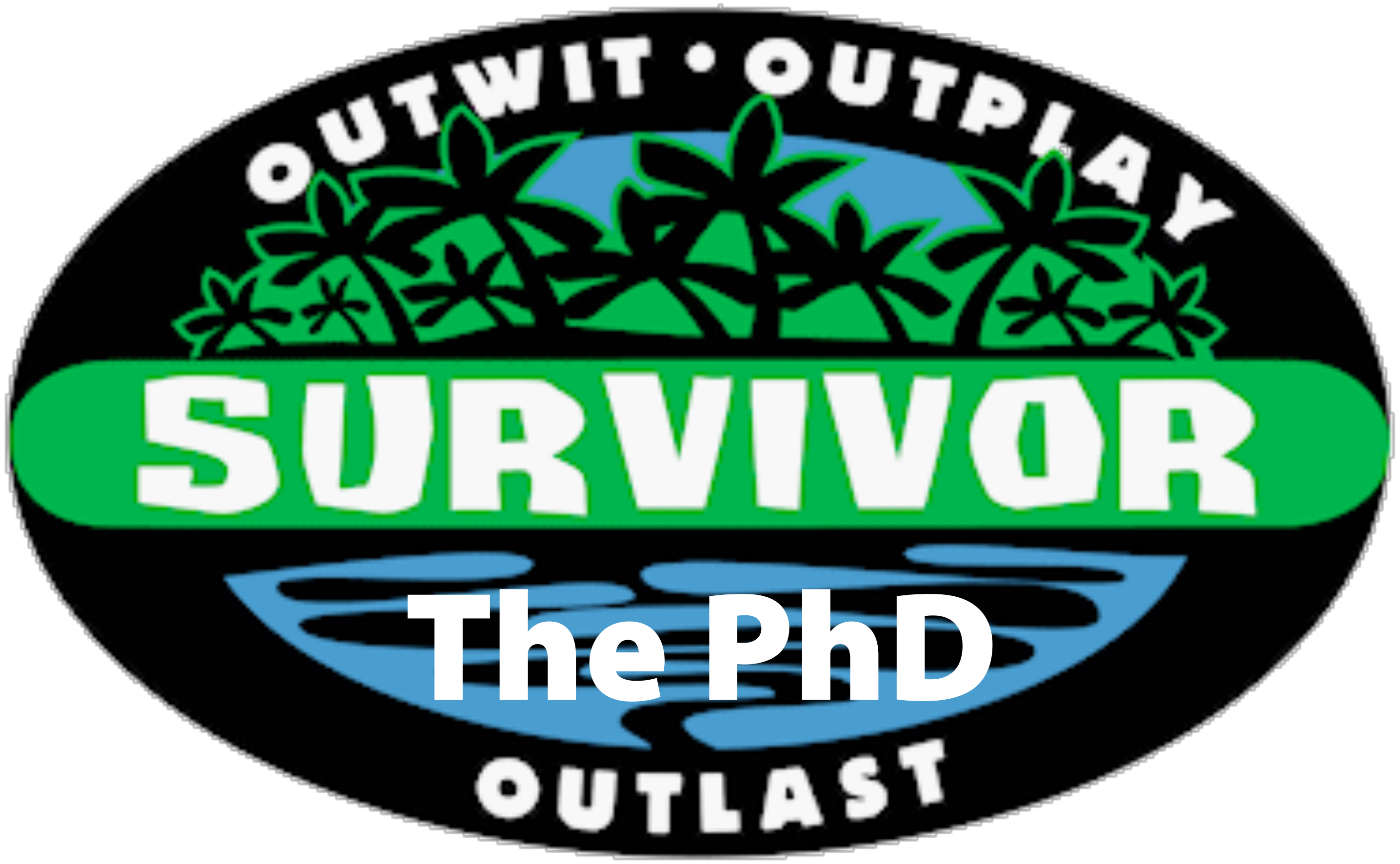
Pretty Handy Degree

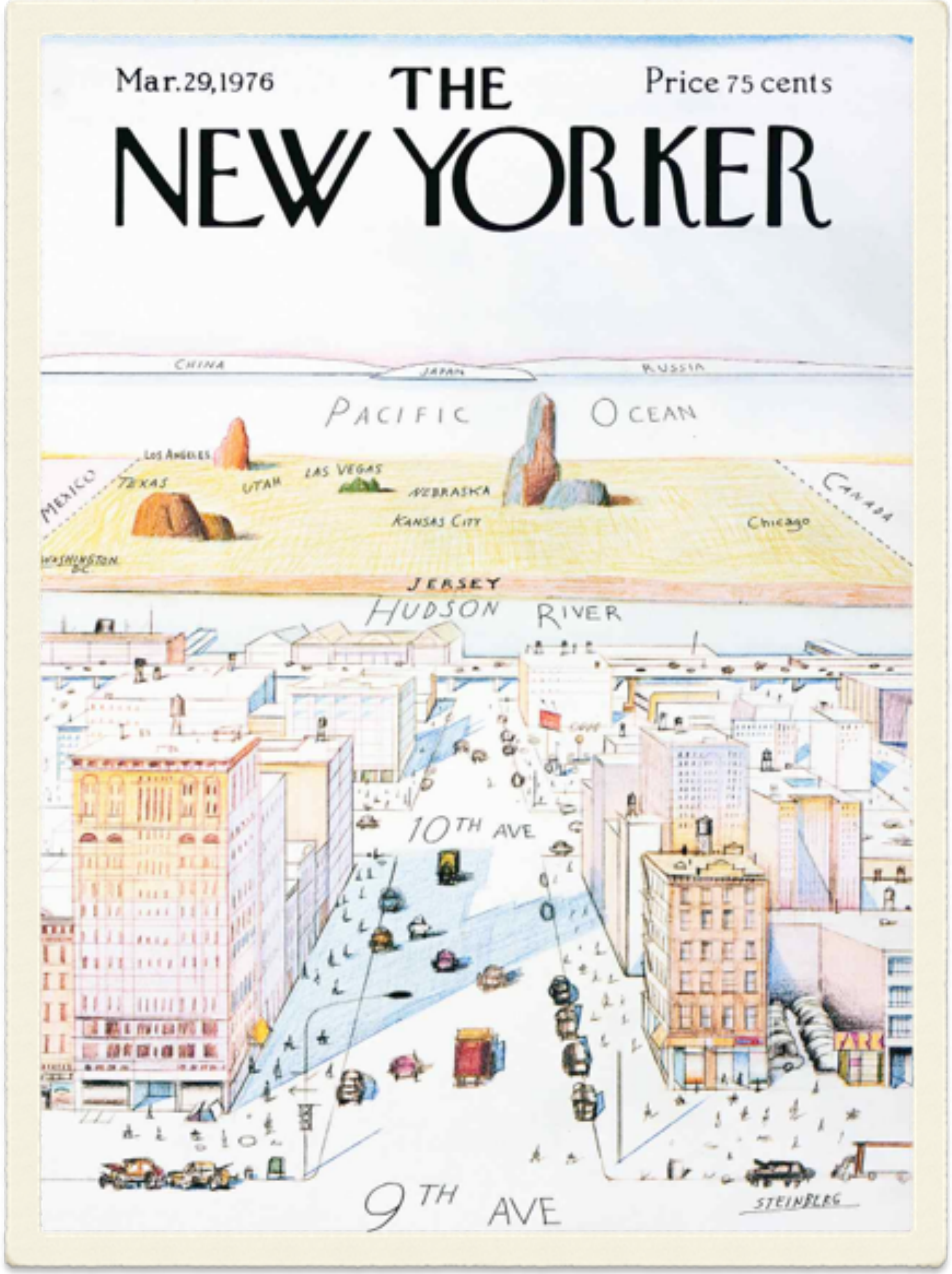
Nate Foster
Cornell University

Prelude

Disclaimer: A Survivor’s Story



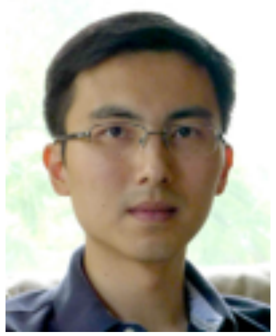
. . . Geographically Biased



…And A Personal Story

Acknowledgments

Kim
Bruce



Phillip
Guo

Nick
Feamster

Matt
Might

Andrew
Myers

Benjamin
Pierce

Yannis
Smaragdakis

Ross
Tate

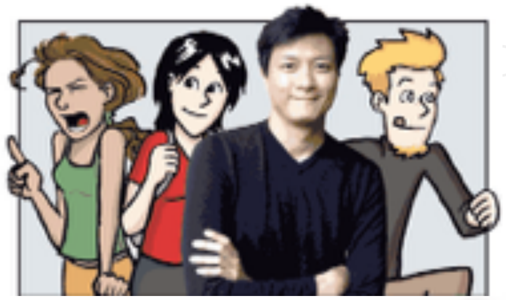
Acknowledgments

Kim
Bruce

Phill
Gu

ck
ster

Matt
Might



Jorge
Cham

Andrew
Myers

Benjamin
Pierce

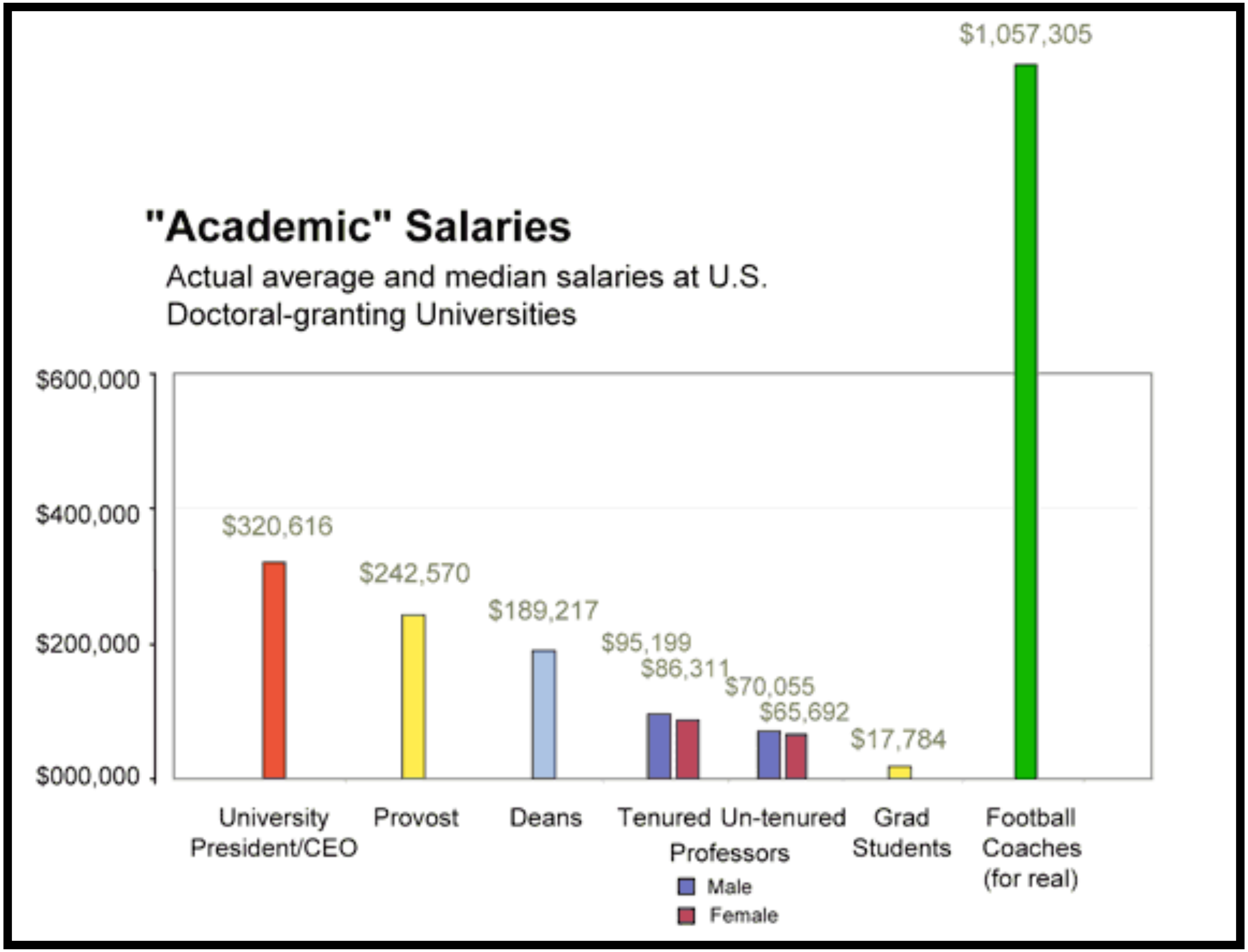
Yannis
Smaragdakis

Ross
Tate

Why do a PhD?

