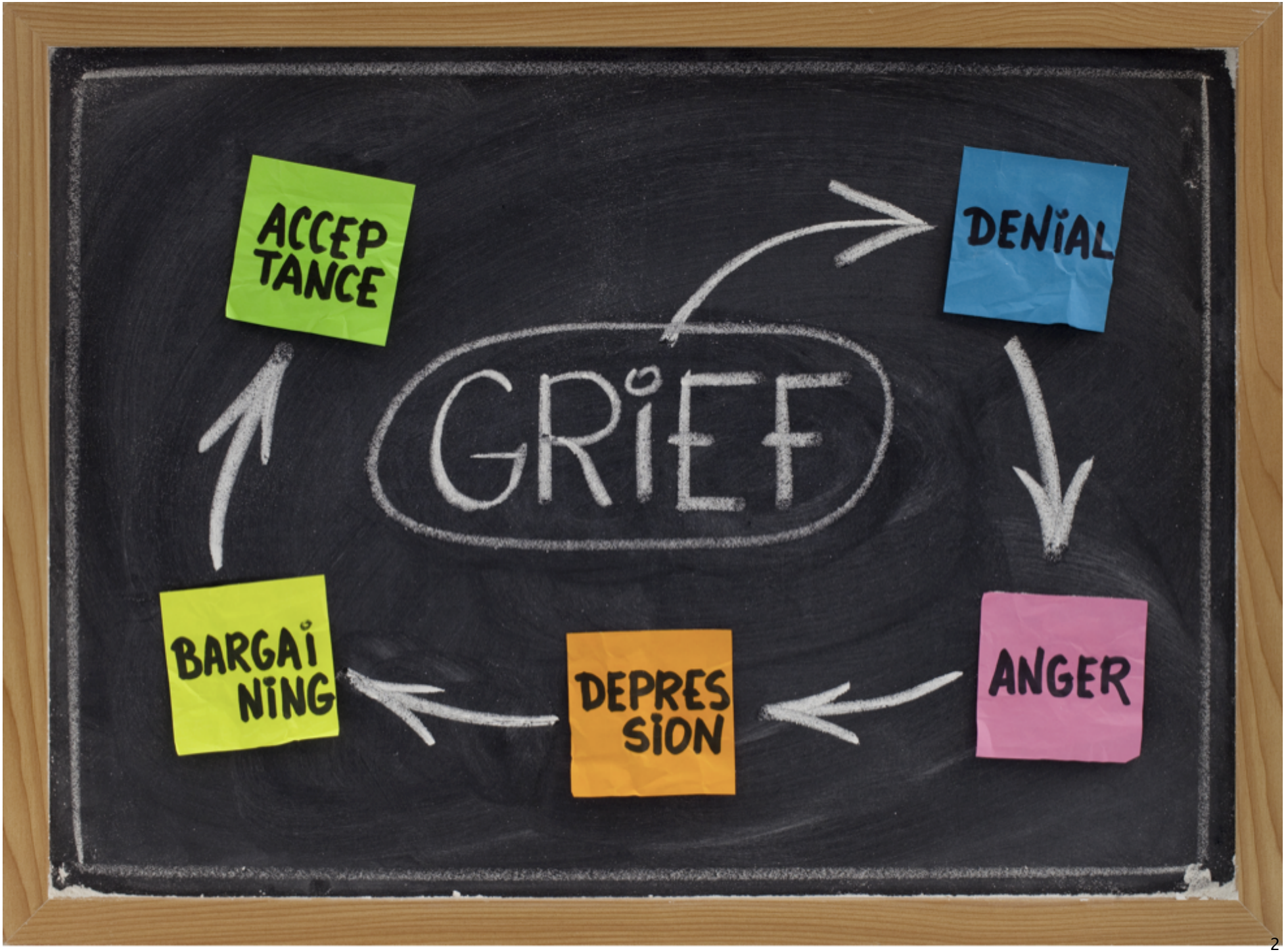
MEASURING RETICULATED PYTHON

Ben Greenman, Northeastern University

with Zeina Migeed


GRIEF
DENIAL
ANGER
DEPRESSION
BARGAINING
ACCEPTANCE

Reticulated Python

- Gradual typing for Python [DLS 2014]
- Static type checking
- Dynamic type enforcement
- Formal model is type is sound [POPL 2017]

