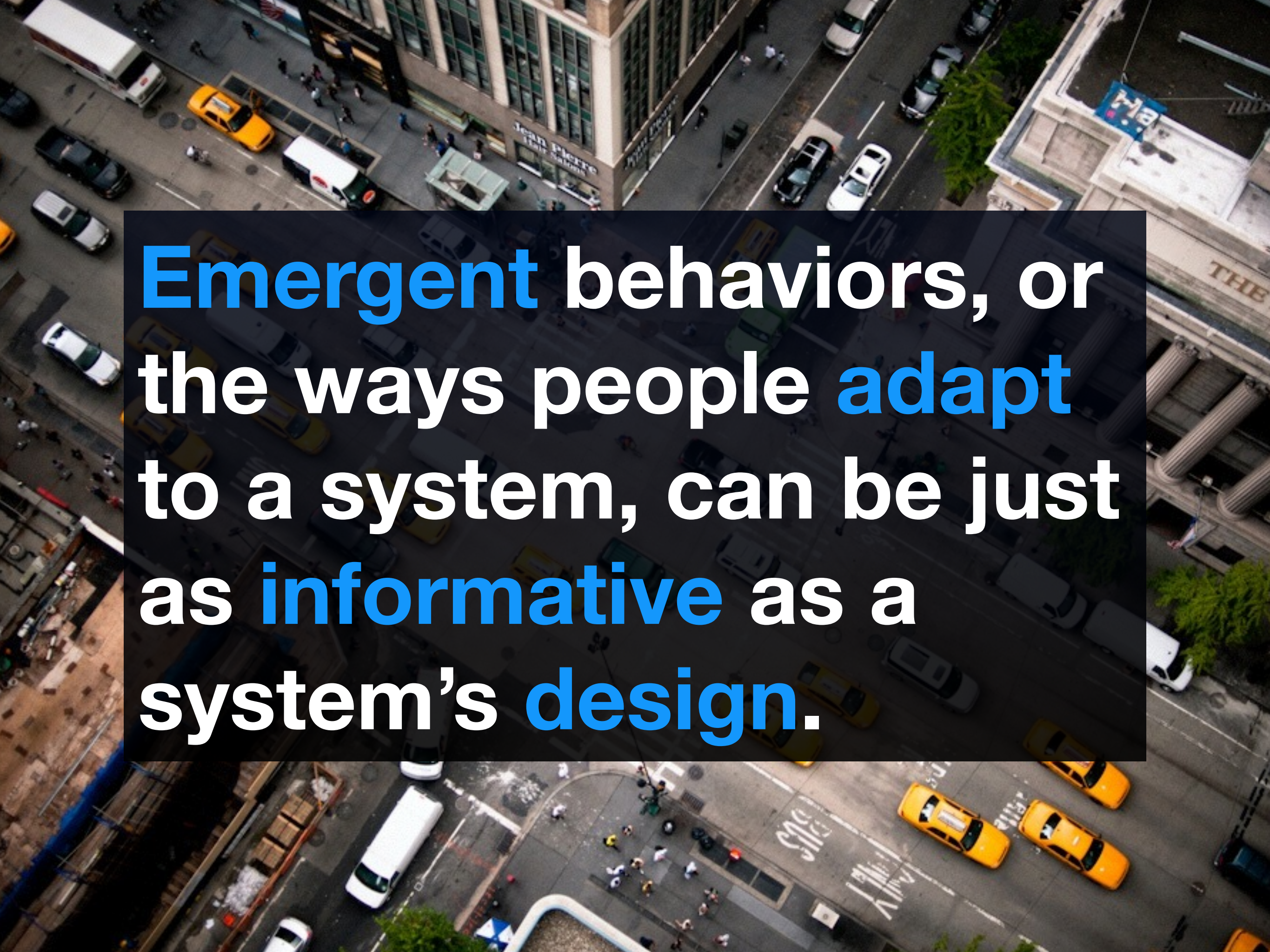


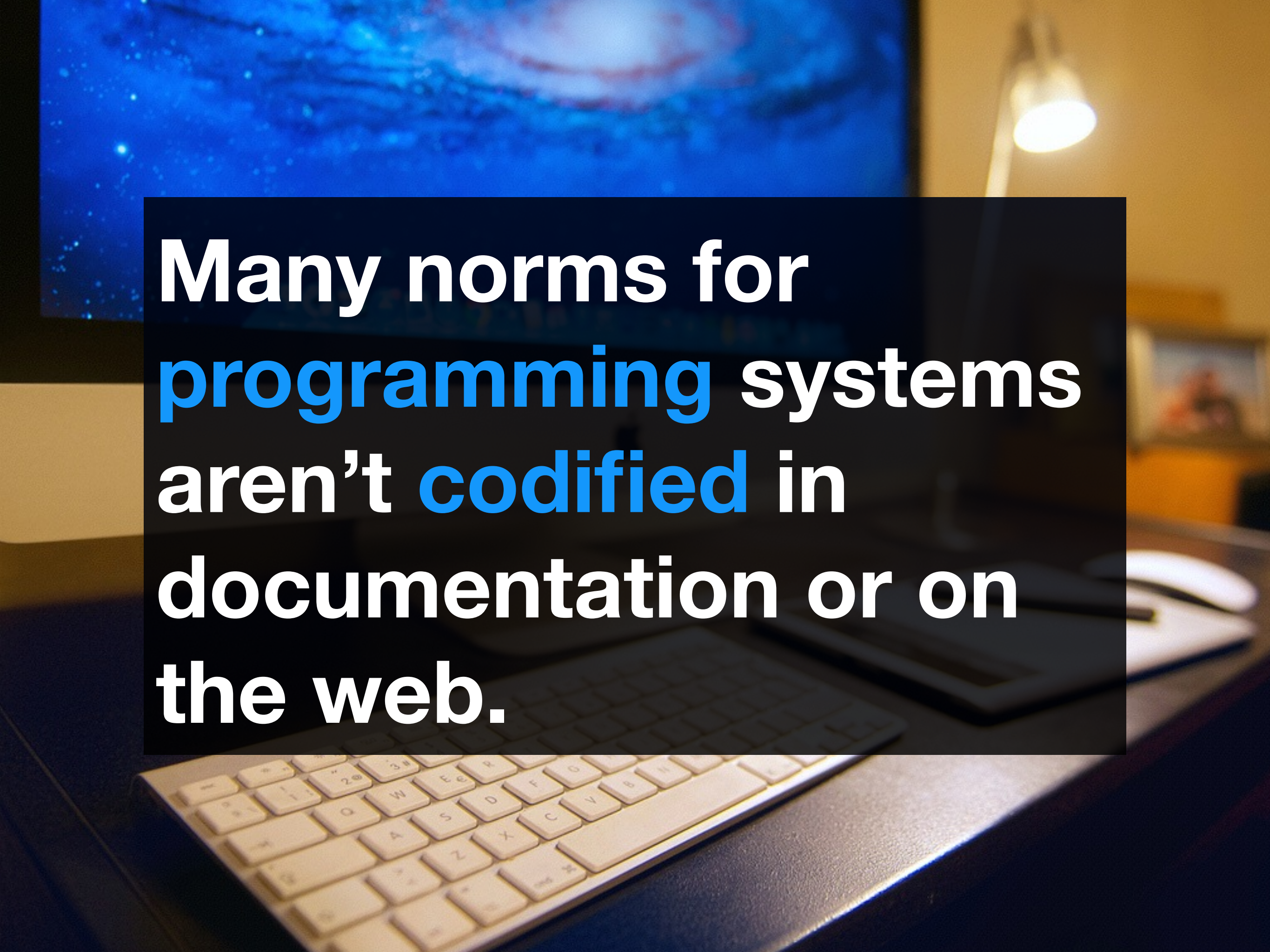
Emergent, Crowd-scale Programming Practice in the IDE

