My goal in teaching is to challenge students intellectually and make the classroom inclusive to students with diverse backgrounds and learning styles. In addition to teaching undergraduates at all levels of interaction, I have run hands-on-activities for K-8 students, tutored high school students who are deaf or hard of hearing, and instructed software professionals and mountaineers. As a teaching assistant and instructor, I’ve learned how to teach core computer systems courses more effectively. My teaching experiences have also taught me the importance of building trust and rapport with students.

How I teach systems in Computer Science & Engineering (CSE)

From my experience teaching CSE courses, I’ve developed strategies for creating effective assignments and discussion activities. Two skills I emphasize are experimentation and technical communication.

When I teach a computer architecture discussion section, I first have students organize the new topics by applying them to new scenarios. One way I do this is by developing exercises on 2D grids, a template I learned from Berkeley professor Krste Asanovic. The x-axis was an architectural change from the week’s topics (e.g., wider microcode, narrower cache lines) and the y-axis are relevant engineering metrics (e.g., clock frequency, miss rate). To fill in a grid cell, students must understand what the new design changes are and then infer their implications. One student told me in office hours that “[the grids] help me view the new information from lecture”; several times a row or column ignited interesting debate. I hope that the high-level nature of these exercises reinforces intuitions and encourages the strategy of forming hypotheses for proposed architectural designs.

I want programming assignments to challenge students without misusing their time. I believe that students in systems courses learn the most from programming projects; however, projects are time consuming and students need sufficient guidance to make their effort well-spent. To increase the ratio of student learning to time, I observe how students spend their time. When I assigned a “memory allocator” project to my non-major “Programming Tools” students, I learned in office hours that many students had spent hours tracking down memory faults without a clear strategy. I have revised the assignment to incorporate debugging practices. For example, the assignment now requires students to write a “check_heap” procedure for their C data structure, which makes it more likely that crashes related to memory corruption happen near the root cause.

I teach my students that experimentation is an effective learning and problem solving strategy in computer science. This idea comes to life in programming, where you can get immediate feedback as often as you demand it. To encourage experimentation, I set an example—one that I learned from Berkeley senior lecturer Brian Harvey. When live-programming in “Programming Tools” lecture, I answered questions of “what if X?” with “what program can we write to find out?” I posted the programming log from lecture so students can revisit it. In the teaching evaluations, students appreciated the examples and that they were posted. As much as I value experimentation, I cannot assume experimentation works the same for everyone. Students have a variety of preferred learning modes, so I strive to find new ways of including everyone. There is a body of literature to draw on for creating an inclusive learning environment in CSE.

To give students the opportunity to experiment, I need to know when to step back. When introducing high school students to programming in the Saturday Computing Experience, I almost always worked 1-on-1 with students. This patently fortunate ratio tempted me to be very hands-on, so the student was more likely to immediately ask for help than to experiment. During the course of the summer, I gradually tuned my method of interference to balance learning and frustration.

I plan to include speaking and writing practice in the technical courses I teach. When I TAed advanced computer architecture, we split the project into two parts so that students received feedback on their proposal; writing was also 1/3 of the grading rubric. I think that many computer science and engineering students are given too few opportunities to exercise technical communication (the one required semester of techcomm in my undergraduate program was cut right after I took it). Two of the most important skills that I have improved the most in graduate school are technical speaking and technical writing. The two research labs I am a member
of have a strong culture of vigilantly improving our oral presentations. Student talks are the result of at least two peer practice talks and faculty input. I find as many opportunities as I can to read the writing drafts of peers.

**Trust and rapport**

One moment where I first realized the value of trust in the classroom happened in 2012 when I had 30 minutes to teach 5th graders what programming was. One of the students instructed me—a receptive and obtuse robot—that spreading jelly on my pressed shirt was the next crucial step in making PB&J. I had just suggested to them that computers will thoughtlessly follow simple instructions, and they were starting to get it. Breaking character would betray their trust. My quick glance toward the parents in the back of the room indicated that they were empathetic yet equally helpless amidst a class of 30 energetic 10-year-olds. I got lucky: the mob deemed my “does not compute” as acceptable with their laughter.

I’ve not always gained trust on the first try. I practiced writing exams as a TA by helping my instructors, but when I finally assumed the role of instructor, my first midterm was bumpy. Even after piloting the midterm with my 4 TAs, I misjudged either the time it would take new students or how they would feel about an exam that was too long. Although I assured my students were “in the same boat”, they remained frustrated by the inability to finish much of the exam. I took this to heart, and the final exam I wrote was manageable in the allotted time. The spread of grades told me the final was challenging, yet when the students left the exam, this time I saw smiles and greetings. I had written a test that allowed them to show me what they knew.

I rate lectures where I have riveting interaction from 2 very enthusiastic students as “needs improvement”. Assessing student engagement and comprehension by the loudest voices would diminish the trust that I am present for every student. I recall a talk that Harvey Mudd President Maria Klawe gave at UW, where she said that giving dominating students interaction outside of the classroom dramatically improved the environment inside the classroom. One way that I engage every willing student in a class of 80 is to break up lecture with 1-minute comprehension and application questions that they discuss with their neighbor. For classes where there are students who resist interrupting with questions yet ask me questions later, I’d like to try something I saw recently in a research panel: real-time crowdsourcing of popular—optionally anonymous—questions using a tool like Twitter.

I have also learned to engage experienced students by acknowledging what they bring to the classroom. I am a volunteer instructor for climbing courses offered by the Mountaineers. Matt P., a spectacular climber and high school science teacher, summarized his attitude for teaching adults in three points: 1) acknowledge/ask what your students know, 2) give your students space for learning, and 3) do not be a sage. Following this advice, I find that students are cheerful, friendly, engaged, and satisfied. I have noted when other instructors fail to apply this advice and then face disengagement. To provide busy professional Master’s students with extra time to share their ideas, Cray Principal Engineer Brad Chamberlain and I held “happy office hours” after after the weekly evening class. Sometimes we talked shop and other times we talked about coffee. Happy office hours received positive reviews and attracted regulars.

**My teaching plans**

I am excited to continue to teach computer systems courses across the stack, from digital logic up to programming languages and advanced courses in architecture, parallel computing, programming languages, compilers, and data management. While the fundamentals of these subareas have not changed much since I was an undergraduate, their contexts have. Architecture is not just about instruction level parallelism, new parallel languages abound, clouds provide accessible compute resources, and there is as much interest in program analysis for security, distributed coordination, and concurrency as for performance. In an intro to data management course, I will give students hands-on experience with at least one data processing system that is not a conventional database. I’d like to create a course “big data systems” that combines elements of databases, distributed systems, security, and algorithms. A special topics course I’d like to teach is “implementation of
data management systems on modern architectures”.

Although I have not yet tried “flipping the classroom” with pre-recorded material myself, I would like to experiment with adding it to lecture-based courses. My interest came from seeing the instructors for “Hardware/software interface” add it with success and the support offered by UW Center for Teaching and Learning for instructors who want to start flipping their classroom. Another form of flipping I’m inspired by, which I found enjoyable as an undergraduate, is the option of lab-based versions of CS1/CS2/CS3, where there is 1 hour of lecture and 6 hours of lab per week. I plan to stay informed of developments in education research that are specific to science and engineering.