Example Program

```
def f(n):
  return n*(n+1) // 2

def get_numbers(count):
  nums = []
  for i in range(1, 1+count):
    nums.append(f(i))
  return nums

get_numbers(4)
# [1, 3, 6, 10]
```

Example Program, Fully-Typed

```
def f(n:Int)->Int:
  return n*(n+1) // 2

def get_numbers(count:Int)->List(Int):
  nums = []
  for i in range(1, 1+count):
    nums.append(f(i))
  return nums

get_numbers(4)
# [1, 3, 6, 10]
```

Example Program, Partially Typed

```
def f(n:Int):
  return n*(n+1) // 2

def get_numbers(count)->List(Int):
  nums = []
  for i in range(1, 1+count):
    nums.append(f(i))
  return nums

get_numbers(4)
# [1, 3, 6, 10]
```

Example Program, Partially Typed

```
def f(n:Int):
  return n*(n+1) // 2

def get_numbers(count)->List(Int):
  nums = []
  for i in range(1, 1+count):
    nums.append(f(i))
  return nums

get_numbers(4)
# [1, 3, 6, 10]

f("not a number")
# Static type error
```

Example Program, Partially Typed

```
def f(n:Int):
  return n*(n+1) // 2

def get_numbers(count)->List(Int):
  nums = []
  for i in range(1, 1+count):
    nums.append(f(i))
  return nums

get_numbers(4)
# [1, 3, 6, 10]

f("not a number")
# Static type error

get_numbers("not a number")
# Dynamic type error
```

Reticulated Python

- Gradual typing for Python [DLS 2014]
- Static type checking
- Dynamic type enforcement
- Formal model is type is sound [POPL 2017]

STAGE I: GRIEF

Something Weird

```
def f(n:Int):
  return n*(n+1) // 2

def get_numbers(count)->List(Int):
  nums = []
  for i in range(1, 1+count):
    nums.append(f(i))
  return nums

get_numbers(4)
```

Something Weird

```
def f(n:Int):
  return n*(n+1) // 2

def get_numbers(count)->List(Int):
  nums = []
  for i in range(1, 1+count):
    nums.append(f)  # typo!
  return nums

get_numbers(4)
```

Something Weird

```
def f(n:Int):
  return n*(n+1) // 2

def get_numbers(count)->List(Int):
  nums = []
  for i in range(1, 1+count):
    nums.append(f)  # typo!
  return nums

get_numbers(4)

# [<fun>, <fun>, <fun>, <fun>]
```

Something Weird

```
def f(n:Int):
  return n*(n+1) // 2

def get_numbers(count)->List(Int):
  nums = []
  for i in range(1, 1+count):
    nums.append(f)  # typo!
  return nums

get_numbers(4)

# [<fun>, <fun>, <fun>, <fun>]

def apply_first(funs):
  return funs[0](10)
apply_first(get_numbers(4))
# 55
```

Another Something Weird

```
@fields({"dollars": Int
         ,"cents": Int})
class Cash:
  dollars = 0
  cents = 0

  def add_dollars(self, dollars):
    self.dollars += dollars
```

Another Something Weird

```
@fields({"dollars": Int
         ,"cents": Int})
class Cash:
  dollars = 0
  cents = 0

  def add_dollars(self, dollars):
    self.dollars += dollars


def get_cash()->Cash:
  c = Cash()
  c.add_dollars(3.14159)
  return c

get_cash()
# Cash(3.14159, 0)
```

STAGE II: DENIAL

Type Soundness

If `e` has type `T`, then either:

- `e` reduces to a value `v` with type `T`
- `e` raises an error due to a partial primitive
- `e` diverges

Reticulated Type Soundness

If `e` has type `T`, then either:

- `e` reduces to a value `v` with type `⌊T⌋`
  - e.g. `⌊Int->Int⌋ = ->`
- `e` raises a blame error
- `e` diverges

Big Types in Little Runtime

Open-World Soundness and Collaborative Blame for Gradual Type Systems

Michael M. Vitousek Cameron Swords Jeremy G. Siek

Indiana University, USA

{mvitouse,cswords,jsiek}@indiana.edu

Abstract

Gradual typing combines static and dynamic typing in the same language, offering programmers the error detection and strong guarantees of static types and the rapid prototyping and flexible programming idioms of dynamic types. Many gradually typed languages are implemented by translation into an untyped target language (e.g., Typed Clojure, TypeScript, Gradualtalk, and Reticulated Python). For such languages, it is desirable to sup-

typed code interacts: the consistency relation plays the role that type equality usually does in the type system. Types are consistent if they are equal up to the presence of $\star$.

