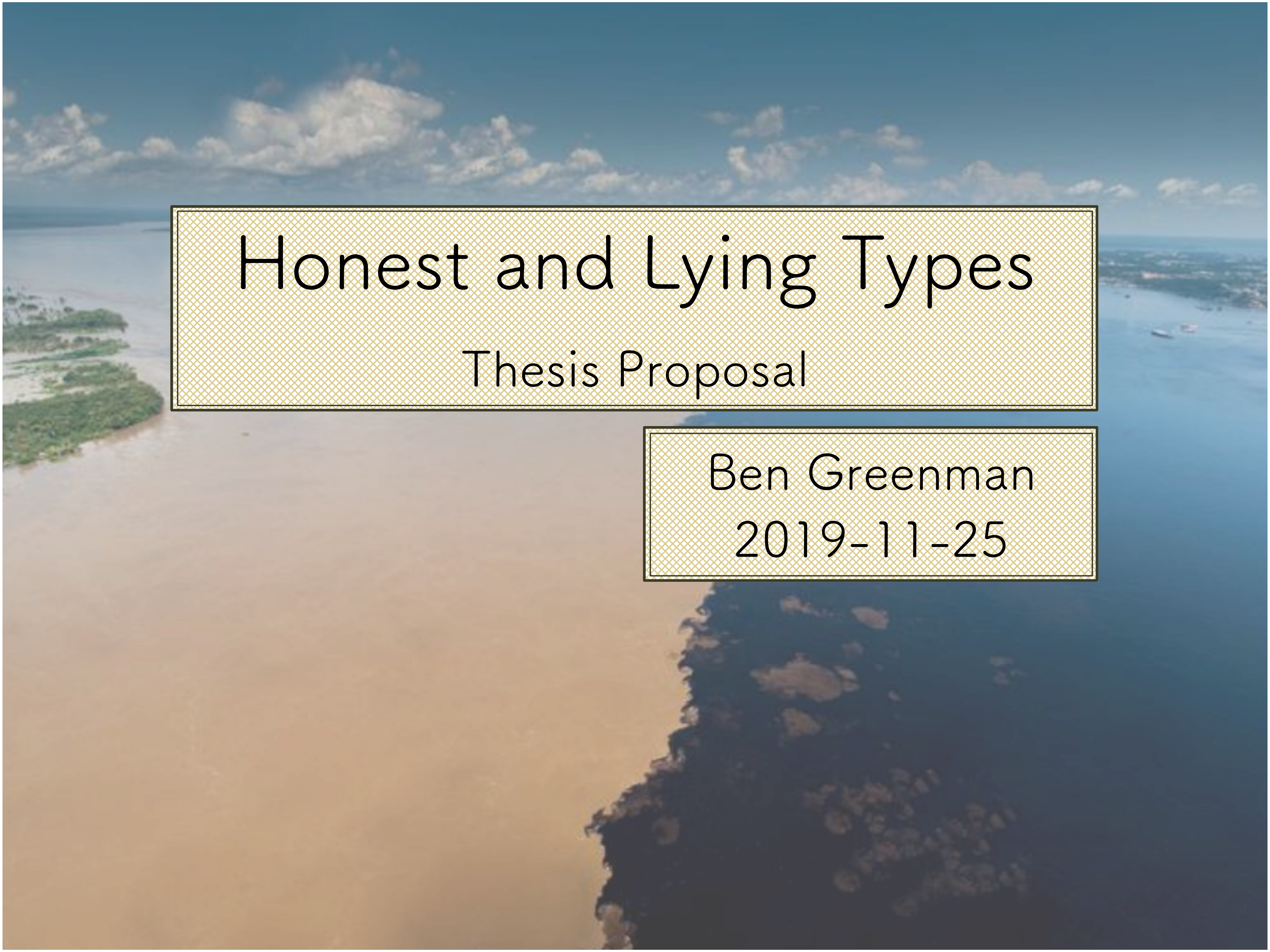


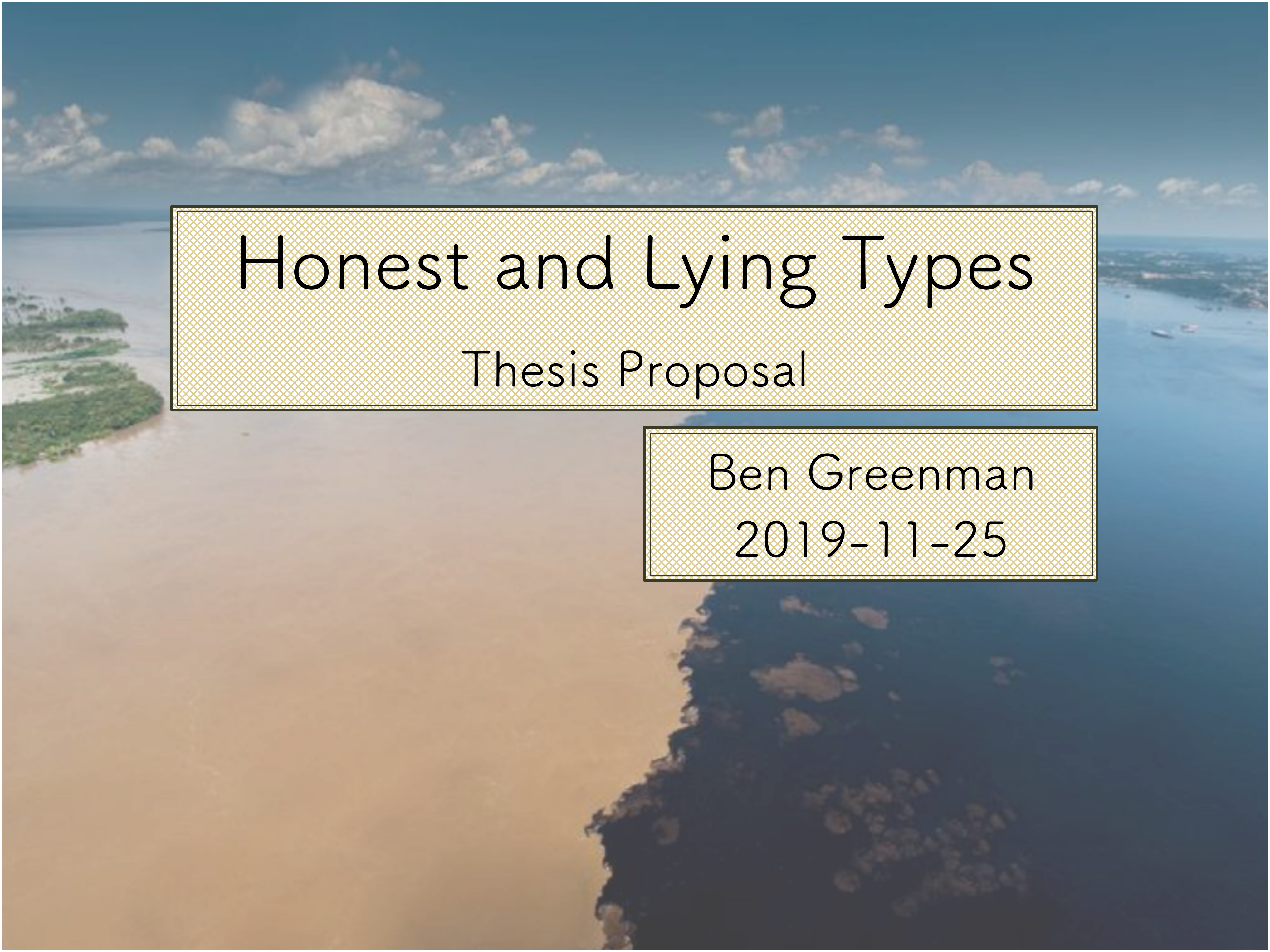
Honest and Lying Types

Thesis Proposal

Ben Greenman
2019-11-25

Committee:

1. Matthias Felleisen
2. Amal Ahmed
3. Jan Vitek
4. Shriram Krishnamurthi
5. Fritz Henglein
6. Sam Tobin-Hochstadt



Honest and Lying Types

Thesis Proposal

Ben Greenman
2019-11-25

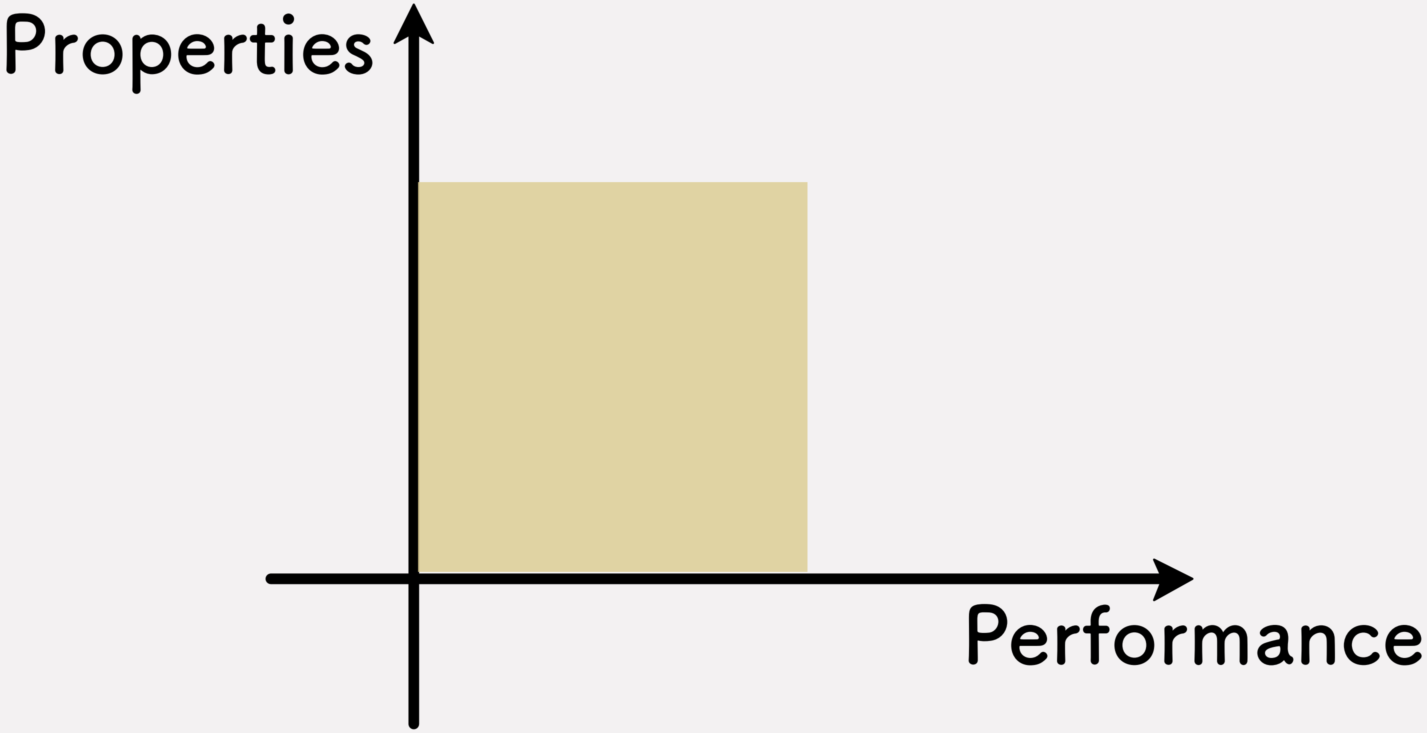
Thesis Proposal: "Gradual Typing"
November 18, 2019

Thesis Proposal: "Gradual Typing"
November 25, 2019

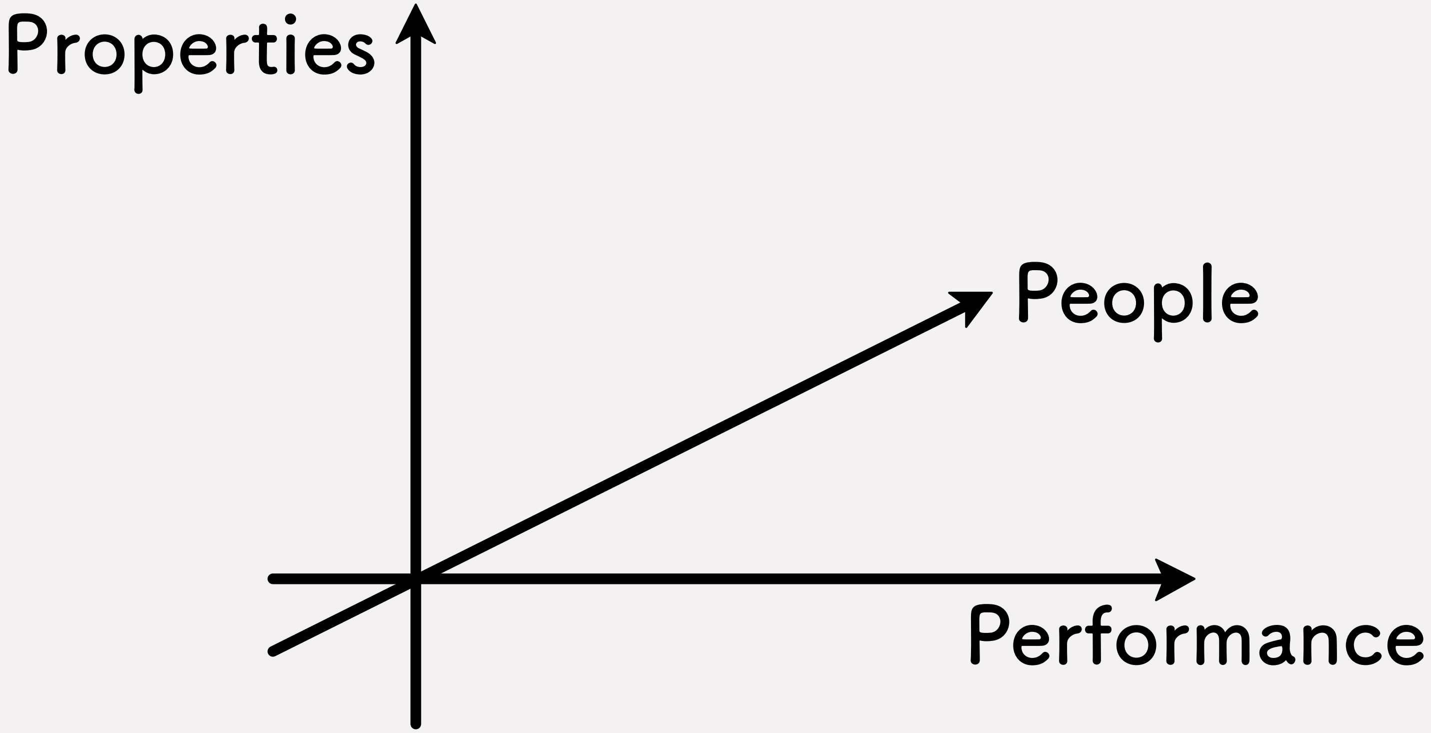
Last Week:

Properties

Today:



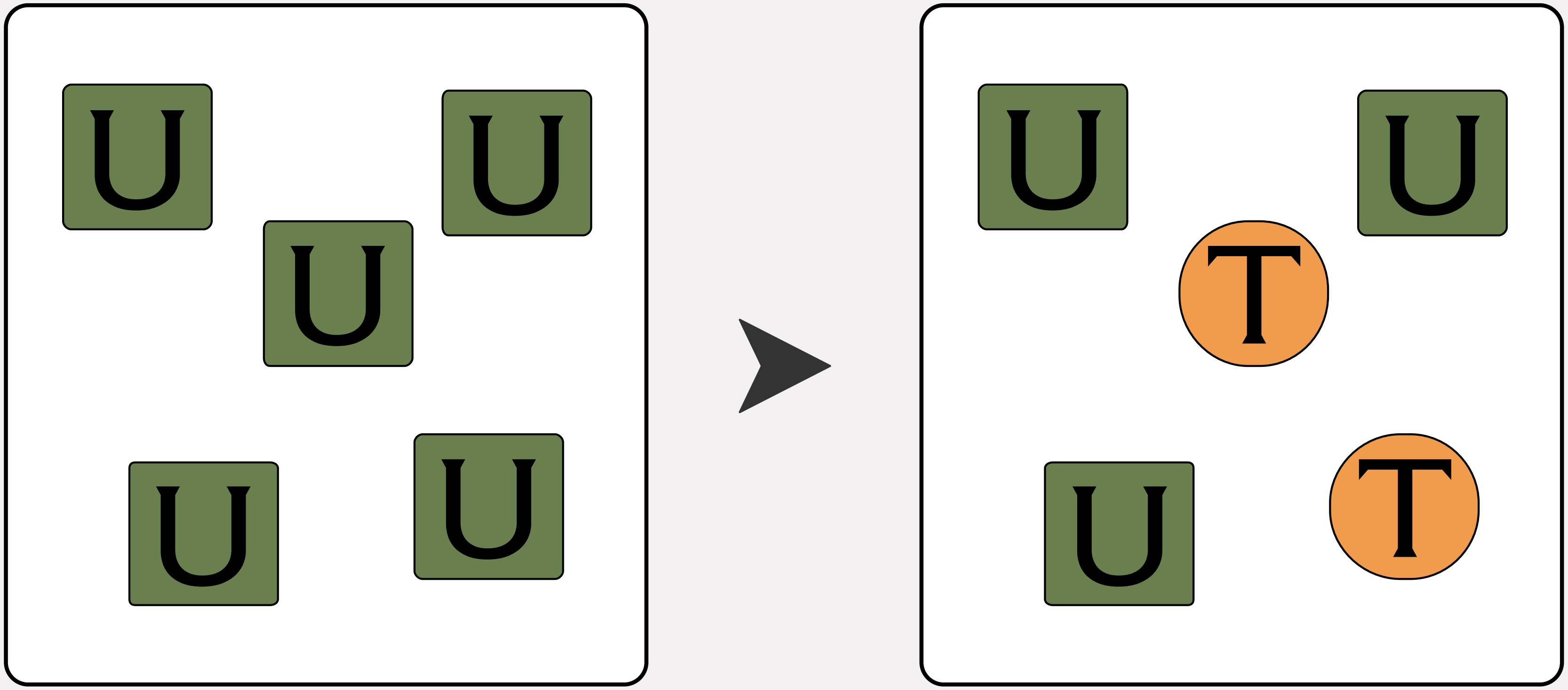
Future:



Migratory Typing

Migratory Typing

Add types to a dynamically-typed language



Untyped code

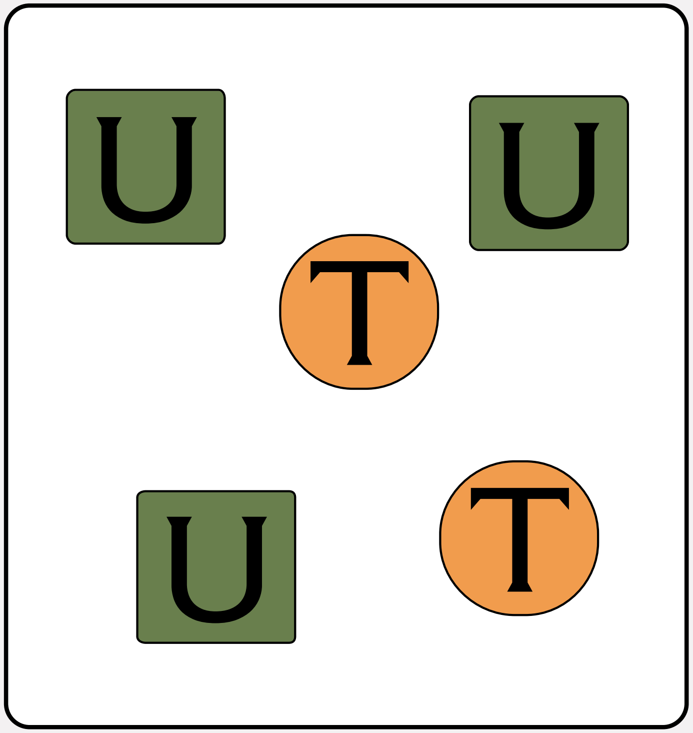
Mixed-Typed code

Migratory Typing

Add types to a dynamically-typed language

U = untyped code

T = simply-typed

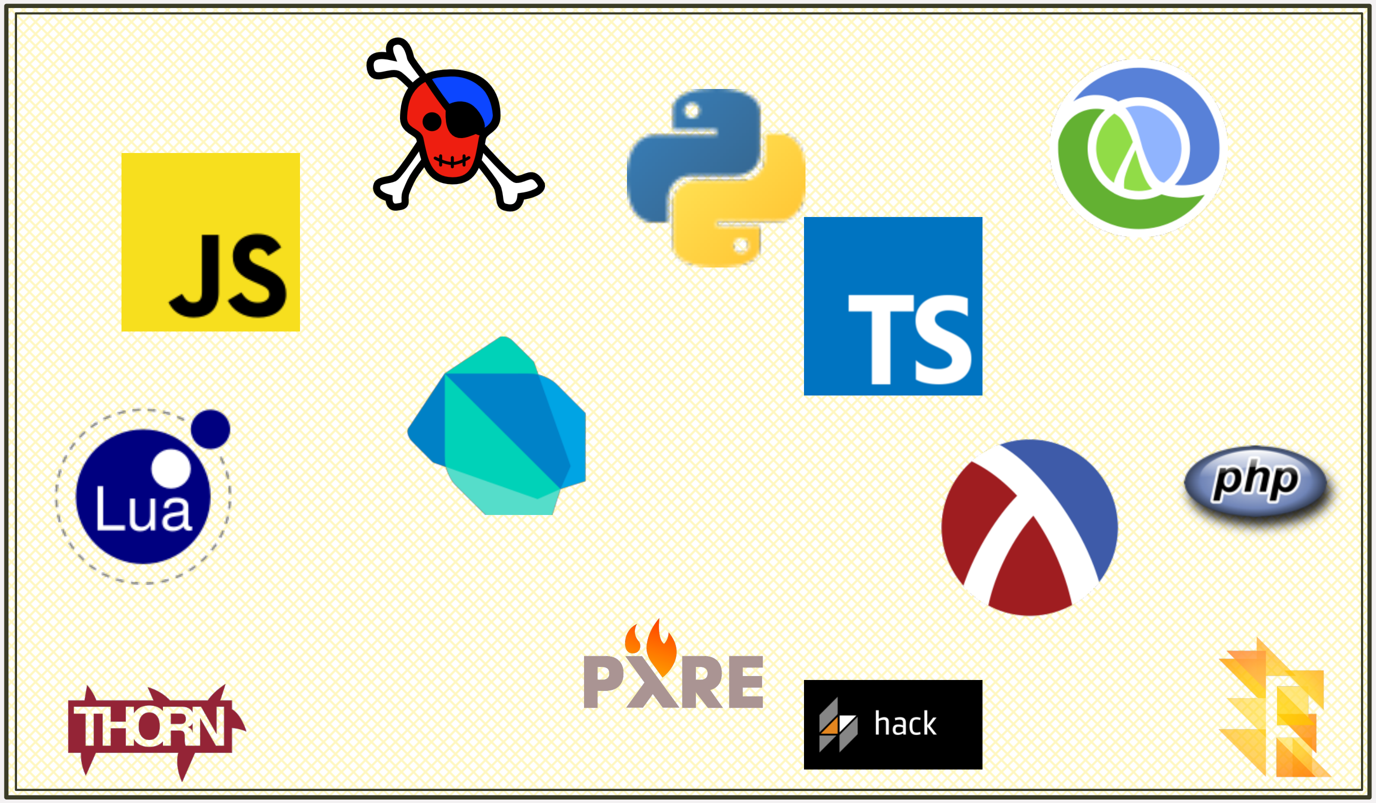


Mixed-Typed code

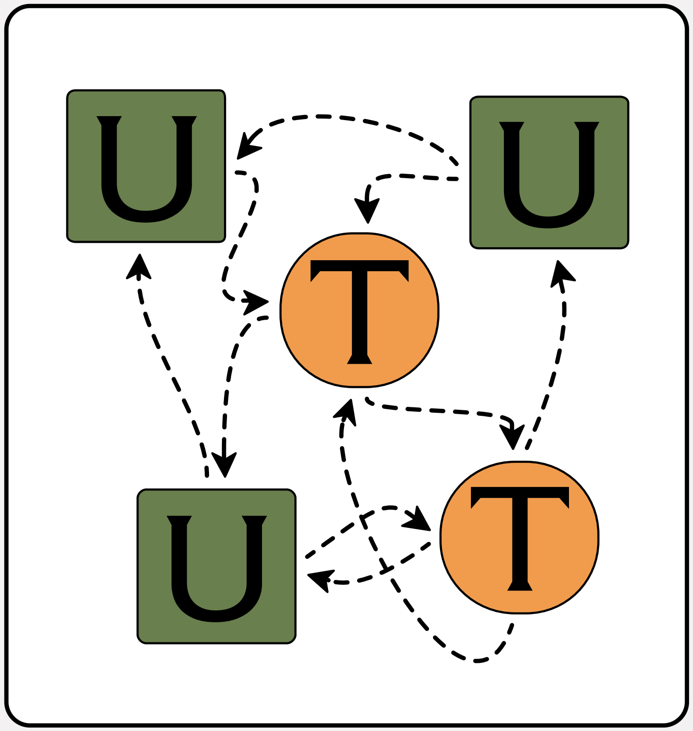
Motivation

Because lots of untyped code exists.

Landscape of Models and Implementations

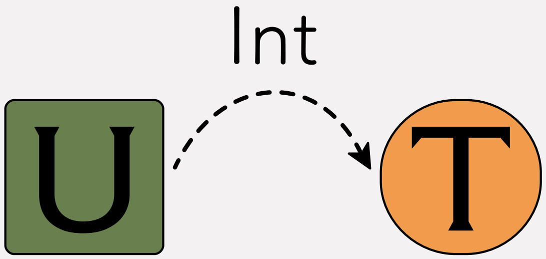


Challenge = Interoperability

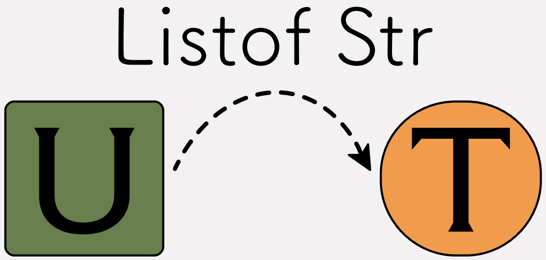


What do types mean when untyped values and typed values **interact**?