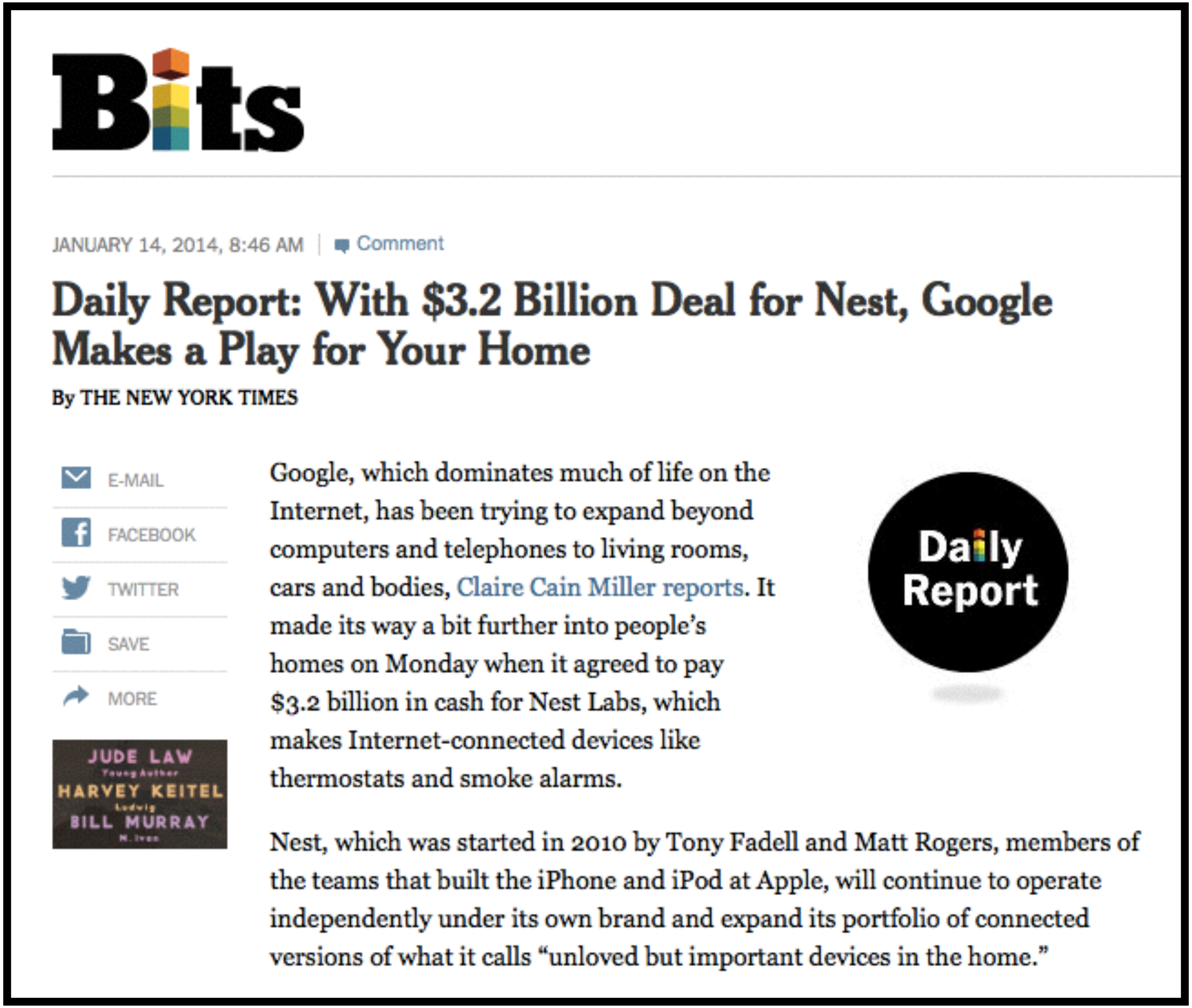
Money

Money



"Academic" Salaries
Actual average and median salaries at U.S. Doctoral-granting Universities
$600,000
$400,000
$200,000
$000,000
$320,616
$242,570
$189,217
$95,199
$86,311
$70,055
$65,692
$17,784
$1,057,305
University President/CEO
Provost
Deans
Tenured
Un-tenured
Professors
🟦 Male
🟥 Female
Grad Students
Football Coaches (for real)

Money



Bits

JANUARY 14, 2014, 8:46 AM | Comment

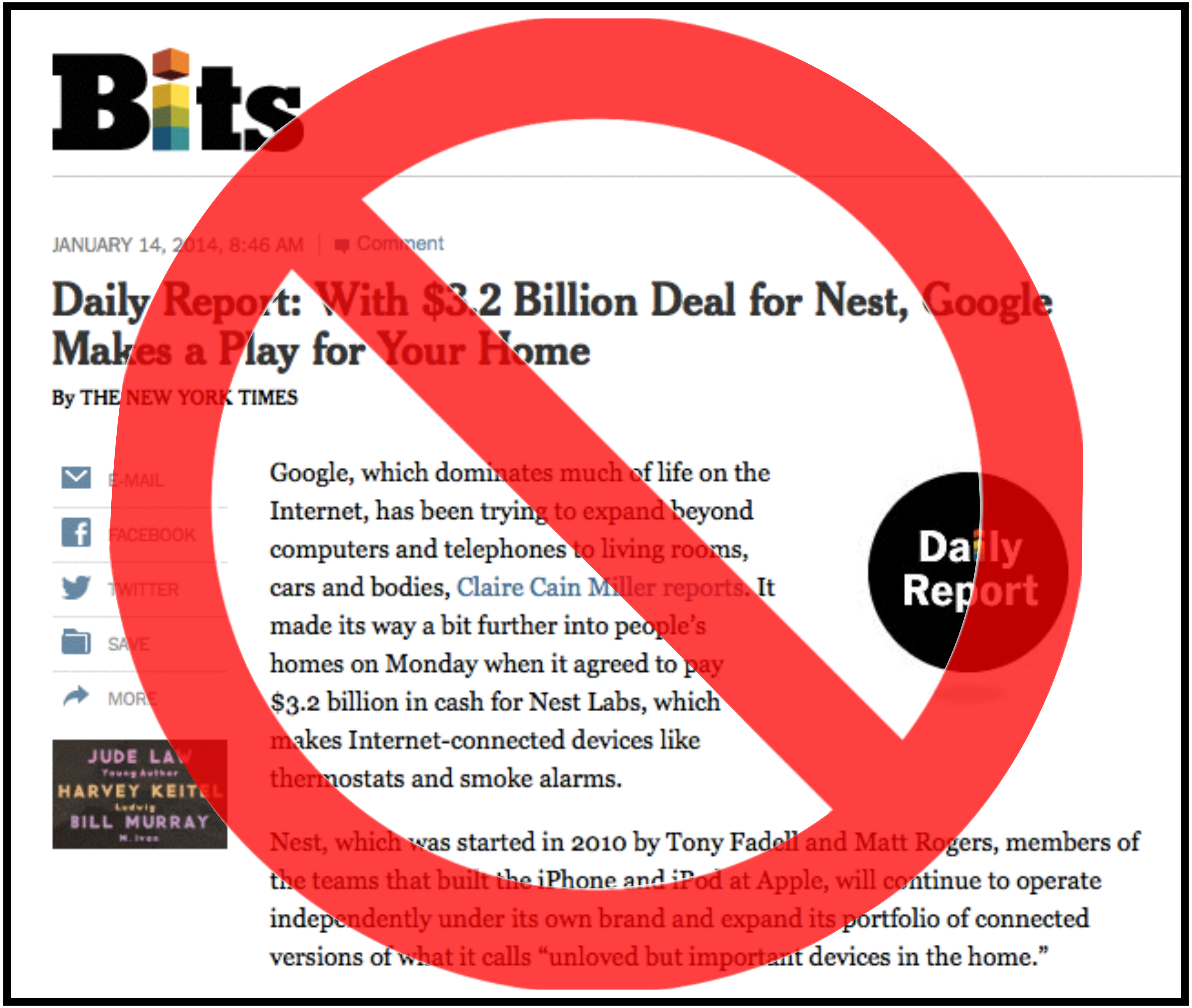
Daily Report: With $3.2 Billion Deal for Nest, Google Makes a Play for Your Home

By THE NEW YORK TIMES

Google, which dominates much of life on the Internet, has been trying to expand beyond computers and telephones to living rooms, cars and bodies, Claire Cain Miller reports. It made its way a bit further into people's homes on Monday when it agreed to pay $3.2 billion in cash for Nest Labs, which makes Internet-connected devices like thermostats and smoke alarms.

Nest, which was started in 2010 by Tony Fadell and Matt Rogers, members of the teams that built the iPhone and iPod at Apple, will continue to operate independently under its own brand and expand its portfolio of connected versions of what it calls "unloved but important devices in the home."

Money



JANUARY 14, 2014, 8:46 AM | Comment

Daily Report: With $3.2 Billion Deal for Nest, Google Makes a Play for Your Home

By THE NEW YORK TIMES

Google, which dominates much of life on the Internet, has been trying to expand beyond computers and telephones to living rooms, cars and bodies, Claire Cain Miller reports. It made its way a bit further into people's homes on Monday when it agreed to pay $3.2 billion in cash for Nest Labs, which makes Internet-connected devices like thermostats and smoke alarms.

Nest, which was started in 2010 by Tony Fadell and Matt Rogers, members of the teams that built the iPhone and iPod at Apple, will continue to operate independently under its own brand and expand its portfolio of connected versions of what it calls "unloved but important devices in the home."

Impress People

Impress People

Impress People

Impress People

Nobody will ever call you "doctor" except your mom…

… you will get asked "when are you finally going to graduate?"

Impress People

Nobody will ever call you “doctor” except your mom…

… you will get asked “when are you finally going to graduate?”

