While research is what initially drew me to computer science, I derive great pleasure from teaching and revel in moments when you see someone understand a concept for the first time. Rather then viewing teaching and research as rivals for my time, I experience them as highly synergistic activities. Research, which requires exploring alternative approaches and paradigms, helps refine and clarify the ideas we teach. In turn, teaching requires articulating ideas more simply, and the search for the clearest explanation often results deeper insights. It is because of this synergy that I am applying for academic positions at universities that combine research and teaching.

Teaching Experience As an undergraduate at Brown I taught extensively as a TA, covering classes on Operating Systems (CS167 and CS169), Distributed Systems (CS138), Computer Security (CS166) and Models of Computation (CS51). In addition to this TAing, I co-developed the course on computer security, since no such class had been offered by Brown previously, and also helped redesign the distributed systems course, which had not been offered for several semesters. I also tutored undergraduates in pre-calculus and introductory calculus courses, and briefly tutored a high-school student in pre-calculus.

As a graduate student at Berkeley I was a TA for undergraduate networking (CS168, previously known as EE122) for two semesters, where I taught a weekly section to approximately 20 students and was responsible for developing and grading a project.

Teaching Approach My own experiences as a student and my time TAing have led me to believe that students learn better when they can develop and test their own hypotheses, rather than merely be told the accepted wisdom. In a small classroom this might be achieved through the Socratic method where students have a chance to be engaged by questions, rather than to be drowned in pre-packaged answers, through dialog both among themselves and with the instructor. In addition, many topics in computer science, especially ones that I have been involved in teaching, revolve around building and examining artifacts (*e.g.,* operating systems, network applications, and distributed systems); allowing students time to design or interact with these artifacts can help them develop new hypotheses, and prove that the lessons they have learned apply to the real world.

While this approach – combining the ancient wisdom of the Socratic method with the modernity of real computational artifacts – may seem idealistic and impractical, I have tried this (albeit on a small scale) in my TAing at Brown and Berkeley. For most students, this appeared to work quite well, though it was hard for students who were uncomfortable participating in class. In the future I hope to develop a broader range of participatory modalities (*e.g.,* through allowing students to interact through online forums or anonymously), that would ensure this method is more inclusive.

Scaling this approach to larger classes is challenging, and addressing this is especially important given the growing enrollments in computer science. While I have not directly attempted to solve this problem, I have seen my advisor and others find creative ways – *e.g.,* through the use of real-time group design exercises – to accomplish similar goals. I am interested in trying this technique, and developing other scalable techniques that allow greater interaction and participation in large lectures.

As a student and TA, I have also found that projects provide a mechanism to teach skills that cannot otherwise be taught through classes. For instance, CS169 at Brown required students to develop a complete operating system in a little less than 14 weeks, and is widely regarded as one of the hardest undergraduate CS courses at Brown. This was a transformational course for me, introducing me to the material in more depth and intensity than any lecture course could have.

Finally, in my time as an undergraduate, I benefited greatly from courses that discussed ongoing research recent advances in particular areas. These courses were both useful as a way to learn about new results, but also provide a simple entry point into computer science research. For example, I started working on my undergraduate honors thesis [1] as a result of taking a graduate class on Constraint Satisfaction which used

simple problems (*e.g.,* graph coloring, NFL scheduling, etc.) to present then current research on addressing these problems.

Based on these experiences, I would hope to rely heavily on projects, and incorporate some recent reading, into any upper-level undergraduate course. I believe the combination of participation-oriented lectures, programming-oriented projects, and research-oriented readings would leave to a more enjoyable and rewarding classroom experience.

Courses I Can Teach I would be excited and interested in teaching courses on operating systems, distributed systems (both theory and practice), networking, and cloud computing. I can also teach introductory computer science courses, and basic material on programming languages and security.

References

[1] D. Heller, A. Panda, M. Sellmann, and J. Yip. Model Restarts for Structural Symmetry Breaking. In *CP*, 2008.