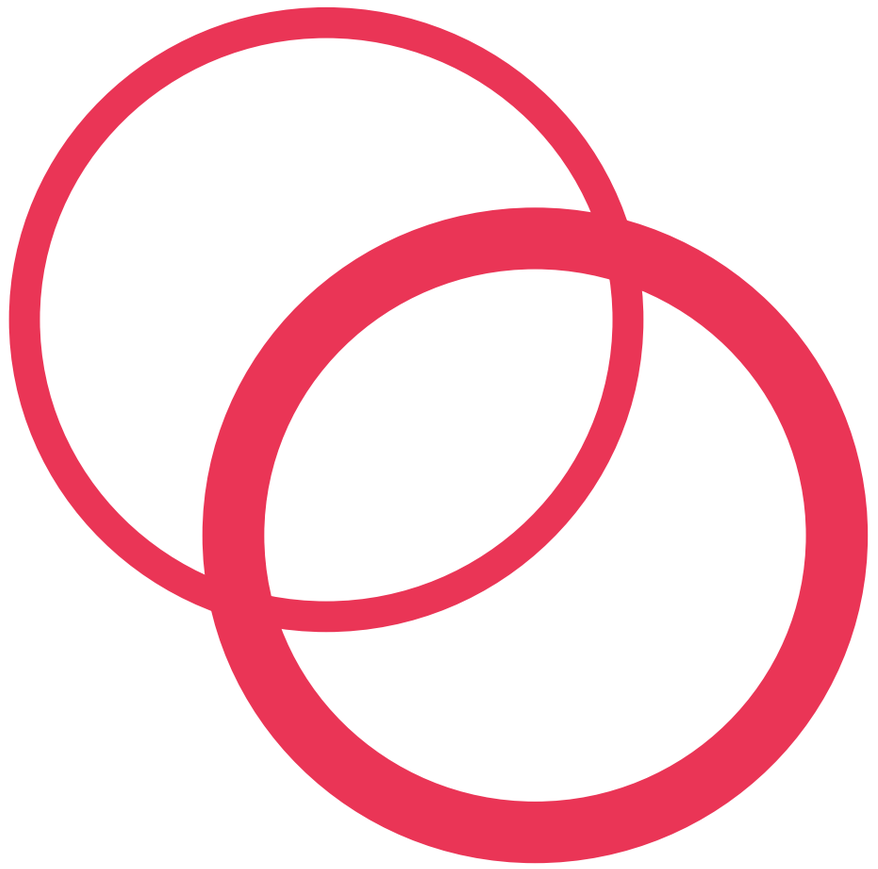
Unison: Programming the Global Supercomputer

Rúnar Bjarnason (@runarorama)
Lambda World Seattle, September 2018



unison

http://unisonweb.org

Who am I?

- Cofounder, **Unison Computing**

- Author, **Functional Programming in Scala**

Unison

- A new programming language
- Permissive MIT license
- Modern, statically typed, functional
- Some unique features and constraints

UNDER CONSTRUCTION

The Plan

1. Motivation for Unison
2. A small Unison program
3. Unison's approach to names
4. Rant about error messages
5. Hate on monads for a bit
6. Roadmap for Unison

Why a new language?

By Bill Bertram - Own work, CC BY-SA 2.5, https://commons.wikimedia.org/w/index.php?curid=170050

