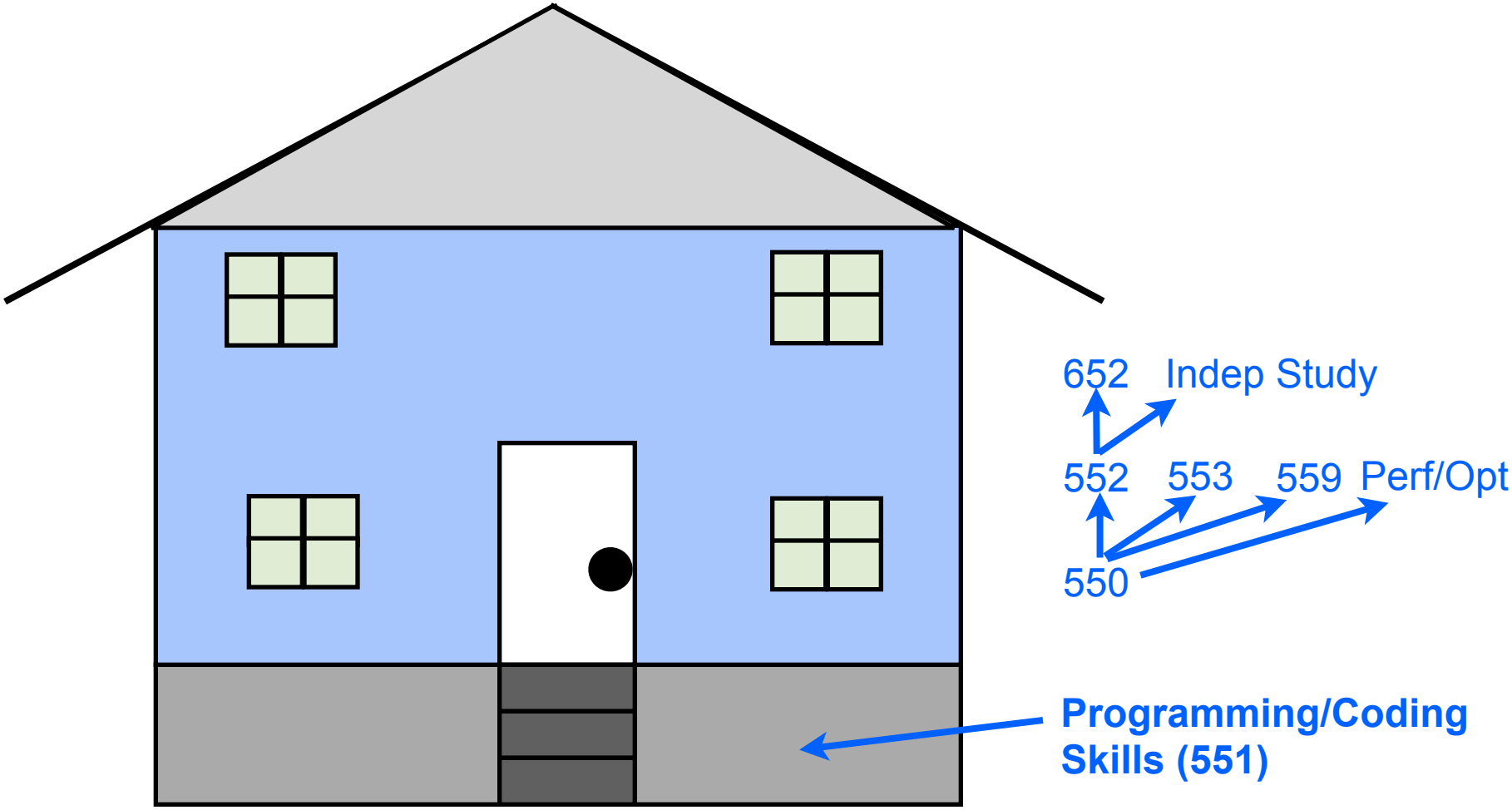
Technical Depth

- Technical depth: knowledge of some sub-field
  - Architecture **550, 552, 554, 559, 652, …**
  - Machine Learning **581, 582, 681, Vector Spaces, Text Analy, …**
  - Artificial Intelligence
  - Graphics
  - Software Engineering **651**
  - Cryptography
  - Compiler Design **553**
  - Security
  - Distributed/Systems, Parallel Programming **(various)**
  - Image Processing
  - …

