Microsoft Research

Each year Microsoft Research hosts hundreds of influential speakers from around the world including leading scientists, renowned experts in technology, book authors, and leading academics, and makes videos of these lectures freely available.

|galois|

Continuously Verified Cryptography

Mike Dodds

PNW PLSE Workshop, May 2018

Who Galois are

Research and development lab of ~70 people

Locations

- Portland, OR
- Arlington, VA
- Dayton, OH


What we do

PL research meets real-world applications

Programming languages, analysis, verification, security, cryptography

Our tools

Haskell

- Symbolic execution
- Model checking
- Interactive theorem provers
- Functional programming (esp. Haskell)

Who I am

Principal scientist at Galois (August '17); former U.York, UK professor

Main areas: Separation logic, concurrency, relaxed memory

Recent papers:

- Verified compiler optimizations (ESOP'18)
- Concurrency verification tools (CAV'17)
- Linearizability proofs (ESOP'17)

Continuous Verification

Correctness of core components in Amazon's s2n TLS library.


amazon
UNIVERSITY OF WASHINGTON
1861

2018: static tools used daily by

Google

- 1 billion LOC code base
- 20,000 code reviews per day
- Approach: AST patterns
- Tool: **ErrorProne**

Facebook

- Static analysis of every diff
- Millions of LOC
- Approach: separation logic, abstract interp.
- Tool: **Infer**

Amazon

- Proofs of correctness
- Core infrastructure
- (millions reqs. per sec.)
- Tool: **SAW**

Encouraging code quality

2018: static tools used daily by

Google

- 1 billion LOC code base
- 20,000 code reviews per day
- Approach: AST patterns
- Tool: **ErrorProne**

Facebook

- Static analysis of every diff
- Millions of LOC
- Approach: separation logic, abstract interp.
- Tool: **Infer**

Amazon

- Proofs of correctness
- Core infrastructure
- (millions reqs. per sec.)
- Tool: **SAW**

Encouraging code quality

Revision Control
(git, hg)

Testing
(junit, Travis)

Peer Review
(GitHub Pull Requests)

Static Analysis / verification

2018: static tools used daily by

Google
- 1 billion LOC code base
- 20,000 code reviews per day
- Approach: AST patterns
- Tool: **ErrorProne**

Facebook
- Static analysis of every diff
- Millions of LOC
- Approach: separation logic, abstract interp.
- Tool: **Infer**

Amazon
- Proofs of correctness
- Core infrastructure
- (millions reqs. per sec.)
- Tool: **SAW**

2018: static tools used daily by

Google
- 1 billion LOC code base
- 20,000 code reviews per day
- Approach: AST patterns
- Tool: **ErrorProne**

Facebook
- Static analysis of every diff
- Millions of LOC
- Approach: separation logic, abstract interp.
- Tool: **Infer**

Amazon
- Proofs of correctness
- Core infrastructure
- (millions reqs. per sec.)
- Tool: **SAW**

galois

TLS: Transport Layer Security

TLS (newer version of SSL) provides us most of the

Confidentiality
Data-Integrity
Authentication

guarantees that we enjoy on the internet today.

Amazon s2n: A TLS Implementation

- Inspired by TLS vulnerabilities discovered by researchers in other implementations.
- Written with security and performance as primary goals.
- Drops some arguably insecure/less secure features.
    - Result: Much smaller, clearer, more auditable code.
    - OpenSSL TLS is 70k lines of C code.
    - s2n is only 6k.
- Used in production at Amazon (and therefore used by pretty much everyone on the internet)

HMAC: A Component of TLS

- keyed-Hash Message Authentication Code
- Provides a signature for a message that confirms:
  - Authenticity: the message was signed by the expected sender
  - Integrity: the message has not been modified

message

derived from key

$$\mathrm{HMAC}(K, m) = H((K \oplus opad) \parallel H((K \oplus ipad) \parallel m))$$

arbitrary hash function

constants from NIST

- Code is still complex
  - 521 lines of C code

HMAC Specification

C HMAC

$$\mathrm{HMAC}(K, m) = H((K \oplus opad) \,\|\, H((K \oplus ipad) \,\|\, m))$$

Easily auditable

Concise

Goal: bridge this gap

Fast

Interoperable

Summary of Approach

1. Write the formal specification.
2. Write some "scaffolding" to bridge the gap between specification and C code.
3. Apply automated tools.
4. Integrate into development environment.

About 2 months of effort.

Summary of Approach

1. Write the formal specification.
2. Write some “scaffolding” to bridge the gap between specification and C code.
3. Apply automated tools.
4. Integrate into development environment.

About 2 months of effort.

Summary of Approach

$$HMAC(K, m) = H((K \oplus opad) \parallel H((K \oplus ipad) \parallel m))$$

Step 1: Capture this specification in a formal language (we used Cryptol - https://cryptol.net/)

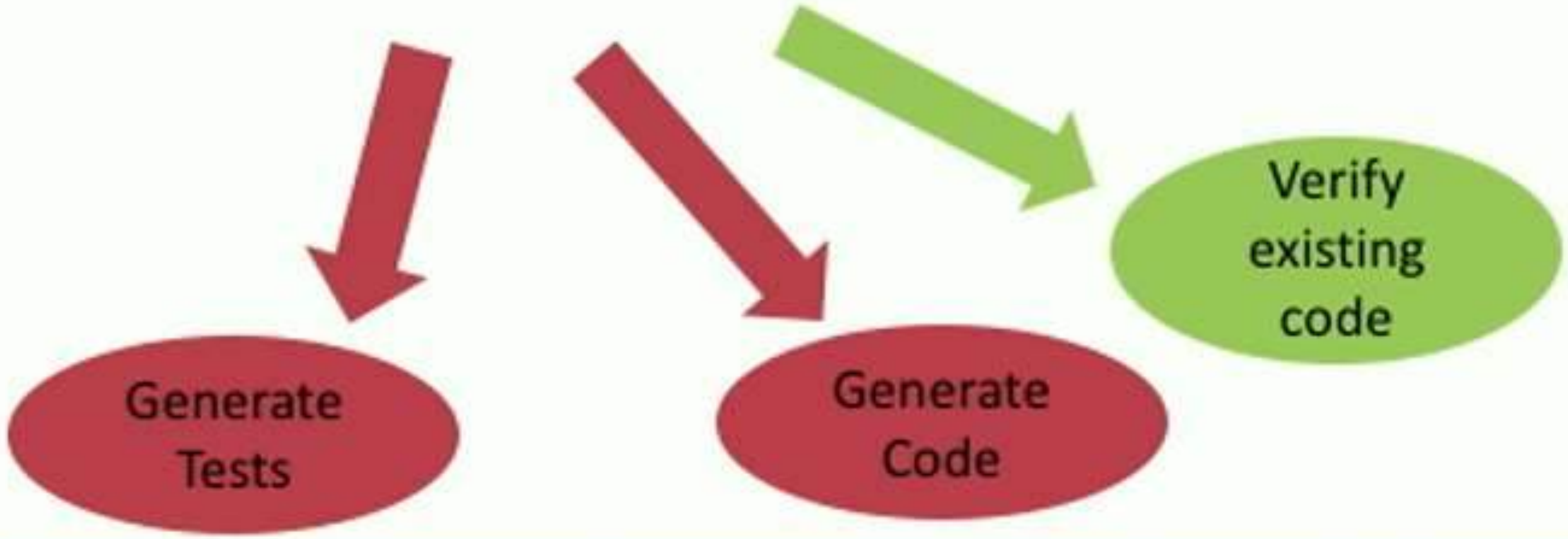
```
hmac k message =
  H((k ^ opad) # H((k ^ ipad) # message))
```

Summary of Approach

$$\mathrm{HMAC}(K, m) = H((K \oplus opad) \parallel H((K \oplus ipad) \parallel m))$$

Step 1: Capture this specification in a formal language (we used Cryptol - https://cryptol.net/)

```
hmac k message =
  H((k ^ opad) # H((k ^ ipad) # message))
```



HMAC Specification

C HMAC

```
hmac k message =
   H((k ^ opad) # H((k ^ ipad) # message))
```

Concise

Easily auditable

Goal: bridge this gap

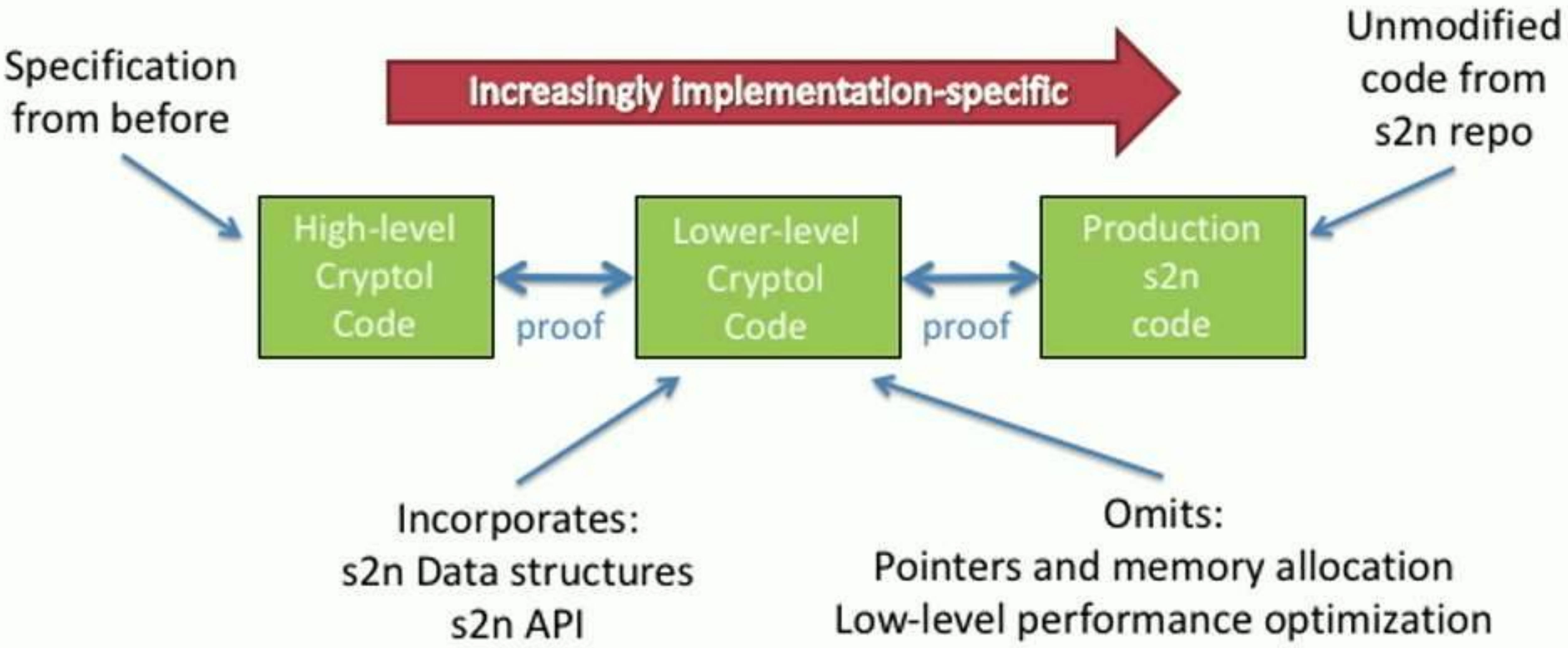
Summary of Approach

1. Write the formal specification.
2. Write some "scaffolding" to bridge the gap between specification and C code.
3. Apply automated tools.
4. Integrate into development environment.

About 2 months of effort.

Bridging the gap

Solution: Layers of Abstraction



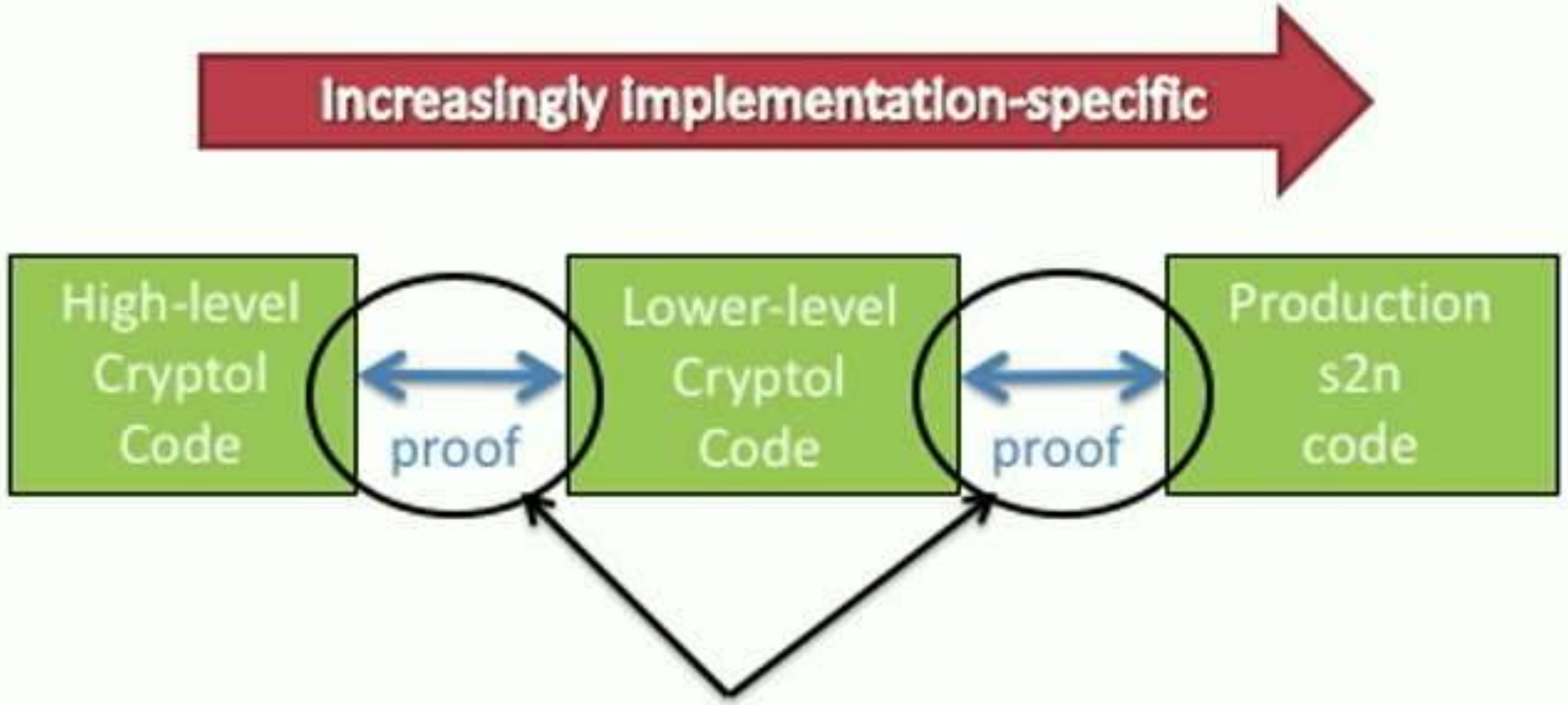
Summary of Approach

1. Write the formal specification.
2. Write some “scaffolding” to bridge the gap between Specification and C code.
3. Apply automated tools.
4. Integrate into development environment.

About 2 months of effort.

Bridging the gap

Solution: Layers of Abstraction



Automatically Constructed by SAW
(Software Analysis Workbench)
via translation to SMT and application of constraint solvers

Summary of Approach

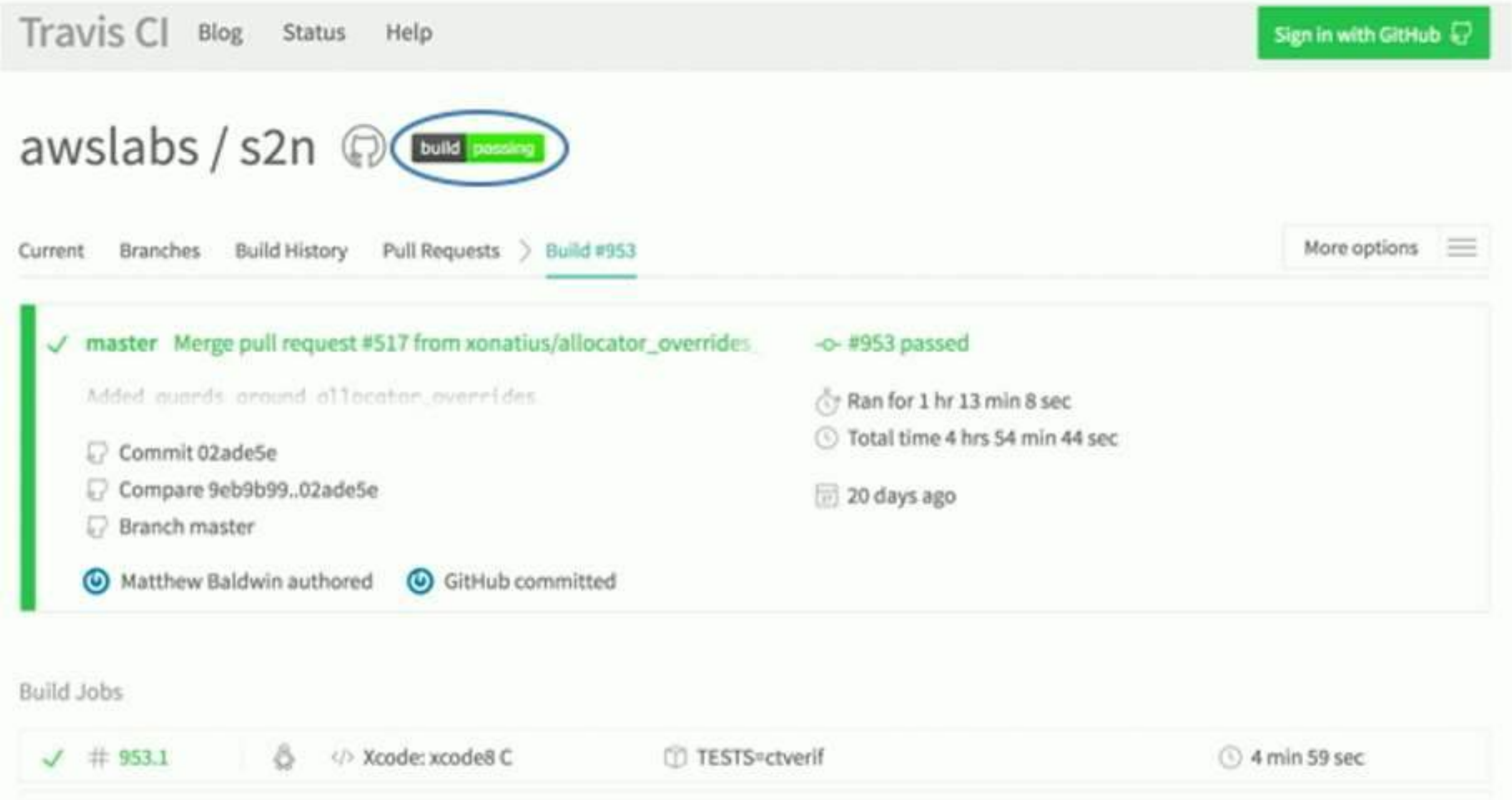
1. Write the formal specification.
2. Write some “scaffolding” to bridge the gap between Specification and C code.
3. Apply automated tools.
4. Integrate into development environment.

