Our interconnected world is increasingly reliant on distributed systems of unprecedented scale, serving applications which must share state across the globe. And, despite decades of research, we're still not sure how to program them! Modern global scale stretches existing abstractions—such as shared memory concurrency or transactional data stores—to the breaking point, pushing the task of successfully building against these abstractions ever deeper into the domain of experts. **My research focuses on building new systems and designing new language abstractions to make programming at scale feasible for more people, without sacrificing performance, correctness, or expressive power in the process**. In my work, I'm tackling distributed systems programming as both a programming languages *and* a systems problem: improving programmability of existing abstractions on one hand, and improving the scale at which we can offer these abstractions on the other. Truly addressing modern scale demands work *at the intersection* of these fields, requiring co-developed languages and systems that allow programmers to keep simple programming models while allowing distributed systems to operate at speed, without frequent coordination.

Distributed Systems: In core distributed systems, I show that new systems designs inspired by programming languages concepts can scale consistent replication to supercomputer speeds. With Derecho [4, 13], we push back against the notion that consistent replication can't keep up with modern data rates—while in turn taking inspiration from weakly consistent replication. My key insight was to rebuild Derecho's core message delivery protocol around monotonic observations, eliminating the need for blocking or locking on the critical path. Our results show that Derecho scales to the full speed of even infiniband networks, while exposing a convenient, zero-overhead API centered around replicated distributed actors.

Language design: Simply offering consistent replication is not enough. Existing consistent and concurrent abstractions—like shared memory—are made challenging by the potential for destructive data races. Yet shared memory is essential in real-world concurrent programs. In Fearless Concurrency [10], I present a type system that **provably eliminates** destructive data races in concurrent code without placing harsh limits on expressive power, ensuring programmers never need to debug a race on memory again. Crucially, my system is deployable in any C/Java-family language; in fact, it is currently under evaluation by Apple for inclusion into Swift.

While consistent replication is achievable at datacenter scale, it is more challenging to offer at global scale. Thus, many global-scale distributed systems continue to offer various forms of *weak consistency*, resulting in a zoo of confusing guarantees for users to navigate. Applications in this space must reason not just about a single consistency model, but often multiple overlapping consistency models simultaneously. Here, mechanisms that eliminate data races—like fearless concurrency or traditional transactions mechanisms—are not sufficient. In MixT [8], I take inspiration from the security literature to build an information-flow type system that ensures weak consistency is used responsibly. My approach successfully catches subtle bugs in example code, while remaining directly deployable atop existing transactional data stores.

Research at the intersection: MixT, Derecho, and Fearless Concurrency all stop at the same abstraction boundary: shared storage. While this abstraction offers an intuitive programming model, it makes a poor system interface; shared storage systems need to ensure consistency for *all possible uses* of the state they expose, often offering complex guarantees mismatched to what an individual application requires. Overcoming this limitation requires an integrated approach: we must design the language and system *together*. In so doing, we can build systems that adapt to the expressed needs of applications, paying the cost of strong consistency only when required for correctness. I demonstrate this principle by expanding on the key idea from Derecho—monotonic observations—in two projects: Hydro and Gallifrey. In Gallifrey [9, 7], I build a programming model in which programmers interact with shared, mutable, *multi-monotonic* objects. Despite offering strong consistency and race-freedom, Gallifrey can replicate monotonic objects under the weakest convergent consistency model—eventual consistency—with no risk to application correctness, requiring synchronization only for non-monotonic actions. In the wider Hydro project [2, 6], we combine Gallifrey's approach with insights from databases and security, finding new programming models that capture monotonicity without traditional shared memory or replicated objects. These programming models raise questions for future systems and languages research, which I am eager to explore.

Scaling Consistent Replication with Derecho

Derecho is a library for building high-performance distributed code, supporting a *consistently-replicated actor* framework that Derecho calls "subgroups and shards." This abstraction is powerful and intuitive: users express programs as a group of services (or actors), each of which can be invoked via an external remote procedure call (RPC) and may in turn issue RPCs to other services within the system. The abstraction of a communicating replicated actor is difficult to offer with both low complexity and good performance. The main challenge is to ensure fault-tolerant, consistent replication at run-time. Modern industry-strength solutions tend to sidestep this challenge, often avoiding system-provided replication [14, 11] or weakening system-offered consistency [1]. With Derecho, we demonstrated that replication is possible at scale by rebuilding atomic multicast from the ground up, leveraging the emerging transport technology of remote direct memory access (RDMA) and rethinking traditional protocols in terms of RDMA-synchronized state. In particular, I rephrased virtual synchrony (a protocol for distributed consensus) in terms of loosely synchronized replicated tables, containing *monotonically growing* shared state. Boolean queries over monotonic state are stable: once they have returned true, they will *always* do so—even in the presence of racy, uncoordinated state updates. By phrasing protocols in terms of monotonic observations, I can safely eliminate blocking protocol steps and client-level locking—without sacrificing safety or liveness, and without weakening semantic guarantees. The resulting protocol achieves **strongly consistent replication among 16 replicas at 10GB/s**, nearly the line rate of our test cluster.