Ethan Fast, Daniel Steffee, Lucy Wang, Michael Bernstein, Joel Brandt

Stanford HCI, Adobe Research

An aerial, high-angle photograph of a busy city street intersection. The scene is filled with various vehicles, including cars, taxis, and a large white truck. Pedestrians are visible on the sidewalks. Buildings with large windows and signs are visible in the background. The overall atmosphere is one of a bustling urban environment.

Emergent behaviors, or
the ways people **adapt**
to a system, can be just
as **informative** as a
system's **design**.

A photograph of a desk setup. In the foreground, a silver laptop is open, with a white keyboard and a black mouse visible. The background is a blurred office environment with a desk lamp and a wall featuring a blue abstract pattern. A dark semi-transparent box is overlaid on the image, containing white and blue text.

**Many norms for
programming systems
aren't codified in
documentation or on
the web.**



Developers can have **unanswered** questions

What is the **best idiom** or library to use for a certain kind of task?

Does my code follow **common practice**?

How is a language being used **today**?

A Ruby Idiom

```
options_hash.rb - /Users/ethanfast/Desktop/codex presentation
options_hash.rb
1 def complicated_function(*args)
2   some_var =
3     if Hash === args.last
4       args.pop
5     else
6       {}
7     end
8   # The function does some
9   # other important things...
10  ret_result
11 end
12
```

options_hash.rb* 3,1 Ruby [Send Feedback](#)

How does this code work? What is the block doing?

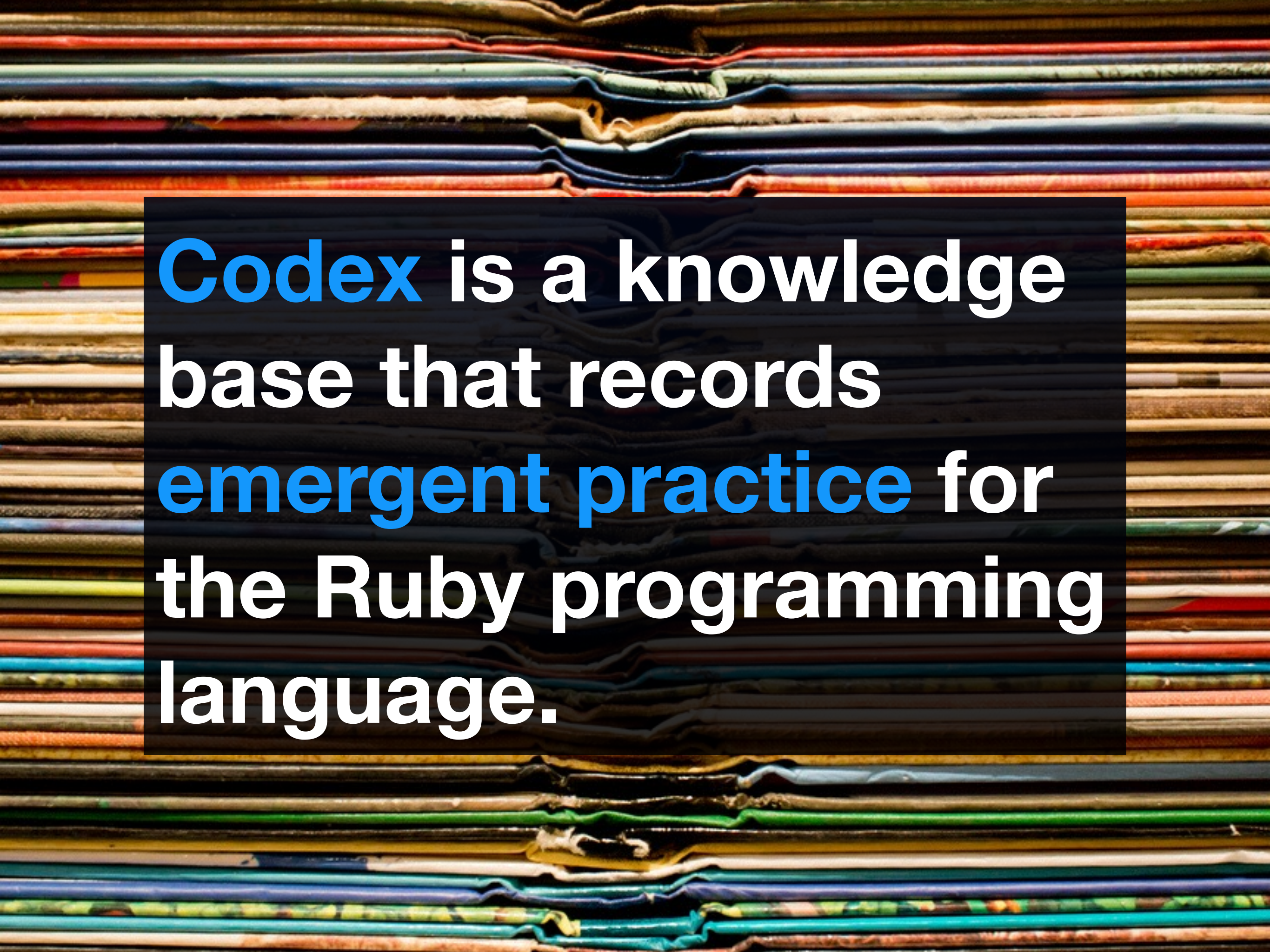
A Ruby Idiom

```
options_hash.rb - /Users/ethanfast/Desktop/codex presentation
options_hash.rb
1 def complicated_function(*args)
2   some_var =
3     if Hash === args.last
4       args.pop
5     else
6       {}
7     end
8   # The function does some
9   # other important things...
10  ret_result
11 end
12
```

options_hash.rb* 3,1 Ruby [Send Feedback](#)

Extracting an options hash from a function that takes any number of arguments

How does this code work? What is the block doing?



Codex is a knowledge base that records **emergent practice** for the Ruby programming language.

Codex normalizes code structure to identify common functions, blocks, and syntactic patterns.

