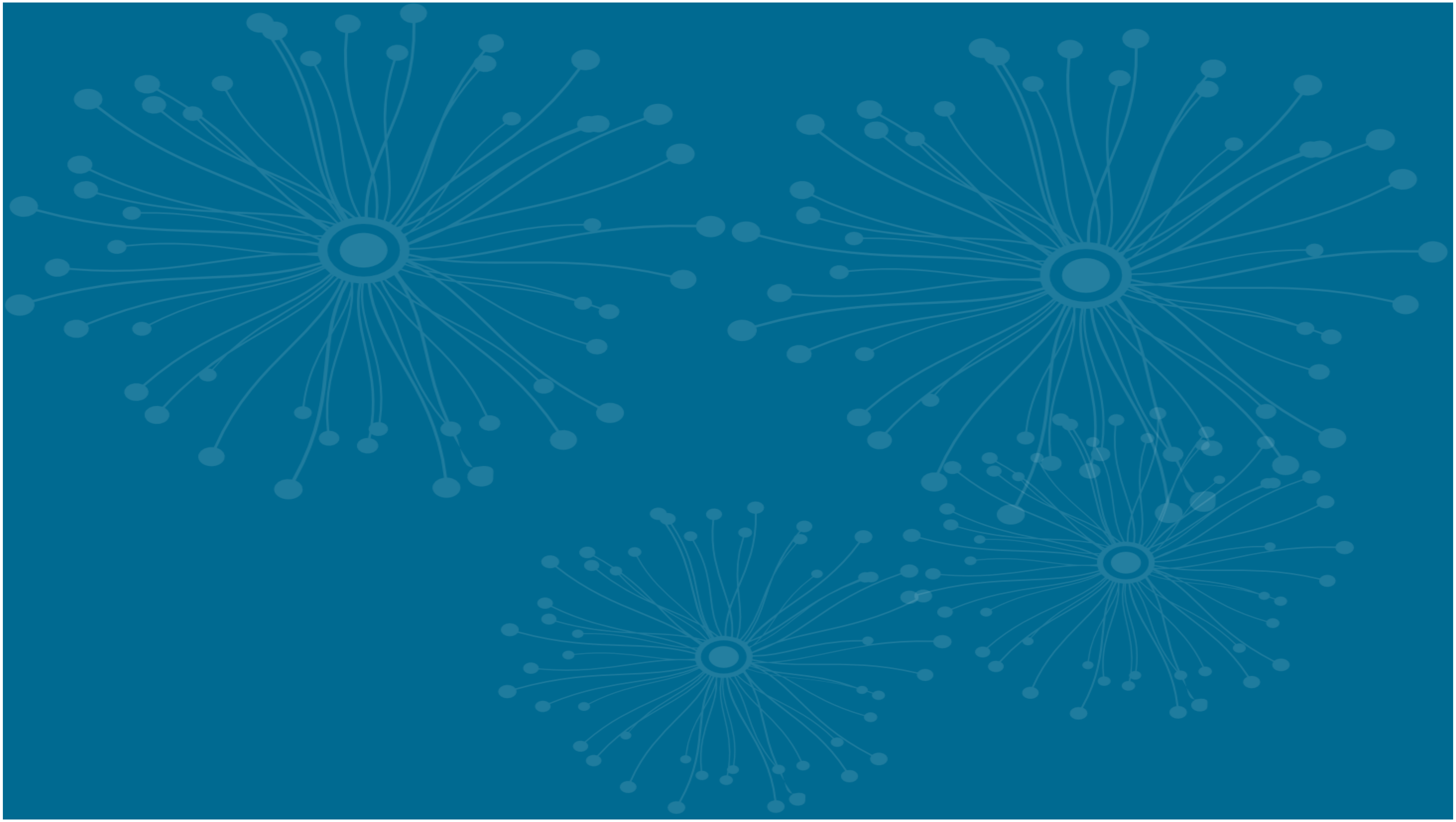


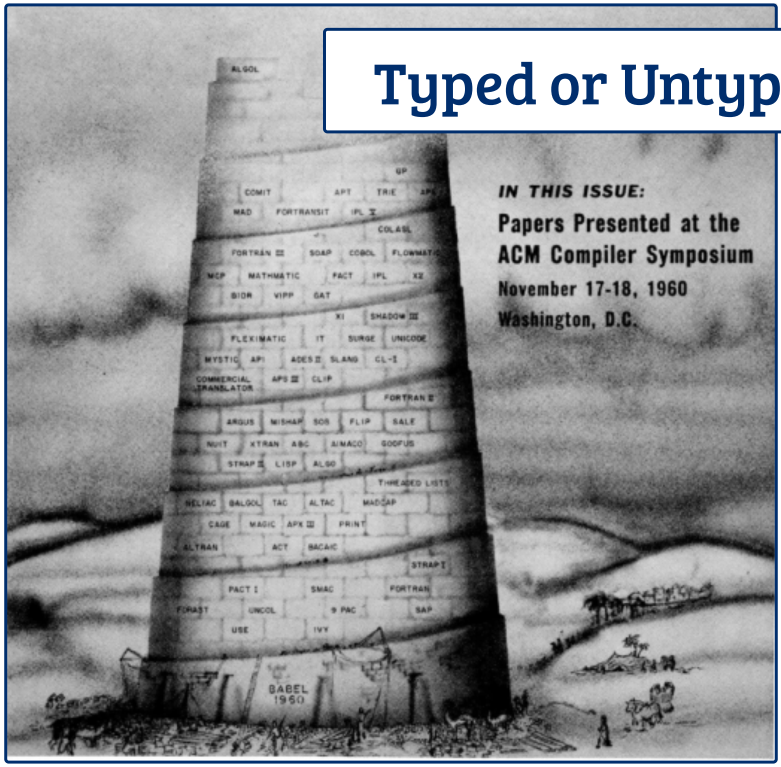
Mixing Typed and Untyped Code

A Tale of Proofs, Performance, and People

Ben Greenman
Spring 2022

BROWN

Typed or Untyped?



IN THIS ISSUE:

Papers Presented at the ACM Compiler Symposium

November 17-18, 1960

Washington, D.C.

Typed or Untyped?

Typed or Untyped?

Java is typed

(statically typed)

```
HashMap<String, Integer> m =
  new HashMap<>();
```

JavaScript is untyped

(dynamically typed)

```
var m = {}
```

Typed or Untyped?

Java is **typed**
(statically typed)

```
HashMap<String, Integer> m =
  new HashMap<>();
```

With types, languages can:

+ Prevent classes of bugs

+ Support tools

JavaScript is **untyped**
(dynamically typed)

```
var m = {}
```

Typed or Untyped?

Java is **typed**
(statically typed)

```
HashMap<String, Integer> m =
  new HashMap<>();
```

With types, languages can:

+ Prevent classes of bugs

+ Support tools

JavaScript is **untyped**
(dynamically typed)

```
var m = {}
```

Without types, programmers can:

+ Focus on the code

+ Build flexible systems

Typed or Untyped?

Java is **typed**
(statically typed)

```
HashMap<String, Integer> m =
  new HashMap<>();
```

With types, languages can:

+ Prevent classes of bugs

+ Support tools

JavaScript is **untyped**
(dynamically typed)

```
var m = {}
```

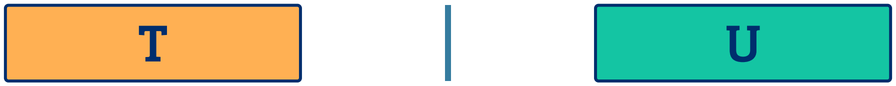
Without types, programmers can:

+ Focus on the code

+ Build flexible systems

Either way, **long-term implications** for development and maintenance

Typed or Untyped



T

U

Strong support for both sides

Typed or Untyped

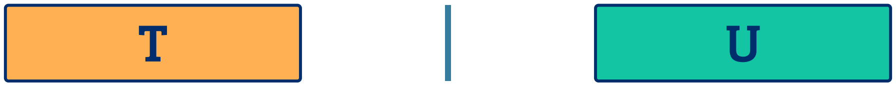
T | **U**

Strong support for both sides

"The advantages of typed PLs are obvious"

Lamport & Paulson, TOPLAS 1999

Typed or Untyped



T

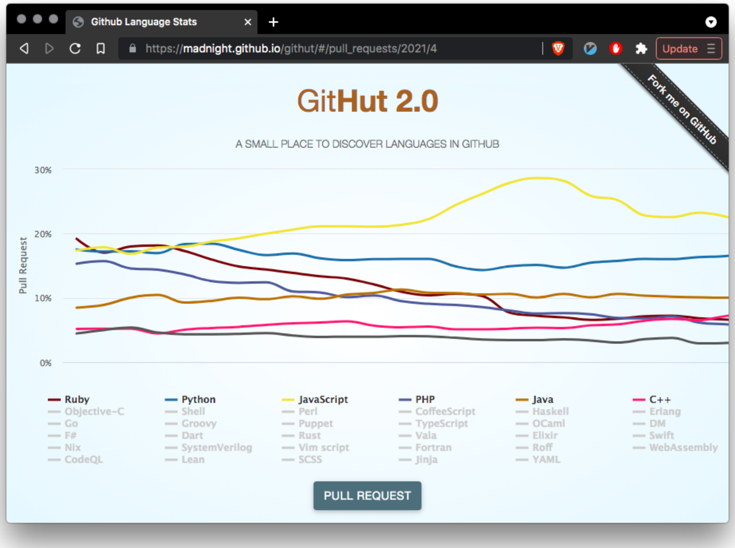
U

Strong support for both sides

Typed or Untyped

T | U

Strong support for both sides


GitHut 2.0

A SMALL PLACE TO DISCOVER LANGUAGES IN GITHUB

s://madnight.github.io/githut/#/pull_requests/2021/4

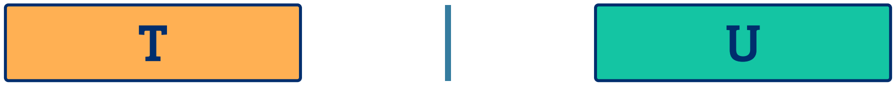
Untyped PLs dominate on GitHub

1. Ruby
2. Python
3. JavaScript
4. PHP
5. Java

✗ Typed or Untyped

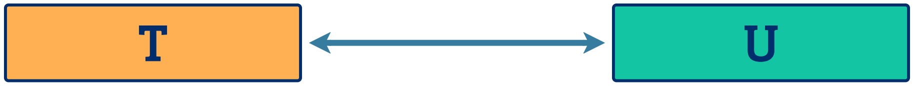
T | U

✗ Typed or Untyped



✓ Typed AND Untyped

Gradual Typing



Gradual Typing

T

U

Key Motivation: **improve stable code** with types

Gradual Typing

T

U

Key Motivation: **improve stable code** with types

```
function parse_lfd_chain(bv, pos, order, max_depth):
  ....
  tag_count = bv_ref(bv, pos, order)
  next_offset = pos + 2 + (* tag_count 12)
  next_pos = bv_ref(bv, next_offset, order)
  pts = parse_tags(tag_count)
  if next_pos == 0:
    return pts
  else:
    return pts ++ parse_lfd_chain(bv, next_pos, order, max_depth - 1)
```

Gradual Typing

Key Motivation: **improve stable code** with types

```
function parse_lfd_chain(bv, pos, order, max_depth):
```

```
function parse_lfd_chain(bv:Bytes, pos:Natural, order:Symbol, max_depth:Natural)
   -> List[PTs]:
  ....
  tag_count = bv_ref(bv, pos, order)
  next_offset = pos + 2 + (* tag_count 12)
  next_pos = bv_ref(bv, next_offset, order)
  pts = parse_tags(tag_count)
  if next_pos == 0:
    return pts
  else:
    return pts ++ parse_lfd_chain(bv, next_pos, order, max_depth - 1)
```

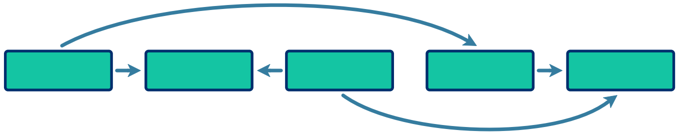
Document the parameters,
benefit from type checks

Gradual Typing

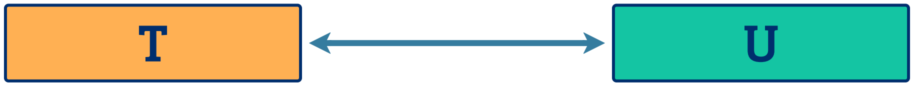
Key Motivation: **improve stable code** with types

Gradual Typing

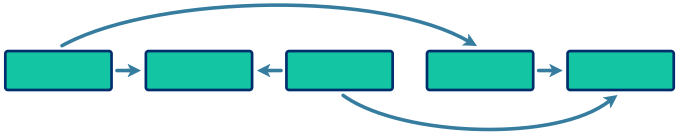
Key Motivation: **improve stable code** with types



Gradual Typing

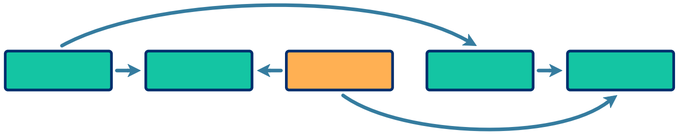


Key Motivation: **improve stable code** with types



Add types to **one** component,
leave the others **unchanged**

➤



Gradual Typing

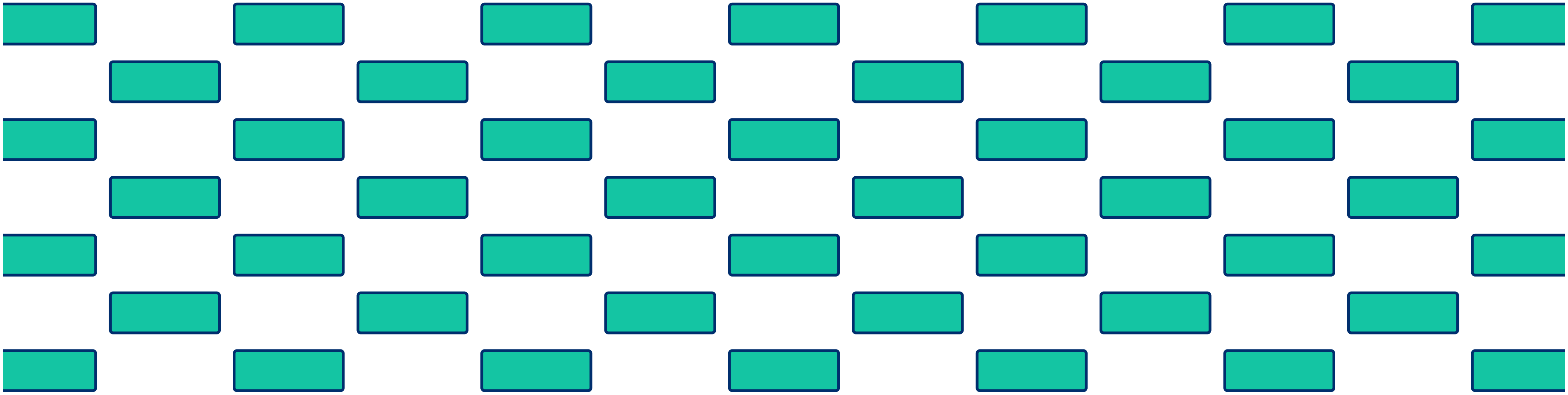
T

U

Key Motivation: **improve stable code** with types

Gradual Typing

Key Motivation: **improve stable code** with types

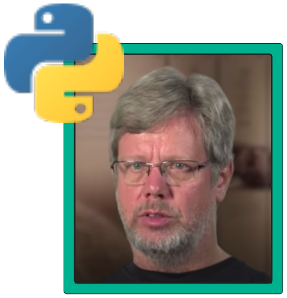


Gradual Typing

T

U

Key Motivation: **improve stable code** with types



"For **really large codebases**, static languages* have their uses"

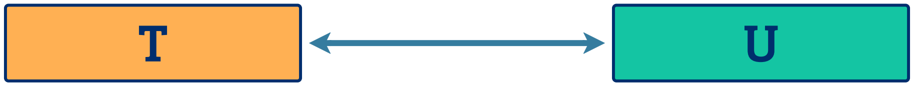
* (despite all their visual overhead and compilation cycles and build tools)

Gradual Typing

T

U

Gradual Typing

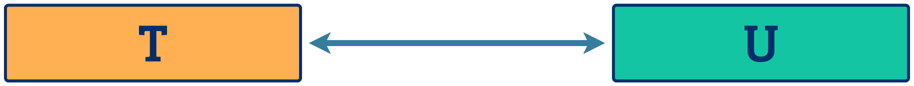


T

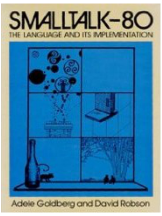
U

Active space!

