

NICKOLAI ZELDOVICH

Stanford University, Computer Science Department
353 Serra Mall, Room 288
Stanford, CA 94305-9025

Phone: (650) 996-4201
nickolai@cs.stanford.edu
<http://www.scs.stanford.edu/~nickolai>

EDUCATION

- 2002–2007 **Stanford University** Stanford, CA
01/2008 Ph.D. in Computer Science
Thesis title: *Securing Untrustworthy Software Using Information Flow Control*.
Advisor: David Mazières.
- 1998–2002 **Massachusetts Institute of Technology** Cambridge, MA
06/2002 M.Eng. in Electrical Engineering and Computer Science
Thesis title: *Concurrency Control for Multi-Processor Event-Driven Systems*.
Advisor: Robert Morris.
- 06/2002 S.B. in Electrical Engineering and Computer Science, minor in Mathematics

RESEARCH INTERESTS

Operating systems, distributed systems, security, and networking.

RESEARCH PROJECTS

- 2005–present **HiStar: a secure operating system [2]**. Led the HiStar project on designing and developing a new operating system that allows applications to minimize the amount of trusted code. HiStar allows applications to specify precise data security policies by specifying how different information can flow through the system. As a result, small amounts of trusted code can reason about the security of large, complex, and potentially buggy applications. HiStar enforces precise information flow restrictions by breaking down and refactoring traditional OS abstractions into six basic types of objects and a small number of operations that make all information flow explicit.
- HiStar solves a number of challenges. In particular, HiStar tracks information flow dynamically without allowing the tracking mechanism itself to leak information. By decoupling resource allocation and revocation from all other forms of resource access, HiStar also allows a system administrator to manage resources without any inherent superuser privileges to read and write all data in the system.
- HiStar's abstractions are flexible enough to implement a self-hosting Unix-like environment in an untrusted user-level library. At the same time, HiStar supports novel, highly-secure applications side-by-side with traditional Unix code, such as an entirely untrusted login process, or privacy-preserving untrusted virus scanners.
- 2006–present **Security in distributed systems [1]**. Designed and implemented DStar, a decentralized network protocol for enforcing information flow control in a distributed system. By controlling information flow, DStar allows building secure distributed applications from largely untrusted code, such as a scalable web server that has little trusted code and no fully-trusted machines or components.
- DStar's decentralized environment poses a number of unique challenges, including trust and resource allocation, whose solution is taken for granted on a single machine. The

key idea in DStar is to use self-certifying names to specify information flow restrictions. This allows individual machines to determine who is trusted to handle what data without any external communication, which could otherwise leak information.

2007–present **Hardware support for security [8].** Co-designed Loki, a hardware architecture for minimizing trusted code, and re-architected HiStar to take advantage of Loki. Loki directly enforces security policies on words of physical memory in hardware, instead of relying on indirect mechanisms, such as page tables, to enforce security. A novel three-layer kernel architecture allows HiStar to reduce the amount of fully-trusted code by a factor of two on Loki, starting from an already-small kernel of under 20,000 lines of code. A small *security monitor*, running with full privileges, enforces traditional read/write memory access control. The rest of HiStar’s kernel code executes on top of the security monitor and enforces information flow control but cannot violate the monitor’s simpler security guarantees. A full-system FPGA prototype of Loki achieves good performance running a variety of workloads in HiStar’s Unix-like environment.

2006–present **Security verification.** Ongoing work in collaboration with a number of research groups at Stanford to verify the security of HiStar, DStar, and Loki. HiStar’s clear definition of security in terms of information flow control allows precisely specifying what it means to be secure, and what it means for security to be broken. This formal definition enables a number of approaches to verifying security, with promising results. Using model checking, we have proven that a subset of HiStar’s kernel interface is secure. We used static analysis to ensure that certain safety properties hold in the implementation of the HiStar kernel. Finally, we are examining the behavior of HiStar’s kernel code on all possible inputs using symbolic execution. This allows verifying a wide range of properties; currently the focus is on test coverage, while the long-term goal is proving the security properties of the information flow control mechanism.

