



**LiquidHaskell**

Usable Language-Based Verification

Niki Vazou

University of Maryland

# Formal Verification

## Prove Properties of Software

# Formal Verification

Prove Properties of Real Software

# Formal Verification

Prove Properties of Real Software



“A plane will not crush”

# Formal Verification

Prove Properties of Real Software



“Medical Equipments will properly work”

# Formal Verification

Prove Properties of Real Software



“Your passwords are SAFE”

# Formal Verification, in Practise

Prove Properties of ~~Real Software~~

*in Verification Specific Tools*

# Formal Verification, in Practise

Prove Properties of ~~Real Software~~

*in Verification Specific Tools*

# Verification Specific Tools

```
Lemma plus_Snm_nSm : forall n m, S n + m = n + S m.  
intros.  
simpl in |- *.  
rewrite (plus_comm n m).  
rewrite (plus_comm n (S m)).  
trivial with arith.  
Qed.
```



“Plus is indeed associative!”

# Verification Specific Tools

Not used for Real Programs

Difficult to install

Special Syntax

No Compiler Optimizations

No Parallelism

**Very Few Users!**

# Verification Specific Tools



Manual  
Translation



Verification

My goal:

Verification of Real Programs

# Verification of Real Programs

My approach:

Turn a Language into a Verifier

Reuse language's syntax, runtime, and users!

# Verification of Real Programs

My approach:

Turn a Language into a Verifier

Target Language: Haskell

# Turn a Haskell into a Verifier

Reuse language's syntax, runtime, and **users**!

**I:** Target Real Haskell Code

**II:** Application in **Industry**

Use Verification to speedup code

**III:** Application in **Education**

Teach verification in Haskell courses!

Haskell

+

Refinement Types

=

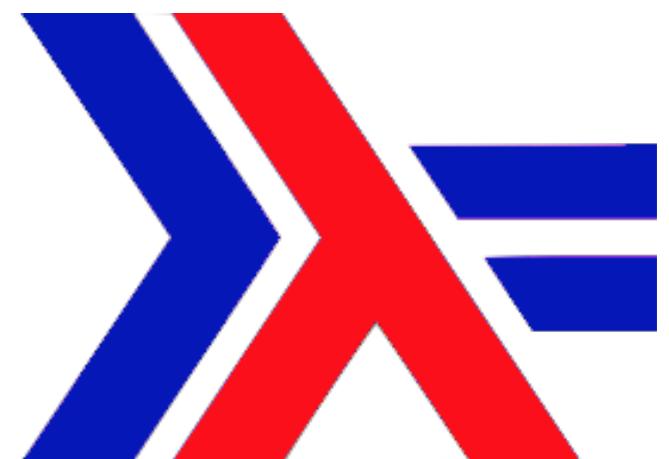


LiquidHaskell

# Haskell

Functional Programming Language

Based on  
Strong Typing +  $\lambda$ -calculus

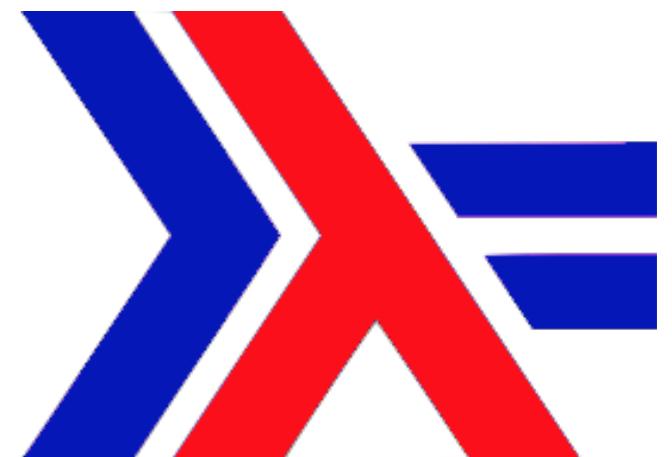


*Well Typed Programs  
cannot go wrong!*

Functional Programming Language



Based on  
Typing +  $\lambda$ -calculus



# **Fact Check.**

## **Can we encode Heartbleed?**



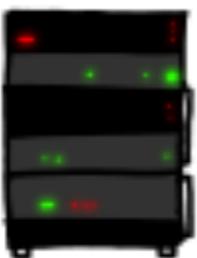
The Heartbleed Bug.  
Buffer overflow in OpenSSL. 2015

# HOW THE HEARTBLEED BUG WORKS:

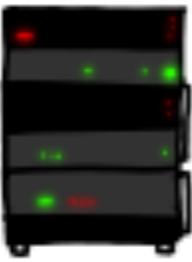
SERVER, ARE YOU STILL THERE?  
IF SO, REPLY "POTATO" (6 LETTERS).



User Eric wants pages about "boats". User Erica requests secure connection using key "4538538374224". User Meg wants these 6 letters: POTATO. User Ada wants pages about "irl games". Unlocking secure records with master key 5130985733435. Macie (chrome user) sends this message: "H



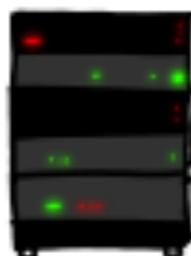
POTATO



SERVER, ARE YOU STILL THERE?  
IF SO, REPLY "BIRD" (4 LETTERS).



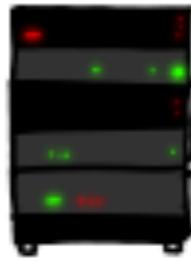
User Olivia from London wants pages about "new bees in car why". Note: Files for IP 375.381. 283.17 are in /tmp/files-3843. User Meg wants these 4 letters: BIRD. There are currently 348 connections open. User Brendan uploaded the file selfie.jpg (contents: 834ba962e2ceb9ff89bd3bfff84)



HMM...



BIRD



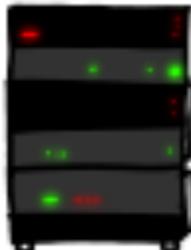
SERVER, ARE YOU STILL THERE?

a connection. Jake requested pictures of deer. User Meg wants these 500 letters: HAT. Lucas

SERVER, ARE YOU STILL THERE?  
IF SO, REPLY "HAT" (500 LETTERS).

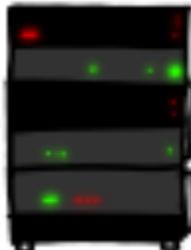


a connection. Jake requested pictures of deer.  
User Meg wants these 500 letters: HAT. Lucas  
requests the "missed connections" page. Eve  
(administrator) wants to set server's master  
key to "14835038534". Isabel wants pages about  
"snakes but not too long". User Karen wants to  
change account password to "CoHoBaSt". User



HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "CoHoBaSt". User

a connection. Jake requested pictures of deer.  
User Meg wants these 500 letters: HAT. Lucas  
requests the "missed connections" page. Eve  
(administrator) wants to set server's master  
key to "14835038534". Isabel wants pages about  
"snakes but not too long". User Karen wants to  
change account password to "CoHoBaSt". User





VS.





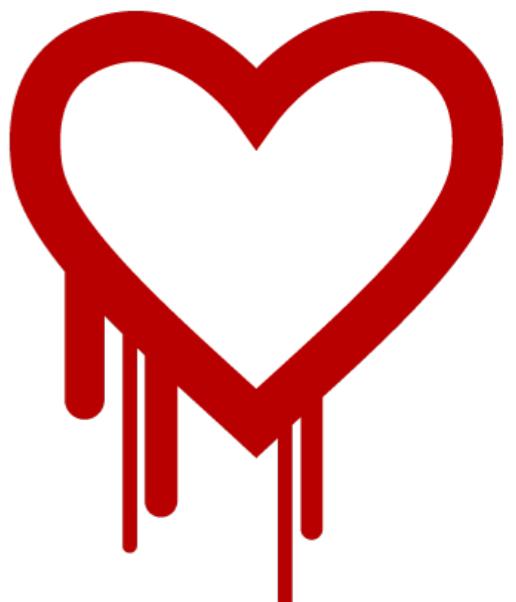
VS.



```
λ> :m +Data.Text Data.Text.Unsafe  
λ> let pack = "hat"  
  
λ> :t takeWord16  
takeWord16 :: Text -> Int -> Text
```



VS.



```
λ> :m +Data.Text Data.Text.Unsafe  
λ> let pack = "hat"  
  
λ> takeWord16 pack True  
Type Error: Cannot match Bool vs Int
```



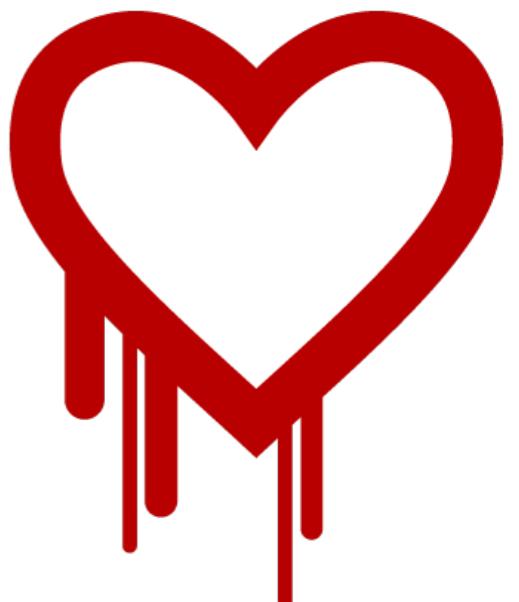
VS.



```
λ> :m +Data.Text Data.Text.Unsafe  
λ> let pack = "hat"  
  
λ> takeWord16 pack 500  
“hat\58456\2594\SOH\NUL...
```



VS.