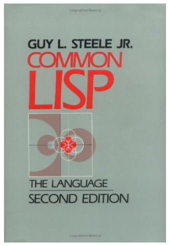
Gradual Typing



Active space!


SMALLTALK-80
THE LANGUAGE AND ITS IMPLEMENTATION
Adele Goldberg and David Robson

StrongTalk
1993



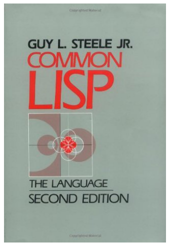
Common Lisp
<1990

Gradual Typing

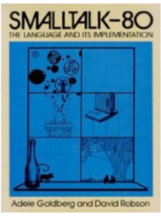
T

U

Active space!



Common Lisp
<1990

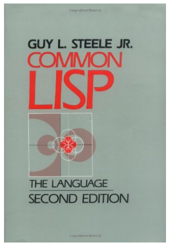

SMALLTALK-80
THE LANGUAGE AND ITS IMPLEMENTATION
Adele Goldberg and David Robson

StrongTalk
1993

Gradual Typing
2006

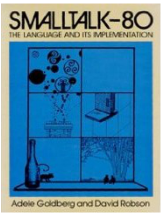
Gradual Typing

Active space!



Common Lisp

<1990

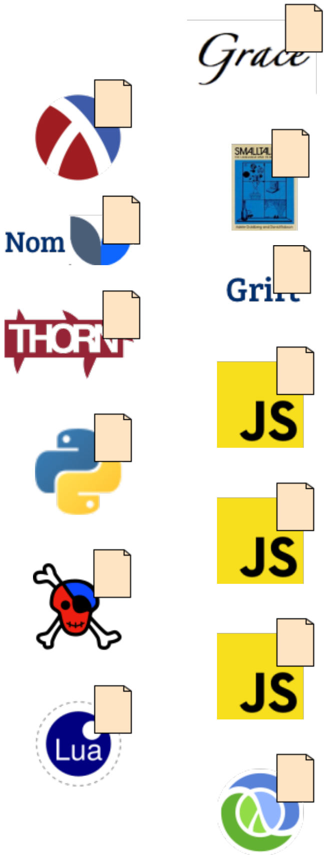

SMALLTALK-80
THE LANGUAGE AND ITS IMPLEMENTATION
Adele Goldberg and David Robson

StrongTalk

1993

Gradual Typing

2006



+ Many Research PLs

Gradual Typing

Active space!

Major companies involved

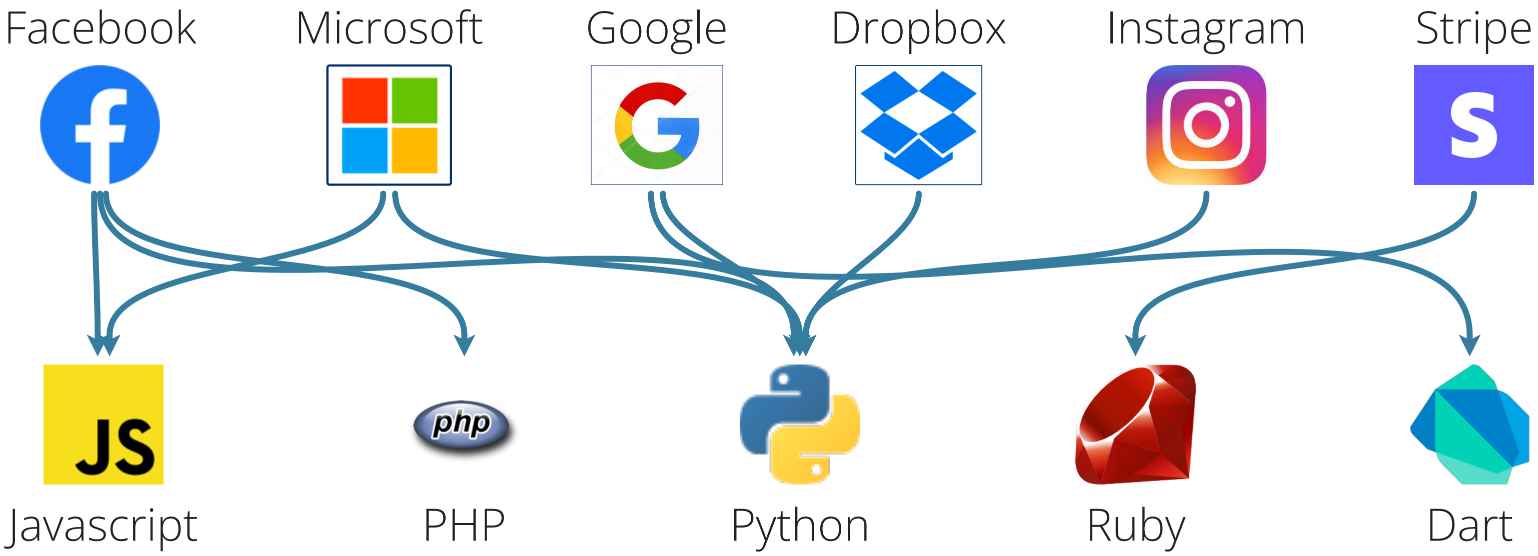
Gradual Typing

T

U

Active space!

Major companies involved



Facebook

Microsoft

Google

Dropbox

Instagram

Stripe

Javascript

PHP

Python

Ruby

Dart

Gradual Typing

Active space!

Major companies involved

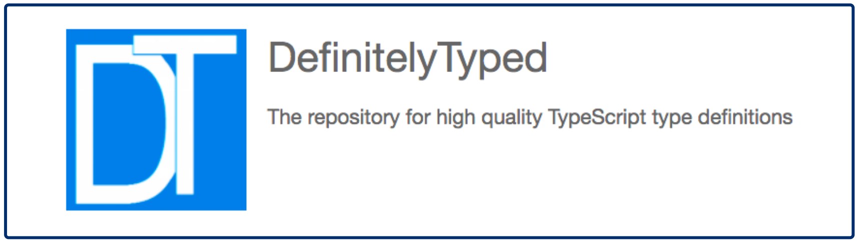
Growing community interest

Gradual Typing

Active space!

Major companies involved

Growing community interest



DefinitelyTyped

The repository for high quality TypeScript type definitions

+ **8k** interfaces

+ **5k** contributors

+ **1 million** clients

Gradual Typing

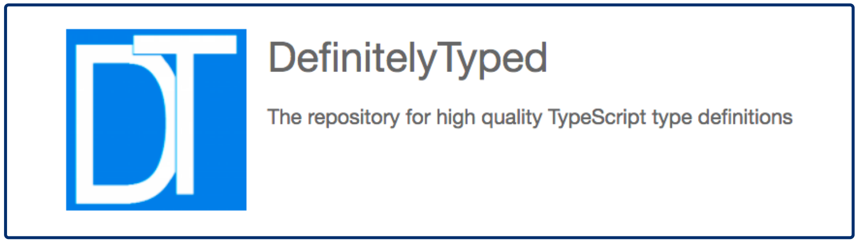
T

U

Active space!

Major companies involved

Growing community interest



DefinitelyTyped

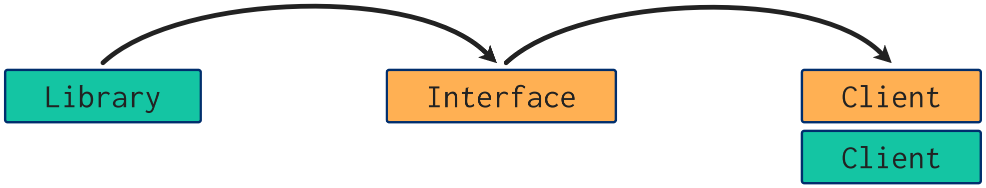
The repository for high quality TypeScript type definitions

+ **8k** interfaces

+ **5k** contributors

+ **1 million** clients

Common case: **new types** for **old libraries**



Library

Interface

Client

Client

Gradual Typing

So what's the **problem**?

Gradual Typing

T

U

So what's the **problem**?

Lots of Languages, but also **Lots of Variety**

Example 1

Typed Function

```
function add1(n : Num)
  n + 1
```

Example 1

Typed Function

```
function add1(n : Num)
  n + 1
```

Q. Is **n** really a number?

Example 1

Typed Function

```
function add1(n : Num)
  n + 1
```

Untyped Caller

```
add1("A")
```

Q. Is **n** really a number?

Example 1

Typed Function

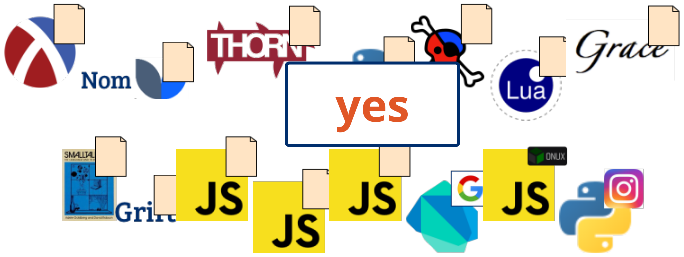
```
function add1(n : Num)
  n + 1
```

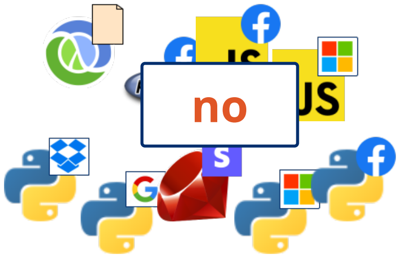
Untyped Caller

```
add1("A")
```

Q. Is **n** really a number?

Some say **yes**, others say **no**





Example 2

Example 2

Untyped Array

```
arr = ["A", 3]
```

Typed Client

```
nums : Array(Num) = arr
nums[0]
```

Example 2

Untyped Array

```
arr = ["A", 3]
```

Typed Client

```
nums : Array(Num) = arr
nums[0]
```

Q. Is **arr** an array of numbers?

Example 2

Untyped Array

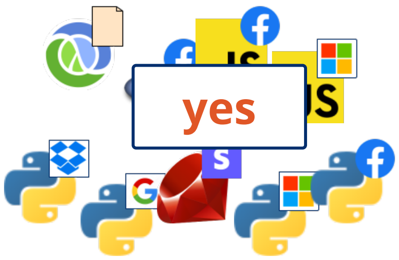
```
arr = ["A", 3]
```

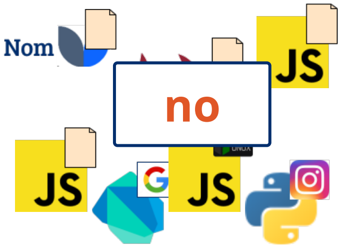
Typed Client

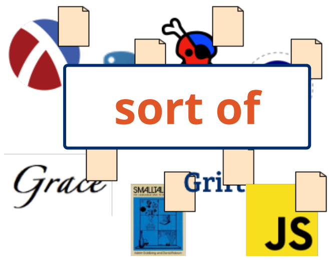
```
nums : Array(Num) = arr
nums[0]
```

Q. Is **arr** an array of numbers?

Three common answers: **yes**, **no**, and **sort of**







What Should Types Mean?

No consensus on basic questions!

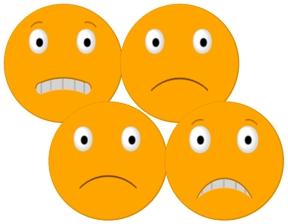
`Num` `Array(Num)`

What Should Types Mean?

No consensus on basic questions!

`Num` `Array(Num)`

Q. Did anyone **ask** programmers?

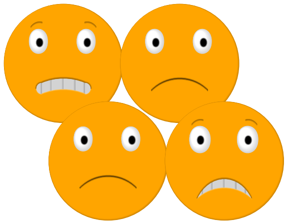


What Should Types Mean?

No consensus on basic questions!

`Num` `Array(Num)`

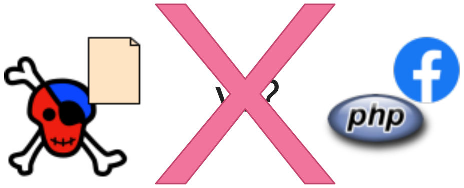
Q. Did anyone **ask** programmers?



Challenge: How to compare languages?

Challenge: How to compare languages?

Challenge: How to compare languages?



Challenge: How to compare languages?

Don't! Compare **semantics** instead.

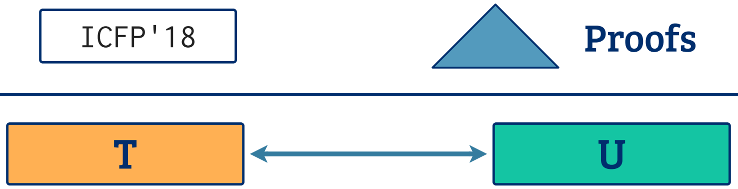
ICFP'18

Proofs

T ↔ **U**

Challenge: How to compare languages?

Don't! Compare **semantics** instead.


