In addition to research, a professor has three roles: teacher, mentor, and contributing member of a department. I strived to engage in each of these roles as a graduate student and will continue to do so as faculty member.

**Teaching** Teaching is both rewarding and challenging. My teaching experience stems from three sources, which inform my teaching philosophy: supervisor at the University of Cambridge (for Prolog, Bioinformatics, E-Commerce and Databases), teaching assistant at Cornell University for a class on Distributed Computing (Fall 2016, Fall 2017 and Fall 2018), and finally, course instructor for CS2110, Object-Oriented Programming and Data Structures (Cornell, Summer 2018) - a class consisting of approximately 90 students. I attempt to *support* students in acquiring the *knowledge* and *reasoning* necessary to become fully-fledged computer scientists. To this effect, I emphasise (*i*) communicating knowledge, not just information (*ii*) developing students' critical thinking as computer scientists (*iii*) inspiring student engagement.

(*i*) *Communicating knowledge* My goal is to train computer scientists through the courses that I teach. I explicitly separate the core reusable building blocks that underpin computer science from the specific engineering challenges that result from applying these principles. For instance, I restructured the CS2110 object-oriented programming lectures to emphasise the core principles of modularity, inheritance, encapsulation, and polymorphism. I highlighted the challenges/trade-offs made by different languages, whilst still delving into the specifics of one such instantiation, Java. This structure allowed students to think about design principles beyond the language, while still becoming confident Java programmers. Similarly, I emphasised links between algorithms seen in class with the explicit goal of extracting design strategies that could be re-applied elsewhere. For example, I explicitly highlighted that breadth-first search, Prim's algorithm, and Djisktra's algorithm all make use of *priority queues* and *relaxation* in similar ways. Finally, I attempted to highlight the limits of Big-O notion by showing students graphs that did not align with the theoretical bounds. This lead to an interesting discussion of low-level architectural concerns that one might need to take into account when designing datastructures.

(*ii*) *Developing CS thinking* Computer scientists must learn how to *approach* and *understand* new algorithms, not simply memorise them. I attempt to guide students in this process, by forcing them to think about the whys and the hows of every algorithm. As a supervisor in Cambridge and in my CS2110, I developed new problem sets that explicitly guided students towards developing an intuitive understanding of each taught concept. I asked students to remove lines in different algorithms and argue where the algorithm broke, or provide an intuitive explanation for specific steps in the algorithm. I asked, for instance, why finishing times must be sorted in reverse order in Kosaraju's algorithm for computing strongly connected components. This forced students to interact with the material in a way that they otherwise would not.

(*iii*) *Engagement* I convey to students that, regardless of their specific career objectives, each class they take is a small step towards becoming computer scientists and try to situate class material accordingly. In the CS2110 class for instance, I use the Java Collections API as motivation for the algorithms part of the course. Students do not learn hashmaps or red-black trees in the abstract, but instead, view these datastructures as "opening up" the box of the Java Collections API. Learning about datastructures and, more importantly, their trade-offs, then becomes part of a necessary "programmer's toolbox" that they need to acquire. Likewise, I often try to include small anecdotes about how and by whom algorithms were developed (ex: why red-black trees are red and black). I want students to feel like that they are part of a scientific endeavour to which they can soon make their contribution.

I often find that the easiest way to convince students to engage with the material is to engage with me. I want students to view me as a three-dimensional human being. Conversely, I hope to convey that I view students as individuals. To this end, I seek to personalise my teaching: I am honest about what I know and do not know. I find that live-coding, and the ensuing typos/bugs, helps students view me as a computer scientist in learning too, not a mere dispenser of facts. I try to draw anecdotes from past struggles, bugs, or my own research. When I find an algorithm truly beautiful, I do not shy away from saying so. I find that humour often helps: in my classes in the US, I occasionally use being British as a hook into a particular

problem. For instance, I motivated the Override notation in Java as catching inconsistencies between British and American spelling. The ensuing laughter often increases participation. I similarly encourage students to come to office hours over sending me emails, as I want to get to know them. I set as a first assignment in my CS2110 class "come talk to me in office hours, and tell me something about yourself". I found that over the duration of the semester, students became increasingly willing to ask questions about research, and other CS topics. I had a fantastic discussion with two students about bioinformatics and how the algorithms we saw in class could be extended to genetic sequencing. I find that this proximity is especially important for internal or URM students. International students often viewed attending office hours as admitting that they were struggling, which could reflect poorly on their grade. URM students disproportionately felt that coming to office hours was a waste of my time. Explicitly inviting them somewhat helped alleviate these beliefs.