# Valid Values for takeWord16?

`takeWord16 :: t:Text -> i:Int -> Text`

**All Ints**

`..., -2, -1, 0, 1, 2, 3, ...`

# Valid Values for takeWord16?

`takeWord16 :: t:Text -> i:Int -> Text`

**Valid Ints**

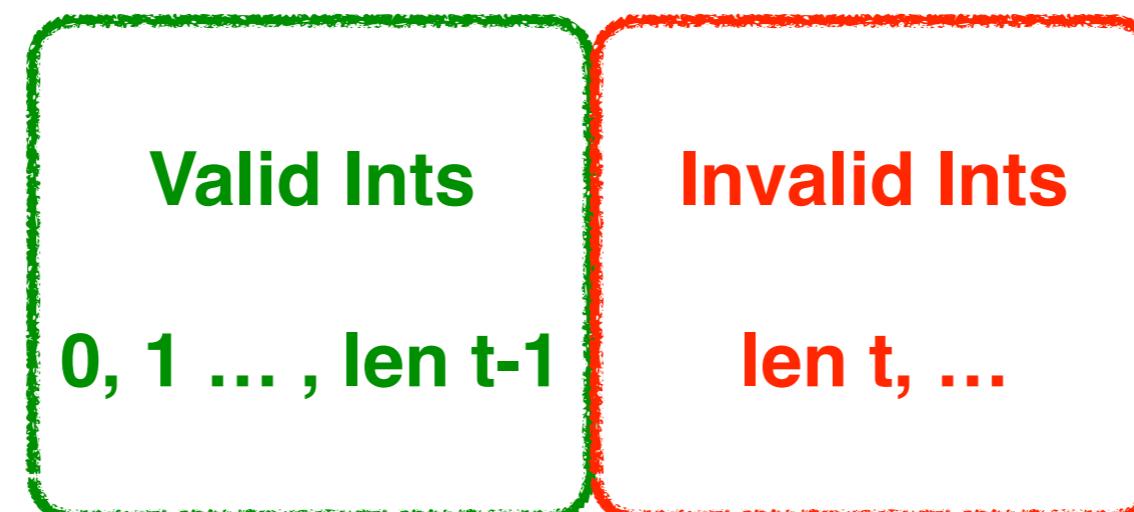
`0, 1 ... , len t-1`

**Invalid Ints**

`len t, ...`

# Refinement Types

`take :: t:Text -> {v:Int | v < len t} -> Text`



# Refinement Types

```
take :: t:Text -> {v:Int | v < len t} -> Text
```

```
λ> :m +Data.Text Data.Text.Unsafe
```

```
λ> let pack = "hat"
```

```
λ> take pack 500
```

```
Refinement Type Error
```

i.e.  **LiquidHaskell**!

# Refinement Types



**Checks valid arguments, under facts.**

# Checks valid arguments, under facts.

```
take :: t:Text -> {v | v < len t} -> Text
heartbleed = let x = "hat"
             in take x 500
```

len x = 3 => v = 500 => v < len x

# Checks valid arguments, under facts.

```
take :: t:Text -> {v | v < len t} -> Text
heartbleed = let x = "hat"
            in take x 500
```

len x = 3 => v = 500 => v < len x

# **Checks valid arguments, under facts.**

```
take :: t:Text -> {v | v < len t} -> Text
```

```
heartbleed = let x = "hat"  
           in take x 500
```

```
len x = 3 => v = 500 => v < len x
```

# Checks valid arguments, under facts.

```
take :: t:Text -> {v | v < len t} -> Text
```

```
heartbleed = let x = "hat"  
           in take x 500
```

len x = 3 =>  $v = 500$  =>  $v < \text{len } x$

# **Checks valid arguments, under facts.**

```
take :: t:Text -> {v | v < len t} -> Text  
heartbleed = let x = "hat"  
            in take x 500
```

len x = 3 => v = 500 => v < len x

# Checks valid arguments, under facts.

```
take :: t:Text -> {v | v < len t} -> Text
heartbleed = let x = "hat"
             in take x 500
```

```
len x = 3 => v = 500 => v < len x
```

# Checks valid arguments, under facts.

```
take :: t:Text -> {v | v < len t} -> Text  
heartbleed = let x = "hat"  
            in take x 500
```

SMT-  
query

len x = 3 => v = 500 => v < len x

# Checks valid arguments, under facts.

```
take :: t:Text -> {v | v < len t} -> Text  
heartbleed = let x = "hat"  
           in take x 500
```

SMT-  
invalid

len x = 3 => v = 500 => v < len x

# Checks valid arguments, under facts.

```
take :: t:Text -> {v | v < len t} -> Text
heartbleed = let x = "hat"
             in take x 500
```

Checker reports **Error**

len x = 3 => v = 500 => v < len x

# Checks valid arguments, under facts.

```
take :: t:Text -> {v | v < len t} -> Text
heartbleed = let x = "hat"
            in take x 500
```

Checker reports **Error**

len x = 3 => v = 500 => v < len x

# Checks valid arguments, under facts.

```
take :: t:Text -> {v | v < len t} -> Text
```

```
heartbleed = let x = "hat"  
           in take x 2
```

Checker reports **OK**

SMT-  
valid

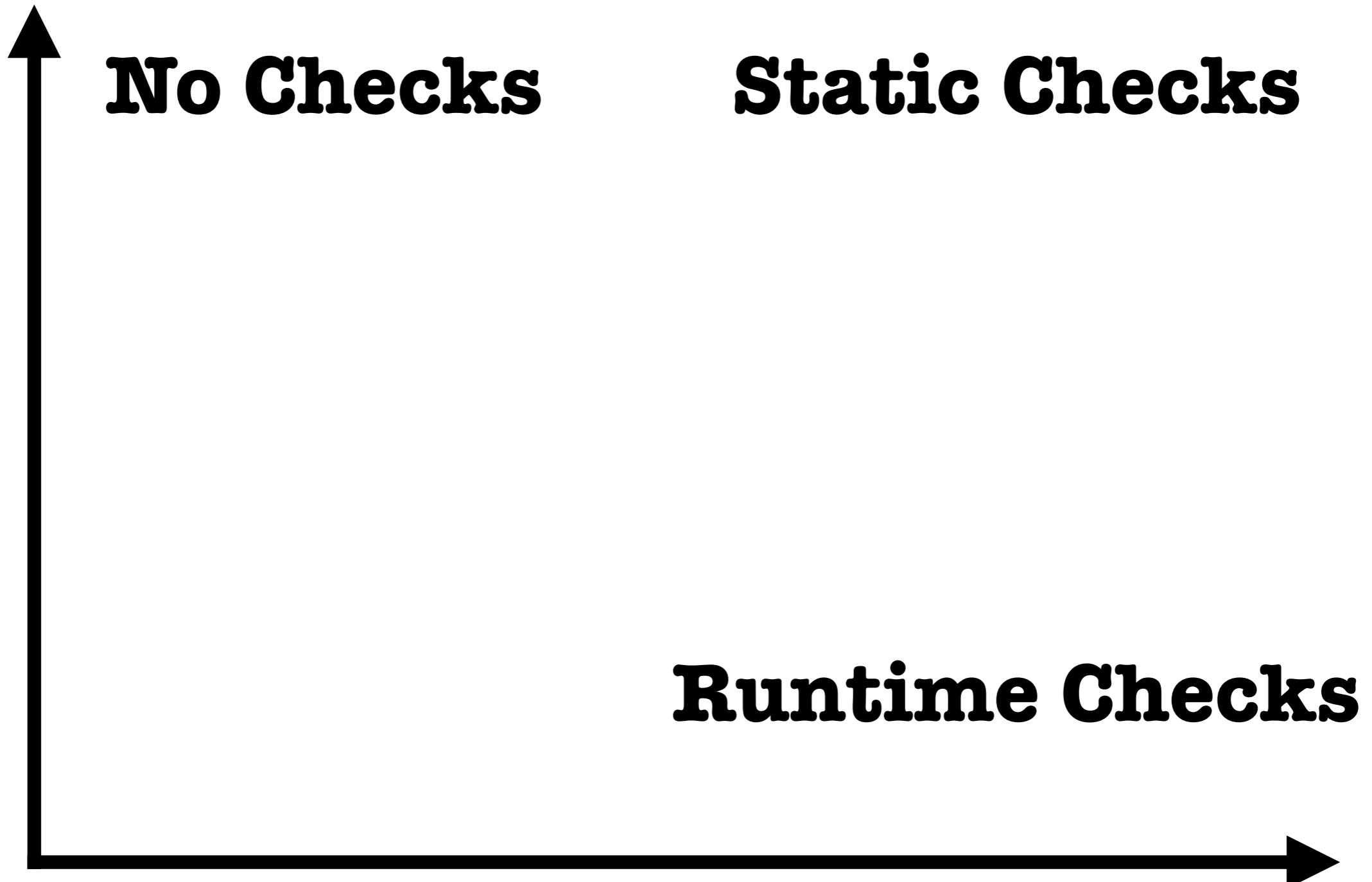
len x = 3 => v = 2 => v < len x



**Checks valid arguments, under facts.**

**Static Checks**

**Efficiency**



**Safety**

# No Checks

```
take :: t:Text -> i:Int -> Text
take i t
= Unsafe.takeWord16 i t
```

# No Checks

```
take :: t:Text -> i:Int -> Text
take i t
= Unsafe.takeWord16 i t
```

```
heartbleed = take "hat" 500
```

OK

# No Checks

```
take :: t:Text -> i:Int -> Text
take i t
= Unsafe.takeWord16 i t
```

OK

```
heartbleed = take "hat" 500
```

UNSAFE

```
λ> heartbleed
λ> “hat\58456\2594\SOH\NUL...
```

# Runtime Checks

```
take :: t:Text -> i:Int -> Text
take i t | i < len t
          = Unsafe.takeWord16 i t
take i t
          = error "Out Of Bounds!"
```

# Runtime Checks

```
take :: t:Text -> i:Int -> Text
take i t | i < len t
          = Unsafe.takeWord16 i t
take i t
          = error "Out Of Bounds!"
```

# Runtime Checks

```
take :: t:Text -> i:Int -> Text
take i t | i < len t
  = Unsafe.takeWord16 i t
take i t
  = error "Out Of Bounds!"
```

```
heartbleed = take "hat" 500
```

OK

# Runtime Checks

```
take :: t:Text -> i:Int -> Text
take i t | i < len t
  = Unsafe.takeWord16 i t
take i t
  = error "Out Of Bounds!"
```

OK

```
heartbleed = take "hat" 500
```

SAFE

```
λ> heartbleed
```

```
λ> *** Exception: Out Of Bounds!
```

# Runtime Checks are expensive

```
take :: t:Text -> i:Int -> Text
take i t | i < len t
          = Unsafe.takeWord16 i t
take i t
          = error "Out Of Bounds!"
```

# Static Checks

```
take :: t:Text -> i:{i < len t} -> Text
take i t | i < len t
  = Unsafe.takeWord16 i t
take i t
  = error "Out Of Bounds!"
```

# Static Checks

```
take :: t:Text -> i:{i < len t} -> Text
take i t | i < len t
  = Unsafe.takeWord16 i t
take i t
error "Out Of Bounds!"
```