ICFP'18

Proofs

Guarded

Types enforce behaviors

Transient

Types enforce top-level shapes

Erasure

Types enforce nothing

Study: Behavior of Gradual Types

DLS'18

People

A **method** to compare semantics

Study: Behavior of Gradual Types

DLS'18

People

A **method** to compare semantics

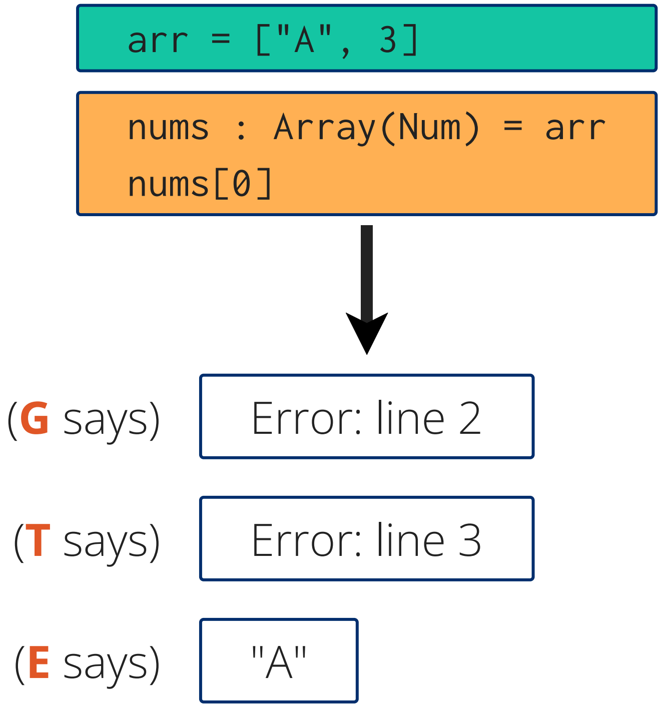
```
arr = ["A", 3]
```

```
nums : Array(Num) = arr
nums[0]
```

Study: Behavior of Gradual Types

DLS'18 People

A **method** to compare semantics



```
arr = ["A", 3]
```

```
nums : Array(Num) = arr
nums[0]
```

(**G** says) Error: line 2

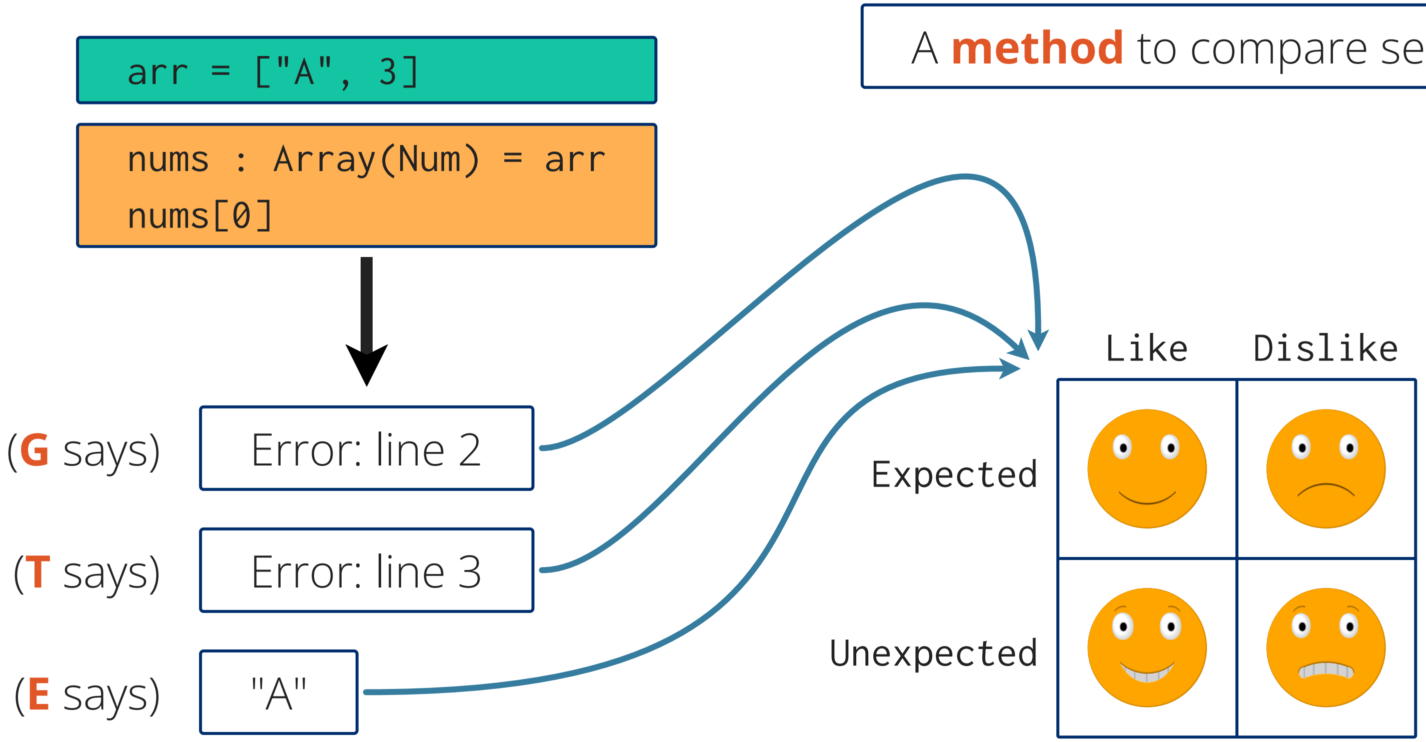
(**T** says) Error: line 3

(**E** says) "A"

Study: Behavior of Gradual Types

DLS'18 People

A **method** to compare semantics



```
arr = ["A", 3]
```

```
nums : Array(Num) = arr
nums[0]
```

(**G** says) Error: line 2

(**T** says) Error: line 3

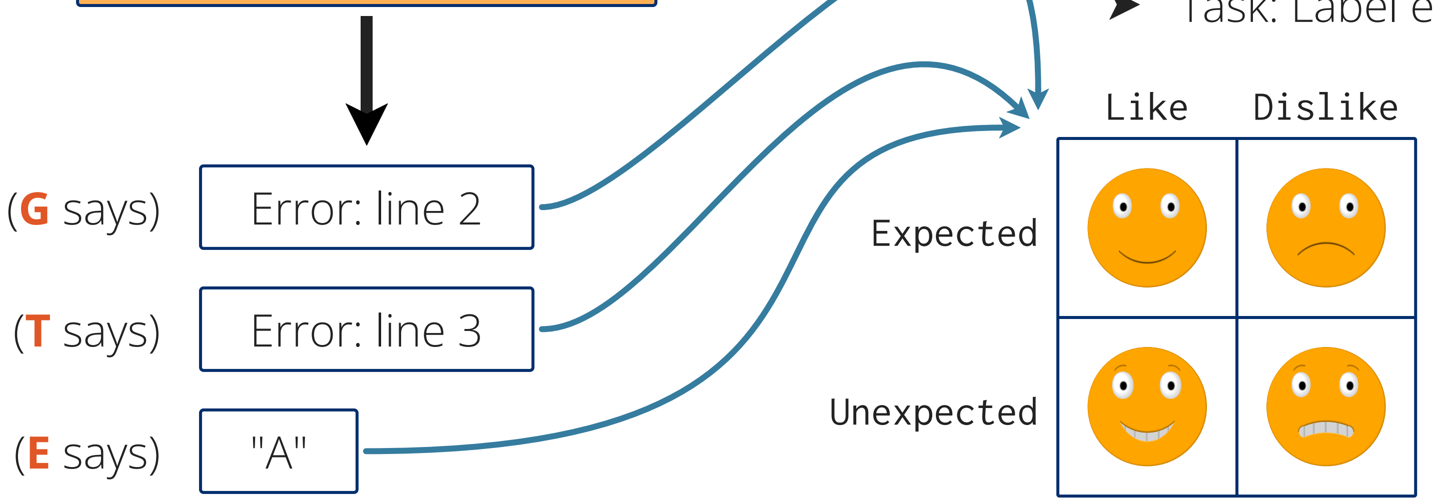
(**E** says) "A"

Study: Behavior of Gradual Types

DLS'18 People

```
arr = ["A", 3]
```

```
nums : Array(Num) = arr
nums[0]
```



A **method** to compare semantics

- One program
- Distinct results
- Task: Label each result

Study: Behavior of Gradual Types

DLS'18 ▲ People

Company Name
Engineers

Students

amazon mechanical turk beta
Turkers

Study: Behavior of Gradual Types

DLS'18

People

Company Name

Engineers

Students

amazon mechanical turk beta

Turkers

How do the responses relate to the 3 **semantics**?

Guarded

Types enforce behaviors

Transient

Types enforce top-level shapes

Erasure

Types enforce nothing

Study: Behavior of Gradual Types

DLS'18

People

Company Name

Engineers

Students

amazon mechanical turk

Turkers

Expected & Like

✓ **Guarded**

Types enforce behaviors

Unexpected & Dislike

✗ **Transient**

Types enforce top-level shapes

✗ **Erasure**

Types enforce nothing

Case Closed?

✓ **Guarded**

Types enforce behaviors

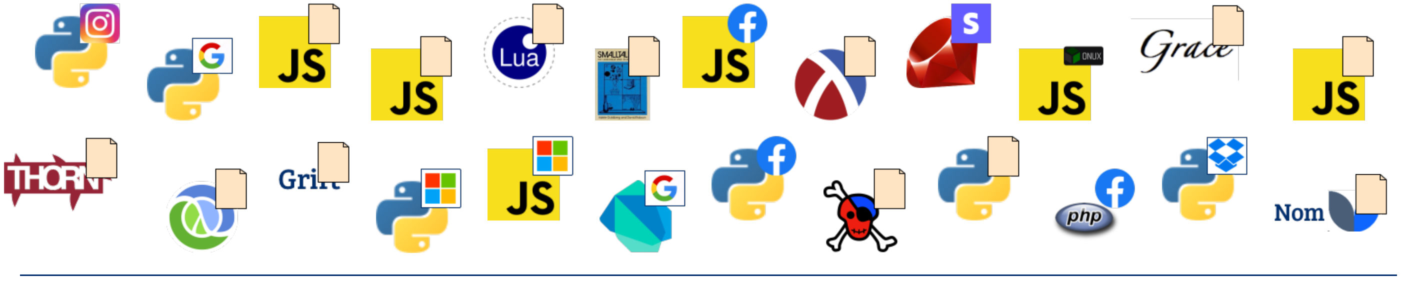
✗ **Transient**

Types enforce top-level shapes

✗ **Erasure**

Types enforce nothing

Case Closed?



✓ **Guarded**

Types enforce behaviors

✗ **Transient**

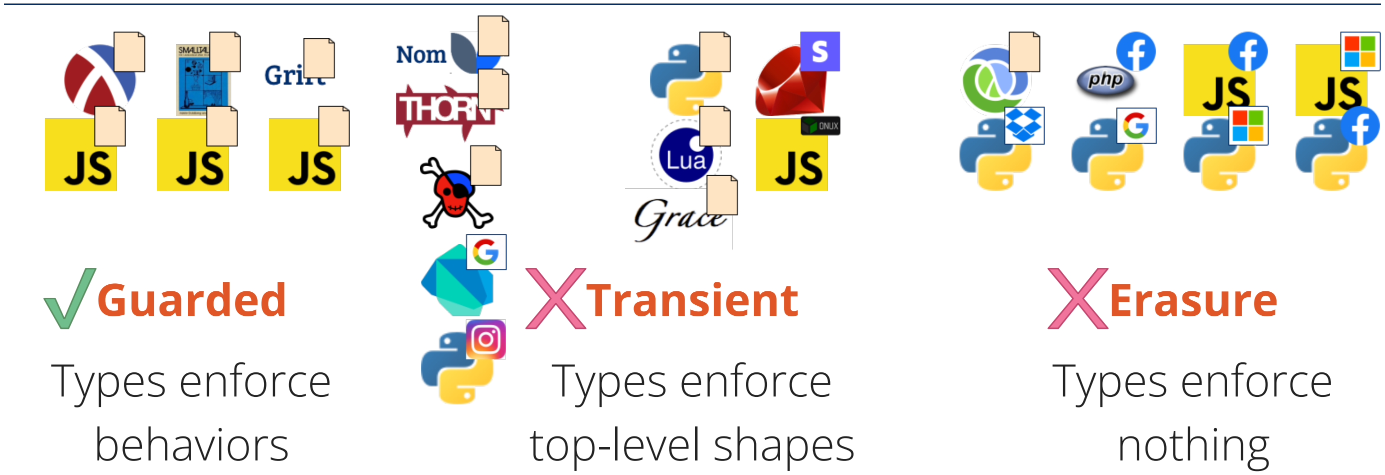
Types enforce top-level shapes

✗ **Erasure**

Types enforce nothing

Case Closed? No!

Funny split …



✓ **Guarded**

Types enforce behaviors

✗ **Transient**

Types enforce top-level shapes

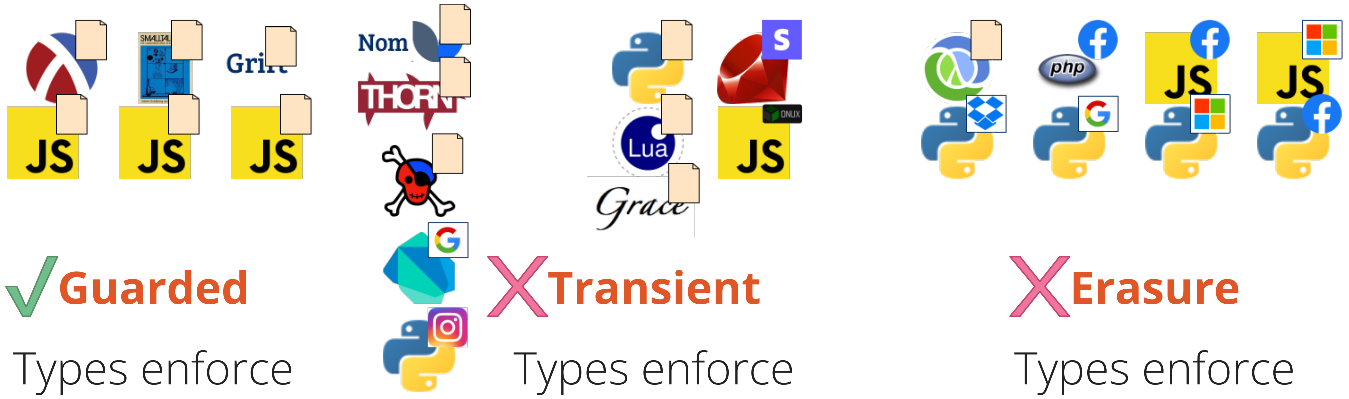
✗ **Erasure**

Types enforce nothing

Case Closed? No!

Funny split …

Research Languages **vs.** Popular Languages



✓ **Guarded**

Types enforce behaviors

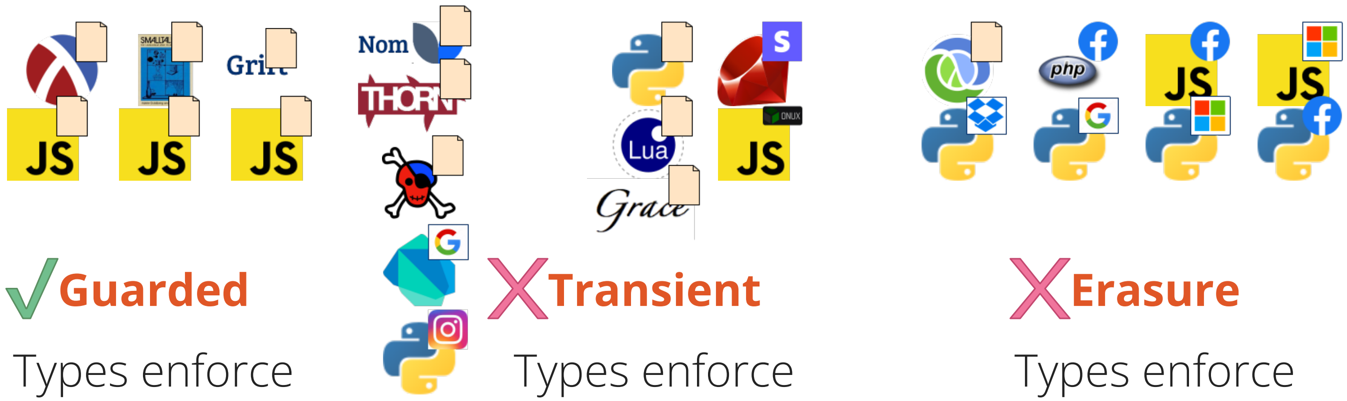
✗ **Transient**

Types enforce top-level shapes

✗ **Erasure**

Types enforce nothing

Case Closed? No!



