

# Rigorous Methods for Language Design

or - Don't take my word for it. Measure!

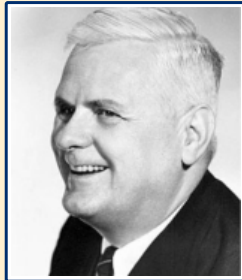
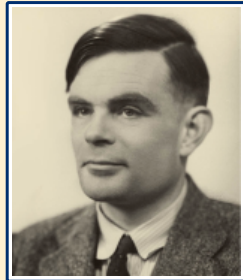


Ben Greenman

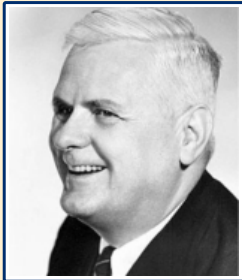
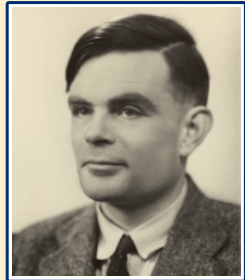
2024-10-04



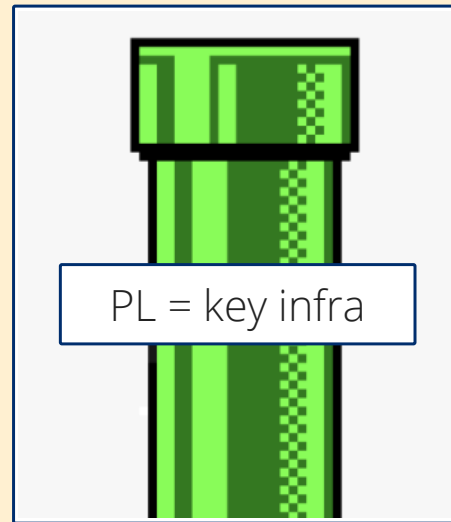
## Programming Languages - Why?



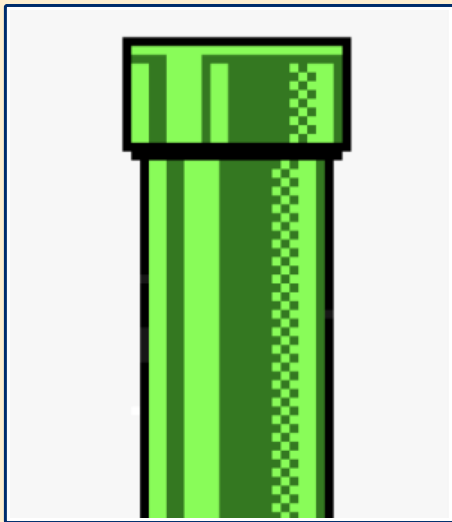
## Programming Languages - Why?

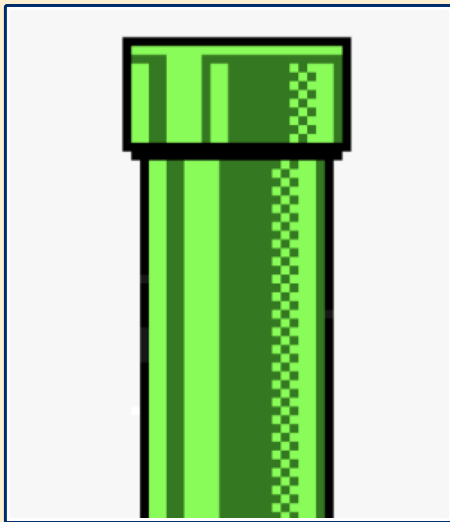


# Programming Languages - Why?









First-class Functions

Type Soundness

Polymorphism

Garbage Collection

Metaprogramming

Gradual typing





20 years?





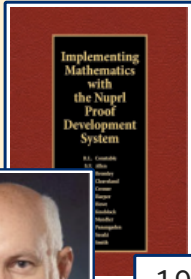
## Is Computing an Experimental Science?

Robin Milner, *Laboratory for Foundations of Computer Science, Edinburgh University*

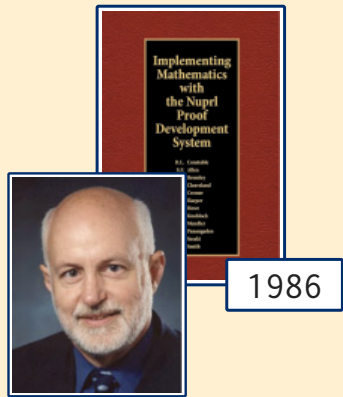
### Is computing an experimental science?<sup>(1)</sup>

At the Laboratory for Foundations of Computer Science at Edinburgh we are beginning an ambitious programme of research. The particular programme which we have put forward is a new

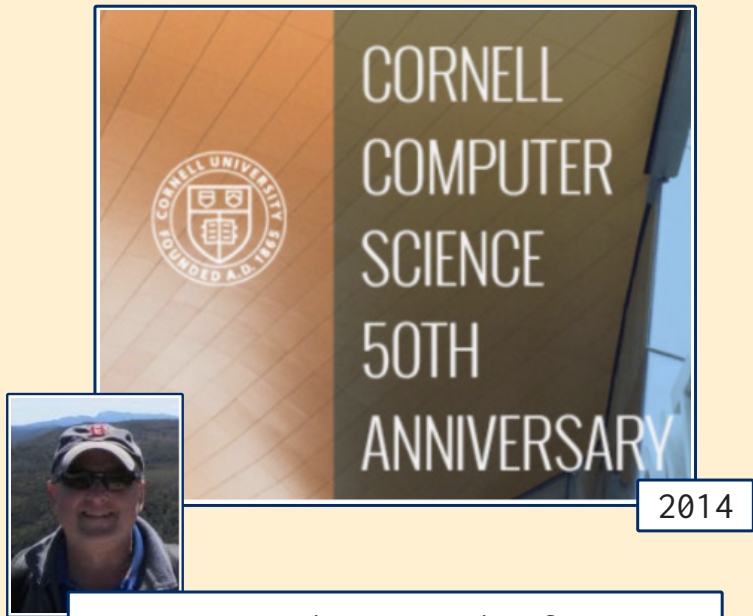
stone of theoretical computer science; it concerns what is computable, which is strongly connected to what is deducible. Around 1900, part of Hilbert's programme was to show that every mathematical truth would turn out to be a deducible from a set of axioms. and it was vital to