# Static Checks

```
take :: t:Text -> i:{i < len t} -> Text
take i t
= Unsafe.takeWord16 i t
```

# Static Checks

```
take :: t:Text -> i:{i < len t} -> Text
take i t
= Unsafe.takeWord16 i t
```

**UNSAFE**

```
heartbleed = take "hat" 500
```



**LiquidHaskell**

**Static Checks**

**Safe & Fast Code!**

**Static Checks**

**Safe & Fast Code!**

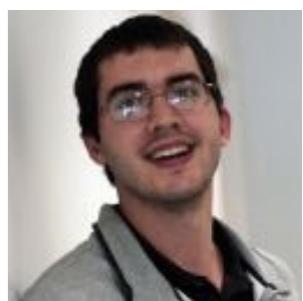
**Application: Speedup Parsing**

# **Application: Speedup Parsing**

## **UDP:User Datagram Protocol**



Gabriel Gonzalez  
 AWAKE

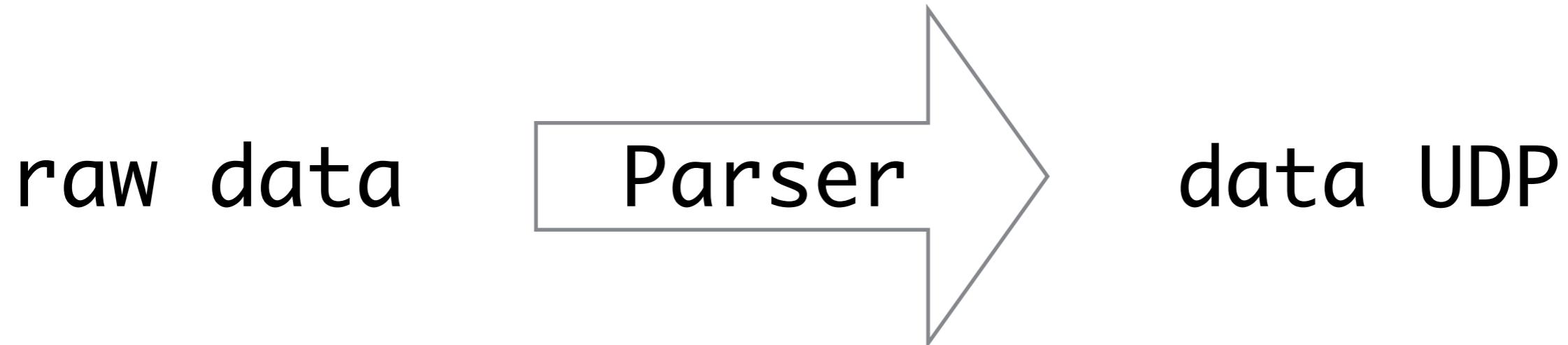


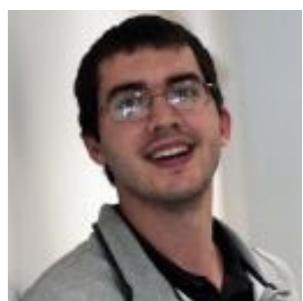
Gabriel Gonzalez



# Application: Speedup Parsing

## UDP:User Datagram Protocol





Gabriel Gonzalez



# Application: Speedup Parsing

```
data UDP = UDP
  { udpSrcPort :: Text -- 2 chars
  , udpDestPort :: Text -- 2 chars
  , udpLength :: Text -- 2 chars
  , udpChecksum :: Text -- 2 chars
  }
```



Gabriel Gonzalez



# Application: Speedup Parsing

```
udpP :: Text -> UDP
udpP bs =
  let (udp1, bs1) = splitAt 2 bs
  let (udp2, bs2) = splitAt 2 bs1
  let (udp3, bs3) = splitAt 2 bs2
  let (udp4, bs4) = splitAt 2 bs3
in UDP (udp1 upd2 udp3 upd4)
```

Safe but Slow (4 runtime checks)

Solution: Merge checks



Gabriel Gonzalez



# Application: Speedup Parsing

```
udpP :: Text -> Maybe UDP
udpP bs =
  if length bs < 8 then
    let (udp1, bs1) = US.splitAt 2 bs
    let (udp2, bs2) = US.splitAt 2 bs1
    let (udp3, bs3) = US.splitAt 2 bs2
    let (udp4, bs4) = US.splitAt 2 bs3
    in Just (UDP udp1 upd2 upd3 upd4)
  else Nothing
```

Safe and Fast (1 runtime check!)



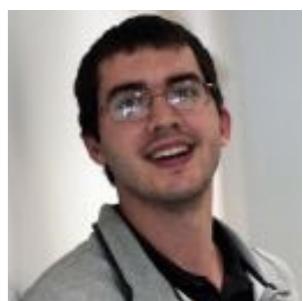
Gabriel Gonzalez



# Application: Speedup Parsing

```
udpP :: Text -> Maybe UDP
udpP bs =
  if length bs < 8 then
    let (udp1, bs1) = US.splitAt 2 bs
    let (udp2, bs2) = US.splitAt 2 bs1
    let (udp3, bs3) = US.splitAt 4 bs2
    let (udp4, bs4) = US.splitAt 2 bs3
    in Just (UDP udp1 upd2 upd3 upd4)
  else Nothing
```

Safe and Fast, but error prone



Gabriel Gonzalez



# Application: Speedup Parsing

```
udpP :: Text -> Maybe UDP
udpP bs =
  if length bs < 8 then
    let (udp1, bs1) = US.splitAt 2 bs
      in Just (UDP udp1 upd2 upd3 upd4)
  else Nothing

splitAt :: i:Int -> t:{i < len t} ->
(tl:{i = len tl}, tr:{len tr = len t - i})
```

Enforce Static Checks!



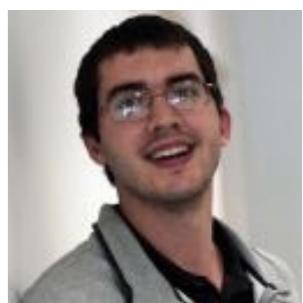
Gabriel Gonzalez

# Application: Speedup Parsing

UNSAFE

```
udpP :: Text -> Maybe UDP
udpP bs =
  if length bs < 8 then
    let (udp1, bs1) = US.splitAt 2 bs0
    let (udp2, bs2) = US.splitAt 2 bs1
    let (udp3, bs3) = US.splitAt 4 bs2
    let (udp4, bs4) = US.splitAt 2 bs3
    in Just (UDP udp1 upd2 upd3 upd4)
  else Nothing
```

Enforce Static Checks!



# Application: Speedup Parsing

Sébastien Gonzalez  
AWAKE

SAFE

```
udpP :: Text -> Maybe UDP
udpP bs =
  if length bs < 8 then
    let (udp1, bs1) = US.splitAt 2 bs
    let (udp2, bs2) = US.splitAt 2 bs1
    let (udp3, bs3) = US.splitAt 2 bs2
    let (udp4, bs4) = US.splitAt 2 bs3
    in Just (UDP udp1 upd2 upd3 upd4)
  else Nothing
```

Provably Correct & Faster (x6) Code!



Gabriel Gonzalez



# **Application: Speedup Parsing**

Provably Correct & Faster (x6) Code!