✓Guarded

Types enforce behaviors

✗Transient

Types enforce top-level shapes

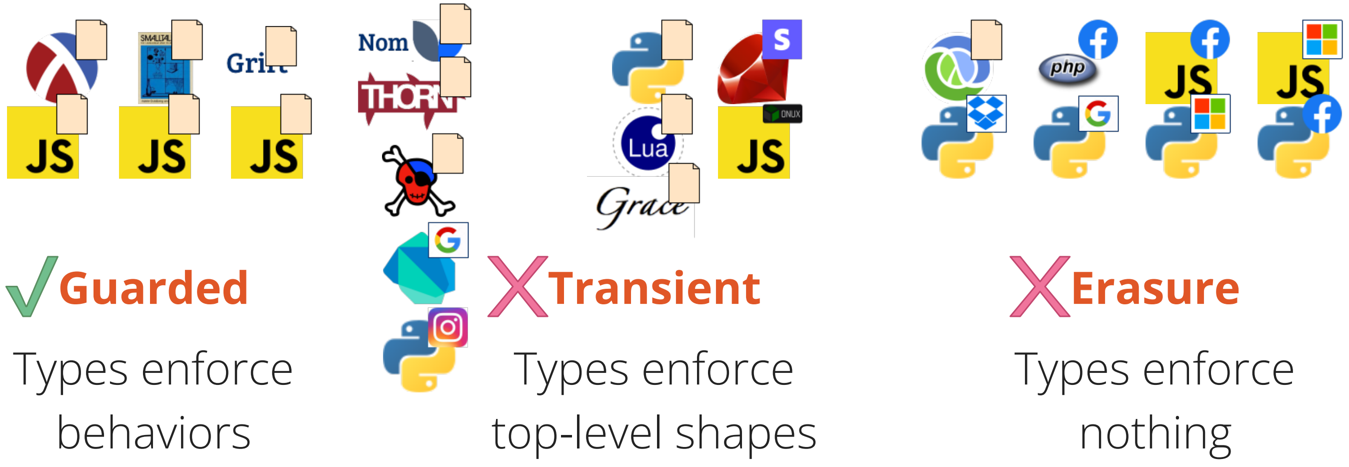
✗Erasure

Types enforce nothing

Case Closed? No!

There are **two** problems:

- How should gradual types **behave**?
- What do behaviors **cost**?



✓ **Guarded**

Types enforce behaviors

✗ **Transient**

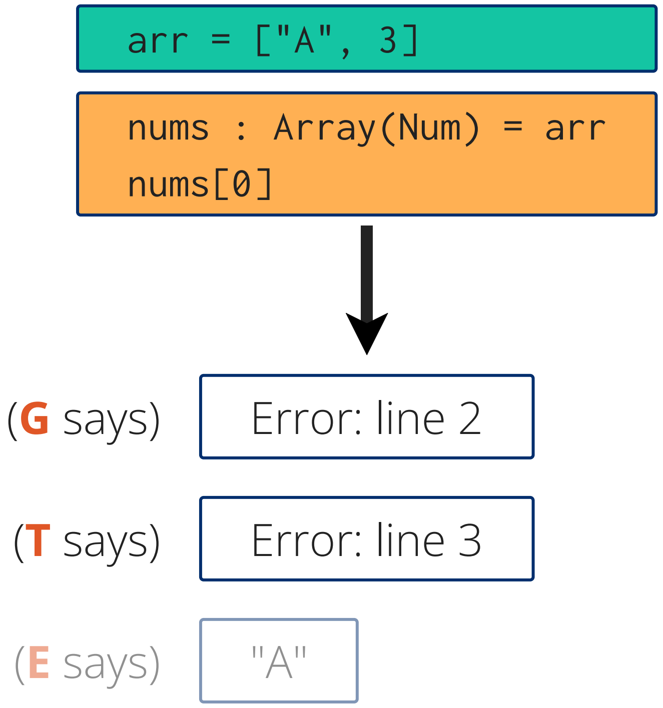
Types enforce top-level shapes

✗ **Erasure**

Types enforce nothing

Where Do Costs Come From?

Where Do Costs Come From?



```
arr = ["A", 3]
```

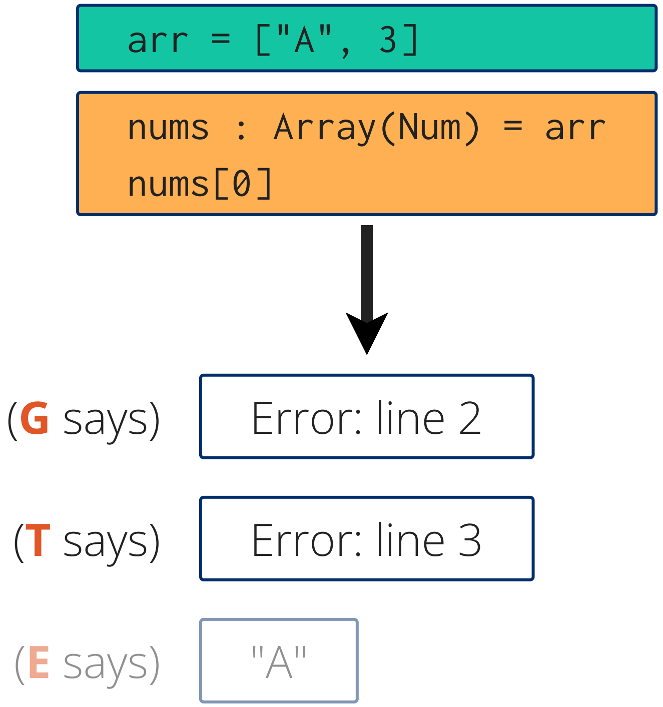
```
nums : Array(Num) = arr
nums[0]
```

(G says) Error: line 2

(T says) Error: line 3

(E says) "A"

Where Do Costs Come From?



```
arr = ["A", 3]
```

```
nums : Array(Num) = arr
nums[0]
```

(**G** says) Error: line 2

(**T** says) Error: line 3

(**E** says) "A"

To detect an Error:

- **traverse** array at boundary
- or **wrap** and delay checks

Cost of **checks** can add up!

Caution: Typed Racket

Guarded type guarantees, but **huge** worst-case costs

25x

180x

1400x

Q. Are bad points common, or rare?

Need a **method** to measure performance

▲ **Performance**

One Program, Many Points

What to Measure = **All** Gradual Possibilities

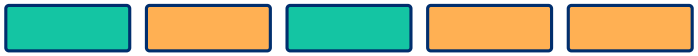
One Program, Many Points

What to Measure = **All** Gradual Possibilities

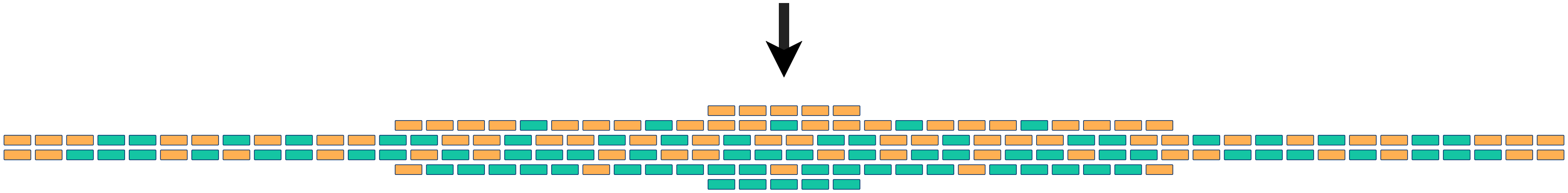
One program with **5** components ...

One Program, Many Points

What to Measure = **All** Gradual Possibilities



One program with **5** components ...



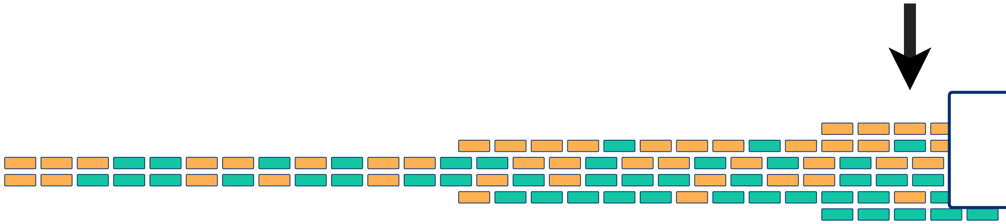
... leads to **32** gradual points

In general, **N** components => **2^N** points

One Program, Many Points

What to Measure = **All** Gradual Possibilities

One program with **5** components ...



Challenge: How to analyze the data?

... leads to **32** gradual points

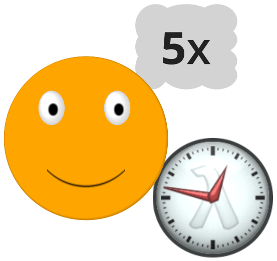
In general, **N** components => **2^N** points

Performance Insight

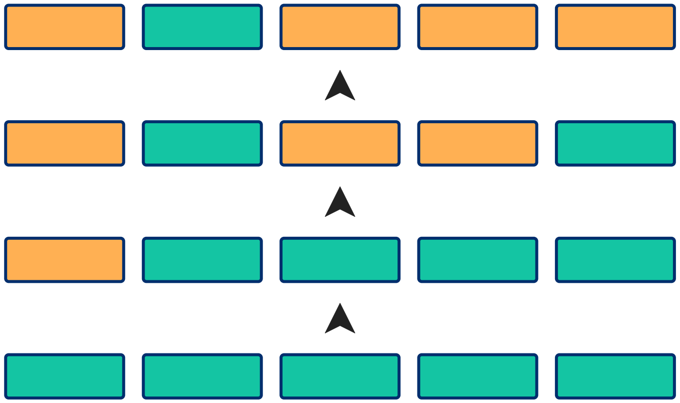
Challenge: How to analyze the data?

Focus on **D-deliverable** configurations

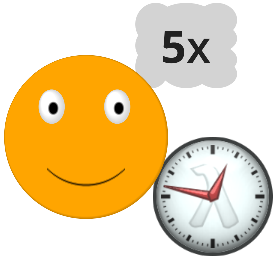
D-deliverable: The Idea



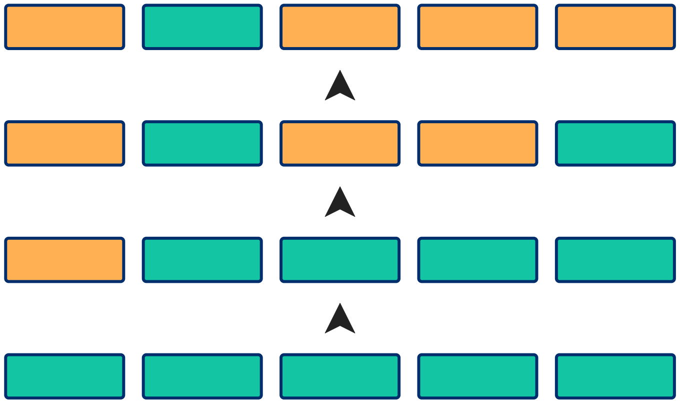
Are we *fast enough*?



D-deliverable: The Idea



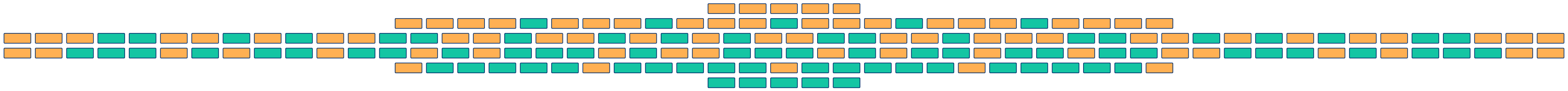
Are we *fast enough*?



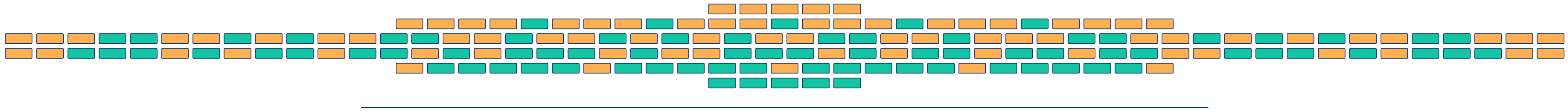
Worst-case overhead is not important

Dx slower is the upper bound

D-deliverable: How to Use



D-deliverable: How to Use



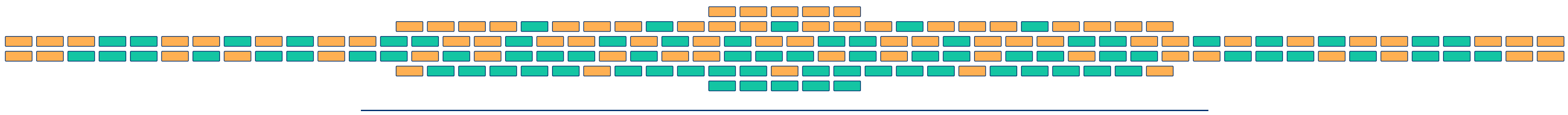
Compress to a **proportion** …

D = 2

➤

50%

D-deliverable: How to Use



Compress to a **proportion** ...