Replicated actors ensure that Derecho's insights reach the real world—in which not everything is monotonic. The API I built allows programmers to specify services as C++ classes, choose replication and resilience factors, and issue reliable RPCs between various services—all using the same syntax and tooling available for normal object oriented C++ code. While RPC among actors is common among industry-strength solutions[1, 14, 11], they often come with significant—and (according to their developers) unavoidable—run-time overhead. With Derecho, I show that through a combination of compile-time code and careful engineering, one can provide these abstractions with **no measurable overhead**.

Bringing Fearless Concurrency to Usable Languages

Destructive data races in shared-memory concurrent programs have long been recognized as a challenge for programmers. **My type system for fearless concurrency [10] proves the absence of destructive races at compile time**, eliminating the need to monitor for, debug, or defend against such races at run time. The key idea presented in my PLDI paper [10] is to statically divide the heap into a tree of *regions*, which may be exchanged between threads. Objects may reference each other within the same region, or across region boundaries; the only requirement, enforced at function boundaries, is that cross-region links point to an *otherwise-unreachable* object subgraph. My type system deploys this invariant with zero run-time overhead and low user-exposed complexity: users need only annotate fields that may cross regions, leaving the rest to decidable type inference. For power users, I include the ability to describe complex region relationships, and feature a dynamic escape-hatch that is both easy to use and does not threaten correctness. Crucially, this system design is real-world ready: it can provide fearless concurrency in any C/Java family language. In reviewing my work for inclusion into Swift, the Swift language team specifically highlighted my system's ability to **represent common patterns with intuitive code**, its **low surface complexity**, its support for **separate compilation**, and its **extensibility** as **key improvements on Rust** and other competing industrial and academic languages.

In work currently under submission to PLDI, we take a radically different approach—exposing a *fully immutable* functional programming model to the user, while addressing the resulting overheads at compile-time. In this project, I solo-advised a team of four undergraduate students to craft a paper on lambda set specialization, a technique that eliminates the overhead of lambdas by statically determining all possible function targets of each call site and generating call-site-specialized versions of those functions. This project spans from **formal type system results** (complete with mechanization) to performance evaluations, demonstrating an **up to 6x improvement in higher-order functional ML code**.

Mixing weak and strong consistency with MixT

At global scale, replicated datastores expose APIs with varying *weak consistency* guarantees, effectively exposing a trade-off between stronger consistency on the one hand and better performance on the other. However, this trade-off cannot be made once and for all; different data will require different consistency guarantees within one application—or even within one transaction. With MixT [8], we propose a model for *mixed-consistency* transactions, allowing programmers to safely manipulate data at various consistency models, all within the same transaction. We give a definition of safety for mixed-consistency transactions based on *noninterference*: changes to data stored at one consistency model should never affect data stored in a stronger model. This semantic condition captures not just direct influence—like copying a weakly consistent value into a strongly consistent store—but *all* forms of influence, including control flow. This condition *also* yields a transaction-splitting strategy; if data at weaker consistency cannot influence data at stronger, then we are *guaranteed* to be able to split a transaction by consistency model. Using this strategy, MixT is able to implement a *single* mixed-consistency transaction in terms of *multiple* single-consistency transactions, executed on any number of traditional data stores. Unlike similar type systems, MixT's transactions do not require explicit type annotations; instead, all annotations are inferred. Our results show that using mixed-consistency transactions can **significantly improve performance** vs. executing all transactions at only a single sufficiently-strong consistency model.

Building at the intersection with Gallifrey and Hydro

The domain of large-scale geo-distributed systems programming is fertile ground for solutions that blend systems and programming languages insights. In the ongoing Gallifrey project [9], I'm looking for ways that programmers can build correct programs against simple, consistent abstractions—despite those programs being deployed in a globally-replicated, weakly-consistent ecosystem. In Derecho's core protocols, I leveraged a simple design principle: monotonic observations of ever-growing states. In Gallifrey, I built an entire programming model based around the idea of sharing *multi*-monotonic objects. The model is easy to explain: programmers write normal, sequential, object-oriented code—without worrying about race conditions or weak consistency. Threads (and processes) communicate by sharing sequential objects with *restricted interfaces*, which expose only a monotone set of operations. In this way, objects can be replicated under eventual consistency while still exposing a strongly-consistent, transactional interface to client programs—keeping both the replication system *and* the programming model simple and performant. In our VLDB paper [6], we argue that such a programming model is both usable and essential.

Were Gallifrey to focus only on simple monotonic shared objects, its programming model would match that of past work—like CRDTs (convergent replicated data types) [12], or LVars [5]. I go a step further, recognizing that a sequential object corresponds to not just one lattice, but one of several choices—each of which exposes a different subset of possible operations. By automatically *transitioning* between various convergent representations, Gallifrey objects are able to express even non-monotone operations, while still avoiding any synchronization outside of transition points. Taken together, Gallifrey's monotone restrictions and transitions ensure that the programming model of a Gallifrey application is expressive, performant, and safe.

At UC Berkeley, the Hydro Project—based on a vision I developed with professors Joe Hellerstein, Natacha Crooks, and Alvin Cheung [2]—combines my existing work with new insights from the databases and security communities. With Hydro, we're bringing the ideas behind Gallifrey, Derecho, and MixT to new paradigms, including program synthesis, query languages, streaming systems, and data storage systems. While this effort is still in its early days, it has yielded results: Katara [7].