**Safe & Efficient Code!**



**Safe & Efficient Code!**

**Efficiency**



**No Checks**

**Static Checks**

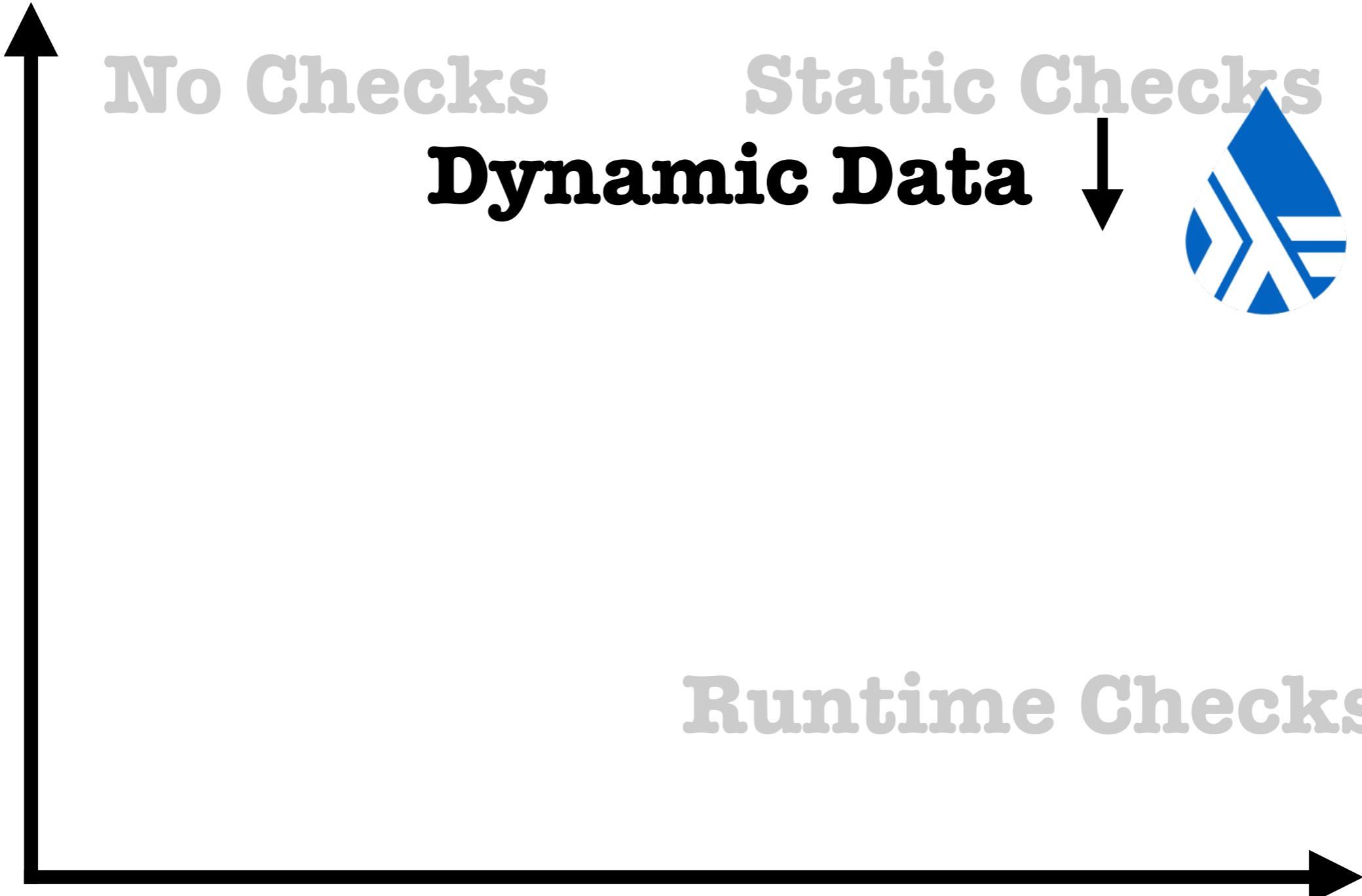


**Runtime Checks**



**Safety**

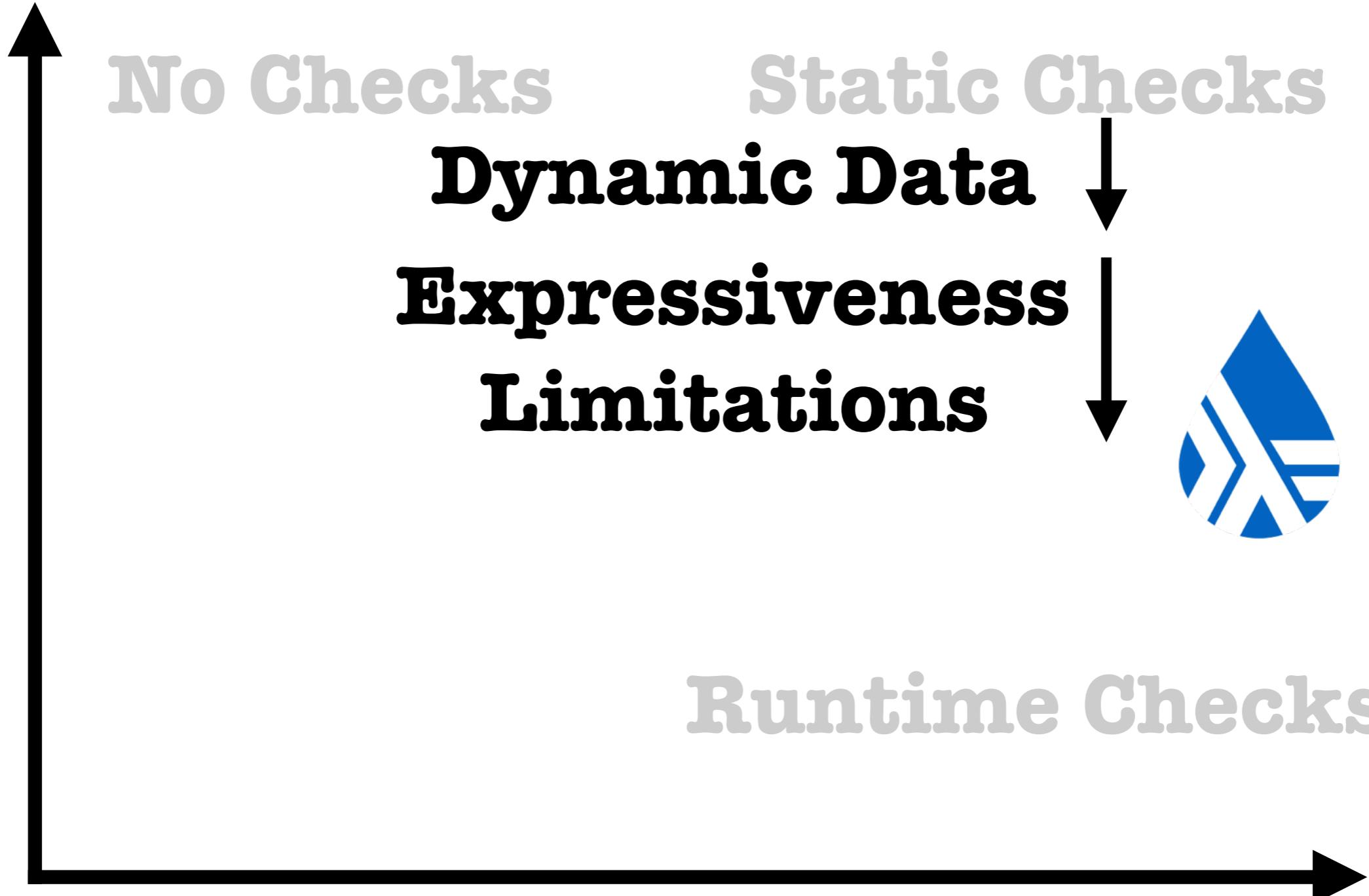
**Efficiency**



**Safety**



**Efficiency**



# **Expressiveness**

What properties can be expressed in types?

# **Expressiveness vs. Automation**

# Expressiveness vs. Automation

If  $p$  is safe indexing ...

```
{t:Text | i < len t }
```

... then SMT-automatic verification.

# **Expressiveness vs. Automation**

**If  $p$  from decidable theories ...**

$$\{t : a \mid p\}$$

**... then SMT-automatic verification.**

# Expressiveness vs. Automation

If  $p$  from decidable theories ...

$$\{t : a \mid p\}$$

Boolean Logic

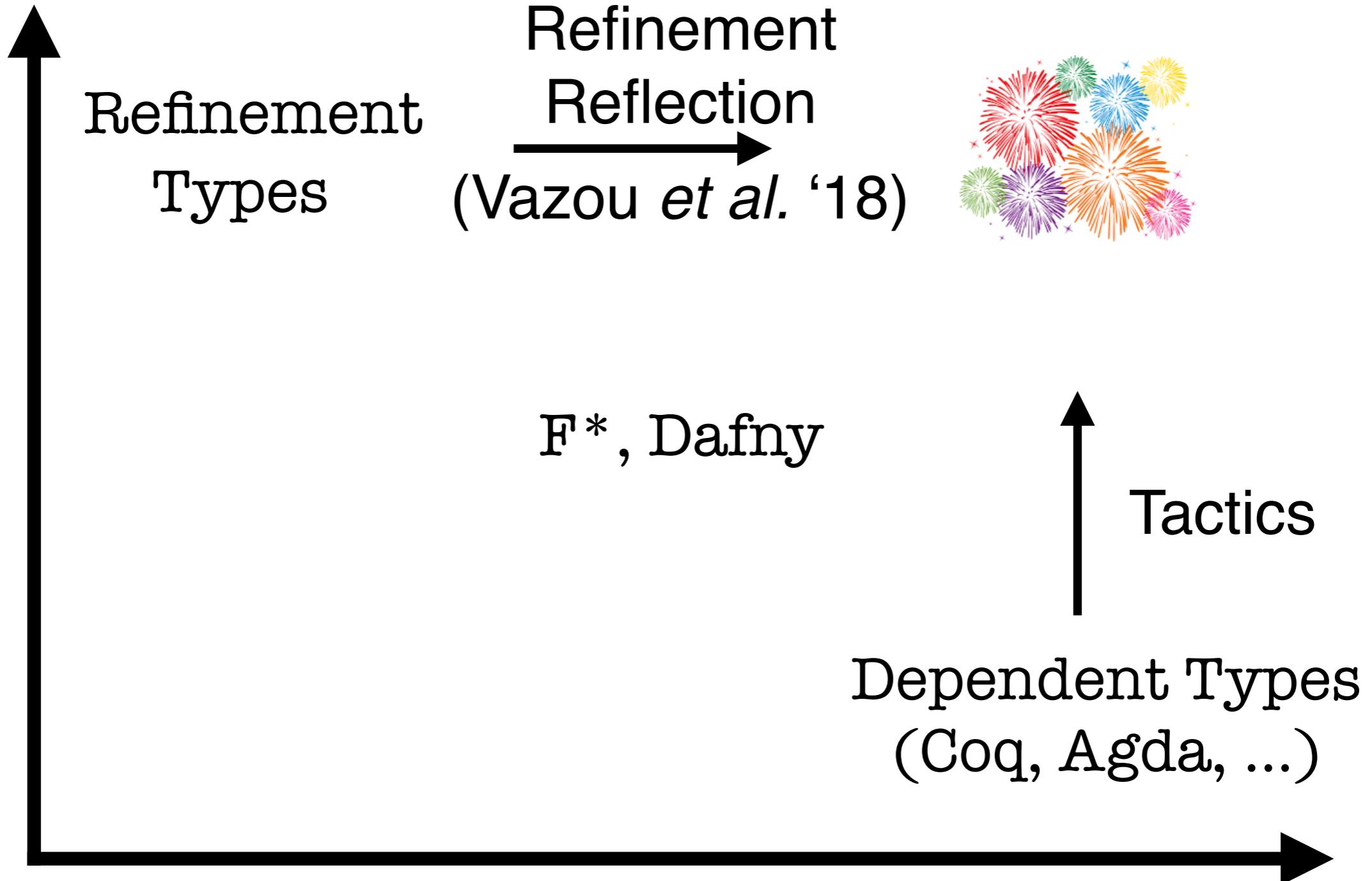
(QF) Linear Arithmetic

Uninterpreted Functions ...

... then SMT-automatic verification.

What about expressiveness?

**Automation**



**Expressiveness**

# Classic Refinement Types

```
fib :: i:{Int | 0≤i} -> {v:Int | 0<v∧i≤v}
fib i
| i≤1          = 1
| otherwise    = fib (i-1) + fib (i-2)
```

SAFE

Can we increase expressiveness?

# Can we increase expressiveness?

```
fib :: i:{Int | 0≤i} -> {v:Int | 0<v ∧ i≤v}
fib i
| i≤1          = 1
| otherwise     = fib (i-1) + fib (i-2)
```

## How to express **theorems** about functions?

```
\forall i. 0 ≤ i => fib i ≤ fib (i+1)
```

# How to express **theorems** about functions?

## **Step 1:** Definition

In SMT **fib** is “Uninterpreted Function”

\forall i j. i = j => fib i = fib j

How to connect logic **fib** with target **fib**?