D = 2

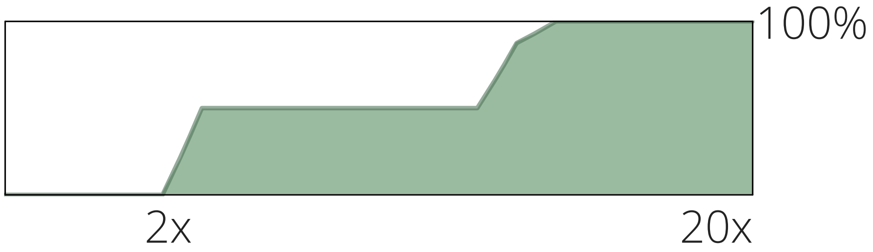
▼

50%

... or to a **CDF**

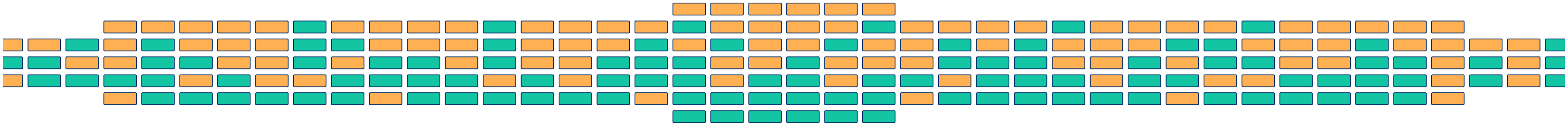
D ∈ [1, 20]

▼

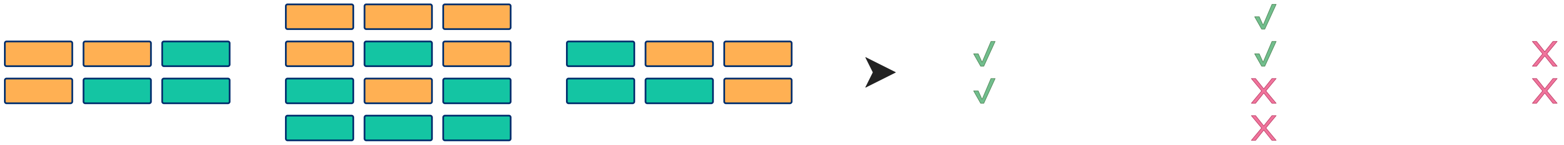


D-deliverable: How to Scale

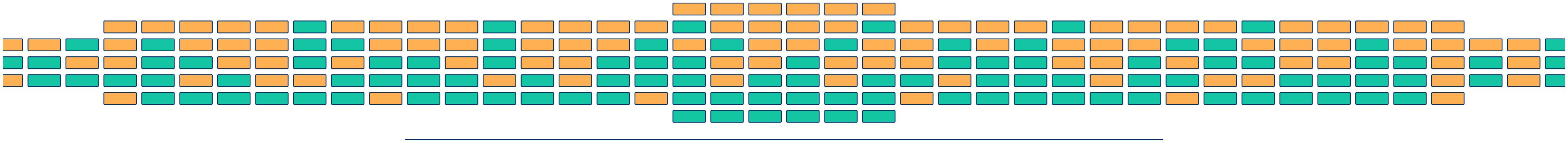
D-deliverable: How to Scale



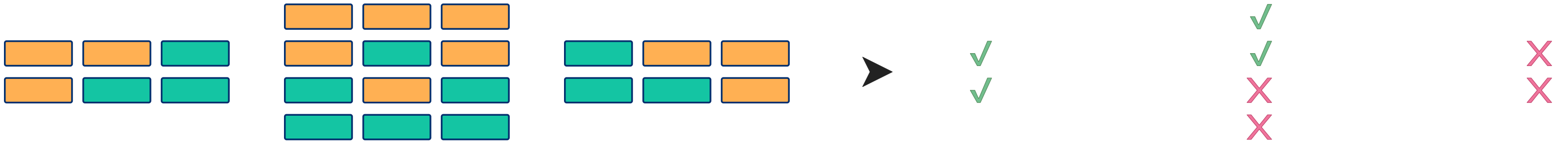
Choosing **D** enables a **Bernoulli random variable**



D-deliverable: How to Scale



Choosing **D** enables a **Bernoulli random variable**

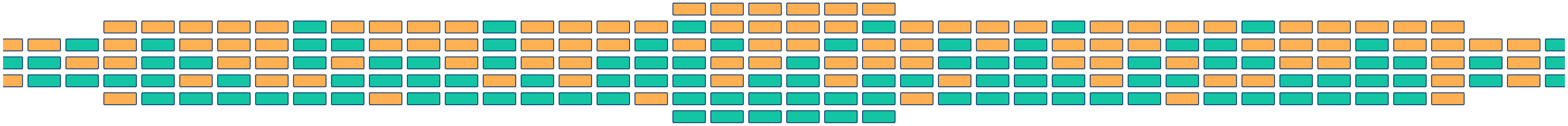


If 50% of **all points** are **D**-deliverable

=>

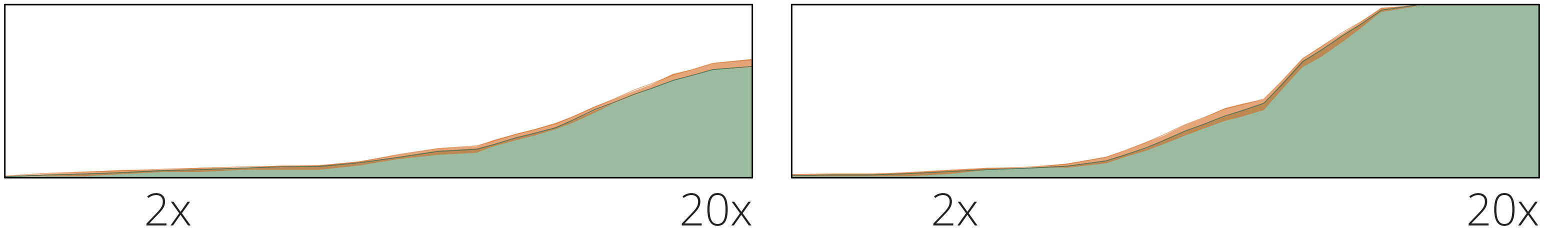
A **random point** has a 50% chance of being fast enough

D-deliverable: How to Scale



Linear sampling has been effective

for **approximating** the **true proportion**



(Orange intervals surround the two green curves)

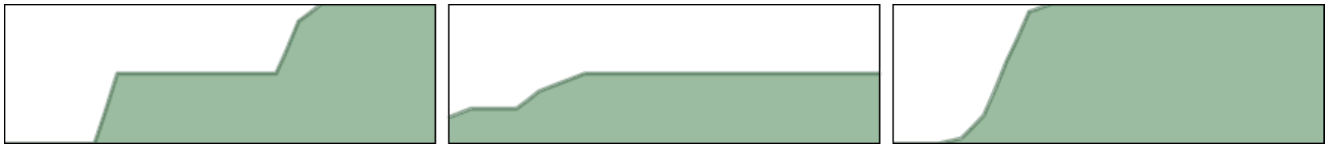
Method

1. Collect benchmark programs

2. Measure all configurations

   or a linear number of samples

3. Focus on the **D-deliverable** configurations



Larger Area = Better Performance

Applications

Typed Racket

POPL'16 JFP'19 OOPSLA'18

Reticulated Python

PEPM'18

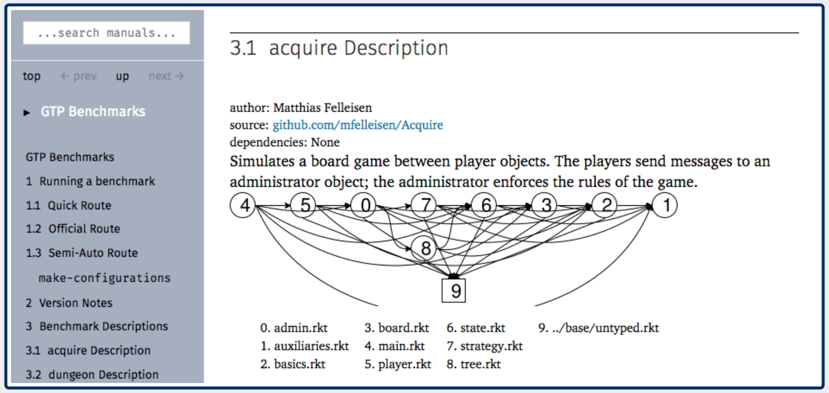
Applications

Curated benchmarks for two languages

Typed Racket

POPL'16 JFP'19

OOPSLA'18



...search manuals...

top ← prev up next →

▸ GTP Benchmarks

GTP Benchmarks

1 Running a benchmark

1.1 Quick Route

1.2 Official Route

1.3 Semi-Auto Route

make-configurations

2 Version Notes

3 Benchmark Descriptions

3.1 acquire Description

3.2 dungeon Description

3.1 acquire Description

author: Matthias Felleisen
source: github.com/mfelleisen/Acquire
dependencies: None

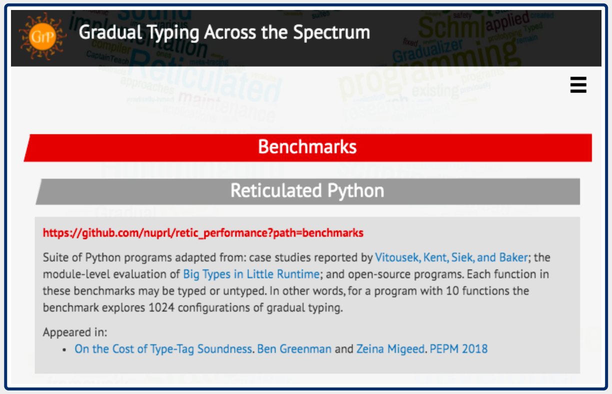
Simulates a board game between player objects. The players send messages to an administrator object; the administrator enforces the rules of the game.

0. admin.rkt
1. auxiliaries.rkt
2. basics.rkt
3. board.rkt
4. main.rkt
5. player.rkt
6. state.rkt
7. strategy.rkt
8. tree.rkt
9. ../base/untyped.rkt

docs.racket-lang.org/gtp-benchmarks

Reticulated Python

PEPM'18



Gradual Typing Across the Spectrum

≡

Benchmarks

Reticulated Python

https://github.com/nuprl/retic_performance?path=benchmarks

Suite of Python programs adapted from: case studies reported by Vitousek, Kent, Siek, and Baker; the module-level evaluation of Big Types in Little Runtime; and open-source programs. Each function in these benchmarks may be typed or untyped. In other words, for a program with 10 functions the benchmark explores 1024 configurations of gradual typing.

Appeared in:

- On the Cost of Type-Tag Soundness. Ben Greenman and Zeina Migeed. PEPM 2018

nuprl.github.io/gtp/benchmarks

from GitHub, Racket packages, Python benchmarks, ... usually without types

Applications

Typed Racket

POPL'16 JFP'19 OOPSLA'18

- **Guarded** semantics
- Bad news! Most **over 20x**
- Better today, but still slow

Reticulated Python

PEPM'18

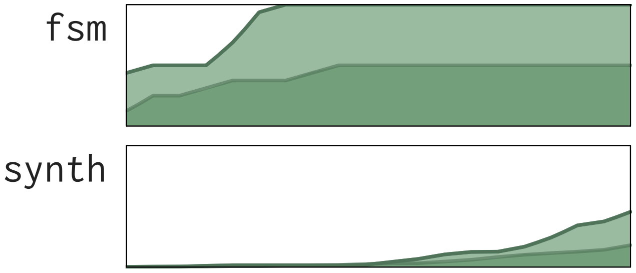
Applications

Typed Racket

POPL'16 JFP'19 OOPSLA'18

- **Guarded** semantics
- Bad news! Most **over 20x**
- Better today, but still slow

Example: 2015 to 2020



Reticulated Python

PEPM'18

Applications

Typed Racket

POPL'16 JFP'19 OOPSLA'18

- **Guarded** semantics
- Bad news! Most **over 20x**
- Better today, but still slow

Reticulated Python

PEPM'18

- **Transient** semantics
- Not bad! All **under 10x**

Applications

Typed Racket

POPL'16 JFP'19 OOPSLA'18

- **Guarded** semantics
- Bad news! Most **over 20x**
- Better today, but still slow

Reticulated Python

PEPM'18

- **Transient** semantics
- Not bad! All **under 10x**

Q. Are **Guarded** and **Transient** "equally" type-sound?

Applications

Typed Racket POPL'16 JFP'19 OOPSLA'18

- **Guarded** semantics
- Bad news! Most **over 20x**
- Better today, but still slow

Reticulated Python PEPM'18

- **Transient** semantics
- Not bad! All **under 10x**

Q. Are **Guarded** and **Transient** "equally" type-sound?

Need a **method** to assess type guarantees

Proofs

Q. Are **Guarded** and **Transient** "equally" type-sound?

Q. Are **Guarded** and **Transient** "equally" type-sound?

Type Soundness (TS) is the standard property for typed languages

"typed code agrees with the types"

Q. Are **Guarded** and **Transient** "equally" type-sound?

Type Soundness (TS) is the standard property for typed languages

"typed code agrees with the types"

Both **Guarded** and **Transient** satisfy **TS** theorems ...

Q. Are **Guarded** and **Transient** "equally" type-sound?

Type Soundness (TS) is the standard property for typed languages

"typed code agrees with the types"

Both **Guarded** and **Transient** satisfy **TS** theorems ...

... but our survey says they're different

✓**Guarded**

✗**Transient**

Q. Are **Guarded** and **Transient** "equally" type-sound?

Type Soundness (TS) is the standard property for typed languages

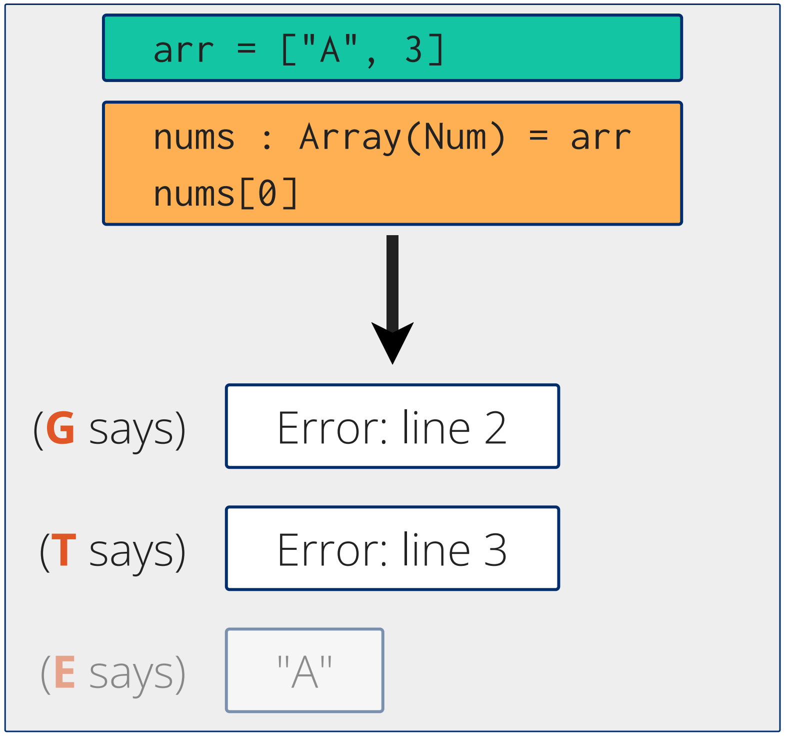
"typed code agrees with the types"

Both **Guarded** and **Transient** satisfy **TS** theorems ...