Most existing gradually typed languages operate by translating a surface language program into an underlying target language, which is then executed. For many gradually-typed systems such as Typed Racket and TypeScript, the target language is a dynamically typed programming language, and gradually-typed programs are

Corollary 5.5.1 (Type soundness). *If* $\emptyset \vdash e_s \rightsquigarrow e : T$ *then* $\emptyset; \emptyset \vdash e : \lfloor T \rfloor$ *and either:*

- $\langle e, \emptyset, \emptyset \rangle \longrightarrow^* \langle v, \sigma, \mathcal{B} \rangle$ *and* $\emptyset; \Sigma \vdash v : \lfloor T \rfloor$ *and* $\Sigma \vdash \sigma$, *or*
- $\langle e, \emptyset, \emptyset \rangle \longrightarrow^* \textsc{Blame}(\mathcal{L})$, *or*
- *for all* ς *such that* $\langle e, \emptyset, \emptyset \rangle \longrightarrow^* \varsigma$, *have that* $\varsigma = \langle e', \sigma, \mathcal{B} \rangle$ *and exists* ς' *such that* $\langle e', \sigma, \mathcal{B} \rangle \longrightarrow \varsigma'$.

$\boxed{\lfloor T \rfloor = S}$

$$\lfloor \star \rfloor = \star \qquad \lfloor \mathsf{int} \rfloor = \mathsf{int}$$
$$\lfloor T_1 \to T_2 \rfloor = \to \qquad \lfloor \mathsf{ref}\, T \rfloor = \mathsf{ref}$$

$\boxed{T \rhd T}$

$$\mathsf{ref}\, T \rhd \mathsf{ref}\, T \qquad \star \rhd \mathsf{ref}\, \star$$
$$T_1 \to T_2 \rhd T_1 \to T_2 \qquad \star \rhd \star \to \star$$

$\boxed{T \sim T}$

$$\mathsf{int} \sim \mathsf{int} \qquad \star \sim T \qquad T \sim \star$$

$$\frac{T_1 \sim T_2}{\mathsf{ref}\, T_1 \sim \mathsf{ref}\, T_2} \qquad \frac{T_1 \sim T_3 \qquad T_2 \sim T_4}{T_1 \to T_2 \sim T_3 \to T_4}$$

Figure 3. Translation from $\lambda^{\star}_{\to}$ to $\lambda^{\Downarrow}_{\ell}$.

STAGE III: ANGER, BARGAINING, DEPRESSION

What are Reticulated Types Good For?

What are Reticulated Types Good For?

- Protect invariants?

What are Reticulated Types Good For?

- Protect invariants?
- Reliable documentation?

What are Reticulated Types Good For?

- Protect invariants?
- Reliable documentation?
- Enable optimizations?

What are Reticulated Types Good For?

- Protect invariants?
- Reliable documentation?
- Enable optimizations?

Any untyped code

=>

No compositional reasoning!

STAGE IV: ACCEPTANCE

Interoperability & Performance

Interoperability

```
def get_numbers(count)->List(Int):
    ....
    return nums
```

- `List` is mutable, standard approach is to proxy

Interoperability

```
def get_numbers(count)->List(Int):
    ....
    return proxy(nums, List(Int))
```

Interoperability

```
def get_numbers(count)->List(Int):
    ....
    return proxy(nums, List(Int))
```

- The proxy must be compatible with existing code

Interoperability

```
def get_numbers(count)->List(Int):
    ....
    return proxy(nums, List(Int))
```

- The proxy must be compatible with existing code

```
nums.append(....)

len(nums)

nums is nums
```

Performance

```
def get_numbers(count)->List(Int):
    ....
    return proxy(nums, List(Int))
```

Performance

```
def get_numbers(count)->List(Int):
    ....
    return proxy(nums, List(Int))
```

- Allocation cost

Performance

```
def get_numbers(count)->List(Int):
    ....
    return proxy(nums, List(Int))
```