2002–2005 **Virtual appliances [3, 4, 5].** Designed and implemented systems that used virtual machines to solve problems of user mobility, software distribution, and system management [3]. Treating the entire virtual machine as the unit of mobility, distribution, and management solves a number of problems, such as missing or mismatched dependencies, broken software installs, and unreliable updates. Virtual machine monitors also provide a reliable recovery mechanism even when the virtual machine has been compromised.

Co-founded a company called moka5 to commercialize the research project’s ideas and prototypes in the context of desktop software management.

Worked on caching policies and prefetching algorithms for downloading virtual machine disks over the network. Designed and implemented a system for measuring interactive performance of desktop environments [4] to evaluate the resulting performance. Developed an object-oriented language to describe, configure, and distribute virtual machines and networks of virtual machines [5].

2001–2002 **Concurrency control [6, 7].** Designed and implemented a simple mechanism for exposing computational concurrency in event-driven programs, by associating a *color* with every event callback to represent the accessed data. Existing programs can incrementally color their callbacks to take advantage of multi-processor systems. Achieved significant performance improvements for an event-driven file server on a multi-processor machine, by modifying 90 lines of code to parallelize cryptographic operations.

WORK EXPERIENCE

- 10/2007–present **Postdoctoral Scholar.** CS Department, Stanford UniversityStanford, CA
05/2005–09/2005 **Co-founder.** SkyBlue Technologies (now moka5)Redwood City, CA
09/2002–09/2007 **Research Assistant.** CS Department, Stanford University Stanford, CA

- 06/2001–08/2002 **Research Assistant.** PDOS Research Group, MIT LCS Cambridge, MA
- 06/2000–08/2000 **Embedded Software Developer.** Kaveri Networks Sunnyvale, CA
Designed and developed network protocols for low-power Internet-connected devices.
- 11/1999–02/2004 **Student Programmer.** Athena Server Operations Group, MIT Cambridge, MA
Helped develop and maintain a variety of computing services at MIT, including the AFS file system, web services, and remote login.
- 06/1999–08/1999 **Research Intern.** Naval Research Laboratory Washington, DC
Incorporating just-in-time signaling protocols in optical cross-connect switches.
- 05/1998–08/1998 **Undergraduate Researcher.**
Computer Vision Lab, University of Central Florida Orlando, FL
Worked on video segmentation, shot segmentation and similarity, and violence detection, using skin detection and optical flow algorithms.
- 08/1997–09/2003 **Systems Administrator.** Craig’s Data Exchange Mount Dora, FL
Helped manage all technical aspects of a medium-size Internet service provider.
- 06/1997–08/1997 **Research Intern.** EE Department, Princeton University Princeton, NJ
Implemented cyclic memory access optimizations for DSP chips in the SUIF compiler.
- 06/1996–08/1996 **Web Application Developer.** University of Central Florida Orlando, FL
Developed dynamic, database-driven web applications.

TEACHING EXPERIENCE

- 12/2007 **Guest Lecturer.** Operating Systems (CS 140), Stanford University.
- 05/2007 **Guest Lecturer.** Advanced Topics in Operating Systems (CS 240), Stanford University.
- 01/2007–03/2007 **Course Assistant.** Distributed Systems (CS 244b), Stanford University. Developed and graded lab assignments that involved students building parts of a network file system replicated using Paxos. Taught lab section. Helped students understand papers covered in course.
- 09/2005–12/2005 **Course Assistant.** Advanced Operating Systems Implementation (CS 240c), Stanford University. Helped students with lab assignments, which involved developing parts of a small operating system. Answered questions about OS research papers covered in class.