About 2 months of effort.

Continuous Integration

- Proofs run automatically on code changes
  - Proof failure is a build failure
- Proof is independent of exact C code, depends only on:
  - Interfaces (arguments and struct layouts)
  - Function call structure
- Proof is easily adapted:
  - Function body changes → likely no proof changes
  - Interface changes → similarly-sized proof changes
  - Call structure changes → tiny proof changes

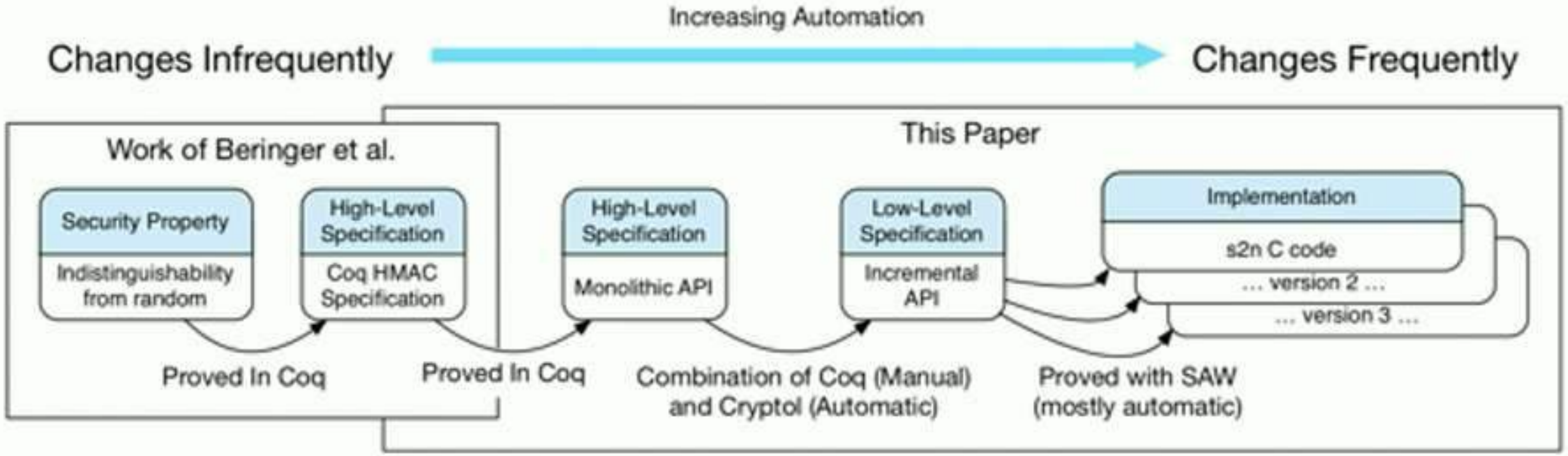
Travis CI


Travis CI
Blog
Status
Help
Sign in with GitHub
awslabs / s2n
build passing
Current
Branches
Build History
Pull Requests
Build #953
More options
master
Merge pull request #517 from xonatius/allocator_overrides
#953 passed
Added guards around allocator_overrides
Ran for 1 hr 13 min 8 sec
Total time 4 hrs 54 min 44 sec
Commit 02ade5e
Compare 9eb9b99..02ade5e
Branch master
20 days ago
Matthew Baldwin authored
GitHub committed
Build Jobs
953.1
Xcode: xcode8 C
TESTS=ctverif
4 min 59 sec

Continuous Integration

- Proofs run automatically on code changes
  - Proof failure is a build failure
- Proof is independent of exact C code, depends only on:
  - Interfaces (arguments and struct layouts)
  - Function call structure
- Proof is easily adapted:
  - Function body changes → likely no proof changes
  - Interface changes → similarly-sized proof changes
  - Call structure changes → tiny proof changes

Verified HMAC pipeline



Other s2n Work

- Also verified (see paper at **CAV'18**):
  - DRBG: Deterministic Random Bit Generator: The main source of cryptographic randomness
  - TLS Handshake protocol (state machine)
- Working on the rest
  - Correctness and security of parsing
  - Correct handling of keys in memory
  - Correct session management
  - Other crypto primitives, eg. HKDF

Conclusions

- For crypto / authentication / access control: **Behavioral bugs are security bugs**
- **Verification is feasible** for real-world cryptography
- Proof can be integrated into development workflow to prevent errors and give **continuous verification**
- Continuous analysis / verification is now being **applied in industry**


galois
galois.com/careers/

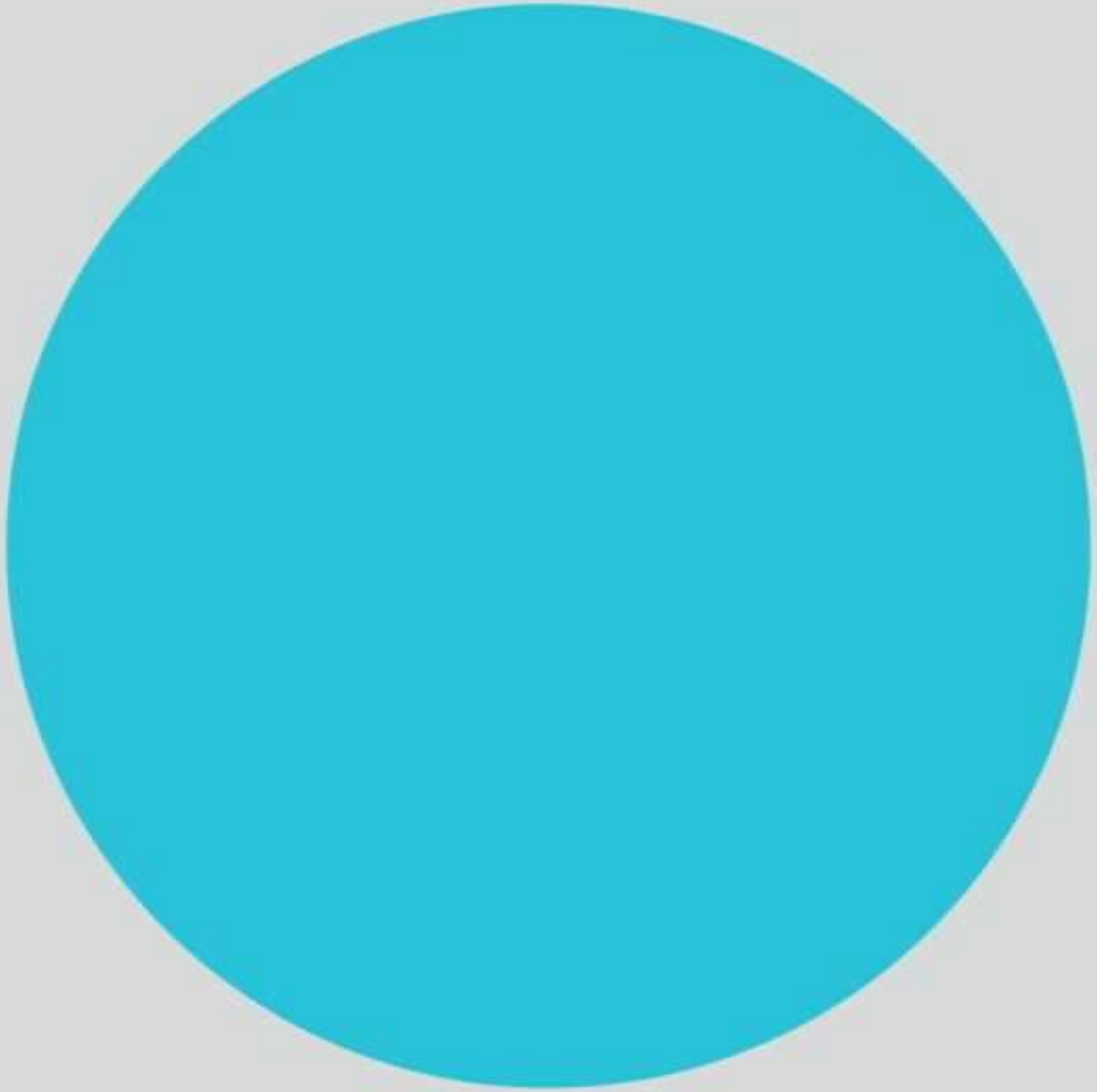
Helena: A web automation language for end users

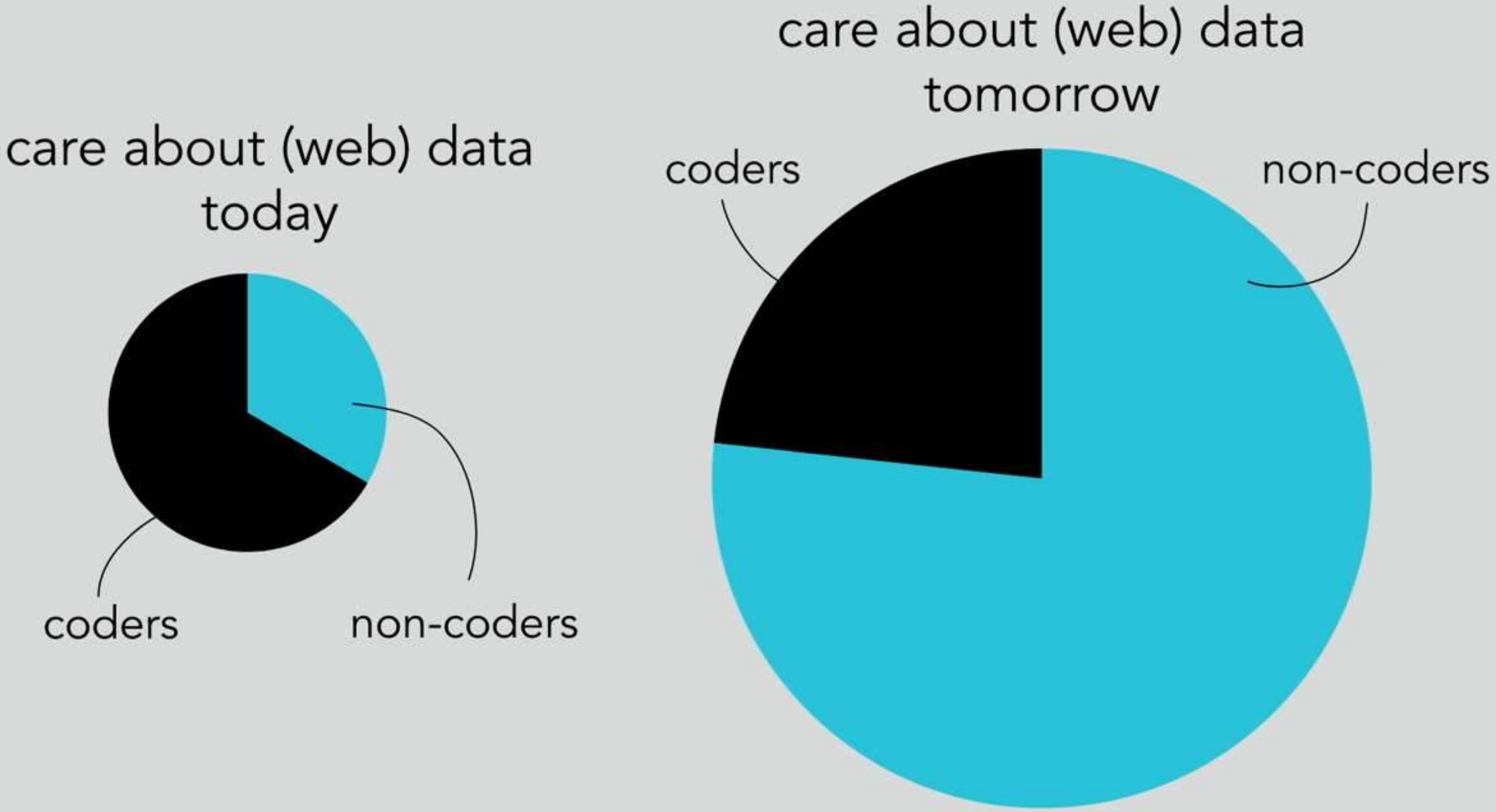
Sarah Chasins
University of California, Berkeley

Rastislav Bodik
University of Washington

care about (web) data
today

care about (web) data
tomorrow




care about (web) data today
coders
non-coders
care about (web) data tomorrow
coders
non-coders

DEPARTMENT OF ECONOMICS

UNIVERSITY *of* WASHINGTON

How is the minimum wage affecting Seattle restaurants?

CIVIL & ENVIRONMENTAL ENGINEERING

UNIVERSITY *of* WASHINGTON

Can we design a better carpool matching algorithm?

EVANS SCHOOL OF PUBLIC POLICY & GOVERNANCE

UNIVERSITY *of* WASHINGTON

How do charitable foundations communicate with supporters?

What web data collection tools do we have?

Scrapy

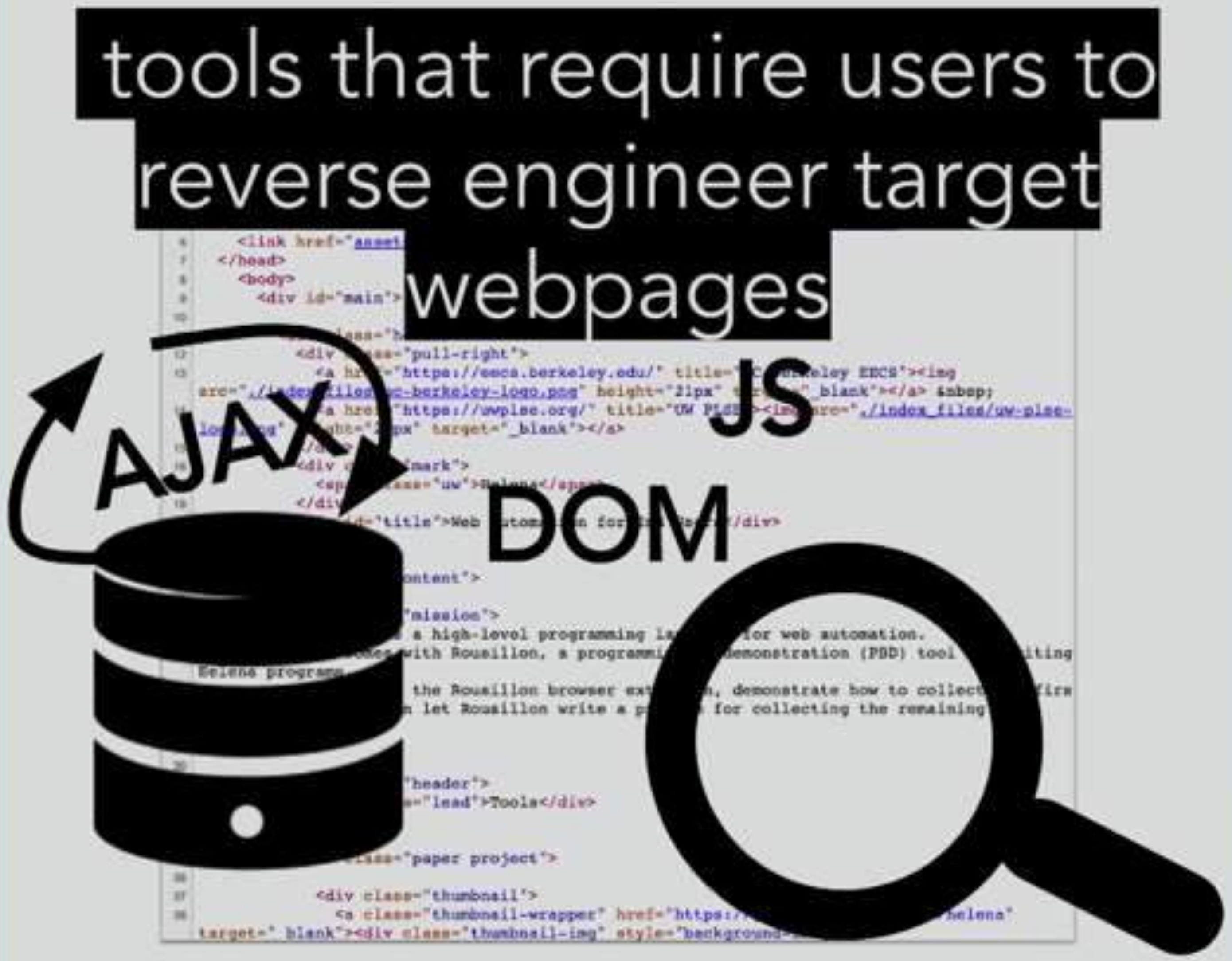
BeautifulSoup

Nokogiri 鋸 •••

coders

non-coders

What web data collection tools do we have?



coders

non-coders

What web data collection tools do we have?

tools that require users to reverse engineer target webpages

coders

- hire a human to copy & paste
- hire a coder to use one of these

non-coders

What web data collection tools do we have?