- Allocation cost
- Traverse, recursively proxy

Performance

```
def get_numbers(count)->List(Int):
    ....
    return proxy(nums, List(Int))
```

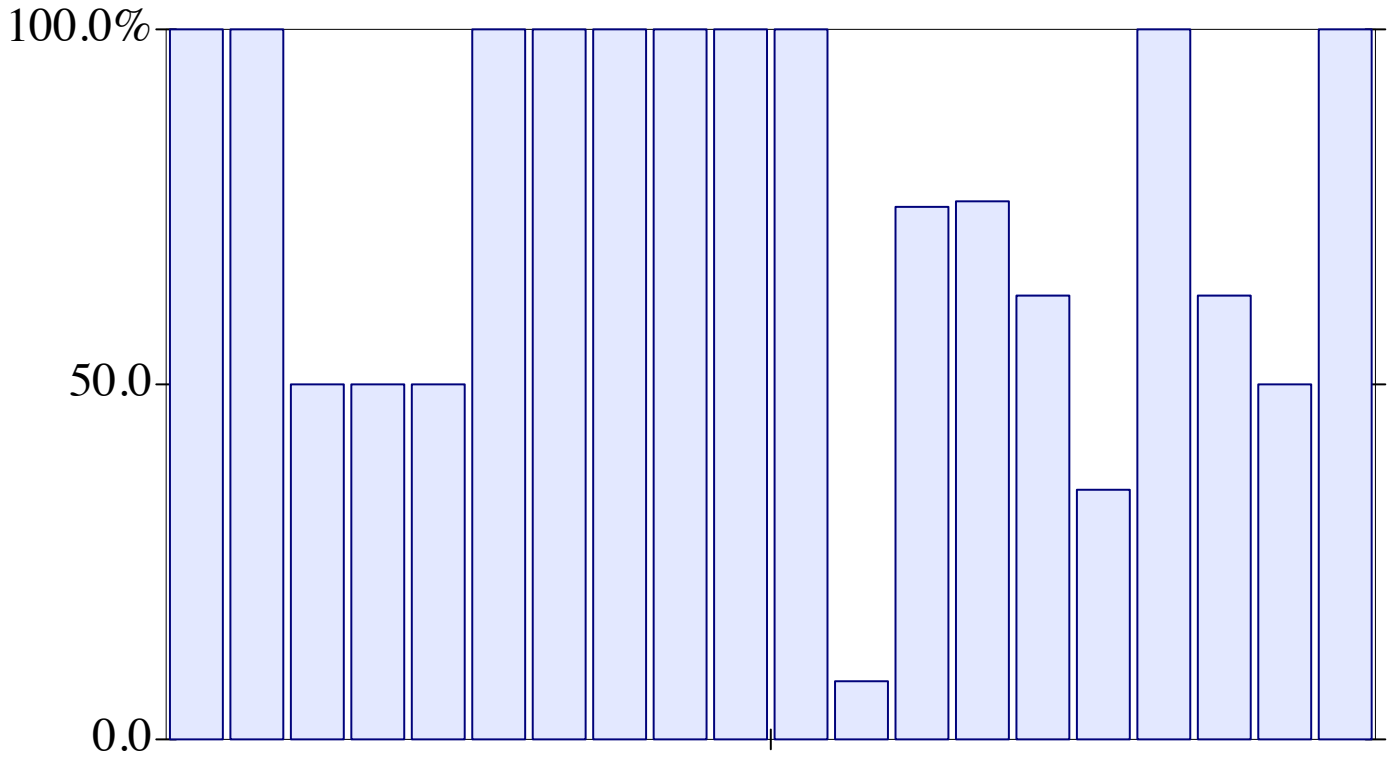
- Allocation cost
- Traverse, recursively proxy
- Interpose on future operations

Measuring Typed Racket

- 20 programs
- Measured all gradually-typed configurations
- How many 20-deliverable?

Measuring Typed Racket

- 20 programs
- Measured all gradually-typed configurations
- How many 20-deliverable?