Codex enables new **data-driven** interfaces for **programming**

Detect unlikely code

Annotate common idioms

Create a living library

Building the Codex Knowledge Base

**The goal: identify
emergent patterns that
good programmers
would use**

Each **record** in the
Codex knowledge base
is an **AST** node

Each **record** in the
Codex knowledge base
is an **AST** node

Are these snippets equivalent?

```
novels.map { |title| title.downcase + "!" }
```

```
movies.map { |name| name.downcase + "?" }
```

Part 1: Building the Knowledge Base

```
# Snippet 1
uist_hash = Hash.new do |hash, key|
  hash[key] = {}
end
my_hash[:UIST][“2014”] = “Hawaii”
```

```
# Snippet 2
chi_hash = Hash.new do |h, k|
  h[k] = {}
end
chi_hash[:CHI][“2014”] = “Toronto”
```

Part 1: Building the Knowledge Base

```
# Snippet 1
uist_hash = Hash.new do |hash, key|
  hash[key] = {}
end
my_hash[:UIST][“2014”] = “Hawaii”
```

```
# Snippet 2
chi_hash = Hash.new do |h, k|
  h[k] = {}
end
chi_hash[:CHI][“2014”] = “Toronto”
```

Part 1: Building the Knowledge Base

```
# Snippet 1
var0 = Hash.new do |var1, var2|
  var1[var2] = {}
end
var0[:UIST][“2014”] = “Hawaii”
```

```
# Snippet 2
var0 = Hash.new do |var1, var2|
  var1[var2] = {}
end
var0[:CHI][“2014”] = “Toronto”
```


Part 1: Building the Knowledge Base

```
# Snippet 1
var0 = Hash.new do |var1, var2|
  var1[var2] = {}
end
var0[:UIST]["2014"] = "Hawaii"
```

```
# Snippet 2
var0 = Hash.new do |var1, var2|
  var1[var2] = {}
end
var0[:CHI]["2014"] = "Toronto"
```

Part 1: Building the Knowledge Base

```
# Snippet 1
var0 = Hash.new do |var1, var2|
  var1[var2] = {}
end
var0[:SYM0]["2014"] = "Hawaii"
```

```
# Snippet 2
var0 = Hash.new do |var1, var2|
  var1[var2] = {}
end
var0[:SYM0]["2014"] = "Toronto"
```

Part 1: Building the Knowledge Base

```
# Snippet 1
var0 = Hash.new do |var1, var2|
  var1[var2] = {}
end
var0[:SYM0]["2014"] = "Hawaii"
```

```
# Snippet 2
var0 = Hash.new do |var1, var2|
  var1[var2] = {}
end
var0[:SYM0]["2014"] = "Toronto"
```

Part 1: Building the Knowledge Base

```
# Snippet 1
var0 = Hash.new do |var1, var2|
  var1[var2] = {}
end
var0[:SYM0][“STR0”] = “STR1”
```

```
# Snippet 2
var0 = Hash.new do |var1, var2|
  var1[var2] = {}
end
var0[:SYM0][“STR0”] = “STR1”
```

Statistical Linting

Chaining & Composition

```
1
2 name = "Ethan Fast"
3 lc_name = name.downcase!
4 #=> "ethan fast"
5
6 # But downcase! has a side-effect.
7 # It changes the value of name.
8
9 name
10 #=> "ethan fast"
11
12
13
14
15
16
17
18
19
```

Warning: Line 3

Codex observes `var0 = var1.downcase` more than 200 times, but `var0 = var1.downcase!` only 1 time.

Chaining & Composition

```
1  
2 name = "Ethan Fast"  
3 lc_name = name.downcase!  
4 #=> "ethan fast"  
5  
6 # But downcase! has a side-effect.  
7 # It changes the value of name.  
8  
9 name  
10 #=> "ethan fast"  
11  
12  
13  
14  
15  
16  
17  
18  
19
```



Warning: Line 3

Codex observes `var0 = var1.downcase` more than 200 times, but `var0 = var1.downcase!` only 1 time.

The function *downcase!* has a side-effect and changes name

Unlikely variable names

```
1  
2 array = {}  
3  
4 array["CHI"] = "Toronto"  
5 array["UIST"] = "Hawaii"  
6  
7 array.keys.each do |k|  
8   puts "#{k} is a conference."  
9 end  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19
```

Warning: Line 2

Codex observes variables named array 116 times and variables assigned a Hash value 1248 times, but has never seen the two together.

Unlikely variable names

```
1  
2 array = {}  
3  
4 array["CHI"] = "Toronto"  
5 array["UIST"] = "Hawaii"  
6  
7 array.keys.each do |k|  
8   puts "#{k} is a conference."  
9 end
```

Warning: Line 2

Codex observes variables named array 116 times and variables assigned a Hash value 1248 times, but has never seen the two together.

You might wonder: does an Array really have a method named *keys*?

Other kinds of analysis

```
1
2 # Function chaining: .join, not .to_s
3 "my string".split.to_s
4
5 # Type analysis: reversed arguments
6 arr = Array.new({}, 10)
7
8 # Block checks: puts returns nil
9 nums.map do |x|
10   puts x
11 end
12
13
14 |
15
16
17
18
19
```