**New courses** I would be able to teach general courses on databases, distributed systems, and, having taught this course in the past, object-oriented programming and datastructures. I hope to design courses emphasising the rationale behind large-scale system building. At the undergraduate level, I envision two upper-level division courses. A *System Principles* course would introduce students to the core abstractions that permeate systems, like batching, mutual exclusion, or modularity. While students would already be familiar with these concepts through their OS or networking courses, there is benefit to seeing these concepts synthesised across all areas. At the graduate level, I want to develop two classes. First, a class called *From theory to practice: how to disagree*. As an intern at MSR, I worked on an open-source project called Orleans, which temporarily violates mutual exclusion in the presence of cascading failures. This design choice seemed counter-intuitive for PhD interns like us, but was essential for the performance requirements of Microsoft's clients. Likewise, many database systems make decisions that appear odd from a research point of view. I would love to design a class that compares and explores the differing choices made by industry and academic state-of-the-art systems. Second, I would like to develop a *Languages for system building* class that would explore the impact of language design on system building, asking questions like: should a system be written in C, OCaml, Rust, Java or Go? Should one favour a type-safe, memory managed language?

**Group Structure** A successful PhD takes a village, and requires a dynamic and collaborative environment, both within and across research groups. I actively tried to foster such an environment as a graduate student, and hope to continue as faculty.

Within my own research group, I would prefer to structure projects in which one to two students collaborate together throughout their PhD, taking turns to lead projects. Companionship softens the challenges that comes with building large system projects, from debugging to paper rejections; shared responsibility encourages higher standards and discourages shortcuts. I believe that close collaboration also favours student development: students can build and learn from each others' strengths, and discussing ideas or implementation strategies in a way that advisors rarely have bandwidth for. I would favour a pipeline model in which younger students become familiar with the research process by helping out senior students (while making clear that they are an integral contributor to the project). This approach exposes new students to good design/experimental practices that senior students learn over time. It additionally allows students to observe how papers are written and how problems are motivated. I believe that these can tremendously help new students identify good problems to tackle in their own research. In the context of my OSDI'18 publication, for instance, I asked Sitar Harel to identify a medical application to benchmark our system. He not only conducted a large review of existing commercial medical record applications, but ported a full application to our framework and evaluated its performance.The graphs shown in the paper are the ones that he himself generated. Throughout this process, we had lengthy discussions about how to design sound experiments. Seeing Sitar go from having no research experience, to observing him confidently argue about application bottlenecks, is without doubt one of my most enjoyable graduate school experiences.

I similarly hope to encourage an environment in which students effectively make use of their peers. Throughout my PhD, I systematically verified experimental results and discussed design options with other students. It consequently felt natural to me to support others during paper deadlines when additional help was necessary to make a successful submission. I hope to replicate this as faculty. Likewise, I would like

my students to view me as a senior collaborator. I want my students to take ownership of their projects and understand that they, not me, will be the domain expert in the area. Fostering such an environment, much like for teaching, requires that all members of a research group interact regularly in both formal and informal contexts. I believe in regular group meetings in which students teach each other about papers or projects, supplemented by occasional informal meals. The former give students confidence that they are the expert on a topic, and can rely on each other to collaborate and learn. The latter can help alleviate barriers that may exist due to cultural or language differences . It also offers opportunities to informally discuss the "research process", rather than research itself. How/why do papers get accepted, old conference stories, are topics that I hope to regularly discuss with my students. I hope to make them feel part of a broader research community, not simply coders for their current system.

This collaborative environment should not be restricted to an individual advisor's group: students are an integral part of a research lab, department, and larger research community. I hope to foster an environment in which students across groups naturally discuss ideas and seek out peers for guidance and advice. Ethan Cechetti's advice on security was essential to the success of our OSDI'18 publication. I hope to encourage a systems community in which senior students feel responsible for helping younger students, through reading fellowship applications, introducing them to others at conferences, etc. This environment requires effort to setup: it comes through reading groups, casual "lunches", and by religiously encouraging students to collaborate and discuss ideas. At UT Austin and Cornell, I helped start a regular systems reading group. In these groups, I seek to foster an environment in which younger students feel confident to take an active part in discussions through careful moderation. Junior students too often feel shame in not knowing much about a particular area and consequently reluctant to participate in reading groups. To mitigate this issue, I intentionally ask questions in which I explicitly acknowledge my lack of background. In this way, I hope to give students the confidence to ask for clarifications and admit their own gaps in knowledge. This is especially relevant for students who lack the language skills or confidence to express themselves, or who are unwilling to challenge more senior figures: I often individually invite people to speak, and will regularly cycle back to students who had been interrupted to make sure that they are heard.