1986



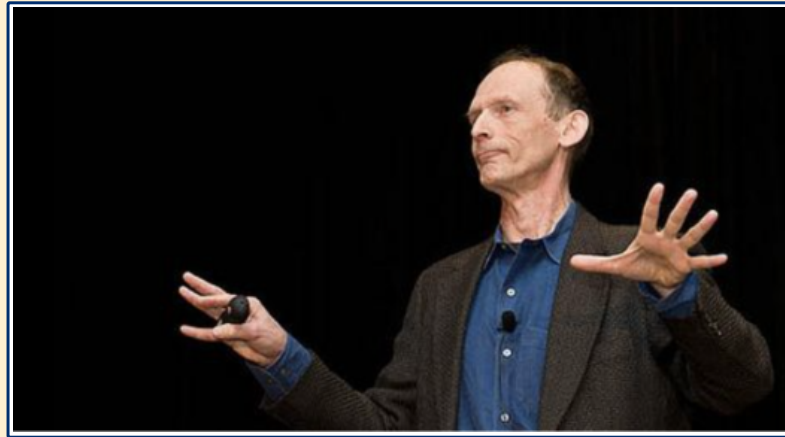
1986



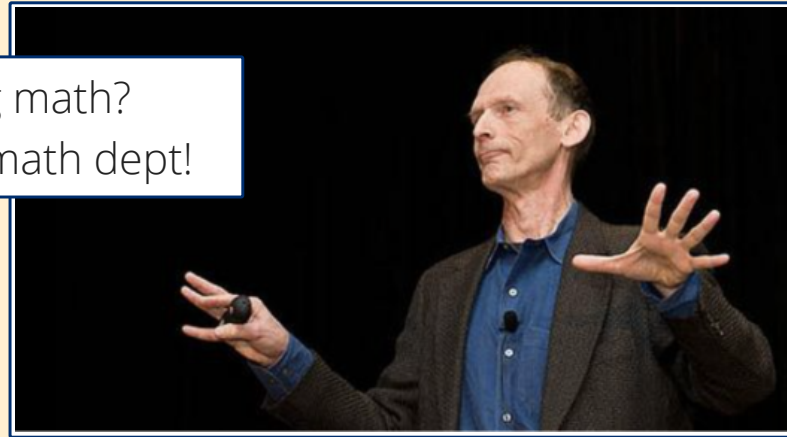
2014

"We were living in the future"

... 2050?

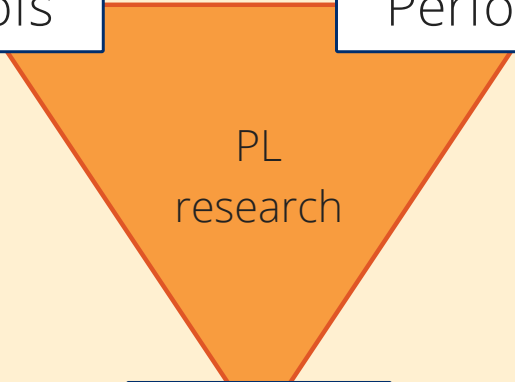


Doing math?  
Join the math dept!



Proofs

Performance



People

PL  
research



Image credit: Alex Aiken

Proofs

Performance

PL  
research

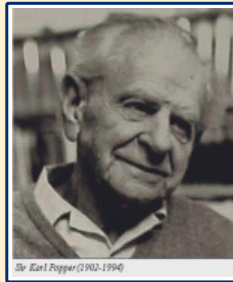
People



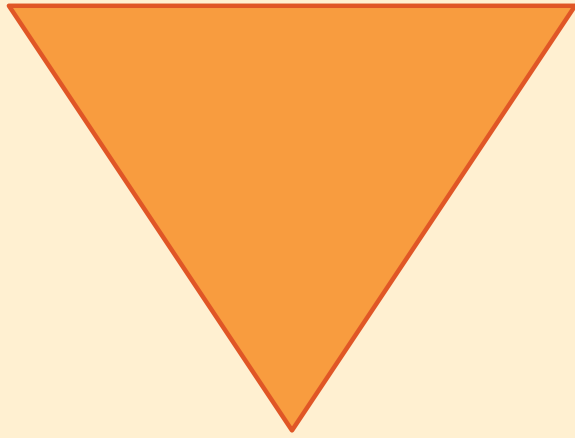
Image credit: Alex Aiken



Some theories are more **testable** than others;  
they take, as it were, greater risks."



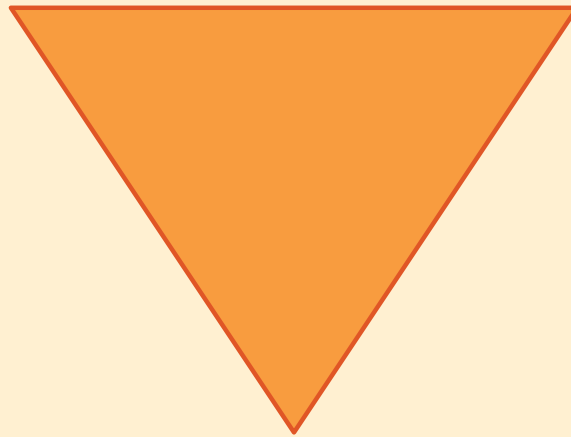
Sir Karl Popper (1902-1994)



Benchmarks for perf,  
for design

Profiling type costs

Gradual soundness



Logic misconceptions

Teaching FM

Privacy-Respecting Telemetry

# Gradual Typing

Untyped



Typed

Why not both?

# Gradual Typing

Untyped



Typed

Why not both?

```
def join(d0,d1,sort,how):  
  ....
```

DataFrame

bool

Left|Right

```
def join(d0:DataFrame,  
        d1:DataFrame,  
        sort:bool,  
        how:Left|Right)  
  -> DataFrame:  
  ....
```

Types where useful, that's all.

Now, what do types mean?

```
def join(d0:DataFrame,  
        d1:DataFrame,  
        sort:bool,  
        how:Left|Right)  
  -> DataFrame:  
  ....
```

```
join("hello", ...)
```

Is **d0** really a data frame?

Now, what do types mean?

```
def join(d0:DataFrame,  
        d1:DataFrame,  
        sort:bool,  
        how:Left|Right)  
  -> DataFrame:  
  ....
```

```
join("hello", ...)
```

Is **d0** really a data frame?

Ideally YES







"The system **lives up to all expectations** that developers have of sound language implementations."





"The system **lives up to all expectations** that developers have of sound language implementations."



"My runtime went from **1 ms to 10 seconds!**"

warning on use trie functions in #lang racket?



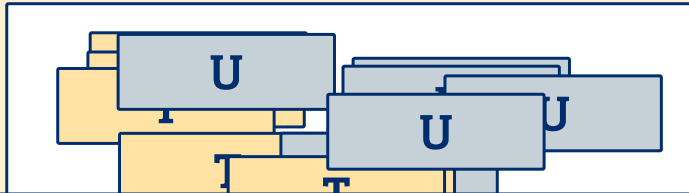
johnbclements  
to Racket Users

This program constructs a trie containing exactly two keys; ea  
access to be 2<sup>62</sup> in the length of the list, so doubling it to 2<sup>64</sup>



Typed Racket

What do **sound types** cost?



Typed Racket

What do **sound types** cost?



1. Start with a program

```
def join(d0,d1,sort,how):  
    ....
```

2. Add full types

```
def join(d0:DataFrame,  
        d1:DataFrame,  
        sort:bool,  
        how:Left|Right)  
    -> DataFrame:  
    ....
```

3. Measure all combos





REP'23: 21 benchmarks, +40k combos

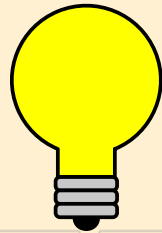
**Table 1: Benchmarks overview: purpose and characteristics**

| Benchmark | Purpose                             | T Init | U Lib | T Lib | Adapt | HOF | Poly | Rec | Mut | Imm | Obj | Cls |
|-----------|-------------------------------------|--------|-------|-------|-------|-----|------|-----|-----|-----|-----|-----|
| sieve     | <i>prime generator</i>              | ○      | ○     | ○     | ●     | ○   | ○    | ●   | ○   | ●   | ○   | ○   |
| forth     | <i>Forth interpreter</i> [51]       | ○      | ○     | ○     | ○     | ○   | ○    | ●   | ○   | ●   | ●   | ●   |
| fsm       | <i>economy simulation</i> [33]      | ○      | ○     | ○     | ○     | ○   | ○    | ○   | ●   | ●   | ○   | ○   |
| fsmoo     | <i>economy simulation</i> [34]      | ○      | ○     | ○     | ○     | ○   | ○    | ○   | ●   | ●   | ●   | ○   |
| mbta      | <i>subway map</i>                   | ●      | ●     | ○     | ○     | ○   | ○    | ○   | ○   | ○   | ●   | ○   |
| morsecode | <i>Morse code trainer</i> [23, 148] | ○      | ○     | ○     | ○     | ○   | ○    | ○   | ●   | ○   | ○   | ○   |
| zombie    | <i>HTDP game</i> [151]              | ○      | ○     | ○     | ●     | ●   | ○    | ●   | ○   | ●   | ○   | ○   |
| zordoz    | <i>bytecode tools</i> [53]          | ○      | ●     | ○     | ●     | ●   | ○    | ●   | ●   | ●   | ○   | ○   |
| dungeon   | <i>maze generator</i>               | ○      | ○     | ○     | ○     | ●   | ●    | ●   | ●   | ●   | ●   | ●   |
| inag      | <i>image tools</i> [161]            | ●      | ●     | ●     | ○     | ○   | ○    | ○   | ●   | ●   | ○   | ○   |