Example: My last job

- My previous job in industry:
  - Hired for my computer architecture skills
  - Programming: what I did every day
    - I'm a top-notch programmer... but that wasn't what got me the job
    - ...and someone else with my programming skill but weak architecture knowledge would have been lost

House Analogy: Architecture Skills



- Example Technical Depth: built on foundation

Other Fields

- Can "build on the foundation" in many fields
  - We have some really great courses/profs in a variety of fields
  - Pick one you love!
    - Excitement about what you do goes a long way
  - Supporting breadth from other ("nearby") fields can be helpful
    - Increases options/flexibility too
    - Example: compilers skills very useful in my last job

Not 100% always the case

- This thought is general guidance: not a hard rule
- Example:
  - Friend of mine in industry was talking about interest in interns
    - Would rather have "smart people with solid programming skills"
    - ...teach technical depth in their particular company/field on job
  - Being an **amazing programmer** is itself technical depth
    - Distinguish yourself from everyone else: raw programming skill
      - ...but won't help if specific sub-field depth required
      - ...may help with "we'll teach specifics on the job"
    - My first undergrad internship: only raw programming skill
      - Side note: completely new language

Side Note

- Faculty in your sub-field of technical depth: great resource
  - We tend to know lots of people in industry
  - Impress them technically:
    - Do well in class, ambitious projects
    - Independent studies
    - Work in their labs in the summer
  - May know people who would like to hire you...
    - ...and might put in a good word for you
  - Also, key to good letters if you want to go into PhD programs later

Soapbox #3

Programmer's Editors Emacs or VIM

Emacs and vim

- Learn and use a programmer's editor: **emacs** or **vim**
  - I use emacs
  - Feel free to try both, pick one and become an expert at it
  - Basic competence at the other is a plus
    - Open, edit, save, close
- Designed by programmers, for programmers

Programmer's editors: Emacs or vim

- "But I like <eclipse, visual studio, ….>"
  - That is only because you do not know the power of a real editor
  - Also, every job I've ever had except one I've used emacs
    - And my co-workers used emacs or vim

  - IDEs edit one (or a few) languages decently
    - Vim/emacs edit **everything** well:
      - C, C++, Java
      - SML
      - Scheme/LISP
      - Assembly
      - LaTeX
      - …

Programmer's Editors: continued

- Emacs and vim are also ubiquitous
  - Almost every Linux/UNIX system has both of them
  - My experience: Had to work on 1 AIX system with only vim
    - Wrote most of my code in emacs, scped it, and made minor edits
- They are also useable across remote connections
  - Graphical IDE over X-11 forwarding (or VNC) from halfway around the world? Soooo painful
  - Note: working on remote systems incredibly common in the real world
- So: my advice to you
  - Learn emacs and/or vim
  - Become an expert in ONE (you can't be a true expert in both): **muscle memory**

Keyboard + Muscle Memory

- GUIs are good for novice users
  - Menus are slow... but simple
- Keyboards and command lines: good for expert users
  - Learning curve
  - But worth it if you use often
  - I expect you all to become expert users
    - Of the command shell (bash)
    - Of a programmer's editor (emacs or vim)
- I can't stand having to use the mouse
  - Slows me down below speed of thought