Measuring Typed Racket

Worst-Case Overhead

acquire	5	quadBG	4
dungeon	10	quadMB	139
forth	27	sieve	43
fsm	1527	snake	32
fsmoo	233	suffixtree	29
gregor	2	synth	47
kcfa	5	take5	1
lnm	1	tetris	34
mbta	1	zombie	292
morsecode	1	zordoz	1

Measuring Typed Racket

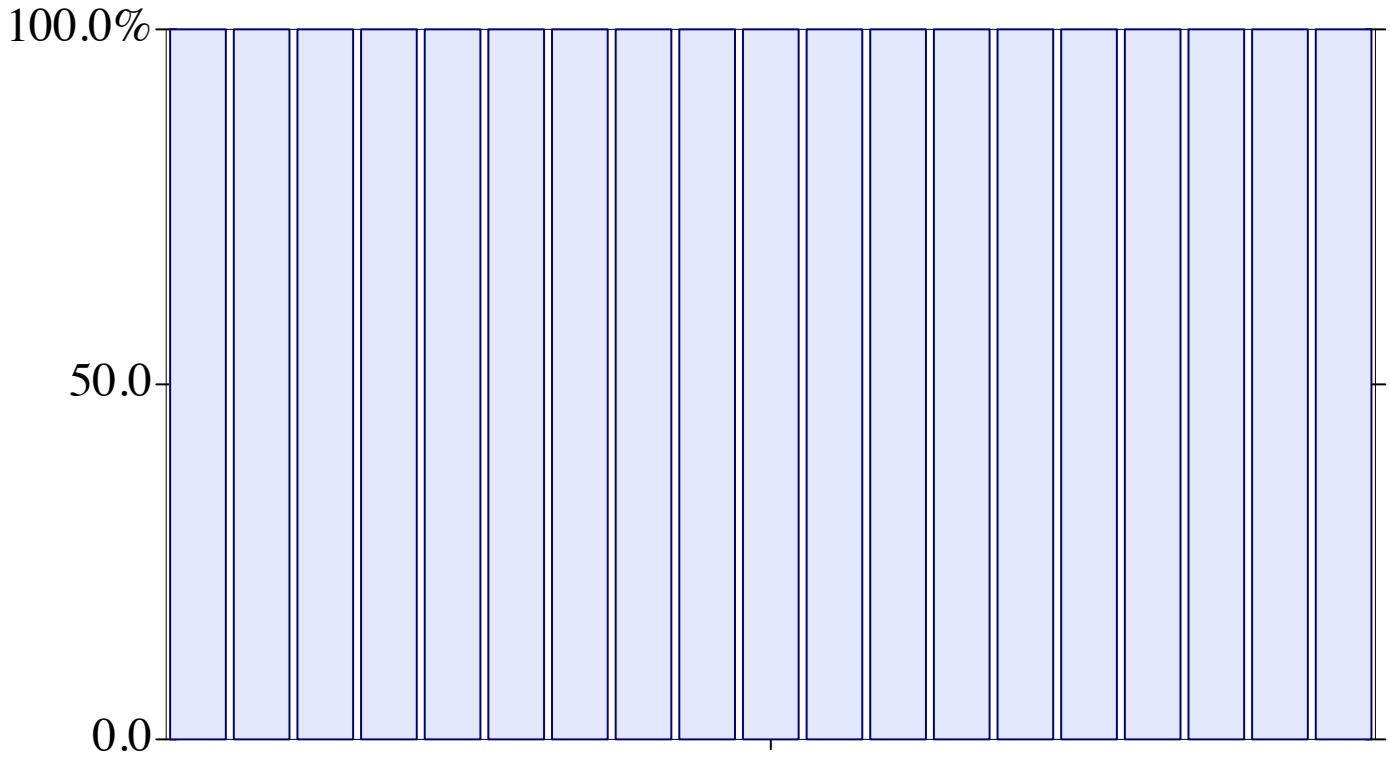
Frequently an order-of-magnitude slowdown

Measuring Reticulated

- 19 **different** programs
- Measured all *function-level* configurations
- How many 20-deliverable?

Measuring Reticulated

- 19 **different** programs
- Measured all *function-level* configurations
- How many 20-deliverable?