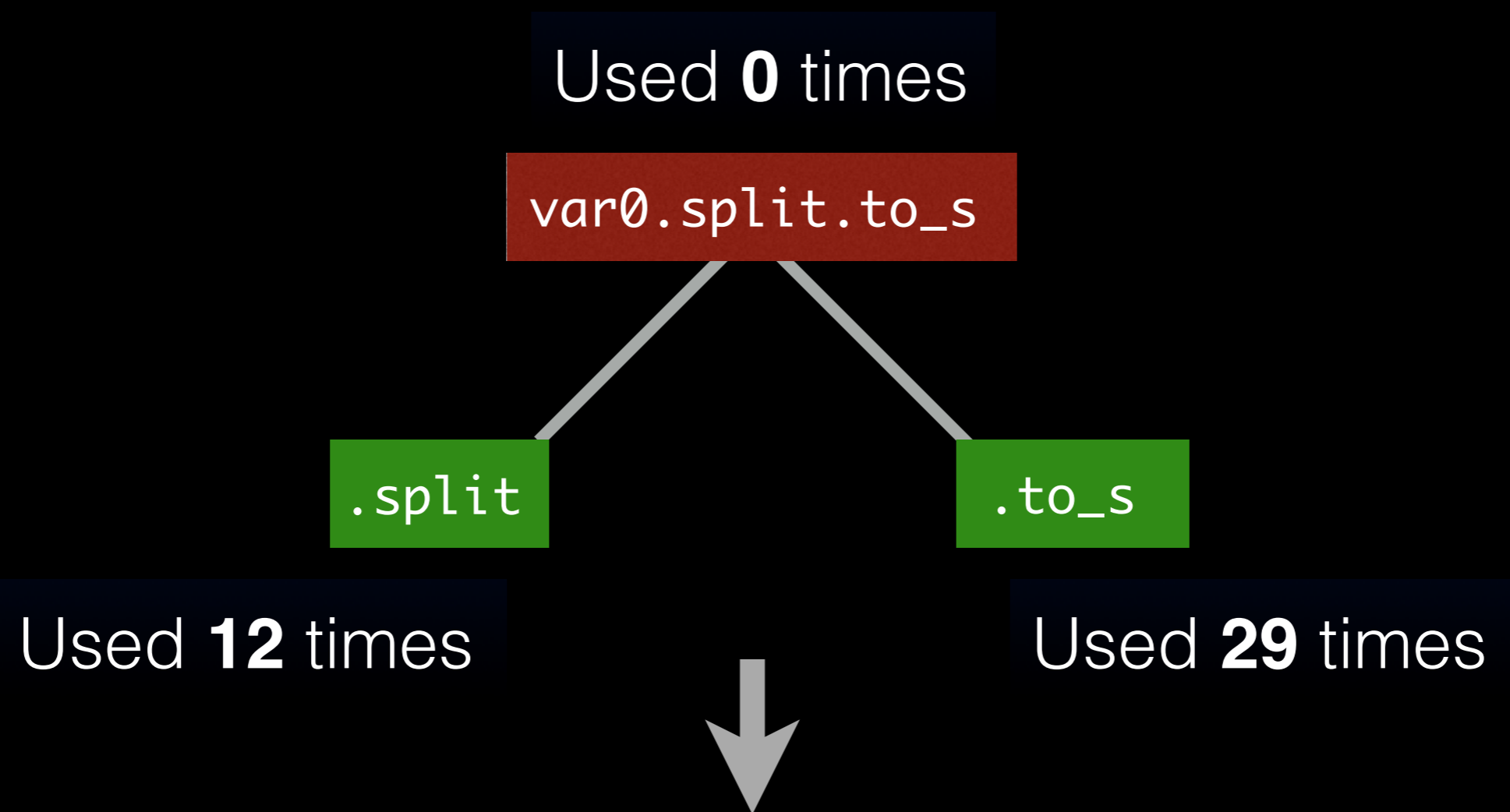
+

Function chains

Function types

**Block return
values**

`var0.split.to_s` $\#=>$ Error: Array \Rightarrow String



“Function **split** has appeared **29** times and **to_s** has appeared **12** times, but they’ve never been chained together.”

Pattern Annotation

Pattern annotation:
finds common **idioms**,
then **annotates** them
using **crowds**

rs-lex

Clone / Upgrade Database

Query for snippets with sufficient commonality and complexity

ast_stats

Documents Indexes Settings

```
find({ "type": "block", "project_count": 1, "func": 1, "body": 1, "ret_val": 1, "type": 1, "info": 1, "norm_code": 1 }).sort({ "project_count": -1 }).limit( 10 )
```

fields{} sort{}

Query results

```
db.ast_stats.find({"type": "block"}, {"project_count": 1, "func": 1, "body": 1, "ret_val": 1, "type": 1, "info": 1, "norm_code": 1}).sort({"project_count": -1}).limit(10)
```

{ "_id": ObjectId("520a8e3584d6462dcd00028f"), "project_count": 23, "type": "block", "info": 4, "func": "Hash.new", "body": "var0[var1] = {}", ret_val: "[]=", norm_code: "new do var0, v\n\nend" }
{ "_id": ObjectId("520a8e3584d6462dcd000112"), "project_count": 18, "type": "block", "info": 2, "func": "each", "body": "yield(var0)", ret_val: "yield", norm_code: "each do var1 \n yield" }
{ "_id": ObjectId("520a8e3084d6462dcd000000"), "project_count": 13, "type": "block", "info": 3, "func": "map", "body": "var0.to_s", ret_val: "to_s", norm_code: "map do var1 \n var1.to" }
{ "_id": ObjectId("520a8e3584d6462dcd000000"), "project_count": 11, "type": "block", "info": 4, "func": "Hash.new", "body": "var0[var1] = {}", ret_val: "[]=", norm_code: "new do var0, v\n\nend" }
{ "_id": ObjectId("520a8e3984d6462dcd000000"), "project_count": 10, "type": "block", "info": 3, "func": "each", "body": "var0 << var1", ret_val: "<<", norm_code: "each do var1 \n var2" }
{ "_id": ObjectId("520a8e2f84d6462dcd000000"), "project_count": 9, "type": "block", "info": 3, "func": "each", "body": "var0[var1] = var2", ret_val: "[]=", norm_code: "each do var1, va\nvar2\nend" }
{ "_id": ObjectId("520a8e3884d6462dcd000989"), "project_count": 9, "type": "block", "info": 3, "func": "each", "body": "self << var0", ret_val: "<<", norm_code: "each do var1 \n self <" }
{ "_id": ObjectId("520a8e3684d6462dcd000453"), "project_count": 9, "type": "block", "info": 3, "func": "map", "body": "var0.inspect", ret_val: "inspect", norm_code: "map do var1 \n v" }
{ "_id": ObjectId("520a8e3084d6462dcd000046"), "project_count": 8, "type": "block", "info": 6, "func": "File.open", "body": "var0.write(var1)", ret_val: "write", norm_code: "open/var0" }

```
mongo_query = {  
  project_count: { gt: .02 },  
  total_count: { lt: 0.9 },  
  file_count: { lt: 0.2 },  
  token_count: { lt: 0.8 },  
  function_count: { gt: 2.0 }  
}
```

Job Postings

Post a Job

Find Freelancers

Saved Freelancers

Find Freelancers

Find Freela

New post added! Check out what's new on oDesk.

Recruit

Search for Freelancers

Search

Post a Job

Browse Freelancers by category

My Open Jobs

View all job postings (including filled and closed jobs)

Popular jobs

Next we crowdsource
a **title, description,** and
vote of usefulness
from oDesk workers

Logo Design

Web Programming

Email Marketing

Typically
\$30 - \$100 / logo

Typically
\$15 - \$25 / hr

Typically
\$5 - \$10 / hr

[Browse all jobs posted on oDesk »](#)

Recently Hired

1 hour ago - **Karol Koziol** was hired for the job Fill in Bibtex citations. [View Contract](#)

Nested Hashes

```
1
2 # Creating a nested Hash
3 my_hash = Hash.new { |h,k|
4   h[k] = {}
5 }
6
7 my_hash[:CHI][:Toronto] = true
8
9 # Naive way:
10 Hash.new({}) # This is a bug!
11
12
13
14
15
16
17
18
19
```

Creating a Nested Hash

Total count: 66 **Project count:** 10

Creates a Hash with a new empty Hash object as a default key value

Nested Hashes

```
1
2 # Creating a nested Hash
3 my_hash = Hash.new { |h,k|
4   h[k] = {}
5 }
6
7 my_hash[:CHI][:Toronto] = true
8
9 # Naive way:
10 Hash.new({}) # This is a bug!
11
12
13
14
15
16
17
18
19
```

Creating a Nested Hash

Total count: 66 Project count: 10

Creates a Hash with a new empty Hash object as a default key value

This simple idiom is easy to mess up!