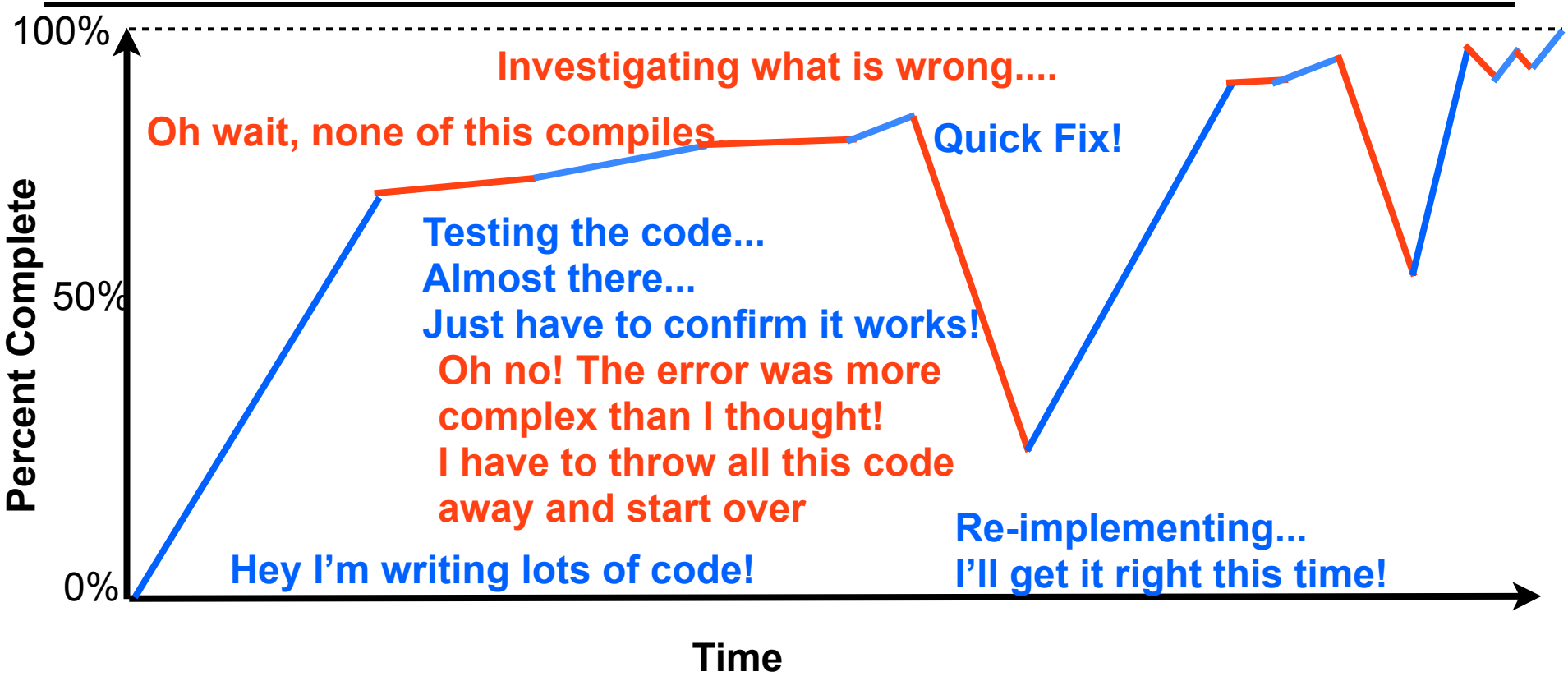
Emacs

- I use Emacs
  - And AoP has an Appendix on the basics
- Totally worth your time to master it
  - Broader lesson: master your tools. Boost your productivity