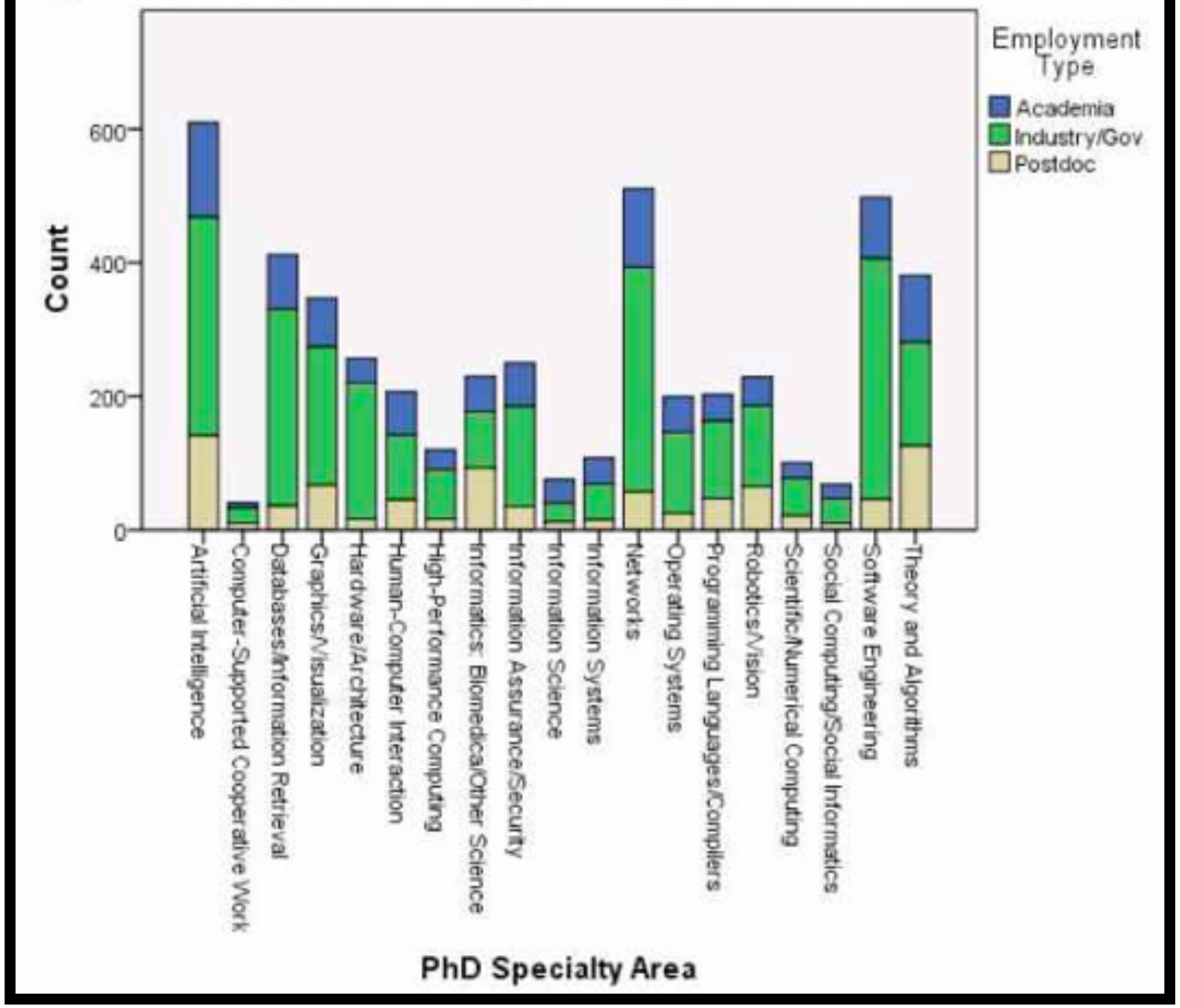
Undergrad Was Fun

Undergrad Was Fun

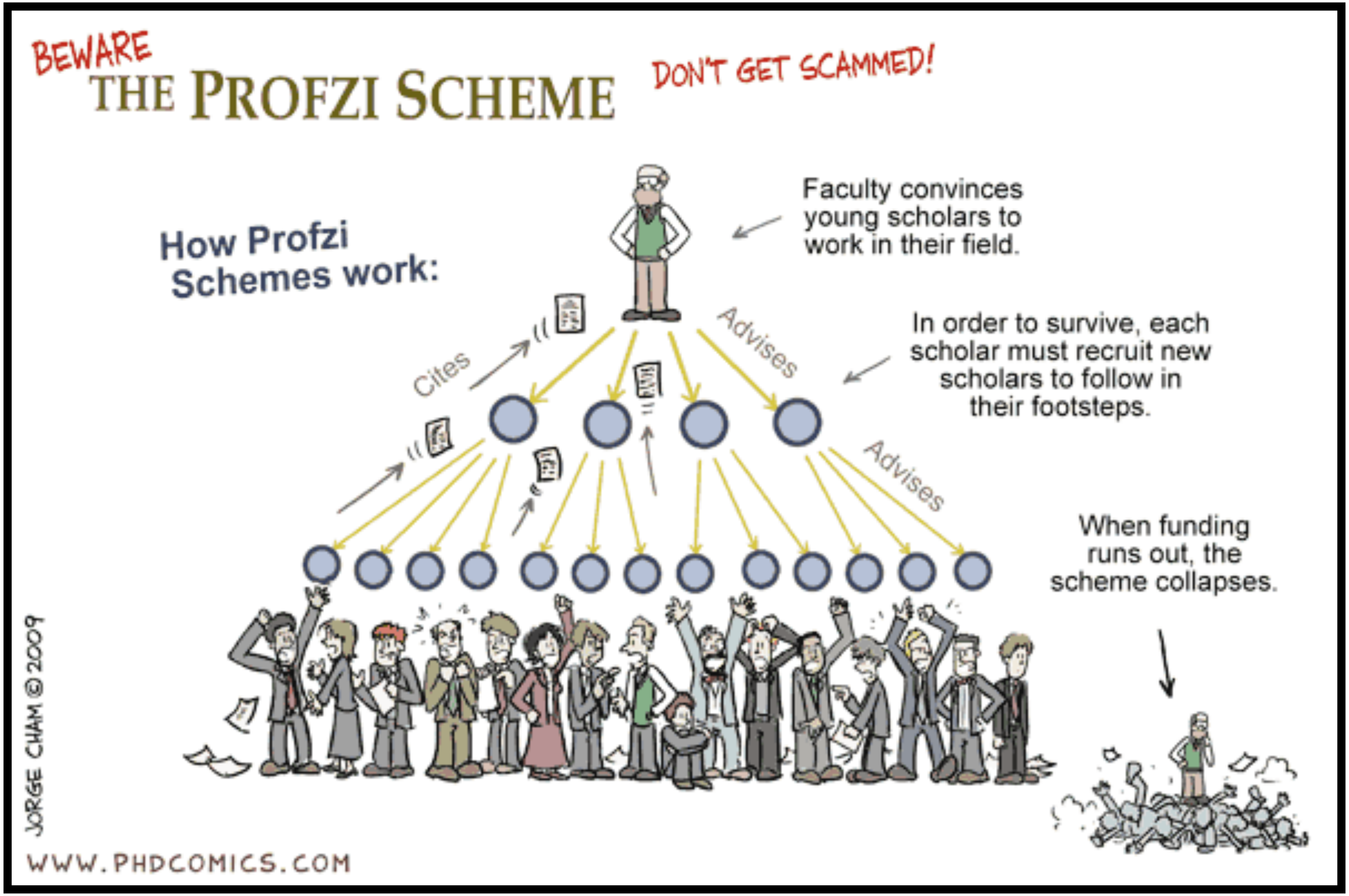
Undergrad Was Fun

Become a Professor

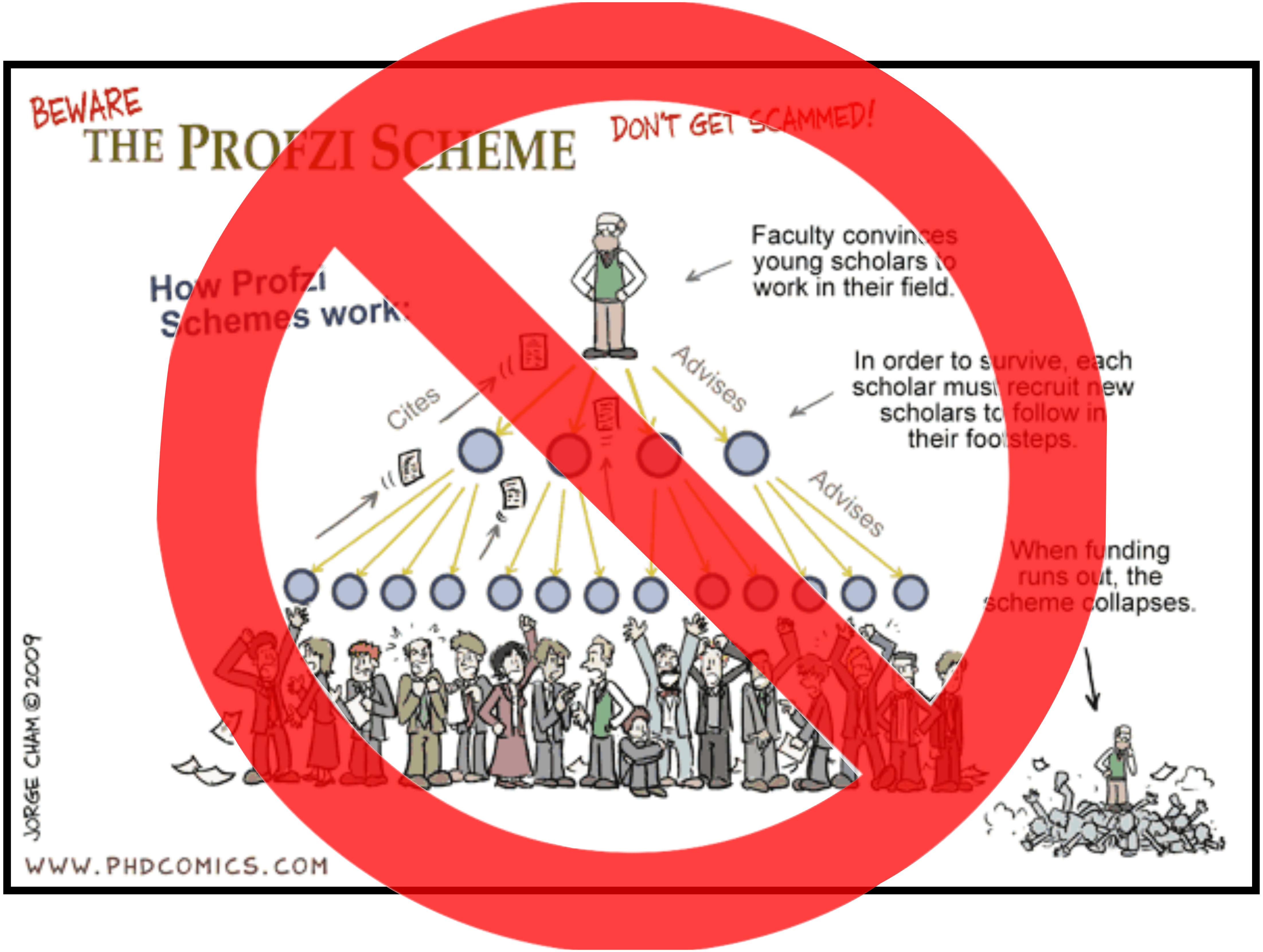
Figure 1. Employment of PhD Specialties from 2008-2011.



Become a Professor

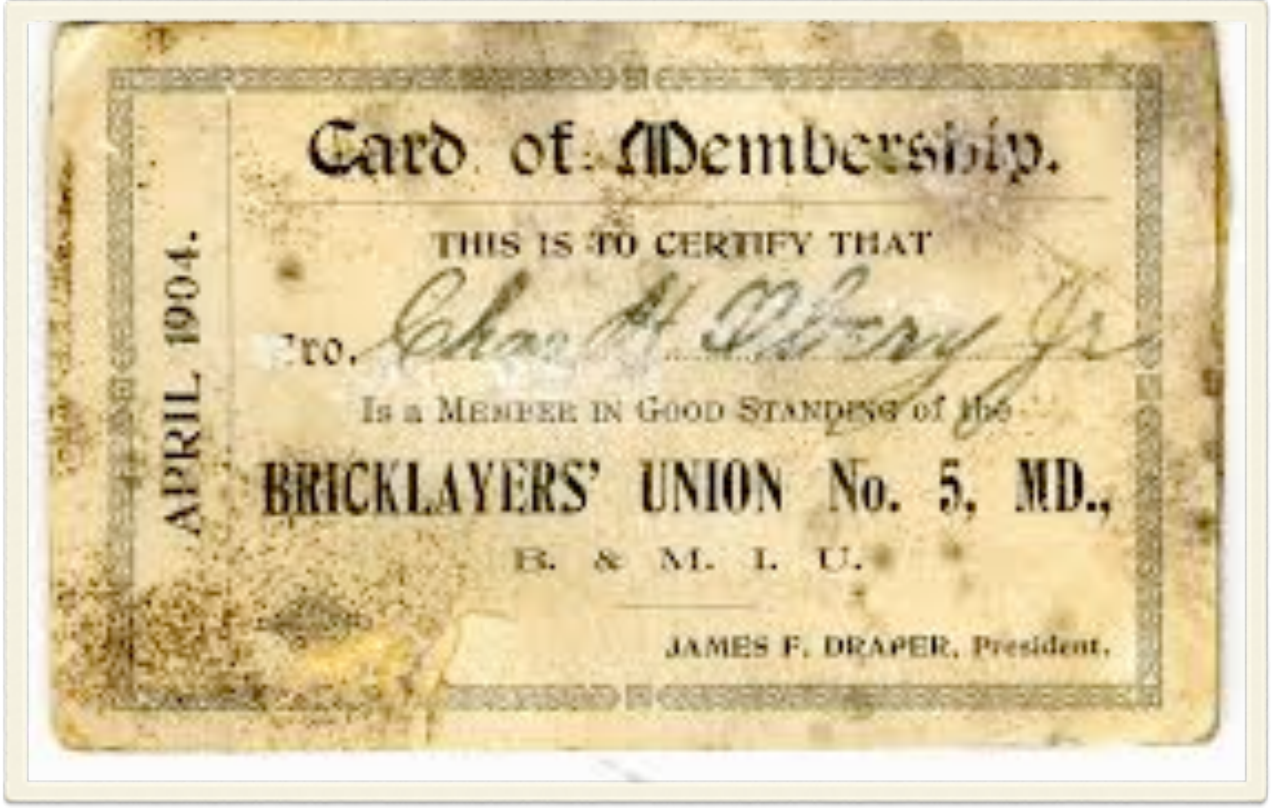

BEWARE
THE PROFZI SCHEME
DON'T GET SCAMMED!
How Profzi Schemes work:
Faculty convinces young scholars to work in their field.
In order to survive, each scholar must recruit new scholars to follow in their footsteps.
When funding runs out, the scheme collapses.
Cites
Advises
Advises
JORGE CHAM © 2009
WWW.PHDCOMICS.COM

Become a Professor


BEWARE
THE PROFZI SCHEME
DON'T GET SCAMMED!
How Profzi Schemes work:
Faculty convinces young scholars to work in their field.
Cites
Advises
In order to survive, each scholar must recruit new scholars to follow in their footsteps.
Advises
When funding runs out, the scheme collapses.
JORGE CHAM © 2009
WWW.PHDCOMICS.COM

Opens Opportunities

Opens Opportunities



APRIL 1904.