I originated and advised the Katara project to fill a key need of Gallifrey: the ability to share sequential objects by restricting their interface. Doing this efficiently requires generating CRDTs that are *semantically equivalent* to shared sequential objects. Following the established pattern of *verified lifting* [3], Katara uses program synthesis to generate CRDT implementations of data structures from traditional single-threaded equivalents, finding encodings that **match the most efficient CRDTs in the literature**. Katara's key insight is to reframe verified lifting to require only a partial semantic match between the sequential object and the CRDT, allowing concurrent CRDT operations to be arbitrarily ordered.

Future Work

In the Hydro Project, we're building out a vision to revolutionize the way cloud applications are built. Our 2021 CIDR paper [2] calls for the development of a new intermediate language for the cloud, capable of expressing existing application logic written in frameworks ranging from streaming systems to actor-based programs. This intermediate language needs to be relatively small and declarative, making it possible to leverage the techniques of Katara [7] to *automatically lift* existing framework-based code into equivalent declarative specifications. This technology would enable a wide variety of distributed programs to enjoy the established benefits of declarative code, from its capacity for whole-program optimizations to the ability to easily specify and verify correctness conditions. Fully realizing this vision will take many years; projects that we complete along the way have the potential for wide impact beyond the domain of distributed programming.

An opportunity for early impact lies in combining the insights of Gallifrey, Derecho, and Fearless Concurrency. In Gallifrey and Derecho, I take advantage of the fact that mutable objects may be safely shared concurrently so long as all updates and observations are monotonic. Integrating this insight into a system capable of *also* managing regions (as in Fearless Concurrency) would push such a system beyond the current holy grail of easy mutability recovery, and into the space of concurrent sharing with *mixed read-write access*—still without risking destructive data races. But this observation also raises questions: what *other* classes of mutability can we support without blocking? How should monotonicity be integrated into the language design—without exceeding our spare complexity budget? And most importantly, what real-world problems in concurrent programming would be solvable under such an analysis? Answering these questions would have wide-reaching impact across computer science: we have the chance to *nearly eliminate* the overhead of coordination and communication in applications ranging from modern distributed learning frameworks and scientific computing applications to auto-parallelizing compilers and mobile computing.

Such questions also arise in the space of weak-consistency and mixed-consistency programming. In MixT, we built a definition of mixed-consistency transactions, and an accompanying safety property based on noninterference. This definition is cogent and semantically sound, but it is too conservative; it fails to recognize that, under certain circumstances, one can *consistently observe* inconsistently-stored data. My work already demonstrates two ways to safely observe inconsistent data: via statically-tracked regions in Fearless Concurrency, and via monotonic updates in Gallifrey. Integrating these three efforts is a major undertaking; designing a type system which mixes both substructural and information-flow components with temporarily-restricted objects—while keeping things programmable and avoiding excessive complexity or annotation overheads—is fertile grounds for exciting research. These efforts also begin to map out what I believe to be a larger design space: there is daylight yet between the guarantees of full convergence provided by monotonicity, the guarantees of strong consistency assumed by most applications, and the guarantees of data-race freedom provided in fearless concurrency.

Regardless of how tightly we are able to integrate our languages and systems, general-purpose storage systems will still need a way to describe the consistency guarantees they make available. I believe that the wider question of how to correctly model consistency is not yet resolved; key to that is the question of what it *means* to observe potentially-inconsistent state. What are the *general mechanisms* by which inconsistent state can be consistently observed? How might we parameterize the set of consistent observations based on the guarantees offered by specific consistency models? In security, there are definitions for information flows that violate noninterference in controlled, acceptable ways. What might be the equivalent in consistency? These questions challenge a paradigm of shared-memory and distributed consistency that was established long before I was born. Answering them requires integrating insights from distributed systems, databases, formal consistency reasoning, type systems, and programming languages—and I believe that I am well-placed to work at—and across—these boundaries.

References

[1] *Akka Documentation: Replicated Event Sourcing*. Nov. 2022. URL: https://doc.akka.io/docs/akka/current/typed/replicated-eventsourcing.html.

[2] Alvin Cheung, Natacha Crooks, Joseph M. Hellerstein, and Mae Milano. "New Directions in Cloud Programming". In: *CIDR (Conference on Innovative Data Systems Research)* (2021).

[3] Alvin Cheung, Armando Solar-Lezama, and Samuel Madden. "Optimizing Database-Backed Applications with Query Synthesis". In: *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI '13. Seattle, Washington, USA: Association for Computing Machinery, 2013, pp. 3–14. ISBN: 9781450320146. DOI: 10.1145/2491956.2462180. URL: https://doi.org/10.1145/2491956.2462180.

[4] Sagar Jha, Jonathan Behrens, Theo Gkountouvas, Mae Milano, Weijia Song, Edward Tremel, Robbert Van Renesse, Sydney Zink, and Kenneth P. Birman. "Derecho: Fast State Machine Replication for Cloud Services". In: *ACM Trans. Comput. Syst.* 36.2 (Apr. 2019). ISSN: 0734-2071. DOI: 10.1145/3302258. URL: https://doi.org/10.1145/3302258.

[5] Lindsey Kuper and Ryan R Newton. "LVars: Lattice-Based Data Structures for Deterministic Parallelism". In: *Proceedings of the 2nd ACM SIGPLAN workshop on Functional high-performance computing*. 2013, pp. 71–84.