# How to connect logic fib with target fib?

~~fib :: i:{Int | 0≤i} > {v: Int | 0< v ∧ i ≤ v}~~

~~fib i~~

~~| 1≤1~~

~~| otherwise = fib (i-1) + fib (i-2)~~

**NOT decidable**

~~SMT AXIOM~~

~~\forall i.~~

~~if i ≤ 1 then fib i = 1~~

~~else fib i = fib (i-1) + fib (i-2)~~

**Decidable**

# How to connect logic fib with target fib?

```
fib :: i:{Int | 0≤i} -> {v:Int | 0<v∧i≤v}
fib i
| i≤1          = 1
| otherwise    = fib (i-1) + fib (i-2)
```

## Refinement Reflection

```
fib :: i:{Int | 0≤i} -> {v:Int | v=fib i ∧
  if i≤1 then fib i = 1
  else fib i = fib (i-1) + fib (i-2)
}
```

# Refinement Reflection

**Step 1:** Definition

**Step 2:** Reflection

```
fib :: i:{Int | 0≤i} -> {v:Int | v=fib i ∧
  if i≤1 then fib i = 1
  else fib i = fib (i-1) + fib (i-2)
}
```

# Refinement Reflection

## Step 1: Definition

## Step 2: Reflection

```
fib :: i:{Int | 0≤i} -> {v:Int | v=fib i ∧  
    if i≤1 then fib i = 1  
    else fib i = fib (i-1) + fib (i-2)  
}
```

## Step 3: Application

```
fib 0 :: {v:Int | v=fib 0 ∧ fib 0 = 1}
```

# Application is Type Level Computation

fib 0

fib 0 = 1

# Application      Type Level Computation

fib 0

fib 0 = 1

fib 1

fib 1 = 1

fib 2

fib 2 = fib 1 + fib 0

fib i

?

- ? if  $i \leq 1$  then fib i = 1
- ? else fib i = fib (i-1) + fib (i-2)

# Application      Type Level Computation

fib 0

fib 0 = 1

fib 1

fib 1 = 1

fib 2

fib 2 = fib 1 + fib 0

if 1 < i then

fib i

fib i = fib (i-1) + fib (i-2)

if  $i \leq 1$  then fib i = 1

else fib i = fib (i-1) + fib (i-2)

# Application      Type Level Computation

**fib 0**

**fib 0 = 1**

**fib 1**

**fib 1 = 1**

**fib 2**

**fib 2 = fib 1 + fib 0**

**if 1 < i then**

**fib i**

**fib i = fib (i-1) + fib (i-2)**

**fib (i+1)**

**fib (i+1) = fib i + fib (i-1)**

# Theorem Proving

```
fibUp :: i:Nat -> {fib i ≤ fib (i+1)}
fibUp i
| i == 0
= fib 0 <. fib 1
*** QED
| i == 1
= fib 1 <=. fib 1 + fib 0 <= fib 2
*** QED
| otherwise
= fib i
<=. fib i + fib (i-1)
<=. fib (i+1)
*** QED
```

# Reflection for Theorem Proving

**Theorems** are refinement types.

**Proofs** are functions.

**Check** that functions prove theorems.

**Proofs** are functions.

`fibUp :: i:Nat -> {fib i ≤ fib (i+1)}`

# Proofs are functions.

```
fibUp :: i:Nat -> {fib i ≤ fib (i+1)}
```

Let's call them!

```
fibUp 4 :: {fib 4 ≤ fib 5}
```

```
fibUp i :: {fib i ≤ fib (i+1)}
```

```
fibUp (j-1) :: {fib (j-1) ≤ fib j}
```

# Proofs are functions. Let's call them!

```
fibMono :: i:Nat -> j:{Nat | i < j}  
         -> {fib i ≤ fib j}
```

```
fibMono i j  
| i + 1 == j  
= fib i  
<=. fib (i+1) ? fibUp i  
==. fib j  
*** QED  
| otherwise  
= fib i  
<=. fib (j-1) ? fibMono i (j-1)  
<=. fib j      ? fibUp (j-1)  
*** QED
```

# Proofs are functions. Let's call them!

```
fibMono :: i:Nat -> j:{Nat | i < j}  
         -> {fib i ≤ fib j}
```

```
fibMono i j  
| i + 1 == j  
= fib i  
<=. fib (i+1) ? fibUp i  
==. fib j  
*** QED  
| otherwise  
= fib i  
<=. fib (j-1) ? fibMono i (j-1)  
<=. fib j      ? fibUp (j-1)  
*** QED
```

# Proofs are functions. Let's abstract them!

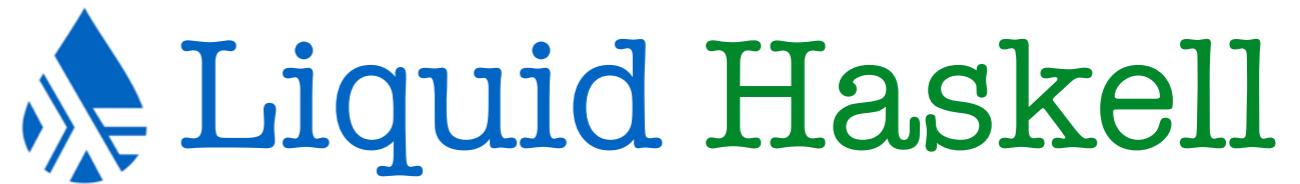
```
fibMono :: i:Nat -> j:{Nat | i < j}
          -> fib:(Nat -> Int)
          -> (k:Nat -> {fib k ≤ fib (k+1)})
          -> {fib i ≤ fib j}
```

```
fibMono i j fib fibUp
| i + 1 == j
= fib i
<=. fib (i+1) ? fibUp i
==. fib j
*** QED
| otherwise
= fib i
<=. fib (j-1) ? fibMono i (j-1)
<=. fib j      ? fibUp (j-1)
*** QED
```