Structure and Interpretation of Computer Programs

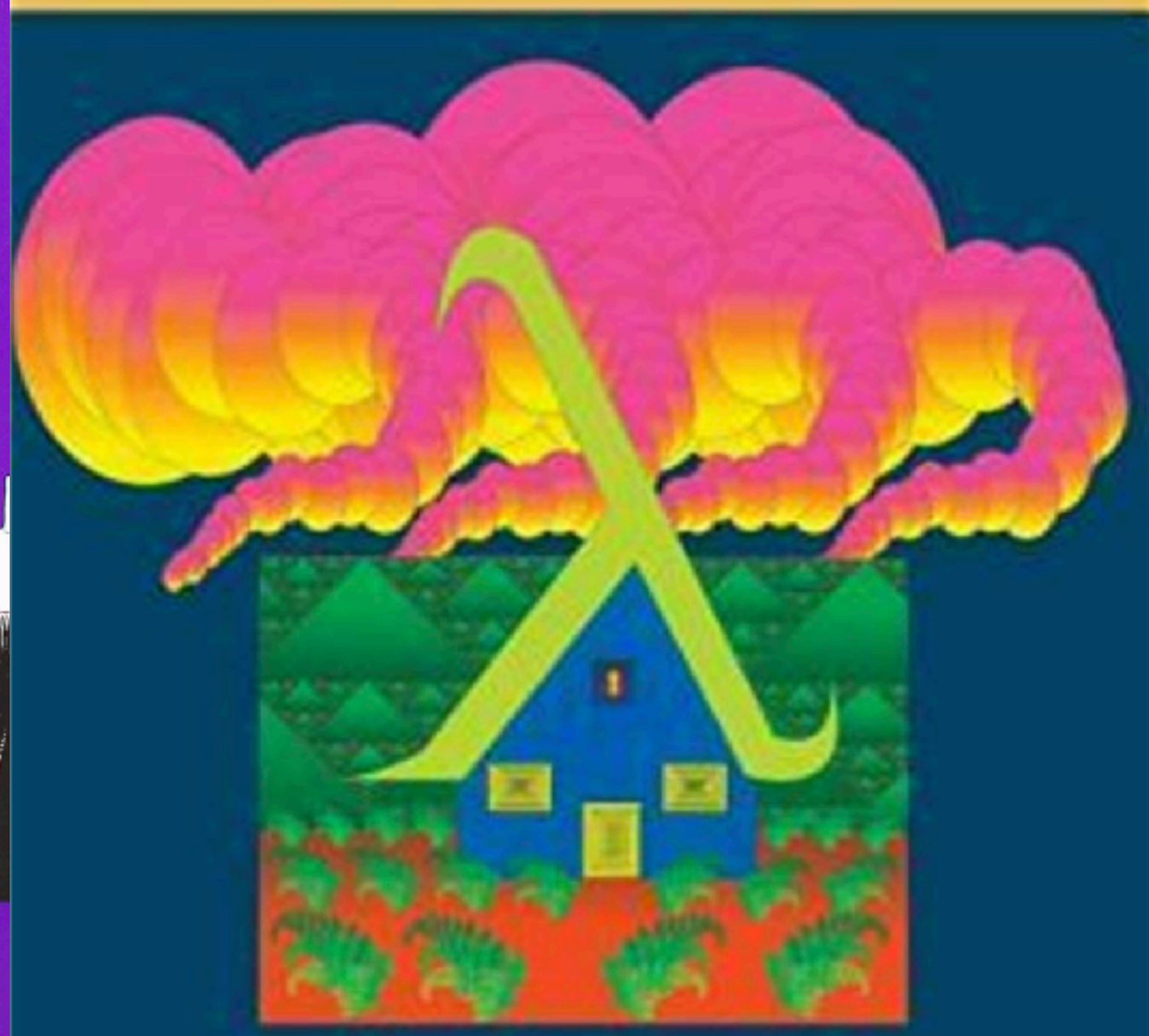
Second Edition

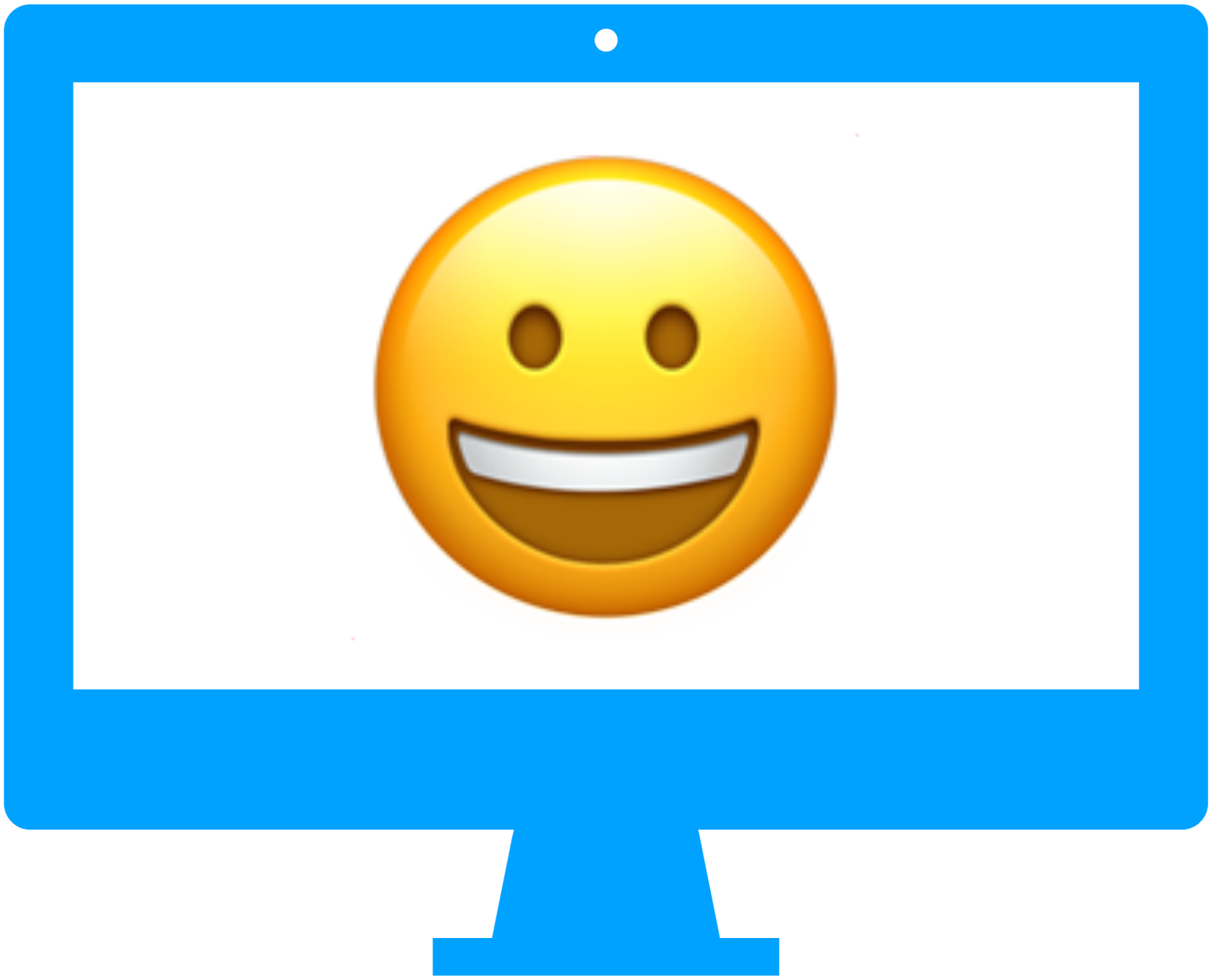
Harold Abelson and
Gerald Jay Sussman
with Julie Sussman

