Teaching Statement

Riad S. Wahby

My approach to teaching, informed by my work as an engineer and by my experience teaching implementation-focused classes, relies heavily on experimentation. This means implementing and, even more importantly, debugging, which is arguably the most crucial basic skill that young engineers lack. It also means breaking things—especially in security and crypto classes!

My focus on experimentation dovetails naturally with a strong emphasis on the relationship between concepts and their applications. In an operating systems class, for example, teaching students how a semaphore works is important, but asking them to write code that uses a semaphore (say, to coordinate producer and consumer threads) is invaluable. Anecdotally, this emphasis on applications helps students both retain and build upon the material. More importantly, it keeps students interested—which is especially important to motivate those who are struggling because of inadequate preparation or under-developed study habits. I can sympathize: as an undergraduate, I suffered from both.

I also aim to engage students in office-hours conversations that go beyond lecture recaps and homework questions, encouraging students to identify the limits of current knowledge or to understand the influence of historical context on the material. In addition, my students appreciate the way I use course message boards to expand the discussion: my teaching reviews say that I give "well-explained, extremely complete answers," "insanely impressive detail," "excellent follow-on resources," and "some of the BEST Piazza responses I've ever seen."

Research papers are another important teaching tool, both to encourage student engagement and to prepare students for the academy and industry. Every computer scientist should know how to absorb and apply new research, and instilling the habit of reading papers in all students helps to broaden academic researchers' influence on the practice.

Finally, the most important element for every teacher is humility: admitting when something is not working, maintaining flexibility to accommodate a wide range of learning styles, and above all, remembering what it was once like to sit on the other side of the desk.

My teaching experience at Stanford includes TAing for Introduction to Computer Networking, Operating Systems, and Functional Systems in Haskell; I also plan to join the course staff for Topics in Cryptography this spring. At MIT, I was a TA for Feedback Systems, arguably the most difficult undergraduate course in the circuit design sequence.

I also have experience developing course material: in addition to serving as a TA for Introduction to Computer Networking, I worked with Professor Keith Winstein and Alex Ozdemir to completely rewrite the course's programming assignments. I created a wire-compatible, user-space TCP implementation (in C++17) that formed the backbone of the assignments; Alex undertook a major effort to define new abstractions and fine-tune organization for pedagogy.

I would love to teach courses covering standard topics in systems, security, and applied cryptography. In addition, my decade of experience as a circuit designer means that I am well-prepared to teach both introductory and advanced circuits classes to EE students. Beyond the standard courses in these areas, I imagine **developing and teaching several new classes**:

- *Building and breaking real-world crypto.* The demand for skilled crypto implementors is outstripping the supply of fresh talent. Meanwhile, accepted wisdom says, "don't implement crypto"—but the obvious danger in hewing too closely to this advice is that, in the limit, the only people implementing cryptography will be those brave or foolish enough to ignore it. Instead, we should teach the knowledge and skills that will lead to better foundations for secure systems. To this end, I envision developing a class that teaches students both how to implement security-critical systems and how to break them, covering real-world attack techniques and modern tools and methods for high-assurance software development. Most importantly, this class will impart a strong sense of how easily things go wrong.
- *Operating systems: from sand to syscall.* When I taught Operating Systems, I saw that many students suffered because of poor or nonexistent mental models for the underlying

hardware. In response, I would like to teach a sequence of classes in which students develop software and hardware in tandem. Students will start by designing a simple CPU, instantiating it on an FPGA, and writing a rudimentary operating system for it. Next, they will successively refine their designs, for example, by adding an MMU to their processor and then implementing virtual memory in their OS. Students will also learn important concepts from the systems literature by implementing those concepts on the platform they've built, inspired by Dawson Engler's CS 140e course at Stanford.

- *Control theory for computer scientists.* A problem I have repeatedly seen, both in networking courses and in real-world code, is that software systems frequently contain implicit feedback control loops that are given little or no thought. To address this, I will teach a course that applies classical control theory to software systems. Students will design and analyze continuous- and discrete-time feedback systems using standard tools like linear transforms and Nyquist plots; learn about important building blocks such as delay-locked loops; and apply their knowledge, for example, to control physical systems.

My mentoring experience includes work in both the academy and industry.

When I worked as an integrated circuit designer at Silicon Labs, I supervised interns and junior engineers. With them, I focused on communication and real-world circuit design, because they had scant practical experience and very little skill in communicating ideas or defending design choices in front of their colleagues.

I have also mentored several undergraduate students on research projects in computer science. Of particular note is Max Howald, whom Professor Michael Walfish and I recruited when he was an undergraduate at The Cooper Union, and who participated in research leading to multiple top-tier papers. I have also worked with undergraduates in Stanford's summer research program, and I am currently mentoring a student who is just starting her CS studies.

Finally, I have worked with junior graduate students, both formally and informally: I have mentored first-year PhD students as part of a Stanford program for improving diversity in CS, given feedback on countless practice talks, worked on research projects with junior students, and helped many students push papers towards deadlines by giving editorial feedback, helping develop an evaluation plan, giving implementation advice, or just writing text.

My approach to advising PhD students will focus on building the skills to succeed as researchers while encouraging students to strengthen their technical abilities.

This starts with acculturation to research and to the broader community: un-learning short-term focus on problem sets in favor of the time-management and self-motivation skills crucial for making progress on a long-term project; understanding the rhythm of a paper deadline and the responsibilities researchers have to their collaborators; knowing how to interact with other researchers; and learning how to review papers, interpret reviews, and respond to them.

Since research success depends crucially on choosing the right problem, I will help students to sharpen their own ideas, for example, by helping articulate research questions and standing in for potential reviewers. If a student asks for more guidance or even an explicit "assignment," I will happily take a more active role in problem selection while simultaneously helping them to build confidence and making clear that the goal is to become an independent researcher.

Finally, because I believe communication is by far the most important skill that students learn during a PhD, I will spend significant effort helping students to organize ideas, communicate clearly, write with empathy for the reader, and speak with empathy for the audience.

Teaching and advising are among the best ways for researchers to advance the practice of computer science in the academy and beyond. I eagerly anticipate the opportunity to do both.