Card of Membership.

THIS IS TO CERTIFY THAT

Bro. Chas H [illegible] Jr

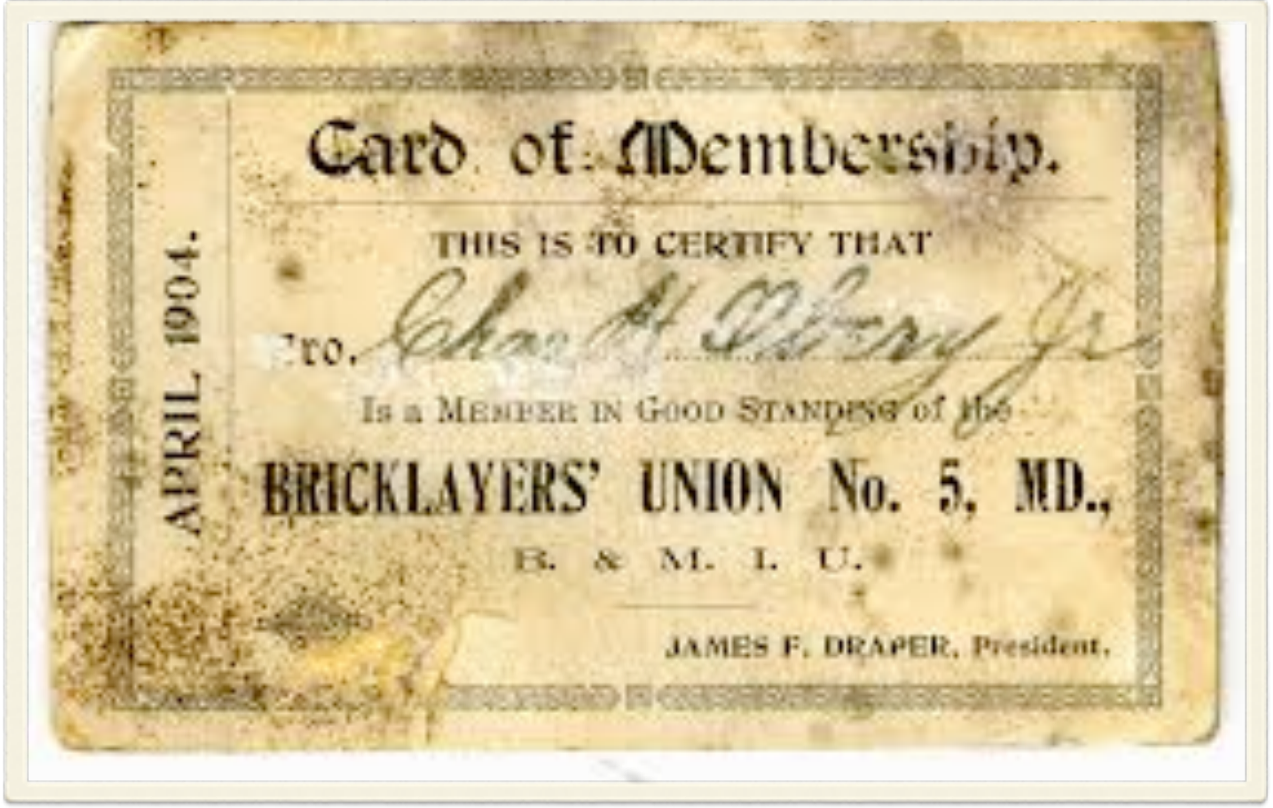
Is a Member in Good Standing of the

BRICKLAYERS' UNION No. 5, MD.,

B. & M. I. U.

JAMES F. DRAPER, President.

Opens Opportunities



APRIL 1904.

Card of Membership.

THIS IS TO CERTIFY THAT

Bro. Chas H [illegible] Jr

Is a Member in Good Standing of the

BRICKLAYERS' UNION No. 5, MD.,

B. & M. I. U.

JAMES F. DRAPER, President.

- Universities
- Civil service
- National labs
- Corporate research labs
- Research and development
- Advanced engineering

Freedom!

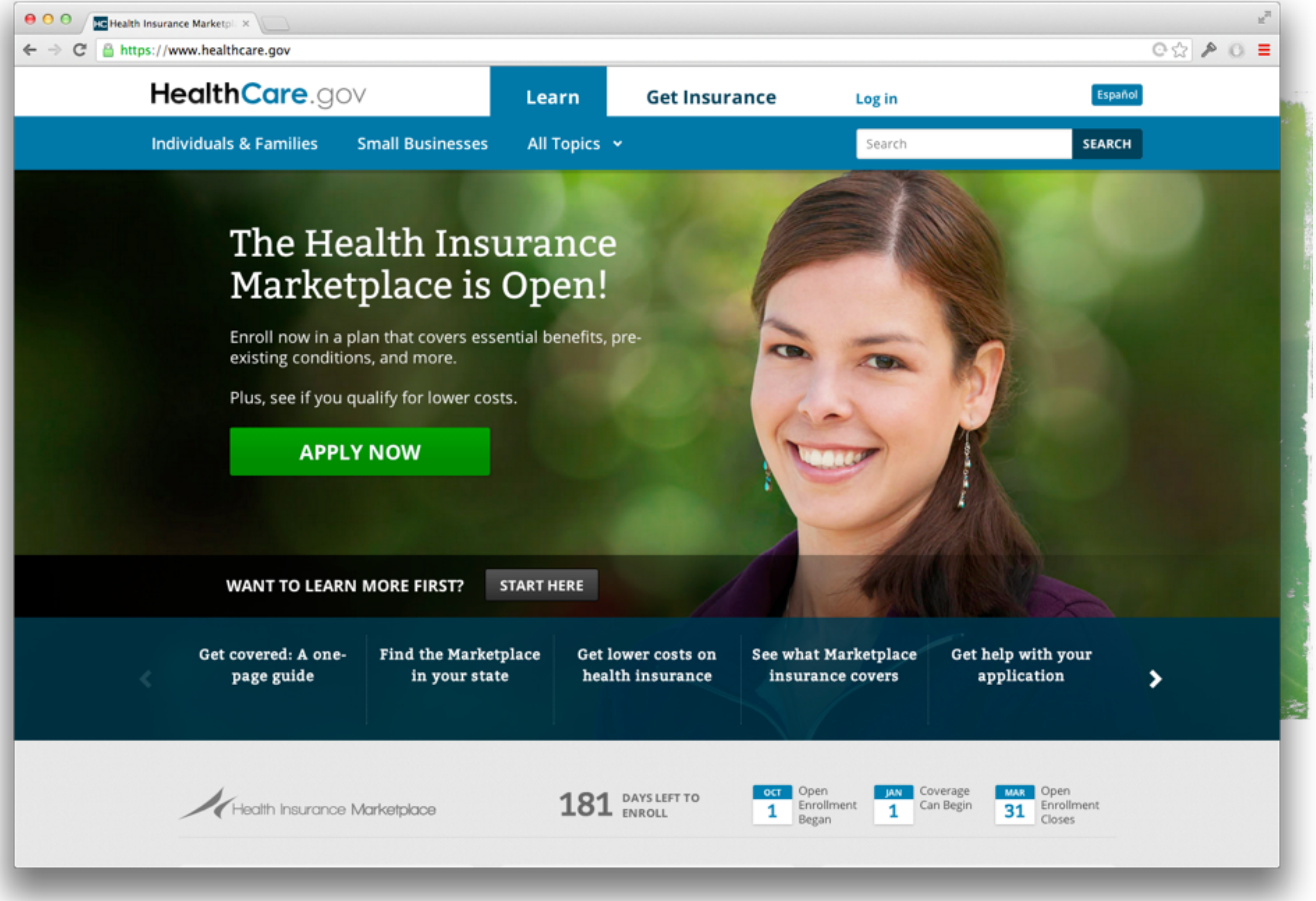
Work on Big Problems

Many grand challenges:

- Verification
- Security and Privacy
- Fault tolerance
- Distributed computing
- Energy-efficient computing
- Systems biology
- …

A lot of them require the tools and techniques of the POPL community!

Work on Big Problems



The Health Insurance Marketplace is Open!

Enroll now in a plan that covers essential benefits, pre-existing conditions, and more.

Plus, see if you qualify for lower costs.

APPLY NOW

WANT TO LEARN MORE FIRST? START HERE

Get covered: A one-page guide

Find the Marketplace in your state

Get lower costs on health insurance

See what Marketplace insurance covers

Get help with your application

Health Insurance Marketplace

181 DAYS LEFT TO ENROLL

OCT 1 Open Enrollment Began

JAN 1 Coverage Can Begin

MAR 31 Open Enrollment Closes

What is a PhD?

Comparing Degrees

Comparing Degrees

High school: basic knowledge of a broad range of topics

Comparing Degrees

High school: basic knowledge of a broad range of topics

Undergrad: broad knowledge and specialization in a field

Comparing Degrees

High school: basic knowledge of a broad range of topics

Undergrad: broad knowledge and specialization in a field

Professional: advanced knowledge and practical skills

Comparing Degrees

High school: basic knowledge of a broad range of topics

Undergrad: broad knowledge and specialization in a field

Professional: advanced knowledge and practical skills

}

Well-defined requirements
High rate of completion

Comparing Degrees

High school: basic knowledge of a broad range of topics

Undergrad: broad knowledge and specialization in a field

Professional: advanced knowledge and practical skills

}

Well-defined requirements
High rate of completion

PhD: advanced knowledge + a research contribution

Comparing Degrees

High school: basic knowledge of a broad range of topics

Undergrad: broad knowledge and specialization in a field

Professional: advanced knowledge and practical skills

PhD: advanced knowledge + a research contribution

Transformation

Transformation

How to do a PhD

Step 0: Pick an Institution

Easy :-)

CORNELL UNIVERSITY FOUNDED A.D. 1865

Step 0: Pick an Institution

Very important:

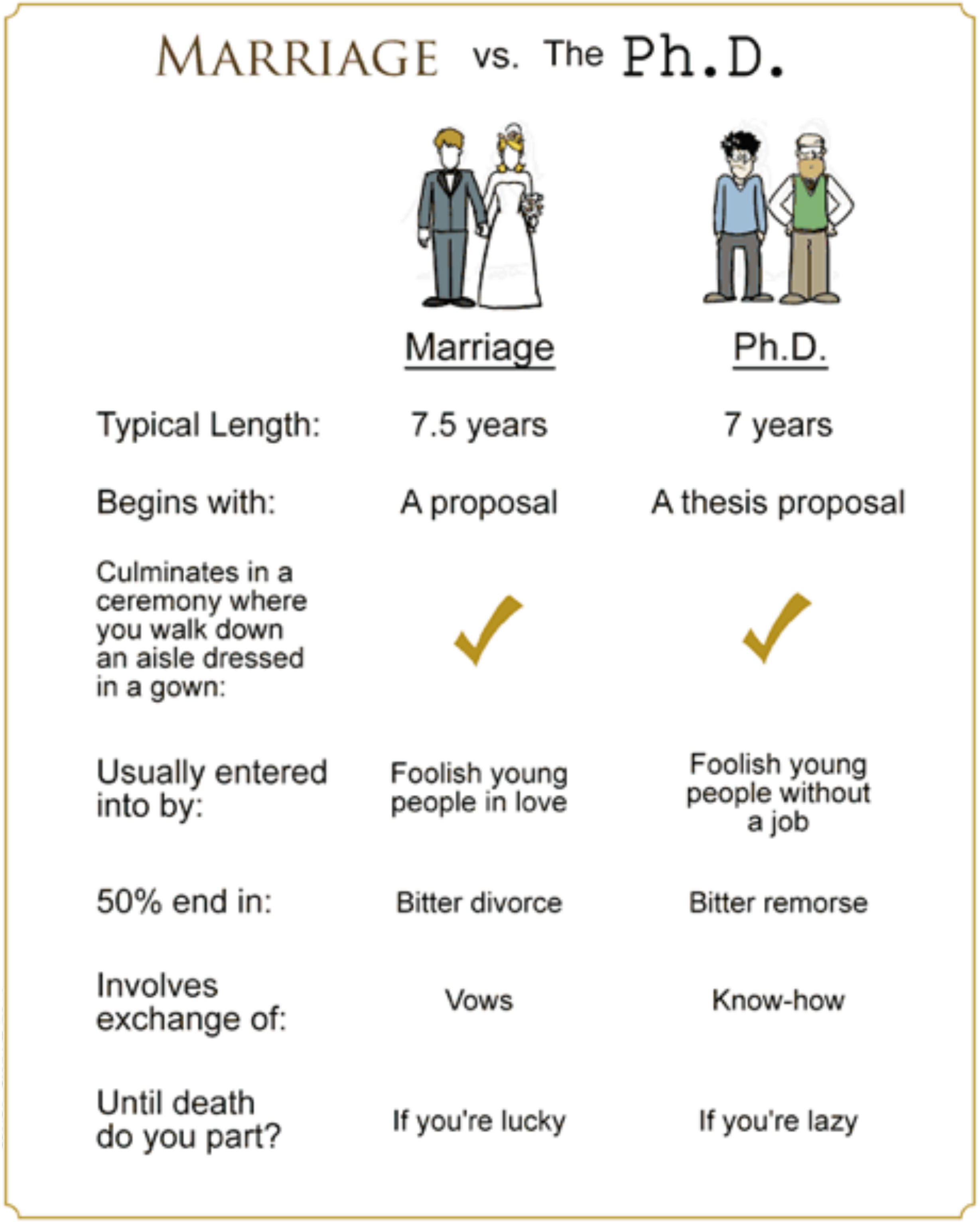
- Advisor
- Opportunity
- Peers

Typically less important:

- Finances
- Institution
- Location

Advice:

- Apply to the top programs in CS and your area
- Talk to current students
- Look at recent results



MARRIAGE vs. The Ph.D.

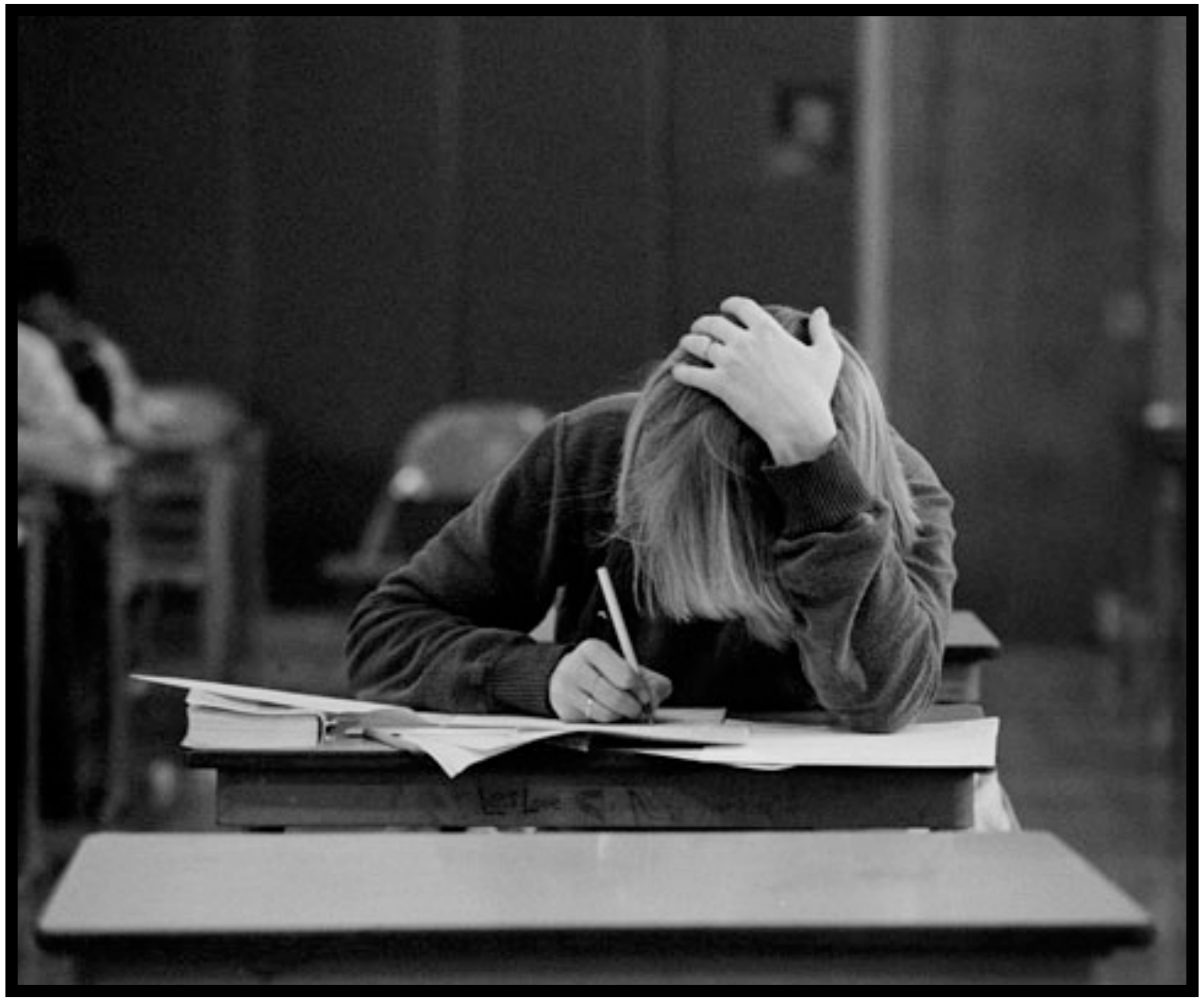
	Marriage	Ph.D.
Typical Length:	7.5 years	7 years
Begins with:	A proposal	A thesis proposal
Culminates in a ceremony where you walk down an aisle dressed in a gown:	✓	✓
Usually entered into by:	Foolish young people in love	Foolish young people without a job
50% end in:	Bitter divorce	Bitter remorse
Involves exchange of:	Vows	Know-how
Until death do you part?	If you're lucky	If you're lazy

Step 1: Preliminaries

Step 1: Preliminaries

Typical requirements

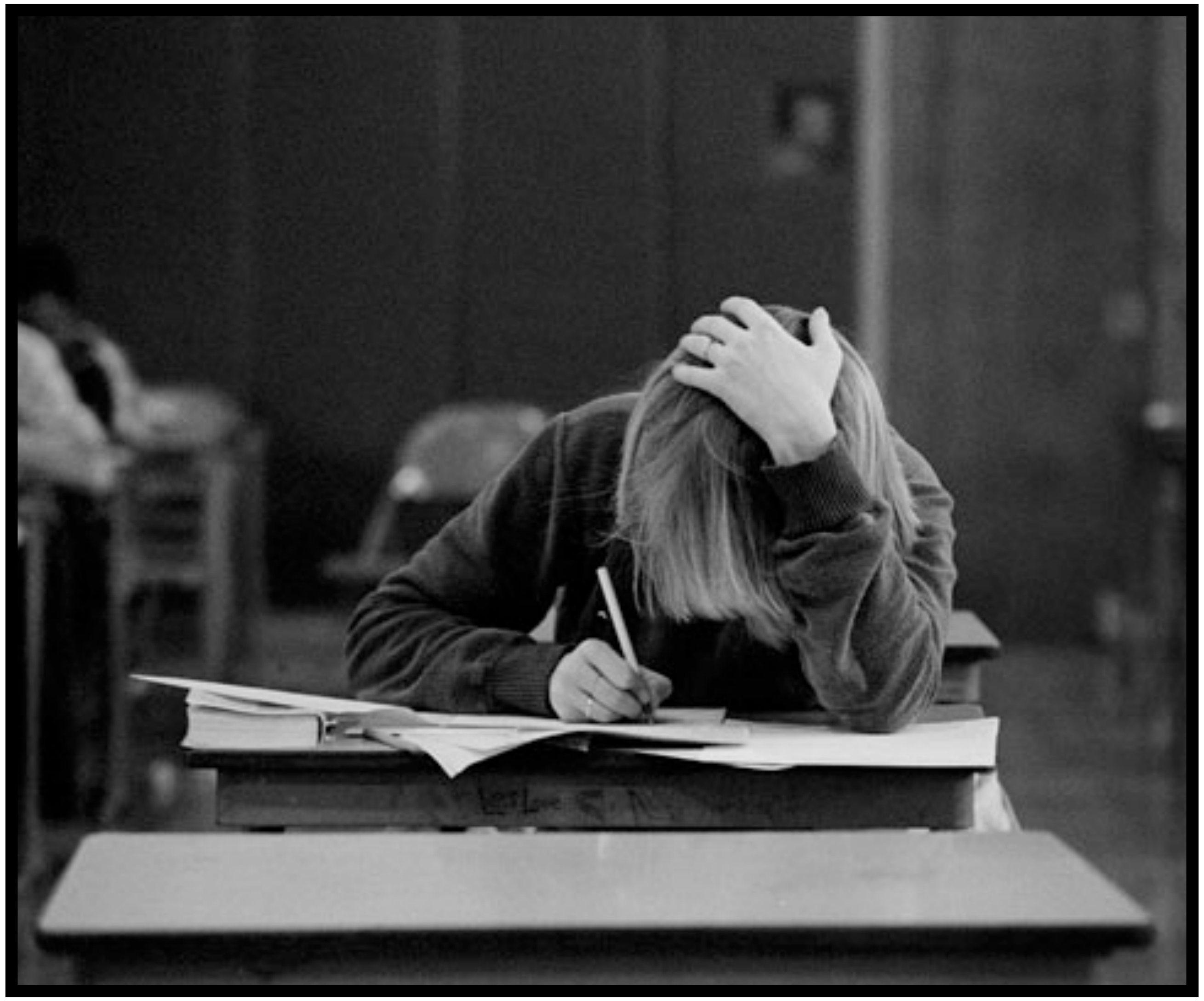
- Demonstrate broad knowledge of Computer Science at the advanced undergrad level
- Demonstrate specific knowledge of your field
- Often assessed as a set of written/oral exams



Step 1: Preliminaries

Typical requirements

- Demonstrate broad knowledge of Computer Science at the advanced undergrad level
- Demonstrate specific knowledge of your field
- Often assessed as a set of written/oral exams