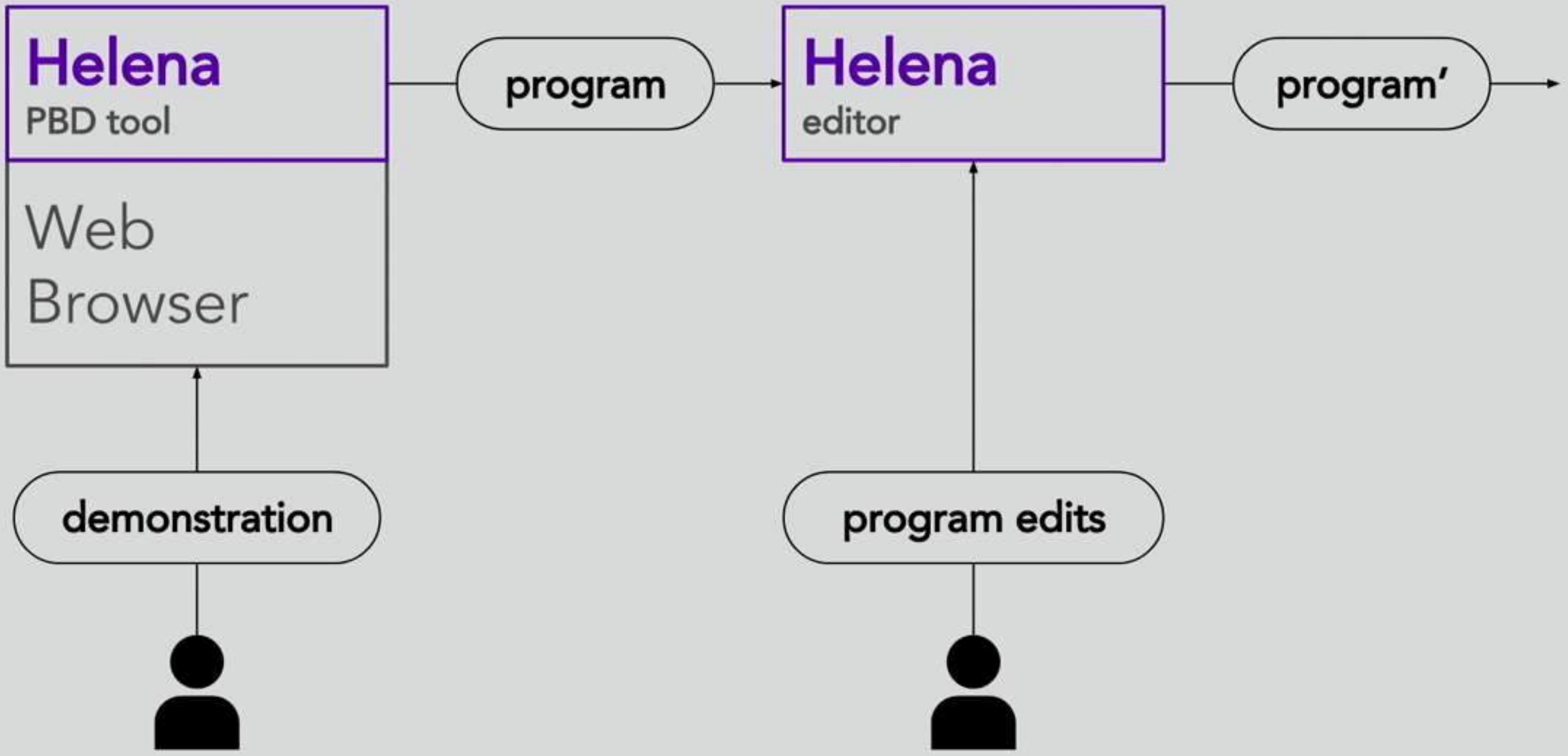
tools that require users to reverse engineer target webpages

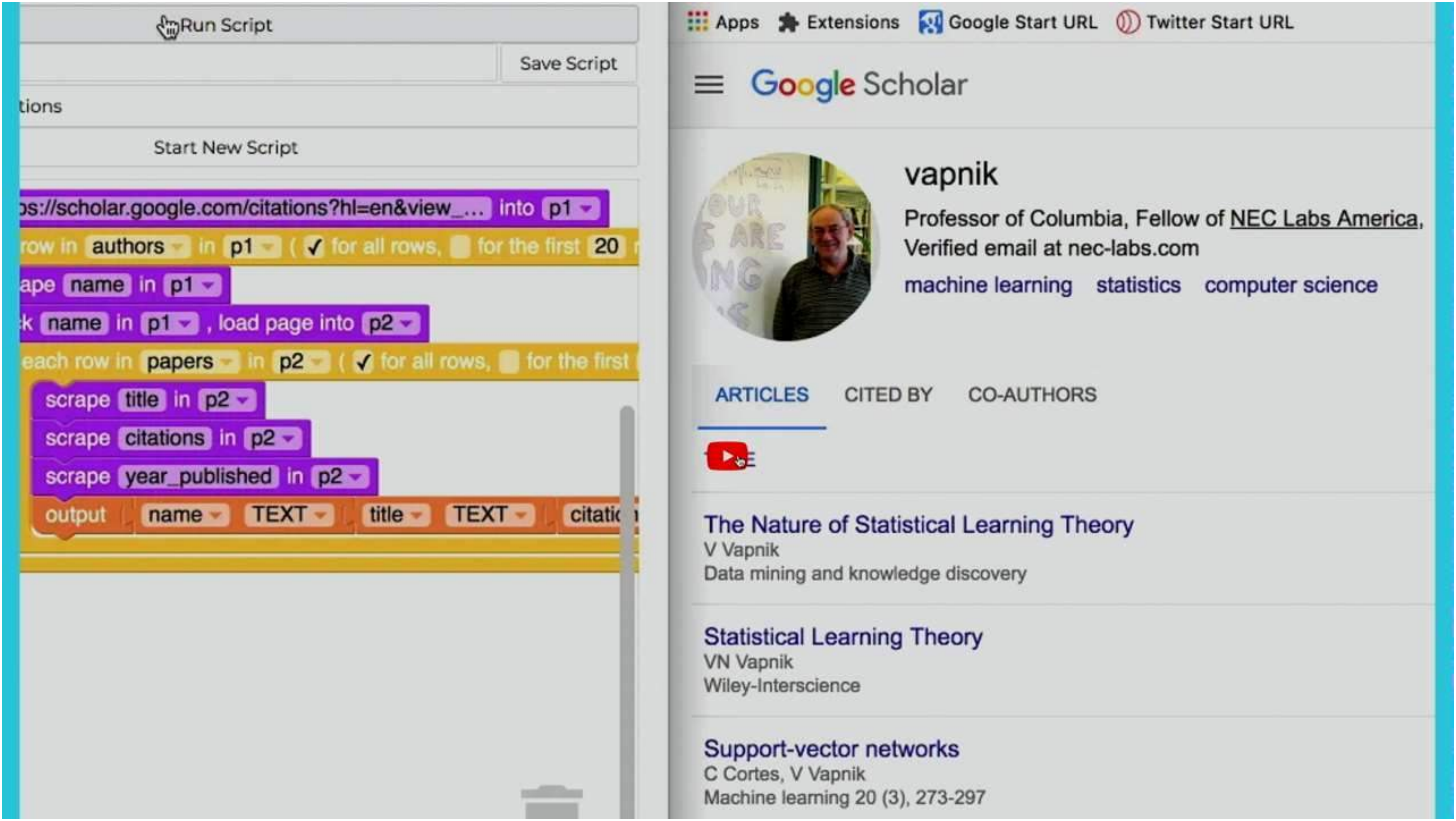
coders

- hire a human to copy & paste
- hire a coder to use one of these
- **Helena**
  WEB AUTOMATION FOR END USERS

our tool!

non-coders

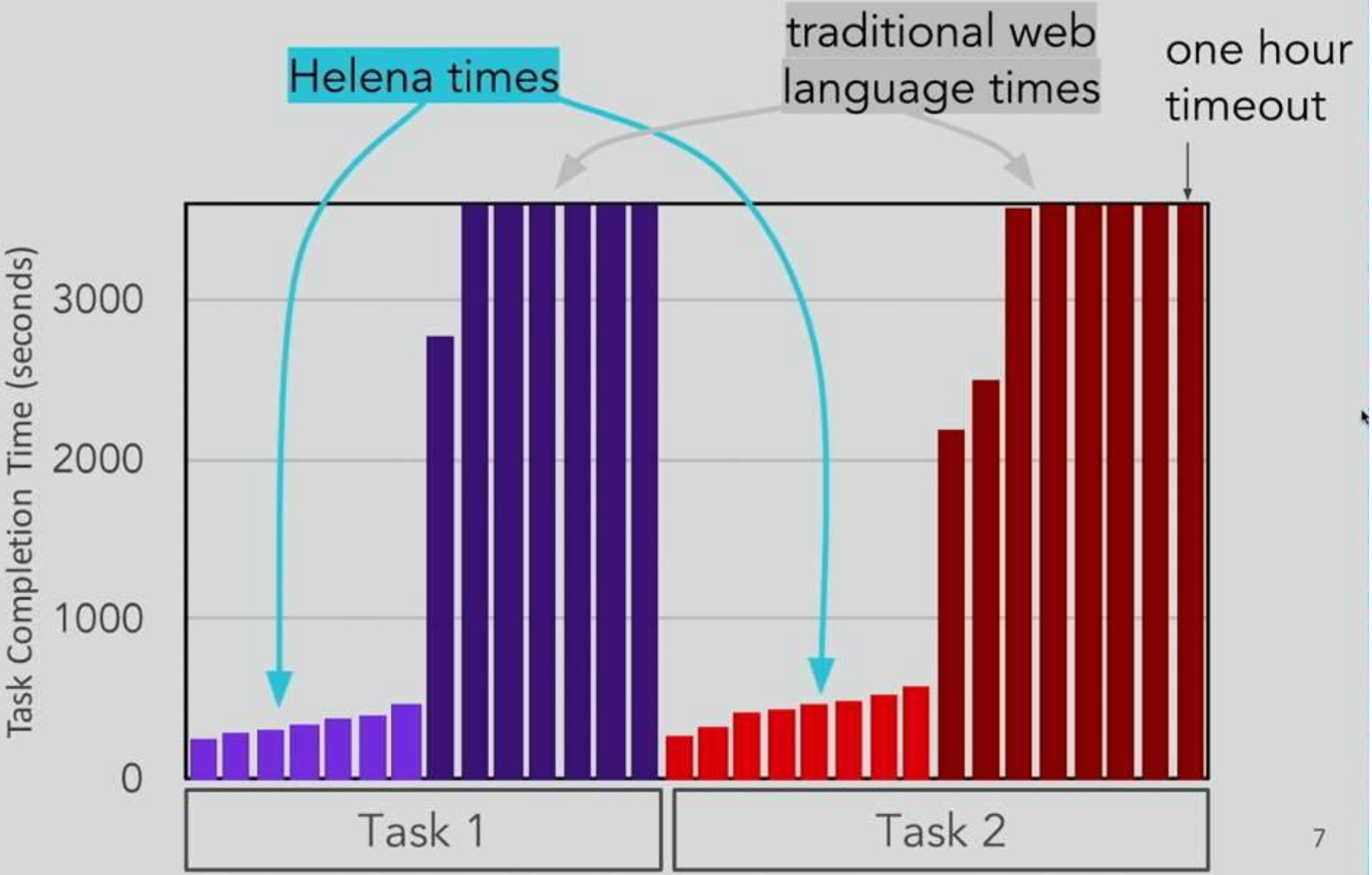

Helena
PBD tool
Web
Browser
program
Helena
editor
program'
demonstration
program edits


Run Script
Save Script
tions
Start New Script
os://scholar.google.com/citations?hl=en&view_... into p1
row in authors in p1 (✓ for all rows, for the first 20
ape name in p1
k name in p1 , load page into p2
each row in papers in p2 (✓ for all rows, for the first
scrape title in p2
scrape citations in p2
scrape year_published in p2
output name TEXT title TEXT citati
Apps
Extensions
Google Start URL
Twitter Start URL
Google Scholar
vapnik
Professor of Columbia, Fellow of NEC Labs America,
Verified email at nec-labs.com
machine learning
statistics
computer science
ARTICLES
CITED BY
CO-AUTHORS
The Nature of Statistical Learning Theory
V Vapnik
Data mining and knowledge discovery
Statistical Learning Theory
VN Vapnik
Wiley-Interscience
Support-vector networks
C Cortes, V Vapnik
Machine learning 20 (3), 273-297

Is this thing any good?

- Measured time to complete first task with each language
- Participants were **programmers**

lower is better



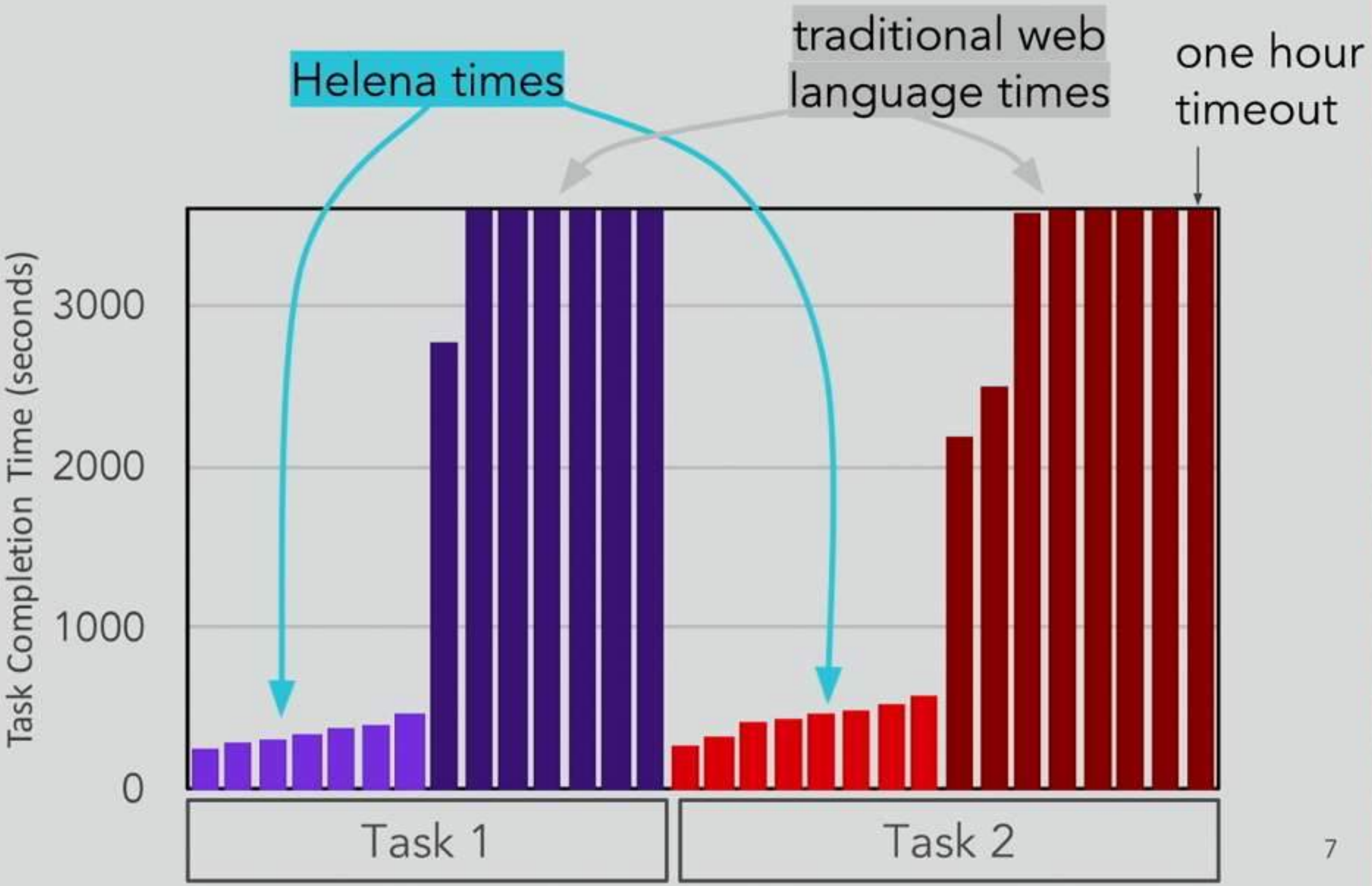
Is this thing any good?

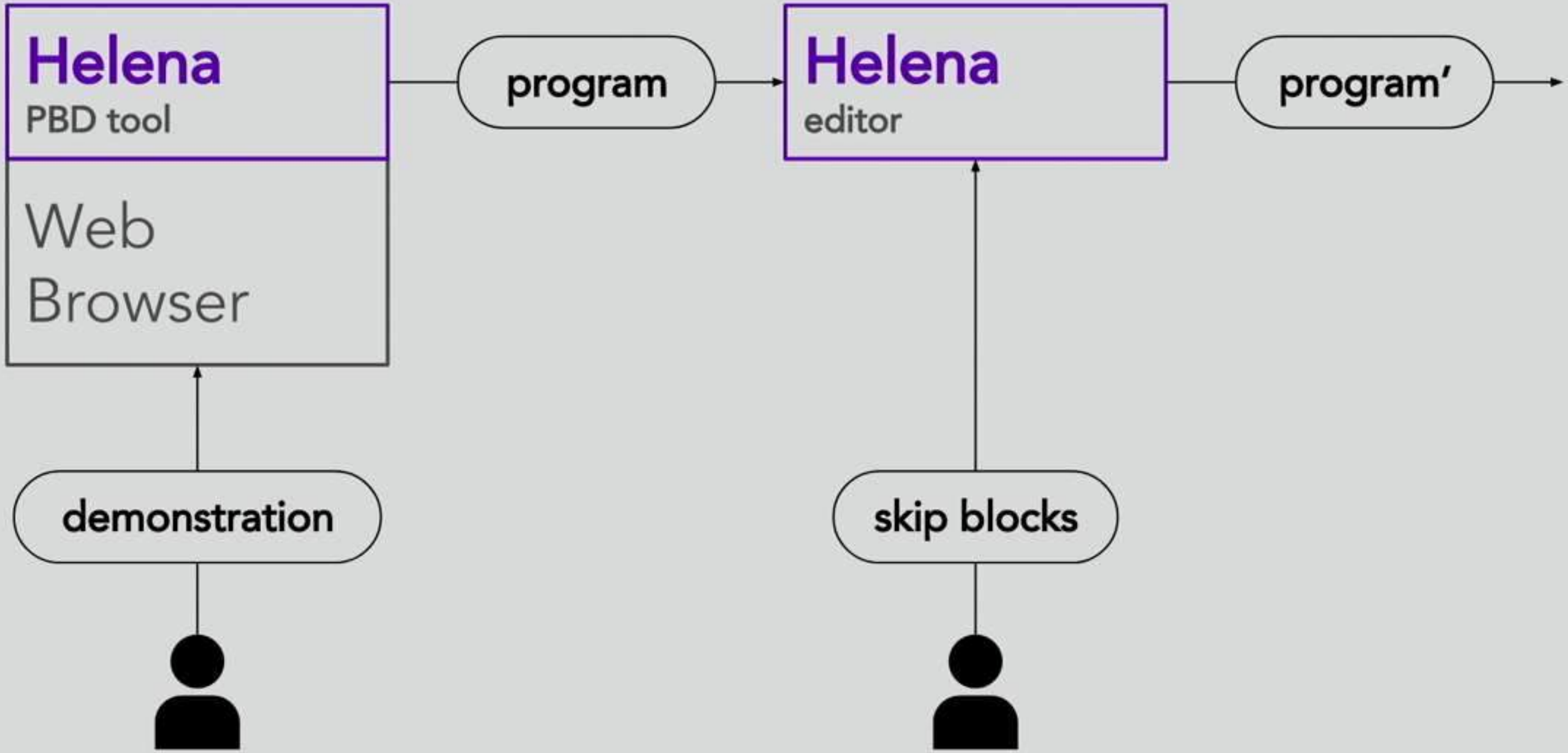
- Measured time to complete first task with each language
- Participants were **programmers**