[6] Shadaj Laddad, Conor Power, Mae Milano, Alvin Cheung, Natacha Crooks, and Joseph M. Hellerstein. "Keep CALM and CRDT On". In: *PVLDB* 16.4 (2023). DOI: 10.14778/3574245.3574268.

[7] Shadaj Laddad, Conor Power, Mae Milano, Alvin Cheung, and Joseph M. Hellerstein. "Katara: Synthesizing CRDTs with Verified Lifting". In: *Proc. ACM Program. Lang.* 6.OOPSLA2 (Oct. 2022). DOI: 10.1145/3563336. URL: https://doi.org/10.1145/3563336.

[8] Mae Milano and Andrew C Myers. "MixT: a language for mixing consistency in geodistributed transactions". In: *39th ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI)*. June 2018, pp. 226–241.

[9] Mae Milano, Rolph Recto, Tom Magrino, and Andrew C. Myers. "A Tour of Gallifrey, a Language for Geodistributed Programming". In: *3rd Summit on Advances in Programming Languages (SNAPL 2019)*. Ed. by Benjamin S. Lerner, Rastislav Bodík, and Shriram Krishnamurthi. Vol. 136. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, 11:1–11:19. ISBN: 978-3-95977-113-9. DOI: 10.4230/LIPIcs.SNAPL.2019.11. URL: http://drops.dagstuhl.de/opus/volltexte/2019/10554.

[10] Mae Milano, Joshua Turcotti, and Andrew C. Myers. "A Flexible Type System for Fearless Concurrency". In: *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. PLDI 2022. San Diego, CA, USA: Association for Computing Machinery, 2022, pp. 458–473. ISBN: 9781450392655. DOI: 10.1145/3519939.3523443. URL: https://doi.org/10.1145/3519939.3523443.

[11] Philipp Moritz et al. "Ray: A distributed framework for emerging {AI} applications". In: *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 2018, pp. 561–577.

[12] Marc Shapiro. "A Comprehensive Study of Convergent and Commutative Replicated Data Types". en. In: *Encyclopedia of Database Systems*. Ed. by Ling Liu and M. Tamer Özsu. New York, NY: Springer New York, 2017, pp. 1–5. ISBN: 978-1-4899-7993-3. DOI: 10.1007/978-1-4899-7993-3_80813-1. URL: http://link.springer.com/10.1007/978-1-4899-7993-3_80813-1.

[13] Weijia Song, Mae Milano, Sagar Jha, Edward Tremel, Xinzhe Yang, and Ken Birman. "Derecho's Extensible, Intelligent Object Store". In: AISys@SOSP (2019).

[14] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. "Spark: Cluster computing with working sets". In: *2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10)*. 2010.

I have enjoyed multiple teaching opportunities at three universities, each proving a distinctive experience. In particular, I have been an instructor three times, a TA eleven times, head TA five, and worked on course development for six distinct courses. While I have learned much through all of these experiences, I will draw my teaching examples primarily from the largest course for which I served as sole instructor of record: CS2043.

Cornell's CS2043 is a technical "hidden curriculum" course with 130 students designed to teach everything from basic Unix shell tools to git and virtualization. Anonymous student course evaluations[1] were overwhelmingly positive, specifically praising the lecture and organization, and noting the course was "intuitive" and "engaging."

Teaching Philosophy

As a teacher, I keep in mind three key tenets: building classes that demystify their subjects, driving engagement with course material, and providing students multiple avenues to show mastery. These three tenets are key ingredients to a strong education experience. As instructor, I found opportunities to put these tenets in play—both at small (12-30 students) and medium (130 students) scale.

Demystifying The Subject I am excited to lead classes that spark student curiosity and inspire student exploration. I encourage students to look behind every curtain, teaching students that they can learn the basics behind even the most magical aspects of computer science. In my classes, this extends even as far as curriculum design and class infrastructure; at the end of my classes, students should know not just the material, but also feel capable of peeking inside even course submission and grading tools. In CS2043, I accomplished this by entwining assignment design and course infrastructure: all course assignments were prepared and submitted on the same server, and all class infrastructure was built using only the tooling taught in the first half of the class. I urged students to poke around, find easter eggs, use course infrastructure to communicate, or even "look inside" the autograding, quiz, and assignment submission systems.

Driving Engagement Active student engagement is essential for successful learning outcomes and directly tied to subject ownership, leading to mastery. In CS2043, I drove engagement through interactive demos and short quizzes. At the start of each class period, students would log into a shared Linux server and complete a short quiz in which they were asked to define any unix command of their choice—provided they had not defined it before. During the class period, I'd use this server for demos and have the students follow along, allowing students to independently explore the live effects of the commands on the server environment.

CS2043's quizzes were graded for completion—not correctness—allowing me to keep close track of student understanding without giving students too much extra stress. Guided by quiz responses, I was able to use review time built into each lecture to proactively correct misunderstandings or reinforce confusing material—before the corresponding assignments were due. Student reviews consistently mentioned high engagement, heralding the class as "so intuitive" and "fast-paced, but reasonable," indicating to me that this strategy was effective.