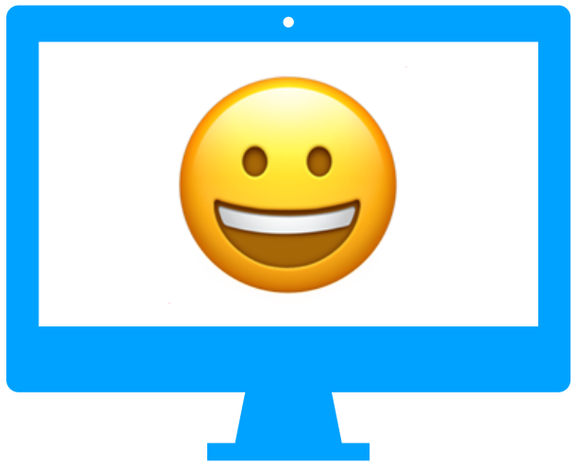
The Haskell School of Expression

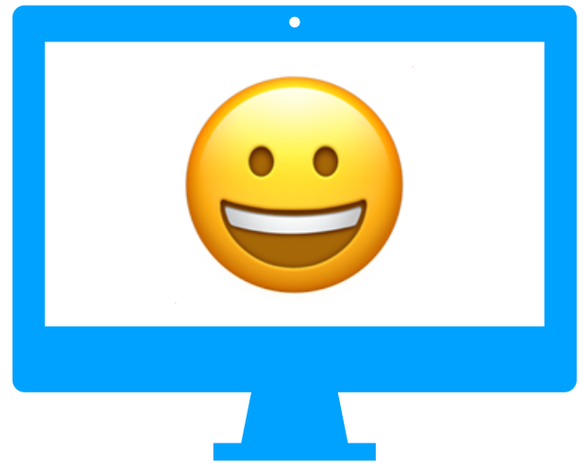
LEARNING FUNCTIONAL PROGRAMMING THROUGH MULTIMEDIA

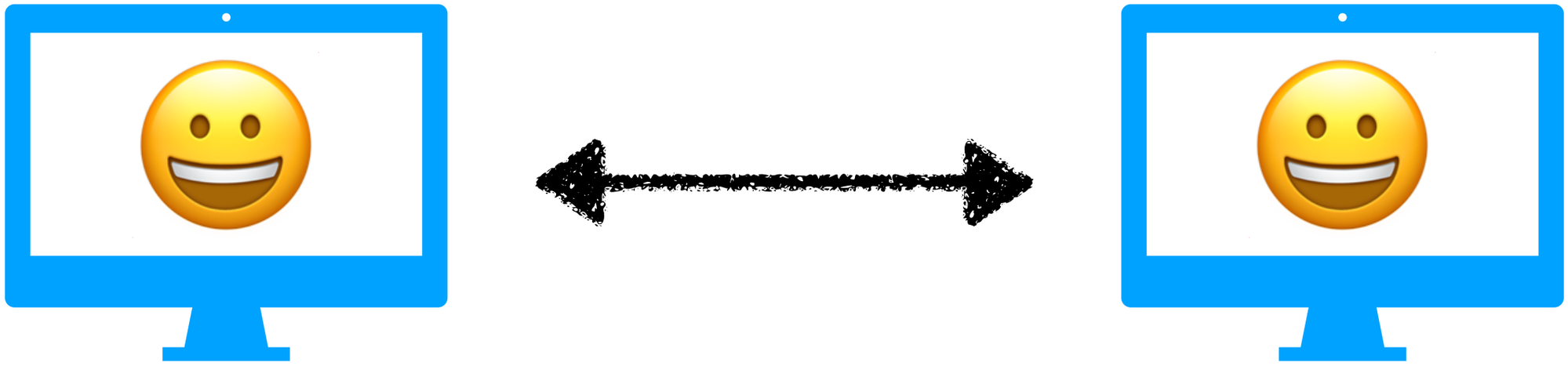
PAUL HUDAK





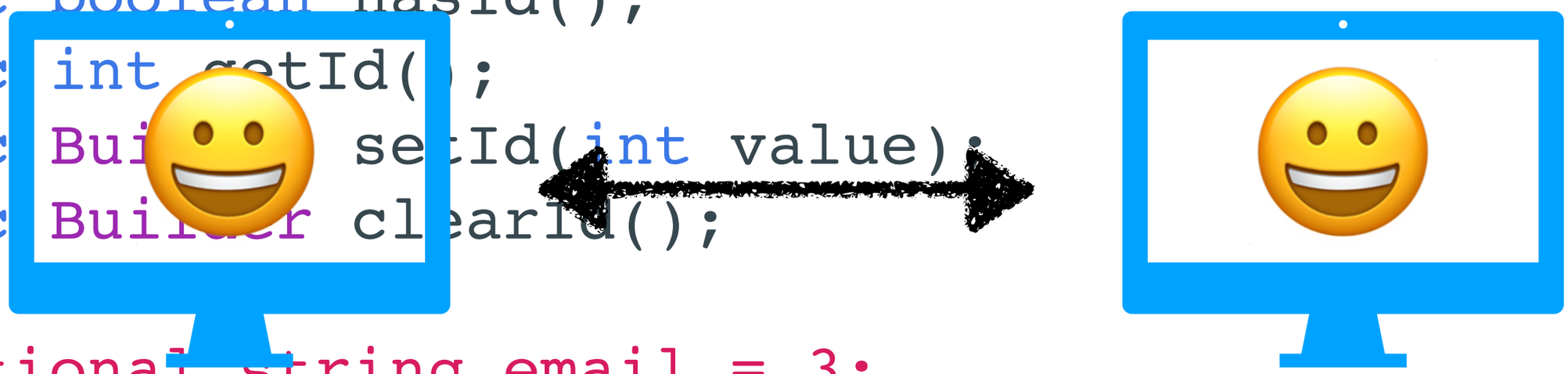


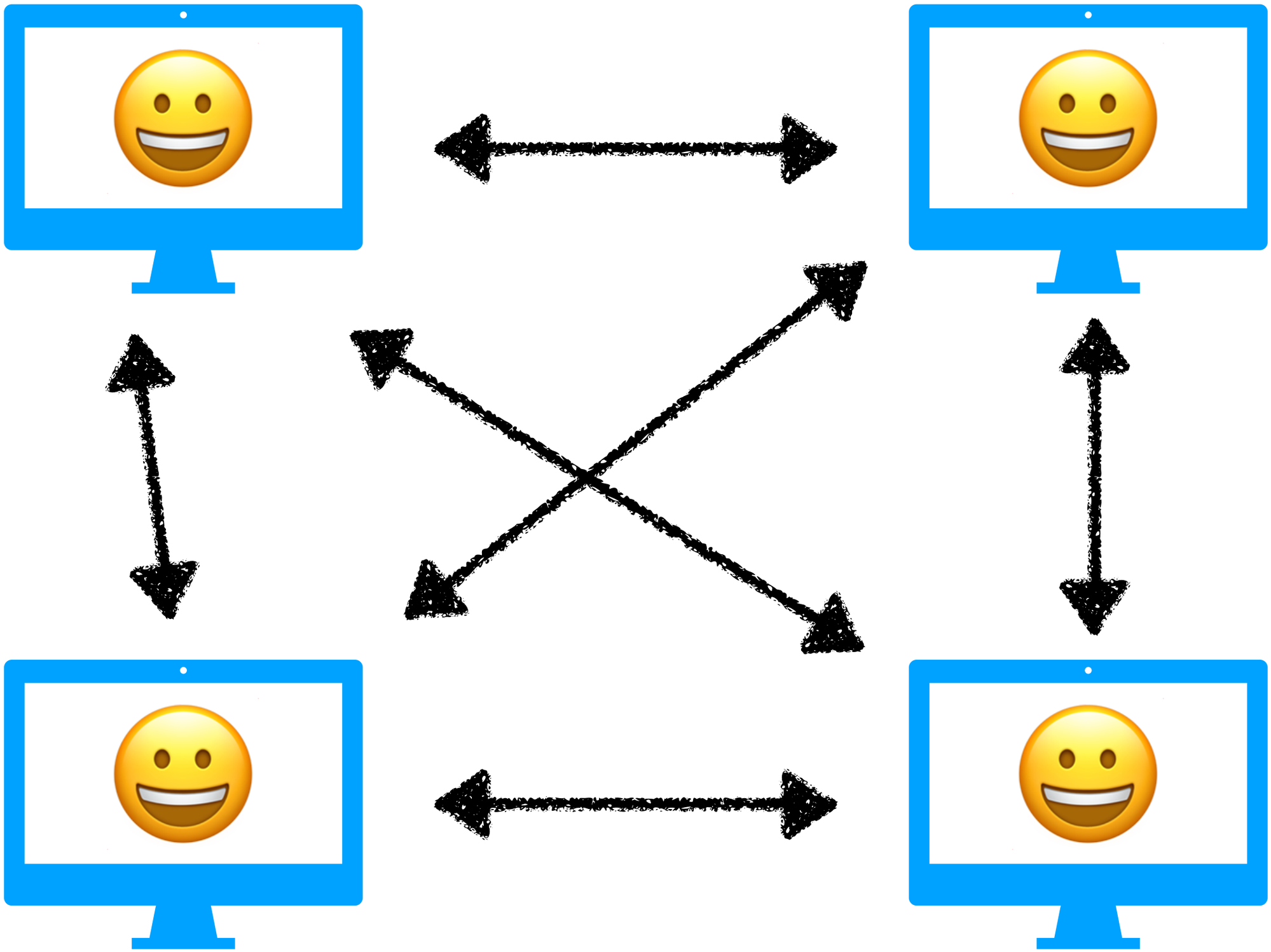


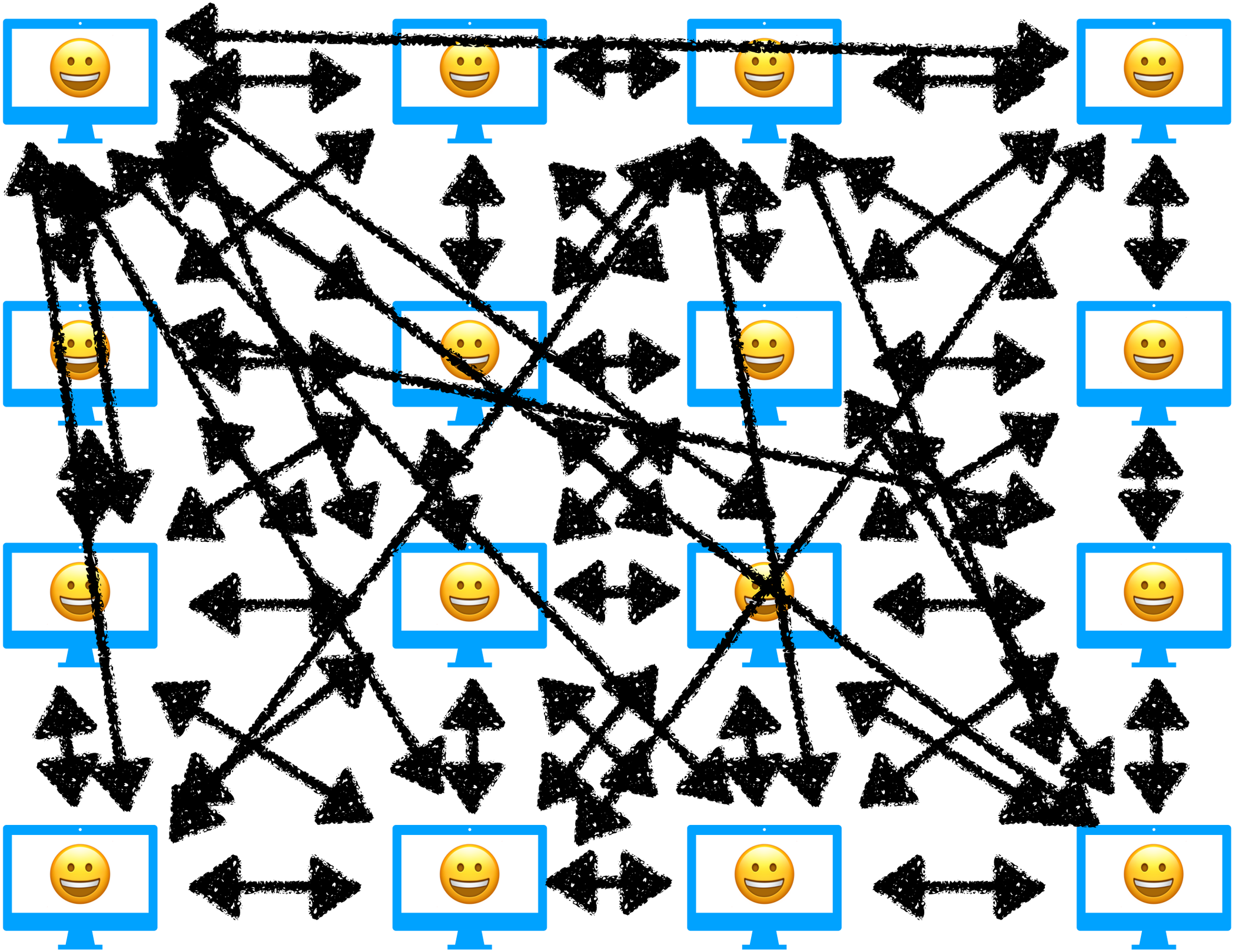


```
    "last": "http://example.com/articles?
page[offset]=10"
  },
  "data": [{
    "type": "articles",
    "id": "1",
    "attributes": {
      "title": "JSON API paints my bikeshed!"
    },
    "relationships": {
      "author": {
        "links": {
          "self": "http://example.com/articles/1/
relationships/author",
          "related": "http://example.com/articles/1/
author"
        },
        "data": { "type": "people", "id": "9" }
      },
      "comments": {
        "links": {
```


```
// required string name = 1;
public boolean hasName();
public java.lang.String getName();
public Builder setName(String value);
public Builder clearName();

// required int32 id = 2;
public boolean hasId();
public int getId();
public Builder setId(int value);
public Builder clearId();

// optional string email = 3;
public boolean hasEmail();
public String getEmail();
public Builder setEmail(String value);
public Builder clearEmail();

// repeated .tutorial.Person.PhoneNumber phones = 4;
public List<PhoneNumber> getPhonesList();
```







```
    "last": "http://example.com/articles?
// required string name = 1;
page[offset]=10"
public boolean hasName();
  },
public java.lang.String getName();
  "data": [{
public Builder setName(String value);
    "type": "articles",
public Builder clearName();
    "id": "1",

    "attributes": {
// required int32 id = 2;
      "title": "JSON API paints my bikeshed!"
public boolean hasId();
    }
public int getId();
    "relationships": {
public Builder setId(int value);
      "author": {
public Builder clearId();
        "links": {
          "self": "http://example.com/articles/1/
// optional string email = 3;
relationships/author",
public boolean hasEmail();
          "related": "http://example.com/articles/1/
public String getEmail();
author"
public Builder setEmail(String value);
        }
public Builder clearEmail();
      }
    }, {
// repeated .tutorial.Person.PhoneNumber phones = 4;
      "type": "people", "id": "9"
    "comments": {
public List<PhoneNumber> getPhonesList();
      "links": {
```