Soapbox #4

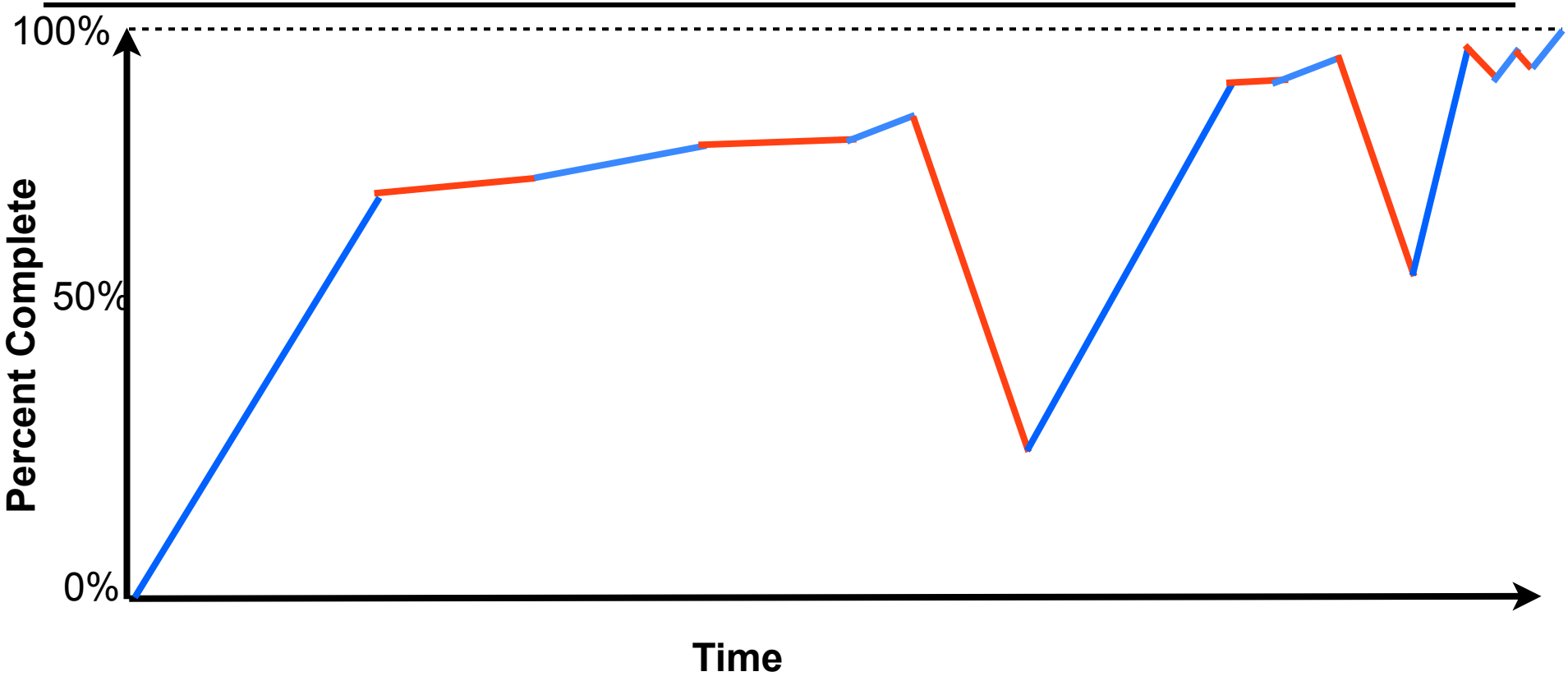
“I’m so close to finished”