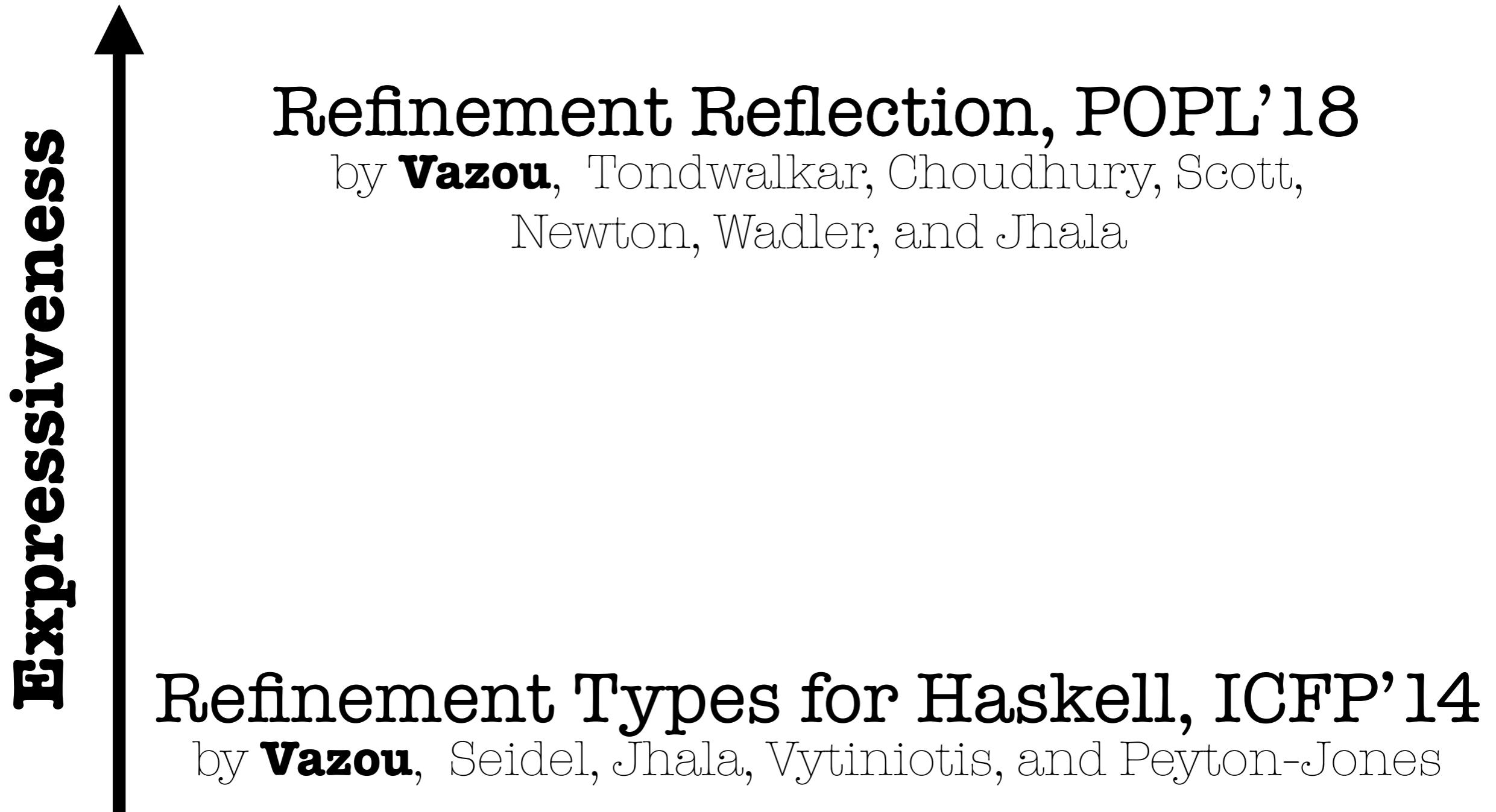
LiquidHaskell

# Refinement Reflection for **Expressiveness**



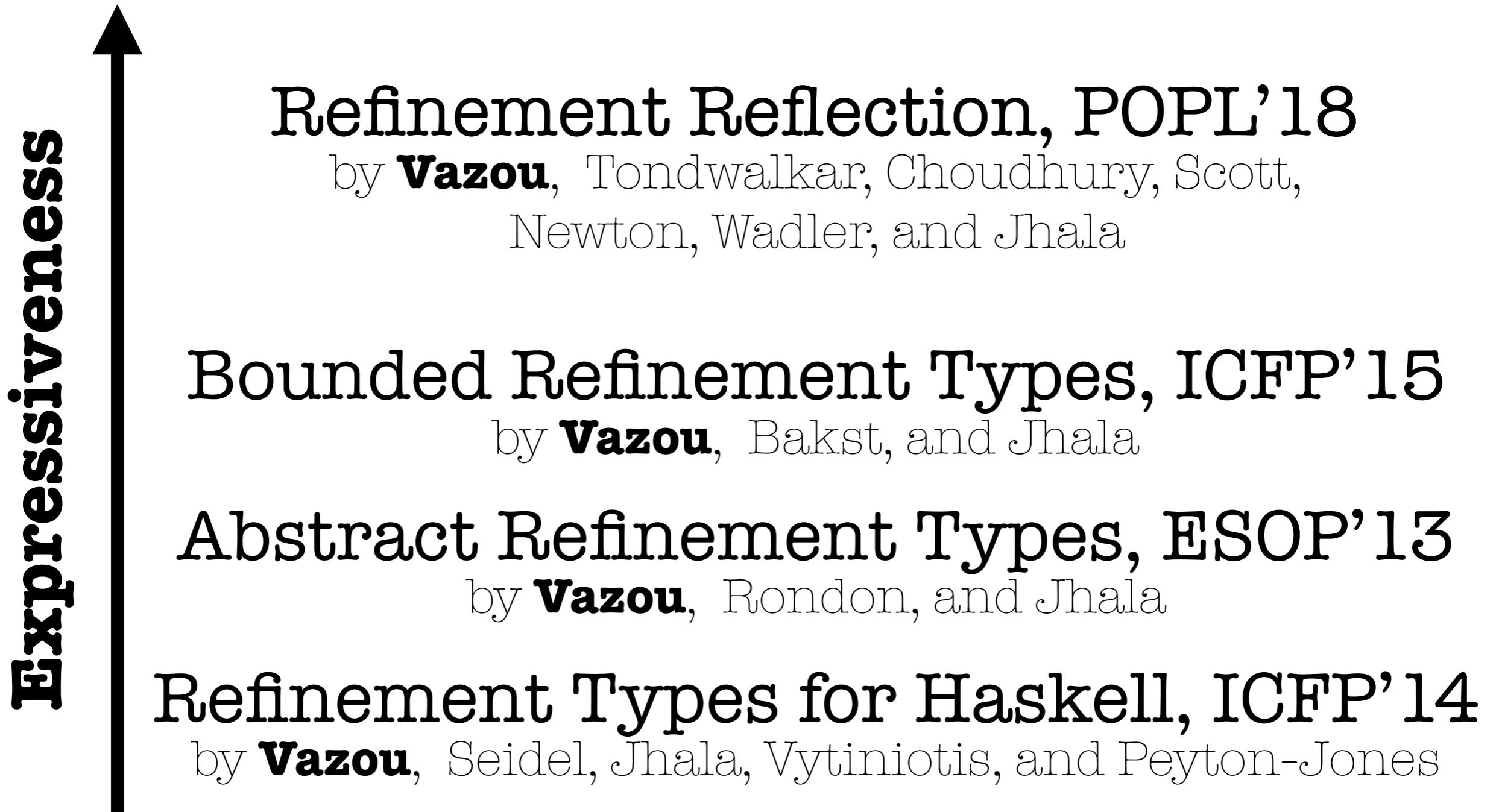


# Liquid Haskell on papers





# Liquid Haskell on papers



# Liquid Haskell on github

ucsd-progsys / liquidhaskell

Unwatch 22 Star 473 Fork 75

Code Issues 201 Pull requests 5 Projects 0 Wiki Insights Settings

Liquid Types For Haskell Edit

Add topics

8,382 commits 94 branches 19 releases 38 contributors

Branch: develop New pull request Create new file Upload files Find file Clone or download

nikivazou Merge pull request #1223 from ucsd-progsys/HaskellFunInRefs ... Latest commit 82f5baa 5 days ago

benchmarks move tests into pos a month ago

devel Fix travis error 6 days ago

# Liquid Haskell on github

Jan 15, 2012 – Jan 25, 2018

Contributions: Commits ▾



[ranjitjhala](#)

3,093 commits 474,271 ++ 304,536 --

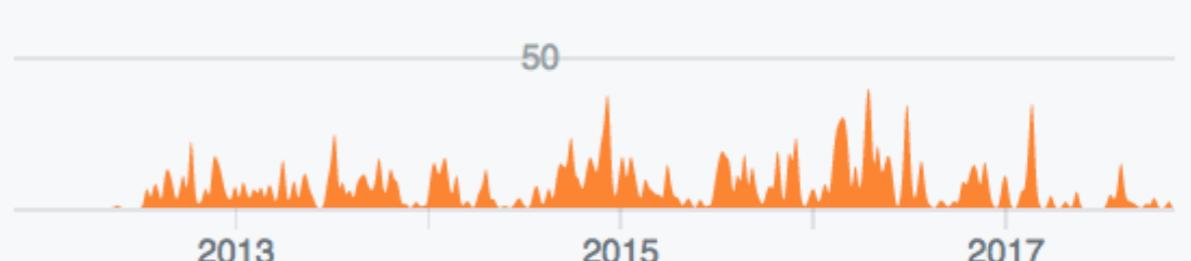
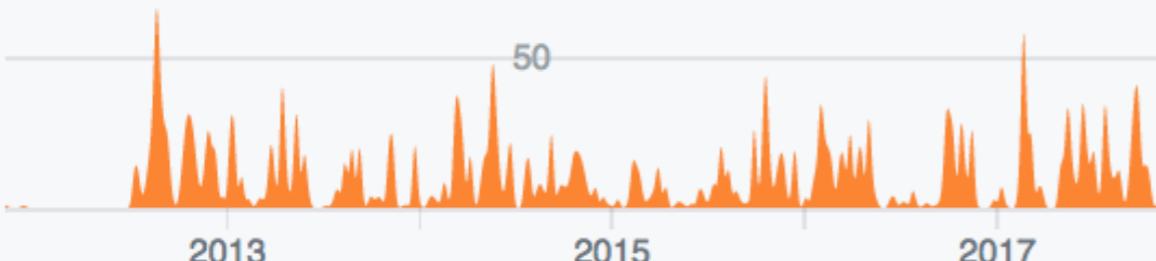
#1



[nikivazou](#)

2,113 commits 358,364 ++ 294,962 --

#2



[gridaphobe](#)

814 commits 114,731 ++ 79,008 --

#3



[spinda](#)

155 commits 7,430 ++ 5,350 --

#4

# Liquid Haskell dev team



Niki Vazou & Ranjt Jhala,  
Liquid Haskell dev team

# Liquid Haskell in the real world

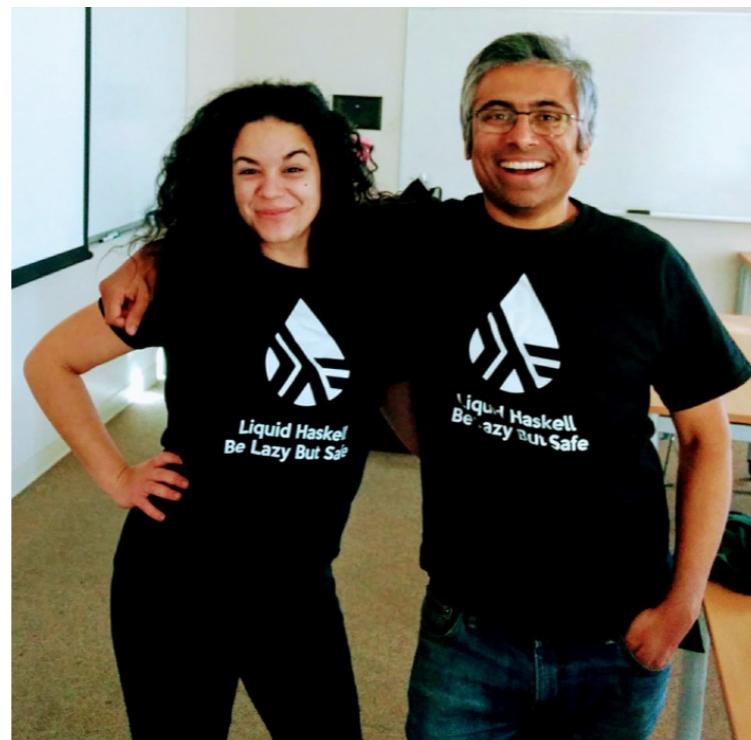
## Education



Will Kunkel,  
undergrad @UMD



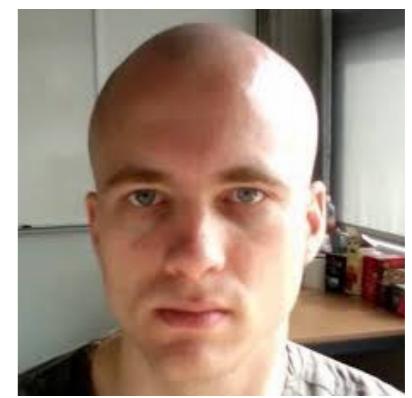
Rachel Xu & Xinyue Zhang,  
undergrads @Bryn Mawr College



Niki Vazou & Ranjt Jhala,  
Liquid Haskell dev team



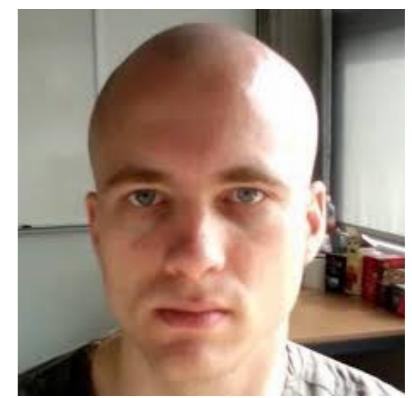
Gabriel Gonzalez  
Awake Security



Edsko de Vries  
Well-Typed



Gabriel Gonzalez  
Awake Security



Edsko de Vries  
Well-Typed

# Liquid Haskell in Education

This is Will Kunkel.

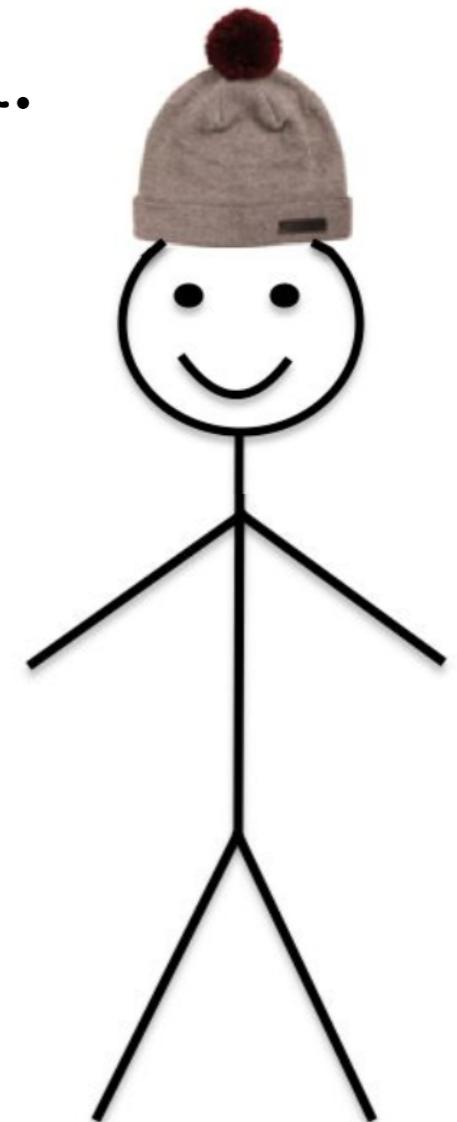
Will took my undergrad course on Haskell.