Challenge = Interoperability

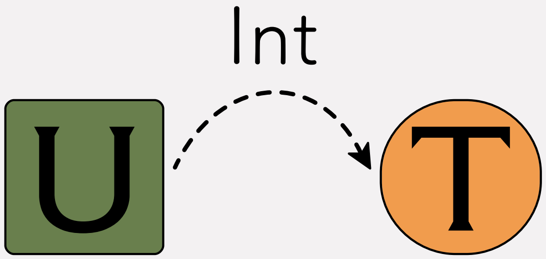


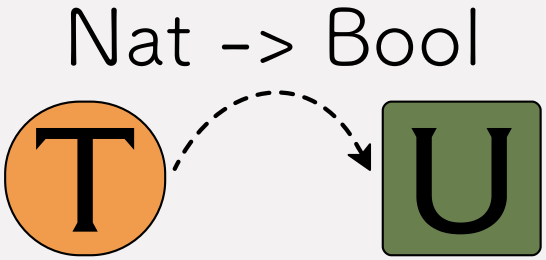
Int

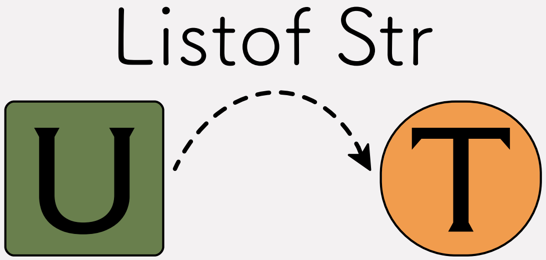


Listof Str

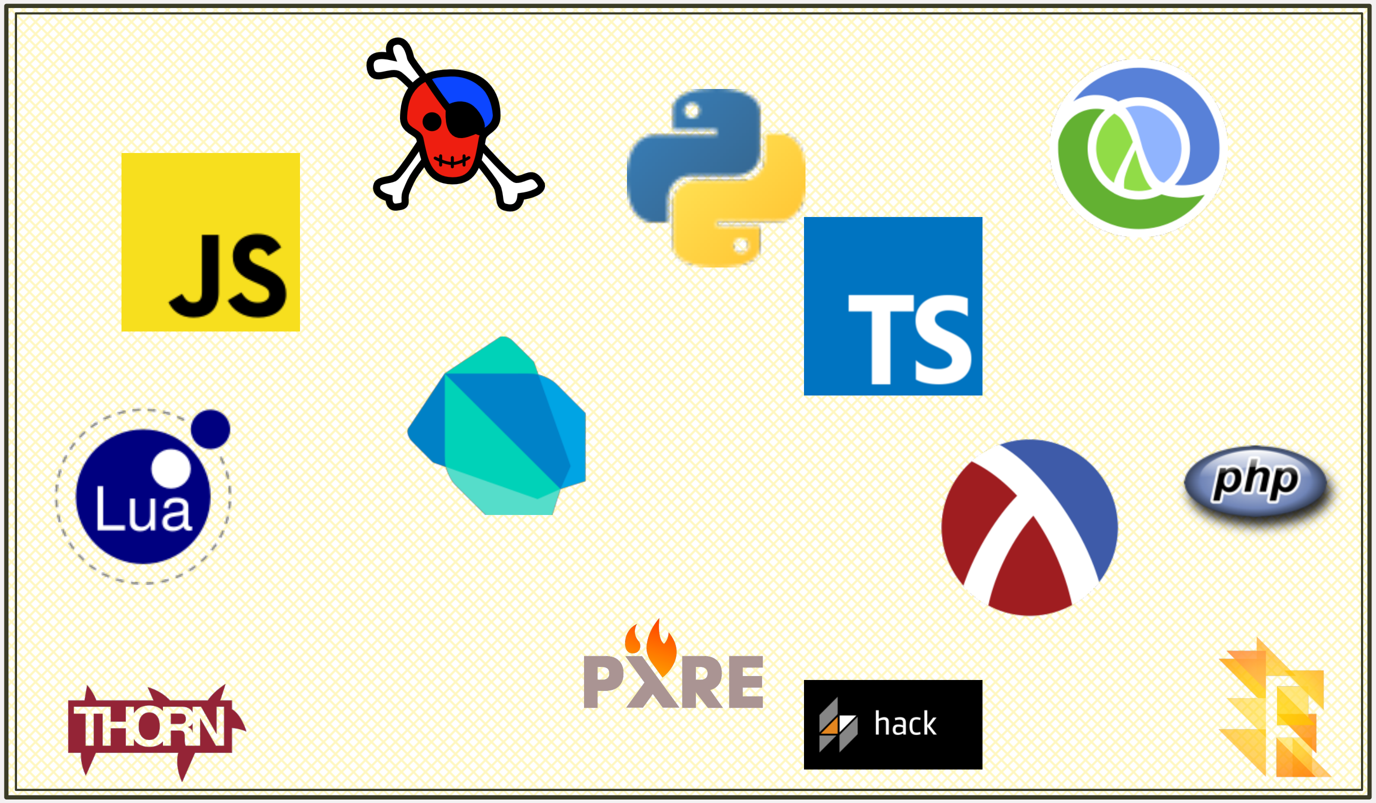
Challenge = Interoperability


Int

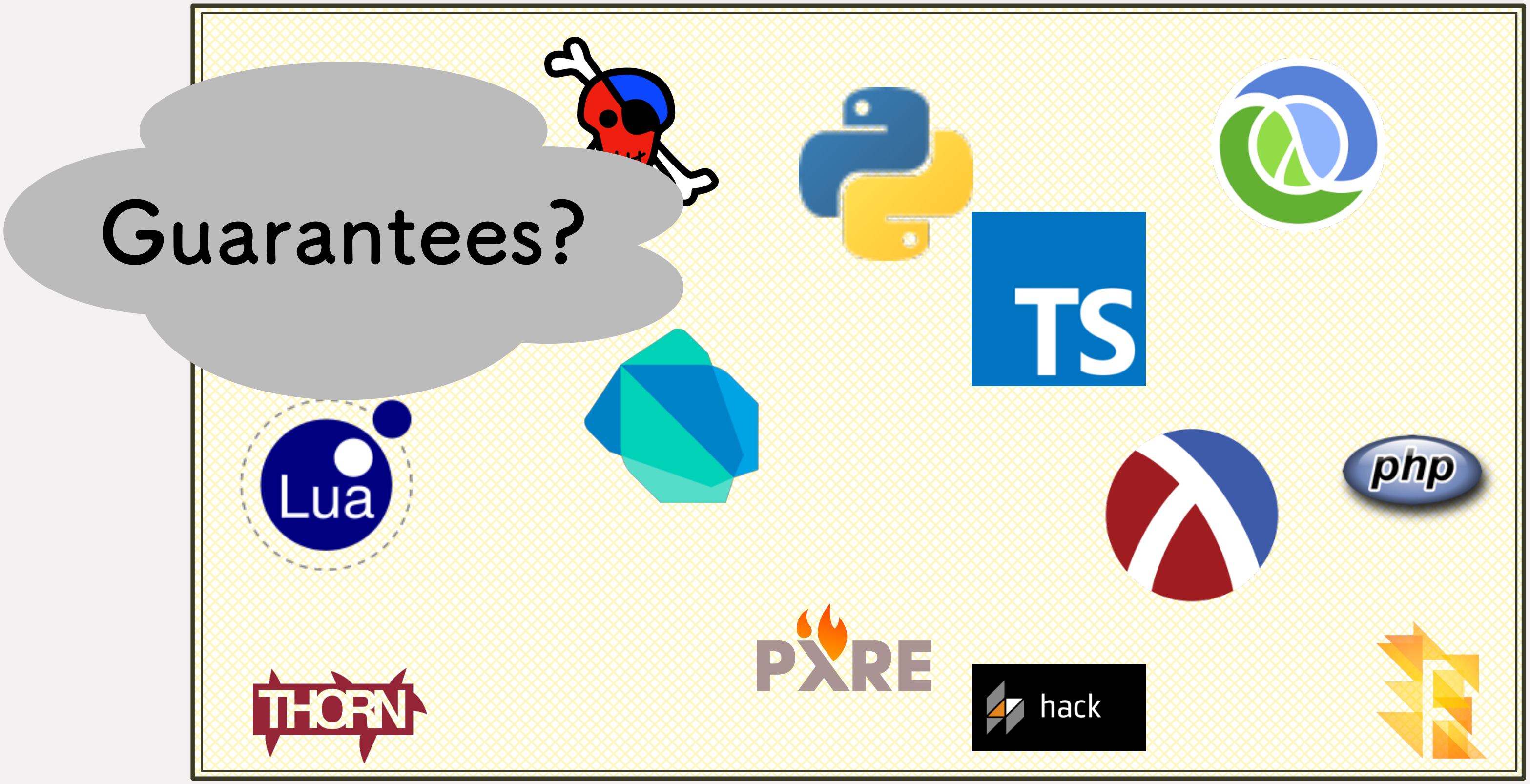

Nat -> Bool


Listof Str

Many Answers, Many Implementations

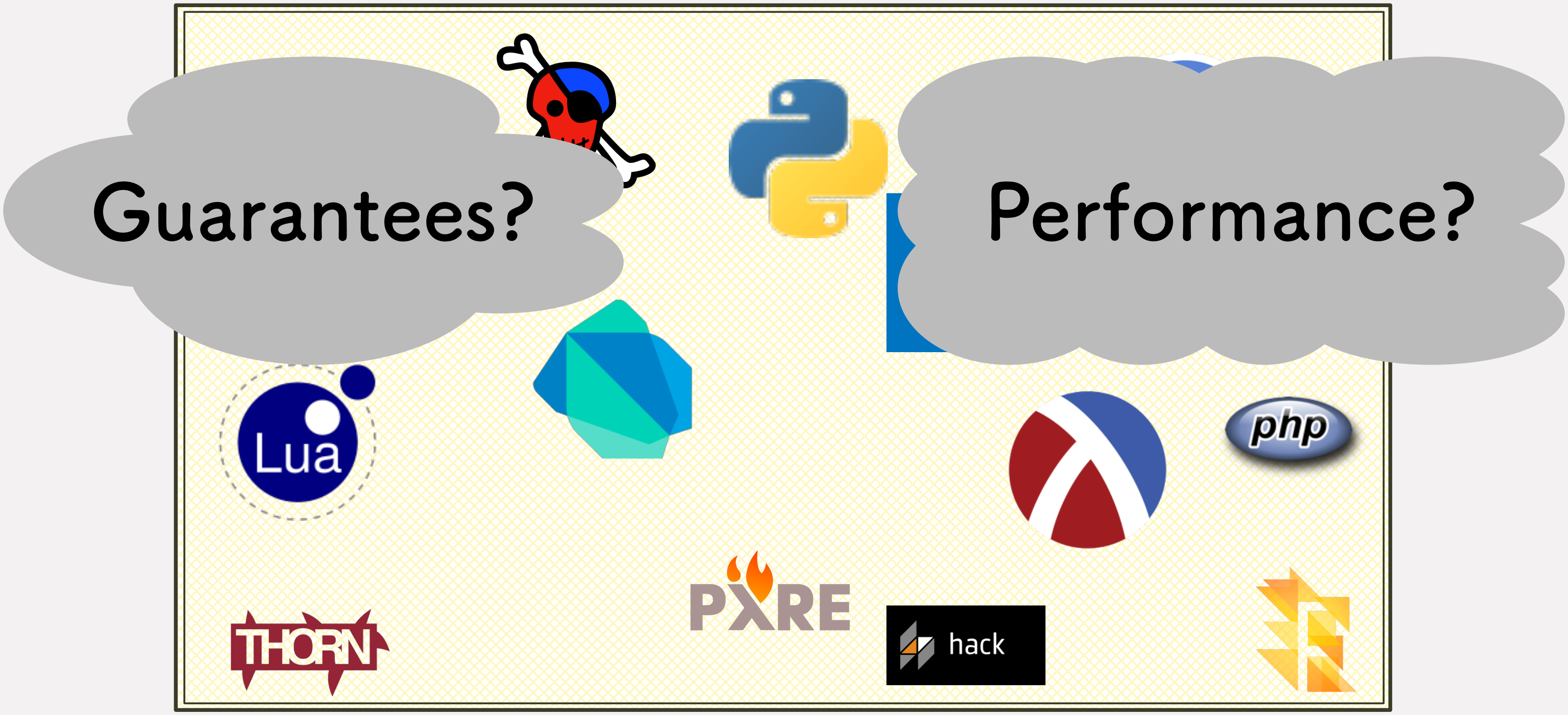


Many Answers, Many Implementations



Guarantees?

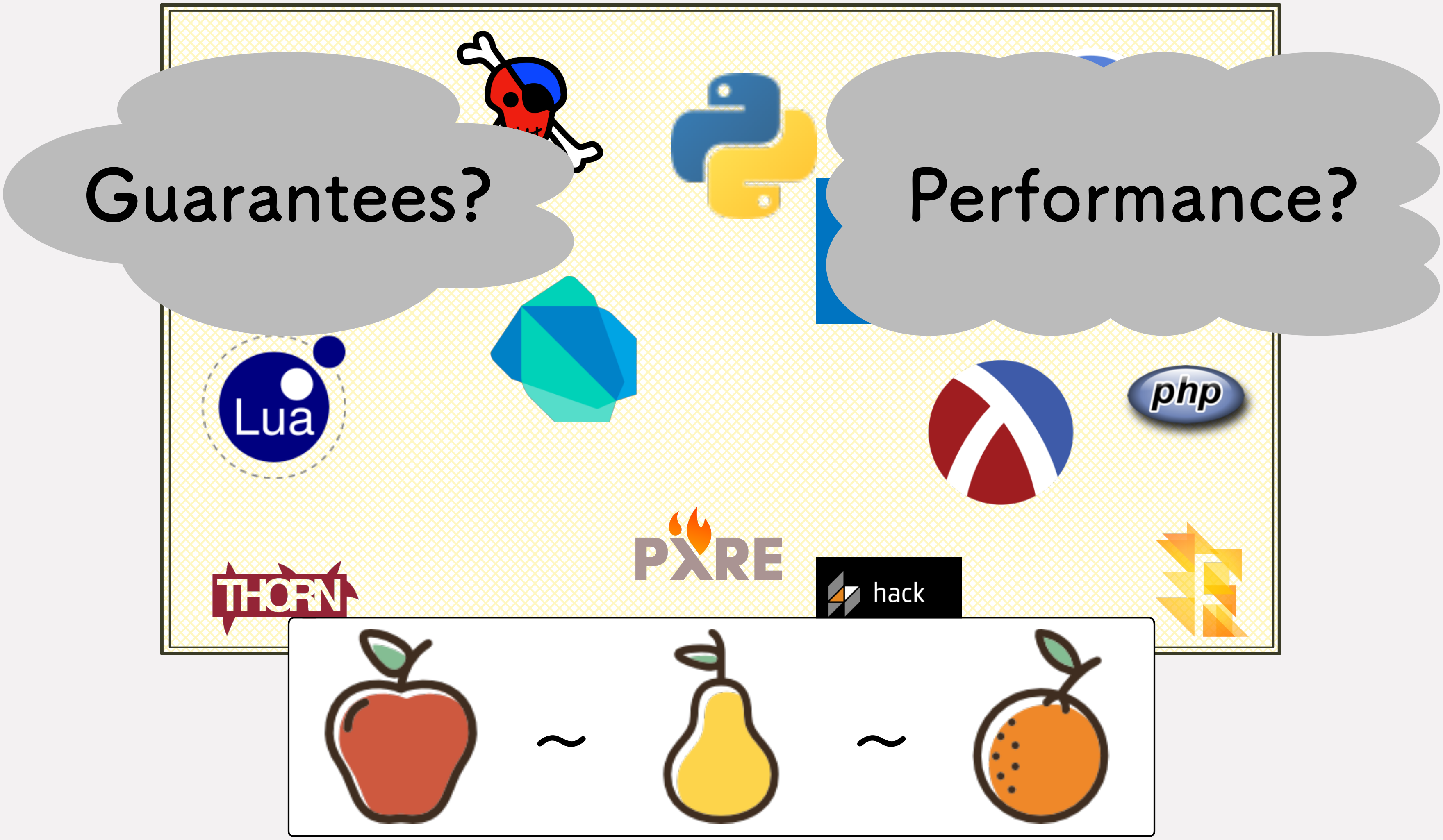
Many Answers, Many Implementations



Guarantees?

Performance?

Many Answers, Many Implementations



Guarantees?

Performance?

Icons made by Freepik from Flaticon.com

My Research

Research Agenda: Scientific Comparison

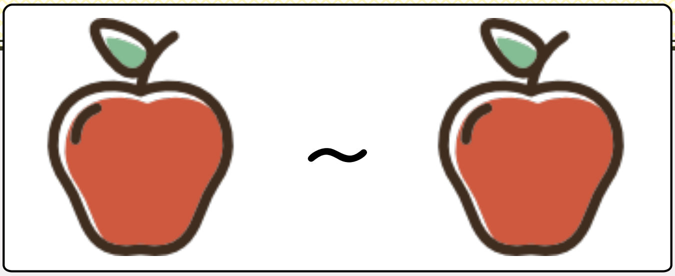
Research Agenda: Scientific Comparison

Guarantees $\lambda \rightarrow \tau$

one syntax, many semantics for what flows across channels

Performance

one syntax, many type-compilers



Icons made by Freepik from Flaticon.com

Research Agenda: Results so Far

Design Space Analysis

OOPSLA 19	Ben Greenman, Matthias Felleisen, and Christos Dimoulas
ICFP 18	Ben Greenman and Matthias Felleisen

Research Agenda: Results so Far

Design Space Analysis

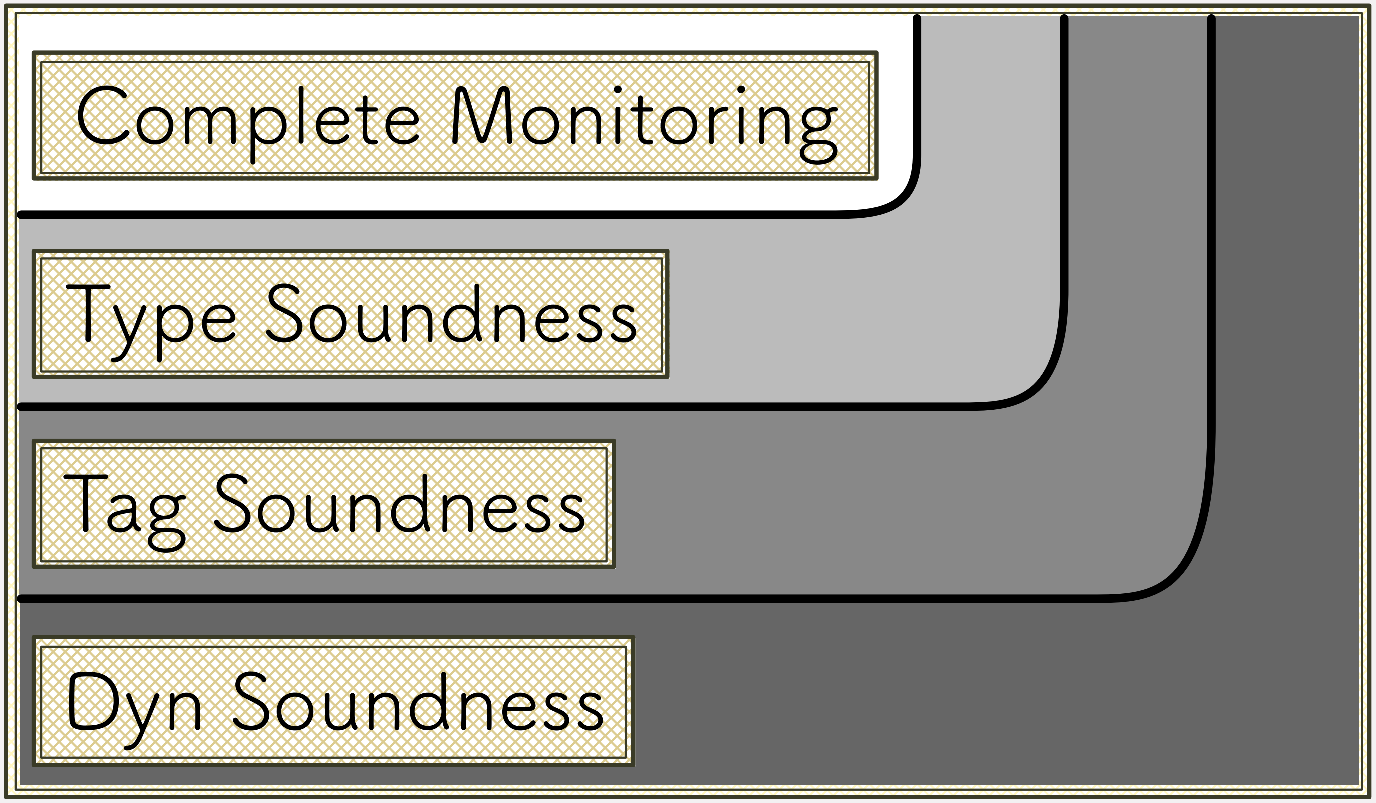
OOPSLA	19	Ben Greenman, Matthias Felleisen, and Christos Dimoulas
ICFP	18	Ben Greenman and Matthias Felleisen

Performance Evaluation

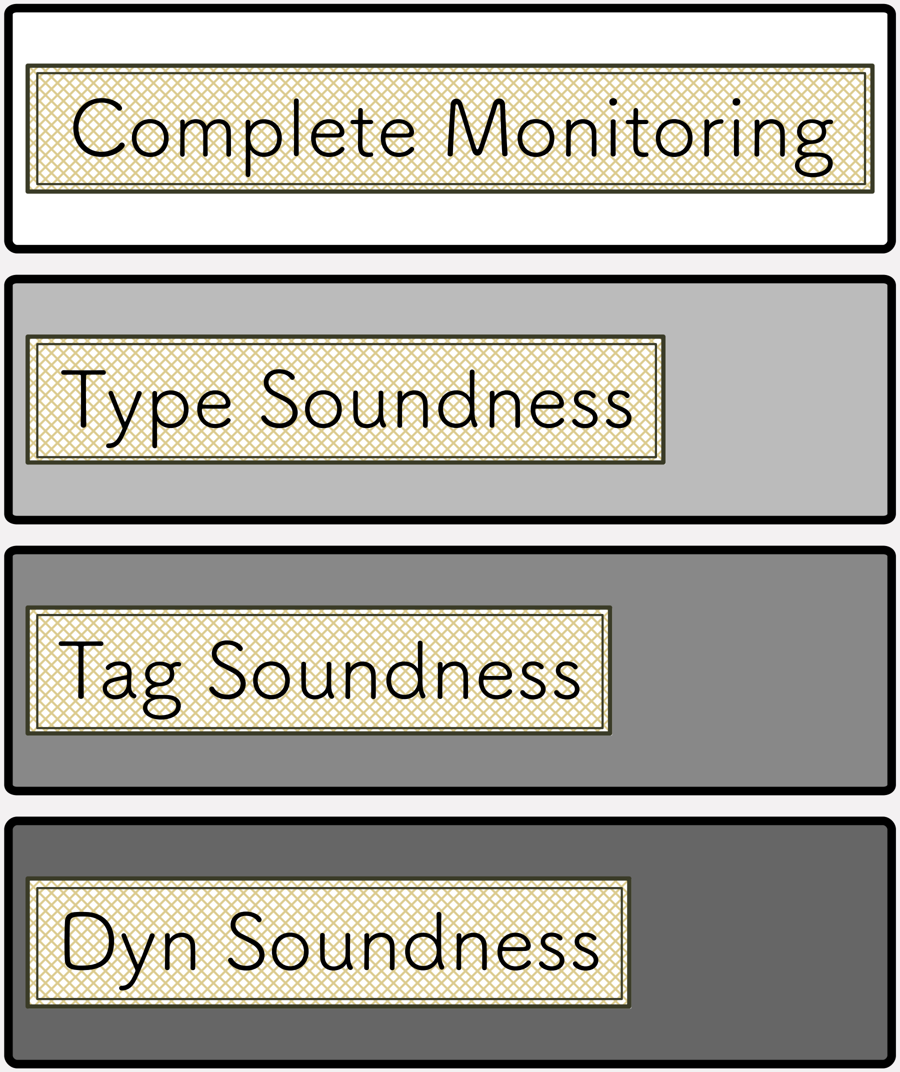
JFP	19	Ben Greenman, Asumu Takikawa, Max S. New, Daniel Feltey, Robert Bruce Findler, Jan Vitek, and Matthias Felleisen
PEPM	18	Ben Greenman and Zeina Migeed
POPL	16	Asumu Takikawa, Daniel Feltey, Ben Greenman, Max S. New, Jan Vitek, and Matthias Felleisen

Landscape: **Guarantees**

Landscape: **Guarantees**



(a *total spectrum*)



Complete Monitoring

types predict behavior

Type Soundness

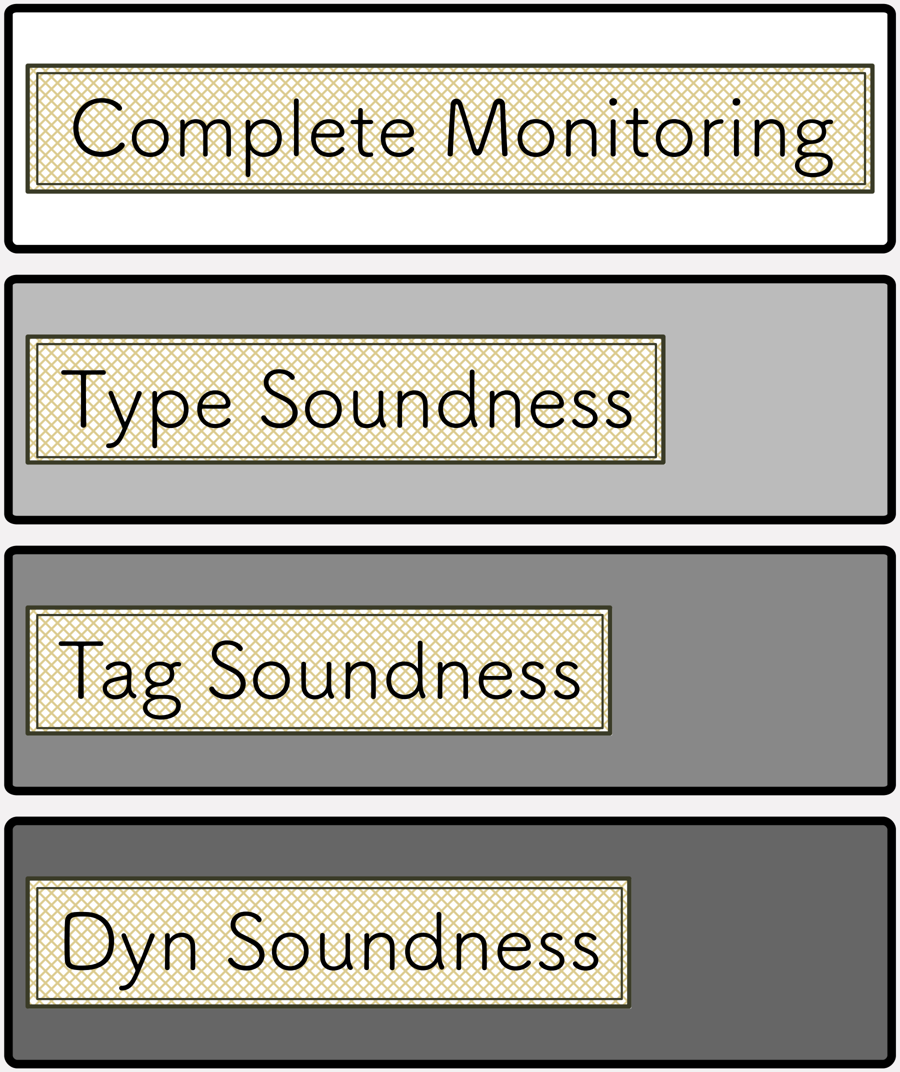
types predict behavior in typed code, nothing in untyped code