Allowing Students to Show Mastery In all my courses, I provide students multiple pathways to demonstrate mastery. In CS2043, our mix of project-based assignments, written assignments, and quizzes allowed students many ways to demonstrate their knowledge. When students missed project targets or began to consistently miss commands in quizzes, we reached out and invited them explicitly to meet with me (or a TA) for an interactive session, giving the students the opportunity to show their understanding and correct their past submissions. Ultimately, no student failed this course; every student was able to catch up and learn the material, no matter how far behind they had fallen in the semester.

As an aspiring professor, I am also keenly aware of the need to balance research and teaching. While the combined efforts I outline here may seem time-consuming, with the right infrastructure and organizational approach I have found the load to be quite manageable. For example, during the semester in which I taught CS2043, I was able to handle all course activities while also managing two distinct undergraduate research projects and submitting two first-author papers—a productive load for a graduate student even without considering teaching.

[1] available at `http://www.languagesforsyste.ms/files/milano-CS2043-evals.pdf`

Teaching Interests

I am especially excited to teach existing undergraduate-level courses in programming languages (including functional languages and compilers) or systems (including distributed systems, operating systems, and networking). I am also keen to introduce new courses, with ideas for a new undergraduate and graduate course.

As an upper-division undergraduate course, I would enjoy teaching a class in programming languages via language design. We would explore the various choices available to language designers and see how these choices manifest in existing languages—including their effect on tooling, community, and the perception of what the language is "good" for. At the graduate level, I would enjoy developing a class on application-focused system design. This class would combine paper-reading with class projects, aiming to motivate students to rethink traditional API assumptions and dive into what real-world applications actually need from their underlying systems in order to operate effectively.

Research Mentorship and Advising Philosophy

I have worked with many gifted undergraduate and graduate students during my time as a PhD student and postdoc. At UC Berkeley, I had the opportunity to prioritize advising and mentoring, including serving as an additional advisor for two junior PhD students and sole advisor for six undergraduates. A majority of my advisees chose to continue to graduate school and now study at Cornell, MIT, and Stanford. To protect students' privacy, I'll avoid referring to them by name or project here.

In mentoring I am guided by the needs and professional development of my students, rather than the immediate needs of a line of research. This means balancing students' research interests and agency, while ensuring students are never left adrift—at every stage of their work with me, from when they first join until project completion. A clear example of this philosophy in action comes when a student first joins my projects. I treat finding a research project for a new student as an exercise in curricular design. Aligning with my teaching approach, I draw up a series of "assignments" that nourish the student's foundational understanding and clarify potential research direction. For example, when an undergrad joined a project that required formal proof work, we began with close reading of similar work, collaborative discussion, and assignments that included concrete, approachable, week-by-week tasks, starting with basic proofs on toy type systems and leading to large-scale proofs on active research. This addressed the student's needs as a researcher, rather than just the narrow needs of a project, and ultimately led to a published paper.

But research is not a class; it is a collaboration. I strive to be a co-conspirator in research; I meet with students in neutral, student-centered spaces, like breakout rooms or hallway whiteboards. I join the students hands-on for projects, hold design sessions and even pair-program, as needed. I try to be available to students beyond the strictures of the hour-long weekly meeting and occasional emails, responding to students via DM instead of email, hopping on discord calls, or grabbing 15 minutes to check-in. I meet students where they are and stretch their development.

A critical component of my philosophy is giving my students a sense of ownership over research, by giving them a say in research direction. If their ideas or solutions come out of left field, they become an exercise in research communication! Through this process, students become stronger researchers, either by learning the skills needed to vet their ideas or by discovering an early—and meaningful—contribution to an ongoing project. For example, when an undergraduate's natural curiosity drew them to set aside tasks relevant to our paper submission and focus instead on extensions to our core ideas, I joined them on a journey to see what insights their curiosity would turn up. While this exploration didn't wind up making it into the paper, it did crystallize exciting future directions—and led the student to find a bug in the original system!

In both formal classroom settings and research mentoring, my student-centered approach has garnered positive feedback from former mentees and former students alike. I look forward to new opportunities to learn from my students as a professor.

Diversity, Equity and Inclusion Statement—Mae Milano (mpmilano@berkeley.edu)

Throughout my life, I have felt a commitment to equity and inclusion, recognizing that diversity—be it diversity of gender, race, income, or even perspective—brings value to every community [3]. This commitment has been borne out through my activities in both a professional and personal capacity. Just since finishing undergrad, I have taught courses to low-income elementary school students, served as transitional mentor to first-generation PhD students of color, created and run workshops to introduce girls to STEM topics (in a fun way!), co-founded an event dedicated to increasing access to research, and served in a national nonprofit, among other activities. As a transgender woman, I also bring a unique perspective. Having first-hand experience of how the world treats both women and men has afforded me an excellent, experiential tutorial on the role of privilege in the academy and beyond.

Inreach in graduate school In all stages of my education, I was privileged to benefit from the attention of incredible mentors and peers, all of whom were committed to lifting me up and encouraging me on the path to an academic career. When I raised issues of departmental climate with administrators, I was listened to, heard, and given the opportunity to make my case directly to the department chair; I had faculty reaching out to me, inviting me to join them on a research project.