PROFESSIONAL ACTIVITIES

- Journal Reviewer:** Journal of Systems and Software (JSS), 2007.
- Conference Reviewer:** VEE 2008, PACT 2007, SOSP 2007, IEEE Security and Privacy 2007, OSDI 2006, SOSP 2003.

REFEREED CONFERENCE PUBLICATIONS

- [1] Nickolai Zeldovich, Silas Boyd-Wickizer, and David Mazières. Securing distributed systems with information flow control. In *Proceedings of the 5th Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, April 2008. To appear.
- [2] Nickolai Zeldovich, Silas Boyd-Wickizer, Eddie Kohler, and David Mazières. Making information flow explicit in HiStar. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI)*, pages 263–278, Seattle, WA, November 2006.

- [3] Ramesh Chandra, Nikolai Zeldovich, Constantine Sapuntzakis, and Monica Lam. The Collective: A cache-based system management architecture. In *Proceedings of the 2nd Symposium on Networked Systems Design and Implementation (NSDI)*, pages 259–272, Boston, MA, May 2005.
- [4] Nikolai Zeldovich and Ramesh Chandra. Interactive performance measurement with VNCplay. In *Proceedings of the USENIX 2005 Annual Technical Conference, FREENIX track*, pages 189–198, Anaheim, CA, April 2005.
- [5] Constantine Sapuntzakis, David Brumley, Ramesh Chandra, Nikolai Zeldovich, Jim Chow, Jim Norris, Monica S. Lam, and Mendel Rosenblum. Virtual appliances for deploying and maintaining software. In *Proceedings of the 17th USENIX Large Installation System Administration Conference*, pages 181–194, San Diego, CA, October 2003.
- [6] Nikolai Zeldovich, Alexander Yip, Frank Dabek, Robert T. Morris, David Mazières, and M. Frans Kaashoek. Multiprocessor support for event-driven programs. In *Proceedings of the USENIX 2003 Annual Technical Conference*, pages 239–252, San Antonio, TX, June 2003.

REFEREED WORKSHOP PUBLICATIONS

- [7] Frank Dabek, Nikolai Zeldovich, M. Frans Kaashoek, David Mazières, and Robert Morris. Event-driven programming for robust software. In *Proceedings of the 10th ACM SIGOPS European Workshop*, pages 186–189, September 2002.

IN SUBMISSION

- [8] Hari Kannan, Nikolai Zeldovich, Michael Dalton, and Christos Kozyrakis. Architectural support for minimizing trusted code. In submission, November 2007.

INVITED TALKS AND PRESENTATIONS

- 12/2007 **Stanford Information Networks Group Seminar** Stanford, CA
Securing Untrustworthy Software Using Information Flow Control
- 10/2007 **Stevens Institute of Technology** Hoboken, NJ
Securing Untrustworthy Software Using Information Flow Control
- 08/2007 **Sun Labs** Menlo Park, CA
Securing Untrustworthy Software Using Information Flow Control
- 05/2007 **University of California, Berkeley Systems Lunch Seminar** Berkeley, CA
Securing Untrustworthy Software Using Information Flow Control
- 04/2007 **Stanford Clean Slate Network Research Seminar** Stanford, CA
Distributed Information Flow Control
- 03/2007 **TRUST Site Visit Poster Session** Berkeley, CA
Making Information Flow Explicit in HiStar
- 11/2006 **2006 Usenix OSDI Conference** Seattle, WA
Explicit Information Flow in the HiStar OS
- 10/2006 **Kryptos Study Tour** Stanford, CA
Making Information Flow Explicit in HiStar
- 04/2005 **2005 Usenix Annual Technical Conference** Anaheim, CA
Interactive Performance Measurement with VNCplay
- 06/2003 **2003 Usenix Annual Technical Conference** San Antonio, TX
Multiprocessor Support for Event-Driven Programs

05/2003 **Stanford Computer Forum Poster Session**Stanford, CA
Virtual Appliances

AWARDS

06/2001 George C. Newton Annual Prize for best undergraduate laboratory project at MIT. The project was a networked thermometer, which involved connecting an 8-bit microcontroller to an Ethernet adapter and temperature sensor, and developing a network stack and web server in assembly.