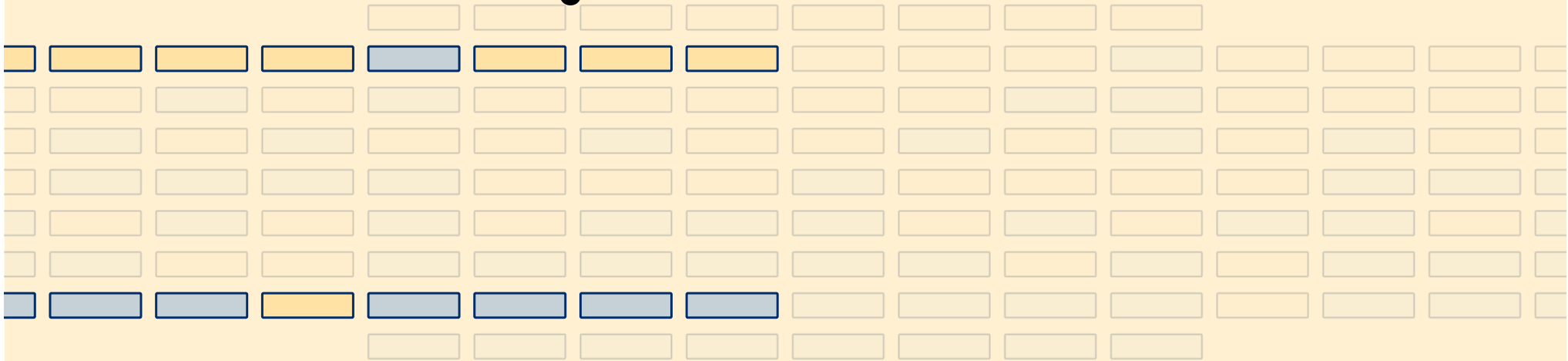


Lots of data!

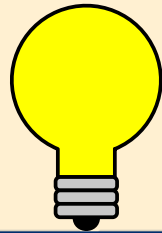
Insights for users?  
for language designers?



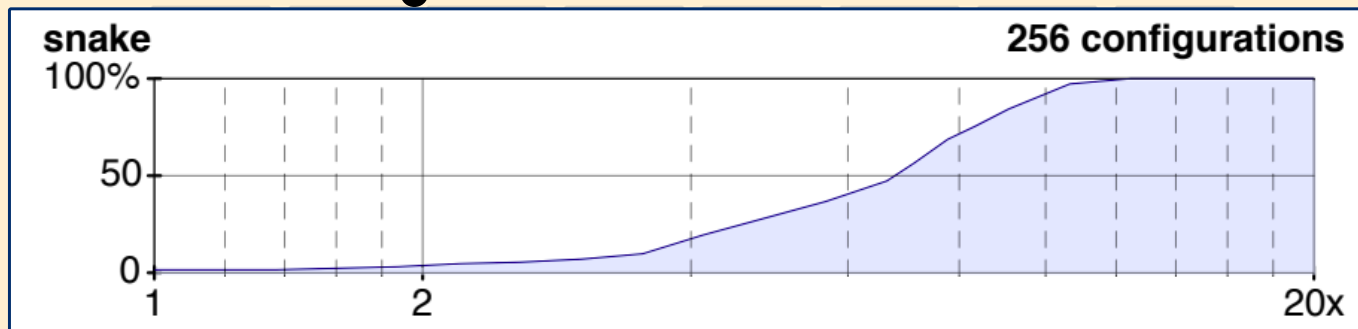
Key: **think like a user**  
too slow = useless





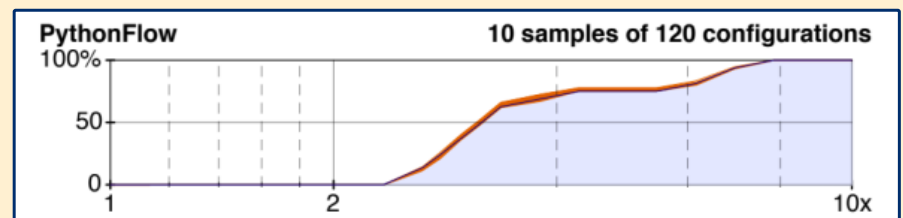
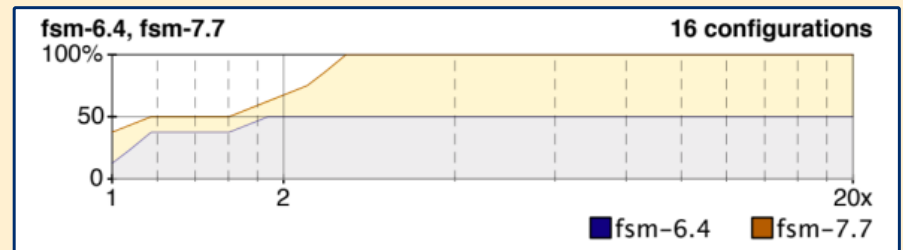
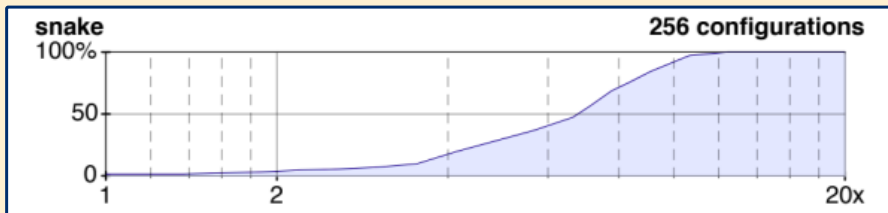


Key: **think like a user**  
too slow = useless



x-axis = "too slow" cutoff vs. untyped code (log scale)  
y-axis = % useful combos

## Scaling further

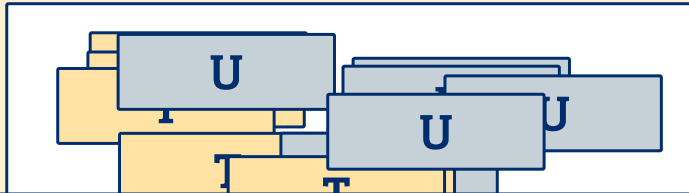




What do **sound types** cost?