You need to make multiple programs talk to each other. A software system these days is typically a mono

Protobuf

Puppet

Spark

YAML

Chef

Ansible

Akka

Airflow

Jenkins

Terraform

Finagle

Kubernetes

JSON

Docker

Mesos

Kafka

Pagerduty

Programming languages only talk about a single OS process.

A Unison program describes an entire distributed system.

- Write a single program for your whole system.
- Deploy the system by running that program.

Unison: The internet as a global supercomputer that you program simply and directly.

```
(mapReduce f monoid list =
  (force (foldMap (a → fork '(f a))
                  (par monoid)
                  list)
```

```
mapReduce : (a →{Remote} b)
          → Monoid b
          → [a]
          →{Remote} b
mapReduce f monoid list =
  force (foldMap (a → fork '(f a))
                 (par monoid)
                 list)
```

```
mapReduce : (a →{Remote} b)
          → Monoid b
          → [a]
          →{Remote} b
mapReduce f monoid list =
  force (foldMap (a → fork '(f a))
                 (par monoid)
                 list)
```

Unison can transfer an arbitrary computation to a remote node.

```
Remote.at usEastProduction
  '(mapReduce clickCount
    clickMonoid
    logFiles)
```

```
Remote.at usEastProduction
'(#2Pkc2rZH #6ayc1iGb
#RexqY99x
#K8xWGcLT)
```

Unison identifies a function or type by a hash of its definition.

Unambiguously send code as data, across time and space.

"There are 2 hard problems in computer science: cache invalidation, naming things, and off-by-1 errors."

— Leon Bambrick

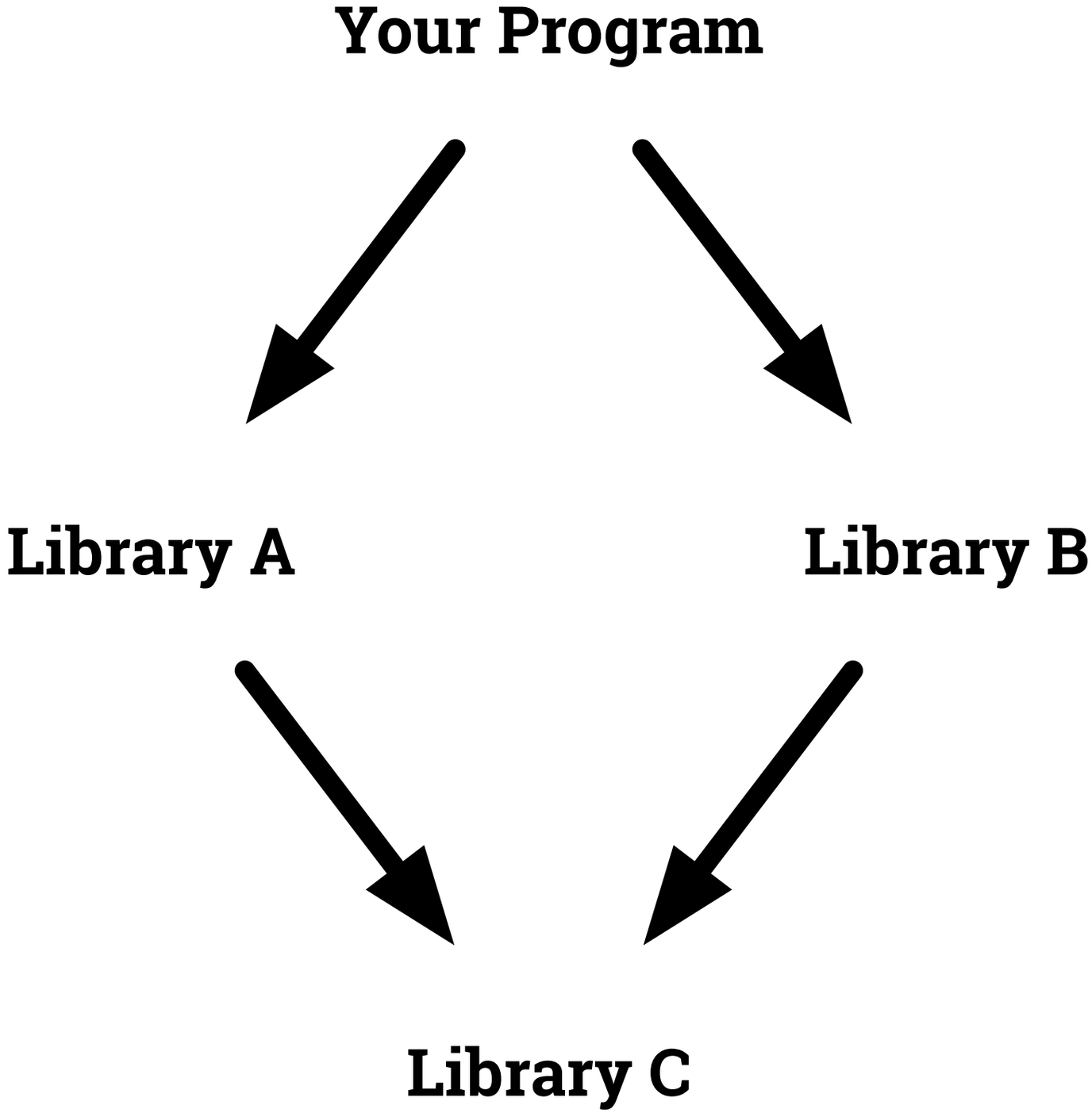
"There are 2 hard problems in computer science: cache invalidation, **naming things**, and off-by-1 errors."