Hoping to pay my experiences forward, I joined Cornell CS's mentoring program as a first-year mentor, serving over the course of my PhD as the direct mentor for seven PhD students, four of whom were people of color and three of whom were first-generation college students. As I helped these students navigate the transition to graduate school, find advisors, and fill out forms, I became keenly aware that my exceptional experiences were just that—an exception. The structure of academia did not work for everyone as it has worked for me.

At the time, Cornell CS's administration eschewed explicit policy documents in favor of an open-door policy and admirable availability. But while this policy served me well, it did not serve all; policies which require students to traverse vast power structures in order to be heard are well-known to disproportionately disenfranchise women and other under-represented minorities [2]. To address this, I worked with my peers to re-found the Cornell Computer Science Graduate Organization (CSGO), and made easy, anonymous, well-advertised mechanisms by which students could raise awareness of issues with the CSGO or run in CSGO elections. Our efforts bore fruit; by its third year of existence, CSGO's board contained 40% women and 25% people of color, and had successfully convinced the department to change its reimbursement policies and codify existing expectations, making conference travel accessible to under-resourced students while eliminating the need to be "in the know" in order to navigate departmental policies.

To increase access to research opportunities, I co-founded the department's ongoing and successful series of *research night* events, which are designed as a low-pressure way for graduate students to show their work to interested undergrads. This event took steps to center inclusion, such as collecting undergraduate attendee emails, and explicitly reaching out to offer those undergrads help navigating research opportunities. This design reduced the intimidation factor for undergraduates, many of whom—especially under-represented minorities—do not feel "permission" to cold email for opportunities [1].

Efforts within the Field As I began engaging with the research community outside of Cornell CS, I benefited from existing structures of mentorship—like the SOSP mentorship program and Programming Languages Mentoring Workshop (PLMW)—which helped ensure that I found my cohorts and was able to integrate into the field. As I gained seniority in the field, I was able to serve as a mentor in my own right, both during conferences (via PLMW) and via the globe-spanning SIGPLAN-M long-term mentoring program. These experiences granted me a window into the acute needs of students, nearly all women and under-represented minorities, whose relationships with their advisors had evolved into an unhealthy state and whose continued progress was at risk. Colored by these experiences, I became a founding member of the Computing Connections Fellowship[1] (CCF) selection committee. This organization aids students who need to change advisors, by both helping these students find new advisors at other universities and by providing initial funding to the student. Through the Herculean efforts of its president Talia Ringer, the CCF has secured funding, launched, and has already issued its first fellowship to a student in need. Through the CCF, we are addressing a thorny issue which disproportionately drives women and under-represented minorities out of our field; we hope to patch one of the many leaks in the "leaky pipeline" of academia and increase diversity as a result.

[1] `https://computingconnections.org/`

Outreach I believe that it is essential to also address issues of inequity outside the ivory tower. As a teacher and academic, I have focused my efforts on ensuring equity of access to STEM education, by showing low-income children and middle-school girls that computer science is a fun, approachable topic for everyone—not just the classic caricature of the neckbearded basement-dwelling hacker.

With a team of three PhD students, I taught a weekly hour-long course on how to "think like a coder" to third-grade students at the Belle Sherman elementary school in Ithaca, NY. In building and running this course, we turned to the expertise of the students' teachers and existing, published curriculum for programming aimed at students of this age group, put out by code.org. Our class prompted engagement and excitement from the students beyond the teachers' expectations; one non-verbal student even volunteered to show off her block-building code assignment, the first time this student had volunteered for anything that academic year.

At Cornell, I was part of a small team of PhD students that built and led "Googling with paper airplanes," an afternoon workshop for middle school girls which explained the basics of computer networking via paper airplanes. This workshop proved popular, and has been offered every year since as part of Cornell's broader "expanding your horizons" weekend of workshops for middle school girls.

Looking Ahead I am excited by the increased opportunities to impact diversity that come with a faculty role, whether it be in recruiting a diverse lab group, ensuring equity in my instruction, or continuing to make an impact in my department and beyond. In my first few years as faculty, my focus would be on bringing versions of the successful programs I have managed in the past to my new university home, while simultaneously learning and celebrating the initiatives already underway at that school. I would also plan to use the visibility that comes with a faculty role to help drive awareness of existing initiatives and to encourage volunteering within our departmental community.

Concretely, I would build on the successful series of fellowship application workshops which I ran as a graduate student, expanding them to incorporate targeted programs—like the Rising Stars program for near-graduation women—which *build cohorts* for women and under-represented minorities in computer science. Such cohorts are invaluable sources of support and community beyond what can be provided at the department level. My suggested workshops would introduce these programs to students, provide sample application materials, and support students in producing their own applications.

I would also encourage our students to look outward, by using the model of a "research night" event to build a "volunteering bazaar" which highlights initiatives both within and beyond the campus community for our students to get involved. Such an event could also serve as a catalyst for students (and faculty) to develop and recruit for their own initiatives, whether it's something as simple as restoring a department tradition or as complex as a multi-unit outreach effort.

Ultimately, each of these contributions is a small part of the larger puzzle of increasing diversity, equity, and inclusion in the academy. I am encouraged by the progress I have seen even in the course of my time at graduate school, and remain optimistic that, by working together, we can achieve the just society for which we all strive.