Configure Rails Caching

```
1
2 Rails.application.configure do
3   config.cache_classes = true
4   config.eager_load = false
5   config.serve_static_assets = true
6   config.static_cache_control = 'public, max-age=3600'
7   config.consider_all_requests_local = true
8   config.action_controller.preform_caching = false
9   config.action_dispatch.show_exceptions = false
10  config.action_controller.allow_forgery_protection = f
11  config.action_mailer.delivery_method = :test
12  config.active_support.deprecation = :stderr
13 end
```

Configure Rails Caching

Total count: 78 Project count: 34

By setting this to false, you can turn off caching for the Rails web framework

Raise StandardError

```
1
2
3 raise StandardError.new("CHI")
4
5 #=> StandardError: Failed for CHI
6
7
8
9
10
11
12
13
14
15
16
17
18
19
```

Raise Custom Error

Total count: 66 **Project count:** 10

Raise a new StandardError using a custom message, passed as a string value

Library Generation

**Library generation
constructs a utility
package that reflects
common practice**

String#capitalize_tokens

```
capitalise.rb - /Users/ethanfast/Desktop/codex presentation
op... o us... x do... x na... o ot... x ne... x ca... o er... x capital... x
1
2 # Capitalize each word token in a string
3 class String
4   def capitalize_tokens
5     self.split(" ").map do |w|
6       w.capitalize
7     end.join(" ")
8   end
9 end
10
11 "hello CHI audience.".capitalize_tokens
12 #=> "Hello CHI Audience."
13
capitalise.rb 3,13 Ruby Send Feedback
```

Capitalize each word token in a string

This idiom occurred **10** times across **5** different projects.

Hash##nested

```
nested_class.rb - /Users/ethanfast/Desktop/codex presentation
o... o u... x d... x n... o o... x n... x c... o e... x c... x nest... x

1
2 # Modifying the Hash class to
3 # create a helper for nested Hashes
4 class Hash
5   def self.nested
6     Hash.new { |h,k| h[k] = {} }
7   end
8 end
9
10 # In practice!
11 my_hash = Hash.nested
12

nested_class.rb 10,15 Ruby Send Feedback
```

Create a helper method for nested Hashes

This idiom occurred **66** times across **12** different projects.

Evaluation

Hit-rate after 500k LOC

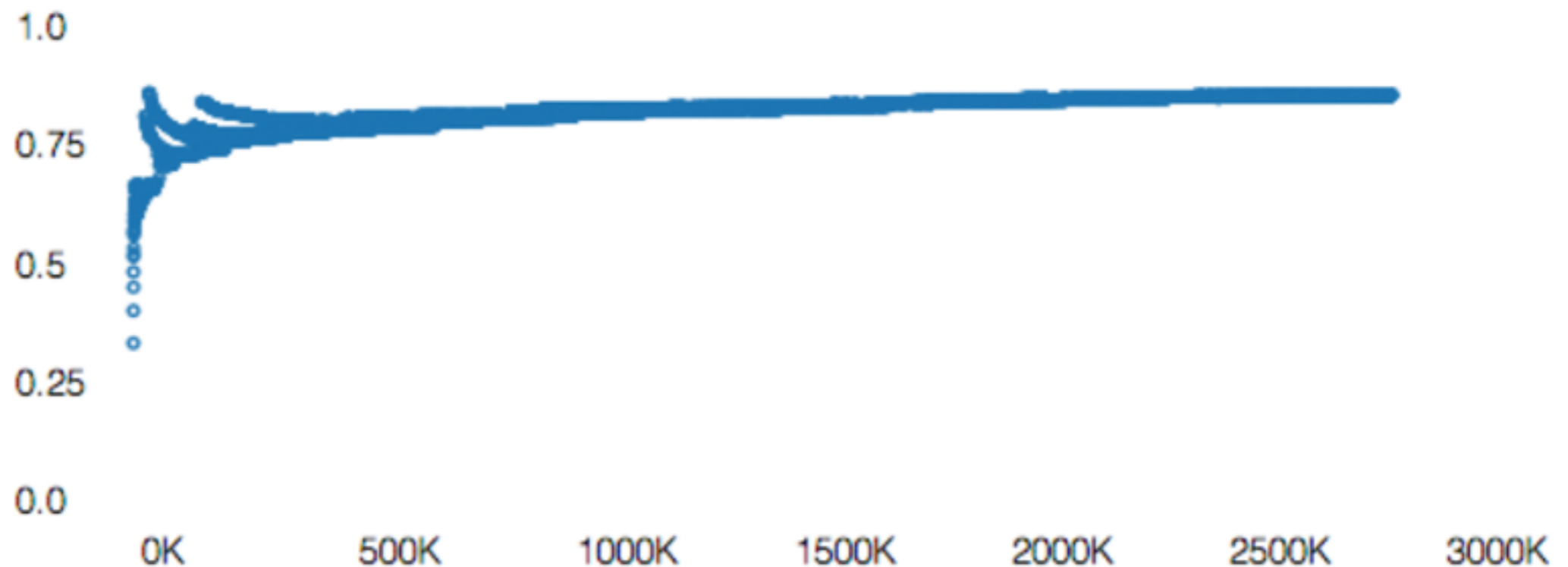
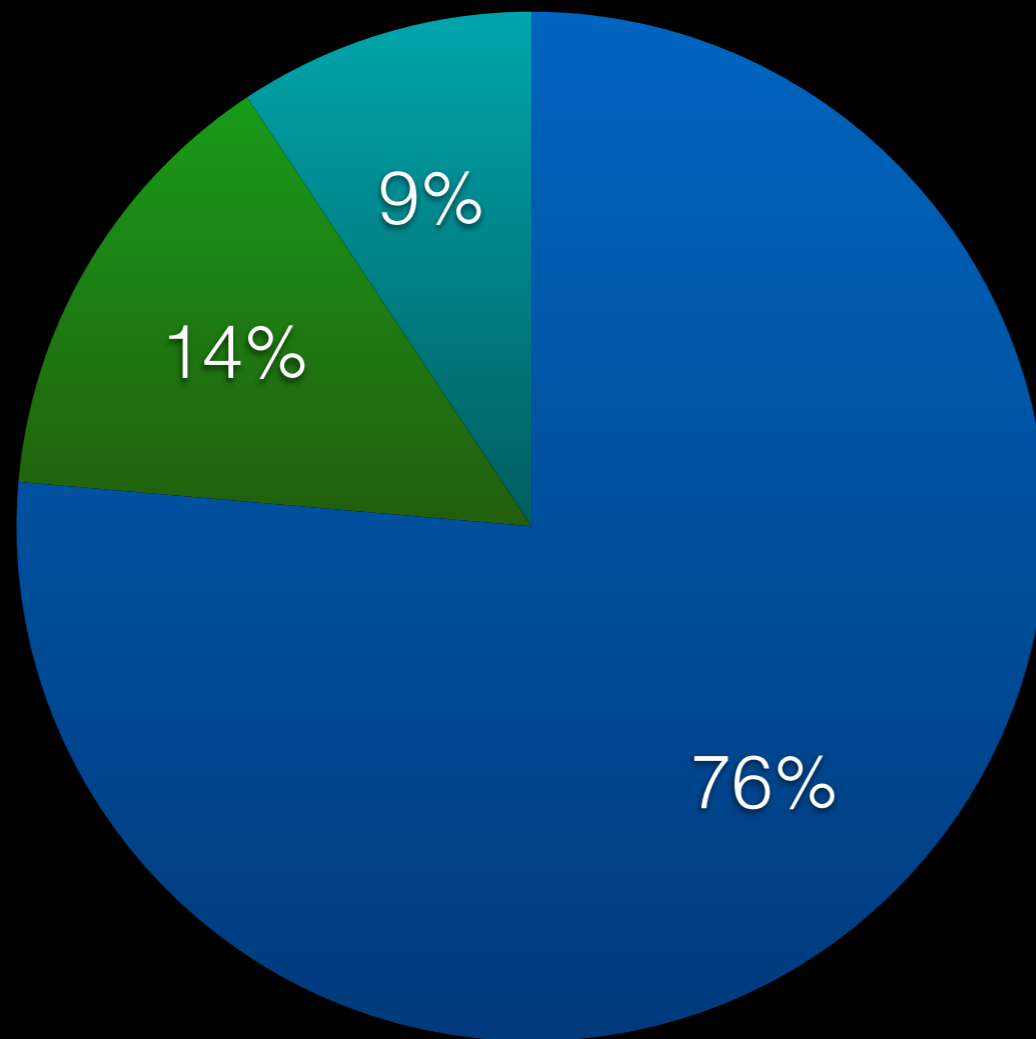