Tag Soundness

types predict shapes in typed code, nothing in untyped code

Dyn Soundness

types predict nothing



Complete Monitoring

types predict behavior

Type Soundness

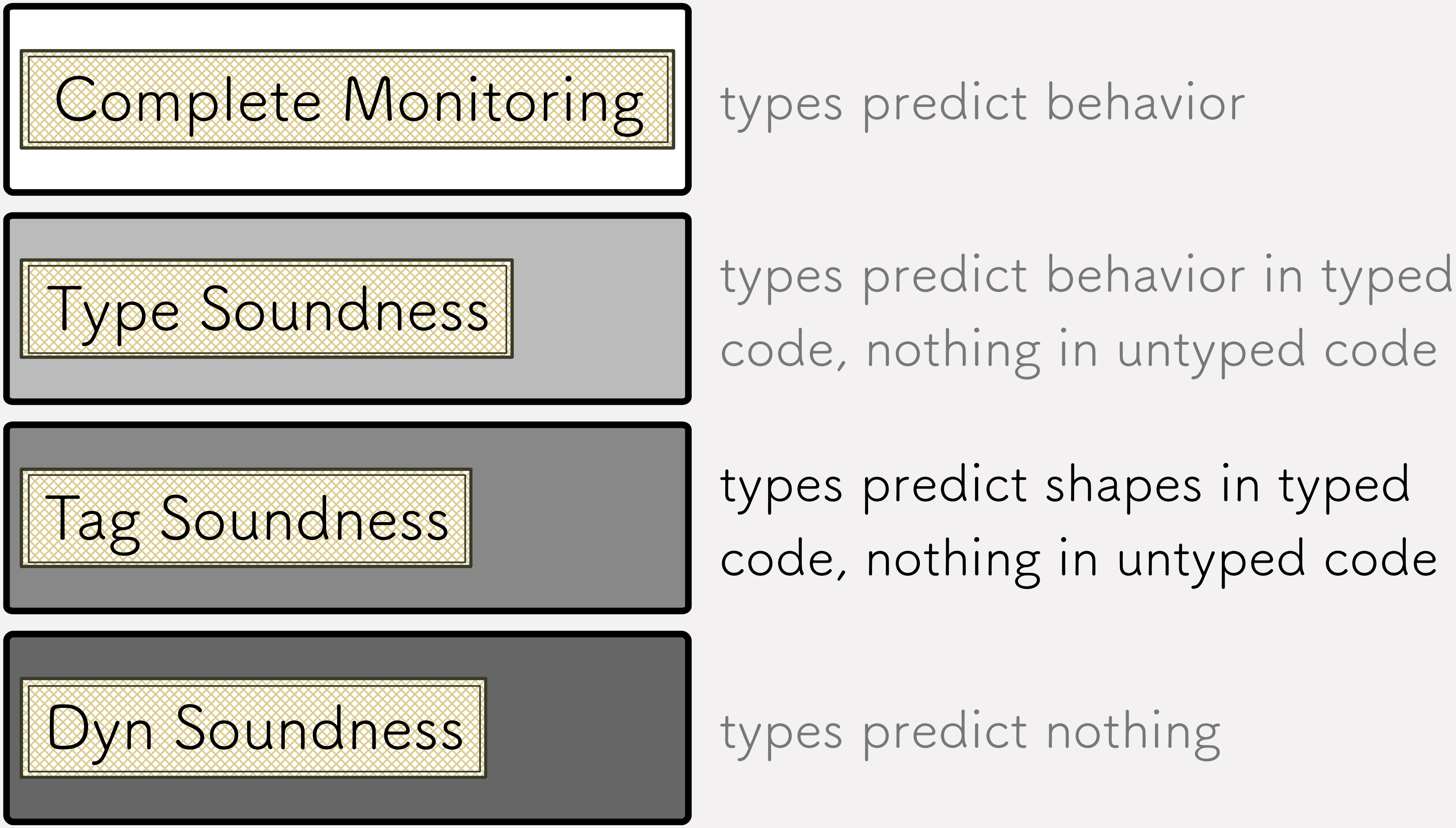
types predict behavior in typed code, nothing in untyped code

Tag Soundness

types predict shapes in typed code, nothing in untyped code

Dyn Soundness

types predict nothing



Complete Monitoring

types predict behavior

Type Soundness

types predict behavior in typed code, nothing in untyped code

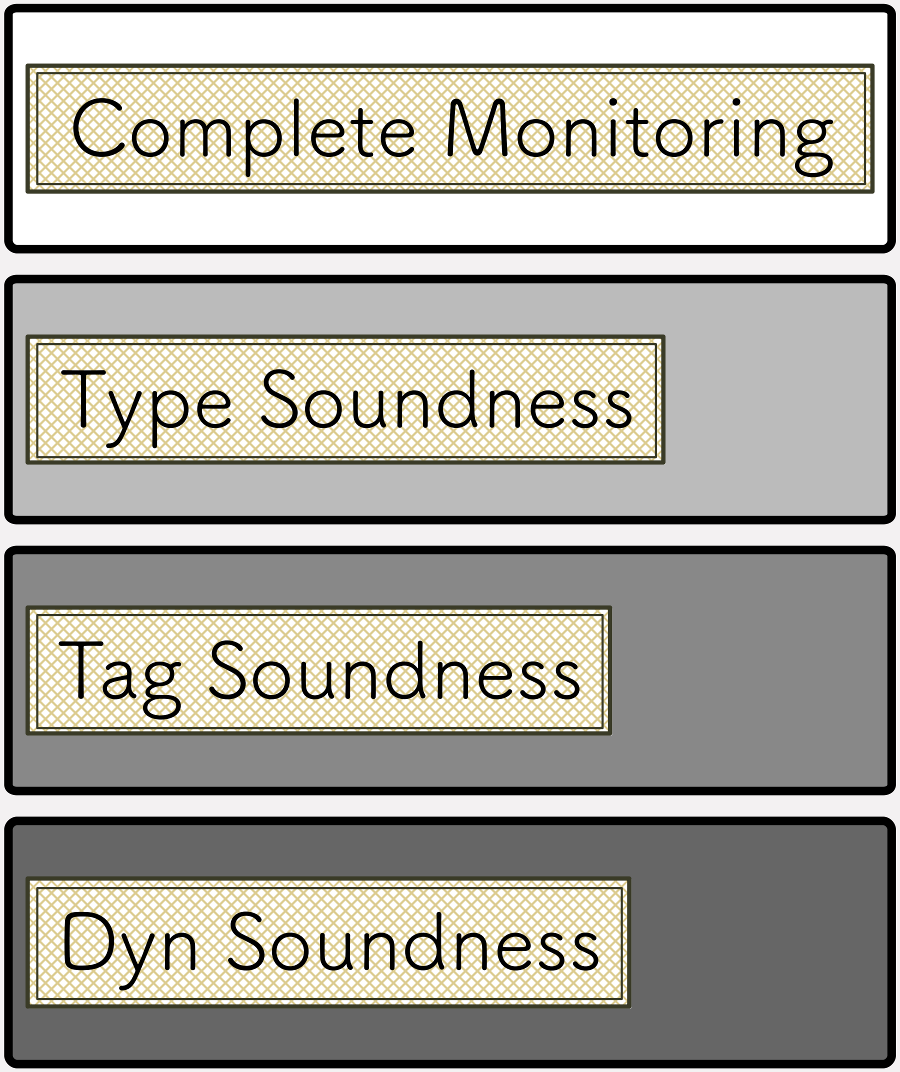
Tag Soundness

types predict shapes in typed code, nothing in untyped code

Dyn Soundness

types predict nothing

Complete Monitoring	types predict behavior
Type Soundness	types predict behavior in typed code, nothing in untyped code
Tag Soundness	types predict shapes in typed code, nothing in untyped code
Dyn Soundness	types predict nothing



Complete Monitoring

types predict behavior

Type Soundness

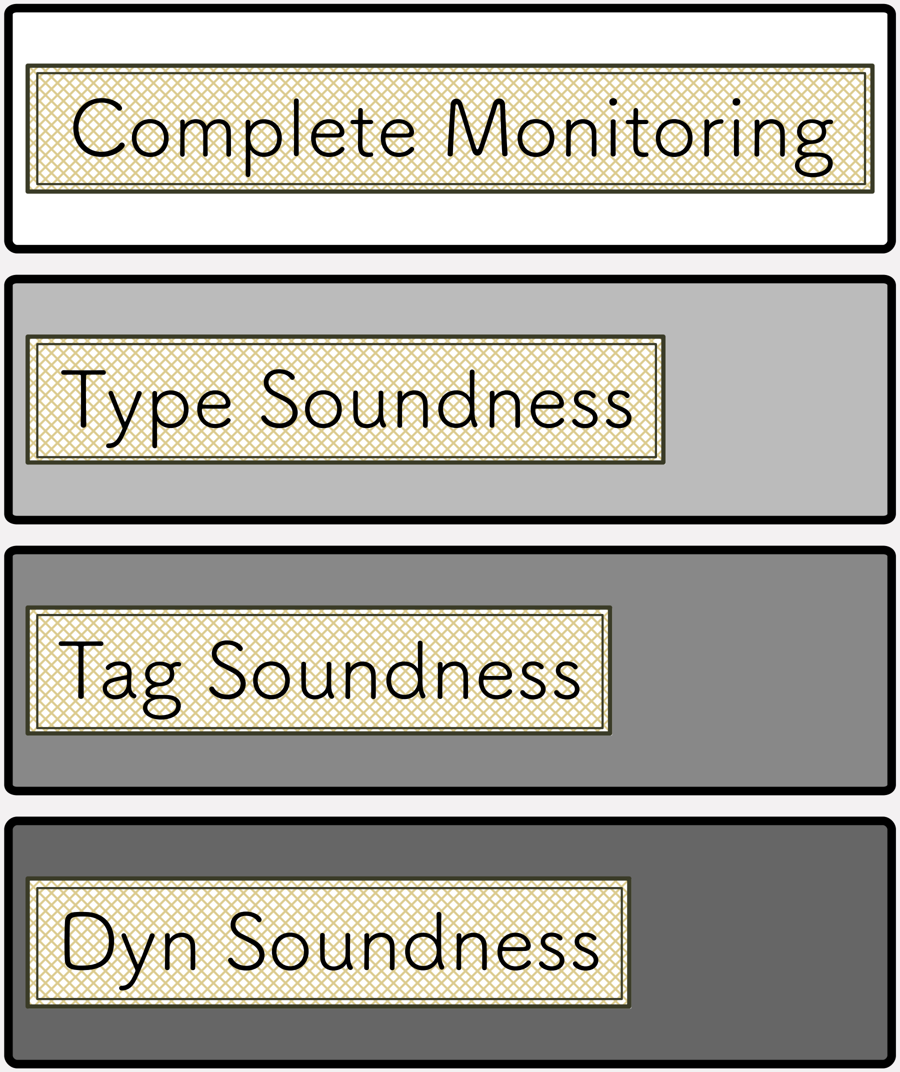
types predict behavior in typed code, nothing in untyped code

Tag Soundness

types predict shapes in typed code, nothing in untyped code

Dyn Soundness

types predict nothing

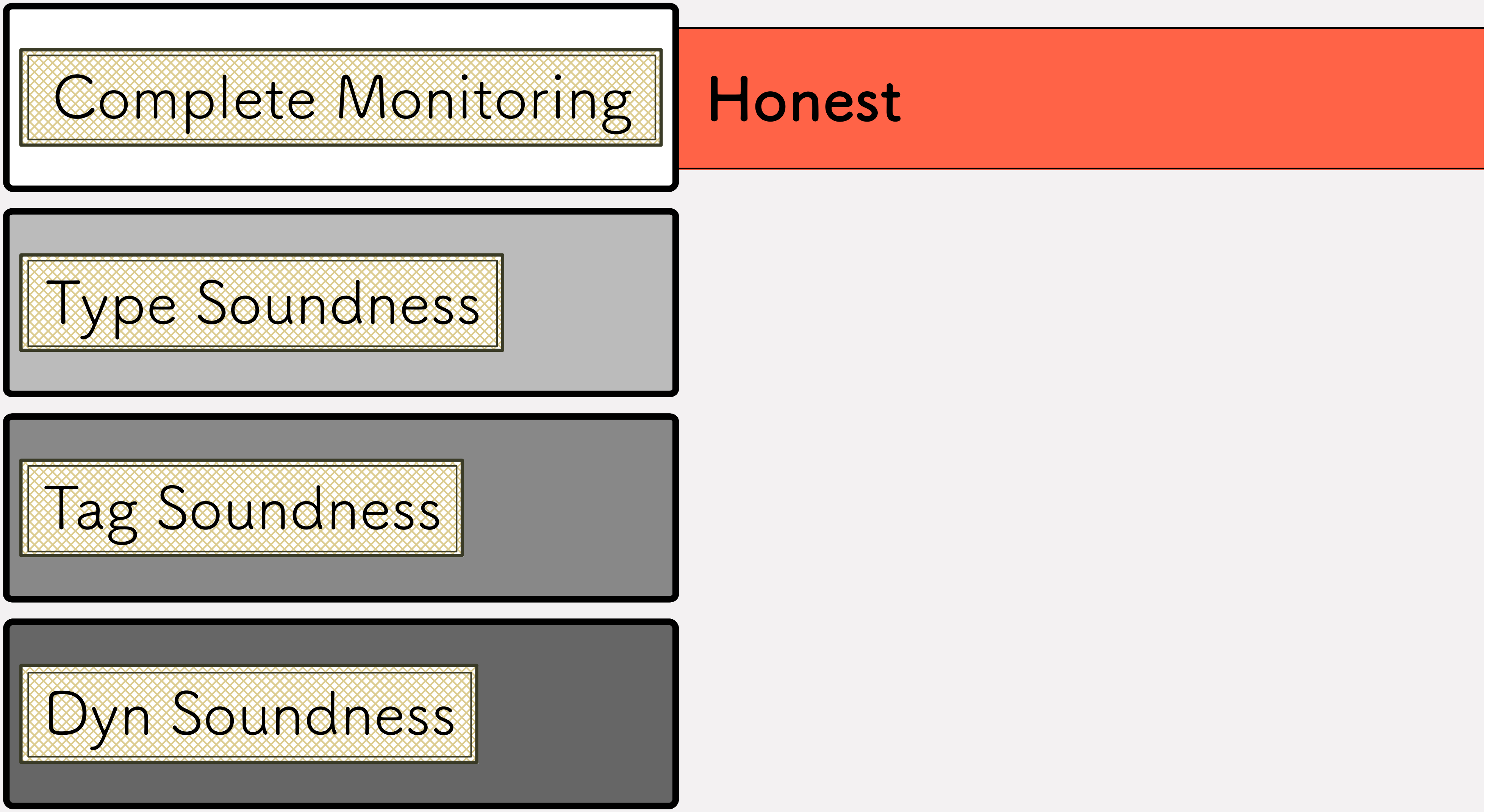


Complete Monitoring

Type Soundness

Tag Soundness

Dyn Soundness



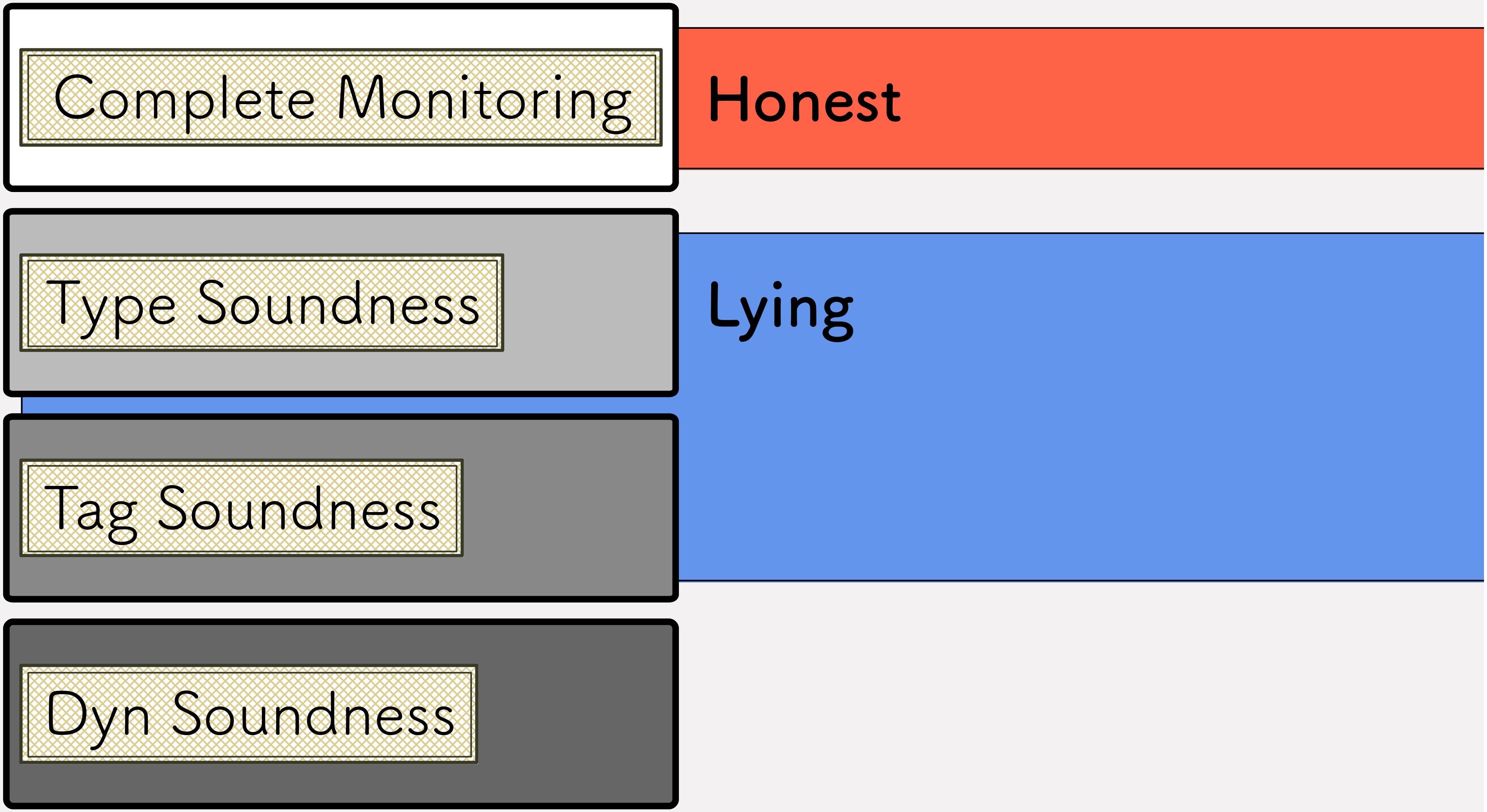
Complete Monitoring

Honest

Type Soundness

Tag Soundness

Dyn Soundness



Complete Monitoring

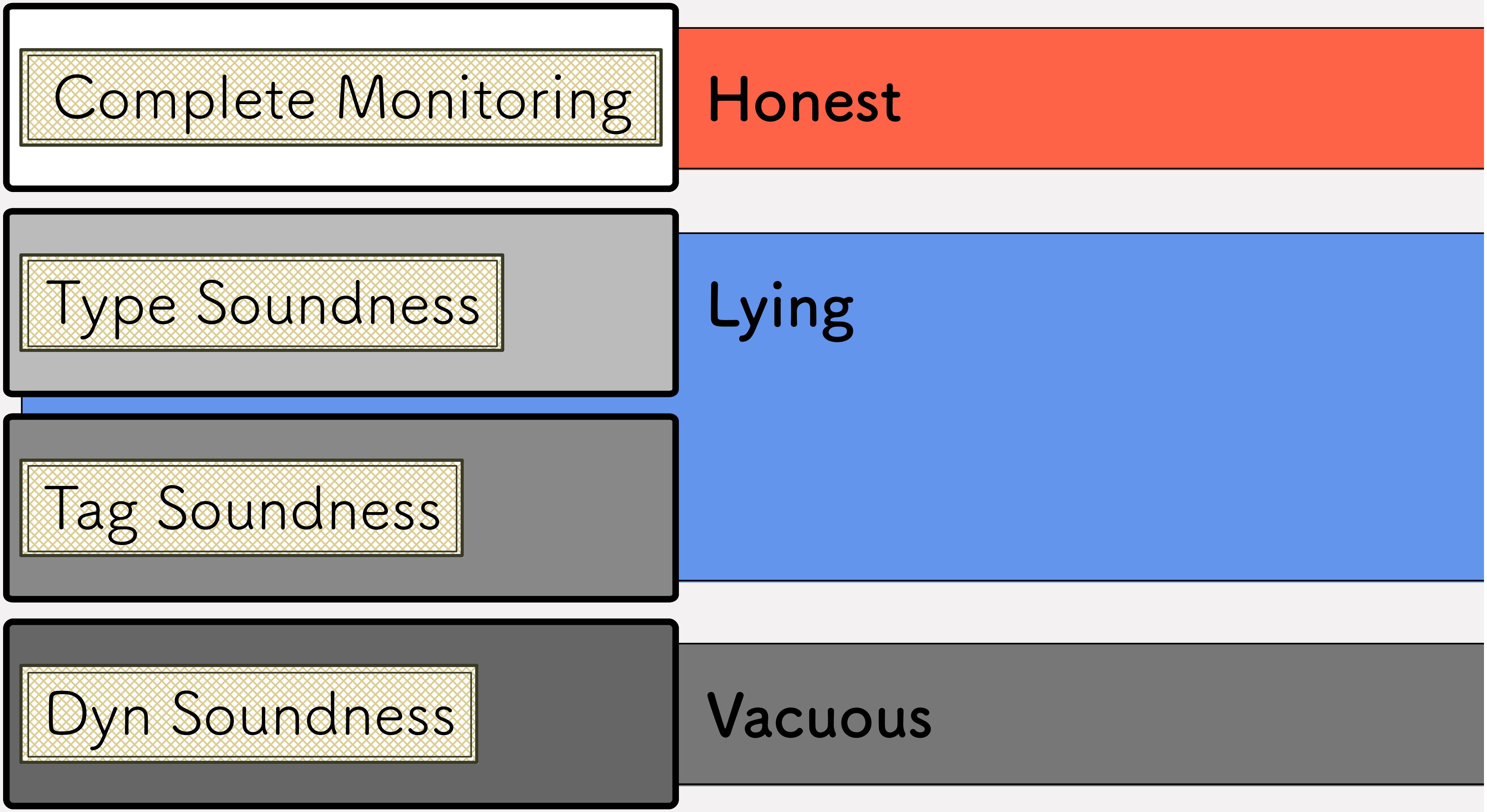
Honest

Type Soundness

Lying

Tag Soundness

Dyn Soundness



Complete Monitoring

Type Soundness

Tag Soundness

Dyn Soundness

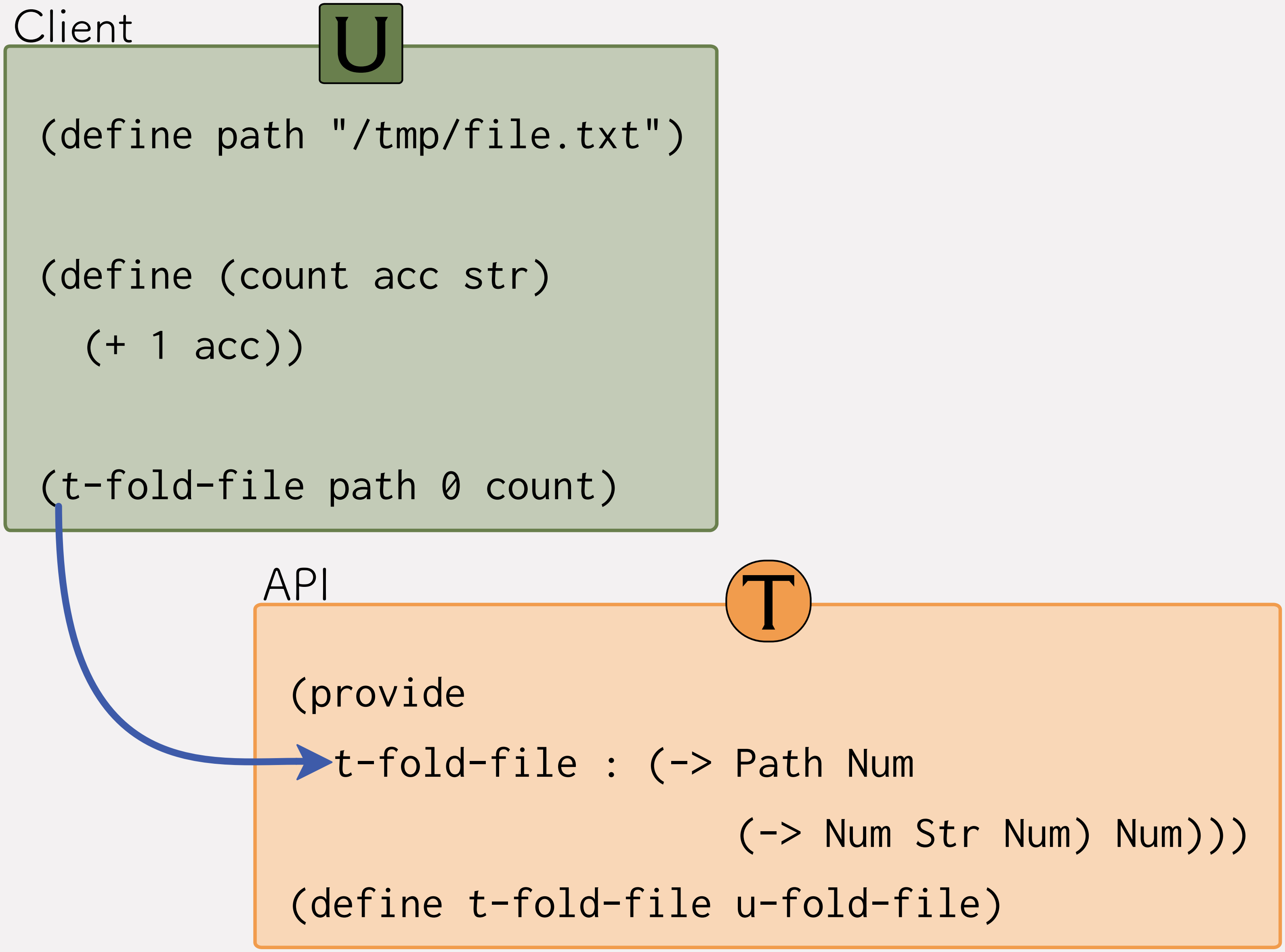
Honest

Lying

Vacuous

Honest vs. **Lying** Types

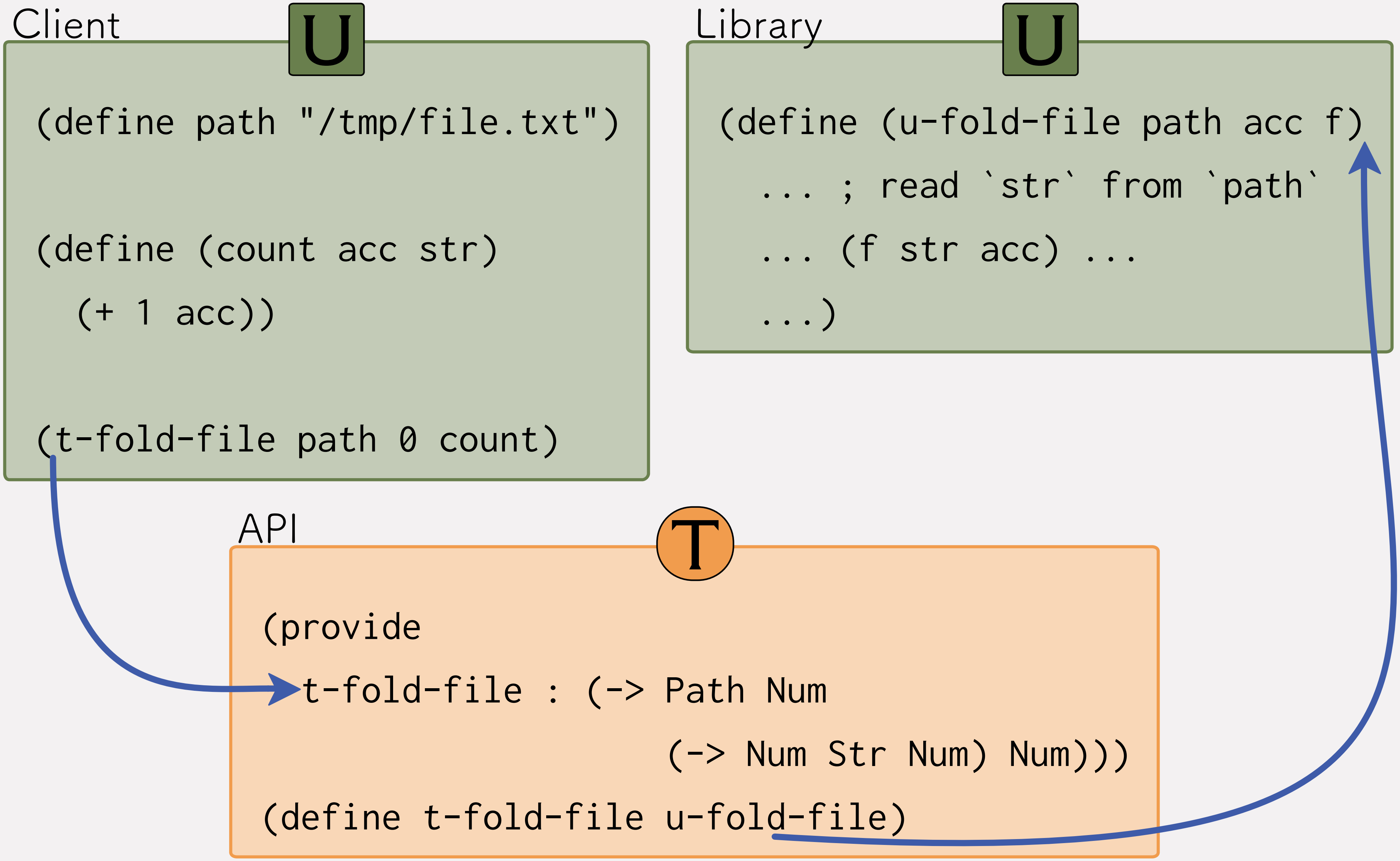
Honest vs. **Lying** Types



```
(define path "/tmp/file.txt")

(define (count acc str)
  (+ 1 acc))

(t-fold-file path 0 count)
```

```
(provide
  t-fold-file : (-> Path Num
                    (-> Num Str Num) Num)))
(define t-fold-file u-fold-file)
```