... but our survey says they're different

✓**Guarded**

✗**Transient**



```
arr = ["A", 3]
```

```
nums : Array(Num) = arr
nums[0]
```

(**G** says) Error: line 2

(**T** says) Error: line 3

(**E** says) "A"

From TS to CM

OOPSLA'19

In Submission'22

Both **Guarded** and **Transient** satisfy **type soundness** (TS)

Only **Guarded** satisfies **complete monitoring** (CM)

From TS to CM

OOPSLA'19

In Submission'22

Both **Guarded** and **Transient** satisfy **type soundness** (TS)

Only **Guarded** satisfies **complete monitoring** (CM)

```
arr = ["A", 3]
```

```
nums : Array(Num) = arr
```

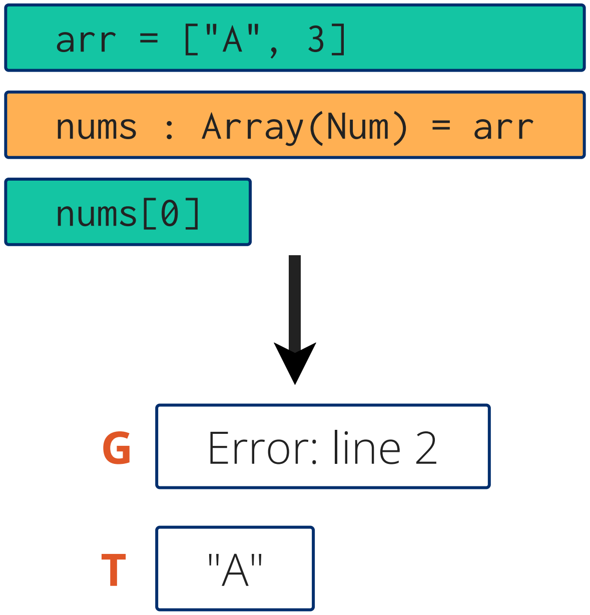
```
nums[0]
```

From TS to CM

OOPSLA'19 In Submission'22

Both **Guarded** and **Transient** satisfy **type soundness** (TS)

Only **Guarded** satisfies **complete monitoring** (CM)



```
arr = ["A", 3]
nums : Array(Num) = arr
nums[0]
```

G Error: line 2

T "A"

From TS to CM

OOPSLA'19 In Submission'22

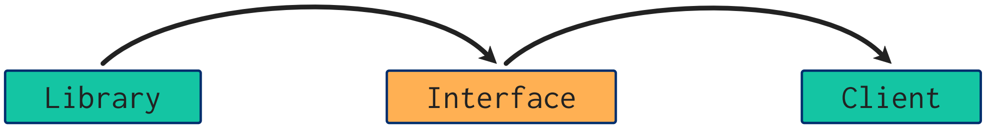
Both **Guarded** and **Transient** satisfy **type soundness** (TS)

Only **Guarded** satisfies **complete monitoring** (CM)

```
arr = ["A", 3]
nums : Array(Num) = arr
nums[0]
```

G Error: line 2

T "A"



From TS to CM

OOPSLA'19

In Submission'22

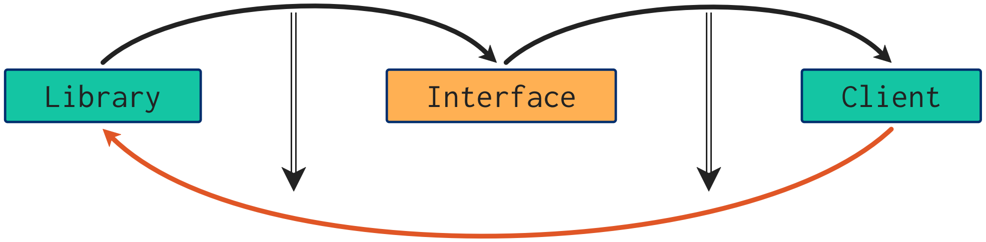
Both **Guarded** and **Transient** satisfy **type soundness** (TS)

Only **Guarded** satisfies **complete monitoring** (CM)

```
arr = ["A", 3]
nums : Array(Num) = arr
nums[0]
```

G Error: line 2

T "A"



From TS to CM

OOPSLA'19 In Submission'22

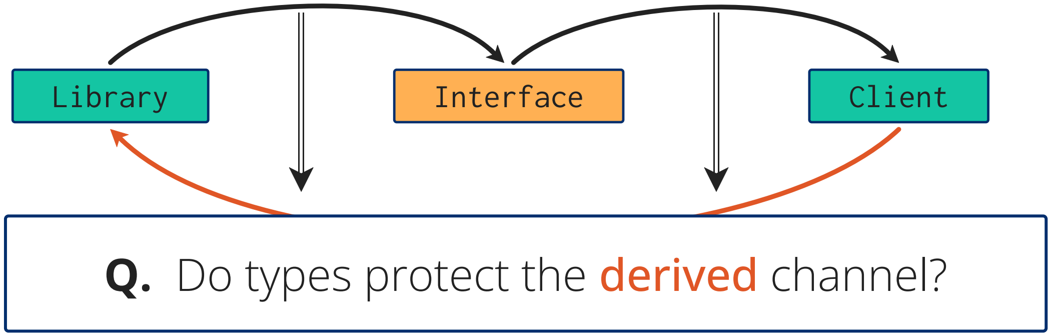
Both **Guarded** and **Transient** satisfy **type soundness** (TS)

Only **Guarded** satisfies **complete monitoring** (CM)

```
arr = ["A", 3]
nums : Array(Num) = arr
nums[0]
```

G Error: line 2

T "A"



Q. Do types protect the **derived** channel?

From TS to CM

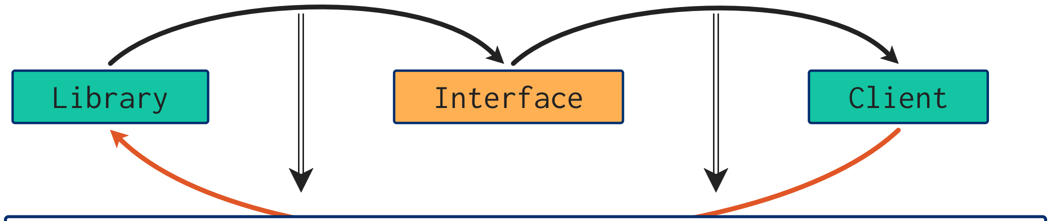
OOPSLA'19 In Submission'22

Both **Guarded** and **Transient** satisfy **type soundness** (TS)
Only **Guarded** satisfies **complete monitoring** (CM)

```
arr = ["A", 3]
nums : Array(Num) = arr
nums[0]
```

G Error: line 2

T "A"



Q. Do types protect the **derived** channel?

Guarded (CM+TS): **Yes**
types made the channel

Transient (TS): **No**
channel is untyped to untyped

Applications

Typed Racket

Reticulated Python

Applications

Typed Racket

Reticulated Python

Q. Are **Guarded** and **Transient** types equally ***strong***?

Applications

Typed Racket

Reticulated Python

Q. Are **Guarded** and **Transient** types equally ***strong***?

No!

Applications

Typed Racket

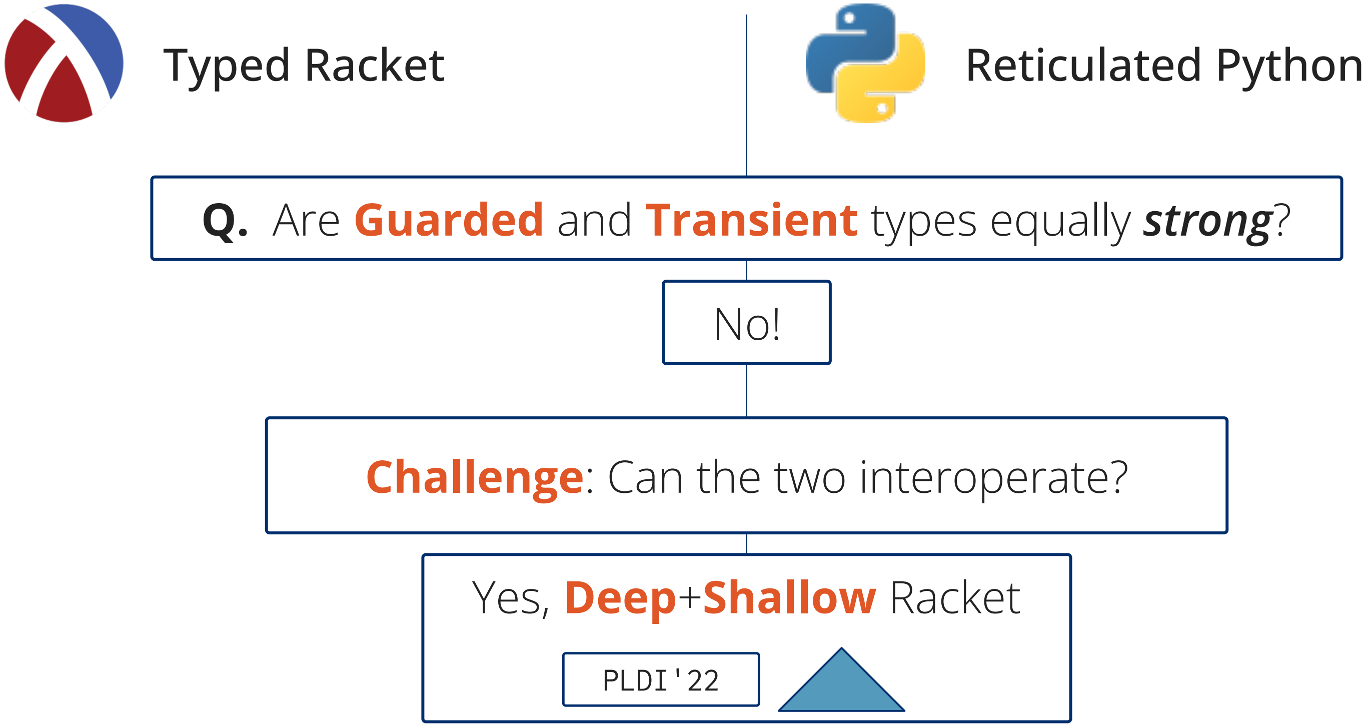
Reticulated Python

Q. Are **Guarded** and **Transient** types equally ***strong***?

No!

Challenge: Can the two interoperate?

Applications



Typed Racket

Reticulated Python

Q. Are **Guarded** and **Transient** types equally ***strong***?

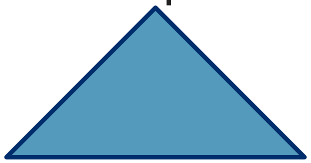
No!

Challenge: Can the two interoperate?

Yes, **Deep**+**Shallow** Racket

PLDI'22

Foundations for Gradual Languages

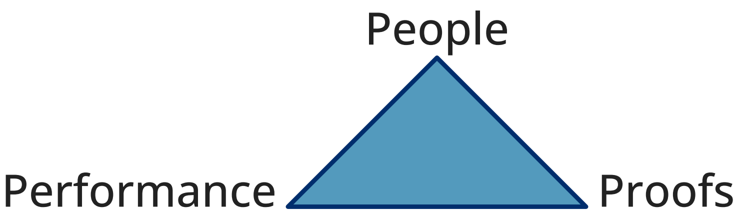


People

Performance

Proofs

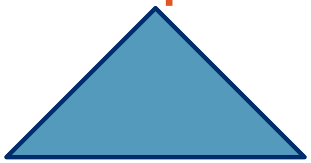
Foundations for Gradual Languages



Research Contributions:

- ➤ Characterizing Designs
- ➤ Directing Improvements
- ➤ Inspiring New Languages

Ongoing Work



People

Performance

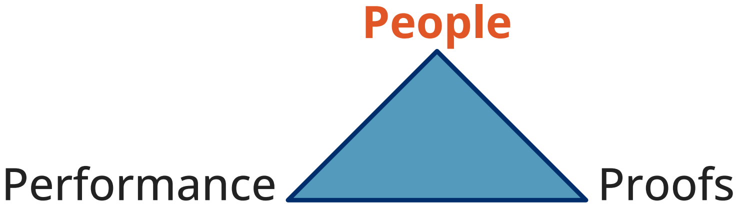
Proofs

Ongoing Work

Static Python at Instagram

Few types, but fast performance

Gradual soundness: type guarantees vs. ease-of-use



People

Performance

Proofs

Ongoing Work

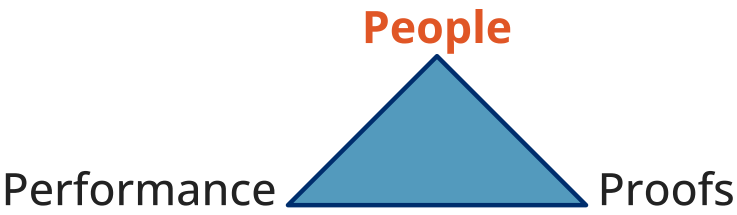
Static Python

Gradual Soundness

Rational Programmer

A method for PL pragmatices

Humans out-of-the-loop



People

Performance

Proofs

Ongoing Work

Static Python

Gradual Soundness

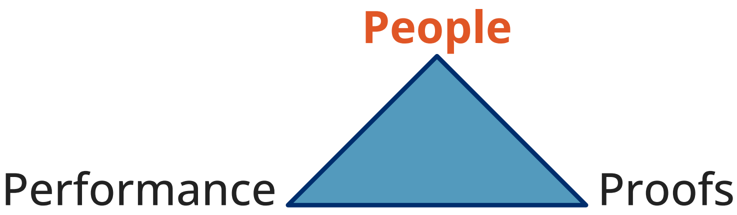
Rational Programmer

Directly measure pragmatics

Human Factors for Formal Methods:

Language levels for **Alloy**

LTL misconceptions (next slide)



LTL Misconceptions

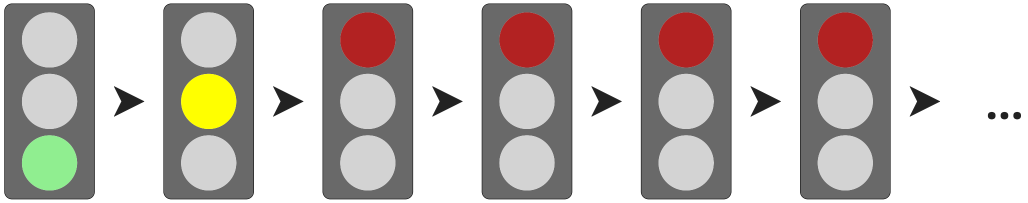
Linear Temporal Logic

used in: **verification**, **synthesis**, and **robot planning**

LTL Misconceptions

Linear Temporal Logic

used in: **verification**, **synthesis**, and **robot planning**

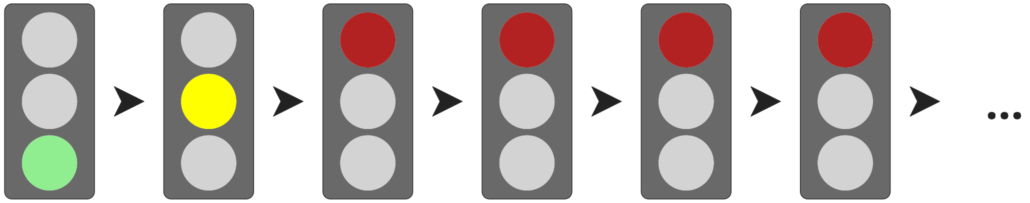


Is Green eventually on?

LTL Misconceptions

Linear Temporal Logic

used in: **verification**, **synthesis**, and **robot planning**



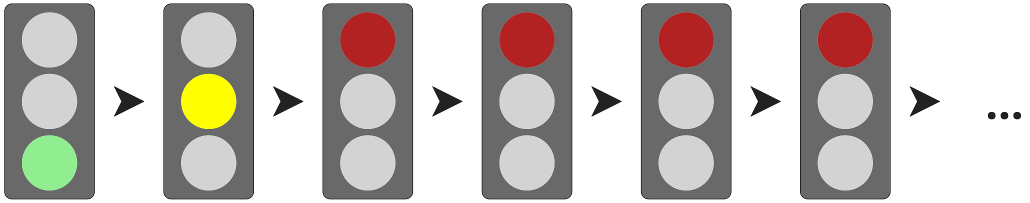
Is Green eventually on?