Will learnt Liquid Haskell in 2 weeks.

Will was **3rd** in POPL'18 **Student Research Competition** with his project “Comparing Liquid Haskell and Coq”

Will is smart.

Be like Will.



# Liquid Haskell in Education

Two winners in POPL'18 Student Research Competition



“Comparing Liquid Haskell and Coq”



Will Kunkel,  
undergrad @UMD



“Comparing Dependent Haskell,  
Liquid Haskell and F\*”



Rachel Xu & Xinyue Zhang,  
undergrads @Bryn Mawr College



# Liquid Haskell in the real world

## Education



Will Kunkel,  
undergrad @UMD



Rachel Xu & Xinyue Zhang,  
undergrads @Bryn Mawr College



Niki Vazou & Ranjt Jhala,  
Liquid Haskell dev team



Gabriel Gonzalez  
Awake Security



Edsko de Vries  
Well-Typed



# Liquid Haskell in Industry

Gabriel Gonzalez



Static checks during parsing.

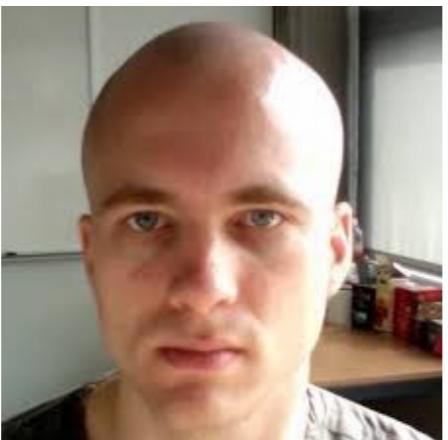
Provably Correct & Faster (x6) Code!

Huge speedup for internet traffic parsing.



# Liquid Haskell in Industry

Gabriel Gonzalez



Edsko de Vries





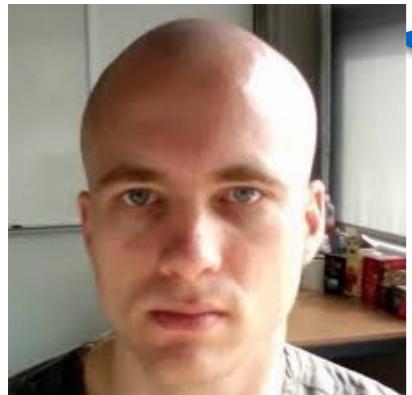
I am a Haskell consultant at a  
**cryptocurrency/blockchain** company.  
We have a blockchain algorithm written  
**in paper & a Haskell implementation.**  
We want Liquid Haskell to connect them.



I am a Haskell consultant at a  
**cryptocurrency/blockchain** company.  
We have a blockchain algorithm written  
**in paper** & a **Haskell implementation**.  
We want Liquid Haskell to connect them.

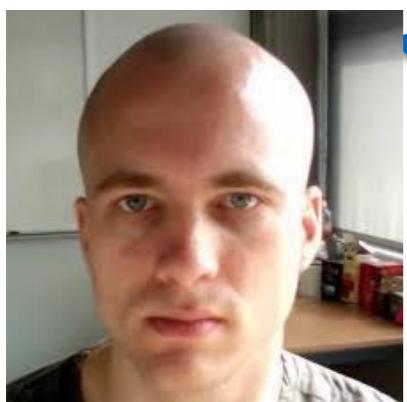
Awesome! Lmk if you need anything!





I am a Haskell consultant at a  
**cryptocurrency/blockchain** company.  
We have a blockchain algorithm written  
**in paper** & a **Haskell implementation**.  
We want Liquid Haskell to connect them.

Awesome! Lmk if you need anything!



I want **interactive proof generation**!

# Future work ideas are crowd sourced!



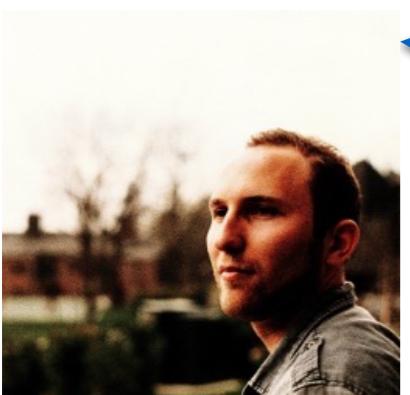
Edsko de Vries  
industrial rep.

I want **interactive proof generation**!



Richard Eisenberg  
ghc devs rep.

Can we use Liquid Haskell proofs for  
**compiler optimizations**?



Leo Lambropoulos  
Coq rep.

I do not trust it! Liquid Haskell should  
generate **proof certificates**!

# Future work ideas are crowd sourced!



Edsko de Vries  
industrial rep.

I want **interactive proof generation**!



Richard Eisenberg  
ghc devs rep.

Can we use Liquid Haskell proofs for  
**compiler optimizations**?



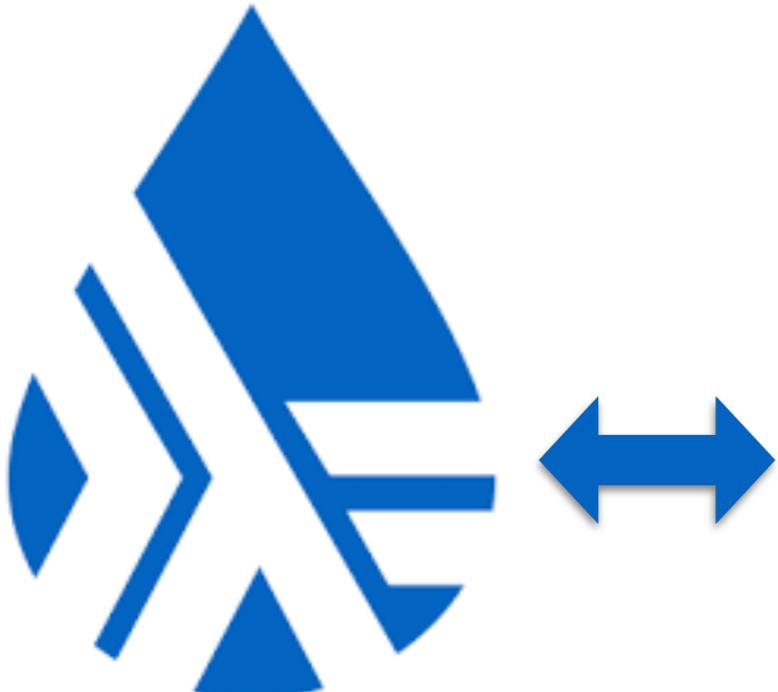
Leo Lambropoulos  
Coq rep.

I do not trust it! Liquid Haskell should  
generate **proof certificates**!

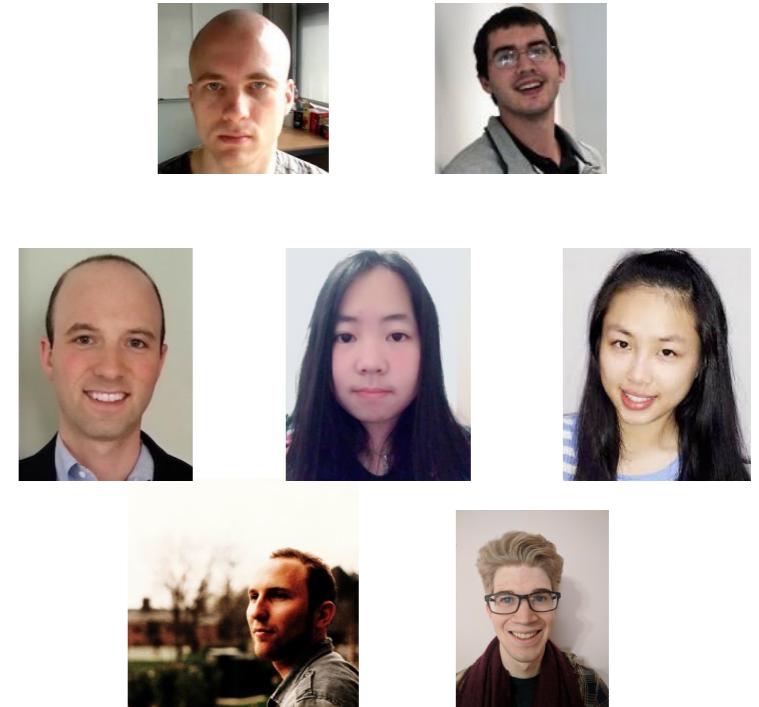


# Liquid Haskell has many users

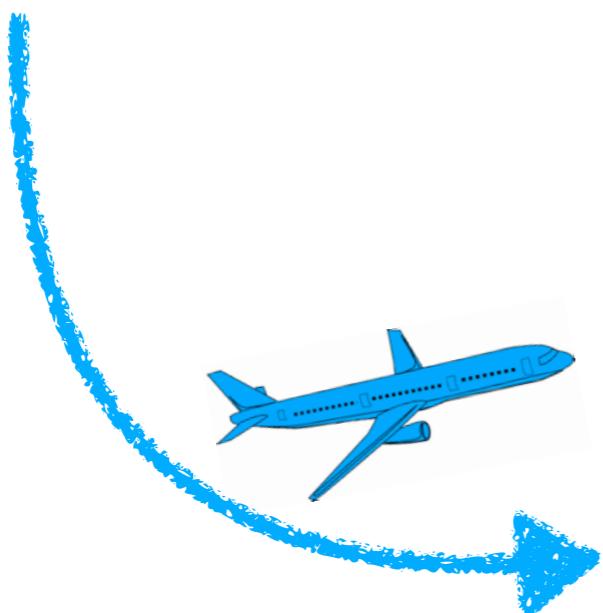
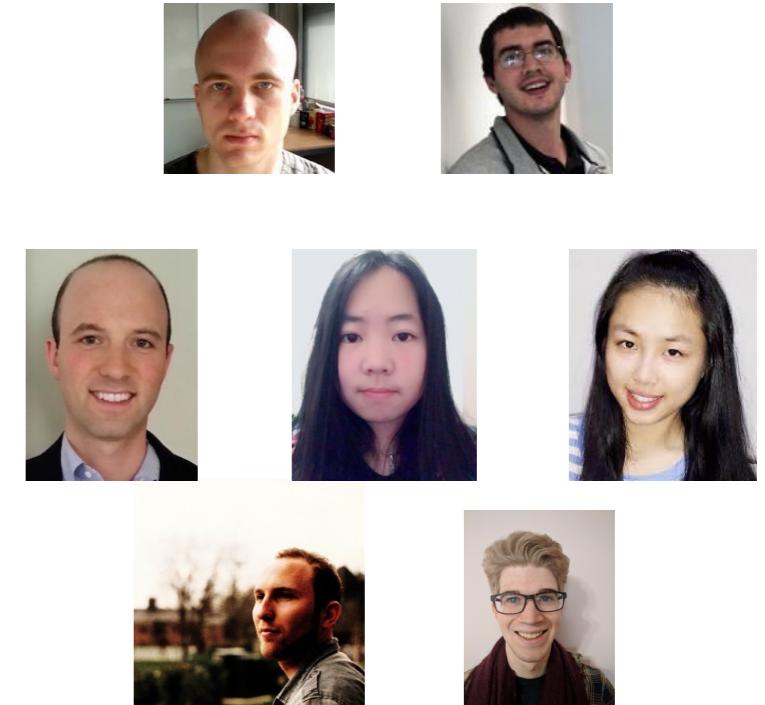
7,765 downloads