PATENTS

Monica Lam, Constantine Sapuntzakis, Ramesh Chandra, Nickolai Zeldovich, Mendel Rosenblum, Jim Chow, and David Brumley. "Virtual Appliance Management." U.S. Patent Application No. 11/007,911. Pending, filed December 2004.

PROFESSIONAL AFFILIATION

Member of ACM (1998), Usenix (2002).

PERSONAL INFORMATION

United States citizen. Fluent in Russian. Date of birth 11/11/1981. Married.

REFERENCES

Prof. David Mazières
Stanford University CS Department
353 Serra Mall, Room 290
Stanford, CA 94305-9025
(650) 723-8777
kolyarec@nospam.scs.stanford.edu

Prof. Monica Lam
Stanford University CS Department
353 Serra Mall, Room 307
Stanford, CA 94305-9030
(650) 725-3714
lam@stanford.edu

Prof. Dawson Engler
Stanford University CS Department
353 Serra Mall, Room 314
Stanford, CA 94305-9030
(650) 723-0762
engler@stanford.edu

Prof. Robert Morris
MIT Computer Science and AI Lab
32 Vassar St., Room 32-G972
Cambridge, MA 02139
(617) 253-5983
rtm@csail.mit.edu

Stanford, CA, January 8, 2008

Research Statement

Nickolai Zeldovich

My research interests are in building secure computer systems. Security problems are everywhere, from desktop computers running spyware, to server compromises revealing millions of social security and credit card numbers, to computer worms bringing down safety systems at nuclear power plants. Lack of security also hampers innovation, as many novel applications are not deployed due to security concerns.

A key problem is that overall security often depends on the correctness of the entire system, which is usually too complex to get right. Most existing approaches to security, such as firewalls or virus scanners, focus on specific aspects of security without considering the system as a whole, and do not qualitatively improve security. My research takes a different approach: it strives to find new system designs in which security largely depends on small parts of the overall system. In most systems, low-level abstractions such as hardware tend to be thoroughly tested and verified. An ideal system would directly leverage these same abstractions to enforce overall application security, such as that of a web server, without unduly increasing the complexity of the abstractions.

Enforcing all security at a low level is difficult because each level of abstraction in a system introduces a *semantic gap* in security, meaning that it is hard to express high-level security policy in terms of lower-level security mechanisms. Instead, most systems implement and enforce different security policies at different levels, and security depends on correctness at every one of these levels. For example, consider an Apache web server running on Linux. Linux provides a security mechanism based on user IDs, which works well for users logging in over SSH. However, Apache has its own security policy, specifying how HTTP users should be authenticated and what web pages and server-side scripts they can access. The Linux user ID mechanism is nearly useless for enforcing such a policy, so Apache implements and enforces its own security. Similarly, the page table mechanism in hardware only protects the kernel from user accesses, and the kernel must enforce the user ID mechanism on its own. The result is that neither secure hardware nor a secure Linux kernel is sufficient for enforcing the web server's security policy.

My research focuses on closing this semantic gap, so that high-level security policies can rest primarily on lower-level security mechanisms. This allows security to cut through complex layers such as application libraries and the kernel, which no longer need to re-implement their own security mechanisms from scratch. The challenge is to design security mechanisms that are *simple* enough to enforce at a low level, such as in a small kernel or in hardware, yet *expressive* enough to let applications specify their high-level security policies.

Of course, building operating systems and hardware around application-specific mechanisms is not a practical solution. Instead, these mechanisms should be flexible enough to support a wide range of applications. I have found that one indicator of an inflexible design is the use of privileged mechanisms for system management, which often indicates a deficiency in the general-purpose mechanisms available to unprivileged applications. For instance, in Unix, only root can set user IDs, which makes it difficult for Apache to use the same user ID mechanism to isolate different HTTP users.