Programmer's Time Estimates



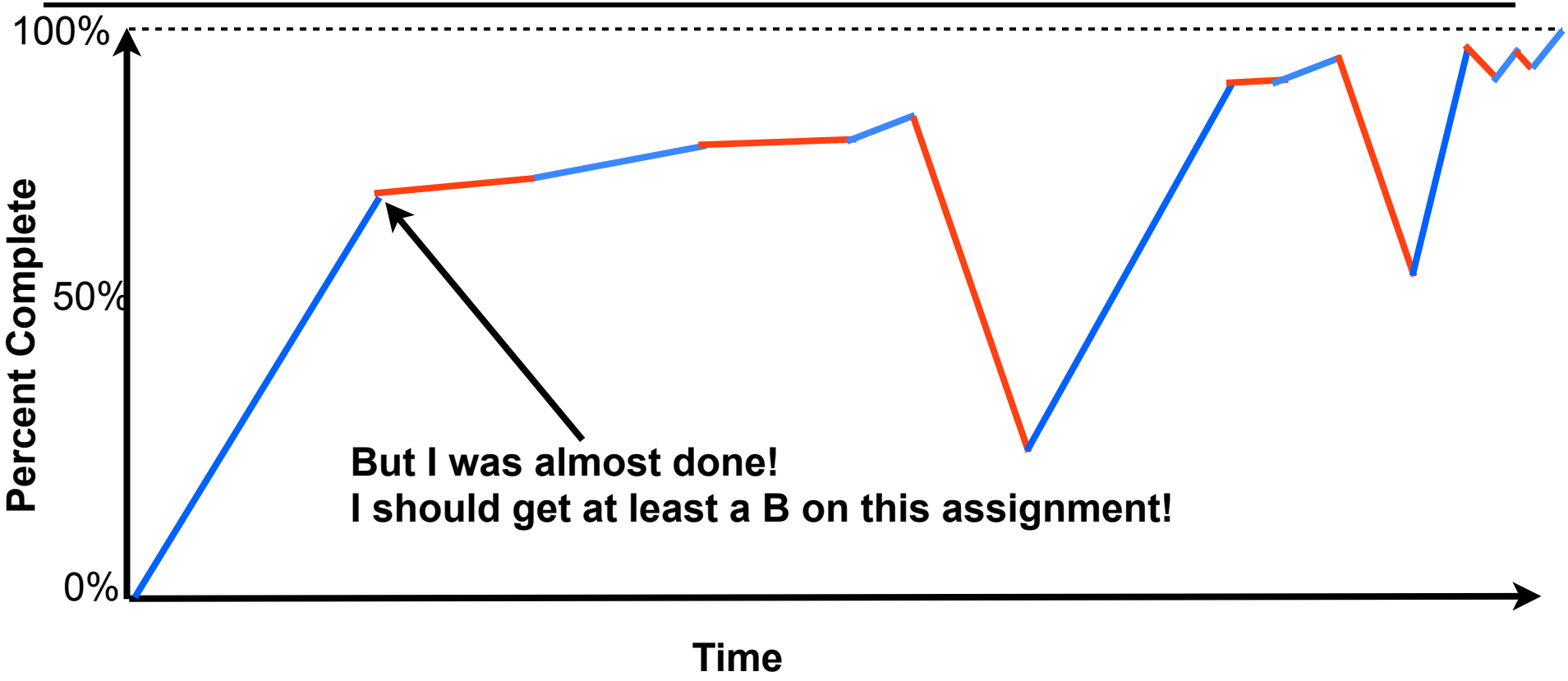
- Programmers often under-estimate time to completion
  - Let us take a look at "typical" programmer's thoughts on something.

