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

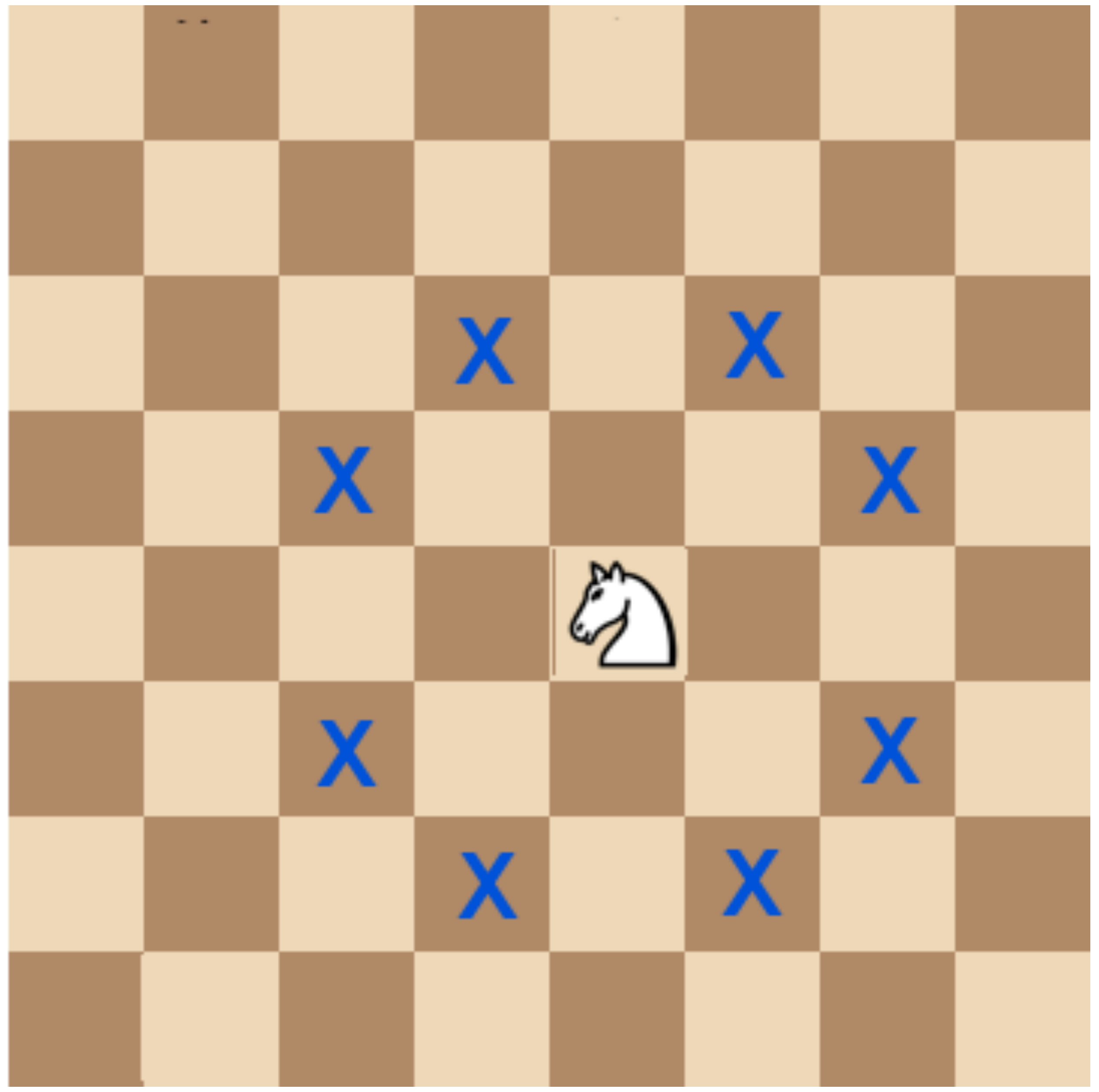
Let's highlight a couple of those points

- 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 to keep in mind:
  - Bridges and chess

Syntax vs…



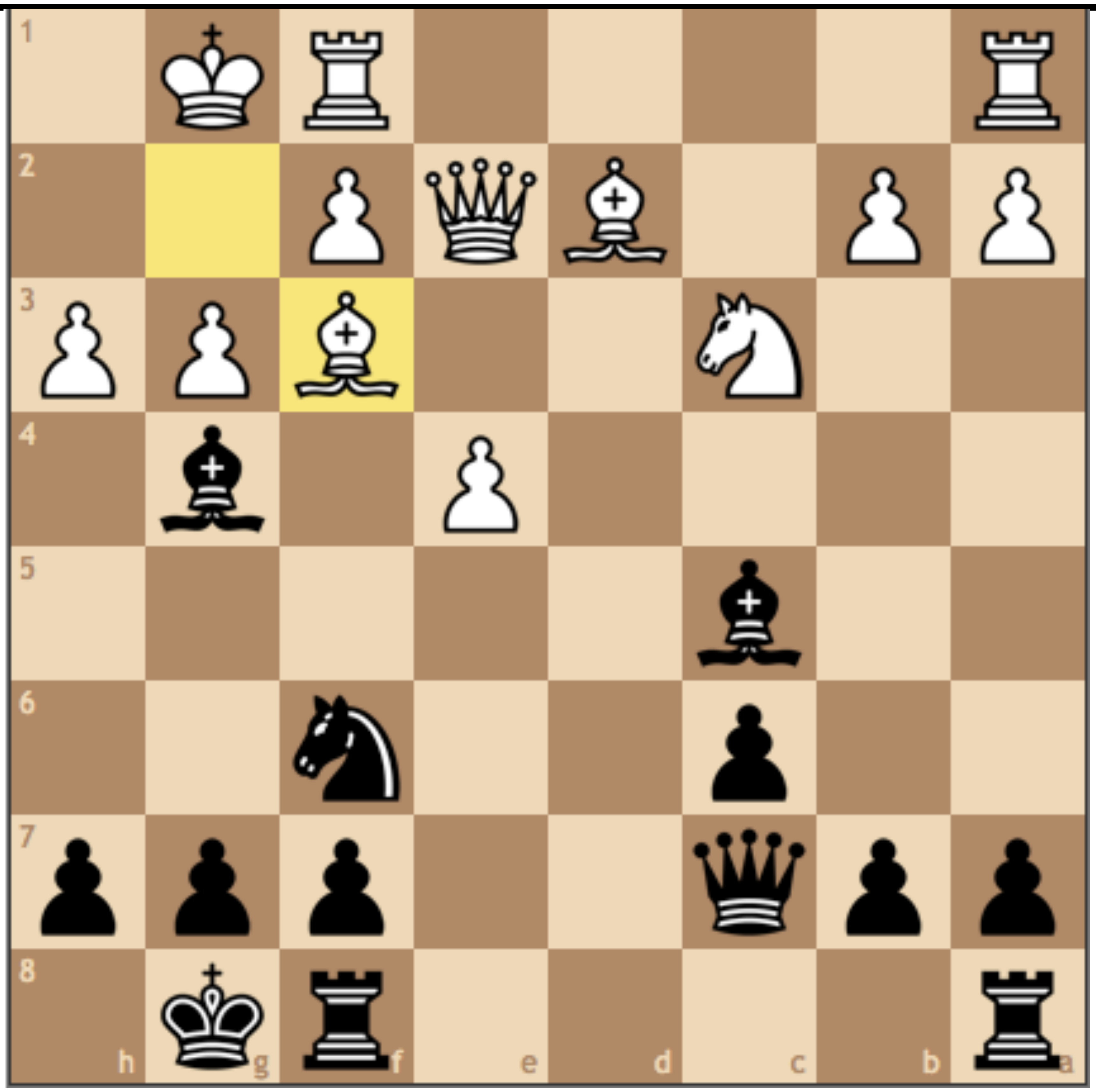
=

```
#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

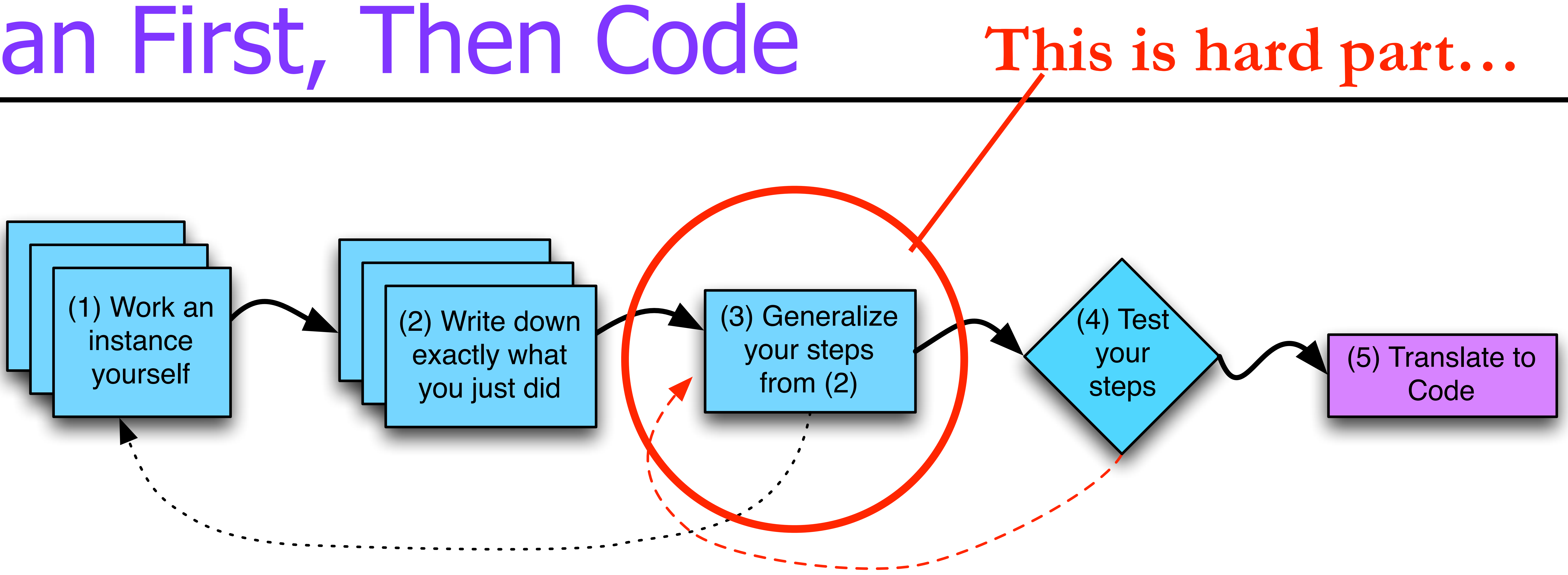
Bridges?

- Why are bridges a good analogy?

Bridges?

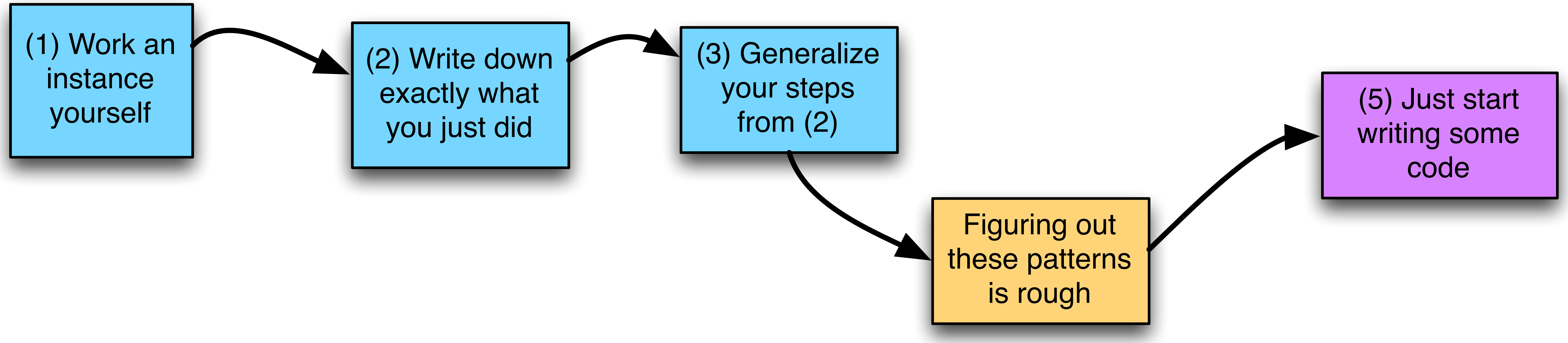
- Plan first, then build
  - Hard part
- Want to learn process, not memorize end results
- **Novices cannot learn from seeing a finished example**

Plan First, Then Code


This is hard part…
(1) Work an instance yourself
(2) Write down exactly what you just did
(3) Generalize your steps from (2)
(4) Test your steps
(5) Translate to 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


(1) Work an instance yourself
(2) Write down exactly what you just did
(3) Generalize your steps from (2)
Figuring out these patterns is rough
(5) Just start writing some code

- 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 we've learned
  - Cultural barriers to asking for help exist
- You can always ask one of us for help
  - Your success is important to us
  - We will make time for you
  - No question is too simple
  - We will not be offended if you did not learn something

Wisdom Continued

- Practice Exams
  - We post a practice midterm and a practice final
  - Use them!... and use them well...
  - We'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 we will give you a step-by-step approach to apply
    - (Use it!)
  - Working through it will help you learn
- Stuck? Ask one of us (or TAs)

Wisdom Continued

- Listen when we 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 we've been saying all along
- Listen to us, trust us, 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 intuition over time
  - Figure out semantics, …
- Learn more quickly if you let us teach you!