I am pursuing a faculty position because I believe that mentorship of new researchers and engineers is an order of magnitude more impactful than technical innovations alone. I also find personal happiness in nurturing meaningful relationships with bright and motivated people.

My teaching process is centered around providing three key experiences to students: (i) systematization of the learning process, (ii) consistent and frequent opportunities to meet one-on-one, and (iii) contextualization of knowledge. My experience from 8 semesters of assistant teaching across 3 courses and 2 institutions, mentoring 6 undergraduate and 2 graduate students, and 8 semesters of coaching a university club sports team has informed this process.

In the remainder of this statement, I describe my vision for teaching as a faculty member. I view faculty teaching responsibilities as falling under two broad, equally important categories: mentorship for students pursuing careers in research and curricular education for students broadly pursuing careers that require computational thinking.

1 Mentorship

Graduate students. My goal when mentoring a researcher is for them to be proficient at recognizing, articulating, and tackling open-ended technical problems. I believe an effective method for achieving this goal is to break down the end-to-end research process into initially concrete technical tasks that progressively require the development of more open-ended skills. This method is akin to how athletes drill individual movements and contrived scenarios. The mastery of these narrowly scoped skills allows for their synthesis during full-fledged competition.

Accordingly, my process for mentoring a new distributed systems researcher starts with developing concrete skills like reviewing related work, coding real systems, and running experiments that measure system behaviors. I plan to provide concrete, well-scoped project seeds to my junior students that initially focus on practicing these skills without delving into more open-ended exploration. I view this as systemization of learning the research process because it defines a framework for students to understand their progress over time. I expect that my senior students will also carve out components of their research projects as starter projects on which to collaborate with the more junior students. I also plan to encourage my students to take a course on distributed systems; a well-structured course provides both an overview of related work in this area and opportunities to code real systems.

To aid my students in progressing from technical proficiency to charting their own research directions, I plan to consistently and frequently hold one-on-one meetings. In my experience, the unstructured and interactive nature of one-on-one meetings with my research advisors allowed me to practice defending my ideas and question existing lines of reasoning - actions that would not have been effective in a larger group setting (e.g., meeting with collaborators) or by myself.

My overarching goal as a mentor is to be viewed more as a collaborator than as a project leader. I intend to augment my students' technical mastery of their projects with my own narrow slice of technical expertise and general experience in seeing research projects from inception to completion. As a graduate student, I worked with two other graduate students in this capacity. At Cornell, I worked with Florian Suri-Payer on the project he led designing a BFT database. Florian is an expert in this design space, so my contribution as a collaborator was to work with him on the implementation and evaluation of the system. Beyond coding together, I taught Florian about prototyping systems in a way amenable to obtaining valid performance measurements. I also introduced Florian to the iterative process of designing experiments, mentally modelling the expected results, and running them on real hardware. I collaborated with Jeffrey Helt from

Princeton in a similar way on his project designing systems for a new strong consistency model. I hope to emulate these collaborative experiences with my future students

Undergraduate students. I also look forward to mentoring undergraduates interested in pursuing research careers. Coming from a family with non-technical and non-academic members, I credit my current career path primarily to the time and energy that my undergraduate research advisor, Wyatt Lloyd, dedicated to integrating me into his research group. I believe the direct mentorship of undergraduates—an experience I am fortunate to have had—is a necessary step to achieving greater diversity in the research community.

My process for mentoring undergraduates in distributed systems research is similar to my process for graduate students. I believe that starting undergraduates with concrete, well-scoped research tasks and providing ample one-on-one face time is effective. This is especially true for undergraduates that are simultaneously taking a full courseload and working on other extracurricular pursuits. I have experience working with several undergraduates, two instances of which I highlight as illustrative of my mentorship process. Audrey Cheng, an undergraduate at Princeton, worked with me on Gryff [1] by designing, implementing, and evaluating a fast write optimization for the system. During this process, I met with Audrey frequently to guide her through the process of implementing and evaluating her design. Audrey is now a PhD student at UC Berkeley working on distributed databases. Janice Chan is an undergraduate from Cornell that expressed interest to me in exploring research after taking Cornell's distributed systems course. We met weekly over the course of a semester as Janice worked to understand how the performance of transactional systems varies with contention. I guided Janice through this process, culminating in her implementing Smallbank, a traditional transactional workload, in a real codebase to measure its performance across several systems. Janice is now a software engineer at Google working on distributed systems.