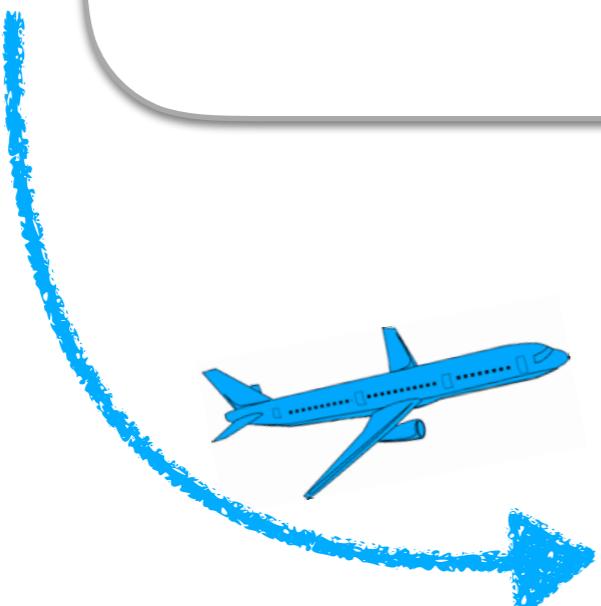
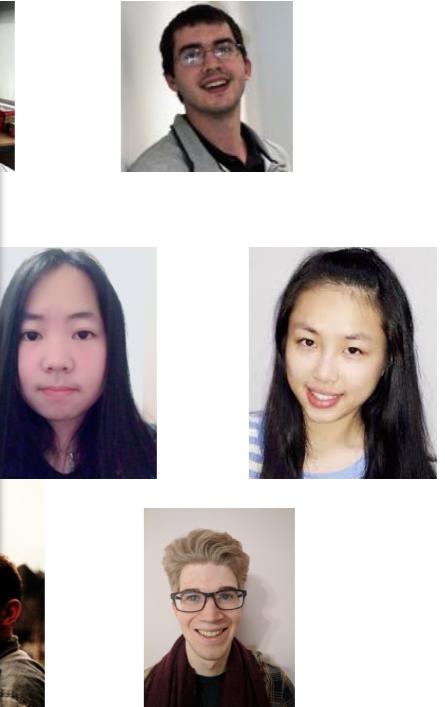
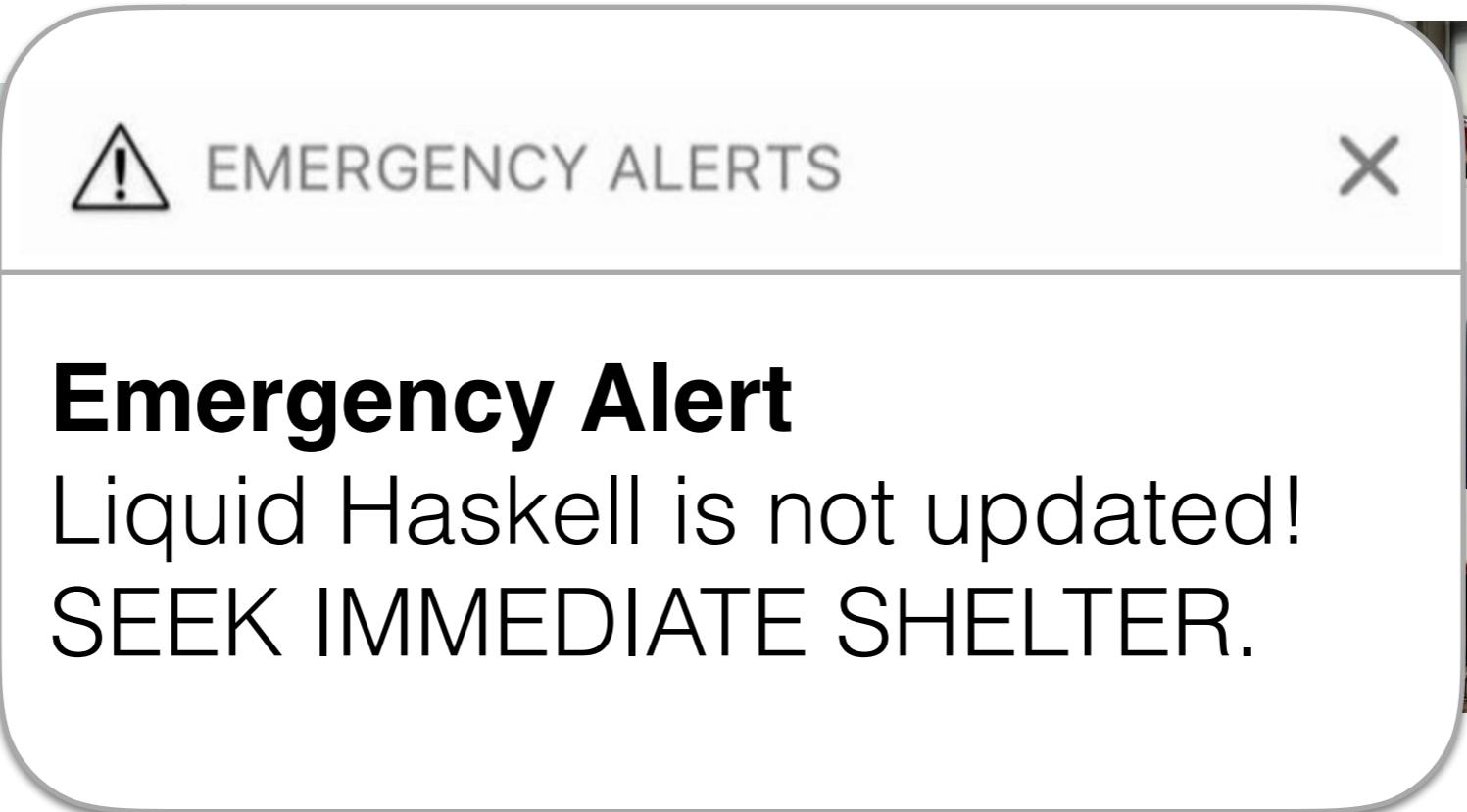
# many users, but two main devs



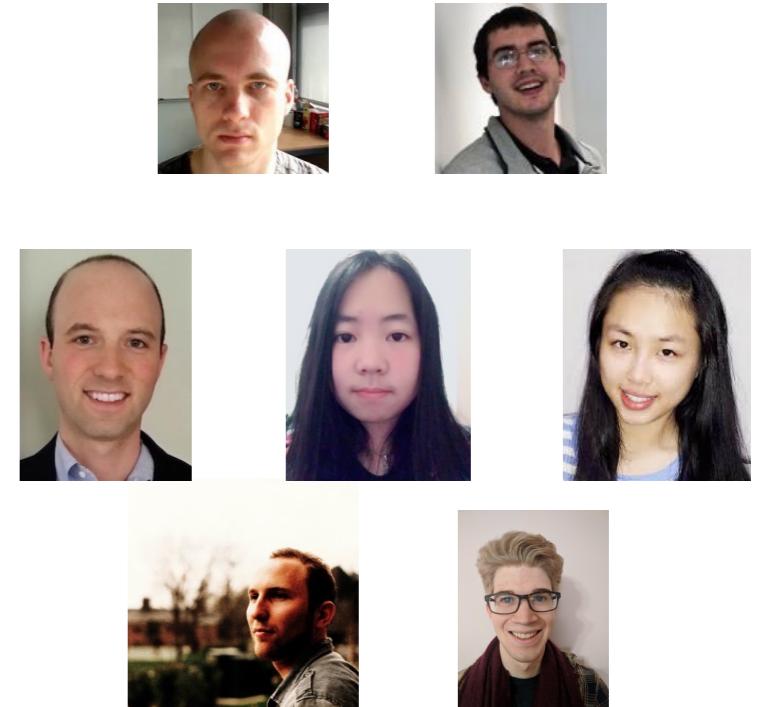
# many users, but two main devs



# many users, but two main devs



# many users, but two main devs



## Goal: Expand the Liquid Haskell Dev Team

# **Goal: Expand the Liquid Haskell Dev Team**

for

compiler optimizations,

interactive proof generation,

easy verification of real-world applications, ...

**Vision:**

**Use verification to aid programming in Haskell**

**Vision:**

# **Use verification to aid programming in Haskell**

Specs are just comments **inside** the languages, ...  
thus, learning effort is small.

Semi-**automatically** machine checked, via SMTs.

For runtime **optimizations**, by the user or the **compiler**.

# Vision: Use verification to aid programming in ~~Haskell~~ **X language**

**X = Ruby, JavaScript, Scala, ...**

Univ

**Refinement Types for Ruby**

**Refinement Types for TypeScript**

**SMT-Based Checking of Predicate-Qualified Types for Scala**

Georg Stefan Schmid    Viktor Kuncak  
EPFL, Switzerland  
`{firstname.lastname}@epfl.ch`



**Use verification to aid programming in Haskell**



# LiquidHaskell

## Use verification to aid programming in Haskell

### Fast & Safe Code Static Checks

### Theorem Proving Refinement Reflection

### Applications Education & Industry

Thanks!