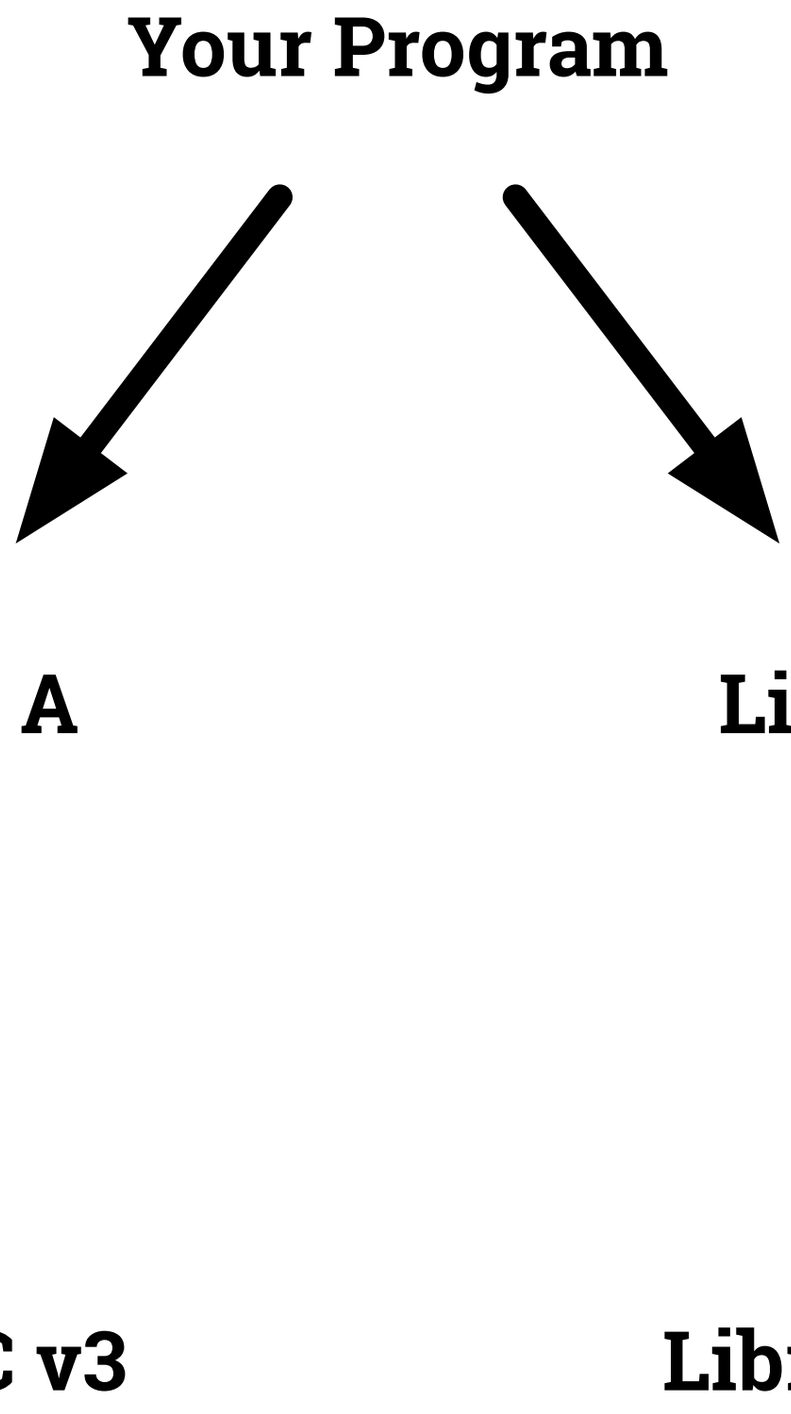
— Leon Bambrick

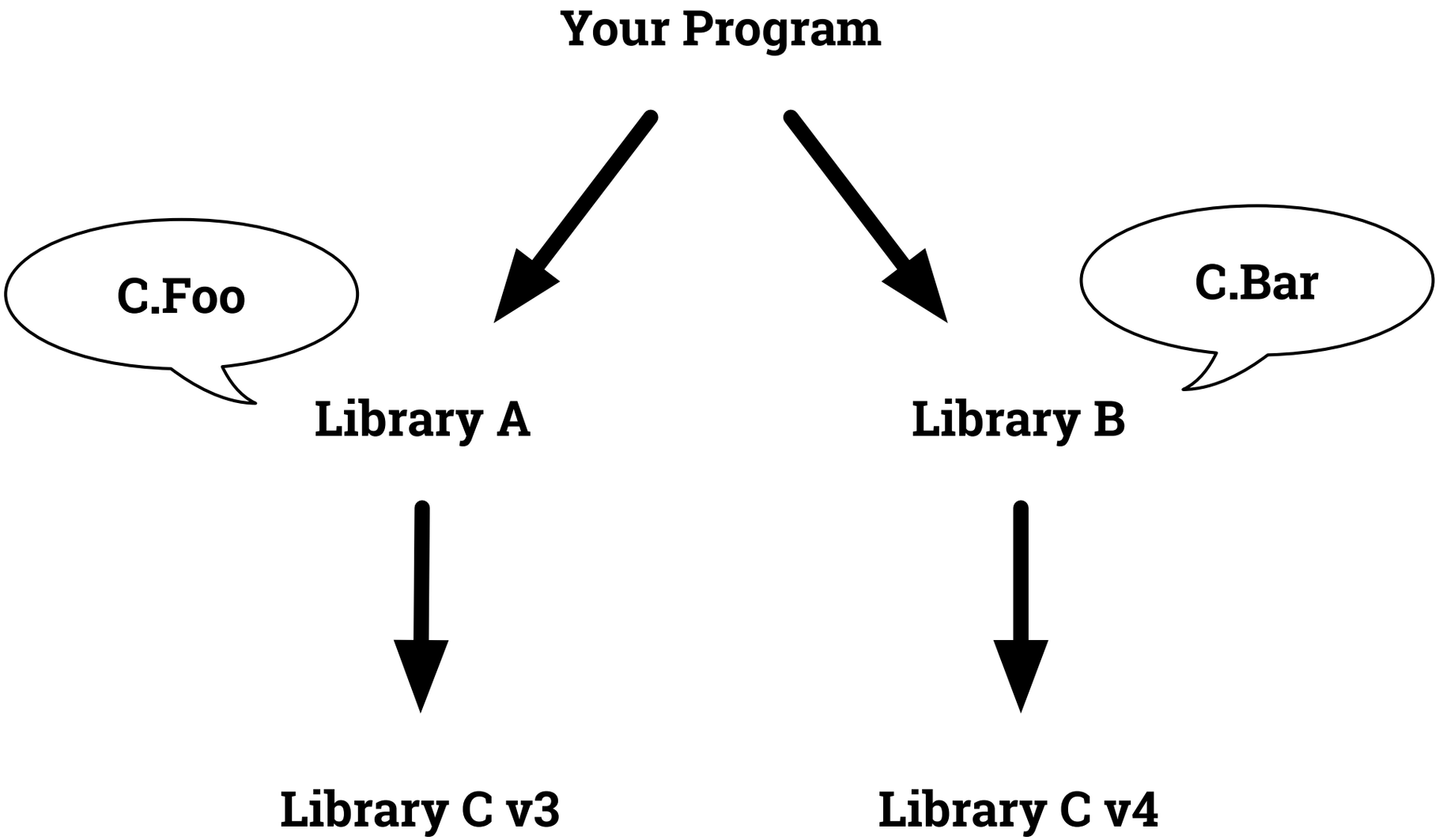
```
curry : ((a,b) → c) → a → b → c
curry f a b = f (a,b)

schönfinkel g x y = g (x,y)
```