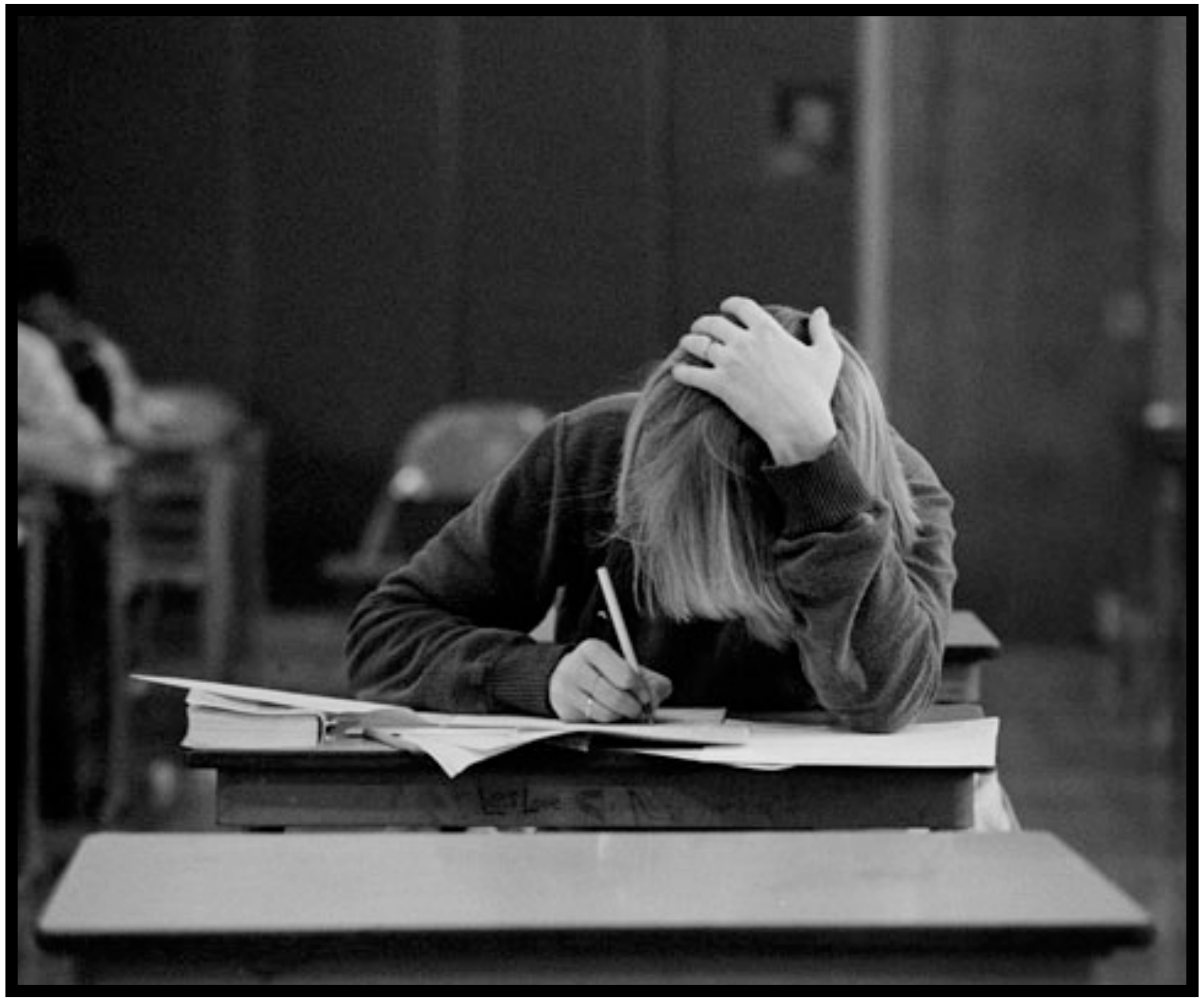


Designed to be straightforward, but can be difficult when juggling living in a new town, teaching, fellowships, research, etc.

Step 1: Preliminaries

Typical requirements

- Demonstrate broad knowledge of Computer Science at the advanced undergrad level
- Demonstrate specific knowledge of your field
- Often assessed as a set of written/oral exams



Designed to be straightforward, but can be difficult when juggling living in a new town, teaching, fellowships, research, etc.

My emotions: extreme stress!

Step 2: Initial Results

Step 2: Initial Results

Ideally, can get involved with research early on…

Step 2: Initial Results

Ideally, can get involved with research early on…

…many advisors will point junior students down a clear path

Step 2: Initial Results

Ideally, can get involved with research early on…

…many advisors will point junior students down a clear path

Combinators for Bi-Directional Tree Transformations: A Linguistic Approach to the View Update Problem

J. NATHAN FOSTER
University of Pennsylvania
MICHAEL B. GREENWALD
Bell Labs, Lucent Technologies
JONATHAN T. MOORE
University of Pennsylvania
BENJAMIN C. PIERCE
University of Pennsylvania
ALAN SCHMITT
INRIA Rhône-Alpes

We propose a novel approach to the view update problem for tree-structured data: a domain-specific programming language in which all expressions denote bi-directional transformations on trees. In one direction, these transformations—dubbed lenses—map a "concrete" tree into a simplified "abstract view"; in the other, they map a modified abstract view, together with the original concrete tree, to a correspondingly modified concrete tree. Our design emphasizes both robustness and ease of use, guaranteeing strong well-behavedness and totality properties for well-typed lenses.

We begin by identifying a natural mathematical space of well-behaved bi-directional transformations over arbitrary structures, studying definedness and continuity in this setting. We then instantiate this semantic framework in the form of a collection of lens combinators that can be assembled to describe bi-directional transformations on trees. These combinators include familiar constructs from functional programming (composition, mapping, projection, conditionals, recursion) together with some novel primitives for manipulating trees (splitting, pruning, copying, merging, etc.). We illustrate the expressiveness of these combinators by developing a number of bi-directional list-processing transformations as derived forms. An extended example shows how our combinators can be used to define a lens that translates between a native HTML representation of browser bookmarks and a generic abstract bookmark format.

Categories and Subject Descriptors: D.3.2 [**Programming Languages**]: Language Classifications—*Specialized application languages*

General Terms: Languages

Additional Key Words and Phrases: Bi-directional programming, Harmony, XML, lenses, view update problem

ACM Transactions on Programming Languages and Systems, Vol. TBD, No. TDB, Month Year, Pages 1–??.

Step 2: Initial Results

Ideally, can get involved with research early on…

…many advisors will point junior students down a clear path

Goal: a taste of research

Model: apprenticeship

Role: provide thrust

My emotions: excitement and perhaps a bit of shell shock

Combinators for Bi-Directional Tree Transformations: A Linguistic Approach to the View Update Problem

J. NATHAN FOSTER
University of Pennsylvania
MICHAEL B. GREENWALD
Bell Labs, Lucent Technologies
JONATHAN T. MOORE
University of Pennsylvania
BENJAMIN C. PIERCE
University of Pennsylvania
ALAN SCHMITT
INRIA Rhône-Alpes

We propose a novel approach to the view update problem for tree-structured data: a domain-specific programming language in which all expressions denote bi-directional transformations on trees. In one direction, these transformations—dubbed lenses—map a "concrete" tree into a simplified "abstract view"; in the other, they map a modified abstract view, together with the original concrete tree, to a correspondingly modified concrete tree. Our design emphasizes both robustness and ease of use, guaranteeing strong well-behavedness and totality properties for well-typed lenses.

We begin by identifying a natural mathematical space of well-behaved bi-directional transformations over arbitrary structures, studying definedness and continuity in this setting. We then instantiate this semantic framework in the form of a collection of lens combinators that can be assembled to describe bi-directional transformations on trees. These combinators include familiar constructs from functional programming (composition, mapping, projection, conditionals, recursion) together with some novel primitives for manipulating trees (splitting, pruning, copying, merging, etc.). We illustrate the expressiveness of these combinators by developing a number of bi-directional list-processing transformations as derived forms. An extended example shows how our combinators can be used to define a lens that translates between a native HTML representation of browser bookmarks and a generic abstract bookmark format.

Categories and Subject Descriptors: D.3.2 [**Programming Languages**]: Language Classifications—*Specialized application languages*

General Terms: Languages

Additional Key Words and Phrases: Bi-directional programming, Harmony, XML, lenses, view update problem

Permission to make digital/hard copy of all or part of this material without fee for personal or classroom use provided that the copies are not made or distributed for profit or commercial advantage, the ACM copyright/server notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists requires prior specific permission and/or a fee.
© 2007 ACM XXX-XXX/XX/XXXX-XXXX $XX.XX

ACM Transactions on Programming Languages and Systems, Vol. TBD, No. TDB, Month Year, Pages 1–??.

Step 3: Iterate

Then one repeats the process
(ideally several times) showing
that the first result was not a fluke

Step 3: Iterate

Then one repeats the process (ideally several times) showing that the first result was not a fluke

Boomerang: Resourceful Lenses for String Data

Aaron Bohannon, University of Pennsylvania
J. Nathan Foster, University of Pennsylvania
Benjamin C. Pierce, University of Pennsylvania
Alexandre Pilkiewicz, École Polytechnique
Alan Schmitt, INRIA Rhône-Alpes

Abstract

A *lens* is a bidirectional program. When read from left to right, it denotes an ordinary function that maps inputs to outputs. When read from right to left, it denotes an "update translator" that takes an input together with an updated output and produces a new input that reflects the update. Many variants of this idea have been explored in the literature, but none deal fully with *ordered* data. If, for example, an update changes the order of a list in the output, the items in the output list and the chunks of the input that generated them can be misaligned, leading to lost or corrupted data.

We attack this problem in the context of bidirectional transformations over strings, the primordial ordered data type. We first propose a collection of bidirectional *string lens combinators*, based on familiar operations on regular transducers (union, concatenation, Kleene-star) and with a type system based on regular expressions. We then design a new semantic space of *dictionary lenses*, enriching the lenses of Foster et al. (2007b) with support for two additional combinators for marking "reorderable chunks" and their keys. To demonstrate the effectiveness of these primitives, we describe the design and implementation of Boomerang, a full-blown *bidirectional programming language* with dictionary lenses at its core. We have used Boomerang to build transformers for complex real-world data formats including the SwissProt genomic database.

We formalize the essential property of *resourcefulness*—the correct use of keys to associate chunks in the input and output—by defining a refined semantic space of *quasi-oblivious* lenses. Several previously studied properties of lenses turn out to have compact characterizations in this space.

Categories and Subject Descriptors D.3.2 [*Programming Languages*]: Language Classifications—Specialized application languages

General Terms Languages, Design, Theory

Keywords Bidirectional languages, lenses, view update problem, regular string transducers, regular types

1. Introduction

> *"The art of progress is to preserve order amid change and to preserve change amid order."*
> —A N Whitehead

Most of the time, we use programs in just one direction, from input to output. But sometimes, having computed an output, we need to be able to update this output and then "calculate backwards" to find a correspondingly updated input. The problem of writing such bidirectional transformations arises in a multitude of domains, including data converters and synchronizers, parsers and pretty printers, picklers and unpicklers, structure editors, constraint maintainers for user interfaces, and, of course, in databases, where it is known as the view update problem. Our own study of bidirectional transformations is motivated by their application in a generic synchronization framework, called Harmony, where they are used to synchronize heterogeneous data formats against each other (Pierce et al. 2006; Foster et al. 2007a).

The naive way to write a bidirectional transformation is simply to write two separate functions in any language you like and check (by hand) that they fit together in some appropriate sense—e.g., that composing them yields the identity function. However, this approach is unsatisfying for all but the simplest examples. For one thing, verifying that the two functions fit together in this way requires intricate reasoning about their behaviors. Moreover, it creates a maintenance nightmare: both functions will embody the structure that the input and output schemas have in common, so changes to the schemas will require coordinated changes to both. (See the appendix for a concrete example.)

A better alternative is to design a notation in which both transformations can be described at the same time—i.e., a *bidirectional programming language*. In a bidirectional language, every expression, when read from left to right, denotes a function mapping inputs to outputs; when read from right to left, the same expression denotes a function mapping an updated output together with an original input to an appropriately updated version of the input. Not only does this eliminate code duplication; it also eliminates paper-and-pencil proofs that the two transformations fit together properly: we can design the language to guarantee it.

Many different bidirectional languages have been proposed, including constraint maintainers (Meertens 1998), pickler combinators (Kennedy 2004), embedding projection pairs (Benton 2005; Ramsey 2003), X/Inv (Hu et al. 2004), XSugar (Brabrand et al. 2007), biXid (Kawanaka and Hosoya 2006), PADS (Fisher and Gruber 2005), and bi-arrows (Alimarine et al. 2005). The design challenge for all these languages lies in striking a balance between expressiveness and reliability—making strong promises to programmers about the joint behavior of pairs of transformations and the conditions under which they can safely be used.

Lenses The language described in this paper is an extension of our previous presentation on lenses (Foster et al. 2007b)—called *basic lenses* here.[1] Among the bidirectional languages listed above,

POPL'08, January 7–12, 2008, San Francisco, California, USA.
Copyright © 2008 ACM 978-1-59593-689-9/08/0001. . . $5.00

[1] Readers familiar with the original presentation will notice some minor differences. First we handle situations where an element of C must be created from an element of A using a create function instead of enriching C with a special element Ω and using put. Second, as we are not considering lenses defined by recursion, we take the components of lenses to be total functions rather than defining lenses with partial components and establishing totality later. Finally, we take the behavioral laws as part of the fundamental

Step 3: Iterate

Then one repeats the process (ideally several times) showing that the first result was not a fluke

Goal: demonstrate ability to repeatedly produce high-quality research results

Model: junior partner

Role: discover technical insights

My emotions: frustration

Boomerang: Resourceful Lenses for String Data

Aaron Bohannon, University of Pennsylvania
J. Nathan Foster, University of Pennsylvania
Benjamin C. Pierce, University of Pennsylvania
Alexandre Pilkiewicz, École Polytechnique
Alan Schmitt, INRIA Rhône-Alpes

Abstract

A lens is a bidirectional program. When read from left to right, it denotes an ordinary function that maps inputs to outputs. When read from right to left, it denotes an "update translator" that takes an input together with an updated output and produces a new input that reflects the update. Many variants of this idea have been explored in the literature, but none deal fully with ordered data. If, for example, an update changes the order of a list in the output, the items in the output list and the chunks of the input that generated them can be misaligned, leading to lost or corrupted data.