Typed Racket

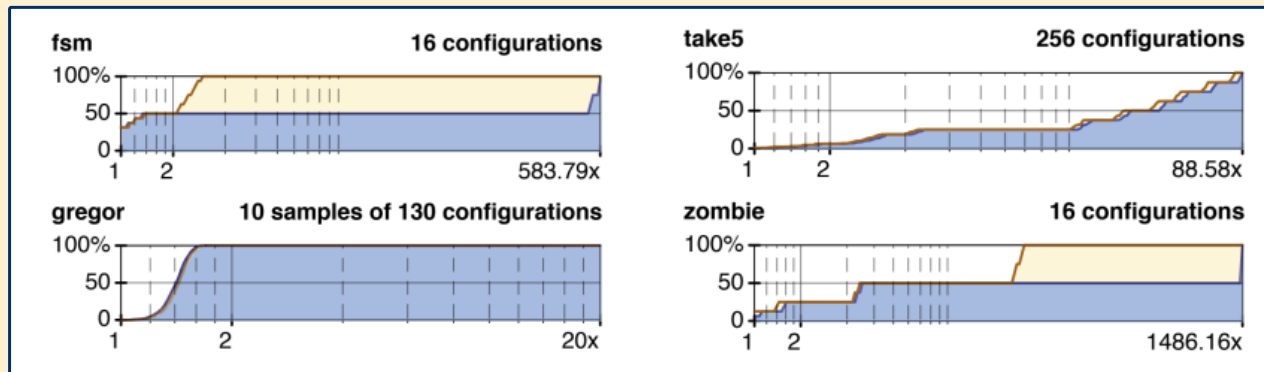


Typed Racket

What do **sound types** cost?

**Too much!**

A modest optimization ... still too slow





Safe and Efficient Gradual Typing

Transient Typechecks are (Almost) Free

Sound Gradual Typing is Nominally Alive and Well



Different behaviors!



Different behaviors!

```
def join(d0:Array[Int]):  
  ....
```

```
join([0,1,2,...])
```

## Different behaviors!

```
def join(d0:Array[Int]):  
  ....
```





```
join([0,1,2,...])
```

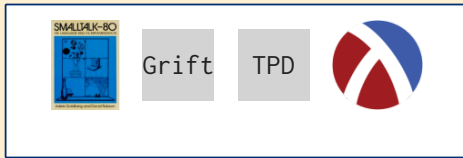
- ✓ every element looks good
- ✓ it's an array
- ✓ I don't care
- ✗ it's untyped data

## Different behaviors!

```
def join(d0:Array[Int]):  
  ....
```

```
join([0,"XXX",...])
```

-  bad element
-  it's an array
-  I don't care
-  it's untyped data





Proofs + People



## Proofs + People



|                     | Guarded | C | F | Transient | A | E |
|---------------------|---------|---|---|-----------|---|---|
| type soundness      | ✓       | ✓ | ✓ | y         | ✓ | ✗ |
| complete monitoring | ✓       | ✓ | ✗ | ✗         | ✗ | ✗ |
| blame soundness     | ✓       | ✓ | ✓ | h         | ✓ | 0 |
| blame completeness  | ✓       | ✓ | ✓ | ✗         | ✓ | ✗ |

# Proofs + People



|                     | Guarded | C | F | Transient | A | E |
|---------------------|---------|---|---|-----------|---|---|
| type soundness      | ✓       | ✓ | ✓ | y         | ✓ | ✗ |
| complete monitoring | ✓       | ✓ | ✗ | ✗         | ✗ | ✗ |
| blame soundness     | ✓       | ✓ | ✓ | h         | ✓ | 0 |
| blame completeness  | ✓       | ✓ | ✓ | ✗         | ✓ | ✗ |

## Question 7

```

1 | var x : Array(String) = ["hi", "bye"];
2 | var y = x;
3 | var z : Array(Number) = y;
4 | z[0] = 42;
5 | var a : Number = z[1];
6 | a

```

Error: line 4 expected String got 42     LE  LU  DE  DU

Error: line 5 expected Number got "bye"   

"bye"



# Proofs + People



|                     | Guarded | C | F | Transient | A | E |
|---------------------|---------|---|---|-----------|---|---|
| type soundness      | ✓       | ✓ | ✓ | y         | ✓ | ✗ |
| complete monitoring | ✓       | ✓ | ✗ | ✗         | ✗ | ✗ |
| blame soundness     | ✓       | ✓ | ✓ | h         | ✓ | 0 |
| blame completeness  | ✓       | ✓ | ✓ | ✗         | ✓ | ✗ |

**Question 7**

```

1 var x : Array(String) = ["hi", "bye"];
2 var y = x;
3 var z : Arra
4 z[0] = 42;
5 var a : Numb
6 a

```

Error: line 4 expected  
Error: line 5 expected "bye"

|   | S.E         | Student     | MTurk       |
|---|-------------|-------------|-------------|
| <b>DEEP</b> →* Error: line 4 expected String got 42       | LE LU DE DU | LE LU DE DU | LE LU DE DU |
| <b>ERASURE</b> →* "bye"                                   | LE LU DE DU | LE LU DE DU | LE LU DE DU |
| <b>SHALLOW</b> →* Error: line 5 expected Number got "bye" | LE LU DE DU | LE LU DE DU | LE LU DE DU |

L = Like D = Dislike E = Expected U = Unexpected

Proofs

Performance

PL  
research

People

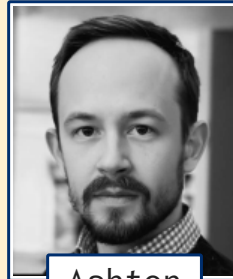
## Research Challenges



Dibri



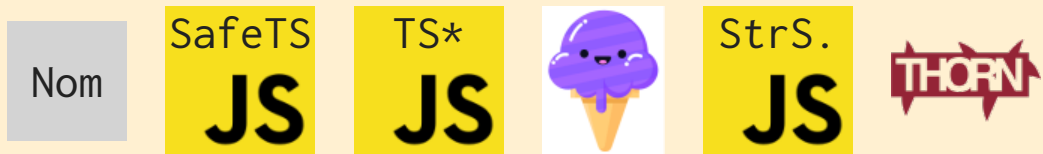
Hanwen



Ashton



Dominic





Same type system??