A Unison codebase is a database.

- Perfect incremental compilation.
- Refactoring is a controlled experience.
- Many sources of merge conflicts cannot occur.
- Renaming is trivial.
- Dependency hell is vastly reduced.

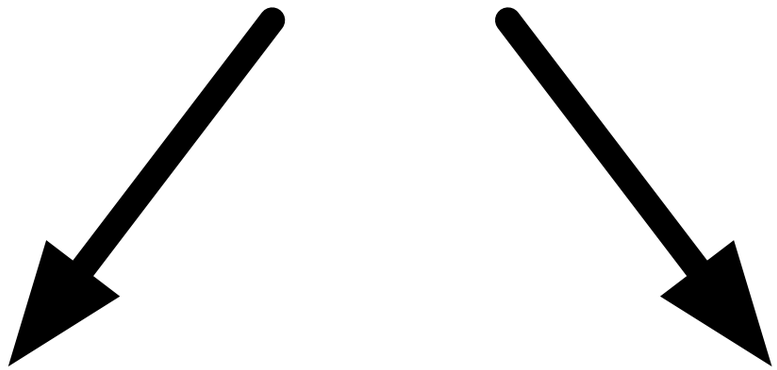

Your Program
Library A
Library B
Library C


Your Program
Library A
Library B
Library C v3
Library C v4

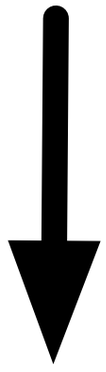

Your Program
C.Foo
C.Bar
Library A
Library B
Library C v3
Library C v4

Unison tracks dependencies at the level of individual hashes.