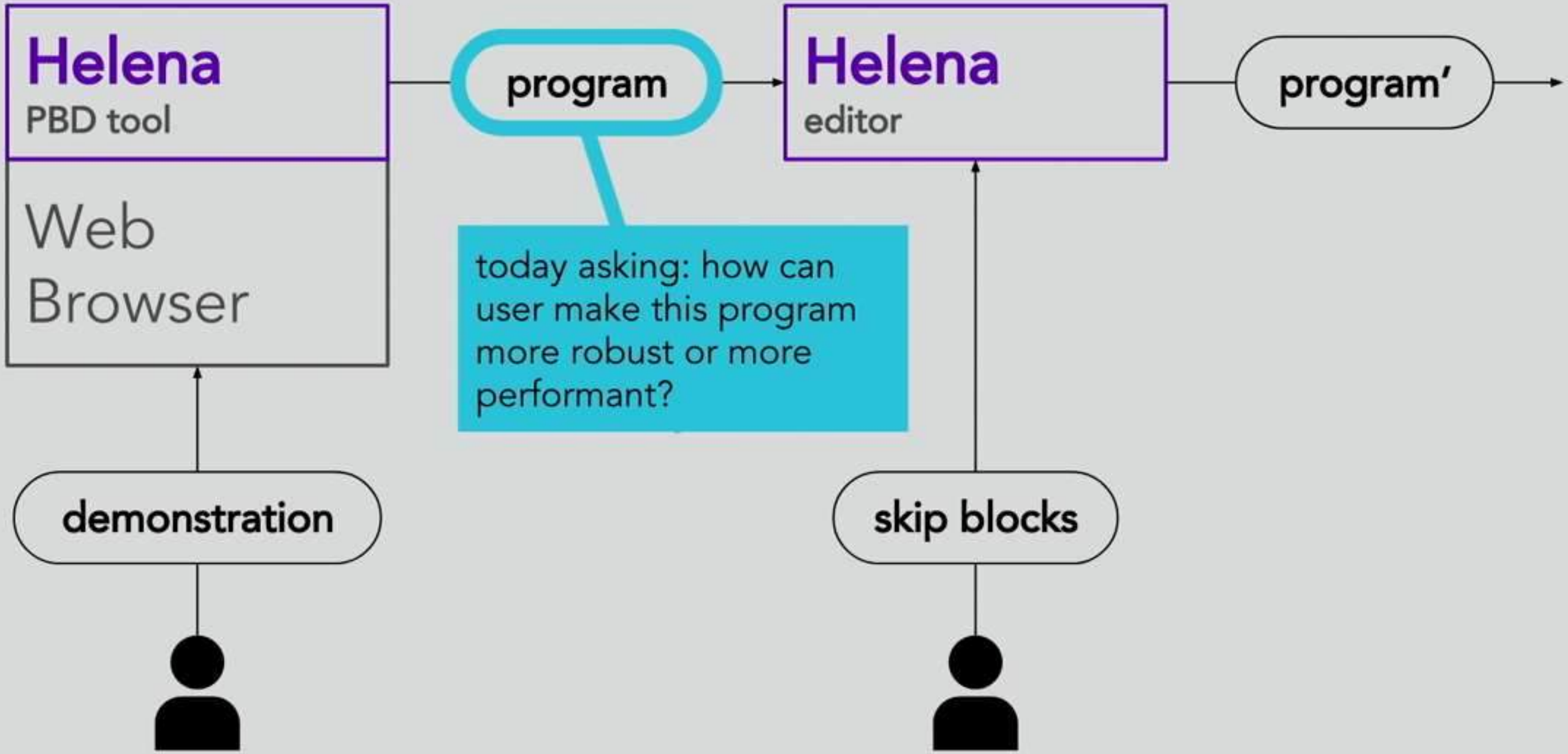
Programmers

8x

faster with Helena!




Helena
PBD tool
Web
Browser
program
Helena
editor
program'
demonstration
skip blocks


Helena
PBD tool
Web Browser
program
today asking: how can user make this program more robust or more performant?
Helena
editor
program'
demonstration
skip blocks

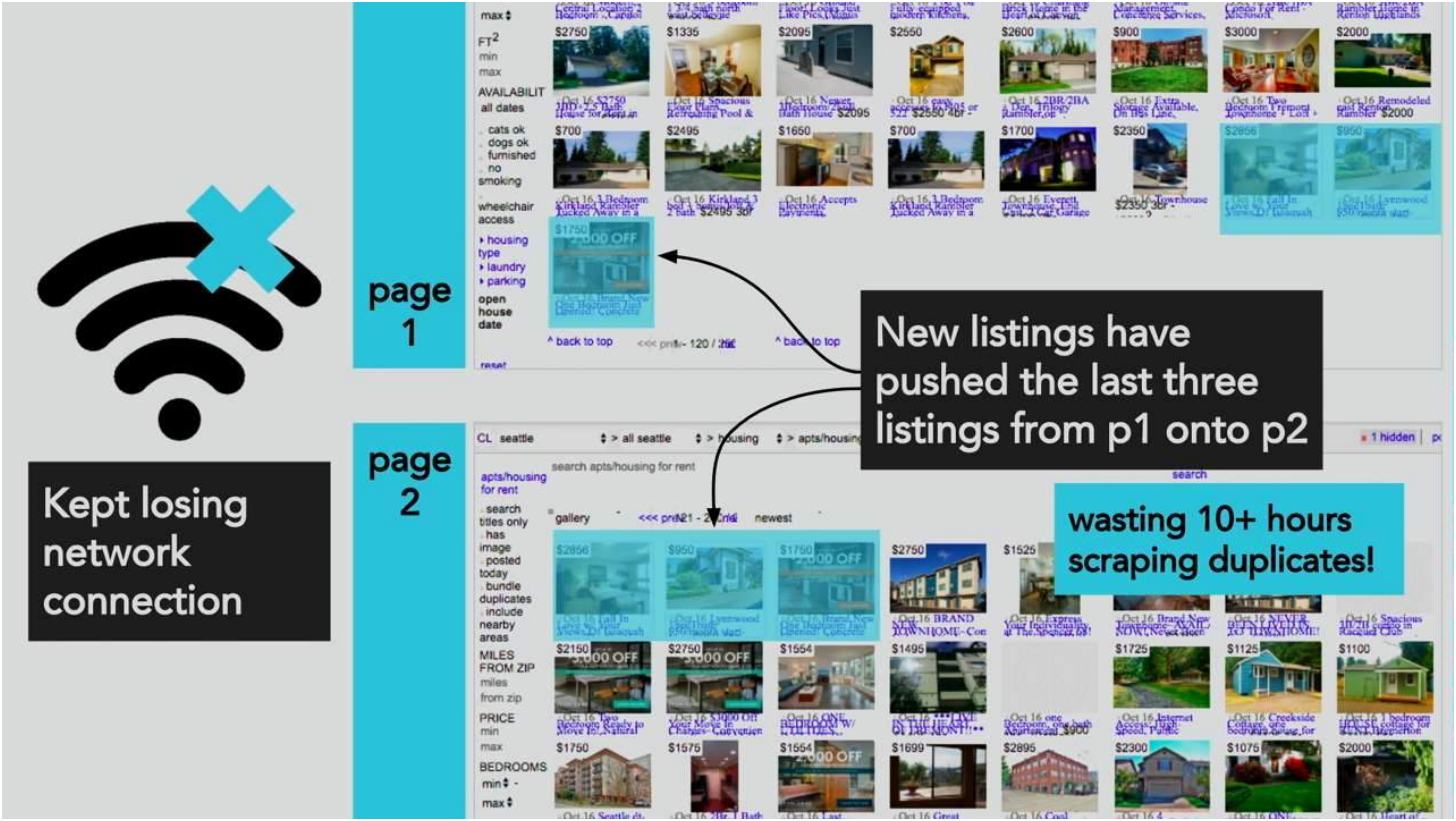
DEPARTMENT OF SOCIOLOGY
UNIVERSITY of WASHINGTON

+

City of Seattle

How is rent changing across Seattle neighborhoods?

Kept losing network connection


Kept losing network connection
page 1
page 2
New listings have pushed the last three listings from p1 onto p2
wasting 10+ hours scraping duplicates!

Problem Statement

(1) **Failures**: What happens when the network fails, the server fails, the computer fails? When we lose our session with the server and have to start over?

(2) **Data changes**: What happens when the server gives the client pages produced from different (potentially conflicting) reads of the underlying data store?

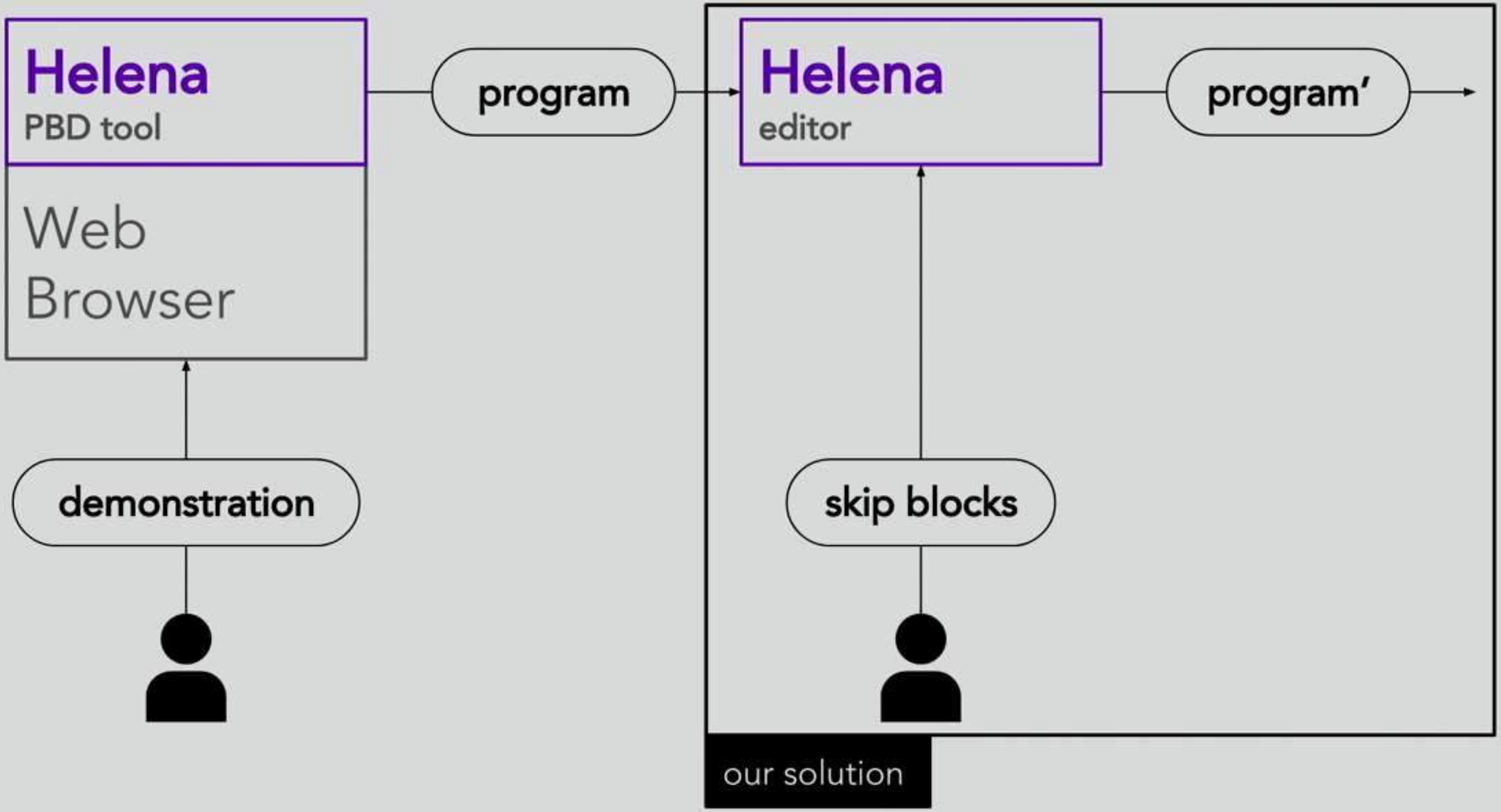
not client side problems → scraping script can't prevent them, must handle them

Problem Statement

(3) **Longitudinal scraping**: What if we want to run this over time?

(4) **Just slow**: What if this thing just takes 40 hours to run and we want to run it every 24 hours? Can we parallelize it?

(5) **Running into captchas**: What if running it from a single IP is running into rate-limiting? Can we distribute it?


Helena
PBD tool
Web
Browser
program
Helena
editor
program'
demonstration
skip blocks
our solution

Solution

on the surface,
seem like very
different problems

failures

data changes

Solution

on the surface,
seem like very
different problems

failures

data changes

"Just don't redo the same work you've already done!"

Solution

on the surface,
seem like very
different problems

failures

data changes

"Just don't redo the same work you've already done!"

But what's the 'same' work? After all, data changes...

Solution

on the surface, seem like very different problems

failures

data changes

"Just don't redo the same work you've already done!"

But what's the 'same' work? After all, data changes...

Our answer: the skip block! User can

- tell us what makes objects the same
- associate the code that operates on an object

Solution

on the surface, seem like very different problems

failures

data changes

“Just don’t redo the same work you’ve already done!”

But what’s the ‘same’ work? After all, data changes...

Our answer: the skip block! User can
- tell us what makes objects the same
- associate the code that operates on an object

- If object already committed (memoized), skip block; else, run block
- No reverse engineering! Reasoning about output data

End-user performance edits

text-ify-ed representation of the block language

```
for (aRow in p1.authors){

        scrape aRow.author_name
        scrape aRow.author_institution
        p2 = click aRow.author_name
        for (pRow in p2.papers){
             scrape pRow.title
             scrape pRow.citations
             output([aRow.author_name, pRow.title, pRow.citations])
        }

}
```

scrape stuff about the author, click the author

for the author's papers, scrape paper stuff

add a row of output with the author and paper info

End-user performance edits

```
for (aRow in p1.authors){
     skipBlock(Author(aRow.author_name, aRow.author_institution)){
          scrape aRow.author_name
          scrape aRow.author_institution
          p2 = click aRow.author_name
          for (pRow in p2.papers){
               scrape pRow.title
               scrape pRow.citations
               output([aRow.author_name, pRow.title, pRow.citations])
          }
     }
}
```

End-user performance edits

key attributes: is the current author the same as another we've already seen?

```
for (aRow in p1.authors){
     skipBlock(Author(aRow.author_name, aRow.author_institution)){
          scrape aRow.author_name
          scrape aRow.author_institution
          p2 = click aRow.author_name
          for (pRow in p2.papers){
               scrape pRow.title
               scrape pRow.citations
               output([aRow.author_name, pRow.title, pRow.citations])
          }
     }
}
```

block: the code that operates on the author object

if ever, **in any run**, script has committed an object with the same key attributes, skips the block

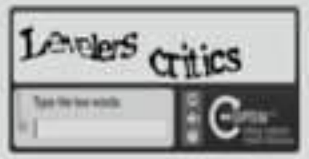
Let’s solve all those disparate problems

(1) **Extrinsic failures**: Objects that already made it into the commit log won’t be rescraped when we restart after a server failure.

(2) **Data changes**: When frequently updated data sources result in repeated objects/wasted work, we’ll skip them.

(3) **Longitudinal scraping**: Skip blocks automatically incrementalize; objects scraped in a prior run won’t be scraped again.

(4) **Just slow**: Each skip block represents an indpendent subtask. Split them across parallel worker processes.



(5) **Running into captchas**: Now split those worker processes across machines.

Fun design questions

- How should we handle nested skip blocks?
- Don't *always* want to skip if object seen before. When should we re-scrape, how should user communicate it?
- How should independent subtasks be split, communicated across parallel or distributed workers?

Data Change

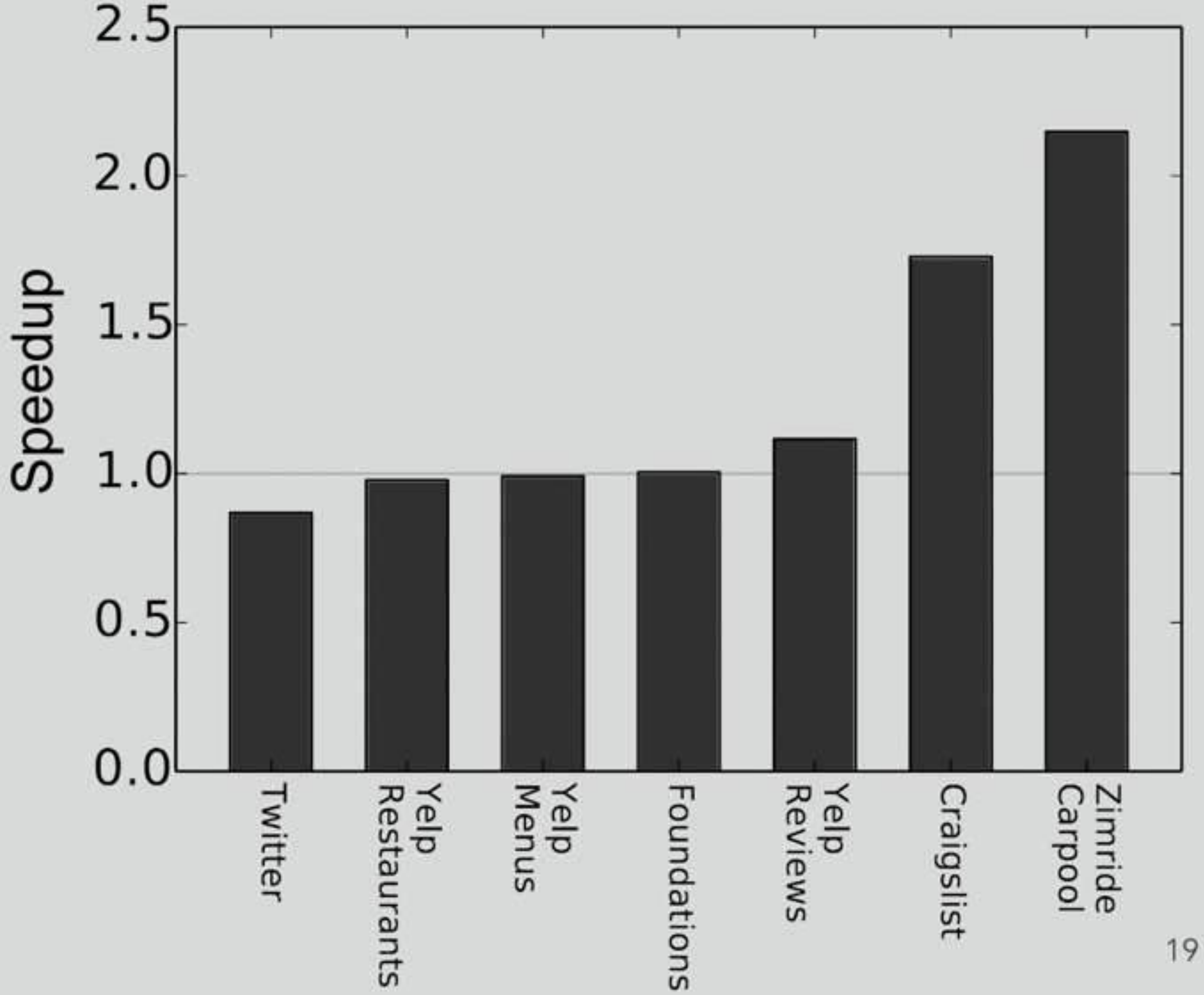
Speedup on the *first run*

Measured full execution time of:

- Script with skip blocks
- Script without skip blocks

Chart shows speedup from using skip blocks

higher is better



Parallelization

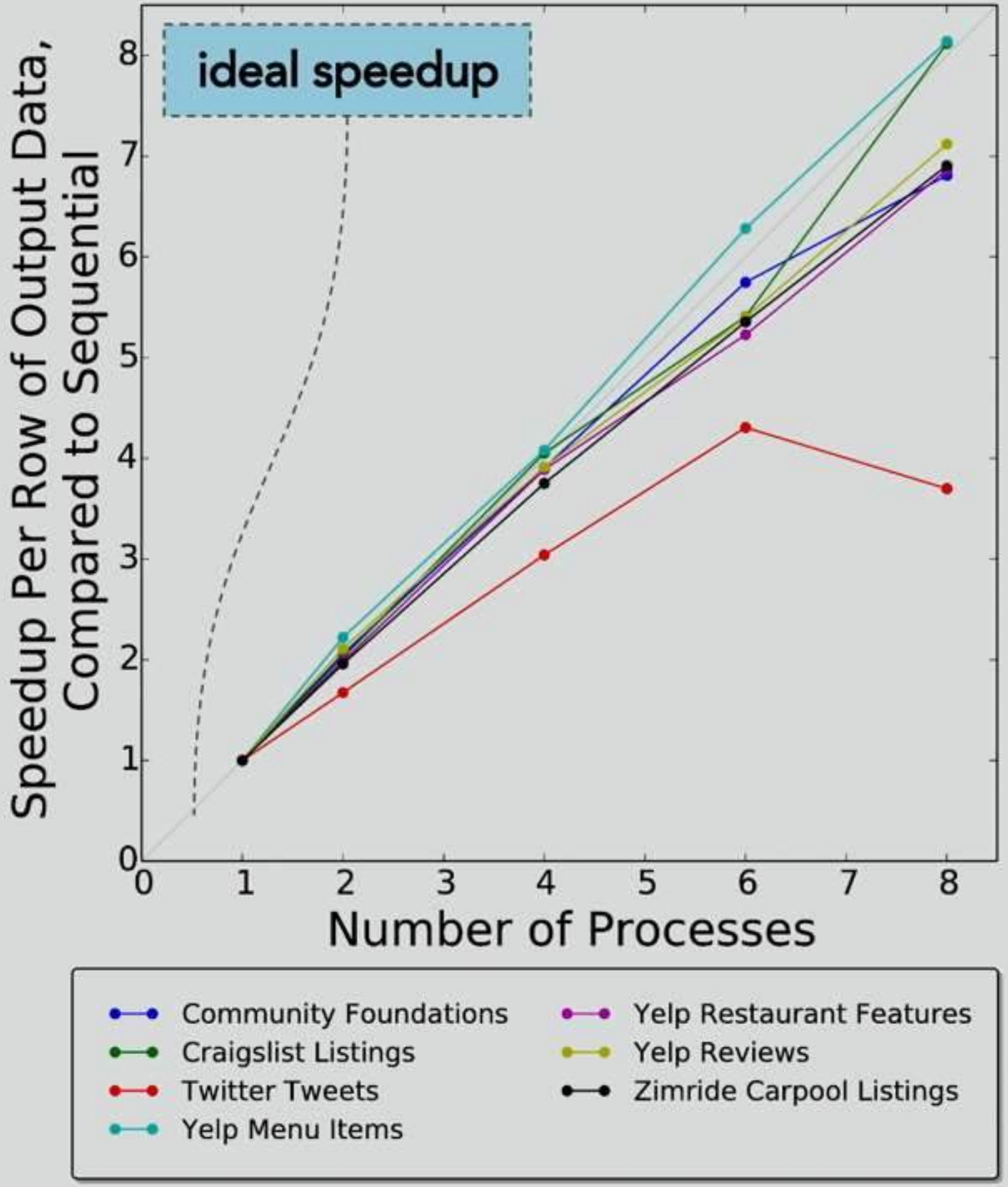
with lock-based skip block assignment

Measured full execution time (on a single machine) of:

- Script with one worker process
- Script with 2, 4, 6, or 8 worker processes

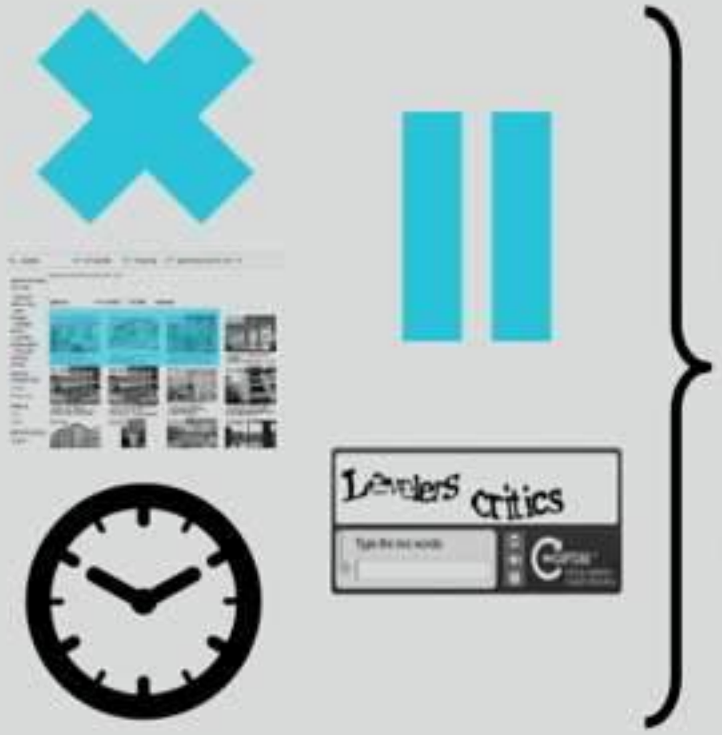
Chart shows speedup from using more processes

higher is better



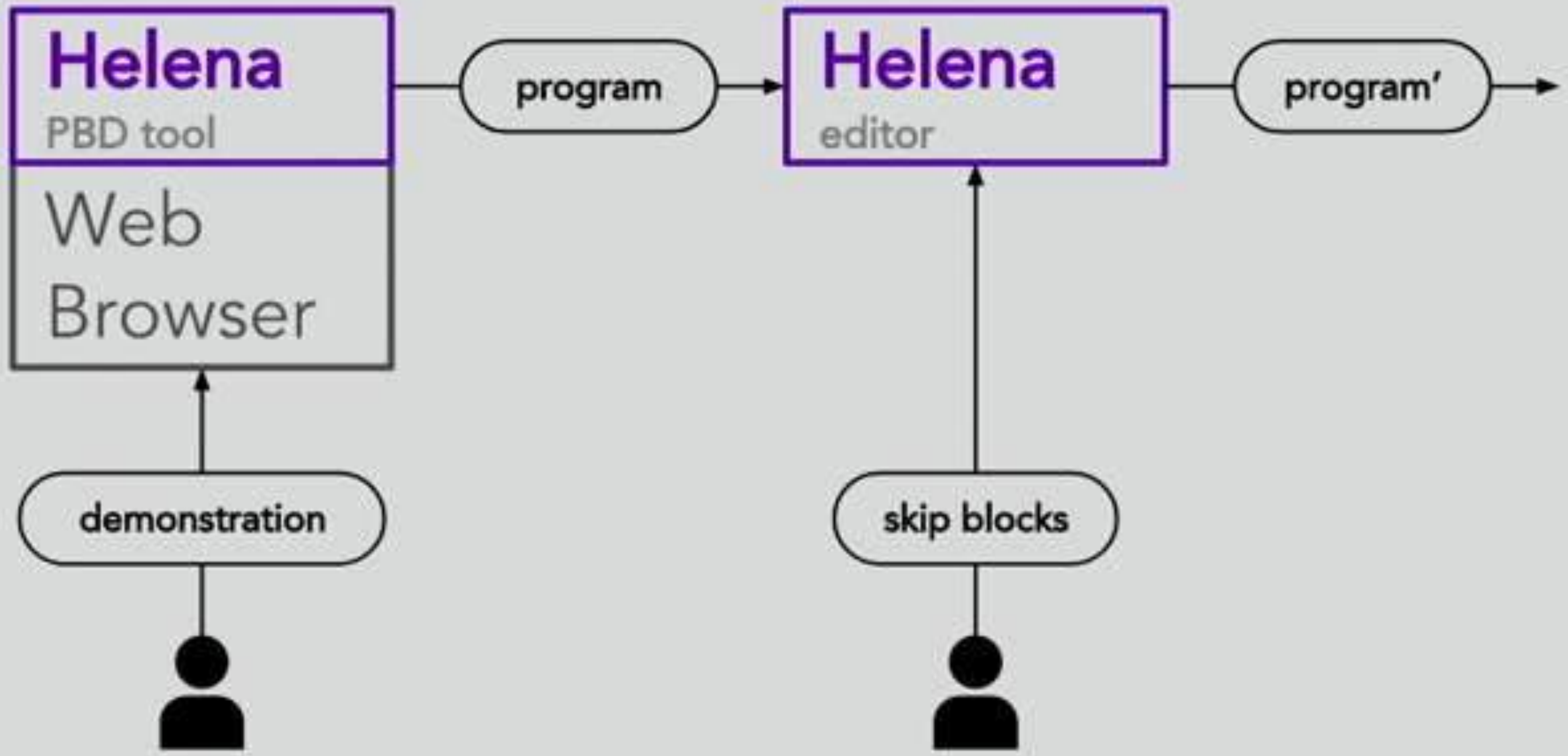
User Study

	coders	non-coders
Time to learn about skip blocks	2 minutes	7 minutes
Time to add one new skip block	52 seconds	61 seconds



Unified handling of apparently disparate challenges with a **single language construct**.

By keeping reasoning at the level of target output data, made skip blocks **usable by non-programmers**.



helena-lang.org

contact: schasins@cs.berkeley.edu

github.com/schasins/helena

Sinking Point

Bill Zorn
Dan Grossman
Zach Tatlock

IEEE 754 floating-point

Fast, portable, completely specified

Each operation is correctly rounded

Sometimes behavior is similar to real numbers, sometimes not

- Can be difficult to reason about

Examples

```
$ python
Python 3.6.3 |Anaconda, Inc.| (default, Oct 13 2017, 12:02:49)
[GCC 7.2.0] on linux
>>> import math
>>> math.pi + 1e16 - 1e16
```

Examples

```
$ python
Python 3.6.3 |Anaconda, Inc.| (default, Oct 13 2017, 12:02:49)
[GCC 7.2.0] on linux
>>> import math
>>> math.pi + 1e16 - 1e16
4.0
```

$$ax^2 + bx + c = 0$$

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

$$ax^2 + bx + c = 0$$

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

a	*b*	*c*	*x* (IEEE 754 double)
.1	2	3	-1.6333997346592444

$$ax^2 + bx + c = 0 \qquad x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

a	*b*	*c*	*x* (IEEE 754 double)
.1	2	3	-1.6333997346592444
.001	2	3	-1.5011266906707066
1e-9	2	3	-1.500000013088254

$$ax^2 + bx + c = 0 \qquad x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

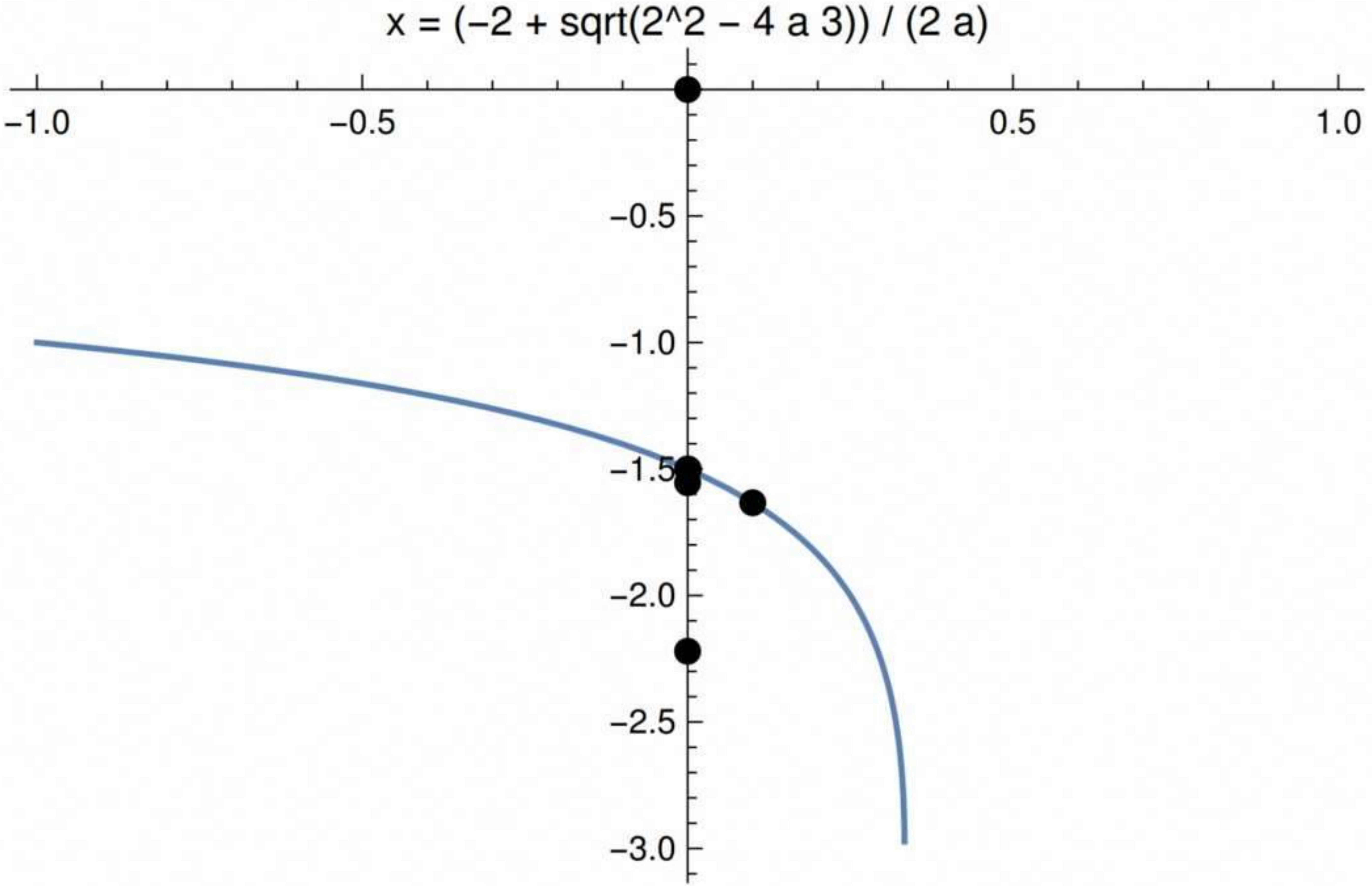
a	*b*	*c*	*x* (IEEE 754 double)
.1	2	3	-1.6333997346592444
.001	2	3	-1.5011266906707066
1e-9	2	3	-1.500000013088254
1e-15	2	3	-1.5543122344752189

$$ax^2 + bx + c = 0 \qquad x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

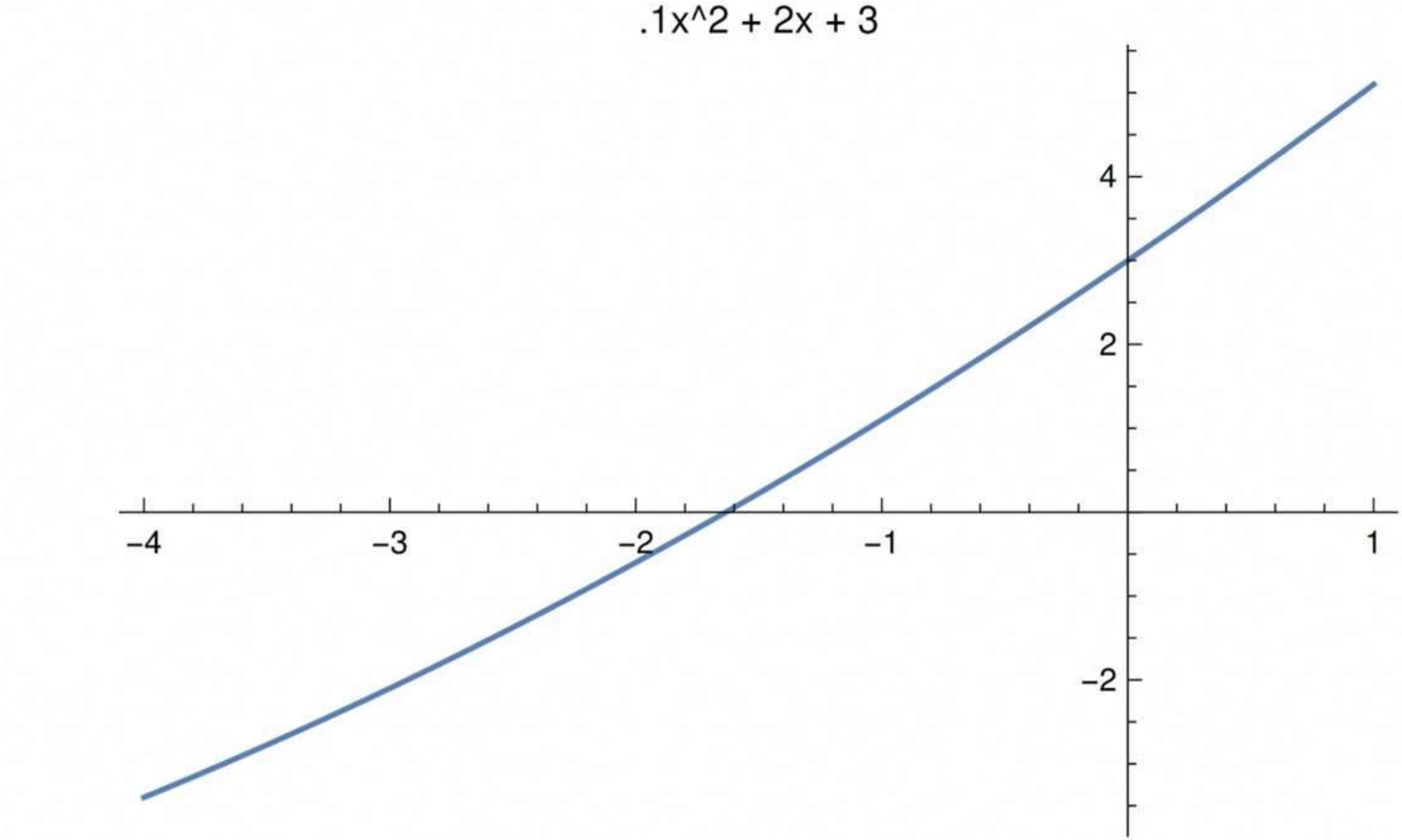
a	*b*	*c*	*x* (IEEE 754 double)
.1	2	3	-1.6333997346592444
.001	2	3	-1.5011266906707066
1e-9	2	3	-1.500000013088254
1e-15	2	3	-1.5543122344752189
1e-16	2	3	-2.220446049250313

$$ax^2 + bx + c = 0 \qquad x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