Figure 3. A plot of Codex’s hit rate as it indexes code over four random samples of file orderings. The y-axis plots the database hit rate, and the x-axis plots the number of lines of code indexed.

Snippet categories

- Standard Library
- External Library
- Data / Control Flow



A survey of expert crowdworkers

86% of snippets are useful

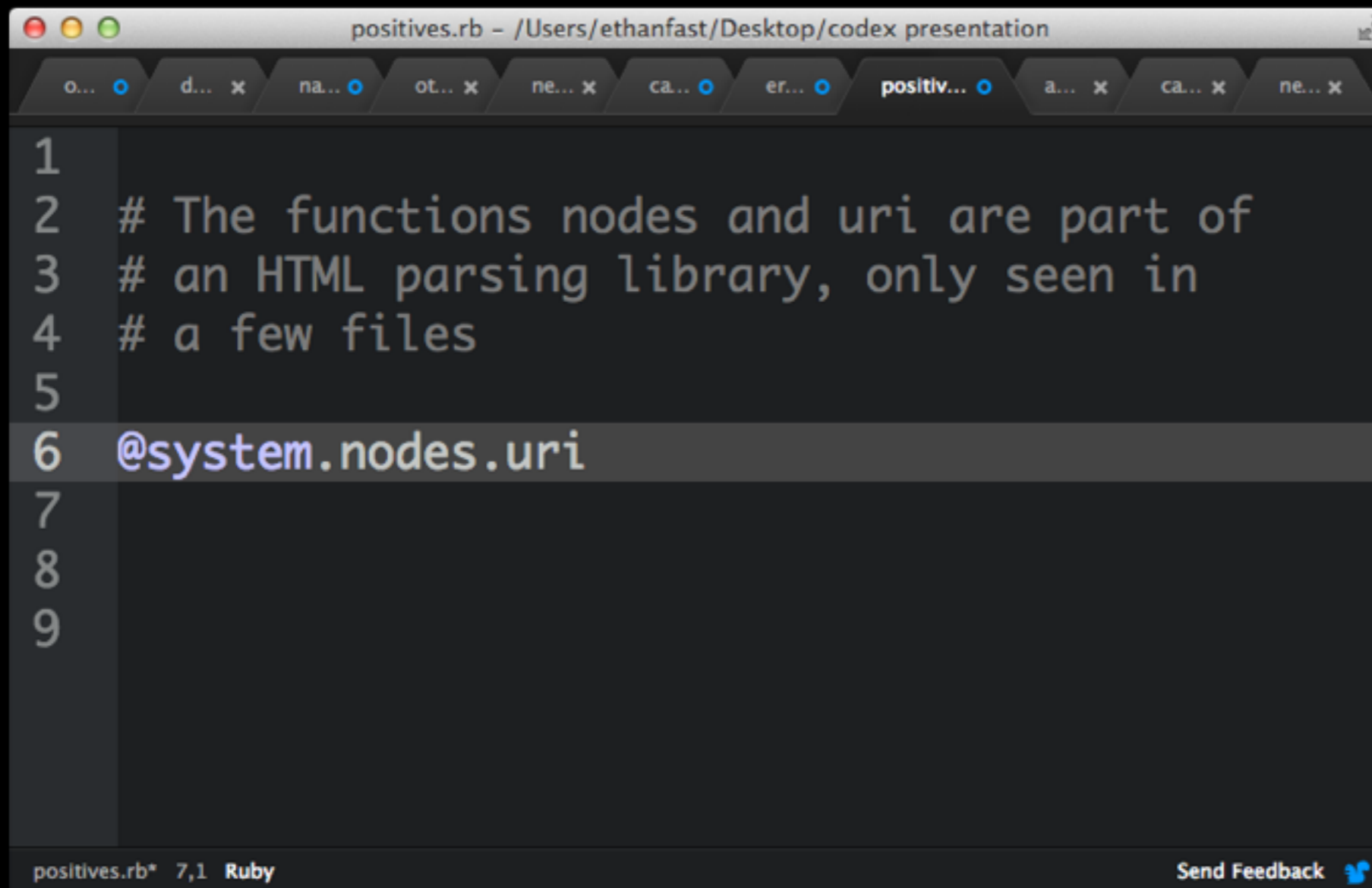
96% are recomposable

91% have no more common form

Statistical linting and false positives

We find **1,248** warnings over **49,735** lines, a rate of **2.5%**.