Your Program

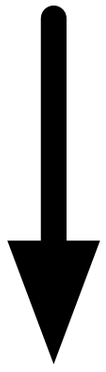


#hgh38g

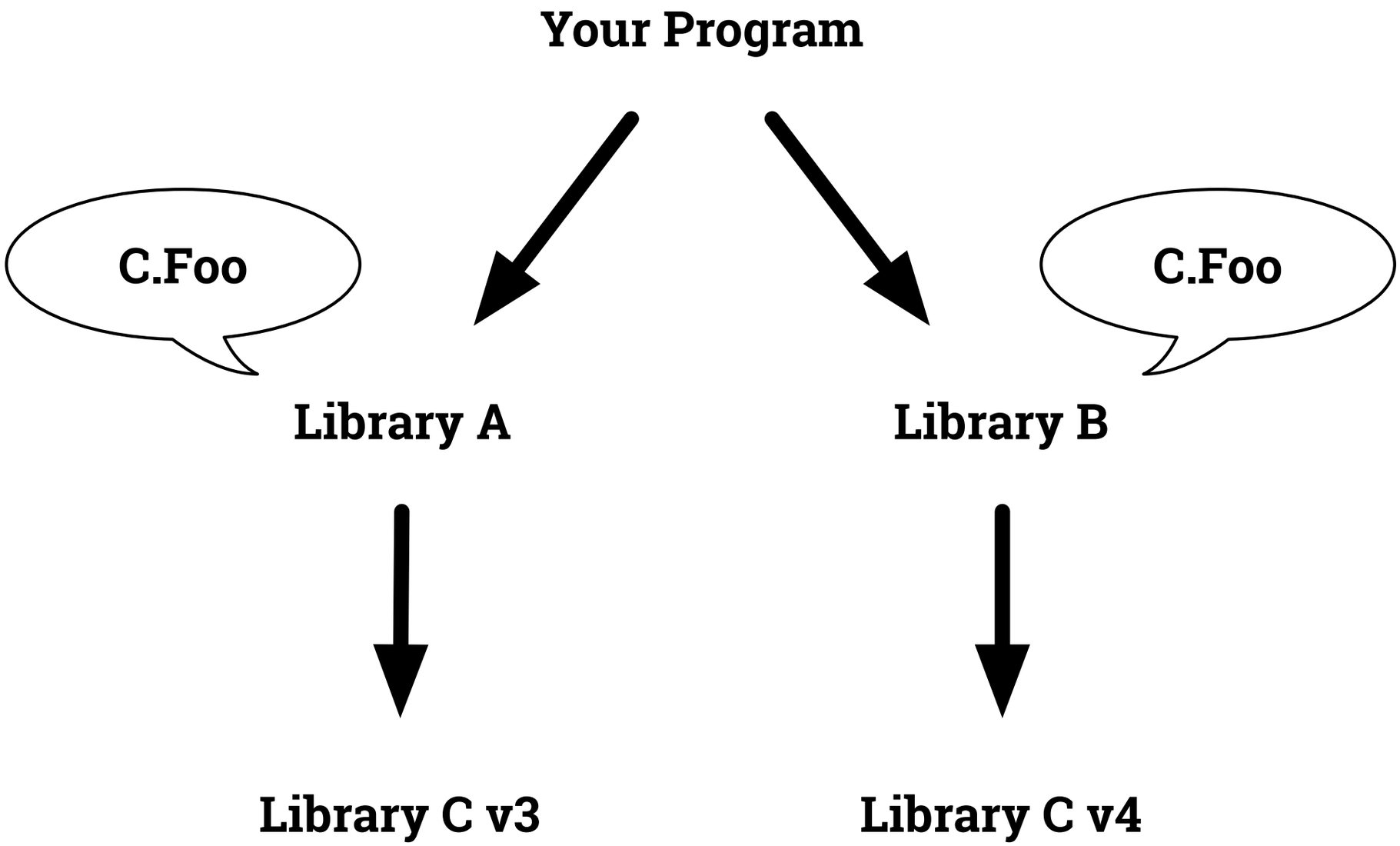


#88g9ff

#8fhsji



#a80f0b


Your Program
C.Foo
C.Foo
Library A
Library B
Library C v3
Library C v4

Unison's type system

Complete and Easy Bidirectional Typechecking for Higher-Rank Polymorphism

Joshua Dunfield, Neelakantan R. Krishnaswami

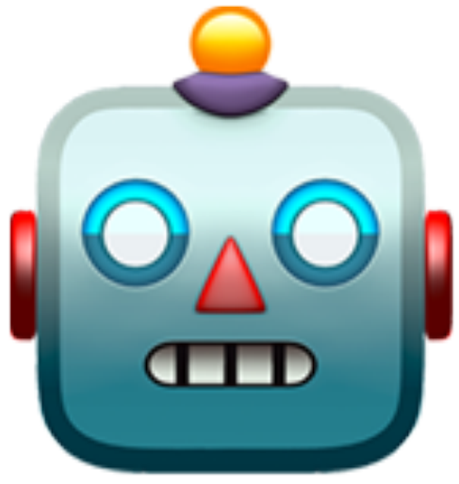
https://arxiv.org/abs/1306.6032

Type Error Provenance

Lennart Augustsson

https://www.youtube.com/watch?v=rdVqQUOvxSU

Low-hanging fruit: Error messages



ERROR

An error should read like another developer explaining the problem to you.

Haskell

```
• No instance for (Num [Char]) arising
from the literal ‘4’
  • In the expression: 4
    In a case alternative: 3 → 4
    In the expression:
      case x of
        3 → 4
        4 → "Surprise!"
```

```
All cases of a case/of expression must
have the same type.
  Here, one is Nat, and another is Text:
2 |    3 → 4
3 |    4 → "Surprise!"
```

```
scala> x match {
     |   case 3 ⇒ 4
     |   case 4 ⇒ "Surprise!"
     | }
```

```
scala> x match {
     |   case 3 ⇒ 4
     |   case 4 ⇒ "Surprise!"
     | }
res0: Any = 4
```

An error should make it obvious how to fix the problem.

This looks like a function call, but with a `Text` where the function should be. Are you missing an operator?

```
48 |   Some name → write ("Hello" name)
```

Haskell

```
3:7: error:
    • Couldn't match expected type ‘[Char] → a’
                   with actual type ‘[Char]’
    • The function ‘"Hello"’ is applied to one argument,
      but its type ‘[Char]’ has none
      In the first argument of ‘Just’, namely ‘("Hello" name)’
      In the expression: Just ("Hello" name)
    • Relevant bindings include
        …
```

1. **Where was the mismatch?**
2. **What types were involved?**
3. **Where did those types come from?**

```
The 4th argument to the function foo is
"Text", but I was expecting Nat

    3 | foo 1 2 3 "hello" 5

because the function has type

 a → a → a → a → a → a

where:

a = Nat, from here:

3 | foo 1 2 3 "hello" 5
```

If the solution to an error is trivial or obvious, just fix it.

If a name is ambiguous, Unison will pick the hash that typechecks.

```
I'm not sure what + means at Line 1, columns
13-14

    1 | foo a b = a + b

Whatever it is, it has a type that conforms to
a → b → o

I found some terms in scope that have matching
names and types. Maybe you meant one of these:

  - Nat.+ : Nat → Nat → Nat
  - Int.+ : Int → Int → Int
  - Float.+ : Float → Float → Float
```

Typed holes

```
I'm not sure what apend means at Line 5,
columns 6-11

    5 | sum (apend left right)

Whatever it is, it has a type that conforms to
Stream Nat → Stream Nat → Stream Nat
```

Unison's Effect System

Let's be honest: monads are awkward.

Do Be Do Be Do

Sam Lindley, Conor McBride,
Craig McLaughlin

https://arxiv.org/abs/1611.09259

Abilities

```
ability State s where
  put : s → {State s} ()
  get : {State s} s
```

```
pop : '{State [a]} Optional a
pop = '(stack = get
          put (drop 1 stack)
          head stack)

push : a → {State [a]} ()
push a = put (cons a get)
```

Applicative programming is the default

```
-- Scala
for {
  a ← x
  b ← y
  c ← z
} yield f(a, b, c)

-- Unison
f x y z
```

```
-- Haskell
f <$> a <*> b <*> c
```

```
-- Unison
f a b c
```

```haskell
-- Haskell
f <$> lift (lift a)
  <*> pure b
  <*> c
```

```haskell
-- Unison
f a b c
```

```haskell
-- Haskell
x >>= (\a -> f a >>= g)
```

```haskell
-- Unison
g (f x)
```

```
stackProgram : '{State [Nat]} ()
stackProgram =
  '( a = pop
     if a == 5
     then push 5
     else
       push 3
       push 8 )
```

```
state : s → '({State s} a) → a
state s c =
  h s e = case e of
    { State.get → k } →
      handle h s in k s
    { State.put s → k } →
      handle h s in k ()
    { a } → a
  handle h s in !c

runStack : [Nat]
runStack = state [5,4,3,2,1] stackProgram
```

```
feed : (a → {Receive Text} b)
     → a
     → {IO} b
feed r a =
  h x = case x of
    { Receive.receive → k } →
      handle h in k readLine
    { a } → a
  handle h in r a
```

The expression at Line 32, columns 1-11 is requesting the {IO} ability, but this location provides no abilities.

```
32 | IO.readLine
```

Yes, you can still use monads.

Wrap-up & Roadmap

As of Sept 2018

- Lexer, parser, pretty-printer, hashing, serialization
- Bidirectional typechecking, type-directed name resolution
- Algebraic data types, pattern matching
- algebraic abilities, ability polymorphism
- Structured type errors, type error provenance.
- JVM-based runtime with partial evaluation
- Native runtime reference implementation written in Haskell

Near-term Roadmap

- Unison CLI (codebase, REPL)
- Distributed runtime
- Core libraries (lists, streaming, storage, I/O, distributed data structures)
- Documentation, formal spec,
  glitter and sparkle
- Sweet-looking website with tutorials and examples

First Unison release: spring 2019

http://unisonweb.org