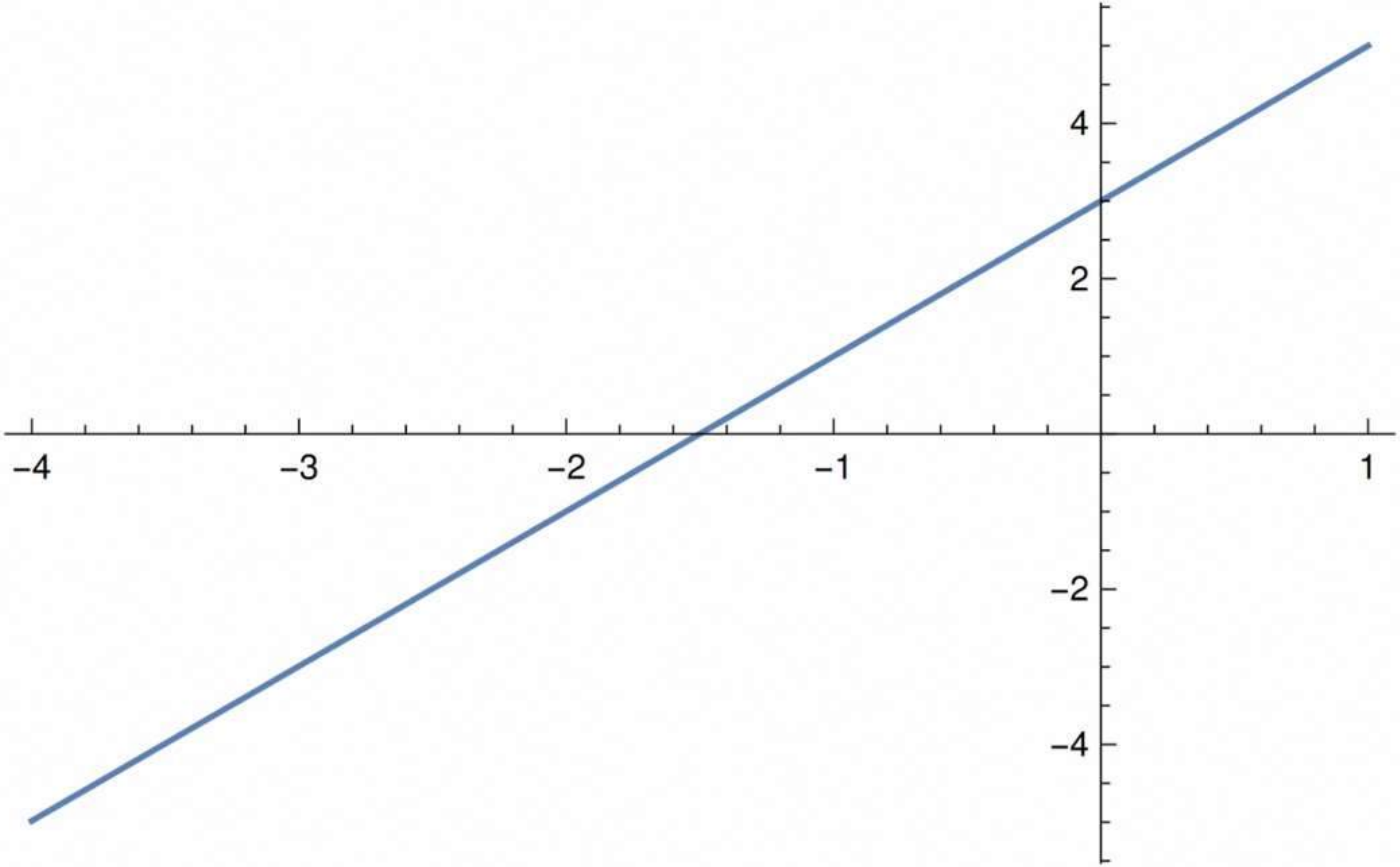
a	*b*	*c*	*x* (IEEE 754 double)
.1	2	3	-1.6333997346592444
.001	2	3	-1.5011266906707066
1e-9	2	3	-1.500000013088254
1e-15	2	3	-1.5543122344752189
1e-16	2	3	-2.2204460492503131
1e-17	2	3	0

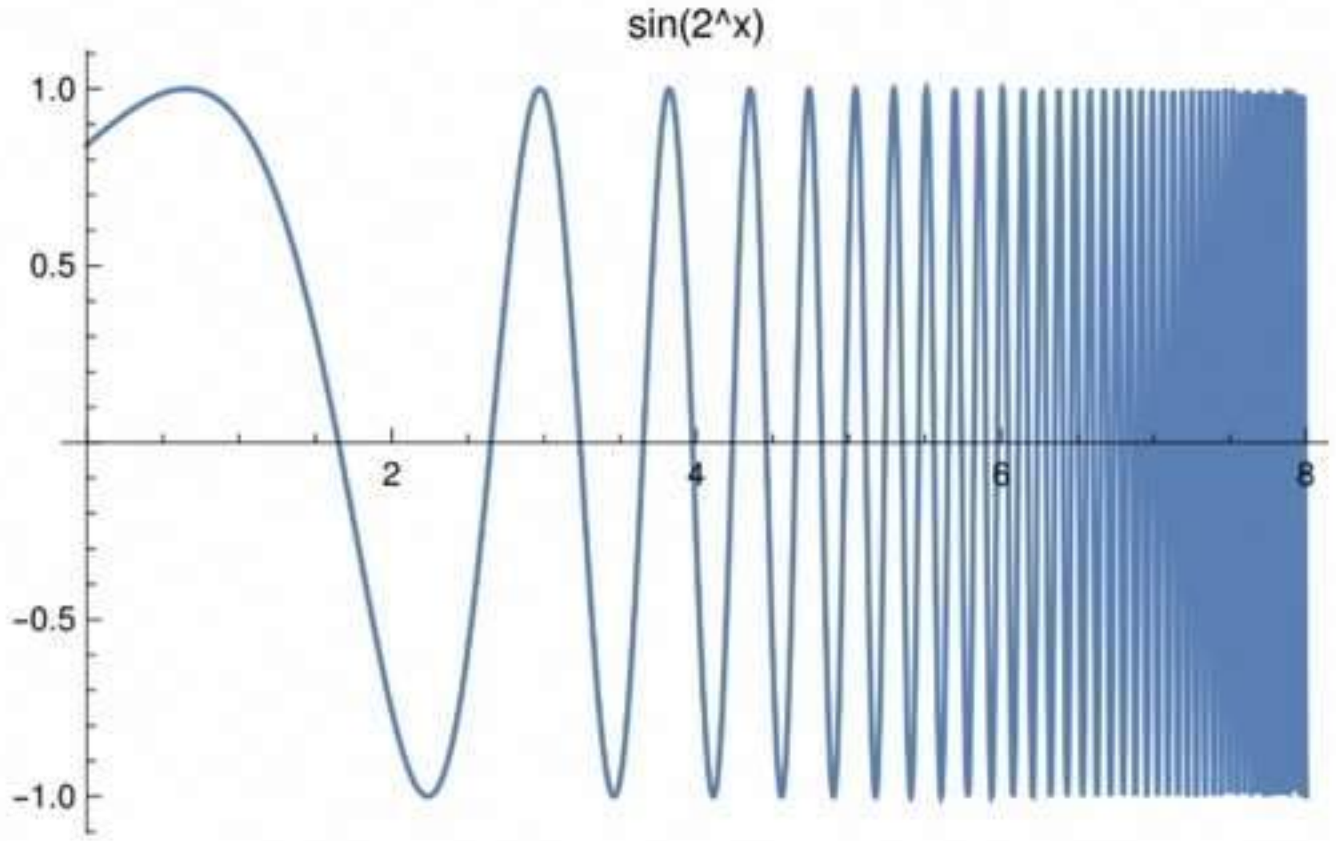


x = (−2 + sqrt(2^2 − 4 a 3)) / (2 a)
−1.0
−0.5
0.5
1.0
−0.5
−1.0
−1.5
−2.0
−2.5
−3.0