**NO**

**RC.** Whence Gradual Types?

**RC.** Whence Gradual Types?

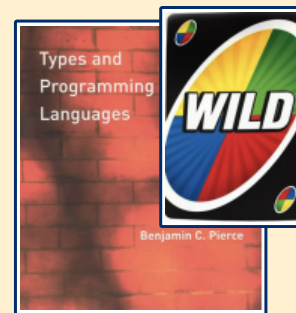


A. think really hard

## RC. Whence Gradual Types?



A. think really hard

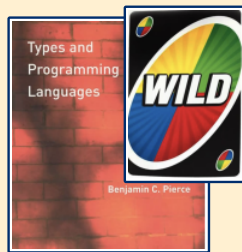


B. be vague





Dibri

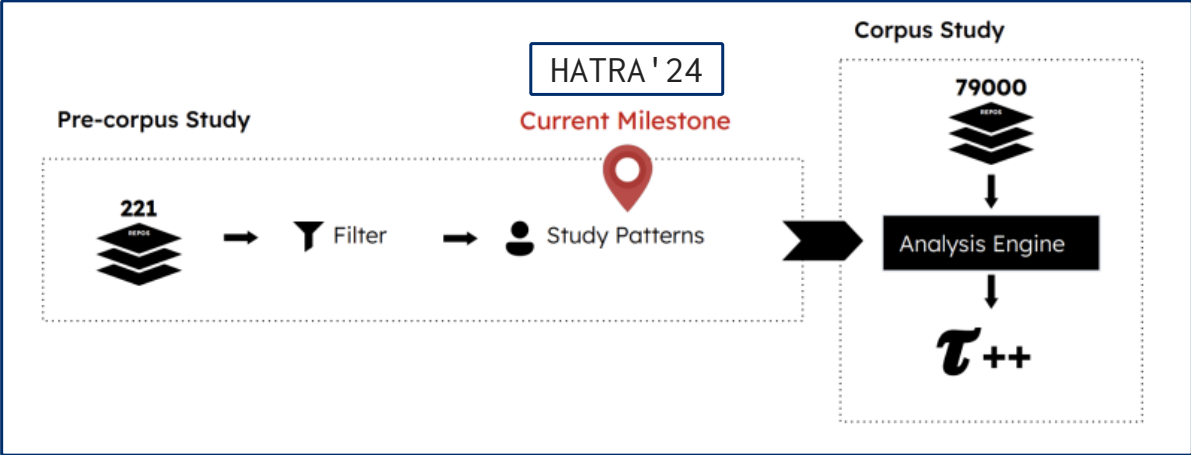


find **actual** type issues

Example pattern: dependent dict

```
def add_tax(item: Dict[Str, Any]) -> float:  
    base = item.get("price", 0) # Any  
    return base + (base * 0.10)
```

**6,000** similar occurrences in 221 sample projects

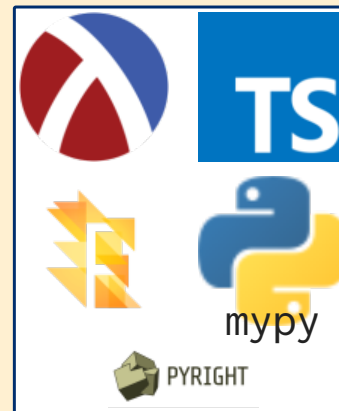




Hanwen



# How to do Type Narrowing?



```
if type(a) is int:  
    return a + 1
```

```
def filter_nums(bs: List[Any]):  
    return sum([b for b in bs if type(b) is int])
```

```
def fst(c : tuple[object, object]):  
    if type(c[0]) is int:  
        return c[0] + 1
```

```
if node.parent is not None:  
    total += node.parent.wins + node.parent.losses
```



```
if type(a) is int:  
    return a + 1
```

```
def filter_nums(bs: List[Any]):  
    return sum([b for b in bs if type(b) is int])
```

```
def fst(c : tuple[object, object]):  
    if type(c[0]) is int:  
        return c[0] + 1
```

```
if node.parent is not None:  
    total += node.parent.wins + node.parent.losses
```

```
if type(a) is int:  
    return a + 1
```

```
def filter_nums(bs: List[Any]):  
    return sum([b for b in bs if type(b) is int])
```

```
def fst(c : tuple[object, object]):  
    if type(c[0]) is int:  
        return c[0] + 1
```

```
if node.parent is not None:  
    total += node.parent.wins + node.parent.losses
```

```
if type(a) is int:  
    return a + 1
```

```
def filter_nums(bs: List[Any]):  
    return sum([b for b in bs if type(b) is int])
```

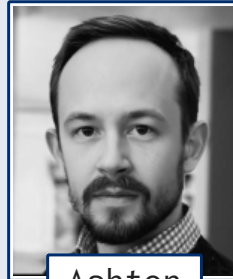
```
def fst(c : tuple[object, object]):  
    if type(c[0]) is int:  
        return c[0] + 1
```

```
if node.parent is not None:  
    total += node.parent.wins + node.parent.losses
```

## The Benchmark

According to the [extracted key features](#), the following benchmark items are proposed.

| Benchmark            | Description  |
|----------------------|--|
| positive             | refine when condition is true                            |
| negative             | refine when condition is false                           |
| alias                | track test results assigned to variables                 |
| connectives          | handle logic connectives                                 |
| nesting_condition    | nested conditionals with nesting happening in condition  |
| nesting_body         | nested conditionals with nesting happening in body       |
| custom_predicates    | allow programmers define their own predicates            |
| predicate_2way       | custom predicates refines both positively and negatively |
| predicate_strict     | perform strict type checks on custom predicates          |
| predicate_multi_args | predicates can have more than one arguments              |
| object_properties    | refine types of properties of objects                    |
| tuple_whole          | refine types of the whole tuple                          |
| tuple_elements       | refine types of tuple elements                           |
| subtyping            | refine supertypes to subtypes                            |
| subtyping_structural | refine structural subtyping                              |



Ashton





Untyped      ➤ Typed

Simply T.      ➤ Dependently T.

Host Lang.      ➤ DSL



Untyped      ➤ Typed

Simply T.      ➤ Dependently T.

Host Lang.      ➤ DSL

**RC.** How to bridge?

Metaprogramming!



ECOOP '24

## Type Tailoring

**Ashton Wiersdorf** ✉ 🌐  
University of Utah, Salt Lake City, UT, USA

**Stephen Chang** ✉ 🌐  
University of Massachusetts Boston, MA, USA

**Matthias Felleisen** ✉ 🌐  
Northeastern University, Boston, MA, USA

**Ben Greenman** ✉ 🌐  
University of Utah, Salt Lake City, UT, USA

---

### Abstract

Type systems evolve too slowly to keep up with the quick evolution of libraries – especially libraries that introduce abstractions. Type tailoring offers a lightweight solution by equipping the core language with an API for modifying the elaboration of surface code into the internal language of

```
<p>  
  <%= link "Register", to: ~p"/users/register" %>  
  <%= link "Log in", to: ~p"/users/login" %>  
</p>
```

Elixir

#### Without Tailoring

Possible 404 at runtime

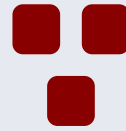
#### With Tailoring

Tailoring error:  
no route path matches /users/login

Chorex

Type tailoring for Elixir choreographies

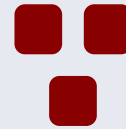
Chor.



# Chorex

Type tailoring for Elixir choreographies

Chor.



Discrete log, zero-knowledge

```
defmodule Zkp.ZkpChor do
  import Chorex

  defchor [Prover, Verifier] do
    # ...

    def do_round(Verifier.({p, g, y}), Prover.({p, g, x})) do
      with Prover.(r) <- Prover.(Enum.random(2..p)) do
        Prover.(:crypto.mod_pow(g, r, p)) ~> Verifier.(c)

        with Verifier.(choice) <- Verifier.challenge_type() do
          if Verifier.(choice == :r) do
            Verifier[L] ~> Prover
            Prover.(r) ~> Verifier.(r)
            Verifier.verify_round(c, r, p, g)
          else
            Verifier[R] ~> Prover
            Prover.(rem(:crypto.bytes_to_integer(x) + r, p - 1)) ~> Verifier.(xr_modp)
          end
        end
      end
    end
  end
end
```



