Mixing Typed and Untyped Code
A Tale of Proofs, Performance, and People

Ben Greenman
Spring 2022
Typed or Untyped?
Typed or Untyped?

Java is typed
   (statically typed)

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

JavaScript is untyped
   (dynamically typed)

```javascript
    var m = {};
```
Typed or Untyped?

Java is typed (statically typed)

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

JavaScript is untyped (dynamically typed)

```
var m = {};
```

With types, languages can:
+ Prevent classes of bugs
+ Support tools
Typed or Untyped?

**Java** is **typed**
(statically typed)

Java

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

**With** types, languages can:
+ Prevent classes of bugs
+ Support tools

**JavaScript** is **untyped**
(dynamically typed)

JavaScript

```javascript
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

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

Strong support for both sides
Typed or Untyped

Strong support for both sides

Untyped PLs dominate on GitHub

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

s://madnight.github.io/gitut/#/pull_requests/2021/4
Typed or Untyped

T | U
Typed or Untyped

Typed AND Untyped
Gradual Typing
Gradual Typing

Key Motivation: improve stable code with types
Gradual Typing

Key Motivation: improve stable code with types

```rust
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

```haskell
definition 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)
```
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

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

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

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

Active space!
Gradual Typing

Common Lisp <1990

StrongTalk 1993

Active space!
Gradual Typing

Active space!

Common Lisp <1990

StrongTalk 1993

Gradual Typing 2006
Gradual Typing

T  U

Active space!

Common Lisp <1990

StrongTalk 1993

Gradual Typing 2006

+ Many Research PLs
Gradual Typing

Active space!

Major companies involved
Gradual Typing

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

+ 8k interfaces
+ 5k contributors
+ 1 million clients
Gradual Typing

Active space!
Major companies involved
Growing community interest

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

+ 8k interfaces
+ 5k contributors
+ 1 million clients
Gradual Typing

So what's the **problem**?
Gradual Typing

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

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

Q. Is n really a number?
Example 1

Typed Function

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

Untyped Caller

`add1("A")`

Q. Is n really a number?
Example 1

Typed Function

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

Untyped Caller

```javascript
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

yes

no

sort of
What Should Types Mean?

No consensus on basic questions!

Num
Array(Num)
What Should Types Mean?

No consensus on basic questions!

Q. Did anyone ask programmers?
What Should Types Mean?
No consensus on basic questions!

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
**Challenge:** How to compare languages?

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

Guarded
Types enforce behaviors

Transient
Types enforce top-level shapes

Erasure
Types enforce nothing

ICFP '18
Proofs

T

U
Study: Behavior of Gradual Types

A method to compare semantics
Study: Behavior of Gradual Types

arr = ["A", 3]
nums: Array(Num) = arr
nums[0]
Study: Behavior of Gradual Types

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

(G says)  Error: line 2
(T says)  Error: line 3
(E says)  "A"

A method to compare semantics
Study: Behavior of Gradual Types

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

(G says) Error: line 2
(T says) Error: line 3
(E says) "A"

A method to compare semantics

Like
Dislike
Expected
Unexpected

68
Study: Behavior of Gradual Types

A method to compare semantics
- One program
- Distinct results
- Task: Label each result

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

(G says) Error: line 2
(T says) Error: line 3
(E says) "A"

Like | Dislike
--- | ---
Expected | Unexpected

69
Study: Behavior of Gradual Types

Engineers  Students  Turkers
Study: Behavior of Gradual Types

DLS'18

People

Engineers
Students
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

Engineers
Students
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 ...

<table>
<thead>
<tr>
<th>Research Languages</th>
<th>vs.</th>
<th>Popular Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="https://example.com/guarded.png" alt="Guarded" /></td>
<td><img src="https://example.com/transient.png" alt="Transient" /></td>
<td><img src="https://example.com/erasure.png" alt="Erasure" /></td>
</tr>
<tr>
<td>Types enforce behaviors</td>
<td>Types enforce top-level shapes</td>
<td>Types enforce nothing</td>
</tr>
</tbody>
</table>

JavaScript (JS), Java, and others vs. Facebook, Microsoft, Google, and others.
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?
Where Do Costs Come From?
Where Do Costs Come From?

```javascript
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?

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

To detect an Error:
- **traverse** array at boundary
- or **wrap** and delay checks

Cost of **checks** can add up!

(G says) Error: line 2
(T says) Error: line 3
(E says) "A"
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 ...