Honest vs. **Lying** Types

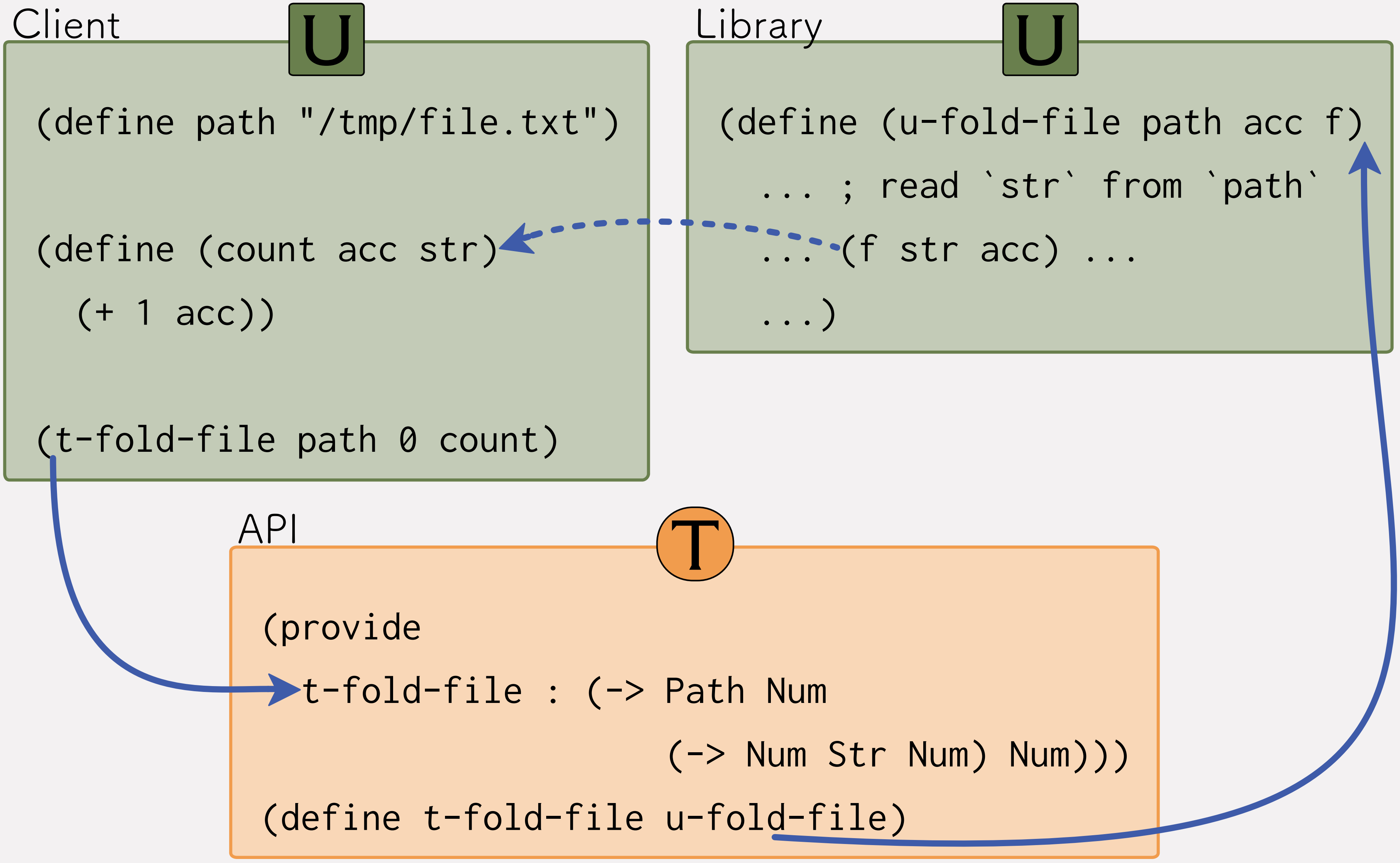


```
(define path "/tmp/file.txt")

(define (count acc str)
  (+ 1 acc))


(t-fold-file path 0 count)
```

```
(define (u-fold-file path acc f)
  ... ; read `str` from `path`
  ... (f str acc) ...
  ...)
```

```
(provide
 t-fold-file : (-> Path Num
                   (-> Num Str Num) Num)))
(define t-fold-file u-fold-file)
```

Honest vs. **Lying** Types

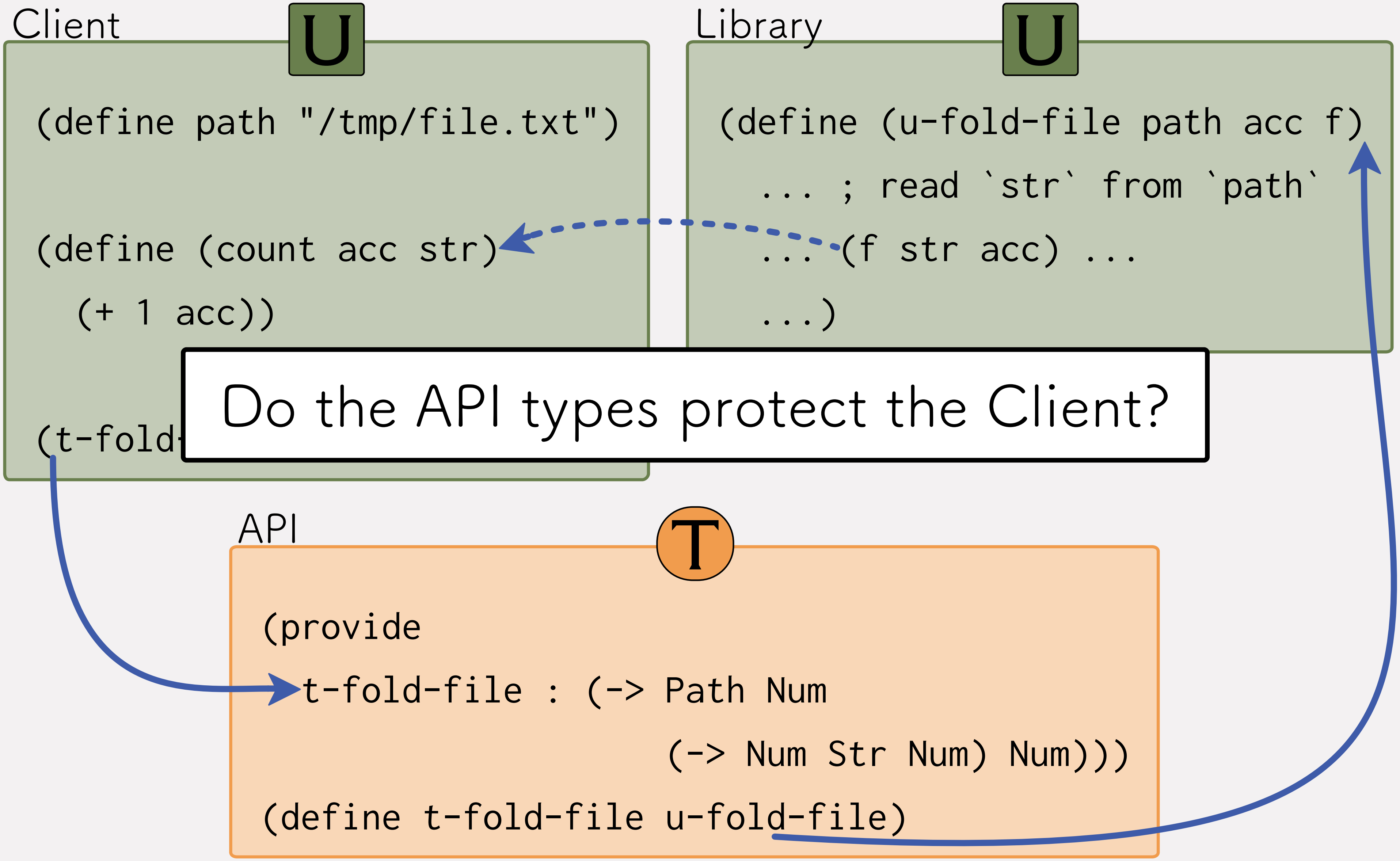


```
(define path "/tmp/file.txt")

(define (count acc str)
  (+ 1 acc))


(t-fold-file path 0 count)
```

```
(define (u-fold-file path acc f)
  ... ; read `str` from `path`
  ... (f str acc) ...
  ...)
```

```
(provide
  t-fold-file : (-> Path Num
                    (-> Num Str Num) Num)))
(define t-fold-file u-fold-file)
```

Honest vs. **Lying** Types



Client

```
(define path "/tmp/file.txt")

(define (count acc str)
  (+ 1 acc))


(t-fold-
```

Library

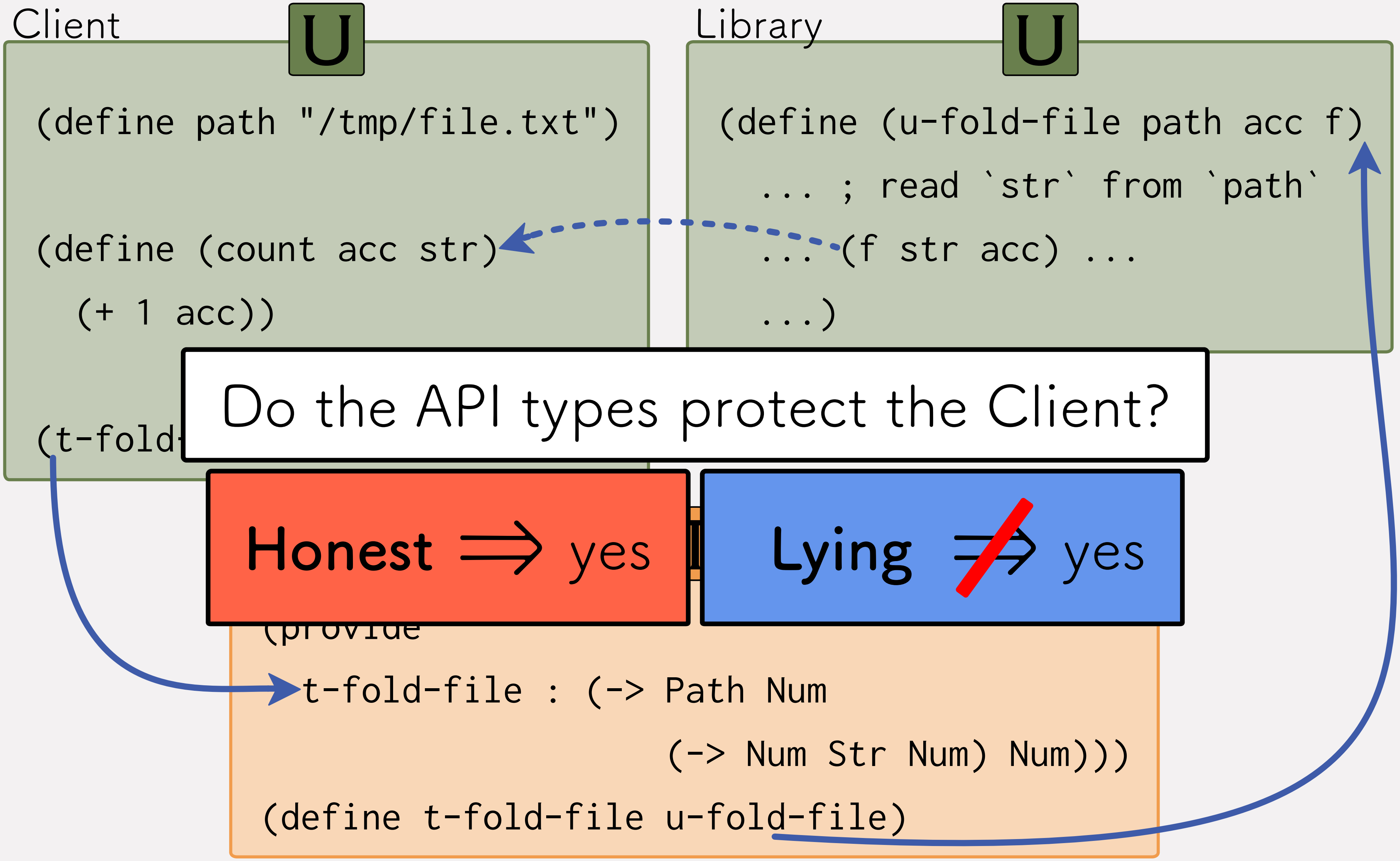
```
(define (u-fold-file path acc f)
  ... ; read `str` from `path`
  ... (f str acc) ...
  ...)
```

Do the API types protect the Client?

API

```
(provide
 t-fold-file : (-> Path Num
                   (-> Num Str Num) Num))
(define t-fold-file u-fold-file)
```

Honest vs. **Lying** Types



Client

U

```
(define path "/tmp/file.txt")

(define (count acc str)
  (+ 1 acc))


(t-fold-
```

Library

U

```
(define (u-fold-file path acc f)
   ... ; read `str` from `path`
   ... (f str acc) ...
   ...)
```

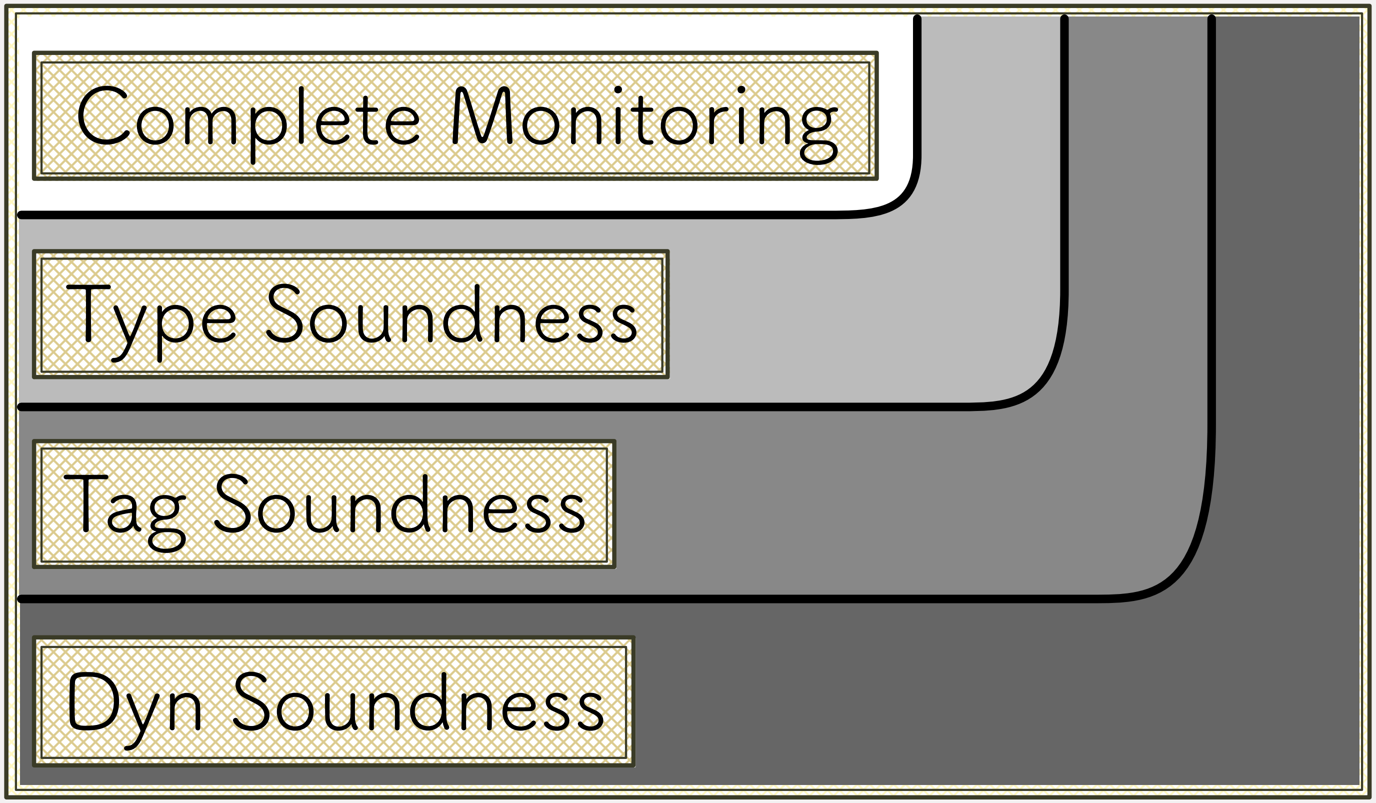
Do the API types protect the Client?

Honest ⇒ yes

Lying ⇏ yes

```
(provide
  t-fold-file : (-> Path Num
                    (-> Num Str Num) Num)))
(define t-fold-file u-fold-file)
```

Landscape: **Guarantees**



Complete Monitoring

Type Soundness

Tag Soundness

Dyn Soundness

Landscape: **Performance**

Landscape: **Performance**

Varied space, difficult to rank alternatives

Performance Comparison

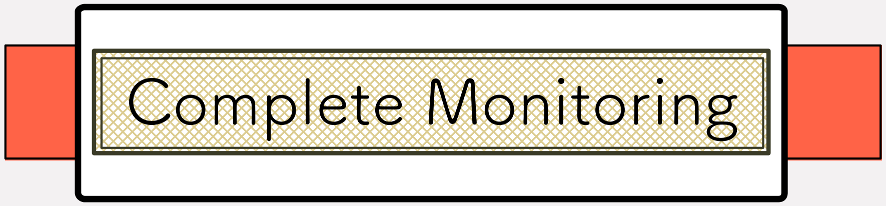
ICFP 2018

Natural vs. **Transient**

Performance Comparison

ICFP 2018

Natural vs. **Transient**



Tag Soundness

guard all boundaries with deep checks

rewrite typed code to tag-check inputs

`(listof int?)`

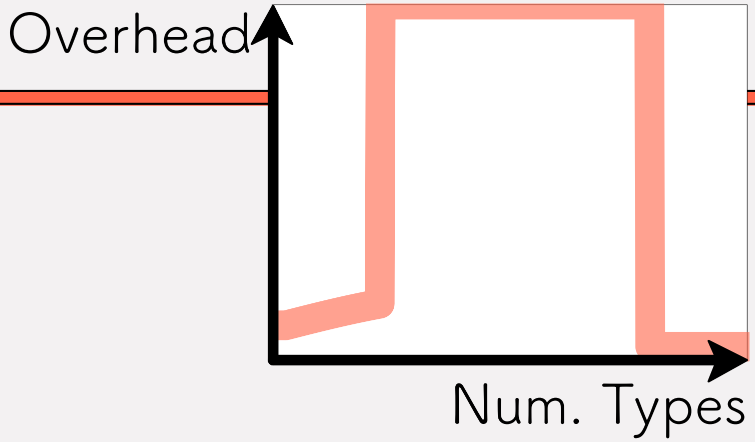
`list?`

Performance Comparison

ICFP 2018

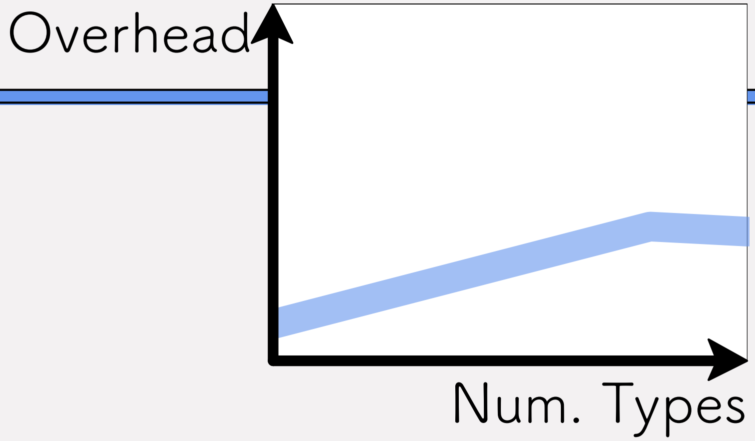
Natural

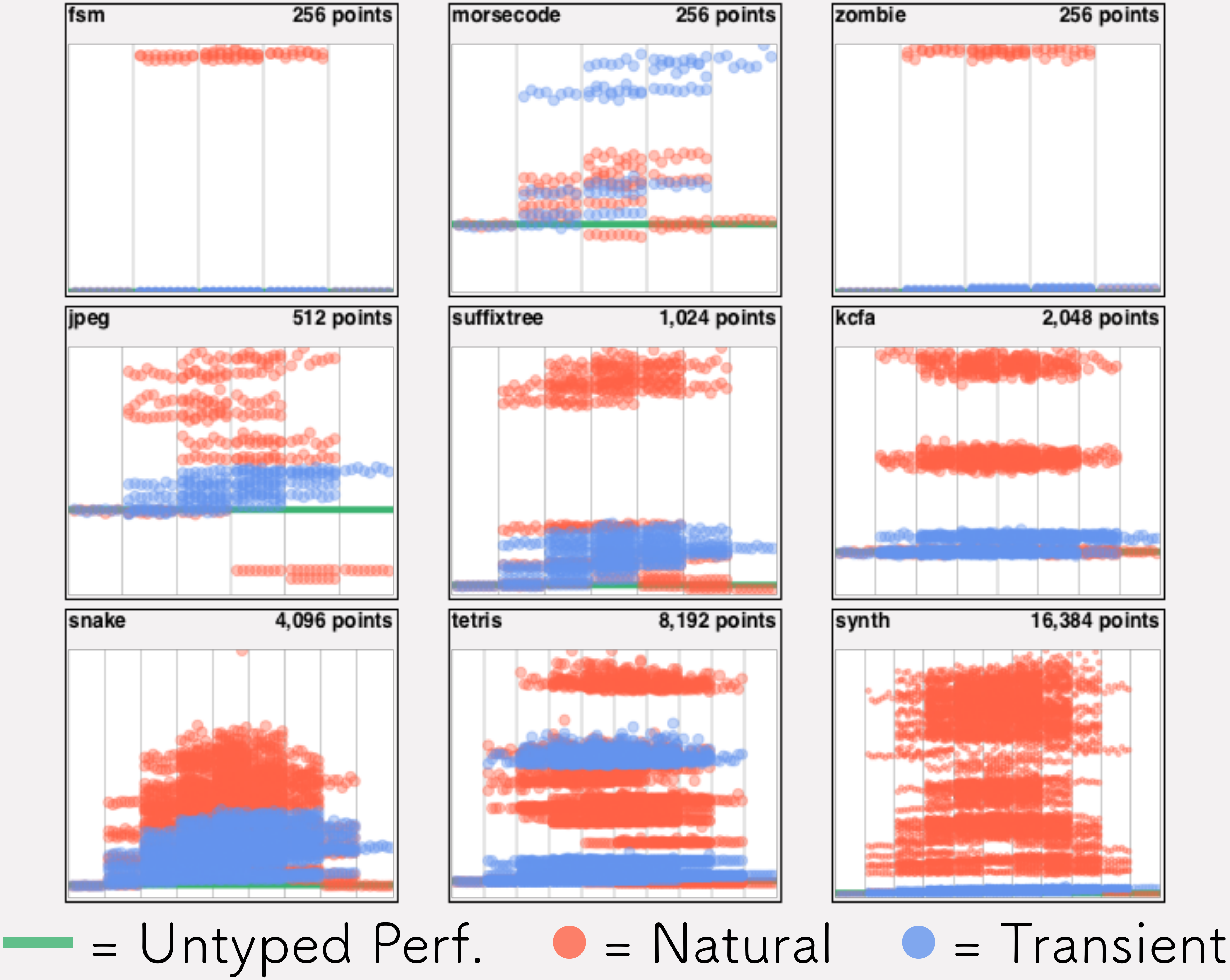
boundaries add "large" overhead



Transient

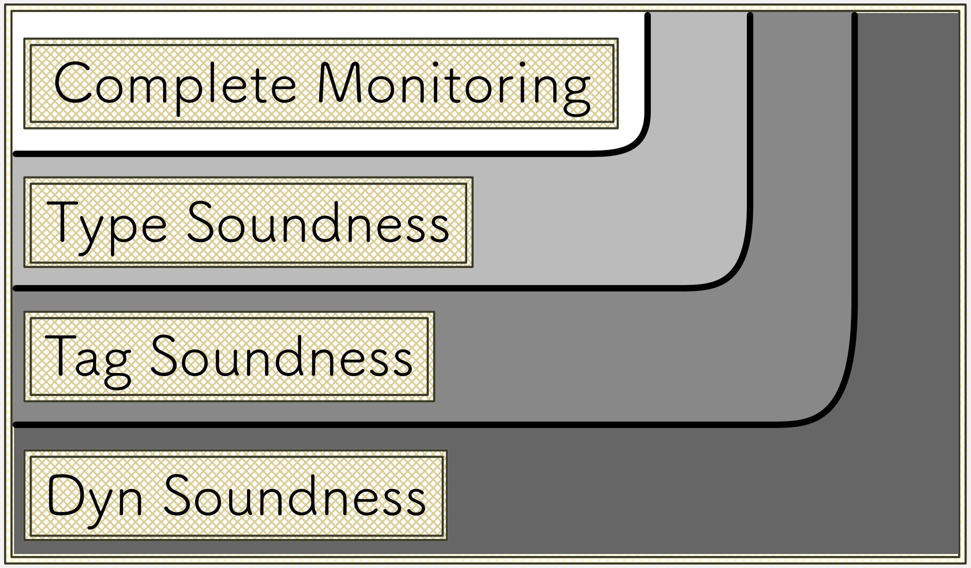
types add "small" overhead




fsm
256 points
morsecode
256 points
zombie
256 points
jpeg
512 points
suffixtree
1,024 points
kcfa
2,048 points
snake
4,096 points
tetris
8,192 points
synth
16,384 points
🟩 = Untyped Perf.
● = Natural
● = Transient

Thesis Question

U T U U T


Complete Monitoring

Type Soundness

Tag Soundness

Dyn Soundness

Goal = Migratory Typing

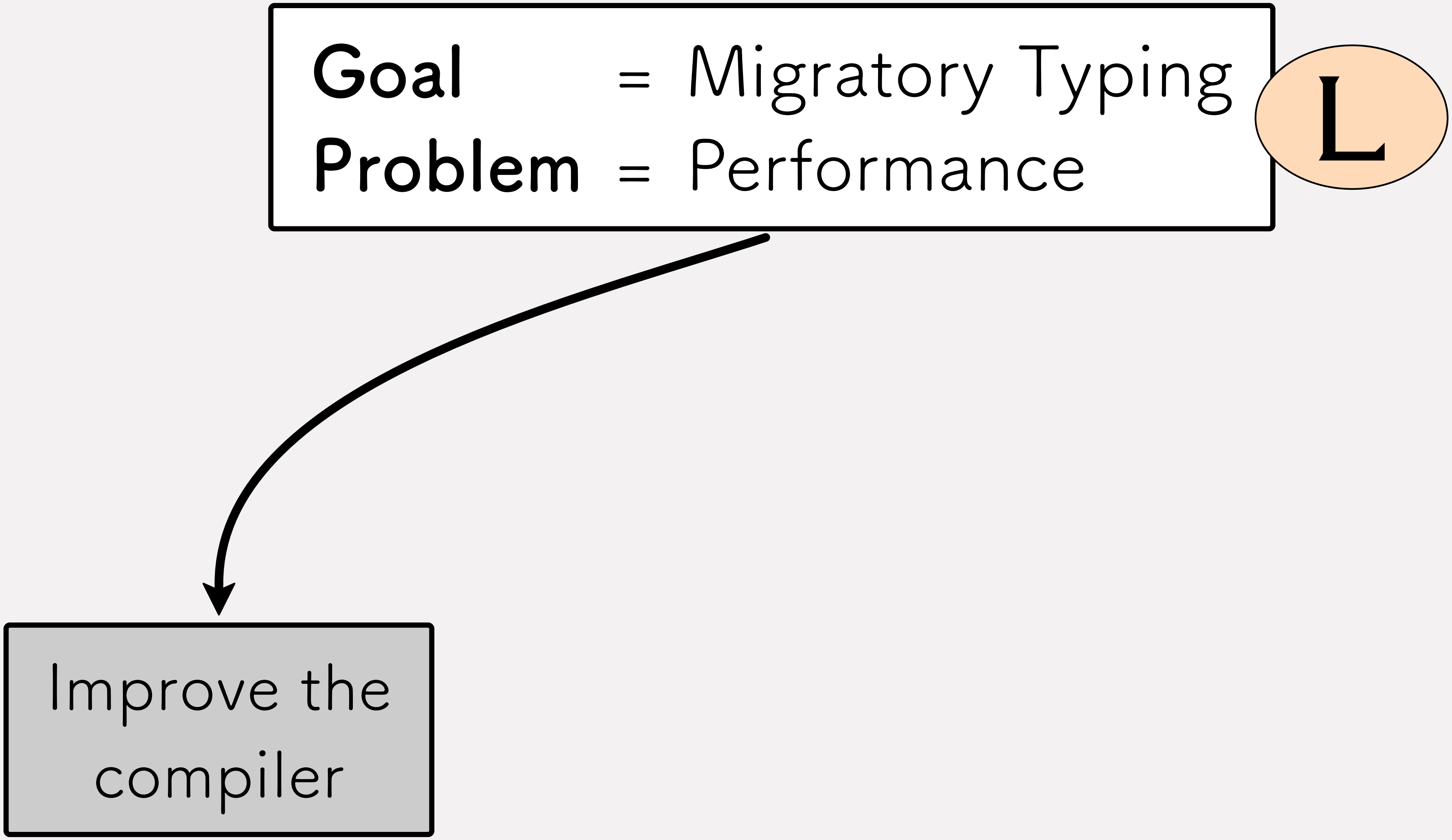
Problem = Performance

L

What to do?