Common false positives

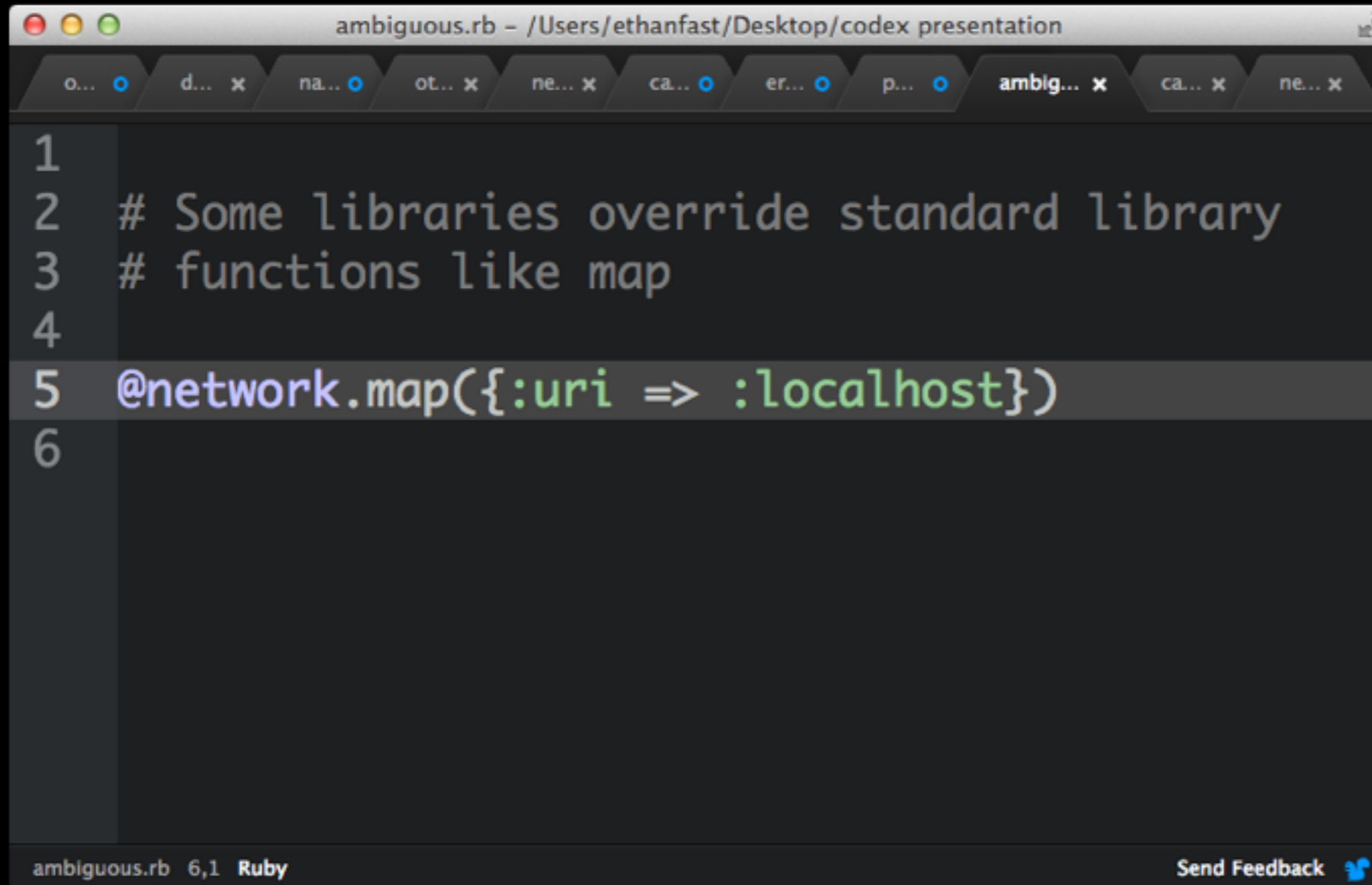


The screenshot shows a code editor window titled "positives.rb - /Users/ethanfast/Desktop/codex presentation". The code contains a comment on lines 2-4 and a variable assignment on line 6. A red squiggly line underlines the variable "@system.nodes.uri" on line 6, indicating a linting error. The status bar at the bottom shows "positives.rb* 7,1 Ruby" and a "Send Feedback" button.

```
1
2 # The functions nodes and uri are part of
3 # an HTML parsing library, only seen in
4 # a few files
5
6 @system.nodes.uri
7
8
9
```

positives.rb* 7,1 Ruby [Send Feedback](#)

Ambiguous false positives



```
ambiguous.rb - /Users/ethanfast/Desktop/codex presentation  
o... o d... x na... o ot... x ne... x ca... o er... o p... o ambig... x ca... x ne... x  
1  
2 # Some libraries override standard library  
3 # functions like map  
4  
5 @network.map({:uri => :localhost})  
6  
ambiguous.rb 6,1 Ruby Send Feedback
```

Conclusion



**Mining emergent
practice can support a
broad set of software
engineering interfaces**

Programming languages can be **living artifacts**

Libraries self-update to the latest idioms

IDEs offer suggestions to suit new coding styles

Languages evolve to better support their users



Emergent, Crowd-scale Programming Practice in the IDE

Ethan Fast, Daniel Steffee, Lucy Wang, Michael Bernstein, Joel Brandt

Stanford HCI, Adobe Research

Extra Slides



**Conventions emerge
among many different
kinds of domains.**

Writing

Photography

Presentations

Programming

Research

Design

...

Chaining & Composition

```
downcase.rb - /Users/ethanfast/Desktop/codex presentation
options_hash.rb  using_options_hash.rb  downcase.rb
1  name = "Ethan Fast"
2  lc_name = name.downcase!
3  #=> "ethan fast"
4
5  # But downcase! has a side-effect.
6  # It changes the value of name.
7
8  name
9  #=> "ethan fast"
10

downcase.rb 3,1 Ruby Send Feedback
```

Chaining & Composition

```
downcase.rb - /Users/ethanfast/Desktop/codex presentation
options_hash.rb  using_options_hash.rb  downcase.rb
1  name = "Ethan Fast"
2  lc_name = name.downcase!
3  #=> "ethan fast"
4
5  # But downcase! has a side-effect.
6  # It changes the value of name.
7
8  name
9  #=> "ethan fast"
10

downcase.rb 3,1 Ruby Send Feedback
```