... leads to 32 gradual points

In general, \( N \) components \( \Rightarrow 2^N \) points

**Challenge**: How to analyze the data?
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

Compress to a proportion ...

\[ D = 2 \]

\[ \downarrow \]

50%
D-deliverable: How to Use

Compress to a **proportion** ...

\[ D = 2 \]
\[ \downarrow \]
\[ 50\% \]

... or to a **CDF**

\[ D \in [1, 20] \]
D-deliverable: How to Scale
Choosing D enables a **Bernoulli random variable**
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

Reticulated Python
- PEPM'18

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

Example: 2015 to 2020

<table>
<thead>
<tr>
<th>fsm</th>
<th>synth</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Reticulated Python

PEPM’18

Guarded semantics

Bad news! Most over 20x

Better today, but still slow
Applications

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

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

**Reticulated Python**
- **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

![Emojis]

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

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

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

```javascript
arr = ["A", 3]
nums : Array(Num) = arr
nums[0]
```
From TS to CM

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

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

G Error: line 2
T "A"
From TS to CM

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

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

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

G  Error: line 2
T  "A"
From TS to CM

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

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

Q. Do types protect the derived channel?

G Error: line 2
T "A"
From TS to CM

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

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

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

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

Q. Are Guarded and Transient types equally strong?

No!
Applications

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

No!

**Challenge**: Can the two interoperate?
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

Research Contributions:
- Characterizing Designs
- Directing Improvements
- Inspiring New Languages
Ongoing Work

Performance  People  Proofs
Ongoing Work

Static Python at Instagram
Few types, but fast performance
Gradual soundness: type guarantees vs. ease-of-use
Ongoing Work

Static Python  🐍
Gradual Soundness

Rational Programmer
A method for PL pragmatics
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

Diagram:
- 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!

Solvers
Alloy

Datasets
L

FFIs
Java
JS
Future Work

Multi-language systems are everywhere!

Solvers

Alloy

Datasets

L

FFIs

Java

Gradual Borrowing?

Gradual Security?

C

R
Future Work

Multi-language systems are everywhere!

Solvers

Alloy

Datasets

L

FFIs

Java

JS

Gradual Borrowing?

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
- Flow[14]
- Hack
- Pyre
- Pytype
- RDL[52]
- Strongtalk[11]
- TypeScript[7]
- Typed Clojure[9]
- Typed Lua[41]

Natural
- Gradualtalk[2]
- Grift[40]
- TPD[81]
- Typed Racket[70]

Transient
- Grace[55]
- Pallene[35]
- Reticulated[77]

Concrete
- C#
- Dart 2
- Nom[46]
- SafeTS[51]
- TS*[65]

Pyret
Static Python [4]

Sorbet
StrongScript[54]
Thorn[83]
Deep + Shallow

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

Percent of gradual points that run fastest with a Deep+Shallow mix
Deep or Shallow (1/2)
Deep or Shallow (2/2)
Prior Work

<table>
<thead>
<tr>
<th>Guarded</th>
<th>Transient</th>
<th>Erasure</th>
</tr>
</thead>
<tbody>
<tr>
<td>type soundness</td>
<td></td>
<td></td>
</tr>
<tr>
<td>dyn. gradual guarantee</td>
<td></td>
<td></td>
</tr>
<tr>
<td>blame theorem</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Prior Work

<table>
<thead>
<tr>
<th></th>
<th>Guarded</th>
<th>Transient</th>
<th>Erasure</th>
</tr>
</thead>
<tbody>
<tr>
<td>type soundness</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>dyn. gradual guarantee</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>blame theorem</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

Standard tools **do not** tell the difference!
A Toolbox to Measure Type Guarantees

Guarded    Transient
A Toolbox to Measure Type Guarantees

<table>
<thead>
<tr>
<th></th>
<th>Guarded</th>
<th>Transient</th>
</tr>
</thead>
<tbody>
<tr>
<td>complete monitoring</td>
<td>✔️</td>
<td>✗</td>
</tr>
</tbody>
</table>

**CM:** Do types protect all channels?
A Toolbox to Measure Type Guarantees

<table>
<thead>
<tr>
<th></th>
<th>Guarded</th>
<th>Transient</th>
</tr>
</thead>
<tbody>
<tr>
<td>complete monitoring</td>
<td>✔</td>
<td>✗</td>
</tr>
<tr>
<td>blame soundness</td>
<td>✔</td>
<td>✗</td>
</tr>
<tr>
<td>blame completeness</td>
<td>✔</td>
<td>✗</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Guarded</th>
<th>C</th>
<th>F</th>
<th>Transient</th>
<th>A</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>type soundness</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>complete monitoring</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>blame soundness</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>blame completeness</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>error preorder</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# A Toolbox to Measure Type Guarantees

<table>
<thead>
<tr>
<th></th>
<th>Guarded</th>
<th>C</th>
<th>F</th>
<th>Transient</th>
<th>A</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>type soundness</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>y</td>
<td>✓</td>
</tr>
<tr>
<td>complete monitoring</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>blame soundness</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>h</td>
<td>✓</td>
</tr>
<tr>
<td>blame completeness</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>error preorder</td>
<td>Guarded</td>
<td>C</td>
<td>F</td>
<td>Transient</td>
<td>A</td>
<td>E</td>
</tr>
</tbody>
</table>

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 \textbf{Guarded} and \textbf{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**

```python
function h(x)
  if 0 < fst(x)
    pumpkin
  else
    fish
p = ClickPlot(h)
p.show()
```

```java
class ClickPlot
  init(onClick)
    # set up
    mouseHandler(evt)
      onClick(evt)
    # add image
    show()
    # display
```

**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