Goal = Migratory Typing
Problem = Performance

L

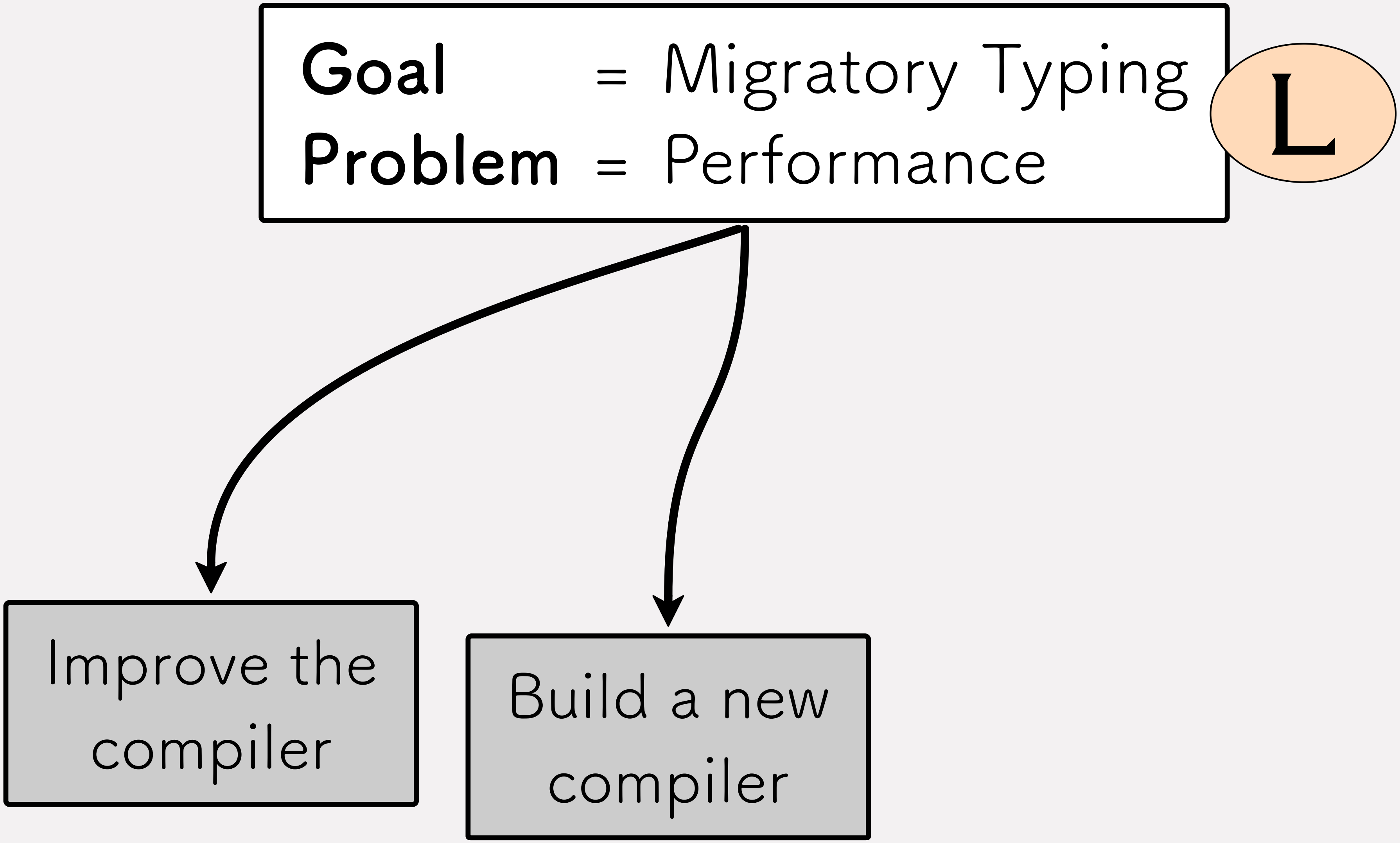


Goal = Migratory Typing

Problem = Performance

L

Improve the compiler



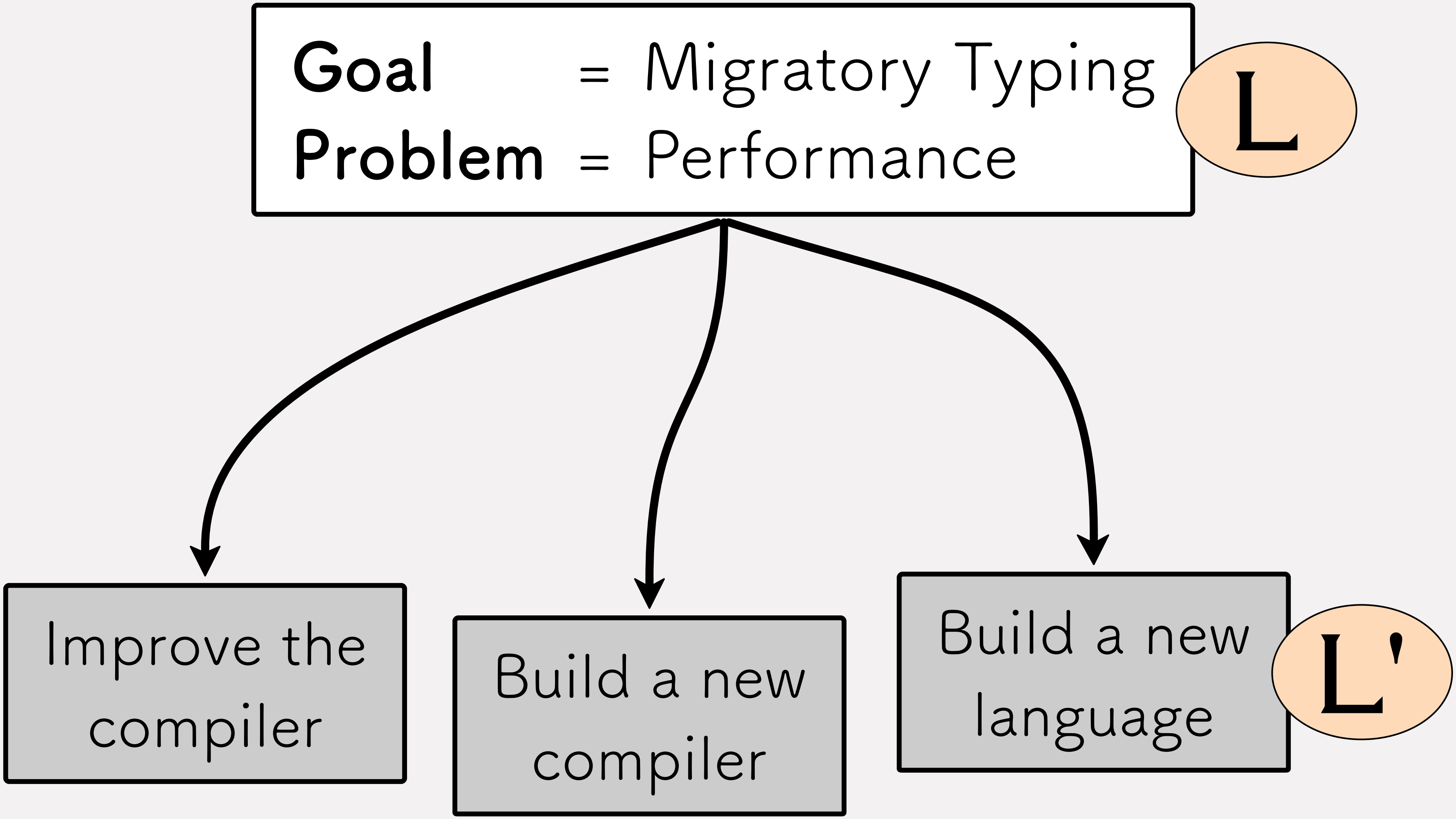
Goal = Migratory Typing

Problem = Performance

L

Improve the compiler

Build a new compiler



Goal = Migratory Typing

Problem = Performance

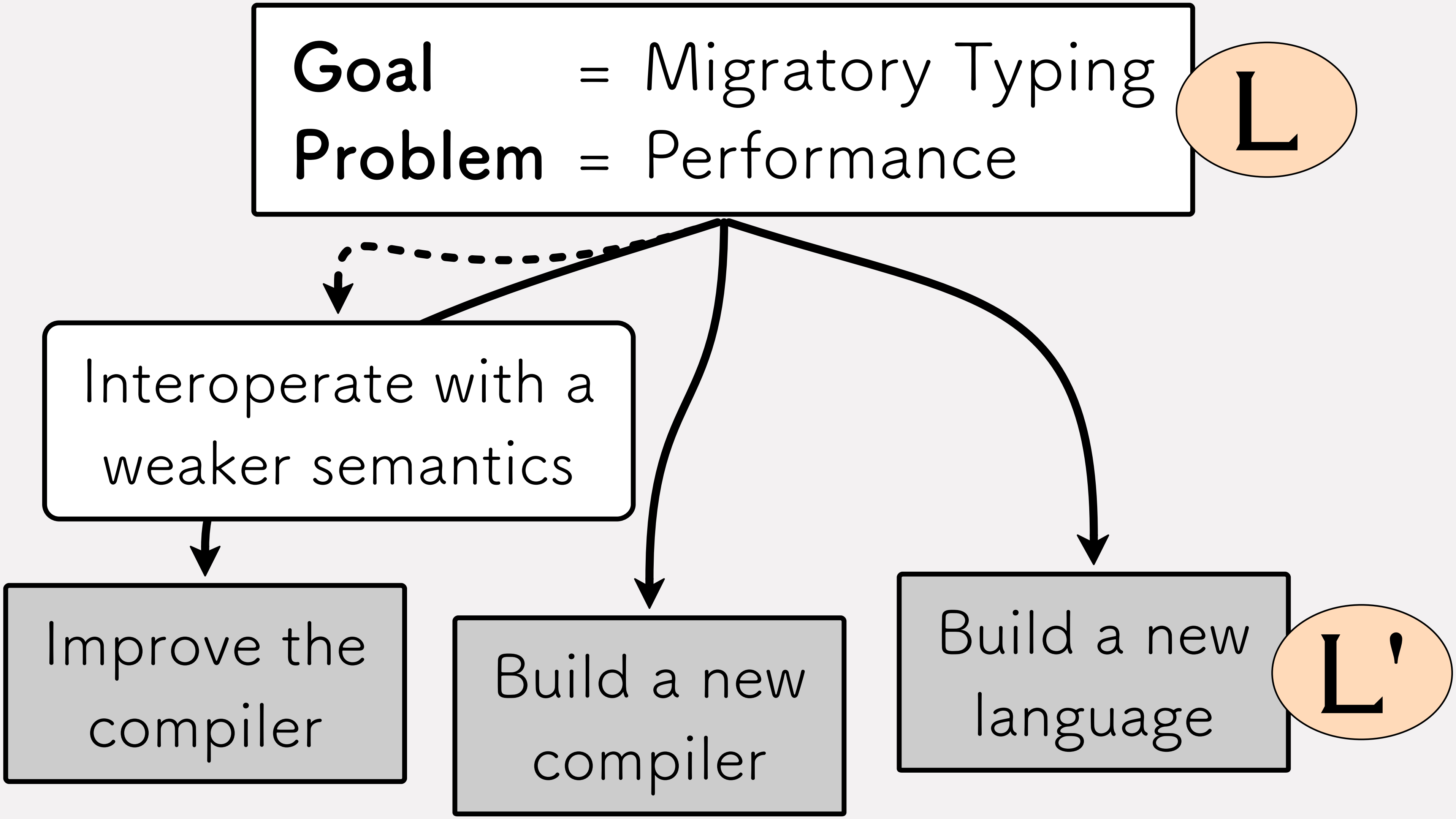
L

Improve the compiler

Build a new compiler

Build a new language

L'



Goal = Migratory Typing

Problem = Performance

L

Interoperate with a weaker semantics

Improve the compiler

Build a new compiler

Build a new language

L'

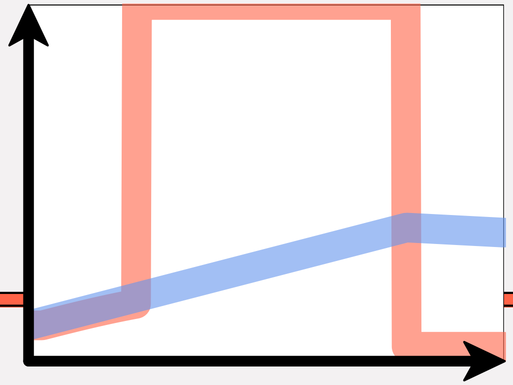
Q. Does **migratory typing** benefit from a combination of **honest** and **lying** types?

Q. Does **migratory typing** benefit from a combination of **honest** and **lying** types?

In particular,

Natural + **Transient**

Complementary Strengths



Natural

types predict full behavior, but need to avoid certain boundaries

Transient

types predict shapes, but add overhead to all typed code

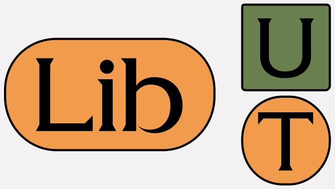
Benefits (1/3): Migration

U ➤ T

U ➤ T Lib $\overset{U}{T}$ +

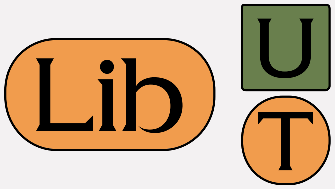
1. Begin with **Natural** types
2. Switch to **Transient** for performance
3. Revisit **Natural** for debugging
4. Return to **Natural** after typing all critical boundaries

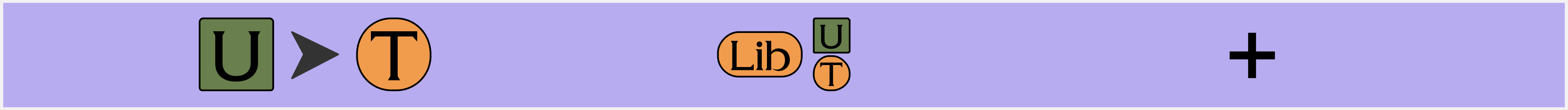
Benefits (2/3): Library Interaction



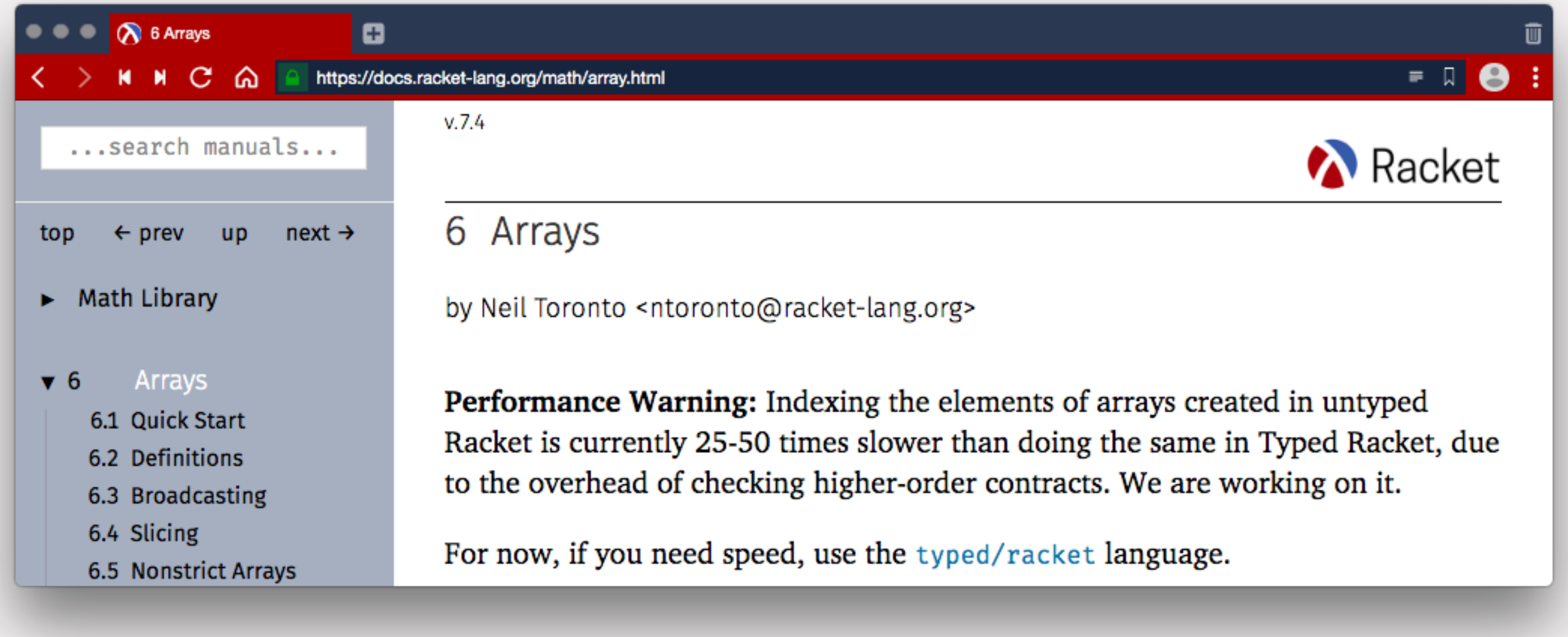
U ➤ T	Lib U T	+

Benefits (2/3): Library Interaction





math/array: "25 to 50 times slower"



6 Arrays

by Neil Toronto <ntoronto@racket-lang.org>

Performance Warning: Indexing the elements of arrays created in untyped Racket is currently 25-50 times slower than doing the same in Typed Racket, due to the overhead of checking higher-order contracts. We are working on it.

For now, if you need speed, use the `typed/racket` language.

Benefits (2/3): Library Interaction

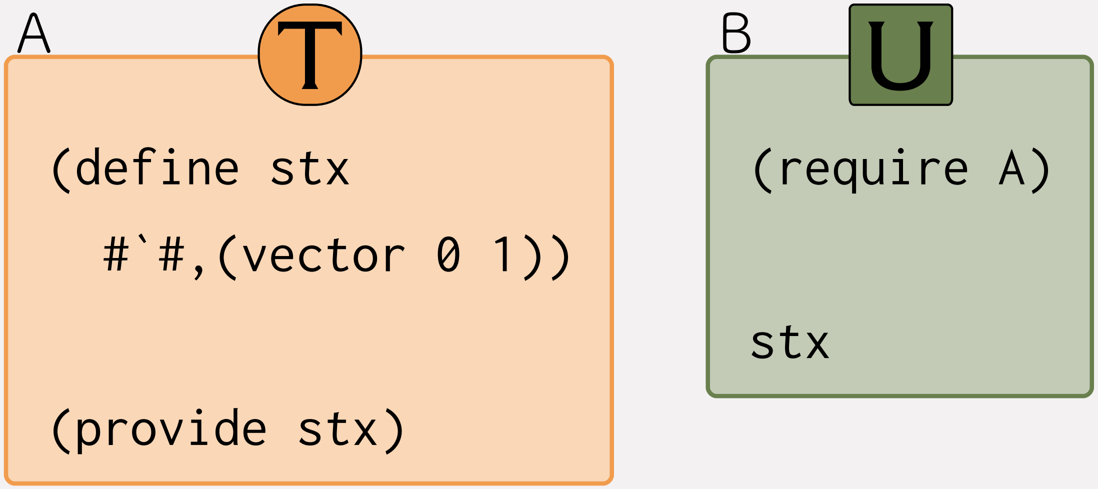
Lib U T

U ➤ T   Lib U T   +

Changing a library to **Transient** may improve overall performance

Benefits (3/3): Compatibility +

U ➤ T Lib U T +



A

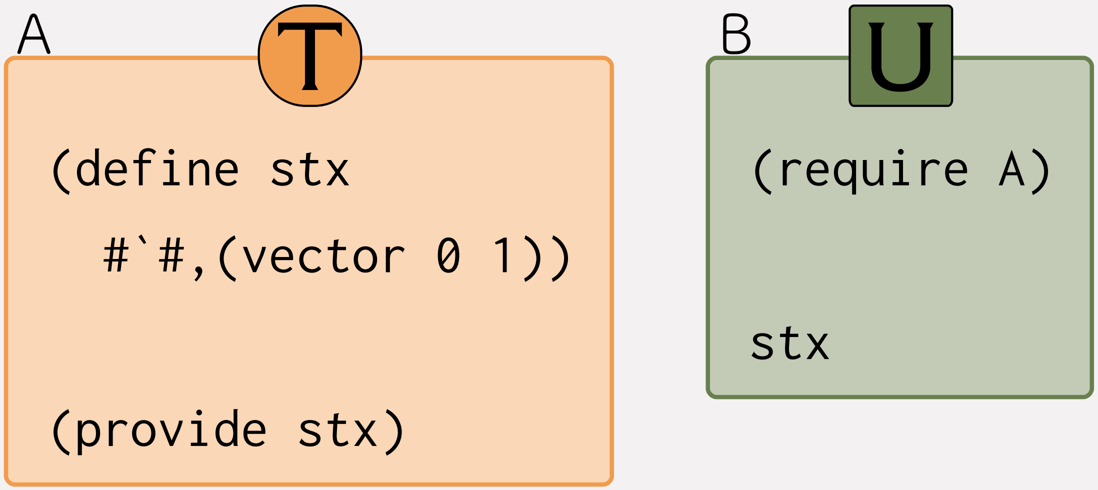
```
(define stx
  #`#,(vector 0 1))

(provide stx)
```

B

```
(require A)

stx
```

Benefits (3/3): Compatibility +

U ➤ T   Lib U T   +



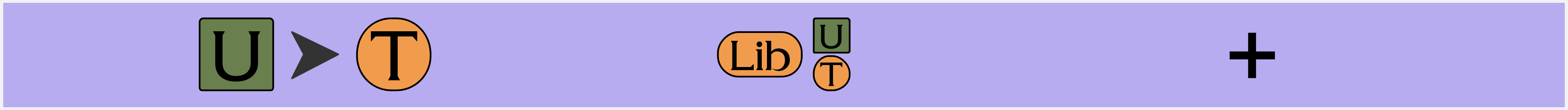
```
(define stx
  #`#,(vector 0 1))

(provide stx)
```

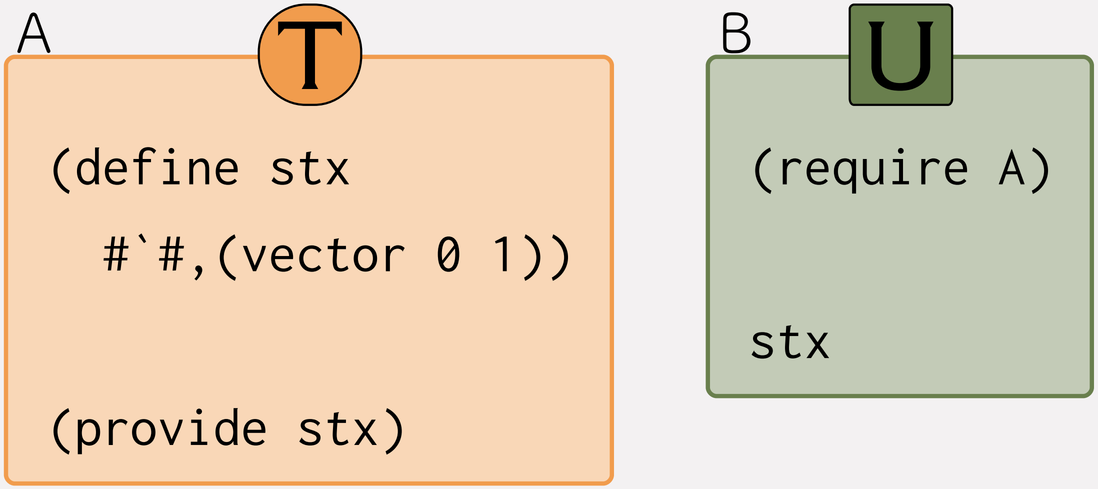
```
(require A)


stx
```

Type Check: Ok

Benefits (3/3): Compatibility

+





```
(define stx
  #`#,(vector 0 1))

(provide stx)
```

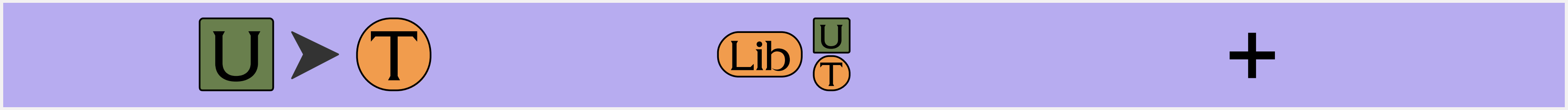
```
(require A)

stx
```

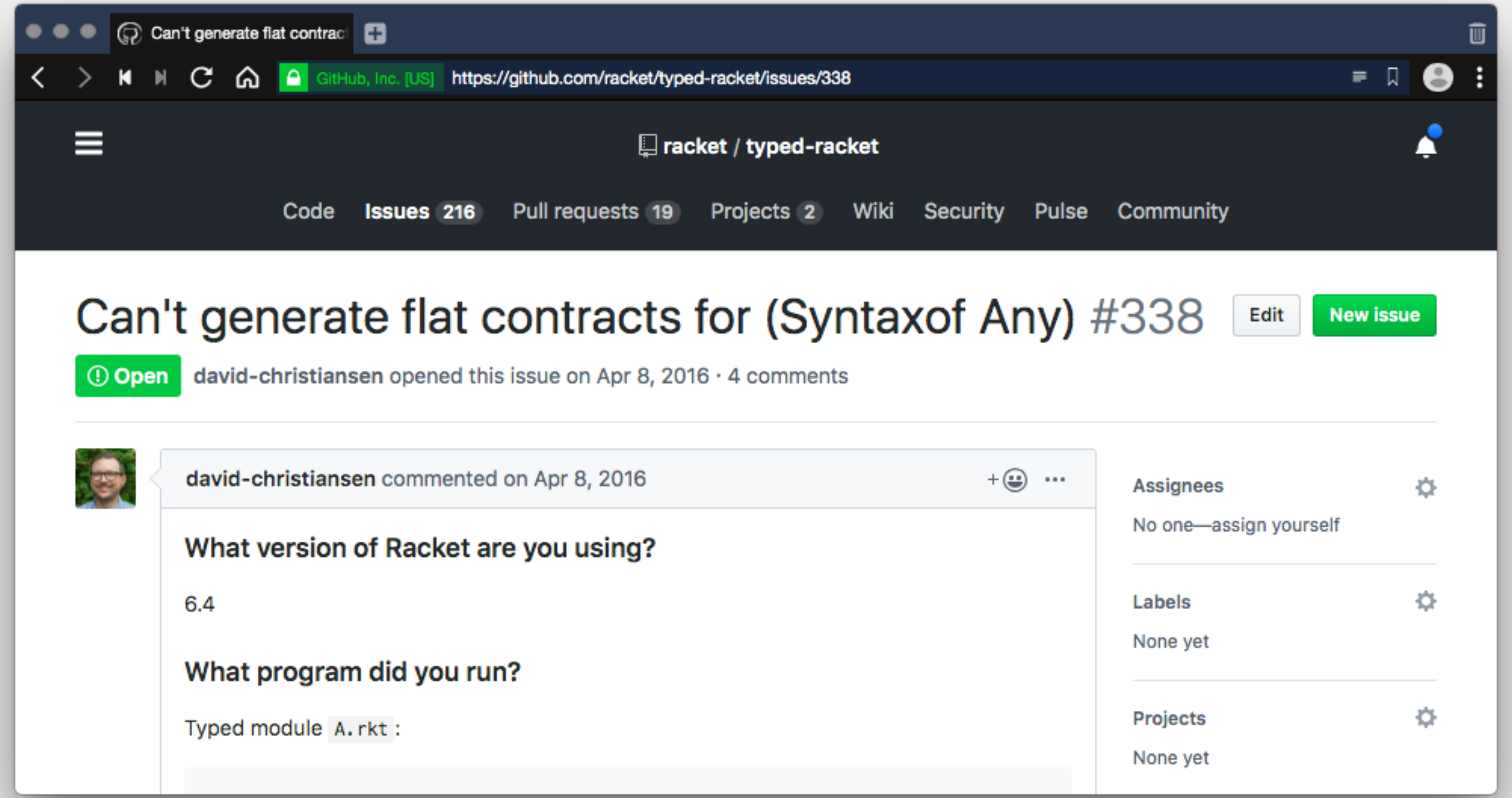
Type Check: Ok

Runtime: Error could not convert type to a contract

Benefits (3/3): Compatibility

+





Can't generate flat contracts for (Syntaxof Any) #338

Open david-christiansen opened this issue on Apr 8, 2016 · 4 comments

david-christiansen commented on Apr 8, 2016

What version of Racket are you using?