The function *downcase!* has a side-effect and changes *name*

Chaining & Composition

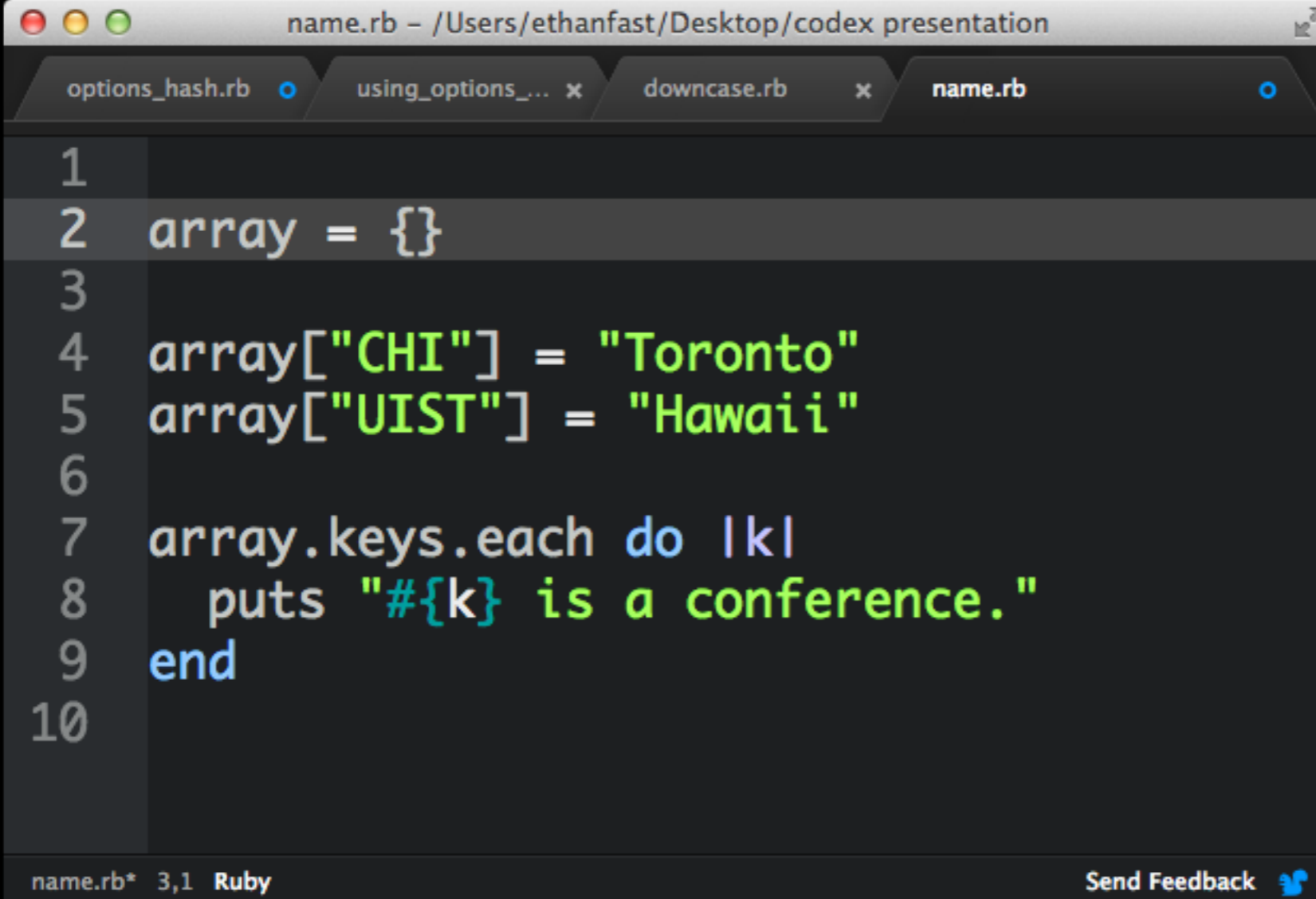
```
downcase.rb - /Users/ethanfast/Desktop/codex presentation
options_hash.rb  using_options_hash.rb  downcase.rb
1  name = "Ethan Fast"
2  lc_name = name.downcase!
3  #=> "ethan fast"
4
5  # But downcase! has a side-effect.
6  # It changes the value of name.
7
8  name
9  #=> "ethan fast"
10

downcase.rb 3,1 Ruby Send Feedback
```

The function *downcase!* has a side-effect and changes *name*

Codex observes `var0 = var1.downcase` more than **200** times, but `var0 = var1.downcase!` only **1** time.

Unlikely variable names



```
name.rb - /Users/ethanfast/Desktop/codex presentation
options_hash.rb  using_options_...  downcase.rb  name.rb

1
2 array = {}
3
4 array["CHI"] = "Toronto"
5 array["UIST"] = "Hawaii"
6
7 array.keys.each do |k|
8   puts "#{k} is a conference."
9 end
10

name.rb* 3,1 Ruby Send Feedback
```

Unlikely variable names

```
name.rb - /Users/ethanfast/Desktop/codex presentation
options_hash.rb  using_options_...  downcase.rb  name.rb
1
2 array = {}
3
4 array["CHI"] = "Toronto"
5 array["UIST"] = "Hawaii"
6
7 array.keys.each do |k|
8   puts "#{k} is a conference."
9 end
10
name.rb* 3,1 Ruby Send Feedback
```

You might wonder: does an Array really have a method named *keys*?

Unlikely variable names

```
name.rb - /Users/ethanfast/Desktop/codex presentation
options_hash.rb  using_options_...  downcase.rb  name.rb
1
2 array = {}
3
4 array["CHI"] = "Toronto"
5 array["UIST"] = "Hawaii"
6
7 array.keys.each do |k|
8   puts "#{k} is a conference."
9 end
10

name.rb* 3,1 Ruby Send Feedback
```

You might wonder: does an Array really have a method named *keys*?

Codex observes variables named *array* **116** times and variables assigned a *Hash* value **many thousands** of times, but we never see the two together.

Nested Hashes

```
nested_hash.rb - /Users/ethanfast/Desktop/codex presentation
options... x using_... x downca... x name.rb o other.rb x nested_hash.rb x
1
2 # Creating a nested Hash
3 my_hash = Hash.new { |h,k|
4     h[k] = {}
5 }
6
7 my_hash[:CHI][:Toronto] = true
8
9 # Naive way:
10 Hash.new({}) # This is a bug!
11
```

nested_hash.rb 3,1 Ruby [Send Feedback](#)

Nested Hashes

```
nested_hash.rb - /Users/ethanfast/Desktop/codex presentation
options... x using_... x downca... x name.rb o other.rb x nested_hash.rb x
1
2 # Creating a nested Hash
3 my_hash = Hash.new { |h,k|
4     h[k] = {}
5 }
6
7 my_hash[:CHI][:Toronto] = true
8
9 # Naive way:
10 Hash.new({}) # This is a bug!
11

nested_hash.rb 3,1 Ruby Send Feedback
```

Assigns an empty Hash as the default key value

Nested Hashes

```
nested_hash.rb - /Users/ethanfast/Desktop/codex presentation
options... using_... downca... name.rb other.rb nested_hash.rb x
1
2 # Creating a nested Hash
3 my_hash = Hash.new { |h,k|
4     h[k] = {}
5 }
6
7 my_hash[:CHI][:Toronto] = true
8
9 # Naive way:
10 Hash.new({}) # This is a bug!
11


nested_hash.rb 3,1 Ruby Send Feedback
```

Assigns an empty Hash as the default key value

This simple idiom is easy to mess up!

Turn off Rails Caching

```
1 Rails.application.configure do
2   config.cache_classes = true
3   config.eager_load = false
4   config.serve_static_assets = true
5   config.static_cache_control = 'public, max-age=3600'
6   config.consider_all_requests_local = true
7   config.action_controller.perform_caching = false
8   config.action_dispatch.show_exceptions = false
9   config.action_controller.allow_forgery_protection = false
10  config.action_mailer.delivery_method = :test
11  config.active_support.deprecation = :stderr
12 end
13
```

Send Feedback 

Turning off default caching for the Rails web framework

Raise StandardError



The screenshot shows a window titled "error.rb - /Users/ethanfast/Desktop/codex presentation". The window contains a code editor with the following content:

```
1  
2 raise StandardError.new("Failed for CHI")  
3  
4 #=> StandardError: Failed for CHI  
5
```

At the bottom of the window, the status bar displays "error.rb 5,1 Ruby" and a "Send Feedback" button with a Twitter icon.

**Raise a new
StandardError
message using
a custom
message**

Data mining for Codex

1. Gather Ruby code from Github

2. Parse the code into AST representation

3. Normalize the ASTs (rename variables, strings, symbols, and numbers)

4. Collapse normalized ASTs

Data mining for Codex

1. Gather Ruby code from Github

2. Parse the code into AST representation

3. Normalize the ASTs (rename variables, strings, symbols, and numbers)

4. Collapse normalized ASTs

An AST node **s** must occur fewer than **t** times, and its children **c_i** must occur more than **t_i** times

E.g., the snippet `var0.split.to_s` is composed of `.split` and `.to_s`