On the other hand, *egalitarian* mechanisms—ones that have no pre-defined privilege levels to determine what kinds of operations can be accessed—tend to be naturally flexible. In my research, I have developed egalitarian operating system security mechanisms that can express security policies of existing systems such as Unix, and at the same time allow applications with different security requirements to implement their own policies using the same mechanisms. Egalitarian mechanisms also make it possible to reuse traditionally privileged system code, such as user authentication, in a less privileged context like a web server, since by design, all operations will still be accessible.

Building a system around a new set of security mechanisms requires rethinking the design at many levels of abstraction, and I enjoy working with all aspects and layers of a system in my research. Making these systems practical also requires building and using full-scale prototypes to understand their limitations and further refine their design. To this end, I have built a fully functional, self-hosting operating system called HiStar that provides novel security properties to applications, developed a decentralized network protocol

called DStar for building secure distributed systems, and co-designed a modified SPARC processor, Loki, running on an FPGA, which provides strong application security guarantees. These systems are described below in more detail.

Operating system security. HiStar [1] is an operating system designed around *information flow control* as the one and only security mechanism. Applications in HiStar specify security policies for *data* by controlling how information can propagate through the system. Many high-level security policies can be expressed in terms of information flow, such as “my financial details cannot be sent over the network,” or “a user’s private data can only be sent to that user’s browser.” HiStar’s information flow control mechanism also makes it possible to express access control policies, which can prevent sensitive data from being read in the first place; for example, traditional Unix file read and write permissions correspond to information flowing from or to the file. Finally, information flow control allows reasoning about causality by controlling communication between different parts of the system. As a result, small amounts of trusted code can enforce complex application security policies through interposition.

HiStar’s well-defined information flow control mechanism is simple enough to be enforced by a small kernel of under 20,000 lines of code. To provide strong security guarantees, HiStar breaks down traditional OS abstractions into six kernel object types, along with a small number of operations that act on one or two objects at a time and precisely define where information is flowing from and to. HiStar’s egalitarian kernel interface has no concept of superuser and allows anyone to specify information flow restrictions. To make it possible to manage a system without superuser privileges, HiStar separately controls access to data, such as private user files, and access to resources, such as disk space. Granting the administrator access to the top of the resource hierarchy allows the administrator to control the system’s resources without having any inherent data access privileges.

Using these kernel objects, HiStar implements a self-hosting Unix environment in an untrusted user-level library. However, applications can also bypass the Unix library to implement novel functionality, such as a web server whose security is largely enforced by the kernel, or an entirely untrusted login process in which users can provide their own password-checking agents. HiStar’s egalitarian mechanism also enables modularity: code initially designed for authenticating Unix users

during login was reused without modification by the HiStar web server for authenticating HTTP users.

Security in distributed systems. While HiStar enforces security on a single machine, DStar [2] can enforce similar information flow security policies across multiple machines. Distributed systems are often composed of partially trusted machines, and the key to security is in the trust relations between them. To this end, DStar provides a fully decentralized, egalitarian mechanism that gives applications full control over what machines are trusted to handle which parts of the application’s data. By using DStar, HiStar’s web server scales to multiple machines without fully trusting any of them.

The main idea behind DStar is self-certifying information flow policies, which include the data owner’s public key in the name of each policy that applies to that data. The resulting network protocol allows each independent machine to verify trust relations, and separates the specification of trust from the enforcement mechanism. Applications sign trust relations with their private keys, and DStar enforces these trust relations by verifying signatures.

Hardware security. HiStar’s information flow control mechanism is simple enough to be partially enforced in hardware, which is an attractive proposition given hardware’s strong track record of thorough testing. The Loki hardware architecture [3] provides a mechanism for associating security *tags* directly with physical memory words. Tags correspond to security policies for memory, and the privileges of a process are defined in terms of the tags it can read or write. Loki’s egalitarian mechanism treats all tag values uniformly, and it is the job of a small *security monitor*, running underneath the HiStar kernel, to translate information flow restrictions specified by applications into tag values and manage the set of accessible tags for each process. As a result, hardware can directly enforce the security of applications such as the HiStar web server.