6.4

What program did you run?

Typed module `A.rkt` :

Assignees

No one—assign yourself

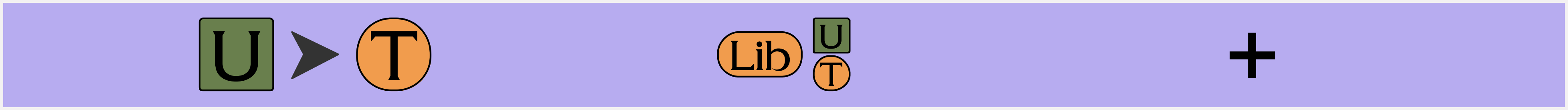
Labels

None yet

Projects

None yet

Benefits (3/3): Compatibility +



Typed Racket provides 203 base types;
12 lack runtime support (wrappers)

Benefits (3/3): Compatibility +

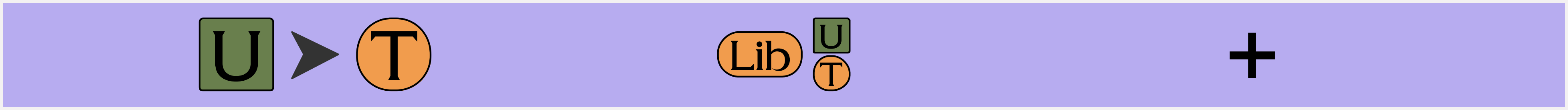
U ➤ T Lib U T +

Typed Racket provides 203 base types;
12 lack runtime support (wrappers)

(Async-Channel T)
(Custodian-Box T)
(C-Mark-Key T)
(Evt T)
(Ephemeron T)
(Future T)
(MPair T T')
(MList T)
(Prompt-Tag T T')
(Syntax T)
(Thread-Cell T)
(Weak-Box T)

Benefits (3/3): Compatibility

+



Typed Racket provides 203 base types;
12 lack runtime support (wrappers)

Transient does not need wrappers,
so more code can run

Plan

Q. Does migratory typing benefit from a combination of honest and lying types?

Q. Does migratory typing benefit from a combination of honest and lying types?

Q1. Can honest and lying types coexist?

Q2. Are the benefits measurably significant?

Q1. Can honest and lying coexist?

Model:

- develop a combined model
- formally prove basic properties
- reduce overlap in runtime checks

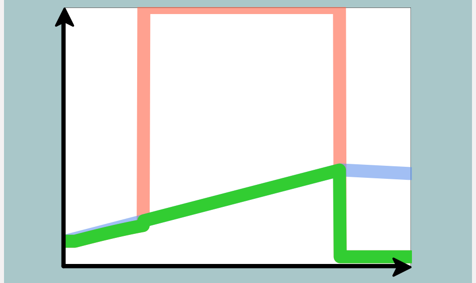
Q1. Can honest and lying coexist?

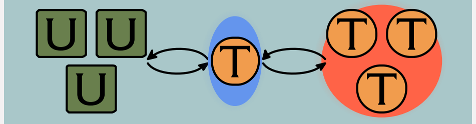
Q1. Can honest and lying coexist?

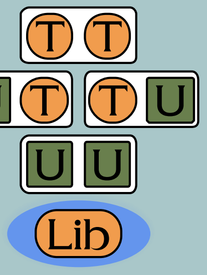
Implementation:

- re-use the type checker
- support all Racket values
- avoid the contract library
- adapt the TR optimizer to lying types

Q2. Are the benefits significant?

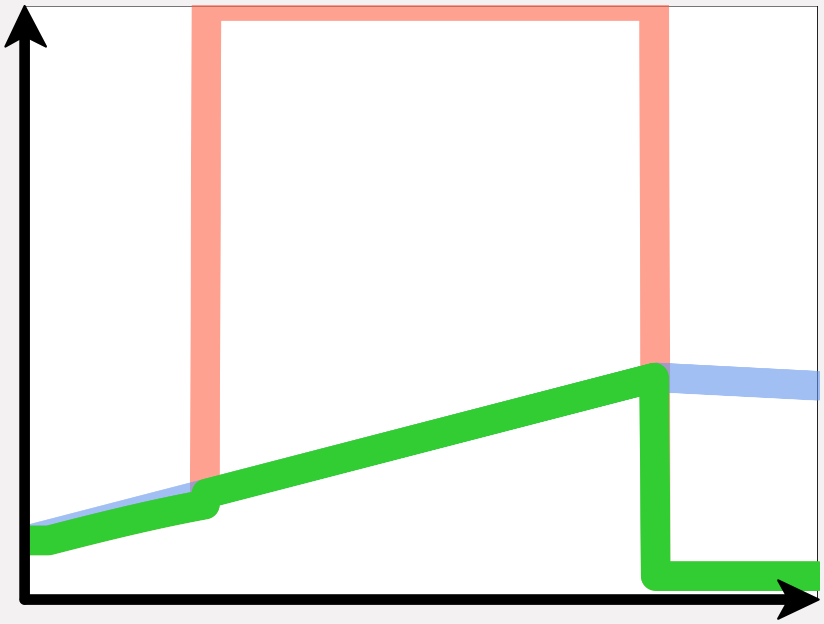


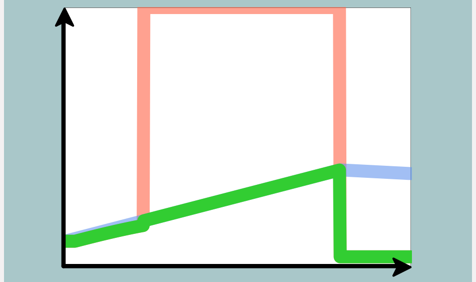


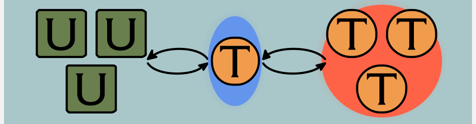


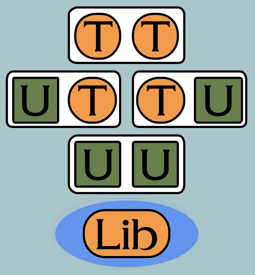
Q2. Are the benefits significant?

Goal: min(Natural, Transient)

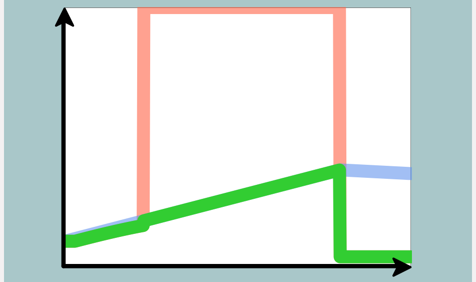


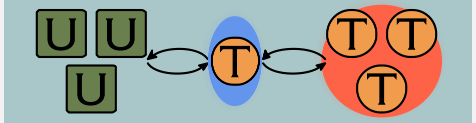


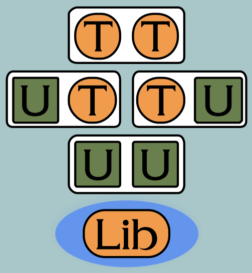




Q2. Are the benefits significant?

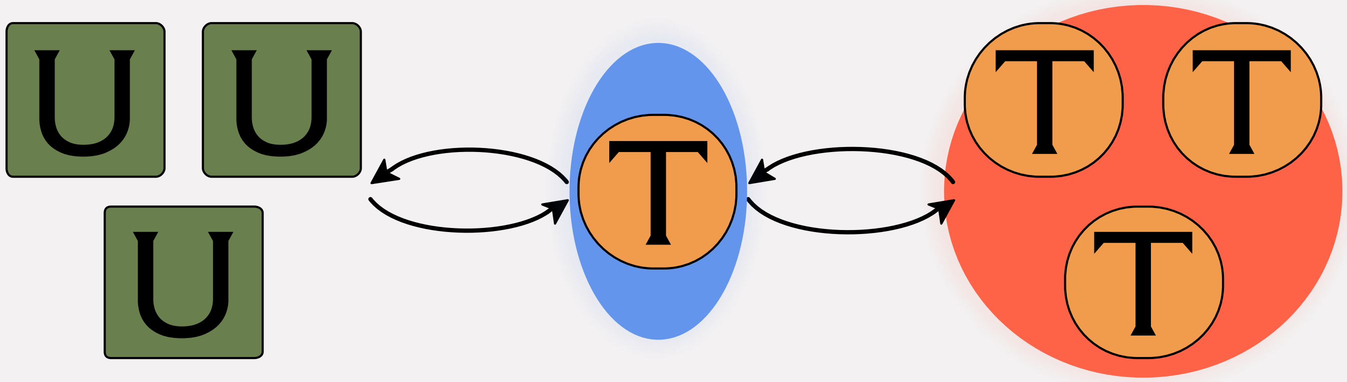


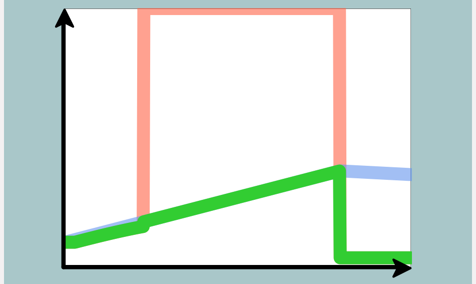


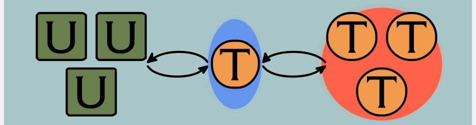


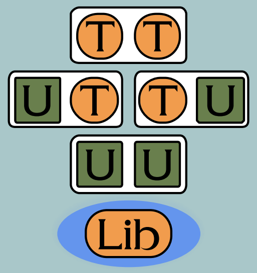
Q2. Are the benefits significant?

Maybe: reduce cost of U/T edge





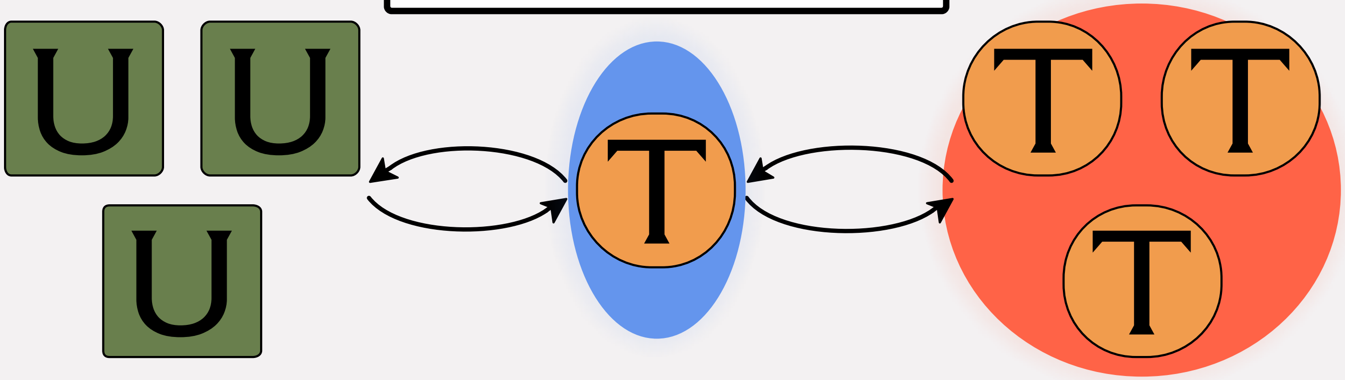


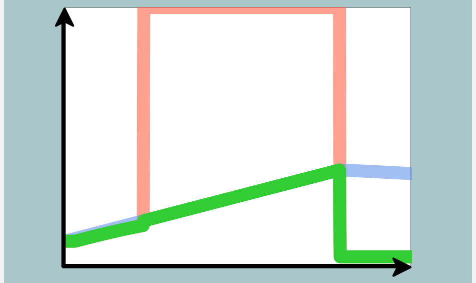


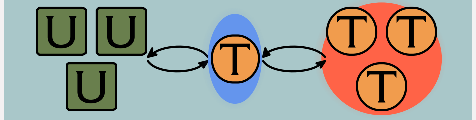
Q2. Are the benefits significant?

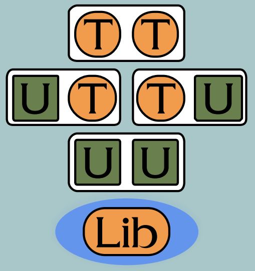
Maybe: reduce cost of U/T edge

How to find?



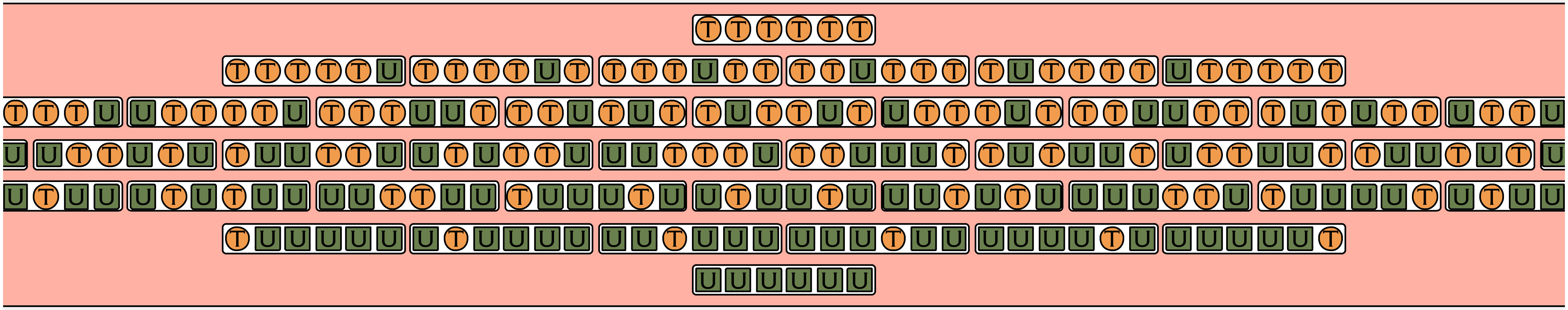






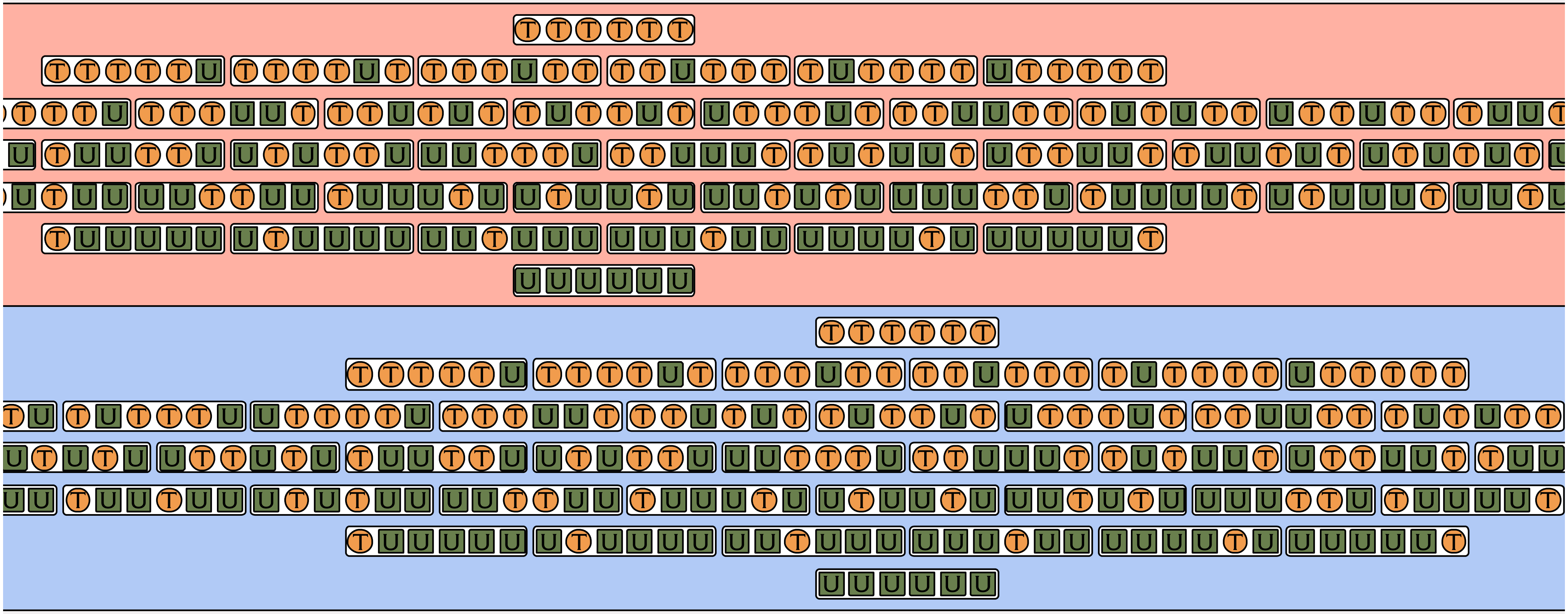
How to measure performance?

POPL 2016 $= 2^N$ measurements


TTTTTT
TTTTTU TTTTUT TTTUTT TTUTTT TUTTTT UTTTTT
TTTU UTTTTU TTTUUT TTUTUT TUTTUT UTTTUT TTUUTT TUTUTT UTTU
U UTTUTU TUUTTU UTUTTU UUTTTU TTUUUT TUTUUT UTTUUT TUUTUT U
UTUU UTUTUU UUTTUU TUUUTU UTUUTU UUTUTU UUUTTU TUUUUT UTUU
TUUUUU UTUUUU UUTUUU UUUTUU UUUUTU UUUUUT
UUUUUU

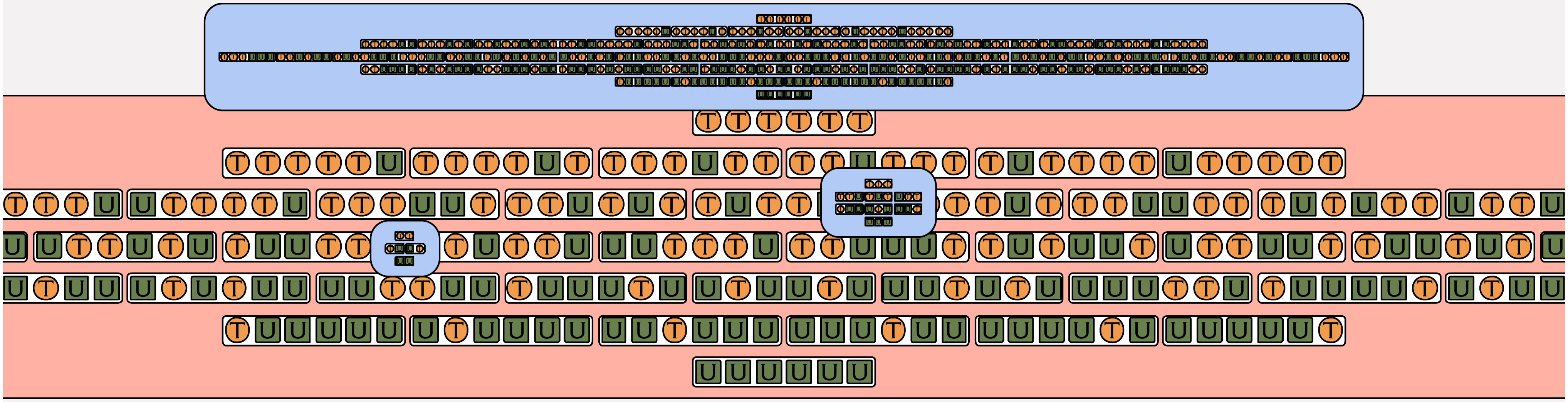
How to measure performance?

ICFP 2018 = $2^{(N+1)}$ measurements



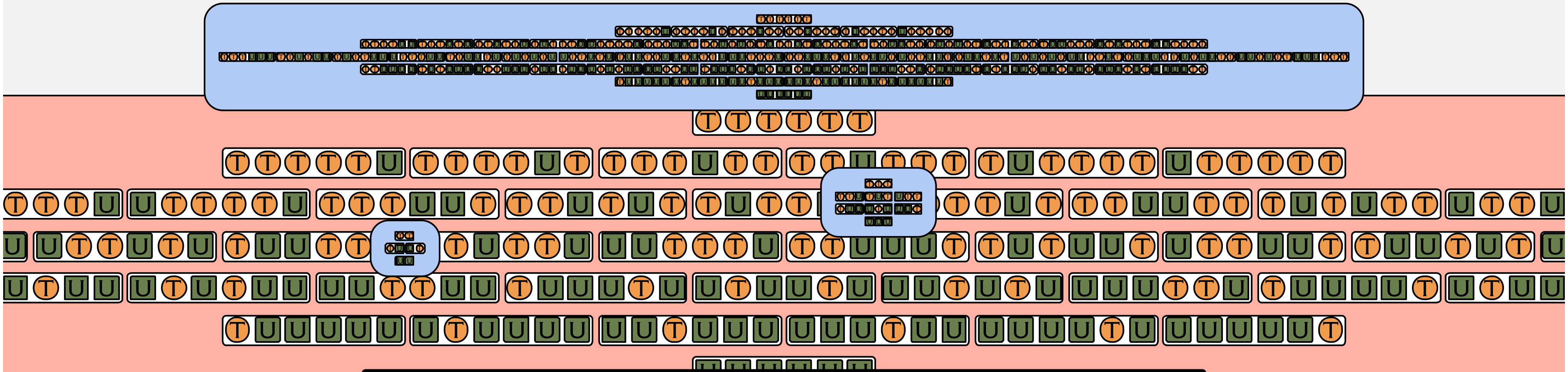
How to measure performance?

Next = 3^N measurements?



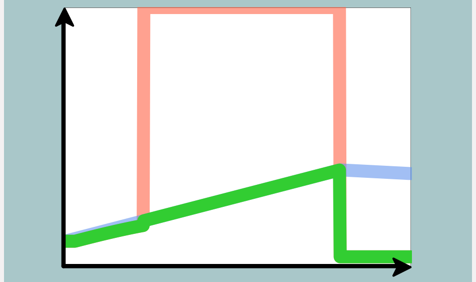
How to measure performance?

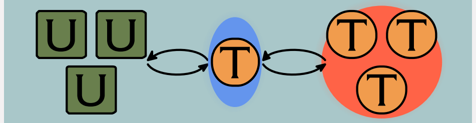
Next = 3^N measurements?

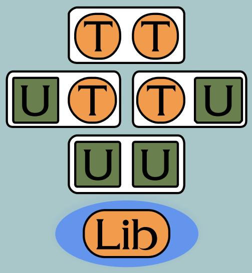


Need an alternative method to measure performance

Q2. Are the benefits significant?

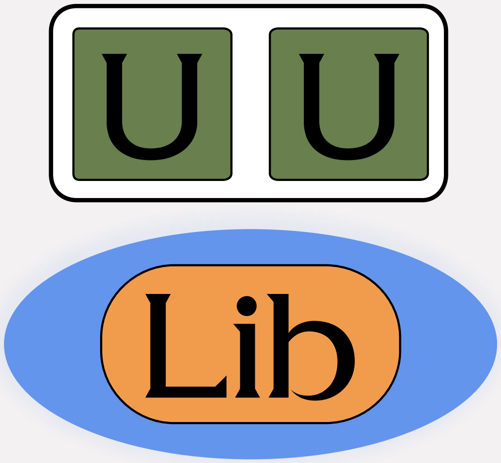


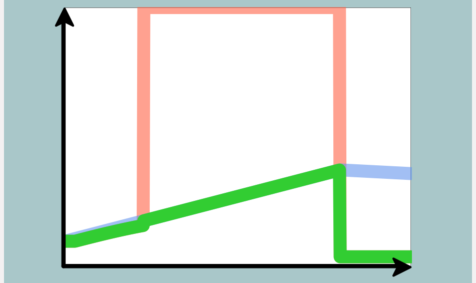


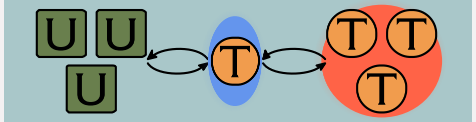


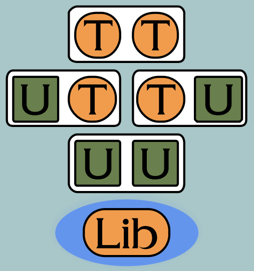
Q2. Are the benefits significant?