Programmer's Time Estimates



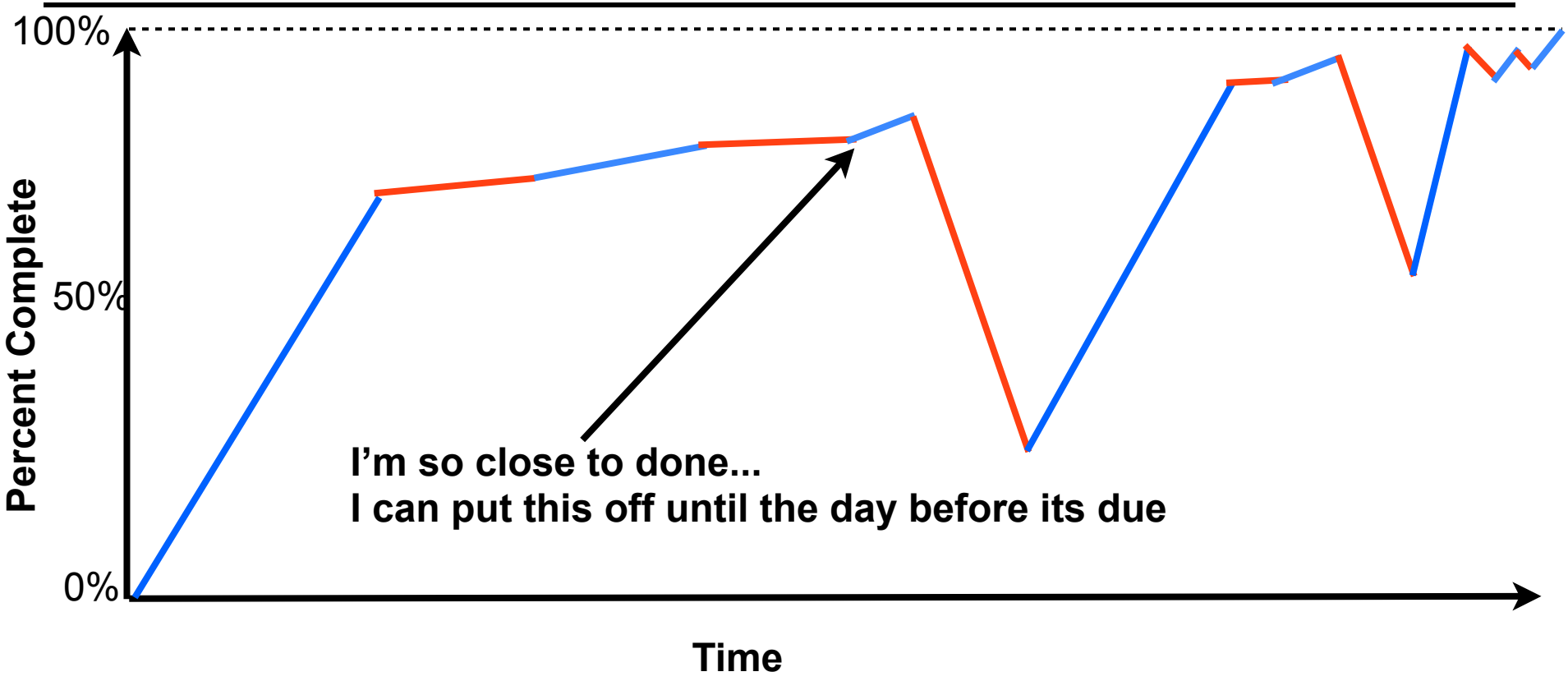
- Why do I bring this up now?