Dominic

**RC.** How to debug designs?





**Forge** = a solver-aided modeling language





**Forge** = a solver-aided modeling language

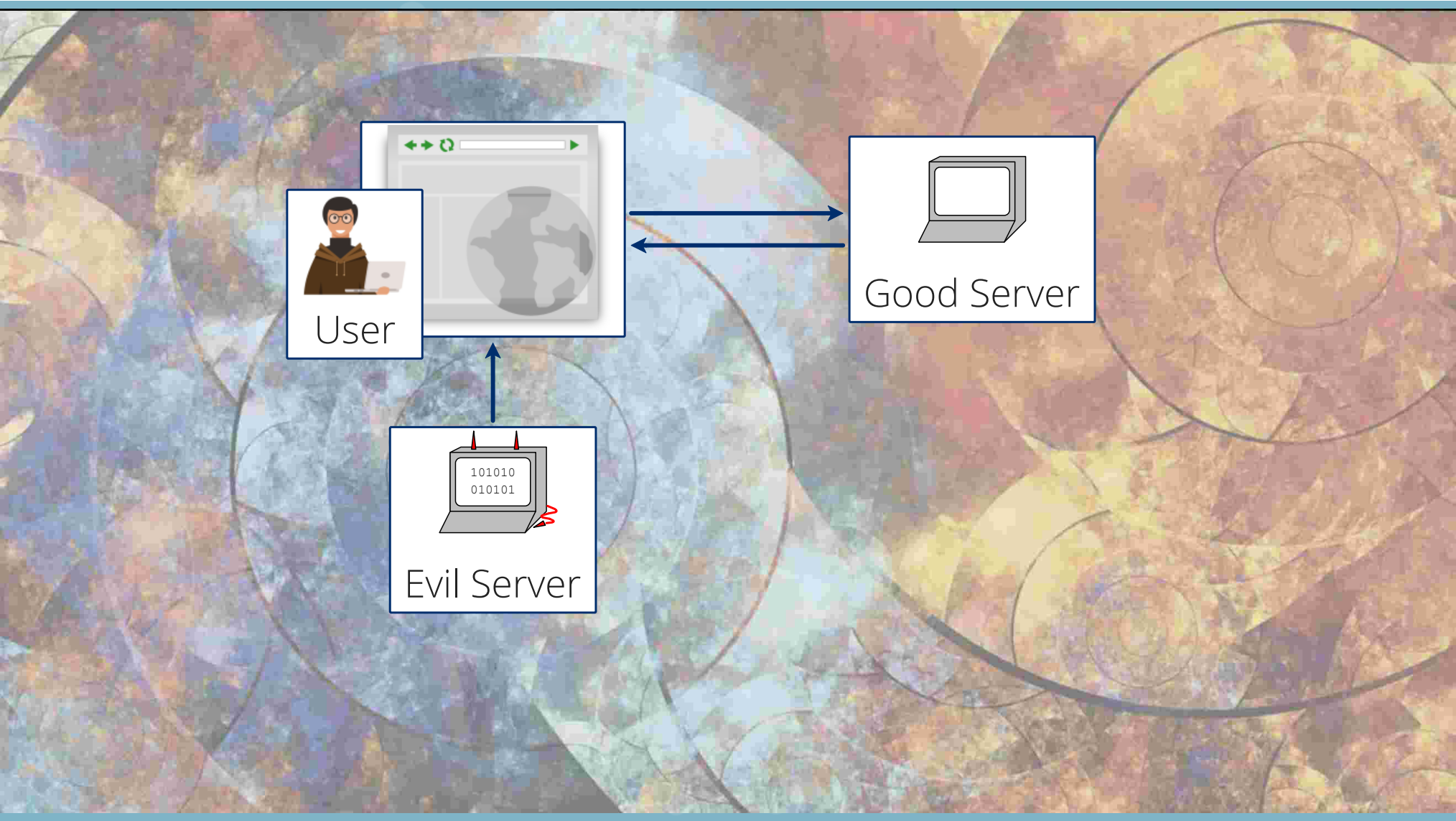


inspired by Alloy





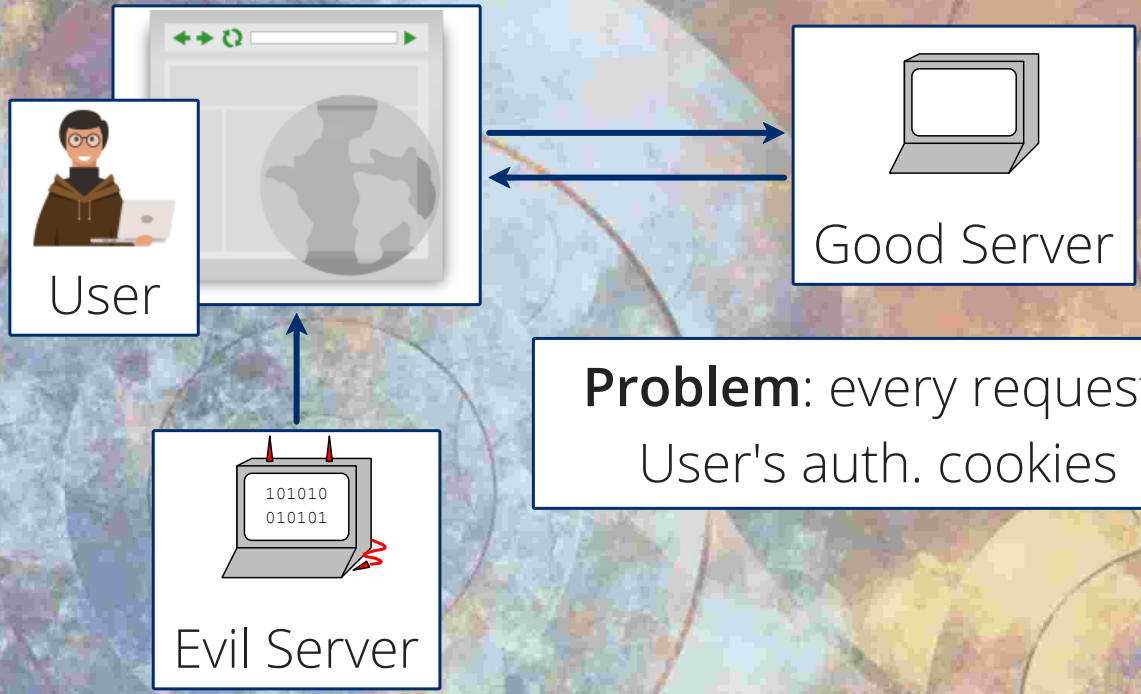
# Cross-Site Request Forgery



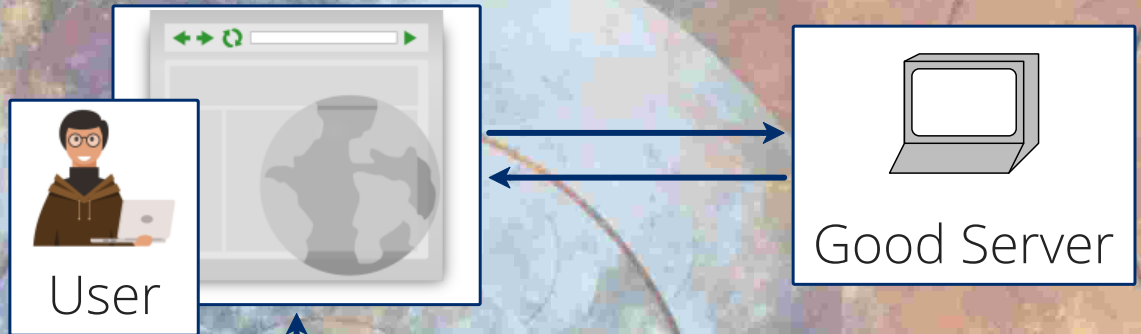
User

Evil Server

Good Server



**Problem:** every request carries User's auth. cookies



**Problem:** every request carries User's auth. cookies

 **Idea:** add origin to requests, validate at Good Server

```
abstract sig EndPoint {}
```

```
sig Client
```

```
  extends EndPoint {}
```

```
abstract sig EndPoint {}
```

```
sig Client
```

```
  extends EndPoint {}
```

```
sig Server
```

```
  extends EndPoint {
```

```
    causes: set HTTPEvent
```

```
  }
```

```
abstract sig EndPoint {}

sig Client
  extends EndPoint {}

sig Server