Goal: change lib, improve overall



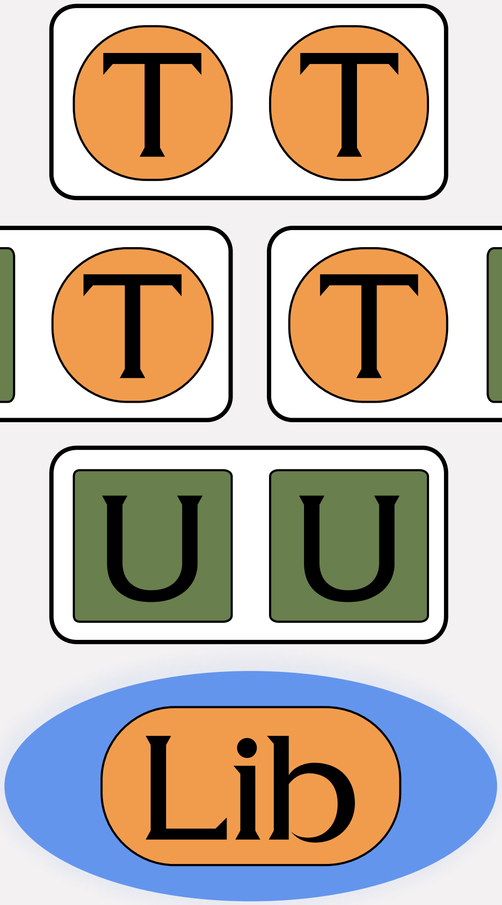


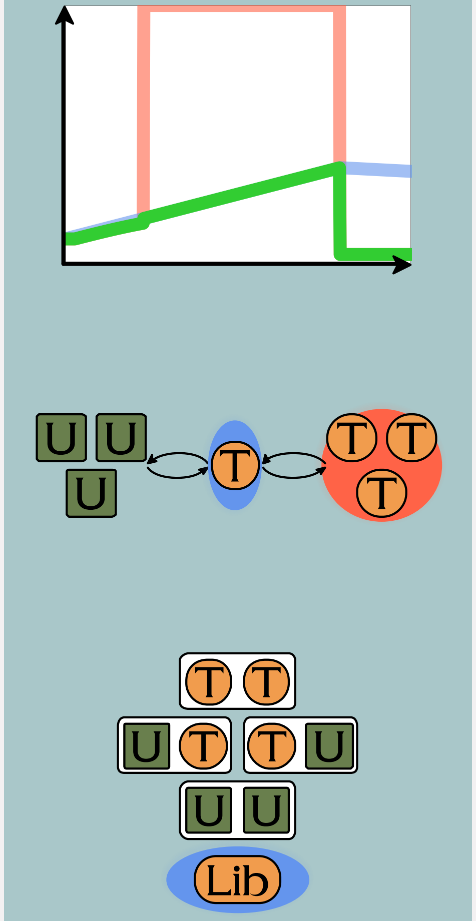


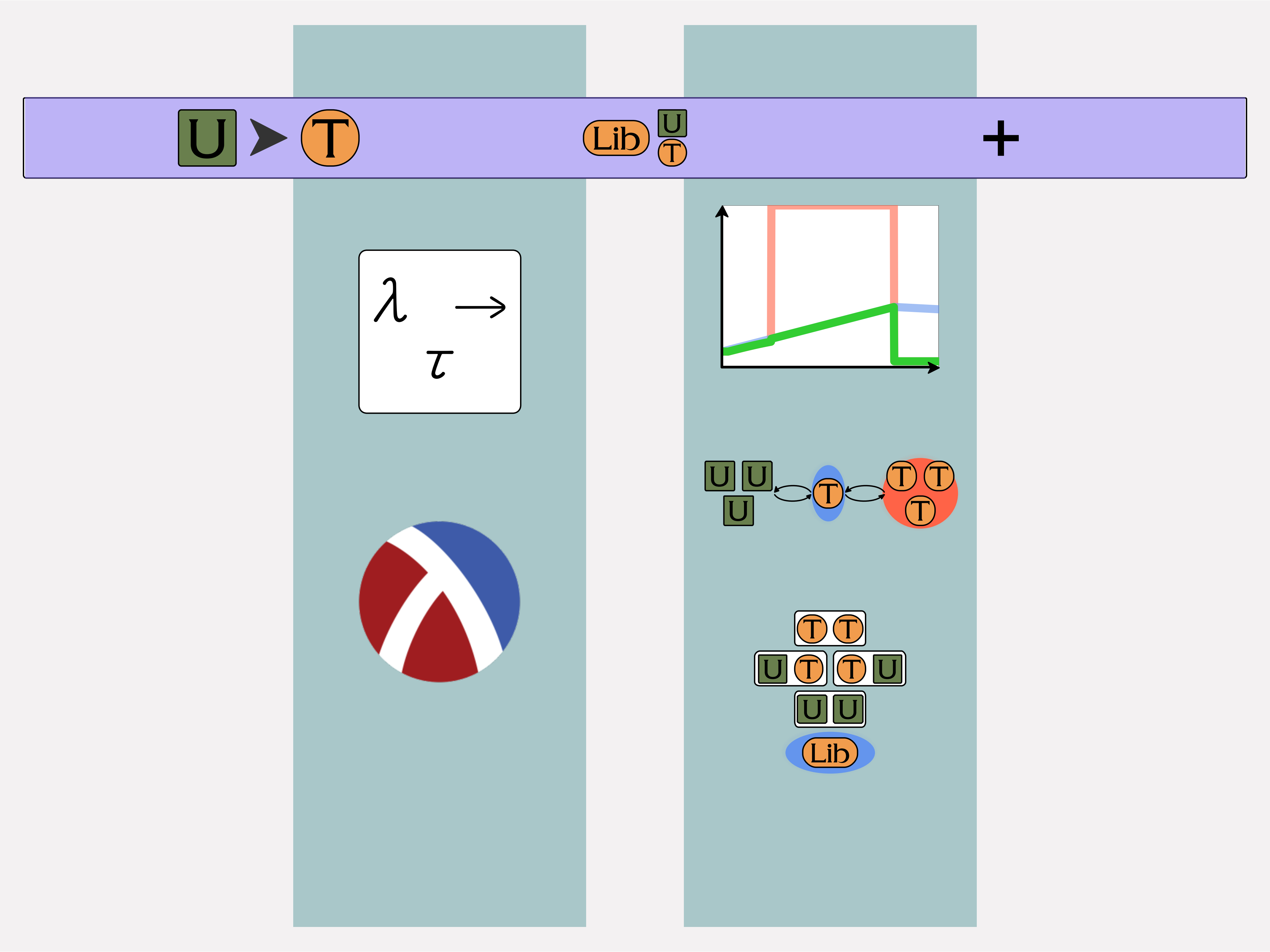


Q2. Are the benefits significant?

Goal: change lib, improve overall






U
T
Lib
U
T
+
λ →
τ
U U
U
T
T T
T
T T
U T
T U
U U
Lib

Timeline

[ ] measure the performance of honest types

[ ] try to directly improve performance

[ ] formally classify alternative types

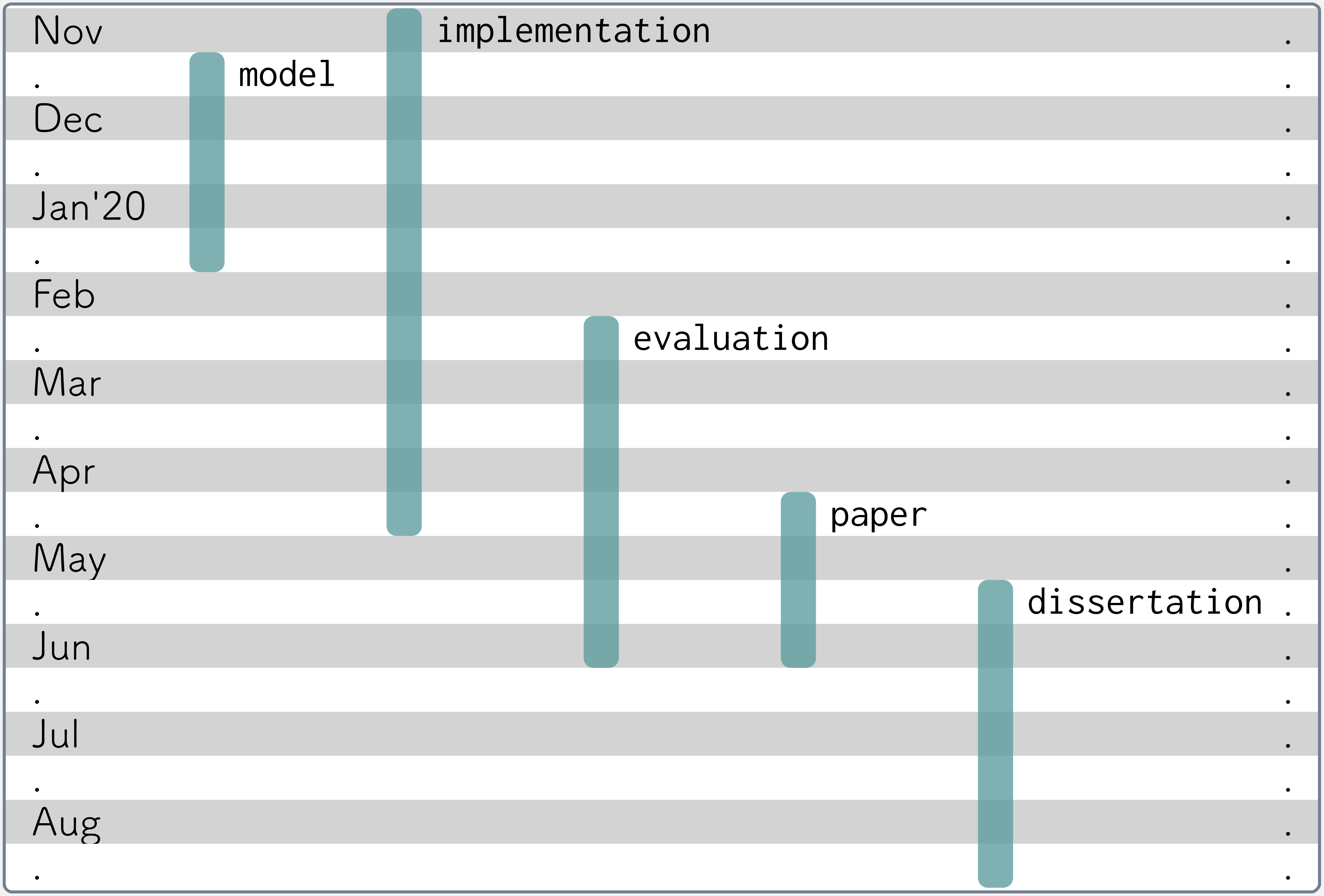
[ ] develop a combined model, measure combined performance

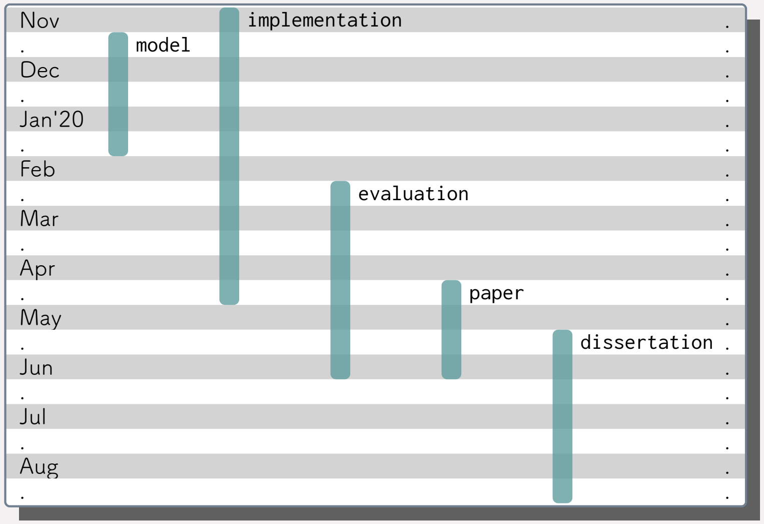
[✓] measure the performance of honest types JFP 2019 POPL 2016

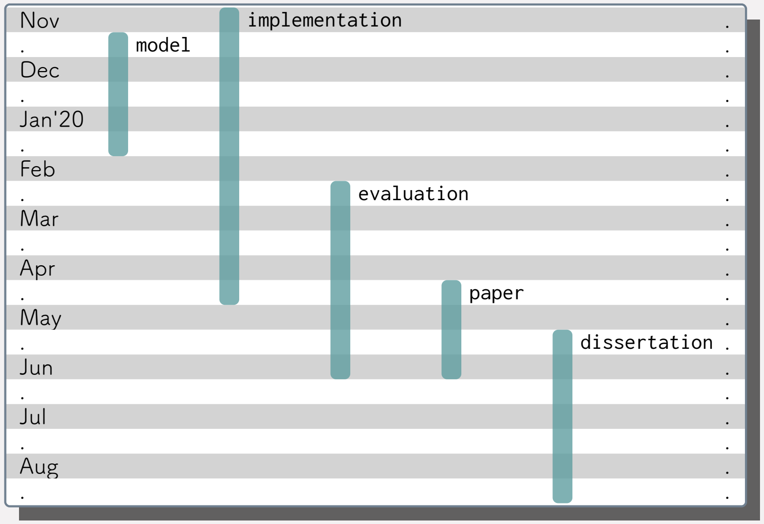
[✓] try to directly improve performance OOPSLA 2018

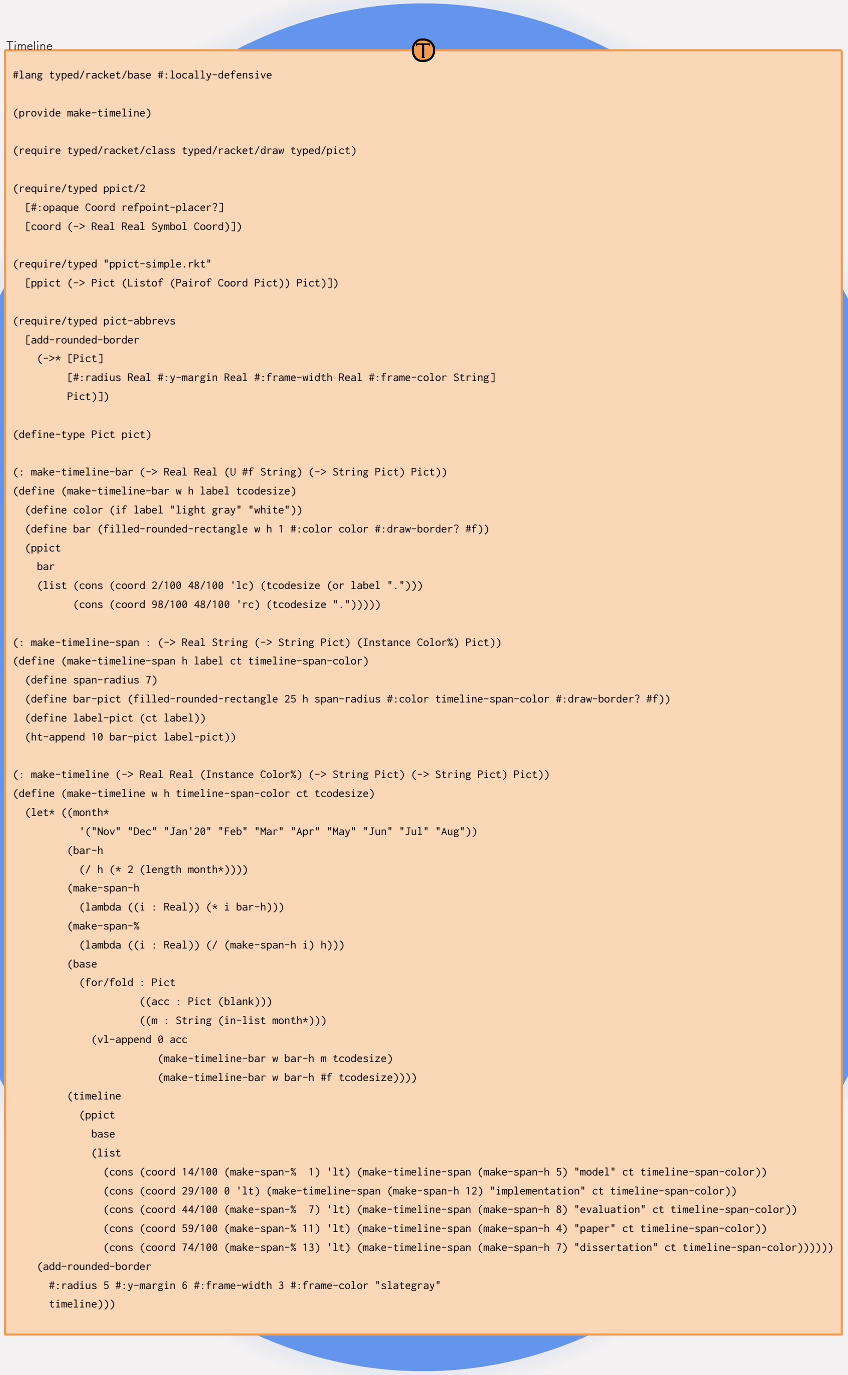
[✓] formally classify alternative types OOPSLA 2019 ICFP 2018

[ ] develop a combined model, measure combined performance


Nov
Dec
Jan'20
Feb
Mar
Apr
May
Jun
Jul
Aug
model
implementation
evaluation
paper
dissertation

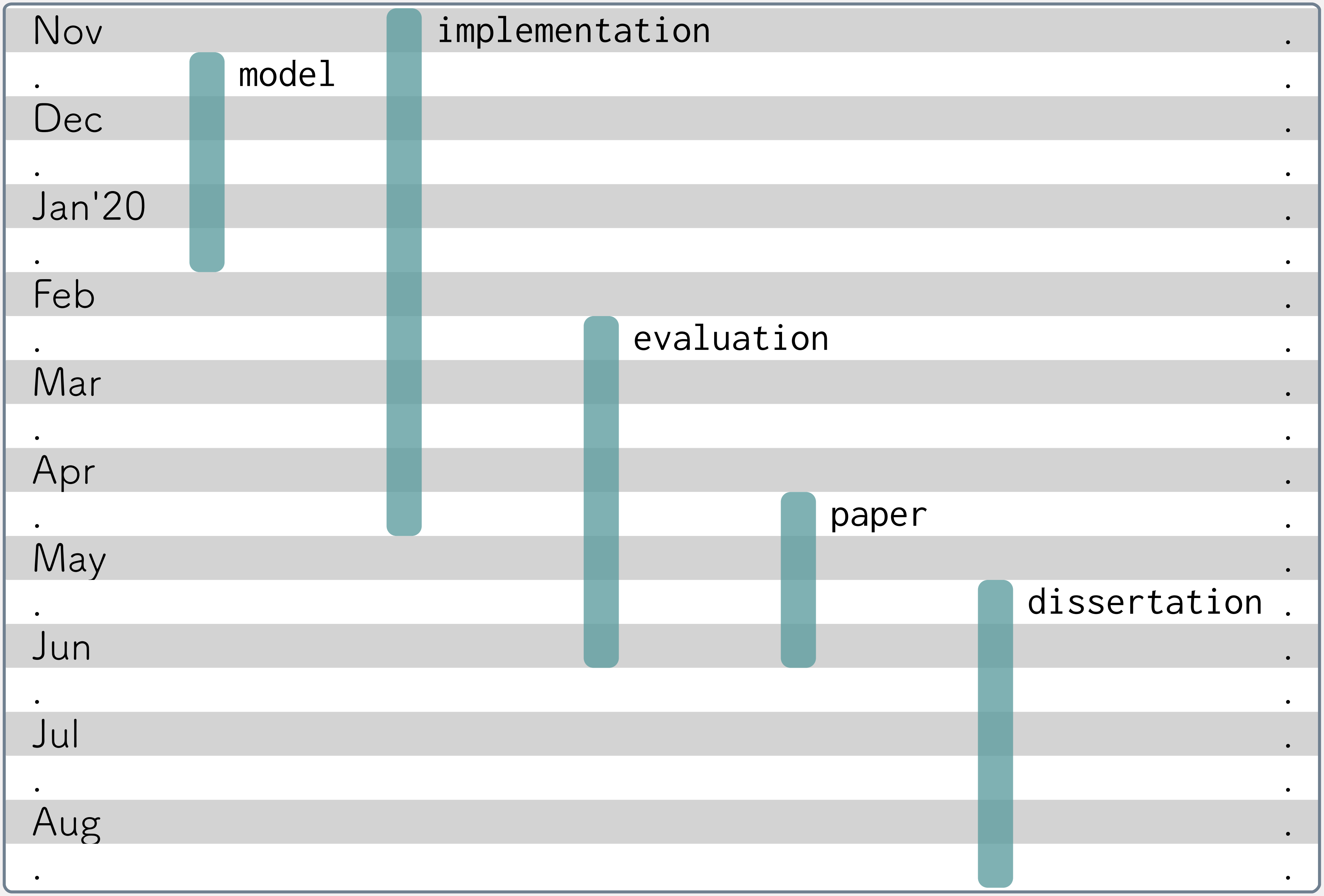

Nov
Dec
Jan'20
Feb
Mar
Apr
May
Jun
Jul
Aug
model
implementation
evaluation
paper
dissertation





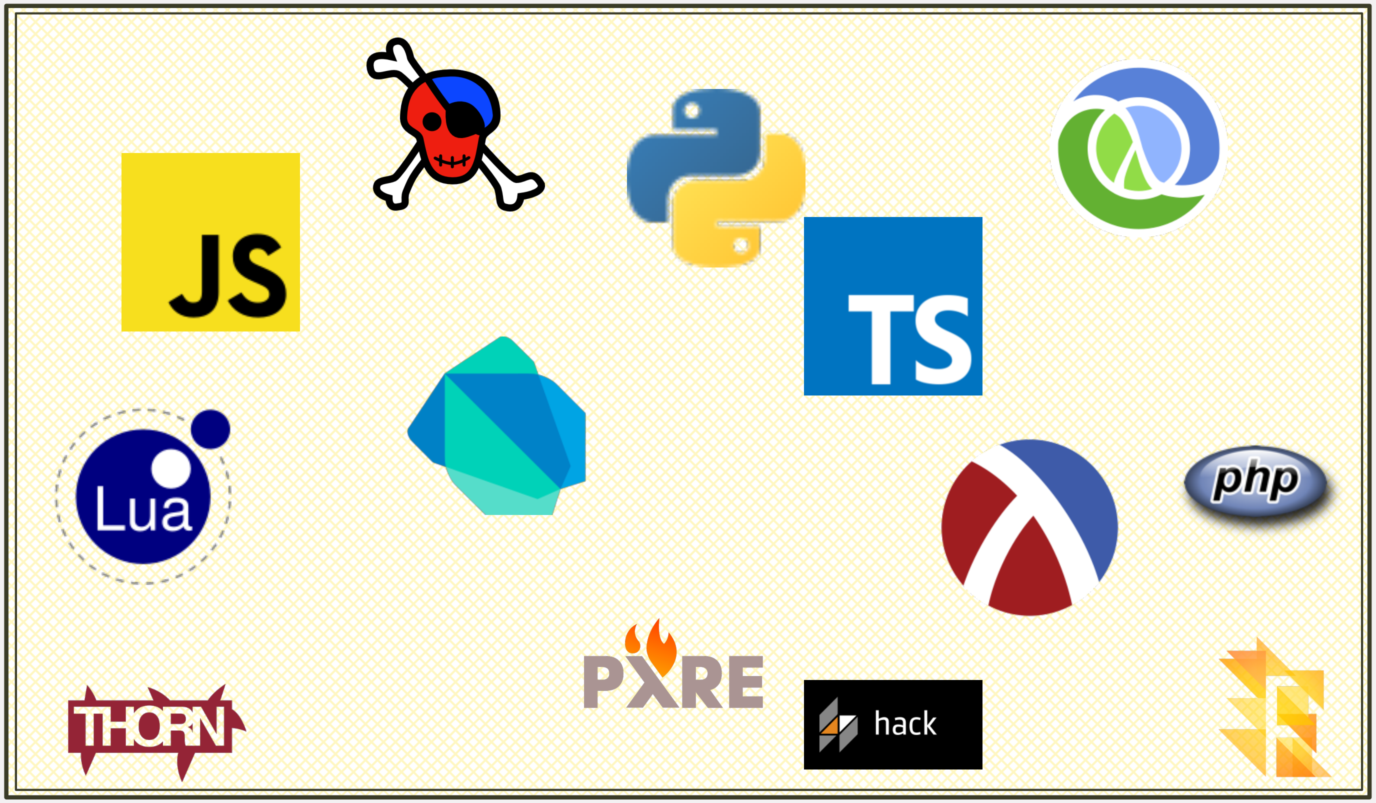
Timeline

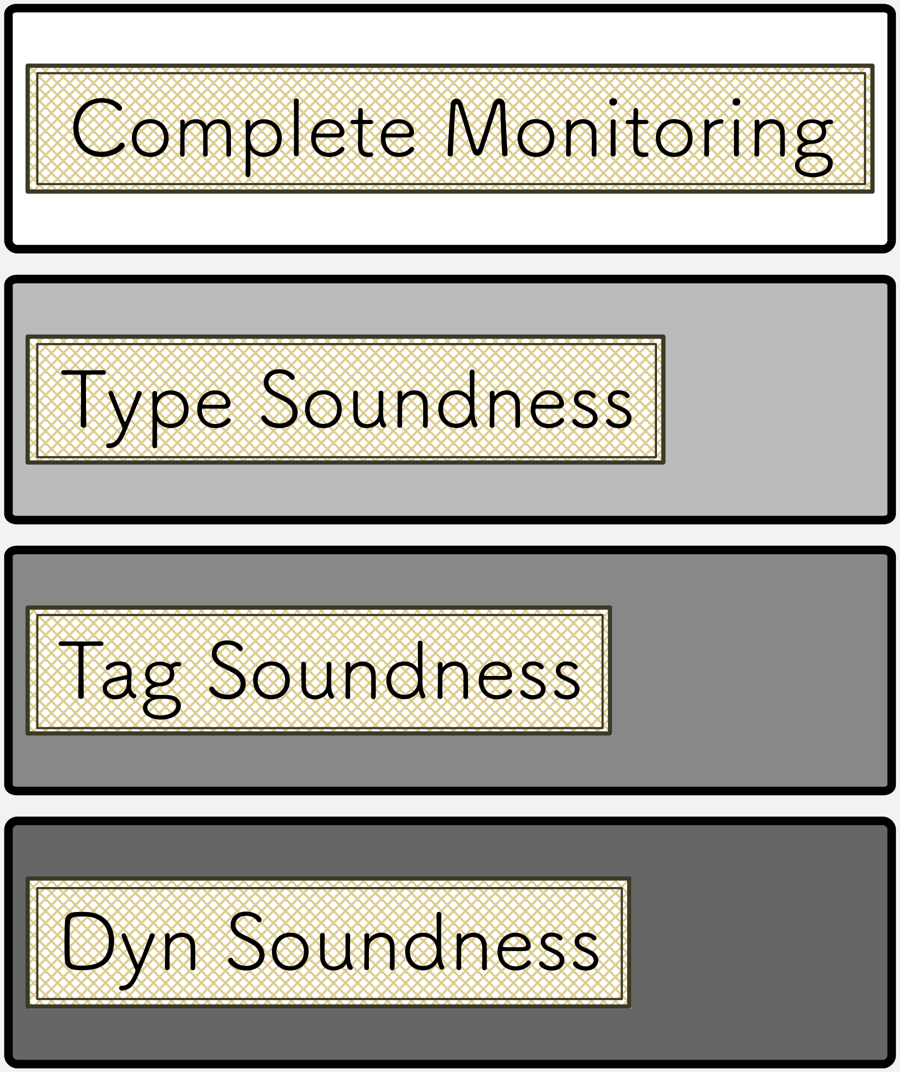
```
#lang typed/racket/base #:locally-defensive

(provide make-timeline)

(require typed/racket/class typed/racket/draw typed/pict)

(require/typed ppict/2
  [#:opaque Coord refpoint-placer?]
  [coord (-> Real Real Symbol Coord)])

(require/typed "ppict-simple.rkt"
  [ppict (-> Pict (Listof (Pairof Coord Pict)) Pict)])

(require/typed pict-abbrevs
  [add-rounded-border
    (->* [Pict]
         [#:radius Real #:y-margin Real #:frame-width Real #:frame-color String]
         Pict)])

(define-type Pict pict)

(: make-timeline-bar (-> Real Real (U #f String) (-> String Pict) Pict))
(define (make-timeline-bar w h label tcodesize)
  (define color (if label "light gray" "white"))
  (define bar (filled-rounded-rectangle w h 1 #:color color #:draw-border? #f))
  (ppict
    bar
    (list (cons (coord 2/100 48/100 'lc) (tcodesize (or label ".")))
          (cons (coord 98/100 48/100 'rc) (tcodesize ".")))))

(: make-timeline-span : (-> Real String (-> String Pict) (Instance Color%) Pict))
(define (make-timeline-span h label ct timeline-span-color)
  (define span-radius 7)
  (define bar-pict (filled-rounded-rectangle 25 h span-radius #:color timeline-span-color #:draw-border? #f))
  (define label-pict (ct label))
  (ht-append 10 bar-pict label-pict))

(: make-timeline (-> Real Real (Instance Color%) (-> String Pict) (-> String Pict) Pict))
(define (make-timeline w h timeline-span-color ct tcodesize)
  (let* ((month*
           '("Nov" "Dec" "Jan'20" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug"))
         (bar-h
           (/ h (* 2 (length month*))))
         (make-span-h
           (lambda ((i : Real)) (* i bar-h)))
         (make-span-%
           (lambda ((i : Real)) (/ (make-span-h i) h)))
         (base
           (for/fold : Pict
                     ((acc : Pict (blank)))
                     ((m : String (in-list month*)))
             (vl-append 0 acc
                        (make-timeline-bar w bar-h m tcodesize)
                        (make-timeline-bar w bar-h #f tcodesize))))
         (timeline
           (ppict
             base
             (list
               (cons (coord 14/100 (make-span-%  1) 'lt) (make-timeline-span (make-span-h 5) "model" ct timeline-span-color))
               (cons (coord 29/100 0 'lt) (make-timeline-span (make-span-h 12) "implementation" ct timeline-span-color))
               (cons (coord 44/100 (make-span-%  7) 'lt) (make-timeline-span (make-span-h 8) "evaluation" ct timeline-span-color))
               (cons (coord 59/100 (make-span-% 11) 'lt) (make-timeline-span (make-span-h 4) "paper" ct timeline-span-color))
               (cons (coord 74/100 (make-span-% 13) 'lt) (make-timeline-span (make-span-h 7) "dissertation" ct timeline-span-color))))))
    (add-rounded-border
      #:radius 5 #:y-margin 6 #:frame-width 3 #:frame-color "slategray"
      timeline)))
```