We attack this problem in the context of bidirectional transformations over strings, the primordial ordered data type. We first propose a collection of bidirectional *string lens combinators*, based on familiar operations on regular transducers (union, concatenation, Kleene-star) and with a type system based on regular expressions. We then design a new semantic space of *dictionary lenses*, enriching the lenses of Foster et al. (2007b) with support for two additional combinators for marking "reorderable chunks" and their keys. To demonstrate the effectiveness of these primitives, we describe the design and implementation of Boomerang, a full-blown *bidirectional programming language* with dictionary lenses at its core. We have used Boomerang to build transformers for complex real-world data formats including the SwissProt genomic database.

We formalize the essential property of *resourcefulness*—the correct use of keys to associate chunks in the input and output—by defining a refined semantic space of *quasi-oblivious lenses*. Several previously studied properties of lenses turn out to have compact characterizations in this space.

Categories and Subject Descriptors D.3.2 [*Programming Languages*]: Language Classifications—Specialized application languages

General Terms Languages, Design, Theory

Keywords Bidirectional languages, lenses, view update problem, regular string transducers, regular types

1. Introduction

"The art of progress is to preserve order amid change and to preserve change amid order"
—A N Whitehead

Most of the time, we use programs in just one direction, from input to output. But sometimes, having computed an output, we need to be able to update this output and then "calculate backwards" to find a correspondingly updated input. The problem of writing such bidirectional transformations arises in a multitude of domains, including data converters and synchronizers, parsers and pretty printers, picklers and unpicklers, structure editors, constraint maintainers for user interfaces, and, of course, in databases, where it is known as the view update problem. Our own study of bidirectional transformations is motivated by their application in a generic synchronization framework, called Harmony, where they are used to synchronize heterogeneous data formats against each other (Pierce et al. 2006; Foster et al. 2007a).

The naive way to write a bidirectional transformation is simply to write two separate functions in any language you like and check (by hand) that they fit together in some appropriate sense—e.g., that composing them yields the identity function. However, this approach is unsatisfying for all but the simplest examples. For one thing, verifying that the two functions fit together in this way requires intricate reasoning about their behaviors. Moreover, it creates a maintenance nightmare: both functions will embody the structure that the input and output schemas have in common, so changes to the schemas will require coordinated changes to both. (See the appendix for a concrete example.)

A better alternative is to design a notation in which both transformations can be described at the same time—i.e., a *bidirectional programming language*. In a bidirectional language, every expression, when read from left to right, denotes a function mapping inputs to outputs; when read from right to left, the same expression denotes a function mapping an updated output together with an original input to an appropriately updated version of the input. Not only does this eliminate code duplication; it also eliminates paper-and-pencil proofs that the two transformations fit together properly: we can design the language to guarantee it.

Many different bidirectional languages have been proposed, including constraint maintainers (Meertens 1998), pickler combinators (Kennedy 2004), embedding projection pairs (Benton 2005; Ramsey 2003), XSugar (Hu et al. 2004), XSugar (Brabrand et al. 2007), biXid (Kawanaka and Hosoya 2006), PADS (Fisher and Gruber 2005), and bi-arrows (Alimarine et al. 2005). The design challenge for all these languages lies in striking a balance between expressiveness and reliability—making strong promises to programmers about the joint behavior of pairs of transformations and the conditions under which they can safely be used.

Lenses The language described in this paper is an extension of our previous presentation on lenses (Foster et al. 2007b)—called *basic lenses* here.[1] Among the bidirectional languages listed above,

POPL'08, January 7–12, 2008, San Francisco, California, USA.
Copyright © 2008 ACM 978-1-59593-689-9/08/0001. . . $5.00

[1] Readers familiar with the original presentation will notice some minor differences. First we handle situations where an element of C must be created from an element of A using a create function instead of enriching C with a special element Ω and using *put*. Second, as we are not considering lenses defined by recursion, we take the components of lenses to be total functions rather than defining lenses with partial components and establishing totality later. Finally, we take the behavioral laws as part of the fundamental

Step 4: Independence

Eventually, begin to show leadership and break ground in new directions

Step 4: Independence

Eventually, begin to show leadership and break ground in new directions

Updatable Security Views

J. Nathan Foster Benjamin C. Pierce Steve Zdancewic
University of Pennsylvania

Abstract

Security views are a flexible and effective mechanism for controlling access to confidential information. Rather than allowing untrusted users to access source data directly, they are instead provided with a restricted view, from which all confidential information has been removed. The program that generates the view effectively embodies a confidentiality policy for the underlying source data. However, this approach has a significant drawback: it prevents users from updating the data in the view.

To address the "view update problem" in general, a number of bidirectional languages have been proposed. Programs in these languages—often called lenses—can be run in two directions: read from left to right, they map sources to views; from right to left, they map updated views back to updated sources. However, existing bidirectional languages do not deal adequately with security. In particular, they do not provide a way to ensure the integrity of source data as it is manipulated by untrusted users of the view.

We propose a novel framework of secure lenses that addresses these shortcomings. We enrich the types of basic lenses with equivalence relations capturing notions of confidentiality and integrity, and formulate the essential security conditions as non-interference properties. We then instantiate this framework in the domain of string transformations, developing syntax for bidirectional string combinators with security-annotated regular expressions as their types.

1. Introduction

Security views are a widely used mechanism for controlling access to confidential information in databases and other systems that manage structured information. By forcing users to access data via views that only expose public information, data administrators ensure that secrets will not be leaked, even if the users mishandle the data or are malicious. Security views are robust, making it impossible for users to leak the source data hidden by the view, and they are flexible: since they are implemented as arbitrary programs, they can be used to enforce extremely fine-grained access control policies. However, they are not usually updatable—and for good reason! Propagating updates to views made by untrusted users can, in general, alter the source data, including the parts that are hidden by the view.

Still, there are many applications in which having a mechanism for reliably updating security views would be extremely useful. For example, consider Intellipedia, a collaborative data sharing system based on Wikipedia that is used by members of the intelligence community. The data stored in Intellipedia is classified at the granularity of whole documents, but many documents actually contain a mixture of highly classified and less-classified data. In order to give users with low clearances access to the portions of documents they have sufficient clearance to see, documents often have to be regraded: i.e., the highly classified parts need to be erased or redacted, leaving behind a residual document—a security view[1]—that can be reclassified at a lower level of clearance. Of course (since we are talking about a wiki), we would like the users of these views to be able to make updates—e.g., to correct errors or add new information—and have their changes be propagated back to the original document.

In general, for a view to be updatable, the program that generates it needs to be *bidirectional*. That is, it must not only be able to transform sources to views but also to map updated views back to updated sources. In previous work, we and many others have proposed a family of languages for describing bidirectional transformations, often called *lenses* [19], [8], [7], [21], [37], [41], [26], [9], [24], [35], [17], [23], [30], [28]. Formally, a lens l mapping between a set S of "source" structures and a set V of "views" comprises three functions

$$\begin{aligned} l.get &\in S \longrightarrow V \\ l.put &\in V \longrightarrow S \longrightarrow V \\ l.create &\in V \longrightarrow S \end{aligned}$$

that obey "round-tripping" laws for every $s \in S$ and $v \in V$.

$$l.get\ (l.put\ v\ s) = v \qquad \text{(PUTGET)}$$

$$l.get\ (l.create\ v) = v \qquad \text{(CREATEGET)}$$

$$l.put\ (l.get\ s)\ s = s \qquad \text{(GETPUT)}$$

The *get* function defines the view and is a total function from S to V. There are two functions that handle updates: the *put* function takes an updated V and the original S and weaves them together to yield a correspondingly modified S, while the *create* function handles the special case where we need to compute an S from a V but have no S to use as the original (it fills in any source data that is not reflected in the view with default values).

Step 4: Independence

Eventually, begin to show leadership and break ground in new directions

Goal: demonstrate independence and maturity as a researcher

Model: colleague

Role: finding the question

My emotions: excitement

Updatable Security Views

J. Nathan Foster   Benjamin C. Pierce   Steve Zdancewic
University of Pennsylvania

Abstract

Security views are a flexible and effective mechanism for controlling access to confidential information. Rather than allowing untrusted users to access source data directly, they are instead provided with a restricted view, from which all confidential information has been removed. The program that generates the view effectively embodies a confidentiality policy for the underlying source data. However, this approach has a significant drawback: it prevents users from updating the data in the view.

To address the "view update problem" in general, a number of bidirectional languages have been proposed. Programs in these languages—often called lenses—can be run in two directions: read from left to right, they map sources to views; from right to left, they map updated views back to updated sources. However, existing bidirectional languages do not deal adequately with security. In particular, they do not provide a way to ensure the integrity of source data as it is manipulated by untrusted users of the view.

We propose a novel framework of secure lenses that addresses these shortcomings. We enrich the types of basic lenses with equivalence relations capturing notions of confidentiality and integrity, and formulate the essential security conditions as non-interference properties. We then instantiate this framework in the domain of string transformations, developing syntax for bidirectional string combinators with security-annotated regular expressions as their types.

1. Introduction

Security views are a widely used mechanism for controlling access to confidential information in databases and other systems that manage structured information. By forcing users to access data via views that only expose public information, data administrators ensure that secrets will not be leaked, even if the users mishandle the data or are malicious. Security views are robust, making it impossible for users to leak the source data hidden by the view, and they are flexible: since they are implemented as arbitrary programs, they can be used to enforce extremely fine-grained access control policies. However, they are not usually updatable—and for good reason! Propagating updates to views made by untrusted users can, in general, alter the source data, including the parts that are hidden by the view.

Still, there are many applications in which having a mechanism for reliably updating security views would be extremely useful. For example, consider Intellipedia, a collaborative data sharing system based on Wikipedia that is used by members of the intelligence community. The data stored in Intellipedia is classified at the granularity of whole documents, but many documents actually contain a mixture of highly classified and less-classified data. In order to give users with low clearances access to the portions of documents they have sufficient clearance to see, documents often have to be regraded: i.e., the highly classified parts need to be erased or redacted, leaving behind a residual document—a security view[1]—that can be reclassified at a lower level of clearance. Of course (since we are talking about a wiki), we would like the users of these views to be able to make updates—e.g., to correct errors or add new information—and have their changes be propagated back to the original document.

In general, for a view to be updatable, the program that generates it needs to be *bidirectional*. That is, it must not only be able to transform sources to views but also to map updated views back to updated sources. In previous work, we and many others have proposed a family of languages for describing bidirectional transformations, often called *lenses* [19], [8], [7], [21], [37], [41], [26], [9], [24], [35], [17], [23], [30], [28]. Formally, a lens l mapping between a set S of "source" structures and a set V of "views" comprises three functions

$$\begin{aligned} l.get &\in S \longrightarrow V \\ l.put &\in V \longrightarrow S \longrightarrow V \\ l.create &\in V \longrightarrow S \end{aligned}$$

that obey "round-tripping" laws for every $s \in S$ and $v \in V$.

$$l.get\ (l.put\ v\ s) = v \qquad (\text{PutGet})$$

$$l.get\ (l.create\ v) = v \qquad (\text{CreateGet})$$

$$l.put\ (l.get\ s)\ s = s \qquad (\text{GetPut})$$

The *get* function defines the view and is a total function from S to V. There are two functions that handle updates: the *put* function takes an updated V and the original S and weaves them together to yield a correspondingly modified S, while the *create* function handles the special case where we need to compute an S from a V but have no S to use as the original (it fills in any source data that is not reflected in the view with default values).