With Loki, hardware enforces access control according to the application’s information flow policy, and the kernel’s job becomes minimizing the bandwidth of covert information flow channels. As a result, an attacker who compromises the kernel gains access to high-bandwidth covert channels, but cannot bypass the hardware access control mechanism, which enforces policies such as file read and write permissions. Loki’s security monitor is half the size of the HiStar kernel, since it does not try to minimize covert channels or manage devices and page tables. At the same time, Loki adds minimal hardware complexity, and our

FPGA prototype runs HiStar at the same clock speed of 65 MHz as the original processor that we modified.

Past work. My previous research explored the role of virtual machines in addressing the problems of system management, security, and mobility [4]. Encapsulating all applications and user state in virtual machines and making virtual machines the unit of management reduces most management tasks into simple operations on virtual disk images. A specialized configuration language further simplifies deployment and management of entire networks of virtual machines [5]. I co-founded a company called moka5 to commercialize these research ideas and prototypes in the context of desktop management.

Future directions. I have a strong interest in continuing research on building secure systems, and see a number of possible directions in which to proceed.

I am interested in how hardware mechanisms can improve security. Hardware resources are becoming abundant, with many-core processors appearing in the near future, and I want to explore OS mechanisms that trade off resources for security. For instance, a separate core could be used to execute code handling a sensitive private key in strong isolation. Other upcoming hardware mechanisms hold promise for security as well, such as I/O virtualization support, which can help move complex device drivers out of the fully-trusted kernel, greatly improving overall security. At the same time, complex hardware features like hyperthreading and shared caches often introduce covert channels, making it difficult to control information flow. I am interested in developing mechanisms that can better control the degree of resource sharing to give applications more control over their security; related hardware mechanisms have been proposed for deterministic performance and quality-of-service.

Network security may also benefit from simpler security mechanisms. Existing mechanisms, such as firewalls and intrusion detection systems, are based on complex heuristics that cannot provide strong security guarantees. If applications could specify their security requirements to the network through a simple mechanism, firewalls could in turn provide higher-level security guarantees. A better mechanism to control network resource allocation can also solve a number of security and reliability problems, such as covert channels and denial-of-service attacks. Indeed, denial-of-service attacks can be viewed as a special case of covert channels, where a flood of low-integrity packets interferes with high-integrity packets of an important application.

Unlike the subjective and often-changing security mechanisms of traditional systems like Unix, well-defined information flow control mechanisms are particularly well suited to formal verification, since any violation can be objectively proven by example. Model-checking can be used to verify whether a design is secure, and preliminary work has proven the security of a subset of the HiStar kernel interface. HiStar's small kernel is also amenable to verifying the implementation for correctness. Preliminary work in this area is focusing on test coverage using static analysis and symbolic execution, but proving security is the long-term goal.

Security mechanisms must be usable by application developers, and I am interested in exploring how programming languages can make it easier to develop secure applications and express security policies. Applications running on mobile devices, such as cell phones, pose yet another set of challenges: what mechanisms are needed to enforce security in a network of wireless devices with intermittent connectivity, or to prevent malicious code from draining battery power? I also want to explore system designs in which high-level application security can primarily rest on the correctness of cryptographic mechanisms.

Finally, security is only a step toward novel functionality. Better system security will enable innovative applications in areas currently limited by security and isolation concerns, with finance, health care, and air traffic control being just a few examples. Exploring such applications would be fruitful research in itself.