  extends EndPoint {
    causes: set HTTPEvent
  }
```

```
abstract sig HTTPEvent {
  from : one EndPoint,
  to : one EndPoint,
  origin : one EndPoint
}

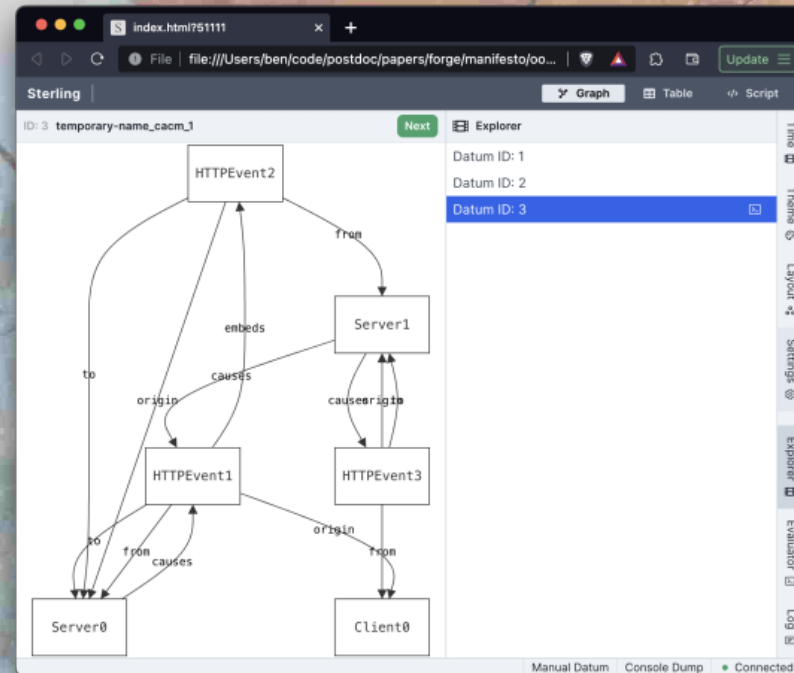
// Request, Response, Redirect
// extends HTTPEvent
```

# Bounded Exploration

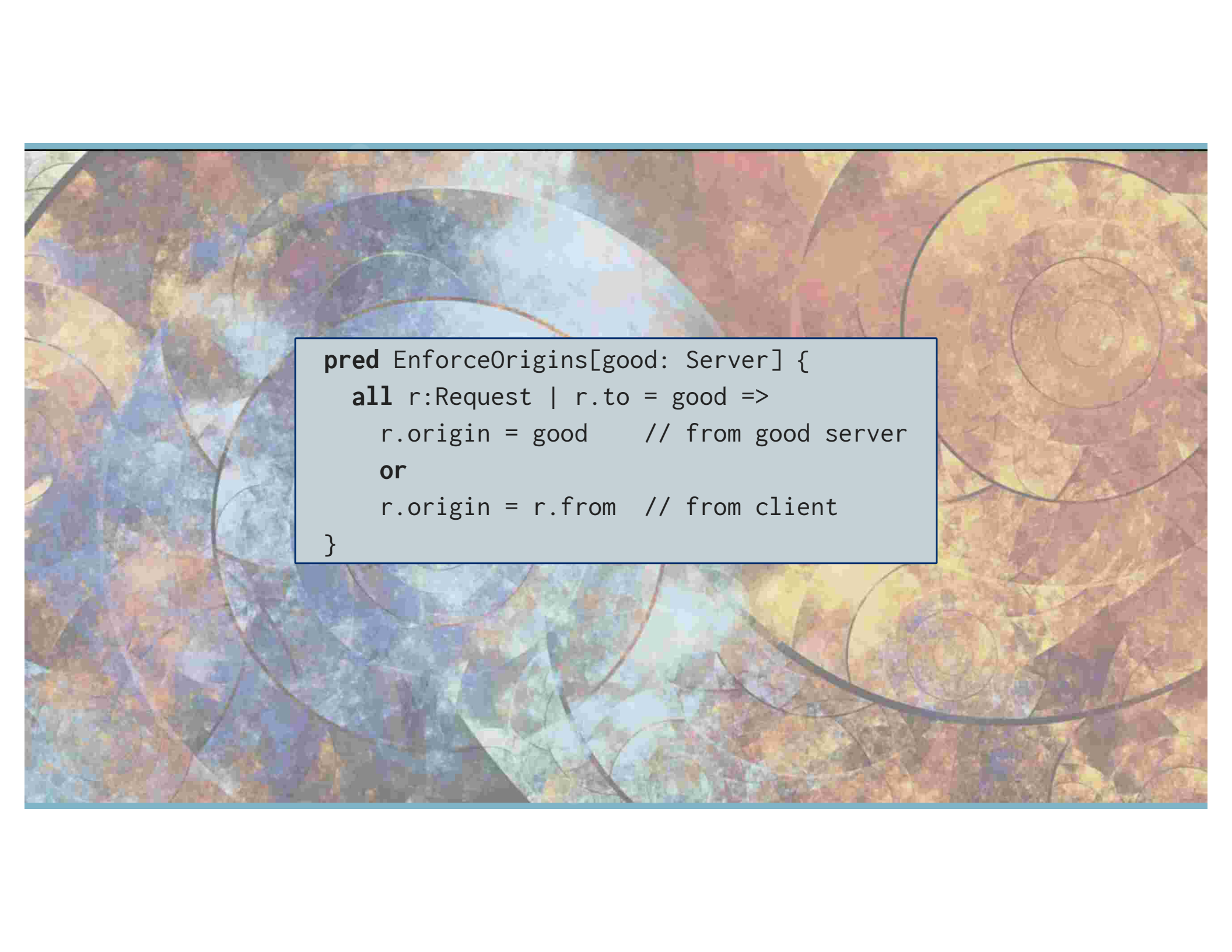
```
cacm.frg - DrRacket
cacm.frg (define ...) Run Stop

1 #lang forge
2 abstract sig EndPoint {}
3
4 sig Server extends EndPoint {
5   causes: set HTTPEvent
6 }
7
8
9 sig Client extends EndPoint {}
10
11 abstract sig HTTPEvent {
12   from : one EndPoint,
13   to : one EndPoint,
14   origin : one EndPoint
15 }
16
17 sig Request extends HTTPEvent {
18   response: lone Response
19 }
20
21 sig Response extends HTTPEvent {
22   embeds: set Request
23 }
24
25 sig Redirect extends Response {}
26
27 run {} for exactly 2 Server, exactly 1 Client
28
```