True

LTL Misconceptions

Linear Temporal Logic

used in: **verification**, **synthesis**, and **robot planning**



Is Green eventually on?

True

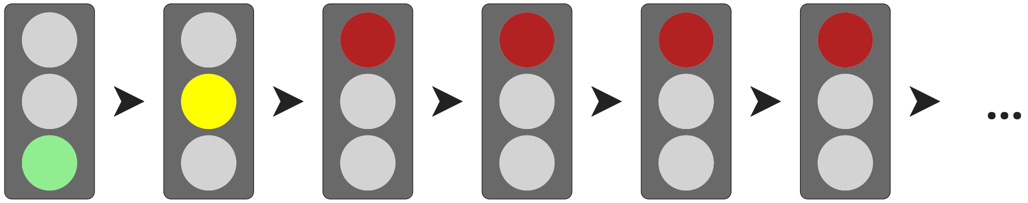
Q. **In what ways** is LTL tricky, and **what can we do** about it?

Studies with researchers & students

LTL Misconceptions

Linear Temporal Logic

used in: **verification**, **synthesis**, and **robot planning**



Is Green eventually on?

True

Q. **In what ways** is LTL tricky, and **what can we do** about it?

Studies with researchers & students

Early outcome: **Better syntax** for Alloy 6

Ongoing Work

Static Python

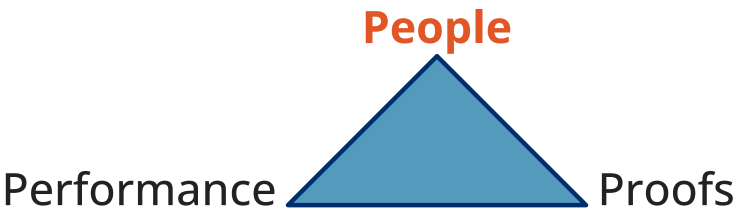
Gradual Soundness

Rational Programmer

Directly measure pragmatics

Human Factors for FM

Alloy and LTL



People

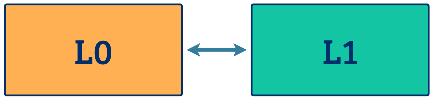
Performance

Proofs

Future Work

Future Work

Typed + Untyped is a multi-language problem



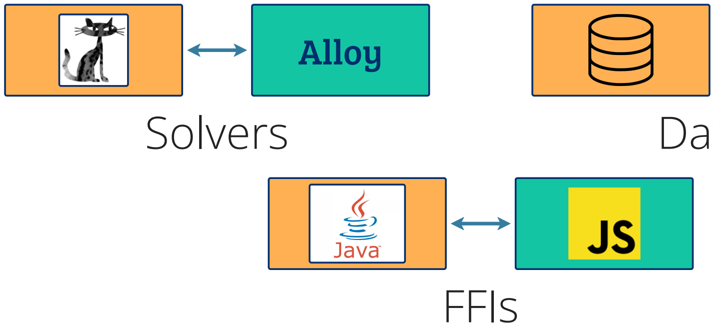
- 2 similar languages
- higher-order interoperability
- strong vs. weak invariants

Future Work

Multi-language systems are everywhere!

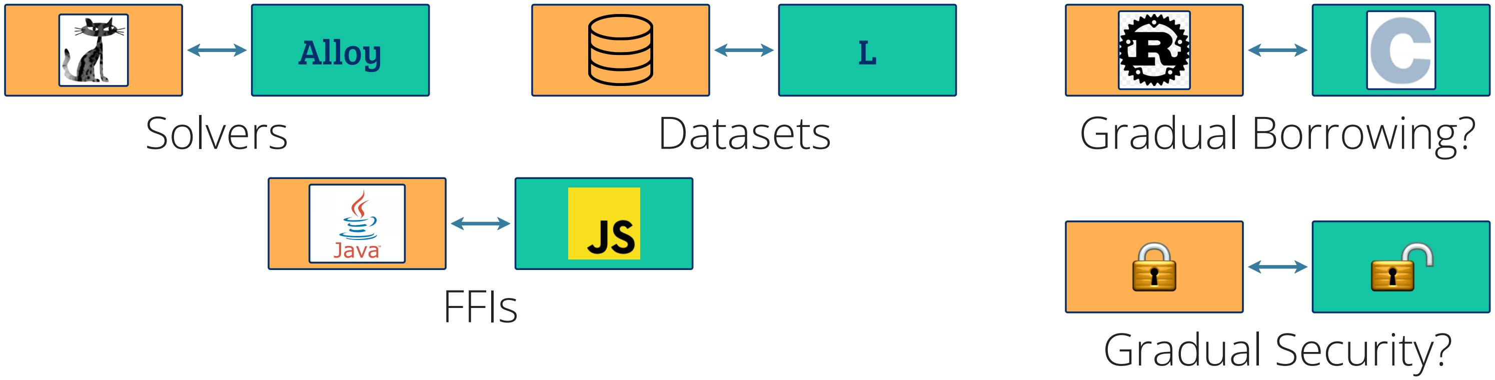
Future Work

Multi-language systems are everywhere!



Future Work

Multi-language systems are everywhere!



Solvers

Datasets

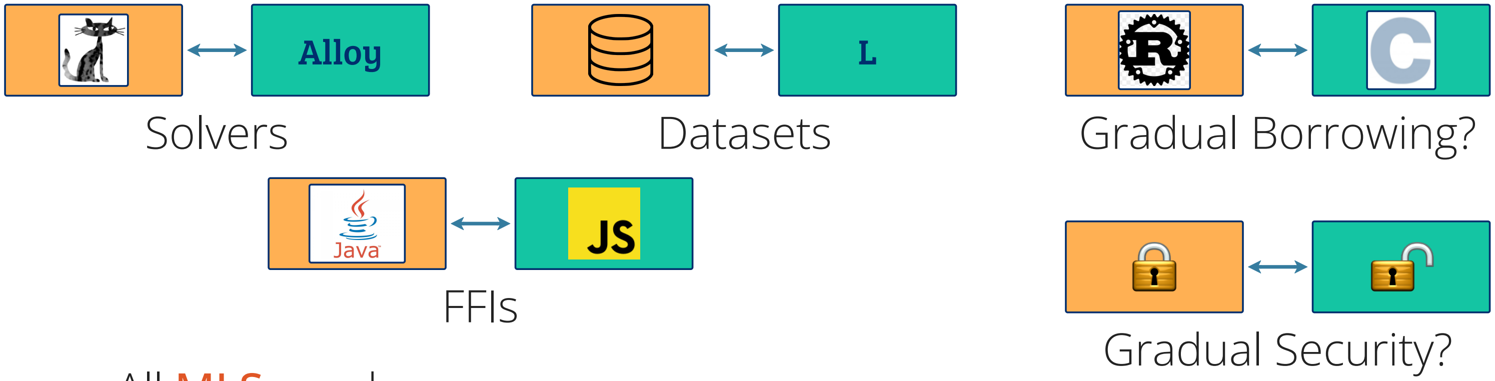
Gradual Borrowing?

FFIs

Gradual Security?

Future Work

Multi-language systems are everywhere!



Solvers

Datasets

Gradual Borrowing?

FFIs

Gradual Security?

All MLS need:

- ➤ Expressive Boundaries
- ➤ Correct & Fast Validation

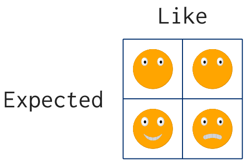
PPP

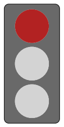
Balanced Foundation

People

Behavior of Gradual Types

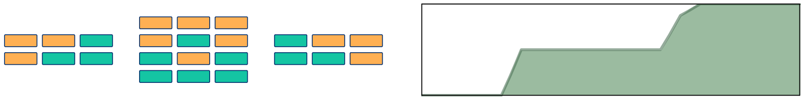
Human Factors for Formal Methods





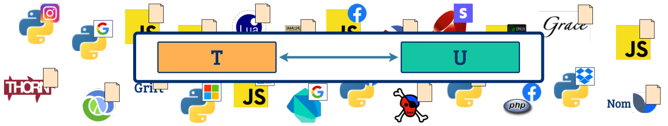
Performance

Measuring Costs at Scale

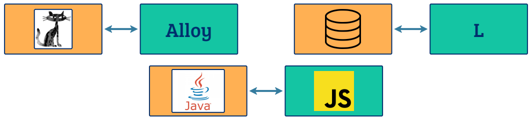


Proofs

Comparing Type Guarantees



Methods for **multi-language systems**



Teaching Alloy

Alloy is a **modeling language** that comes with **two styles**:

Predicate

```
all a, b, c: univ |
  a->b in f and b->c in f
    implies a->c in f
```

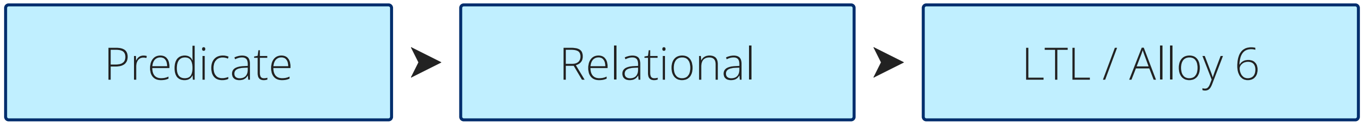
Relational

```
f.f in f
```

(**f** is transitive)

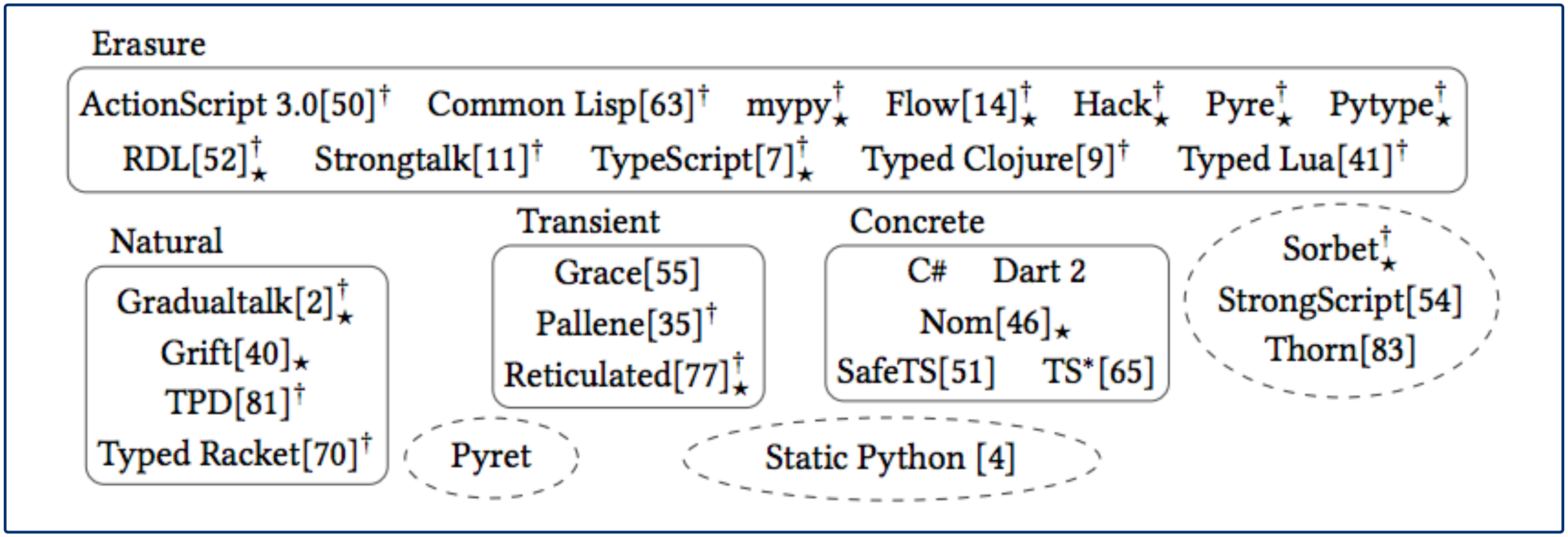
Problem: **errors** assume you know both styles!

Q. Can **language levels** give a smooth introduction?



Predicate ➤ Relational ➤ LTL / Alloy 6

Informal Landscape



Erasure

ActionScript 3.0[50]† Common Lisp[63]† mypy$^{\dagger}_{\star}$ Flow[14]$^{\dagger}_{\star}$ Hack$^{\dagger}_{\star}$ Pyre$^{\dagger}_{\star}$ Pytype$^{\dagger}_{\star}$ RDL[52]$^{\dagger}_{\star}$ Strongtalk[11]† TypeScript[7]$^{\dagger}_{\star}$ Typed Clojure[9]† Typed Lua[41]†

Natural

Gradualtalk[2]$^{\dagger}_{\star}$ Grift[40]$_{\star}$ TPD[81]† Typed Racket[70]†

Transient

Grace[55] Pallene[35]† Reticulated[77]$^{\dagger}_{\star}$

Concrete

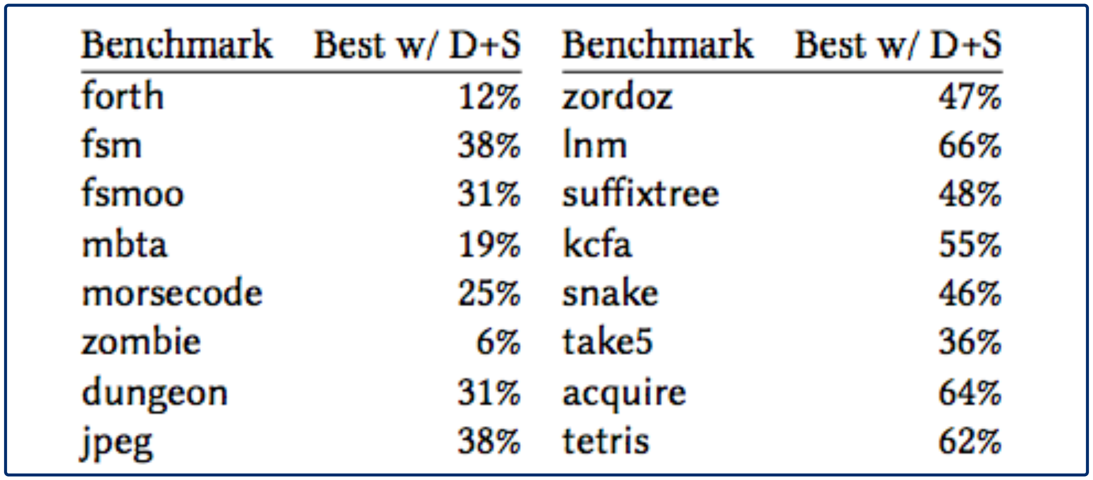
C# Dart 2 Nom[46]$_{\star}$ SafeTS[51] TS*[65]

Sorbet$^{\dagger}_{\star}$ StrongScript[54] Thorn[83]

Pyret

Static Python [4]