REFERENCES

- [1] Nikolai Zeldovich, Silas Boyd-Wickizer, Eddie Kohler, and David Mazières. Making information flow explicit in HiStar. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI)*, pages 263–278, Seattle, WA, November 2006.
- [2] Nikolai Zeldovich, Silas Boyd-Wickizer, and David Mazières. Securing distributed systems with information flow control. In *Proceedings of the 5th Symposium on Networked Systems Design and Implementation (NSDI)*, San Francisco, CA, April 2008. To appear.
- [3] Hari Kannan, Nikolai Zeldovich, Michael Dalton, and Christos Kozyrakis. Architectural support for minimizing trusted code. In submission, November 2007.
- [4] Ramesh Chandra, Nikolai Zeldovich, Constantine Sapuntzakis, and Monica Lam. The Collective: A cache-based system management architecture. In *Proceedings of the 2nd Symposium on Networked Systems Design and Implementation (NSDI)*, pages 259–272, Boston, MA, May 2005.
- [5] Constantine Sapuntzakis, David Brumley, Ramesh Chandra, Nikolai Zeldovich, Jim Chow, Jim Norris, Monica S. Lam, and Mendel Rosenblum. Virtual appliances for deploying and maintaining software. In *Proceedings of the 17th USENIX Large Installation System Administration Conference*, pages 181–194, San Diego, CA, October 2003.

Teaching Statement

Nickolai Zeldovich

My interest in teaching comes from the same motivation for simple, elegant, and practical system design that drives my research. I am interested in making computer systems more secure, and a key part of that is teaching the next generations of students how to build secure systems. In particular, I look forward to teaching courses on operating systems, distributed systems, networking, and security, as well as more specific graduate seminars focusing on my research interests.

I find teaching to be highly rewarding in many ways. Seeing students understand new ideas is very satisfying in and of itself, and at the same time, interactions with students are also invaluable in research. In my experience, teaching—be it the process of advising a student, developing material for a class, or presenting a lecture—invariably brings around various problems and techniques in a new light. This makes teaching inseparable from research, and in fact, I gained important insights for both my operating system and distributed system research projects while I was a teaching assistant for classes on those topics.

In the classroom, teaching provides me with a unique opportunity to find clear explanations for conveying existing knowledge to students. However, once students understand the material, it is even more exciting to see them take the next step and start coming up with their own follow-on ideas and questions. One way that I like to encourage students to explore their newfound knowledge is through hands-on practical experience, by working on real projects that build on material from lecture. Not only does this improve their understanding, but it also often piques their intellectual curiosity, and entices them to think about broader issues.

As a teaching assistant, I took this approach and designed projects to help students explore what they learned about in class. In a distributed systems course, I developed a replicated file server based on the Paxos consensus protocol to highlight the practical aspects of building a distributed system. While the lab assignments for students involved filling in purposely-omitted portions of the code, the point was not mundane coding. To the contrary, the goal was for students to understand how their abstract knowledge translates into the design of actual systems. This gave students a solid understanding of how distributed systems are built, and in a second part of the course, free-form final projects further allowed them to explore how their knowledge can be used to solve a variety of ambitious problems.

As a teaching assistant in an operating systems class, I followed the same path. By filling in missing pieces of a simple operating system, students learned how the different components of an operating system come together in practice, and free-form final projects helped them explore their own ideas in operating system design. Many students found the experience invaluable, and it was particularly rewarding to see one of the students, Silas Boyd-Wickizer, continue on to work on operating systems research, first as part of my research project at Stanford, and now at MIT.

Indeed, an equally important aspect to teaching lies outside the classroom in advising students. Over the past two years, I have been helping advise both Ph.D. and Masters students that worked with me on my research project, HiStar. In doing so, I have found that an advisor's main role is to help each student choose a suitable problem that is both interesting and a good fit for them. For me, the key to doing this has been to understand the student's strengths and interests, and provide them with appropriate starting points for exploring research directions. Once students find a problem that is interesting to them, I find myself learning a lot from their ideas, and working with them becomes immensely rewarding and motivating.