Programmer's Time Estimates



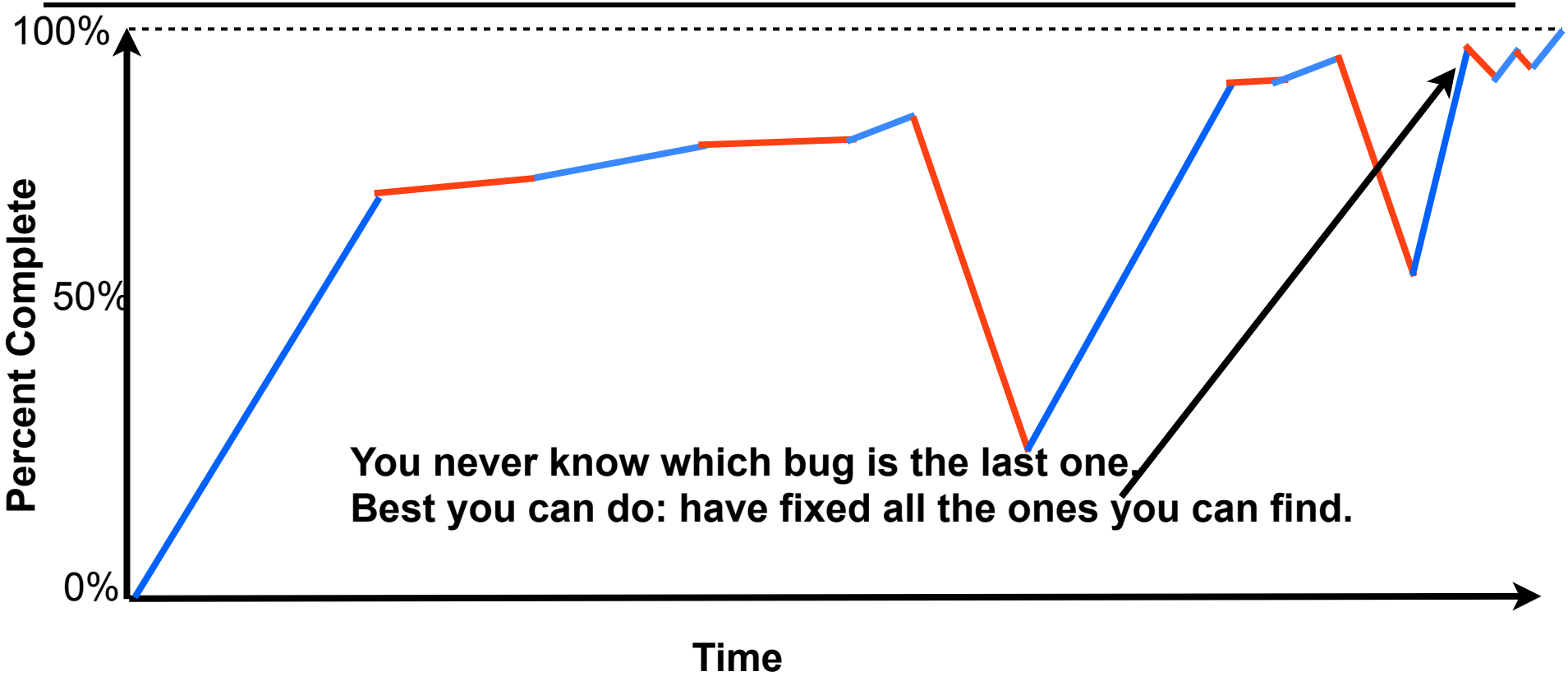
- Why do I bring this up now?
  - Writing some code is not "almost working"
    - Its not working until it works!

Programmer's Time Estimates



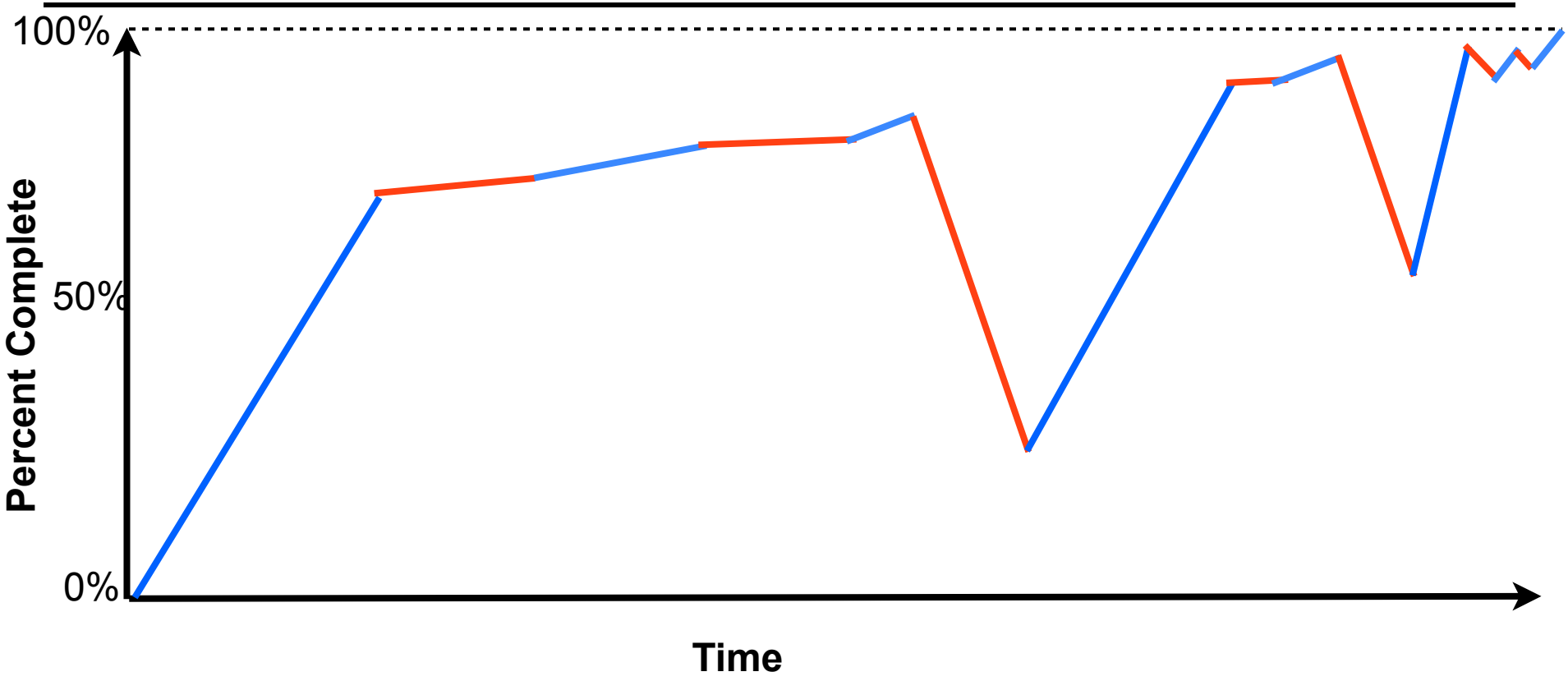
- Why do I bring this up now?
  - Don't wait until the last minute on programming assignments