Determine language from source custom 15:1 585.99 MB





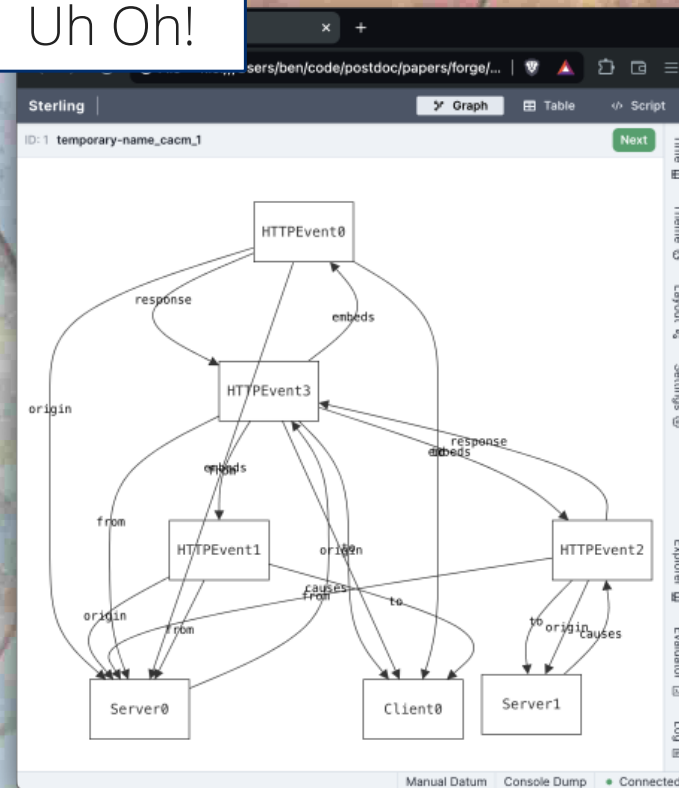


```
pred EnforceOrigins[good: Server] {  
  all r:Request | r.to = good =>  
    r.origin = good    // from good server  
  or  
    r.origin = r.from  // from client  
}
```

```
run {  
  // can we find (hope not)  
  some good, bad: Server {  
    EnforceOrigins[good]  
    // ...  
  }  
} for exactly 2 Server,  
  exactly 1 Client,  
  5 HTTPEvent
```

```
run {
  // can we find (hope not)
  some good, bad: Server {
    EnforceOrigins[good]
    // ...
  }
} for exactly 2 Server,
  exactly 1 Client,
  5 HTTPEvent
```

Uh Oh!



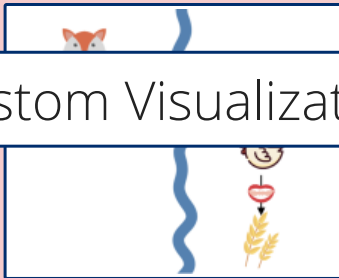


Quickly found a bug!





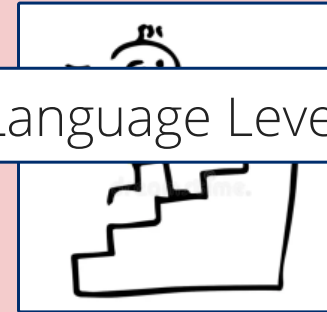
Custom Visualization

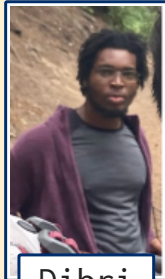


Unit Testing



Language Levels

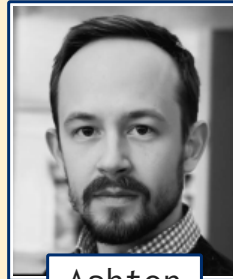




Dibri



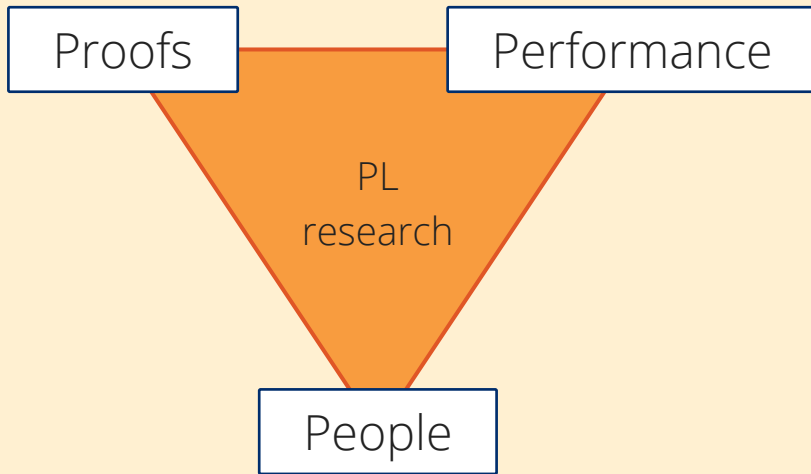
Hanwen



Ashton



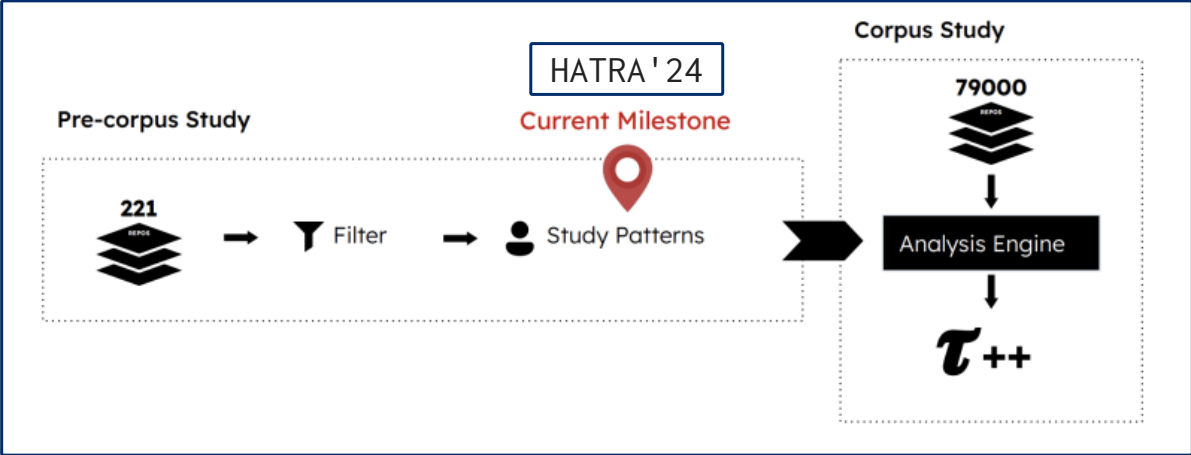
Dominic



**RC.** Types for untyped code

**RC.** Adding layers to languages

**RC.** Debugging for designs





## The Benchmark

According to the [extracted key features](#), the following benchmark items are proposed.

| Benchmark            | Description  |
|----------------------|--|
| positive             | refine when condition is true                            |
| negative             | refine when condition is false                           |
| alias                | track test results assigned to variables                 |
| connectives          | handle logic connectives                                 |
| nesting_condition    | nested conditionals with nesting happening in condition  |
| nesting_body         | nested conditionals with nesting happening in body       |
| custom_predicates    | allow programmers define their own predicates            |
| predicate_2way       | custom predicates refines both positively and negatively |
| predicate_strict     | perform strict type checks on custom predicates          |
| predicate_multi_args | predicates can have more than one arguments              |

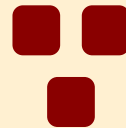
```
if node.parent is not None:
    total += node.parent.wins + node.parent.losses
```

|                      |                               |
|----------------------|-------------------------------|
| subtyping            | refine supertypes to subtypes |
| subtyping_structural | refine structural subtyping   |

Chorex

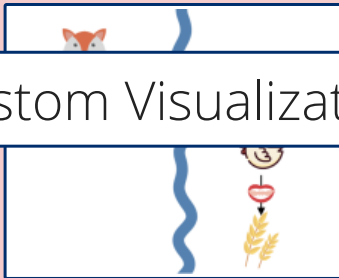
Type tailoring for Elixir choreographies

Chor.





Custom Visualization



Unit Testing



Language Levels



