I am excited to be a part of expanding access to computer science education. Our world is becoming more computational, but, in many places, educational demand is outpacing supply. My department at the University of Washington has nearly doubled its enrollment in introductory classes since I entered graduate school—and we still turn away applicants for the major. I hope to expand on my experience as a teaching assistant to help build students' understanding of computer science.

On the first day of my freshman-year college physics lab, the professor announced the course's mission statement: "The enemy is B.S." (He did not use the acronym.) He explained that vague gaps in understanding that might work elsewhere were doomed in this lab. If the intensity results from our light-diffraction experiment differed slightly from the theory, there could be dozens of fascinating explanations—biased measurement equipment, or a secondary quantum effect—and any would be worth understanding and quantifying. But to just sweep any anomalies under the familiar rug of "human error" was B.S.: a crutch to help avoid a better understanding of physics. Learning something well means never being satisfied with half-explanations and cognitive shortcuts—there are always more questions to ask.

That professor shaped my attitude toward teaching and mentoring. To teach a concept, I need to cut through the shorthands and abstractions that I use unconsciously in day-to-day work. I need to see the idea as if for the first time, which entails excising any comfortable ambiguity—and a recognition of my own blind spots. I use the same teaching process to communicate my research, which means I take research talks seriously. My six conference talks have consistently been positively reviewed, and my preview talk at MICRO 2012 won the "best lightning talk" award.

I am particularly excited to teach the spectrum of computer systems topics—from architecture and operating systems to programming languages and software engineering—because they match this kind of rigorous, scientific style of teaching and learning so well. Concepts that seem arbitrary in compilers can often be explained through insight from languages or architecture: curiosity can carry students from one layer of system design to another. Systems topics represent a blend of theoretical insight and practical intuition, which can make them accessible to an especially broad range of students. I look forward to helping instill the combination of curiosity, creativity, and rigor that makes great computer scientists.

Mentoring. Working with undergraduate researchers has informed my intentions as an advisor. Advising well is incredibly difficult, and mentoring undergraduates—I've worked directly with six or seven—has been a learning process. Not every mentoring relationship has been successful, but I have learned from the unsuccessful ones: early challenges have taught me how to be a better mentor, and I am very proud of my latest collaborations with undergraduate researchers. I have learned three major lessons about advising:

- *Overcommunicate.* Nearly every problem I deal with as a mentor stems from a lapse in communication: of the student's expectations, of my research intentions, or of the student's level of technical comfort with a task. My best mentoring relationships have grown when I have been transparent and direct, even when transparency means being critical yet supportive. I have learned to err on the side of communicating more than seems intuitively necessary.
- *Carefully calibrate the amount of direction.* My instinct when I began advising undergraduates was to let students choose their own research plans—I was overly sensitive to the need for students to find work that matched their interests. I quickly learned that students need far more direct instructions when they are new to research: *build a prototype of this idea with feature X so that we can measure Y next week.* Direct instructions may be frustrating and limiting for more experienced students, but their absence is equally frustrating in the earlier research phases.
- *Scope the paper early.* In the division of labor between advisor and student, a major responsibility in the advisor's column is choosing *which* paper to write. Most research projects have a long list of exciting avenues to explore, but it is in everyone's interest to produce an end result with a finite scope. I have learned to take responsibility early for defining the paper we want to write. The scope inevitably shifts over time, but it is invaluable to have a goal in mind from beginning.

Underrepresented students. Diversity is a primary concern in advising and teaching. We all exhibit unconscious biases that can deny opportunities to underrepresented groups in computer science, including women and people of color. Advisors and instructors can play a crucial role in fighting against the sense of isolation that comes from with feeling different from most of one's classmates. It is imperative that we pay special attention to students from underrepresented groups to help mitigate entrenched inequalities in computer science.