Programmer's Time Estimates



- Why do I bring this up now?
  - Don't wait until the last minute on programming assignments
  - The last part takes the longest!

Programmer's Time Estimates



- Why do I bring this up now?
  - May matter in your professional lives!

Soapbox #5

The Poker Player's Fallacy

The Poker Player's Fallacy

- Poker: a card game (frequently with gambling)
- May have a seemingly good hand to start
  - ...and bet a lot of money
  - ...as more cards are revealed, hand may become bad
  - Novice players fall for "poker player's fallacy"
    - I've already put so much money in the pot, I may as well stay in...
    - ...its unlikely, but I **might** win...
  - Experienced/wise players: bail out if their hand is bad
    - "Cut your losses"---if you expect to lose, don't lose any more

...and programming?

- So what does this have to do with programming?
- More general statement:
  - Should evaluate "sticking with" something based on how good or bad it is
  - ...Not based on how much you have "put into it" before hand
    - Money or effort
    - Emotional attachment to what you have invested
- Programming:
  - This code is a horrible mess... but I've worked so hard on it!
  - I'll just keep trying to fix it...
  - Maybe better to scrap it and start over
    - Very hard to do: emotionally invested in your effort
    - Have to be cognizant of this fallacy
- Also often very true of writing (text)

Soapbox #6

Thoughts on Interviewing

My last soapbox: a bit on interviewing

- "Drew! I have a coding interview tomorrow. What is your advice?" **Be awesome.**
- Seriously, I'm not going to give you gimmicks or tricks
  - You either know your stuff
  - ...or you don't
  - Cramming for it: not likely to help
  - Think of sports championship
    - Training for months/years will matter

My last soapbox: a bit on interviewing

- What you know/can do is key, but...
- Two important factors to go with it:
  - Confidence:
    - Presenting yourself with confidence makes a big difference
  - How well you sell your abilities/accomplishments
    - How you describe what you have done/know may be as important as the skills/knowledge themselves
    - Let's imagine a couple different answers to an interview question such as "Tell me about a major class project"

Wrap Up Wednesday

- This concludes Wednesday's big picture "soapboxes"
  - Questions?

- For Friday Recitation:
  - Read Chapter 2, Appendix C.4
- For Monday,
  - Read Chapter 3,