Nov
Dec
Jan'20
Feb
Mar
Apr
May
Jun
Jul
Aug
model
implementation
evaluation
paper
dissertation

The End

Q. Does migratory typing benefit from a combination of honest and lying types?


JS
TS
Lua
php
THORN
PYRE
hack



Complete Monitoring

types predict behavior

Type Soundness

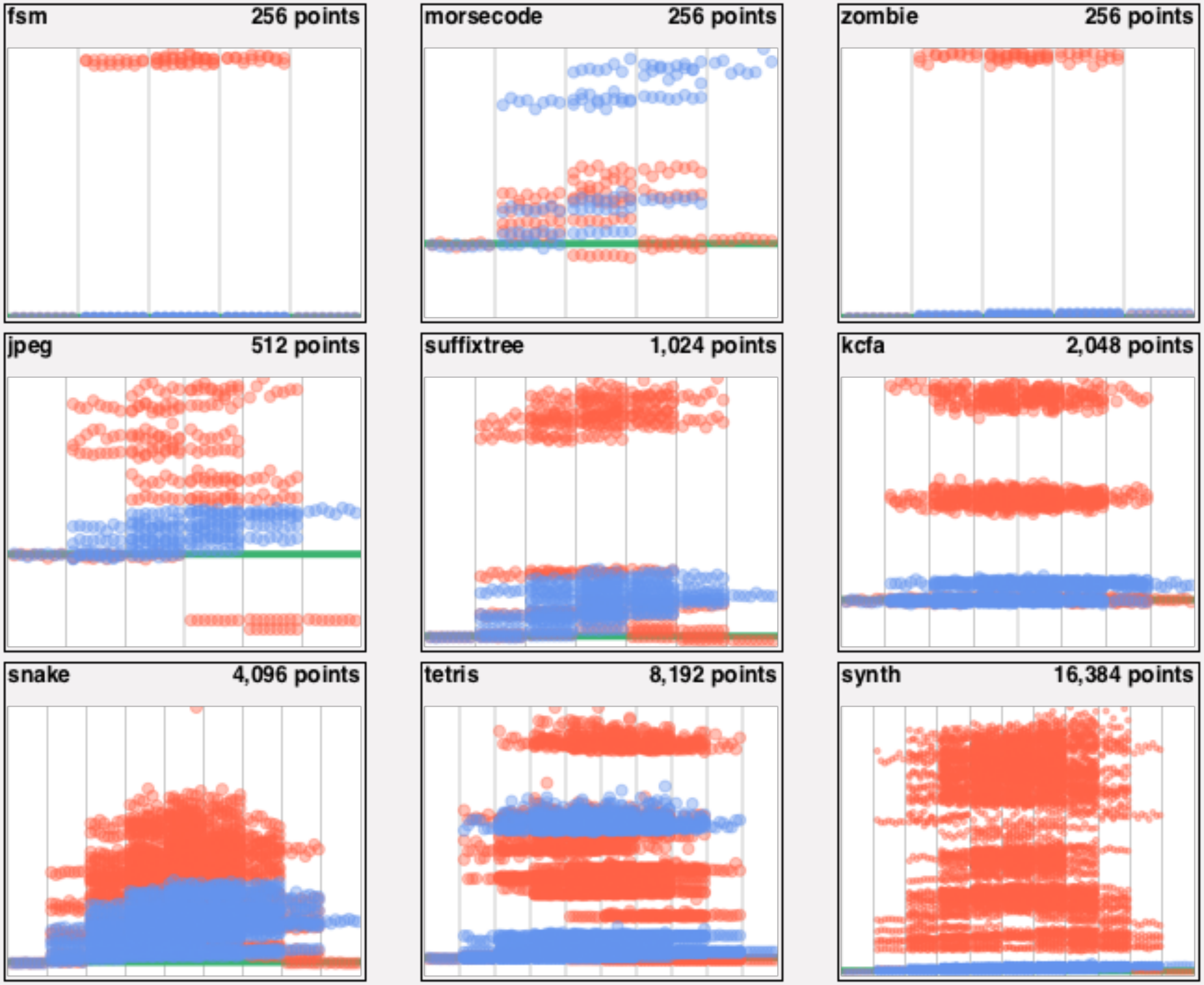
types predict behavior in typed code, nothing in untyped code

Tag Soundness

types predict shapes in typed code, nothing in untyped code

Dyn Soundness

types predict nothing



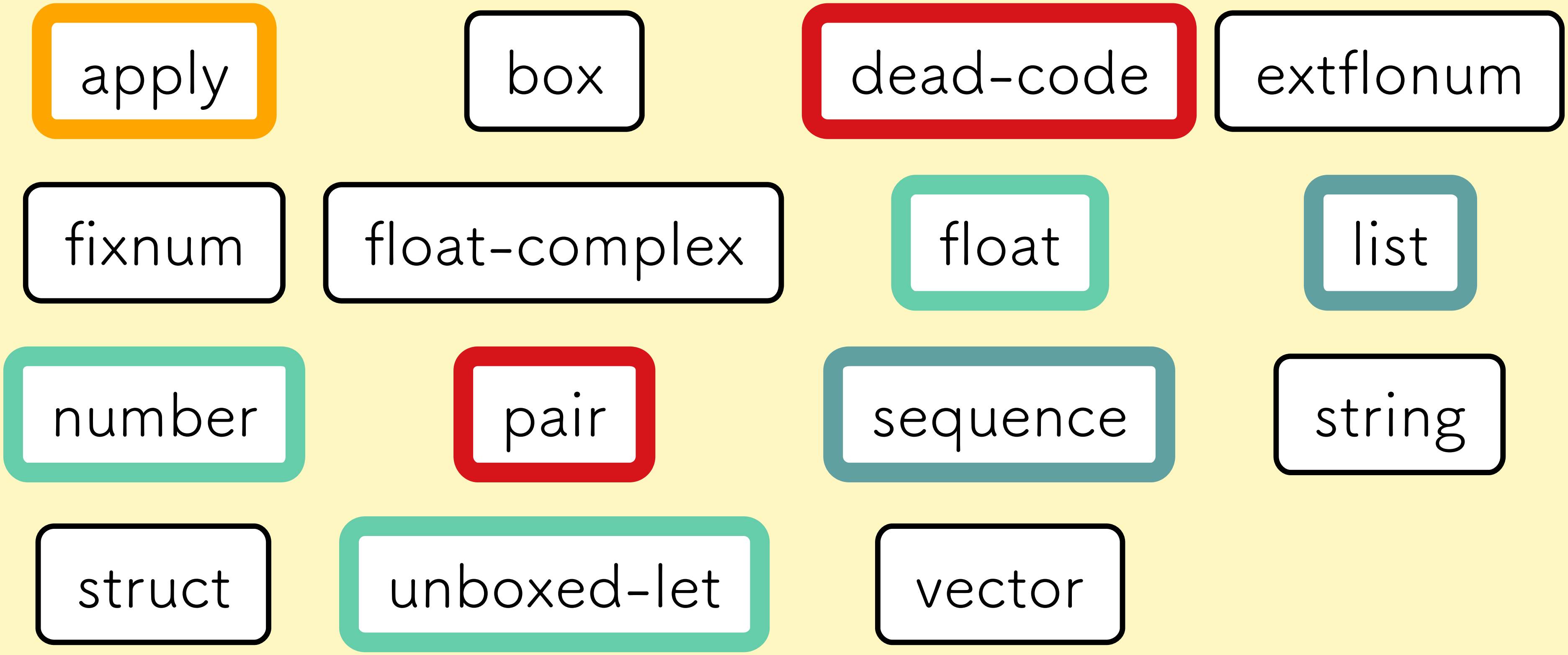
— = Untyped Perf. 🔴 = Natural 🔵 = Transient

Expressiveness

DLS 18

Preston Tunnell Wilson, Ben Greenman,
Justin Pombrio, and Shriram Krishnamurthi

TR Optimizations



apply

box

dead-code

extflonum

fixnum

float-complex

float

list

number

pair

sequence

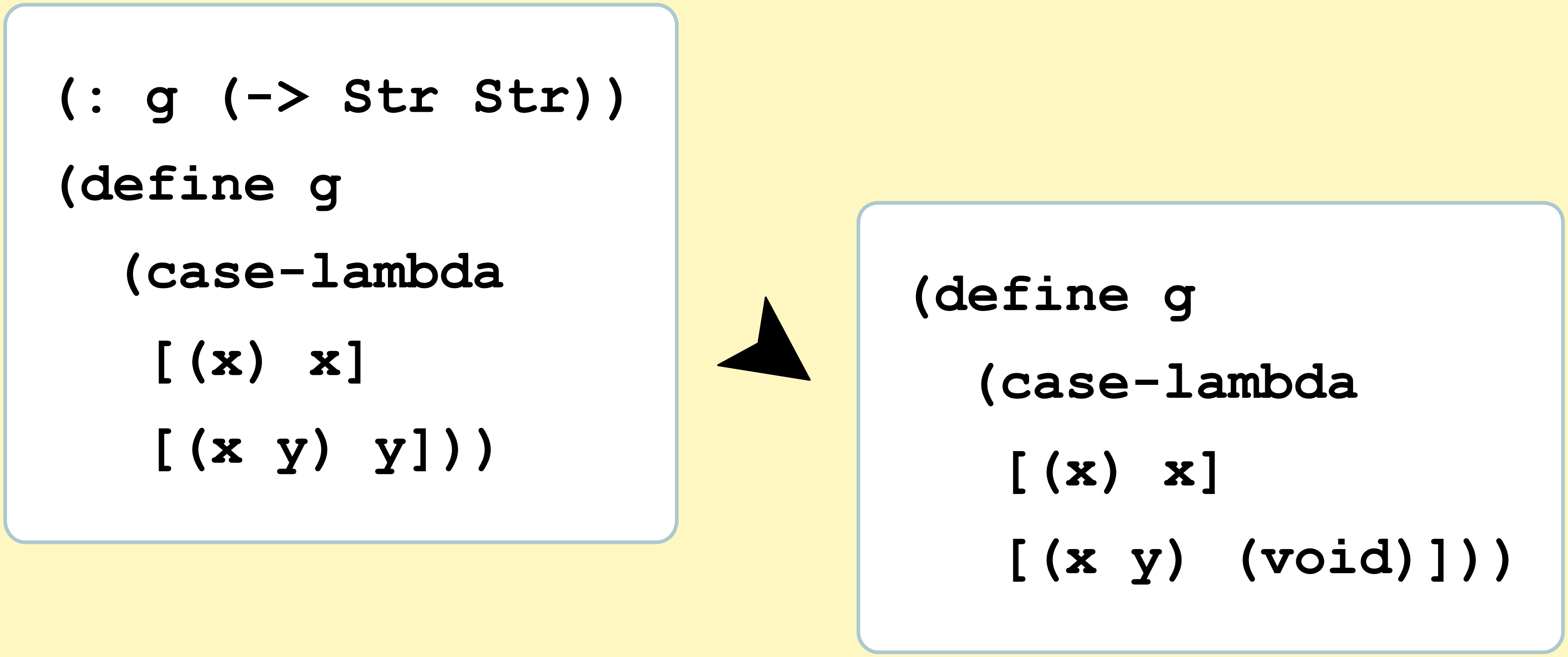
string

struct

unboxed-let

vector

dead-code = unsafe for Transient



```
(: g (-> Str Str))
(define g
  (case-lambda
   [(x) x]
   [(x y) y]))
```

```
(define g
  (case-lambda
   [(x) x]
   [(x y) (void)]))
```

Problem: untyped code can call `(g 0 1)`

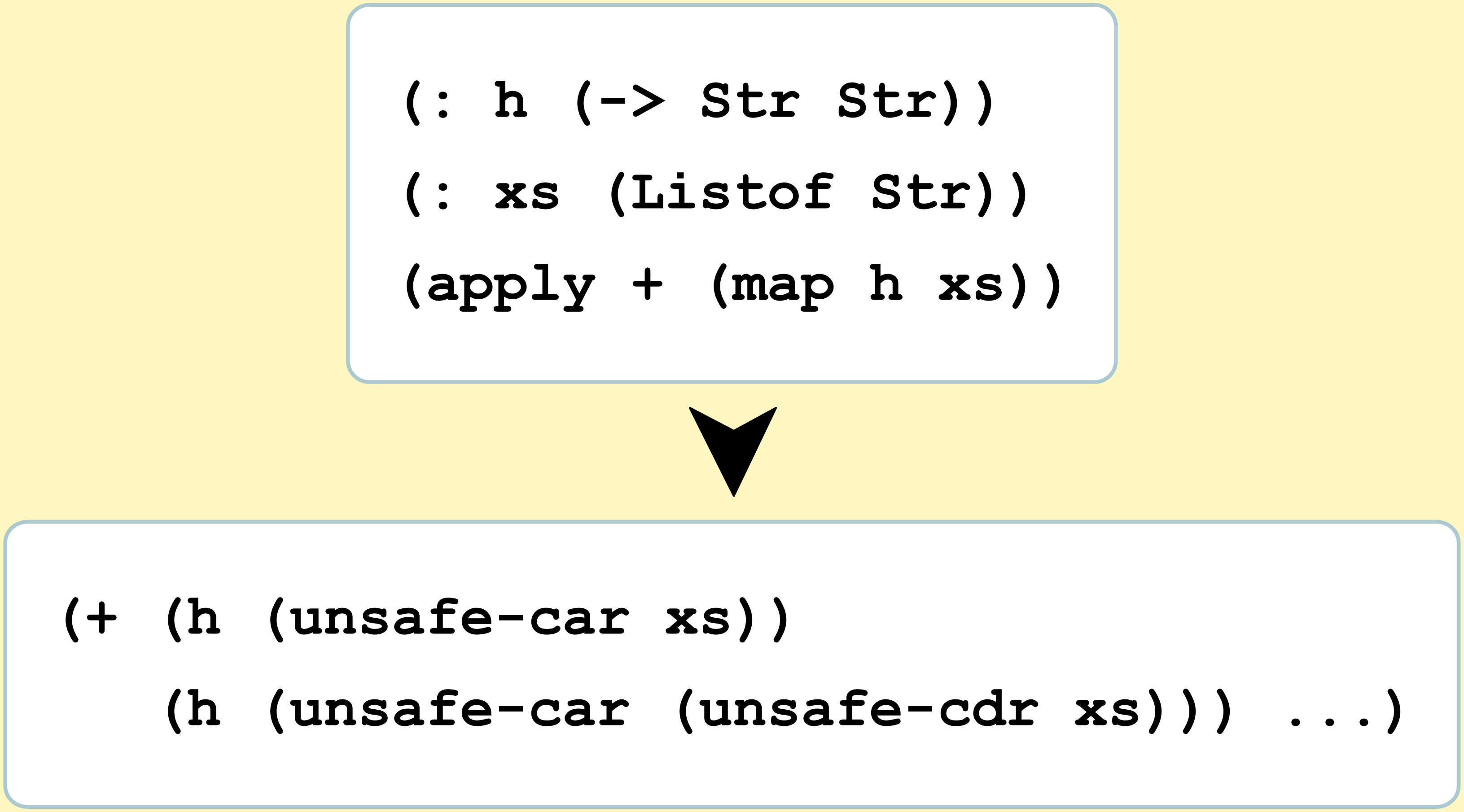
pair = unsound for Transient

```
(: x (Pairof (Pairof Nat Int) Str))
(cdar x)
```

↓

```
(unsafe-cdr (unsafe-car x))
```

Problem: no guarantee `(car x)` is a pair apply = safe but risky for Transient

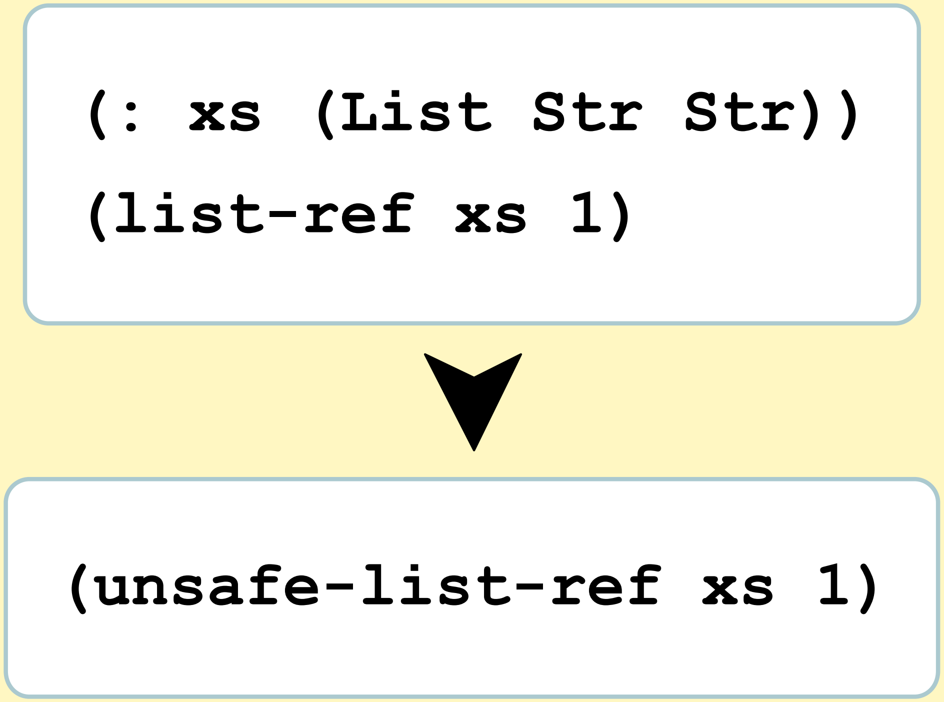


```
(: h (-> Str Str))
(: xs (Listof Str))
(apply + (map h xs))
```

```
(+ (h (unsafe-car xs))
   (h (unsafe-car (unsafe-cdr xs))) ...)
```

Caution: **`h`** must check inputs

list sequence = force choice for ⌊T⌋



```
(: xs (List Str Str))
(list-ref xs 1)
```

```
(unsafe-list-ref xs 1)
```

Note: ⌊**`List Str Str`**⌋ needs more than a tag check

number = $\lfloor T \rfloor$ is more than a tag check

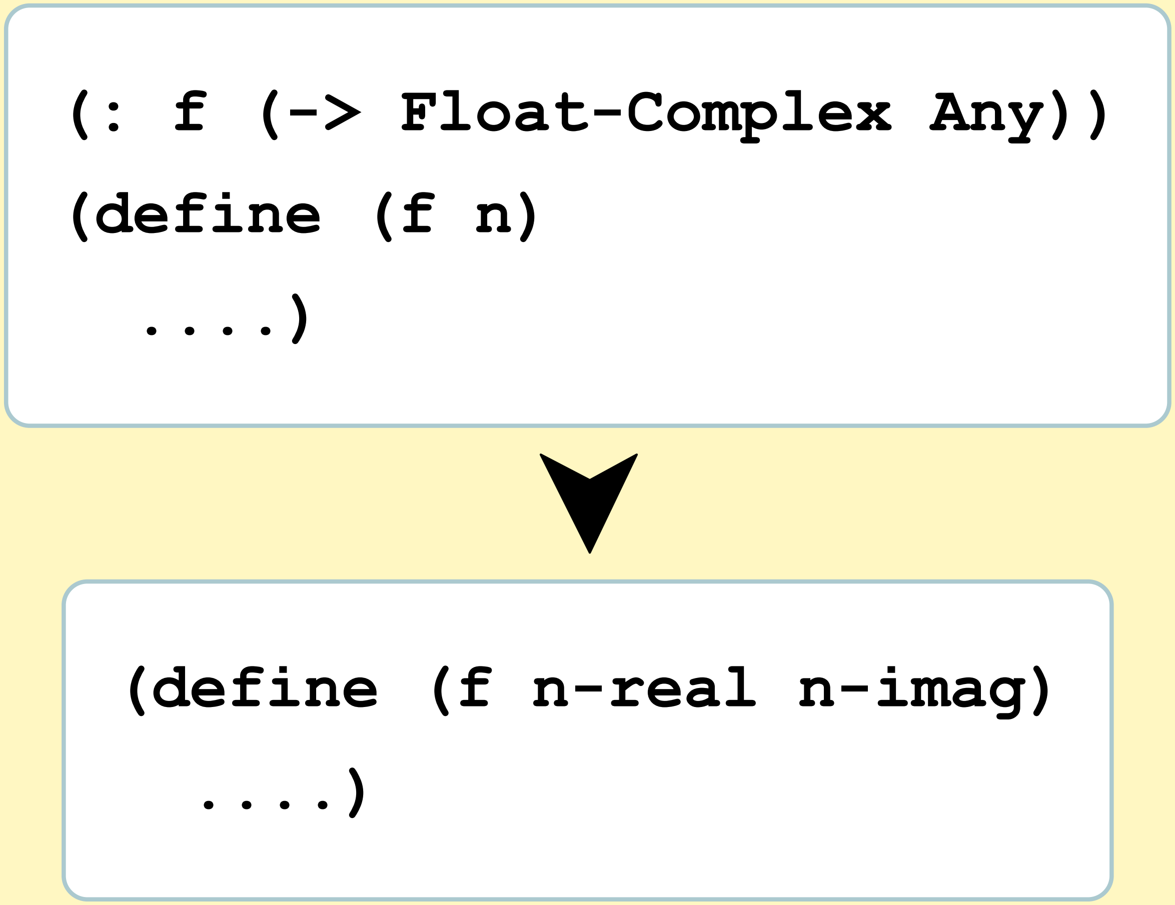
Natural

Exact-Nonnegative-Integer

Nonpositive-Inexact-Real

ExtFlonum-Negative-Zero

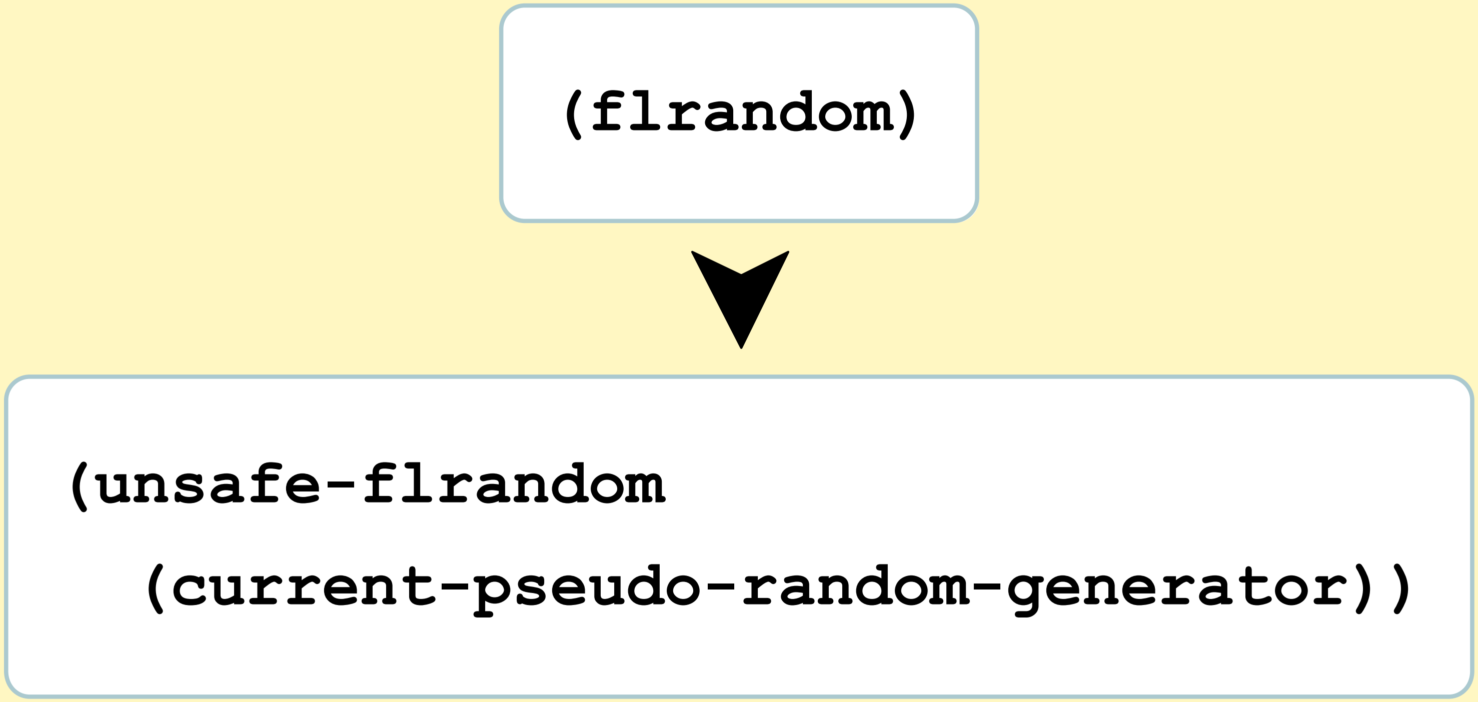
unboxed-let = safe with escape analysis



```
(: f (-> Float-Complex Any))
(define (f n)
  ....)
```

```
(define (f n-real n-imag)
  ....)
```

float = false alarm



```
(flrandom)
```

```
(unsafe-flrandom
   (current-pseudo-random-generator))
```

Ok because the PRNG parameter checks inputs