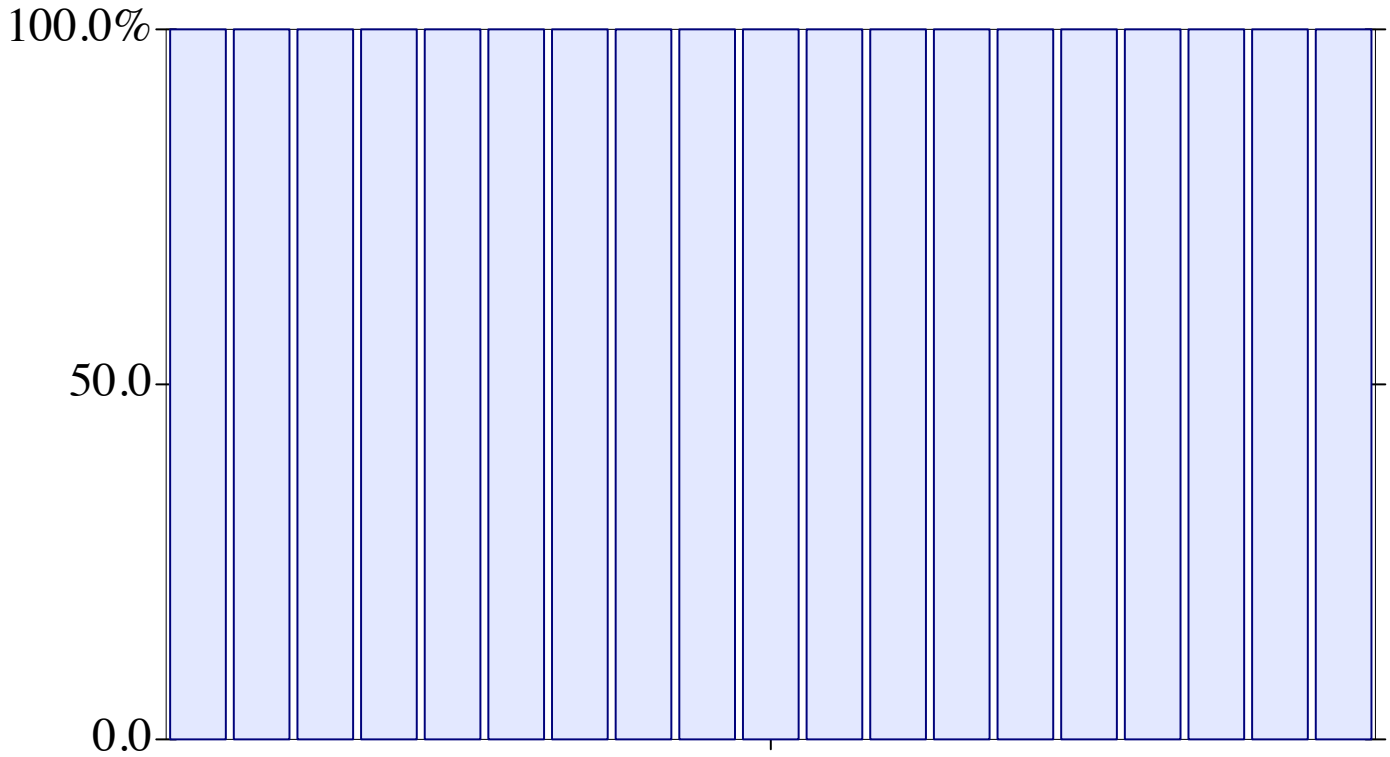


Measuring Reticulated

- 19 **different** programs
- Measured all *function-level* configurations
- How many 10-deliverable?

Measuring Reticulated

- 19 **different** programs
- Measured all *function-level* configurations
- How many 10-deliverable?



Measuring Reticulated

Worst-Case Overhead

futen	1	meteor	2
http2	3	nbody	1
slowSHA	2	nqueens	1
call_method	7	pidigits	1
call_method_slots	8	pystone	2
call_simple	3	spectralnorm	8
chaos	3	Espionage	5
fannkuch	1	PythonFlow	7
float	3	take5	1
go	7		

Measuring Reticulated

Never an order-of-magnitude slowdown

STAGE V: MOVING ON

Moving On

Q1. Is Reticulated's soundness practical?

Q2. Can Typed Racket soundness be performant?

Q3. Is Typed Racket soundness portable?

Q4. Is there a useful, "efficient" Soundness 3.0?

Granularity

```
@fields({"x": Int, "y": Int})
class Point:
  x = 0
  y = 0

  def get_x(self:Point)->Int:
    return self.x
```