Research group. I believe collaboration with a group of likeminded researchers is integral to the development of a researcher. To encourage this among my students, I plan to hold both formal and informal group meetings. The primary purpose of formal group meetings will be to disseminate knowledge of students' progress, challenges, and solutions regarding their projects. I also intend for students to practice non-technical research skills at group meetings like presenting work, networking with colleagues, and reviewing papers. Informal group meetings, such as lunches or day trips, will aim to encourage support between students and will offer unstructured opportunities for students to learn the "hidden curriculum" of a career in research.

In addition, I plan to maintain a shared prototyping and experimental framework that my students can use for their projects. This reduces repeated work across students and also serves as a guide for good practices for reproducibility. My time volunteering on the Artifact Evaluation Committees for OSDI'20 and SOSP'21 (and the Artifact Award Committee for SOSP'21) emboldened my belief that systems research needs to be done reproducibly.

2 Course Instruction

I am well prepared to teach courses on distributed systems, operating systems, data structures, and object oriented design. My approach to teaching these courses would focus on (i) using interactive reasoning to bridge the gap between human and computational thinking and (ii) contextualizing knowledge for students' diverse career paths. These focuses are informed by my experience as a graduate teaching assistant for one semester of Distributed Computing and one semester of Operating Systems at Cornell and as an undergraduate teaching assistant for six

semesters of Data Structures & Object Oriented Designed at USC.

Interactive, hands-on exploration. I believe a powerful pedagogical tool in a computer science educator's workbelt is interactive reasoning. While computer scientists eventually become "conversational" in propositional logic, algorithmic reasoning, and programming languages, it is difficult for new students to think like a computer. I first observed the power of interactivity in my Data Structures office hours at USC. I often met with students who had written out mostly correct programs that only failed in a few cases. I taught these students how to use an interactive debugger to step through their code and examine program state. This consistently led to students identifying the bugs in their program without direct assistance from me.

The interactive debugging approach is practical for beginners to adopt when working on single-process, single-threaded programs like those written for introductory courses. Thanks to recent work from Ellis Michael at the University of Washington [4], this approach can also be applied to distributed systems by providing an interactive model checking tool to students. I drove the adoption of Ellis' open source distributed systems assignments for Cornell's Distributed Computing course and found that students were able to identify and understand tricky distributed executions of their own code after only minor guidance from me on how to use the interactive model checker. I am eager to continue using Ellis' tool for teaching distributed systems and to adopt similar interactive tools for other classes that I teach (e.g., Harmony [3], an interactive model checking tool developed at Cornell for teaching concurrent programming).

Contextualizing curricula. Students have a range of career aspirations, from researchers to engineers to professions outside of the technical scope of computer science. I believe computer science courses should be tailored to these diverse paths in order to situate concepts in a relevant and meaningful context for students. For example, a distributed systems course should teach that consensus is impossible in an asynchronous environment with faulty nodes [2]. On the one hand, students with research aspirations should learn the formal methods used to prove such a result. On the other hand, students planning for an engineering career should learn the practical implications of this result and the consequent challenges in deploying consensus systems. In some cases, practical and principled knowledge are inextricably linked, and it benefits a student, regardless of their goals, to understand both sides of the equation. Thus, contextualization of course materials needs to be prudently applied at both the course design level (e.g., "Distributed Systems Principles" vs. "Engineering Large Scale Systems") and the individual level (e.g., varying approaches during office hours, providing options for completing assignments).

Coaching experience. A considerable amount of my teaching and mentorship experience comes from my time as the head coach of the club men's ultimate frisbee team at Cornell from Fall 2018 to Spring 2022. Though this experience is outside of the scope of research mentorship and academic education, I nevertheless developed highly applicable skills. My primary responsibility as coach was to "give lessons" (run 2 hour practices) 2-3 times per week for each week across each semester. Over the years I developed a curriculum for the Cornell team that I would dynamically draw from to plan practices on a week-to-week basis. I also had the responsibility of providing individualized feedback to players, both in and out of practice. I found that making myself available to players for one-on-one "office hours" aided the feedback process because players could ask very individualized questions about concepts that I introduced in the larger practice environment. These office hours also allowed me to connect with players and establish meaningful relationships. Because I care deeply for both teaching and ultimate frisbee, I would be happy to again be involved with coaching in the campus community as a faculty member.

References

[1] Matthew Burke, Audrey Cheng, and Wyatt Lloyd. Gryff: Unifying Consensus and Shared Registers. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2020.

[2] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[3] `https://harmony.cs.cornell.edu/`, 2022.

[4] Ellis Michael, Doug Woos, Thomas Anderson, Michael D. Ernst, and Zachary Tatlock. Teaching Rigorous Distributed Systems With Efficient Model Checking. In *ACM SIGOPS European Conference on Computer Systems (EuroSys)*, 2019.