References

[1] Kalwant Bhopal. "Academics of colour in elite universities in the UK and the USA: the 'unspoken system of exclusion'". In: *Studies in Higher Education* 47.11 (2022), pp. 2127–2137. DOI: 10.1080/03075079.2021.2020746. eprint: https://doi.org/10.1080/03075079.2021.2020746. URL: https://doi.org/10.1080/03075079.2021.2020746.

[2] Social Sciences Feminist Network Research Interest Group. "The Burden of Invisible Work in Academia: Social Inequalities and Time Use in Five University Departments". In: *Humboldt Journal of Social Relations* 39 (2017), pp. 228–245. ISSN: 01604341. URL: http://www.jstor.org/stable/90007882 (visited on 11/22/2022).

[3] Vivian Hunt et al. *Diversity wins*. Tech. rep. McKinsey, 2020.

Mae Milano

Postdoctoral Scholar

University of California, Berkeley
465 Soda Hall, MC-1776
Berkeley, CA 94720-1776

Email: mpmilano@berkeley.edu
Website: http://www.languagesforsyste.ms/
Twitter: @mbpmilano

Research Interests

Programming Languages, Distributed Systems, Databases, Concurrency

Education

PhD, Computer Science, Cornell University. August 2020
Thesis: Programming Safely with Weak (and Strong) Consistency
Committee: Andrew Myers (chair), Nate Foster, Dexter Kozen

ScB, Music & Computer Science, Brown University. May 2013
Senior Prize.

Academic Appointments

Postdoctoral Scholar, advised by Joe Hellerstein. 2020-Present
RISE Lab, UC Berkeley EECS. *2020-2021*
Sutter Hill Ventures. *2021-2022*
Sky Lab, UC Berkeley. *2022-present*

Publications

Journal Articles

- **Keep CALM and CRDT On**. Shadaj Laddad, Conor Power, **Mae Milano**, Alvin Cheung, Natacha Crooks, Joseph M. Hellerstein. In: Proceedings of the VLDB Endowment 16.4 (2023). https://doi.org/10.14778/3574245.3574268
- **Katara: synthesizing CRDTs with verified lifting.** Shadaj Laddad, Conor Power, **Mae Milano**, Alvin Cheung, Joseph M Hellerstein. In: Proceedings of the ACM on Programming Languages 6.OOPSLA2 (Oct. 2022). https://doi.org/10.1145/3563336
- **Derecho: Fast state machine replication for cloud services.** Sagar Jha, Jonathan Behrens, Theo Gkountouvas, **Mae Milano**, Weijia Song, Edward Tremel, Robbert Van Renesse, Sydney Zink, Kenneth P Birman. In: ACM Tranactions on Computer Systems 36.2 (Apr. 2019). https://doi.org/10.1145/3302258

Peer-reviewed Conference Papers

- **A Flexible Type System for Fearless Concurrency**. **Mae Milano**, Joshua Turcotti, Andrew C Myers. In: Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI 2022). https://doi.org/10.1145/3519939.3523443
- **A Tour of Gallifrey, a Language for Geodistributed Programming. Mae Milano**, Rolph Recto, Tom Magrino, Andrew C Myers. In: 3rd Summit on Advances in Programming Languages (SNAPL), 2019. https://doi.org/10.4230/LIPIcs.SNAPL.2019.11
- **MixT: A language for Mixing Consistency in Geodistributed Transactions. Mae Milano** and Andrew C Myers. In: Proceedings of the 39th ACM SIGPLAN International Conference on Programming Language Design and Implementation (PLDI 2018). https://doi.org/10.1145/3296979.3192375
- **A Coalgebraic Decision Procedure for NetKAT.** Nate Foster, Dexter Kozen, **Mae Milano**, Alexandra Silva, and Laure Thompson. In: Proceedings of the 42nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL 2015). https://doi.org/10.1145/2676726.2677011
- **Python: The Full Monty**. Joe Gibbs Politz, Alejandro Martinez, **Mae Milano**, Sumner Warren, Daniel Patterson, Junsong Li, Anand Chitipothu, Shriram Krishnamurthi. In: Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages & applications (OOPSLA 2013). https://doi.org/10.1145/2544173.2509536

Workshops

- **New directions in cloud programming.** Alvin Cheung, Natacha Crooks, Joseph M Hellerstein, **Mae Milano**. In: The Conference on Innovative Data Systems Research (CIDR), 2021
- **Derecho's Extensible, Intelligent Object Store**. Weijia Song, **Mae Milano**, Sagar Jha, Edward Tremel, Xinzhe Yang, Ken Birman. In: AISys@SOSP, 2019.

Talks (*excluding conference talks for the above papers)*

- **Programming Languages and Distributed Systems: Better Together.** Invited Talk, Harvey Mudd CS department Colloquium, February 2023.
- **Researching with Undergraduates: a Curricular Approach.** Invited Talk, The ACM SIGPLAN conference on Systems, Programming, Languages, and Applications: Software for Humanity (SPLASH) Doctoral Symposium, December 2022.
- **A developer-centric approach to Fearless Concurrency.** Invited Talk, Swift Language Group, Apple Software, August 2022.
- **Languages and Systems for Cloud Programming.** UCSC Distributed Systems class and group (CSE138). University of California at Santa Cruz, November 2021

- **A Tour of Gallifrey, a Language for Geodistributed Programming.** IBM PL Day, December 2019.
- **Derecho: Blindingly Fast RDMA Replication for Cloud and Edge Services.** Tutorial, The 27th ACM Symposium on Operating Systems Principles (SOSP). October 2019
- **Presenting Gallifrey, a new Java-like language for wide-area distributed programming**. Workshop Talk, the 1st Eastern Great Lakes Programming Languages Seminar (EGLPLS). May 2019.