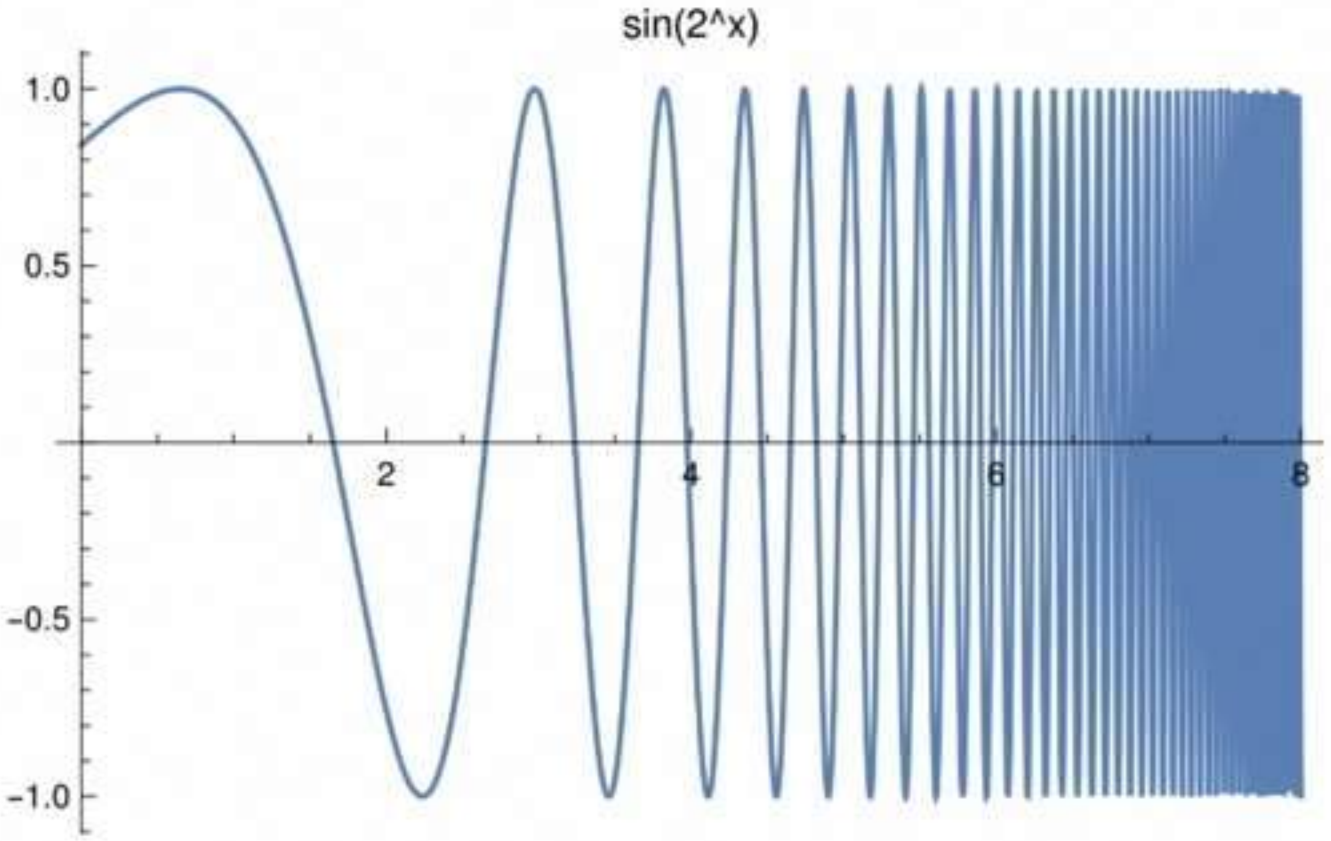

.1x^2 + 2x + 3
4
2
−2
−4
−3
−2
−1
1

.001x^2 + 2x + 3

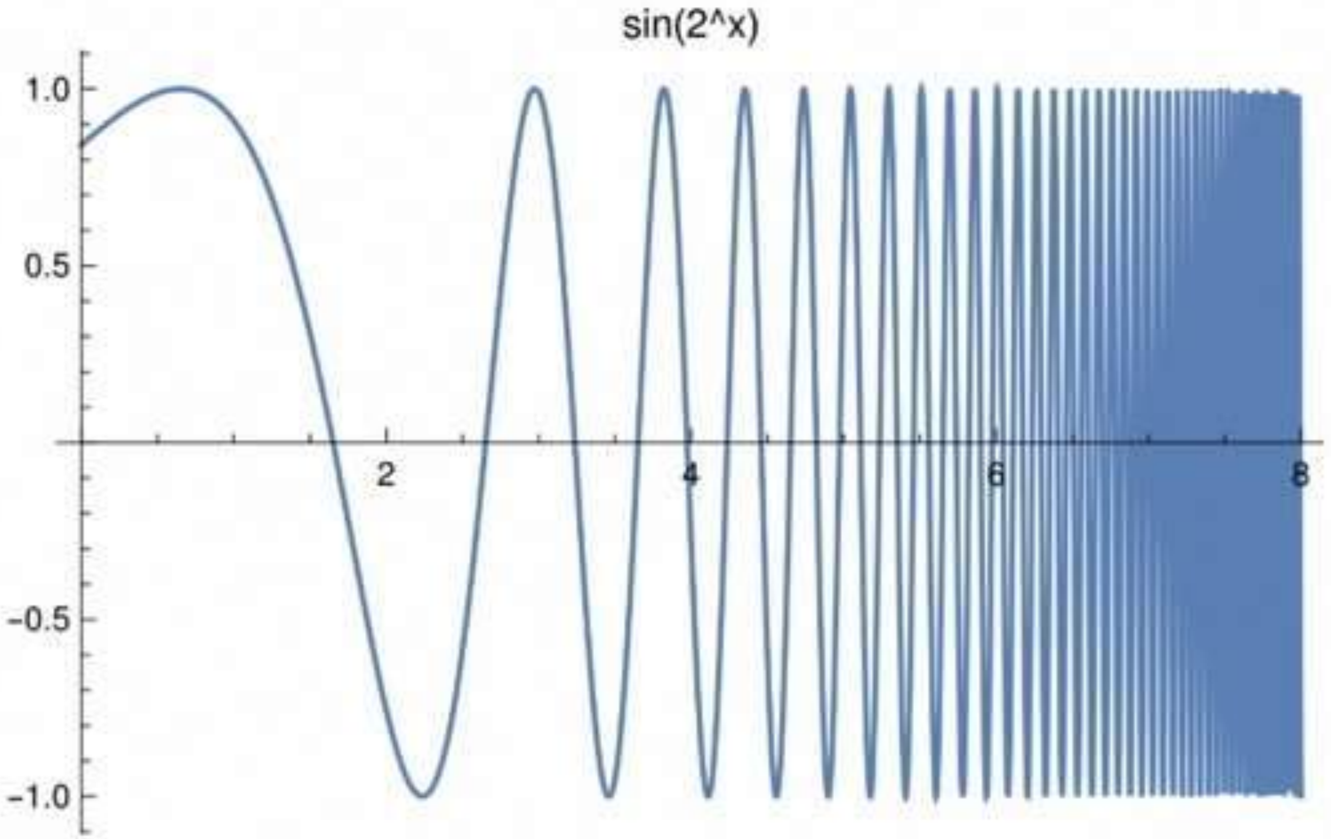




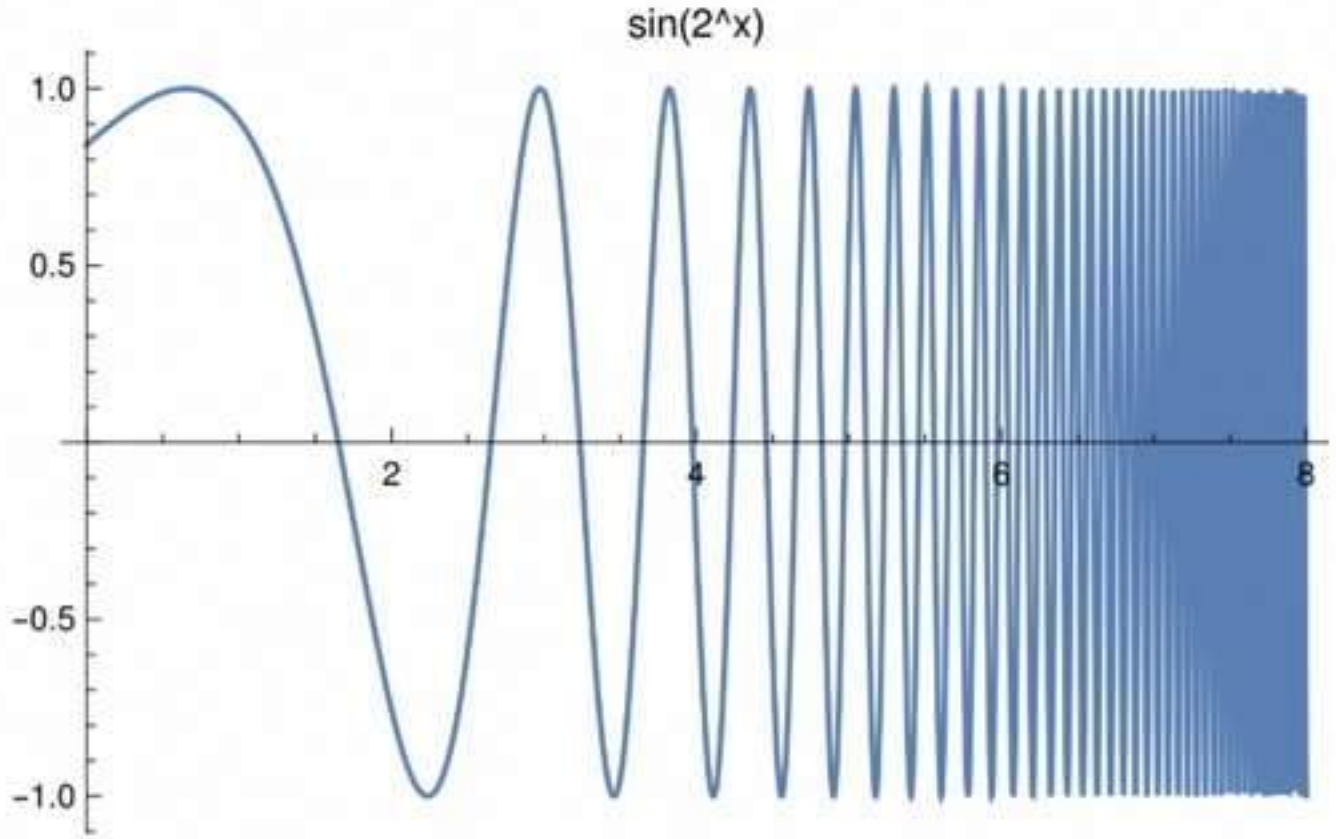
sin(2^x)
1.0
0.5
-0.5
-1.0
2
4
6
8



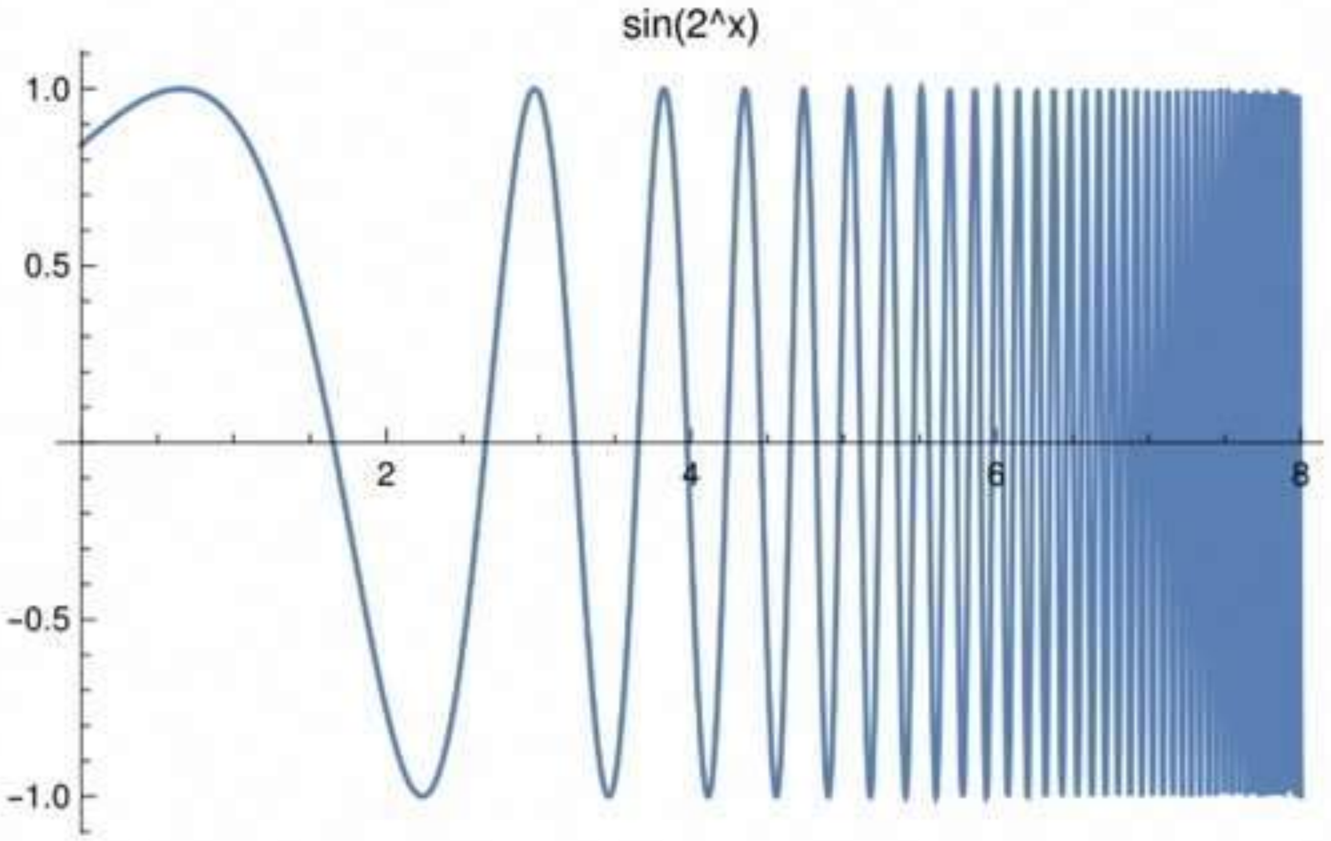
x	IEEE 754 double	Exact
15	0.92785633341392471	0.92785633341392471



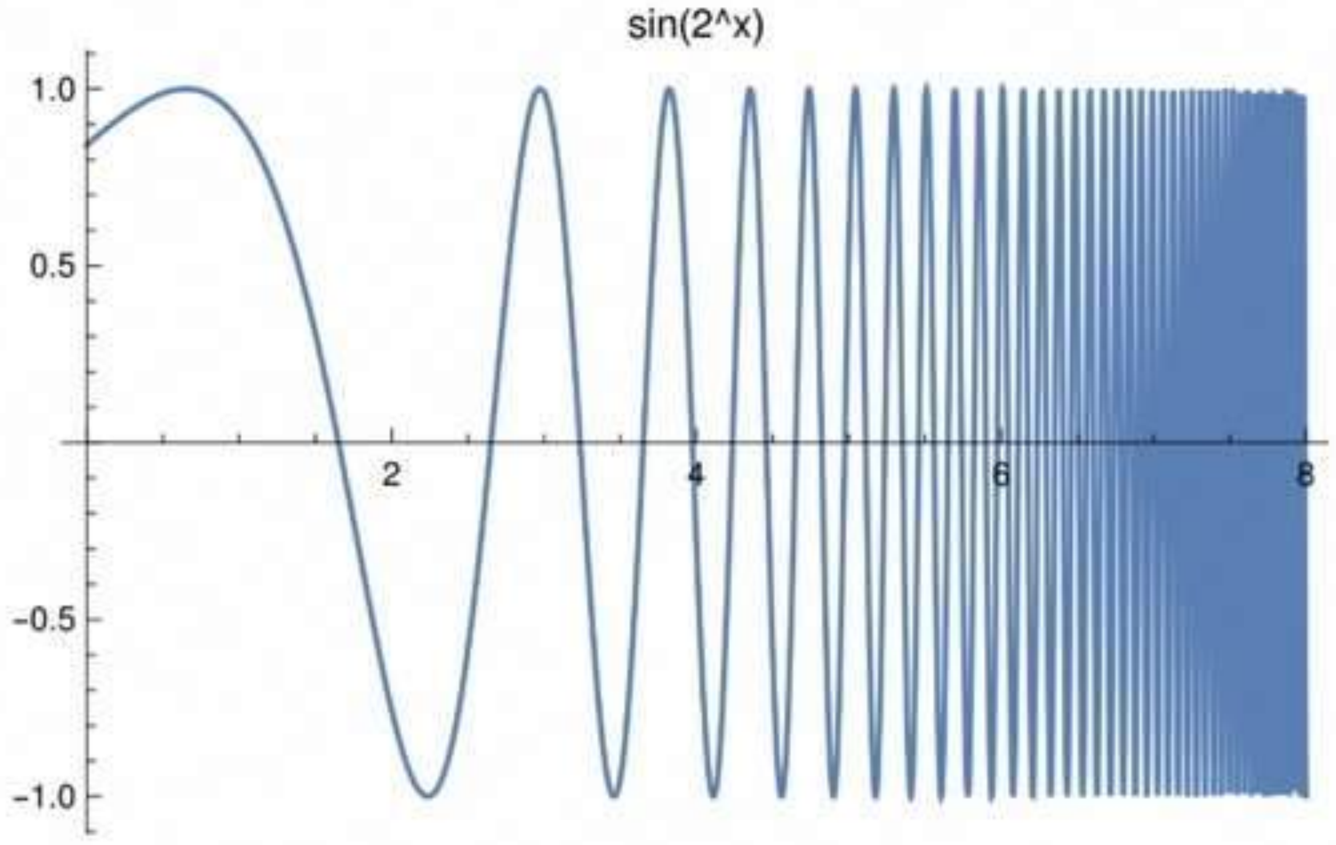
x	IEEE 754 double	Exact
15	0.92785633341392471	0.92785633341392471
25.3	-0.77866261130416781	-0.77866260949297472



x	IEEE 754 double	Exact
15	0.92785633341392471	0.92785633341392471
25.3	-0.77866261130416781	-0.77866260949297472
60	-0.83064921763725463	-0.83064921763725463



x	IEEE 754 double	Exact
15	0.92785633341392471	0.92785633341392471
25.3	-0.77866261130416781	-0.77866260949297472
60	-0.83064921763725463	-0.83064921763725463
60.01	-0.94644010175472126	0.83258728783816349



x	IEEE 754 double	Exact
15	0.92785633341392471	0.92785633341392471
25.3	-0.77866261130416781	-0.77866260949297472
60	-0.83064921763725463	-0.83064921763725463
60.01	-0.94644010175472126	0.83258728783816349
60.125	0.27925526776125142	-0.75293133122316314

None of these examples are surprising

Floating-point is fully specified

Just doing what the specification says

All results have the same amount of precision

Up to the programmer to determine if the bits are meaningful

Sinking Point

Dynamically reduce precision, rather than returning meaningless bits

Not any more accurate than IEEE 754 floating-point

    But the precision you do get corresponds to behavior of reals

Modest overhead: a few extra bits, a few extra bitwise operations

Examples

```
$ python
Python 3.6.3 |Anaconda, Inc.| (default, Oct 13 2017, 12:02:49)
[GCC 7.2.0] on linux
>>> import math
>>> math.pi + 1e16 - 1e16
4.0
```

Examples

```
$ python
Python 3.6.3 |Anaconda, Inc.| (default, Oct 13 2017, 12:02:49)
[GCC 7.2.0] on linux
>>> import math
>>> math.pi + 1e16 - 1e16
4.0000000000000000
```

Examples

```
$ python
Python 3.6.3 |Anaconda, Inc.| (default, Oct 13 2017, 12:02:49)
[GCC 7.2.0] on linux
>>> import math
>>> math.pi + 1e16 - 1e16
4.0000000000000000
>>> import sink
>>> sink('pi') + sink('1e16') - sink('1e16')
4~
```

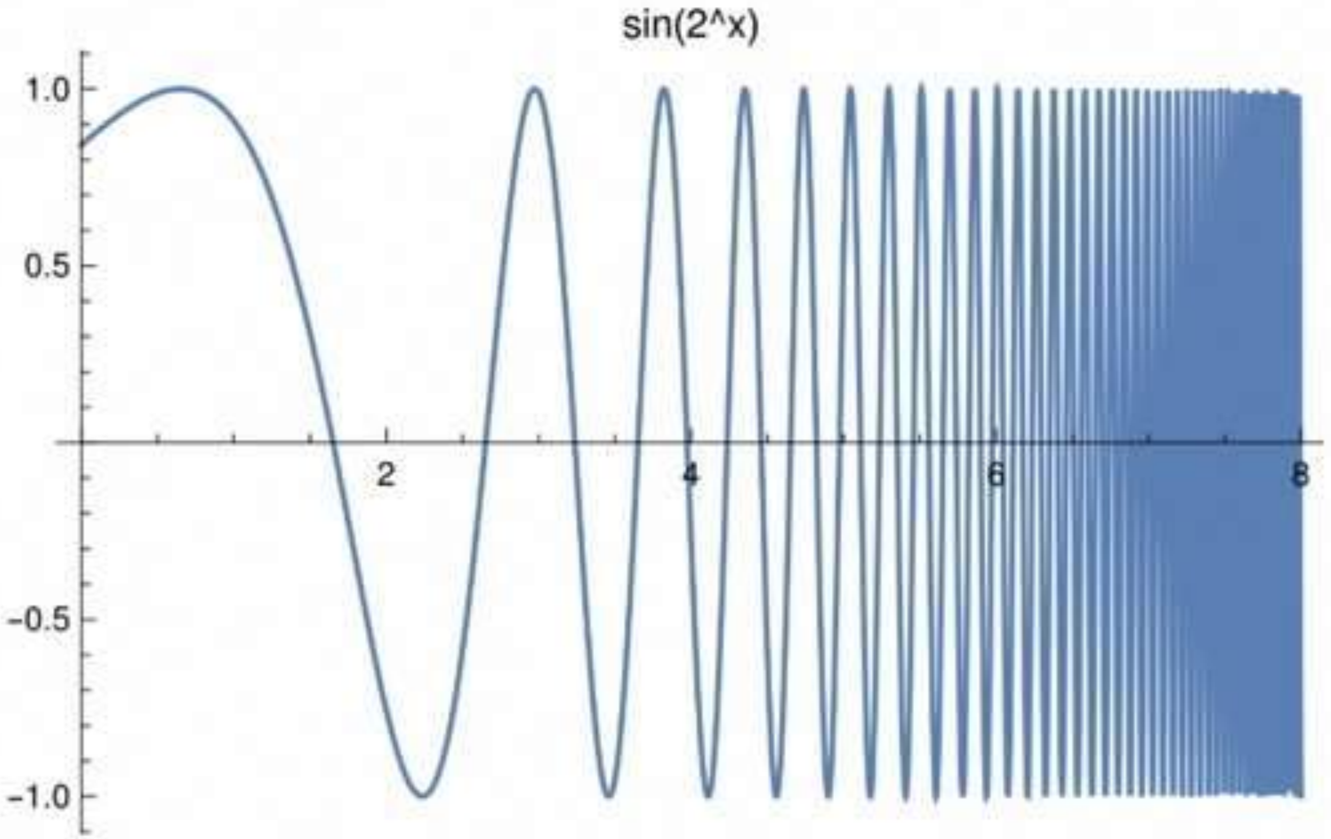
$$ax^2 + bx + c = 0$$

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

a	b	c	*x* (IEEE 754 double)	*x* (Exact)
.1	2	3	-1.6333997346592444	-1.6333997346592446
.001	2	3	-1.5011266906707066	-1.5011266906707219
1e-9	2	3	-1.5000000013088254	-1.5000000011250001
1e-15	2	3	-1.5543122344752189	-1.5000000000000011
1e-16	2	3	-2.2204460492503131	-1.5000000000000002
1e-17	2	3	0	-1.5000000000000000
.1	2	0	0	0

$$ax^2 + bx + c = 0 \qquad x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

a	b	c	*x* (IEEE 754 double)	*x* (Exact)	*x* (Sinking Point)
.1	2	3	-1.6333997346592444	-1.6333997346592446	-1.6333997346592444~
.001	2	3	-1.5011266906707066	-1.5011266906707219	-1.50112669067073~
1e-9	2	3	-1.500000013088254	-1.500000001125001	-1.5000000~
1e-15	2	3	-1.5543122344752189	-1.500000000000011	-1.5~
1e-16	2	3	-2.220446049250313	-1.5000000000000002	-2~
1e-17	2	3	0	-1.5000000000000000	0~@4
.1	2	0	0	0	0



x	IEEE 754 double	Exact	Sinking Point
15	0.9278563334139247 1	0.92785633341392471	0.92785633341392471~
25.3	-0.77866261130416781	-0.77866260949297472	-0.7786626108~
60	-0.83064921763725463	-0.83064921763725463	-0.83064921763725463~
60.01	-0.94644010175472126	0.83258728783816349	0~@0
60.125	0.27925526776125142	-0.75293133122316314	0~@0

Floating-point rounding

Floating-point rounding

exponent

$$5.25 = 21 * 2^{-2}$$

significand

Floating-point rounding

exponent

$$5.25 = 21 * 2^{-2}$$

significand

p := number of bits in significand

n := exponent - 1

Floating-point rounding

exponent

$$5.25 = 21 * 2^{-2}$$

significand

p := number of bits in significand

n := exponent - 1

Rounding can be specified in terms of $\mathbf{p}_{\mathbf{max}}$ and $\mathbf{n}_{\mathbf{min}}$

Floating-point rounding

exponent

$$5.25 = 21 * 2^{-2}$$

significand

p := number of bits in significand

n := exponent - 1

Rounding can be specified in terms of $\mathbf{p}_{\mathbf{max}}$ and $\mathbf{n}_{\mathbf{min}}$

- For IEEE 754 doubles, $\mathbf{p}_{\mathbf{max}} = 53$ and $\mathbf{n}_{\mathbf{min}} = -1075$

Floating-point rounding

exponent

$$5.25 = 21 * 2^{-2}$$

significand

$\mathbf{p}$:= number of bits in significand

$\mathbf{n}$:= exponent - 1

Rounding can be specified in terms of $\mathbf{p}_{\mathbf{max}}$ and $\mathbf{n}_{\mathbf{min}}$

- For IEEE 754 doubles, $\mathbf{p}_{\mathbf{max}} = 53$ and $\mathbf{n}_{\mathbf{min}} = -1075$
- Sinking Point determines $\mathbf{p}_{\mathbf{max}}$ and $\mathbf{n}_{\mathbf{min}}$ per operation

Sinking Point rounding rules

- Addition and Subtraction
  - $\mathbf{n}_{\mathbf{min}}$ = maximum **n** of inexact inputs

Sinking Point rounding rules

- Addition and Subtraction
  - $\mathbf{n}_{\mathbf{min}}$ = maximum **n** of inexact inputs
  - $\mathbf{p}_{\mathbf{max}} = \mathbf{p}_{\mathbf{max}}$ of format

Sinking Point rounding rules

- Addition and Subtraction
  - $\mathbf{n}_{\mathbf{min}}$ = maximum **n** of inexact inputs
  - $\mathbf{p}_{\mathbf{max}}$ = $\mathbf{p}_{\mathbf{max}}$ of format
- Multiplication and Division
  - $\mathbf{n}_{\mathbf{min}}$ = $\mathbf{n}_{\mathbf{min}}$ of format
  - $\mathbf{p}_{\mathbf{max}}$ = minimum **p** of inexact inputs

Sinking Point rounding rules

- Addition and Subtraction
  - $\mathbf{n}_{\mathbf{min}}$ = maximum **n** of inexact inputs
  - $\mathbf{p}_{\mathbf{max}}$ = $\mathbf{p}_{\mathbf{max}}$ of format
- Multiplication and Division
  - $\mathbf{n}_{\mathbf{min}}$ = $\mathbf{n}_{\mathbf{min}}$ of format
  - $\mathbf{p}_{\mathbf{max}}$ = minimum **p** of inexact inputs

Powers and roots are like multiplication.