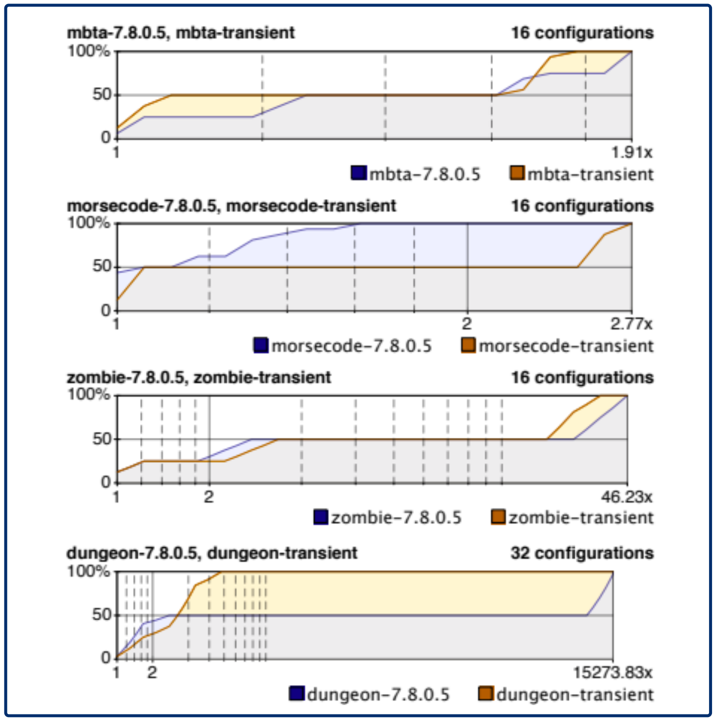
Deep + Shallow

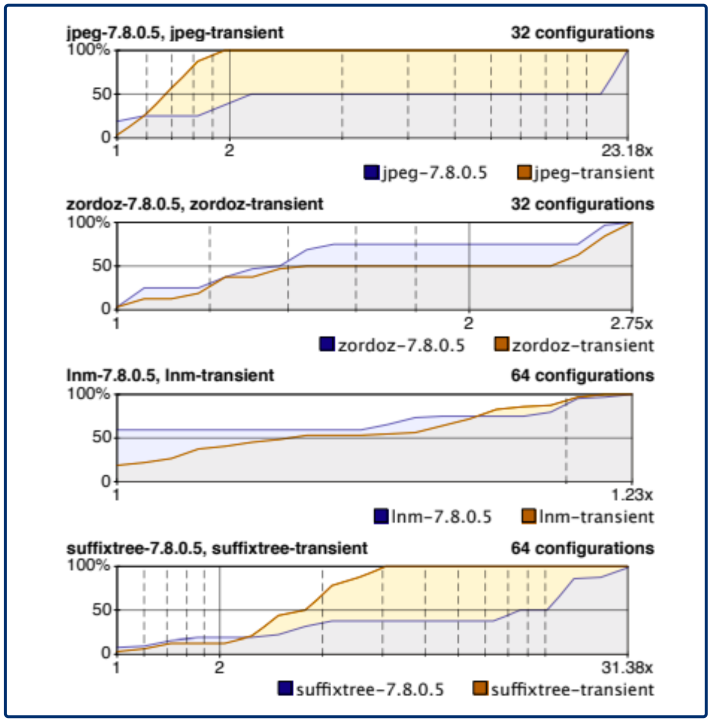


Benchmark	Best w/ D+S	Benchmark	Best w/ D+S
forth	12%	zordoz	47%
fsm	38%	lnm	66%
fsmoo	31%	suffixtree	48%
mbta	19%	kcfa	55%
morsecode	25%	snake	46%
zombie	6%	take5	36%
dungeon	31%	acquire	64%
jpeg	38%	tetris	62%

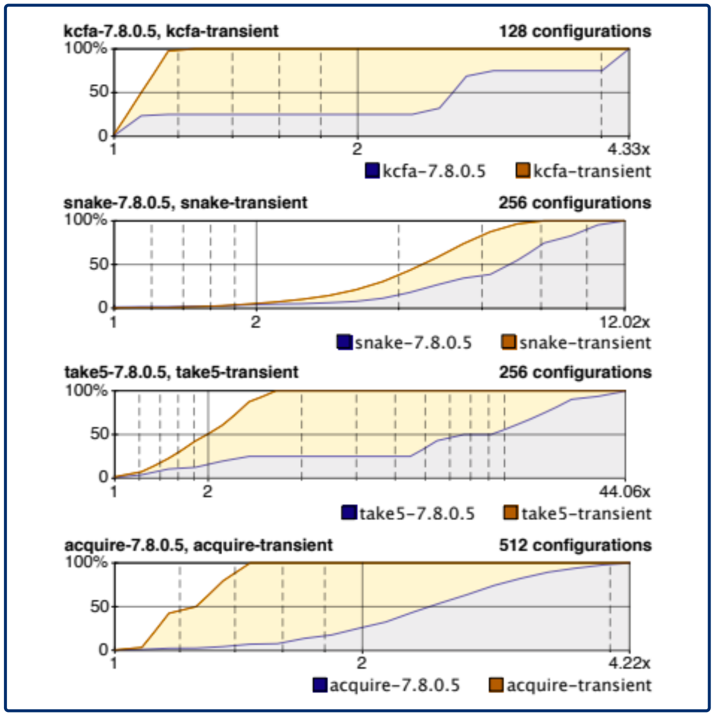
Percent of gradual points that run fastest with a Deep+Shallow mix

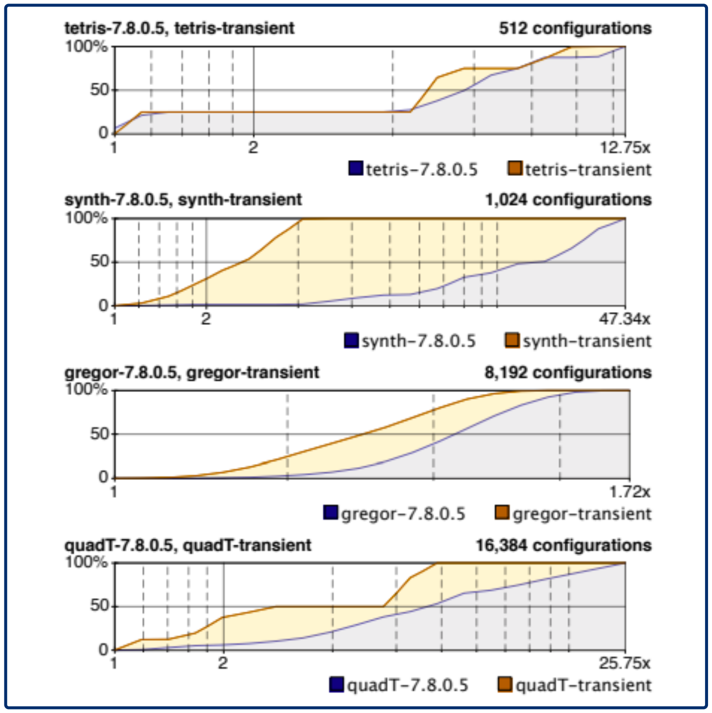
Deep or Shallow (1/2)





Deep or Shallow (2/2)





Prior Work

	Guarded	Transient	Erasure
type soundness			
dyn. gradual guarantee			
blame theorem			

Prior Work

	Guarded	Transient	Erasure
type soundness	✓	✓	✗
dyn. gradual guarantee	✓	✓	✓
blame theorem	✓	✓	✓

Standard tools **do not** tell the difference!

A Toolbox to Measure Type Guarantees

	Guarded	Transient

A Toolbox to Measure Type Guarantees

	Guarded	Transient
complete monitoring	✓	✗

CM: Do types protect all channels?

A Toolbox to Measure Type Guarantees

	Guarded	Transient
complete monitoring	✓	✗
blame soundness	✓	✗
blame completeness	✓	✗

CM: Do types protect all channels?

BS: Do errors point to *only* relevant channels?

BC: Do errors point to *all* relevant channels?

A Toolbox to Measure Type Guarantees

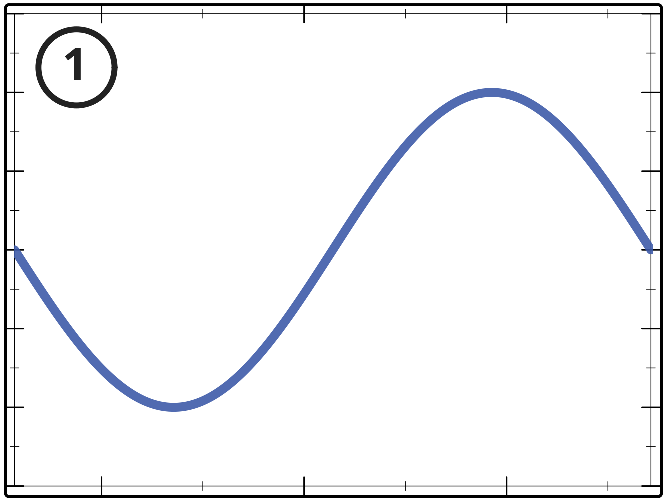
	Guarded	C	F	Transient	A	E
type soundness						
complete monitoring						
blame soundness						
blame completeness						
error preorder						

A Toolbox to Measure Type Guarantees

	Guarded	C	F	Transient	A	E
type soundness	✓	✓	✓	y	✓	✗
complete monitoring	✓	✓	✗	✗	✗	✗
blame soundness	✓	✓	✓	h	✓	0
blame completeness	✓	✓	✓	✗	✓	✗
error preorder	Guarded <	C <	F <	Transient =	A <	E

Example: Clickable Plot

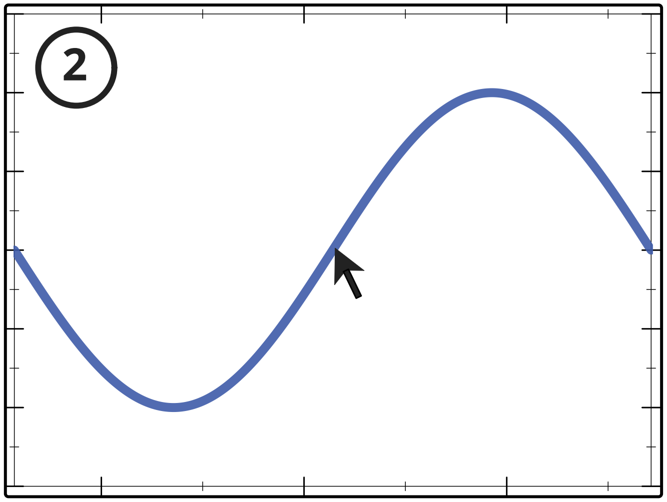
Type Soundness cannot distinguish **Guarded** and **Transient**



1. Plot data
2. Listen for a click
3. Draw an image

Example: Clickable Plot

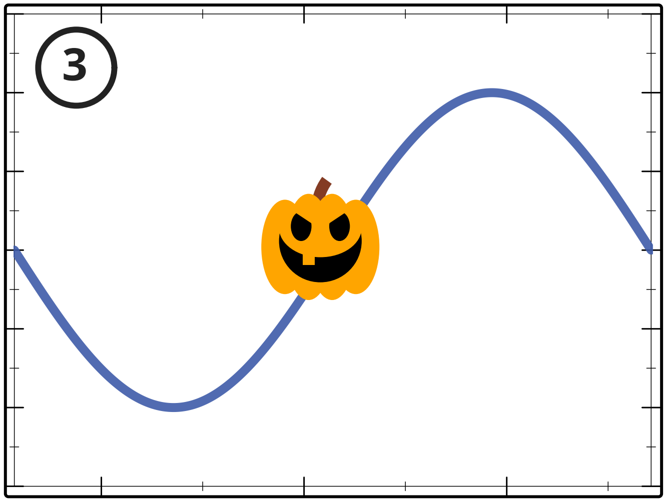
Type Soundness cannot distinguish **Guarded** and **Transient**



1. Plot data
2. Listen for a click
3. Draw an image

Example: Clickable Plot

Type Soundness cannot distinguish **Guarded** and **Transient**



1. Plot data
2. Listen for a click
3. Draw an image

Example: Clickable Plot

Type Soundness cannot distinguish **Guarded** and **Transient**

Example: Clickable Plot

Type Soundness cannot distinguish **Guarded** and **Transient**

```
type ClickPlot
  init
    Num,Num -> Image

  mouseHandler
    MouseEvt -> Void


  show
    -> Void
```

```
class ClickPlot
  init(onClick)
    # set up

  mouseHandler(evt)
    i = onClick(evt)
    # add image

  show()
    # display
```

Example: Clickable Plot

Type Soundness cannot distinguish **Guarded** and **Transient**

```
function h(x)
  if 0 < fst(x)
    pumpkin
  else
    fish

p = ClickPlot(h)
p.show()
# user clicks
```

```
type ClickPlot
  init
    Num,Num -> Image

  mouseHandler
    MouseEvt -> Void


  show
    -> Void
```

```
class ClickPlot
  init(onClick)
    # set up

  mouseHandler(evt)
    i = onClick(evt)
    # add image

  show()
    # display
```

Example: Clickable Plot

Type Soundness cannot distinguish **Guarded** and **Transient**

```
function h(x)
  if 0 < fst(x)
    pumpkin
  else
    fish

p = ClickPl
p.show()
# user clic
```

```
type ClickPlot
  init
    Num,Num -> Image
```

```
class ClickPlot
  init(onClick)
    # set up

         ndler(evt)
         nClick(evt)
          image



         play
```

Guarded: error at the **type boundary**

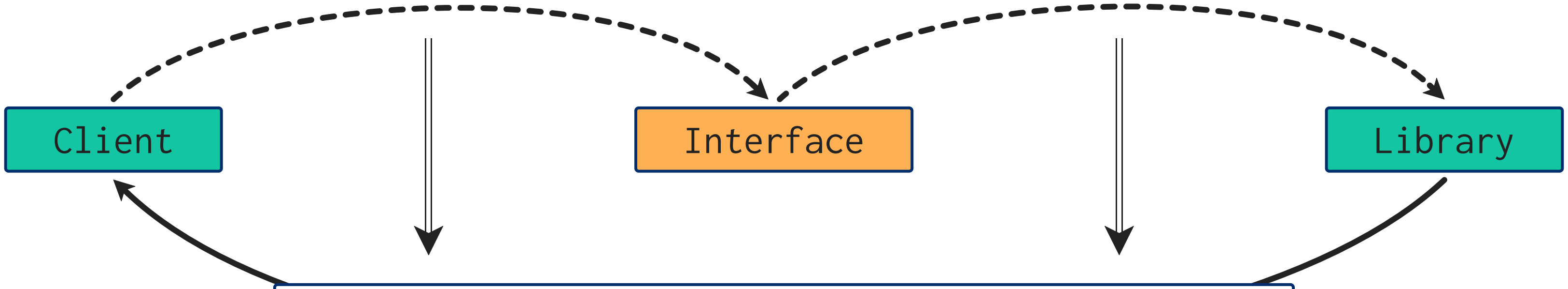
(coordinate pair vs. mouse event)

Transient: error **within** the client

the real issue is **off the stack**!

Example: Clickable Plot

Type Soundness cannot distinguish **Guarded** and **Transient**



Q. Do types protect the **callback** channel?

Guarded: **Yes**

types made the channel

Transient: **No**

the channel is untyped to untyped