Awards & Honors

- Departmental Service Award, 2019-2020
- National Defense Science and Engineering (NDSEG) Fellowship, 2015
- National Science Foundation Graduate Research Fellowship (honorable mention), 2015
- Cornell University Fellowship, 2013
- Senior Prize in Computing, 2013
- Rose Rosengard Subotnik prize for excellence in writing, 2012

Industry

- **Sutter Hill Ventures: Researcher-in-Residence.** 2021. *Palo Alto, CA*
- **Microsoft Research: Research Intern.** Summer 2017
  Security Research Group – Cambridge, UK
- **Princeton Plasma Physics Lab: Research Intern.** Periods Between 2008-2011
  Scientific Visualization within Computational Plasma Physics Group (CPPG) – Princeton, NJ

Teaching Experience

Instructor.

University of California, Berkeley – Berkeley, CA

- CS294-170: Programming the Cloud: Fall 2020
  - Co-instructor with Joe Hellerstein; evenly split duties (such as paper selection, curriculum design, class sessions, and project supervision)

Cornell University – Ithaca, NY

- CS2043: UNIX Tools and Scripting: Spring 2019.
  - Sole instructor of record; reviews available at http://www.languagesforsyste.ms/files/milano-CS2043-evals.pdf

Brown University – Providence, RI

- CS195r: Compilers (In Practice): Spring 2012.
  - Developed curriculum, assignments and lectures; solo instructor

Teaching Assistant.
Cornell University – Ithaca, NY

- CS6410: Advanced Systems: Fall 2014
- CS4410: Operating Systems: Fall 2013

Brown University – Providence, RI

- CS166: Introduction to Computer System Security: Spring 2013
- CS131: Fundamentals of Computer Systems: Spring 2013
- CS033: Introduction to Computer Systems: Fall 2012
- CS 195s: Fundamentals of Computer Systems: Spring 2012
- CS266: Advanced Computer System Security: Spring 2012
- CS031: Introduction to Systems Programming: Fall 2011, Fall 2010
- CS032: Introduction to Software Engineering: Spring 2011
- CS002: The Digital World: Spring 2010

Student Supervision & Mentorship

Masters.
University of California, Berkeley – Berkeley, CA

- Mingwei Samuel: Fall 2020–Fall 2021
- Pranav Saig Gaddamadugu: Fall 2020

Undergraduate.
University of California, Berkeley – Berkeley, CA

- Joshua Turcotti: Fall 2020–Spring 2022
- William Brandon: Fall 2021–Spring 2022
- Benjamin Driscoll: Fall 2021–Spring 2022
- Frank Dai: Fall 2021–Spring 2022
- Wilson Berkow: Fall 2021–Spring 2022

Cornell University – Ithaca, NY

- Patrick LaFontaine: Fall 2019–Spring 2020.
- Danny Yang: Fall 2019–Spring 2020.
- Chris Roman: Fall 2018–Fall 2019
- Yijia Chen: Fall 2018–Spring 2018
- Sahithi Kalvakota: Fall 2018–Spring 2018
- Alex Katz: Fall 2018–Spring 2018

Service

Program Committees
External Review Committee

- Proceedings of the ACM on Programming Languages (OOPSLA) 2023

Workshops

- PaPoC 2022
- EuroDW 2021

Artifact Evaluation Committee

- OOSPLA 2023
- PLDI 2019
- POPL 2017
- SPLASH 2015

Session Chair

- Weak Memory Models, POPL 2022

Service to the Field

- Selection Committee, Computing Connections Fellowship (https://computingconnections.org)
- SIGPLAN-M long-term mentor (https://www.sigplan.org/LongTermMentoring/)

Institutional Service

Cornell University, Computer Science

- Research Night Organizer. Spring and Fall 2019
- PhD Admissions Committee Reader. 2018-2020
- Departmental volunteer coordinator (czar tsar). 2014-2016
- Programming Languages Discussion Group (PLDG) coordinator. 2014-2018
- Vice President, Computer Science Graduate Organization. Spring 2016-2017
- TA Application Coordinator. 2015-2017
- Secretary, Computer Science Graduate Organization. Summer 2015
- Fellowship Archive and Workshop Organizer. 2014-2020

Brown University, Computer Science

- Systems Operator, Programmer, and Consultant (SPOC) 2010-2013

Community Service

- Instructor, Belle Sherman Elementary School. 2014
- Expanding Your Horizons Workshop Leader. 2015, 2016, 2017, 2018, 2019

Recommendation Letter Contact Information

To whom it may concern:

I have requested letters of recommendation from the following four faculty members:

Andrew C Myers, Cornell University: andru@cs.cornell.edu

Ken Birman, Cornell University: ken@cs.cornell.edu

Joe Hellerstein, University of California, Berkeley: hellerstein@berkeley.edu

Nate Foster, Cornell University: jnfoster@cs.cornell.edu

If your search allows only three recommenders, please ask Andrew, Ken, and Joe.

Thank you!

Mae Milano