More interesting functions

- Floor
  - Result is exact, unless n > -1

More interesting functions

- Floor
  - Result is exact, unless n > -1
- Fmod
  - $\mathbf{n}_{\mathbf{min}}$ = **n** of dividend
  - $\mathbf{p}_{\mathbf{max}}$ = **p** of divisor

More interesting functions

- Floor
  - Result is exact, unless n > -1
- Fmod
  - $\mathbf{n}_{\mathbf{min}}$ = **n** of dividend
  - $\mathbf{p}_{\mathbf{max}}$ = **p** of divisor
- Sin
  - $\mathbf{n}_{\mathbf{min}}$ = $\mathbf{n}_{\mathbf{min}}$ of format
  - $\mathbf{p}_{\mathbf{max}}$ = exponent of π - **n** of argument

Computing correctly with finitely many bits

Sinking point gives us confidence that results have meaningful precision

Fast to compute, relative to IEEE 754 floating-point

- Only a few extra bits

Still an approximation, not a sound guarantee

Titanic

Guaranteed to float correctly

FPBench

Common standards for the floating-point research community

Computing correctly with finitely many bits

Sinking point gives us confidence that results have meaningful precision

Fast to compute, relative to IEEE 754 floating-point

- Only a few extra bits

Still an approximation, not a sound guarantee

More interesting functions

- Floor
  - Result is exact, unless $n > -1$
- Fmod
  - $\mathbf{n}_{\mathbf{min}}$ = $\mathbf{n}$ of dividend
  - $\mathbf{p}_{\mathbf{max}}$ = $\mathbf{p}$ of divisor
- Sin
  - $\mathbf{n}_{\mathbf{min}}$ = $\mathbf{n}_{\mathbf{min}}$ of format
  - $\mathbf{p}_{\mathbf{max}}$ = exponent of π - $\mathbf{n}$ of argument

Computing correctly with finitely many bits

Sinking point gives us confidence that results have meaningful precision

Fast to compute, relative to IEEE 754 floating-point

- Only a few extra bits

Still an approximation, not a sound guarantee

Titanic

Guaranteed to float correctly

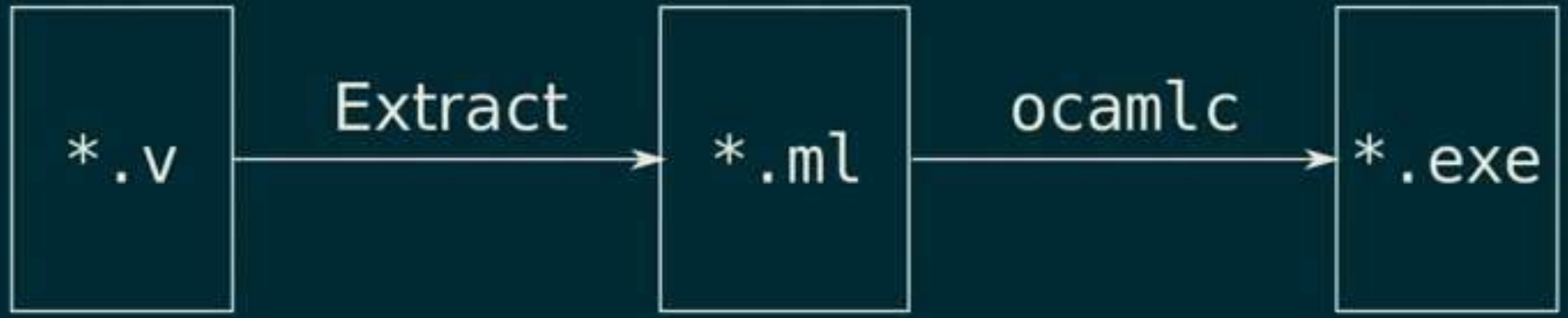
FPBench

Common standards for the floating-point research community

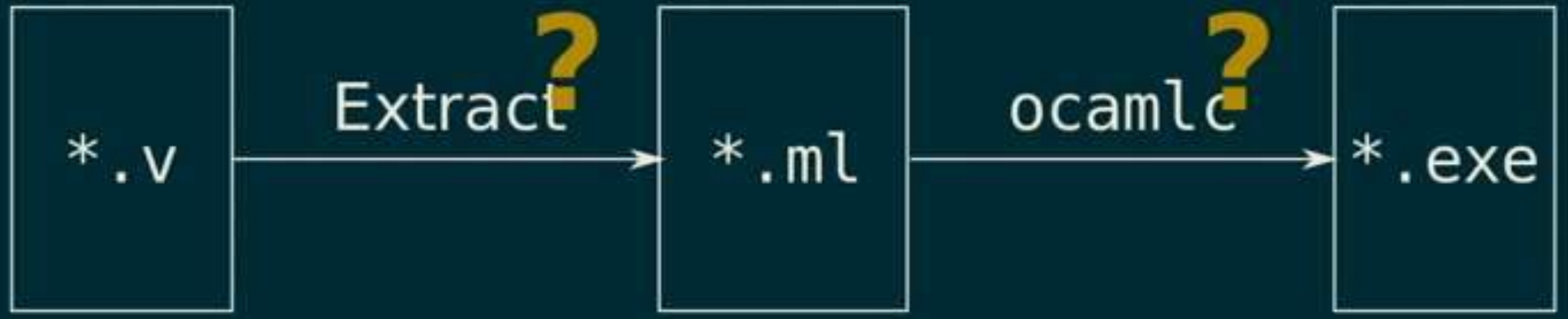
Verified Extraction with Native Types

Stuart Pernsteiner, Eric Mullen, James R. Wilcox,
Zachary Tatlock, Dan Grossman

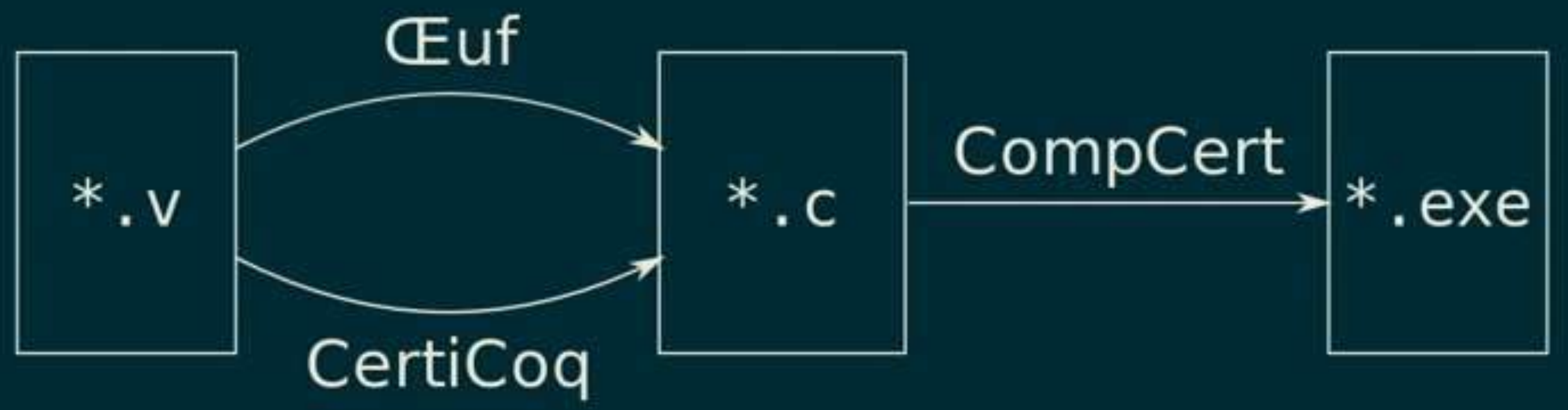
Verified Coq Extraction



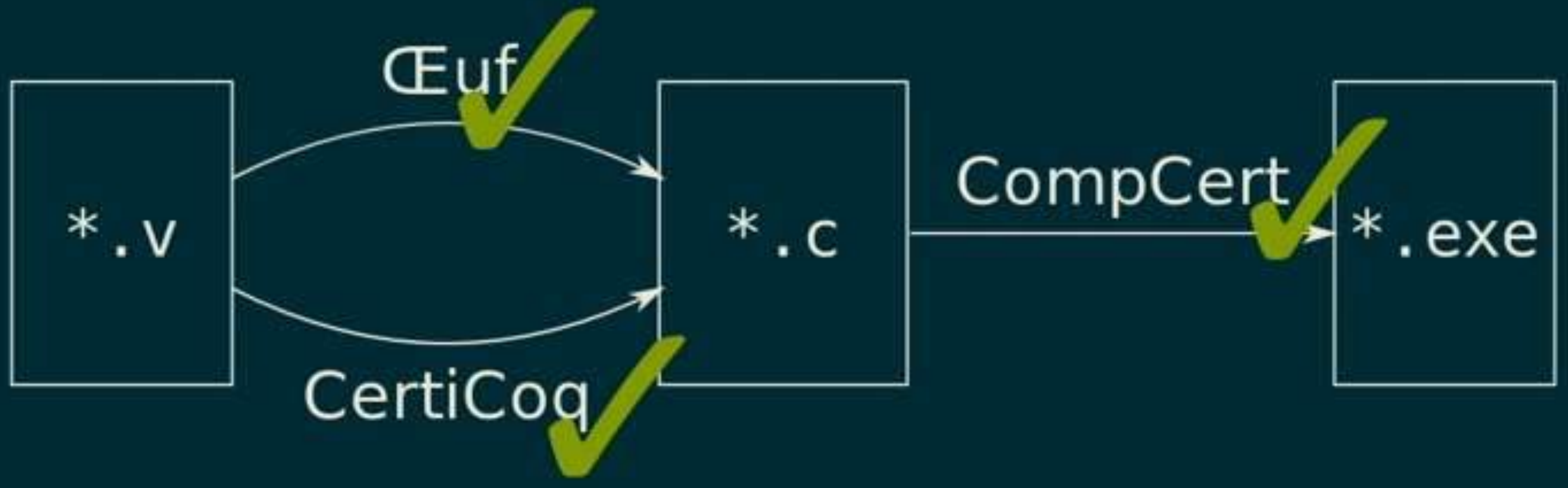
Verified Coq Extraction



Verified Coq Extraction



Verified Coq Extraction



Compiling SHA-256

Andrew W. Appel. "Verification of a Cryptographic Primitive: SHA-256."
TOPLAS 2015.

Compiling SHA-256

Andrew W. Appel. "Verification of a Cryptographic Primitive: SHA-256."
TOPLAS 2015.

Inefficient data representation

```
$ echo hello | time sha256sum
5891b5b522d5df086d0ff0b110fbd9d2...
0:00.00 elapsed
1844k max resident
```

Inefficient data representation

```
$ echo hello | time sha256sum
5891b5b522d5df086d0ff0b110fbd9d2...
0:00.00 elapsed
1844k max resident


$ echo hello | time ./sha256_N
5891b5b522d5df086d0ff0b110fbd9d2...
0:03.90 elapsed
2951208k max resident
```

Inefficient data representation

```
$ echo hello | time sha256sum
5891b5b522d5df086d0ff0b110fbd9d2...
0:00.00 elapsed
        resident
```

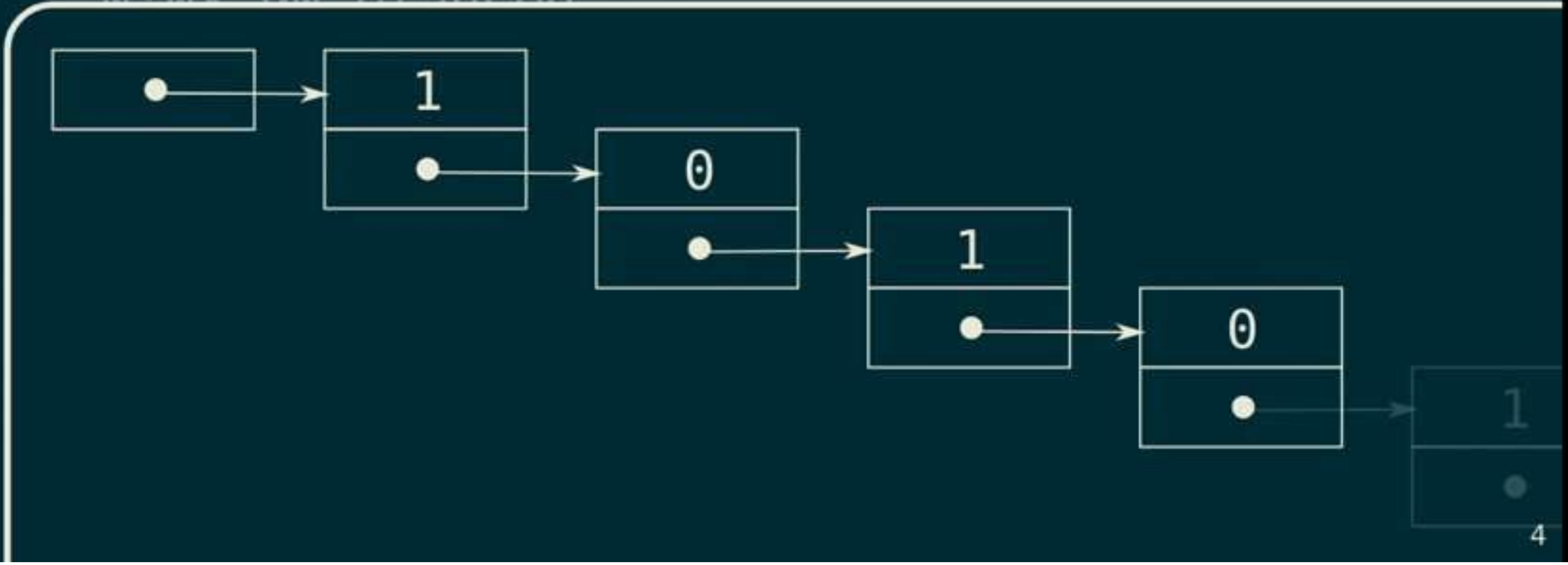
0x5891b5b5

```
$ echo hello | time ./sha256_N
5891b5b522d5df086d0ff0b110fbd9d2...
0:03.90 elapsed
2951208k max resident
```

Inefficient data representation

```
$ echo hello | time sha256sum
5891b5b522d5df086d0ff0b110fbd9d2...
0:00.00 elapsed
1844k max resident
```

```
$ echo hello | time ./sha256_N
5891b5b522d5df086d0ff0b110fbd9d2...
```



We can do better

```
Definition SHA_256 : list word → list word.
```

We can do better

```
Definition SHA_256 : list word → list word.

Definition word := { x : Z | 0 <= x < 2^32 }
```

Native Types

- Map types to custom data representations
  - `word` → `int`

Native Types

- Map types to custom data representations
  - `word` → `int`
- Map functions to custom C-level implementations
  - `(x + y) mod 2^32` → `x + y`

Native Types

- Map types to custom data representations
  - `word` → `int`
- Map functions to custom C-level implementations
  - `(x + y) mod 2^32` → `x + y`
- Maintain existing correctness guarantees

Native Types

- Map types to custom data representations
  - `word` → `int`
- Map functions to custom C-level implementations
  - `(x + y) mod 2^32` → `x + y`
- Maintain existing correctness guarantees
- Extensible to new types and functions

Native Types

- Map types to custom data representations
  - `word` → `int`
- Map functions to custom C-level implementations
  - `(x + y) mod 2^32` → `x + y`
- Maintain existing correctness guarantees
- Extensible to new types and functions
  - Updating proofs is a lot of work!

Extensible Native Types

```
Inductive type :=
| Tnat | Tbool | Tlist (t : type) | ...
| Tint?
```

Extensible Native Types

```
Inductive type :=
| Tnat | Tbool | Tlist (t : type) | ...
| Tnative (nt : native_type_defn)
```

Extensible Native Types

```
Inductive type :=
| Tnat | Tbool | Tlist (t : type) | ...
| Tnative (nt : native_type_defn)
```

```
Definition nty_int : native_type_defn := {|


|}.
```

Extensible Native Types

```
Inductive type :=
| Tnat | Tbool | Tlist (t : type) | ...
| Tnative (nt : native_type_defn)
```

```
Definition nty_int : native_type_defn := {|
    high_rep := word;


|}.
```

Extensible Native Types

```
Inductive type :=
| Tnat | Tbool | Tlist (t : type) | ...
| Tnative (nt : native_type_defn)
```

```
Definition nty_int : native_type_defn := {|
    high_rep := word;
    low_rep := int;

|}.
```

Extensible Native Types

```
Inductive type :=
| Tnat | Tbool | Tlist (t : type) | ...
| Tnative (nt : native_type_defn)
```

```
Definition nty_int : native_type_defn := {|
    high_rep := word;
    low_rep := int;
    (* lemmas... *)
|}.
```

Extensible Native Types

```
Inductive type :=
| Tnat | Tbool | Tlist (t : type) | ...
| Tnative (nt : native_type_defn)
```

```
Definition nty_int : native_type_defn := {|
    high_rep := word;
    low_rep := int;
    (* lemmas... *)
|}.

Definition nty_double : native_type_defn := ...
Definition nty_char : native_type_defn := ...
Definition nty_string : native_type_defn := ...
```

Extensible Native Operations

```
Inductive expr :=
| Evar (...) | Ecall (a b : expr)
| Eadd?
```

Extensible Native Operations

```
Inductive expr :=
| Evar (...) | Ecall (a b : expr)
| Enative (no : native_oper_defn)
          (args : list expr)
```

Extensible Native Operations

```
Inductive expr :=
| Evar (...) | Ecall (a b : expr)
| Enative (no : native_oper_defn)
          (args : list expr)
```

---

native_oper_defn must support the semantics of *every* IR!

Extensible Native Operations

```
Inductive expr :=
| Evar (...) | Ecall (a b : expr)
| Enative (no : native_oper_defn)
          (args : list expr)
```

---

native_oper_defn must support the semantics of *every* IR!

- Every IR semantics needs a way to step Enative exprs

Extensible Native Operations

```
Inductive expr :=
| Evar (...) | Ecall (a b : expr)
| Enative (no : native_oper_defn)
          (args : list expr)
```

---

native_oper_defn must support the semantics of *every* IR!

- Every IR semantics needs a way to step Enative exprs
- We chose to have one implementation for each value representation

Design Tradeoffs

- Simplifies native oper. definitions
    - Less stuff in the record

Design Tradeoffs

- Simplifies native oper. definitions
  - Less stuff in the record
- Simplifies compiler correctness proofs
  - Native oper cases are trivial when value representations don't change

Design Tradeoffs

- Simplifies native oper. definitions
  - Less stuff in the record
- Simplifies compiler correctness proofs
  - Native oper cases are trivial when value representations don't change
- Native operations can't call functions
  - Calls work differently in different IRs
  - Can't implement recursive eliminators as native opers

Results

- It works and is extensible
  - int w/ literals, arith ops, comparisons, int_to_nat
  - double w/ literals, arith ops, comparisons, to/from_int
  - double_array w/ alloc, get, set
  - Avg. 65 lines per type; 210 lines per operation

Results

- It works and is extensible
  - `int` w/ literals, arith ops, comparisons, `int_to_nat`
  - `double` w/ literals, arith ops, comparisons, `to/from_int`
  - `double_array` w/ `alloc`, `get`, `set`
  - Avg. 65 lines per type; 210 lines per operation
- Performance is much improved...

Inefficient data representation

```
$ echo hello | time sha256sum
5891b5b522d5df086d0ff0b110fbd9d2...
0:00.00 elapsed
1844k max resident


$ echo hello | time ./sha256_N
5891b5b522d5df086d0ff0b110fbd9d2...
0:03.90 elapsed
2951208k max resident
```

Efficient data representation

```
$ echo hello | time sha256sum
5891b5b522d5df086d0ff0b110fbd9d2...
0:00.00 elapsed
1844k max resident
```

```
$ echo hello | time ./sha256_N
5891b5b522d5df086d0ff0b110fbd9d2...
0:03.90 elapsed
2951208k max resident
```

Efficient data representation

```
$ echo hello | time sha256sum
5891b5b522d5df086d0ff0b110fbd9d2...
0:00.00 elapsed
1844k max resident
```

```
$ echo hello | time ./sha256_N
5891b5b522d5df086d0ff0b110fbd9d2...
0:03.90 elapsed
2951208k max resident
```

50x less time!

```
$ echo hello | time ./sha256_int
5891b5b522d5df086d0ff0b110fbd9d2...
0:00.07 elapsed
33760k max resident
```

Efficient data representation

```
$ echo hello | time sha256sum
5891b5b522d5df086d0ff0b110fbd9d2...
0:00.00 elapsed
1844k max resident

$ echo hello | time ./sha256_N
5891b5b522d5df086d0ff0b110fbd9d2...
0:03.90 elapsed
2951208k max resident

$                e ./sha256_int
5                0ff0b110fbd9d2...
0:00.07 elapsed
33760k max resident
```

50x less time!

87x less memory!