Step 5: End-Game

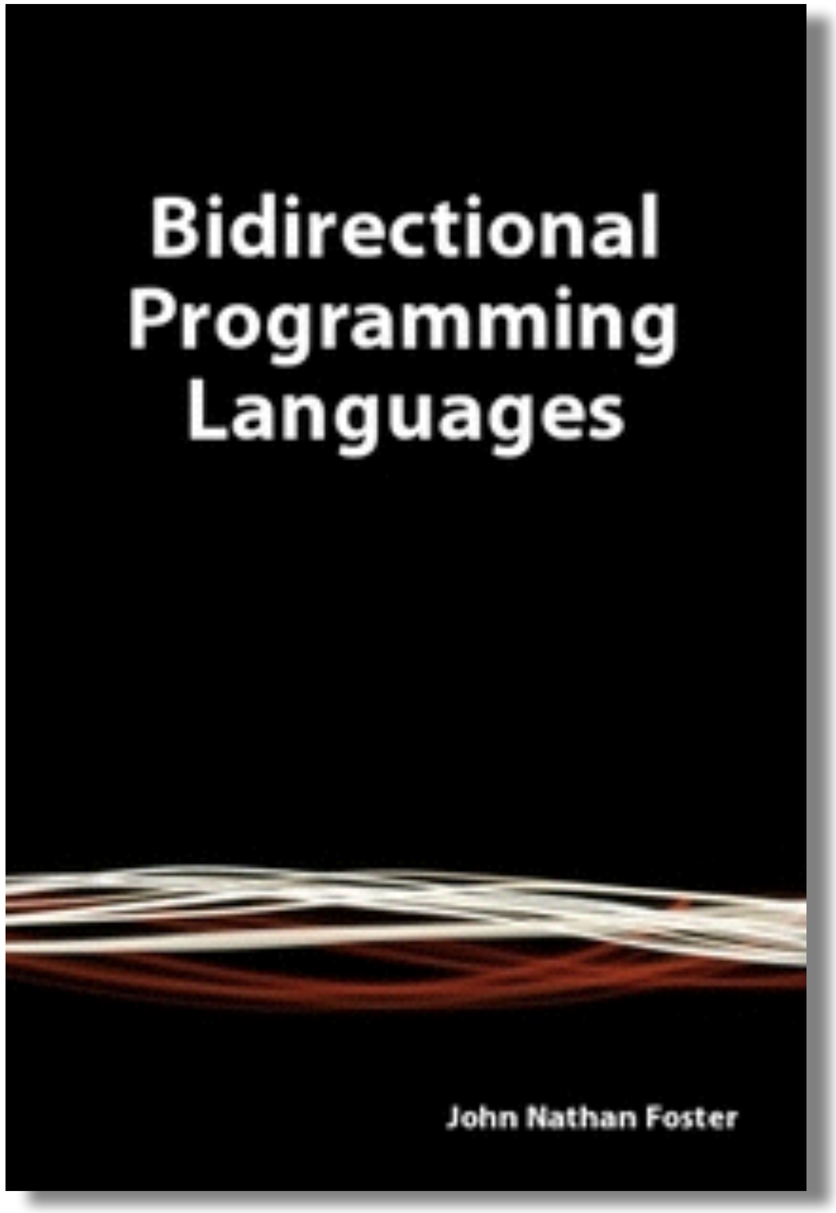
When it is time to move on…

- Form a committee
- Propose the thesis
- Write the dissertation
- Network and give talks
- Find a job

Step 5: End-Game

When it is time to move on…

- Form a committee
- Propose the thesis
- Write the dissertation
- Network and give talks
- Find a job



Bidirectional Programming Languages

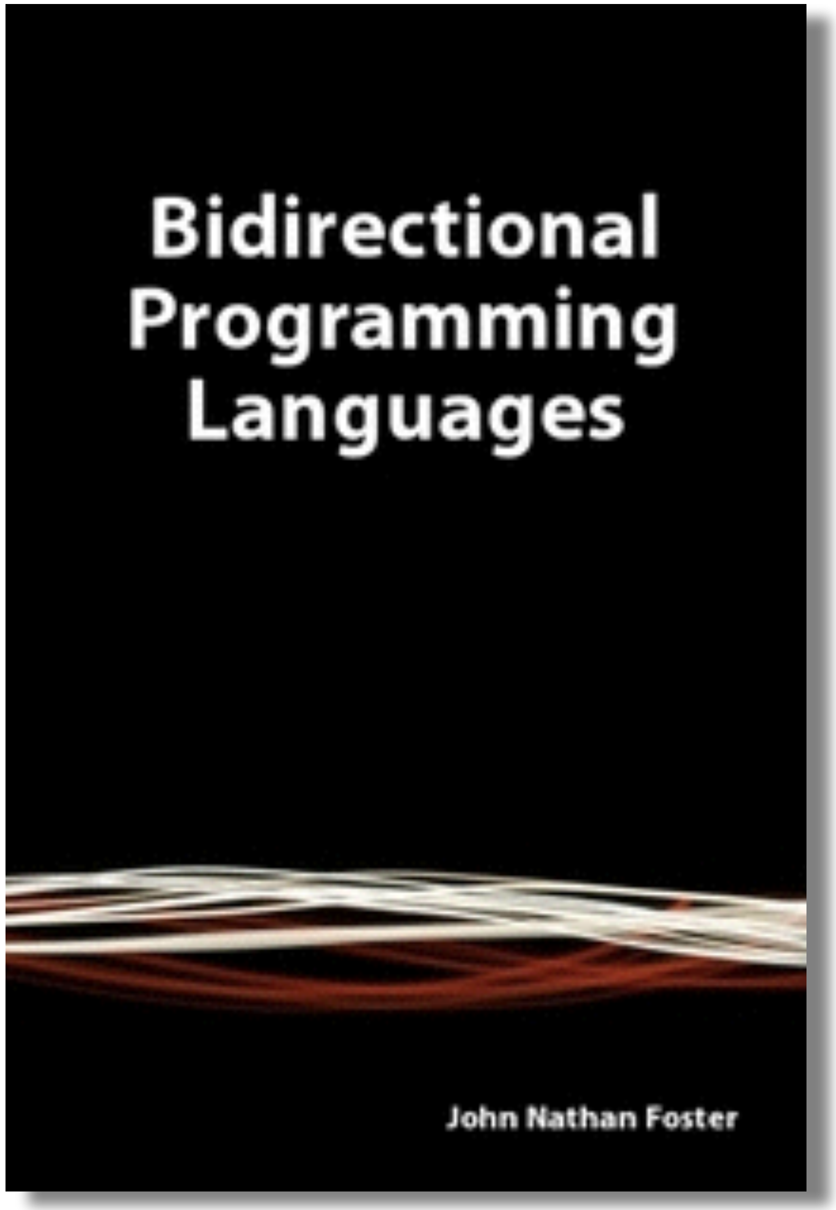
John Nathan Foster

Step 5: End-Game

When it is time to move on…

- Form a committee
- Propose the thesis
- Write the dissertation
- Network and give talks
- Find a job

My emotions: relief and exhaustion



PhD Life

PhD Life

A wonderful time to engage deeply with research

PhD Life

A wonderful time to engage deeply with research

+ Lots of unstructured time

PhD Life

A wonderful time to engage deeply with research

+ Lots of unstructured time

+ Few responsibilities

PhD Life

A wonderful time to engage deeply with research

+ Lots of unstructured time

+ Few responsibilities

+ Substantial freedom

PhD Life

A wonderful time to engage deeply with research

+ Lots of unstructured time

+ Few responsibilities

+ Substantial freedom

- (Relative) poverty

PhD Life

A wonderful time to engage deeply with research

+ Lots of unstructured time

+ Few responsibilities

+ Substantial freedom

- (Relative) poverty

- Not much respect

PhD Life

A wonderful time to engage deeply with research

+ Lots of unstructured time

+ Few responsibilities

+ Substantial freedom

- (Relative) poverty

- Not much respect

> “Don't make fun of grad students;
> they've just made a terrible life choice”
>
> –The Simpsons

PhD Life

A wonderful time to engage deeply with research

+ Lots of unstructured time

+ Few responsibilities

+ Substantial freedom

- (Relative) poverty

- Not much respect

> “Don't make fun of grad students; they've just made a terrible life choice”
>
> –The Simpsons

- Can be difficult to get distance from work

PhD Life

A wonderful time to engage deeply with research

+ Lots of unstructured time

+ Few responsibilities

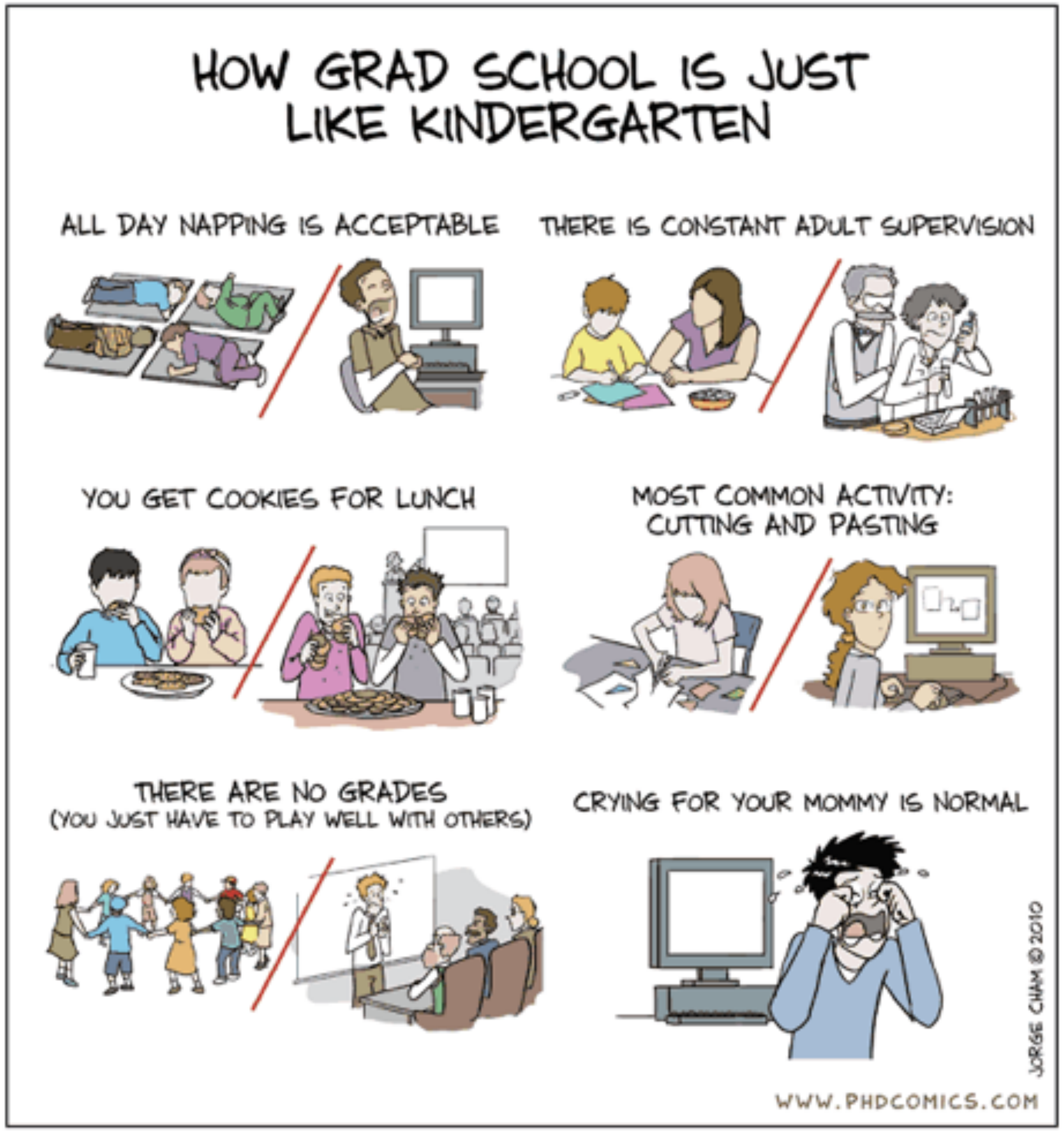
+ Substantial freedom

- (Relative) poverty

- Not much respect

> “Don't make fun of grad students; they've just made a terrible life choice”
>
> –The Simpsons

- Can be difficult to get distance from work


HOW GRAD SCHOOL IS JUST LIKE KINDERGARTEN
ALL DAY NAPPING IS ACCEPTABLE
THERE IS CONSTANT ADULT SUPERVISION
YOU GET COOKIES FOR LUNCH
MOST COMMON ACTIVITY: CUTTING AND PASTING
THERE ARE NO GRADES (YOU JUST HAVE TO PLAY WELL WITH OTHERS)
CRYING FOR YOUR MOMMY IS NORMAL
JORGE CHAM © 2010
WWW.PHDCOMICS.COM

Thank you!