



# Deep and Shallow Types for Gradual Languages

Ben Greenman  
2022-06-16

Northeastern  
-> Brown\*  
-> Utah



Typed



Untyped

Q. Should your PL be typed or untyped?

Typed



Untyped

Q. Should your PL be typed or untyped?

**Gradual typing** says **yes to both**

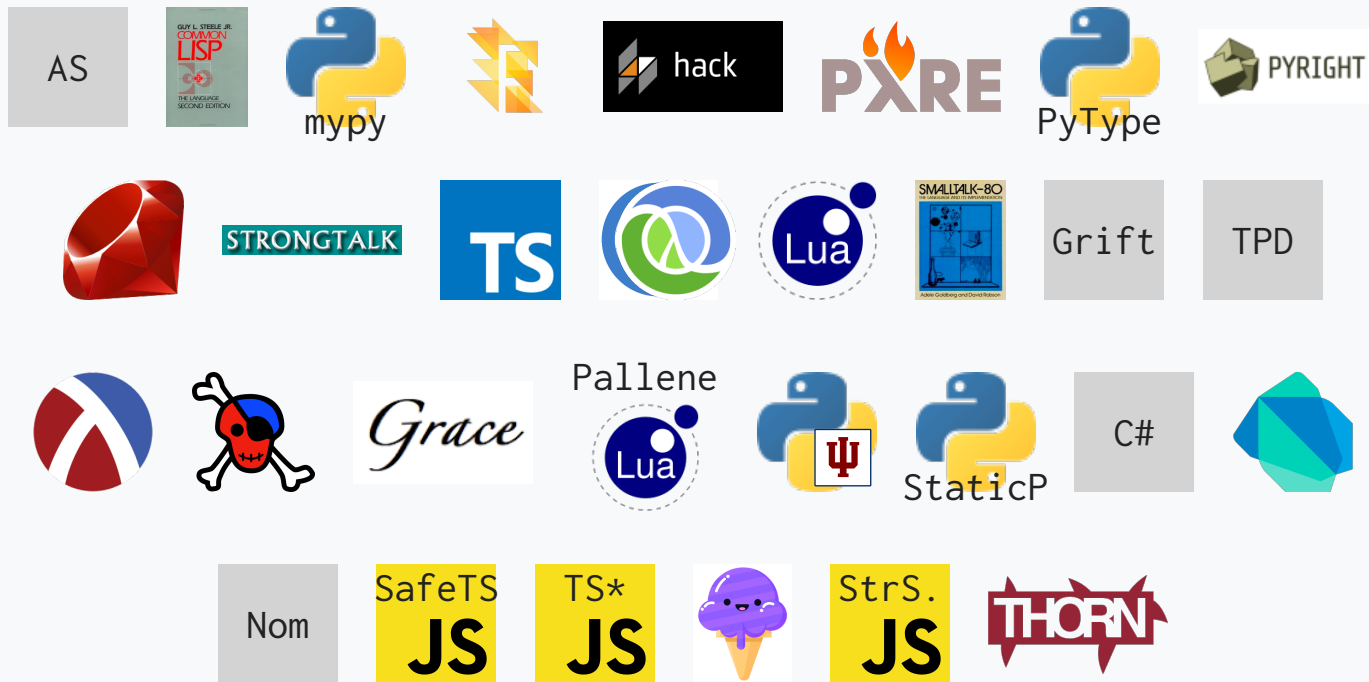
**"best" of two worlds**

## Great Idea!

Inspired MANY Languages Over 16+ Years

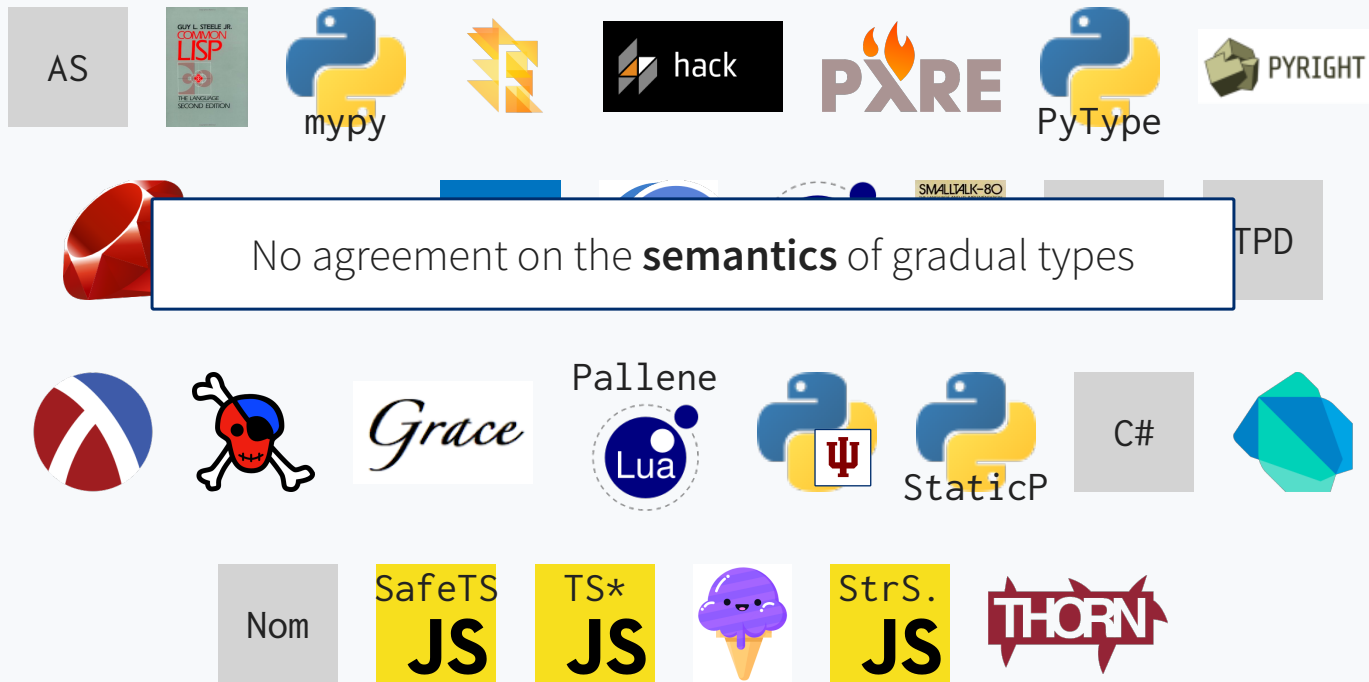
# Great Idea!



Inspired MANY Languages Over 16+ Years




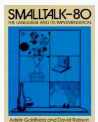
# Great Idea!

Inspired MANY Languages Over 16+ Years





StaticP  Concrete  Nom  SafeTS JS

TS\* JS  StrS. JS THORN



Natural  
Grift TPD



Transient ne  
Grace  

AS



 mypy

FXRE 

Erasure 

hack 

 PyType

 PYRIGHT



STRONGTALK

TS







### Concrete



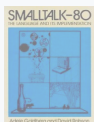
C#



Nom



### Natural



Grift

TPD



4 leading semantics because of a tradeoff:  
type **guarantees** vs. **performance** costs  
vs. **expressiveness**

### Transient

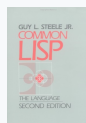


ne



### Erasure

AS



STRONGTALK





StaticP  C# **Concrete**  Nom SafeTS JS

TS\* JS  StrS. JS THORN



**Natural**  
Grift TPD



**Transient**  
Grace  ne 

AS



 mypy



**Erasure**



 PyType





STRONGTALK

TS

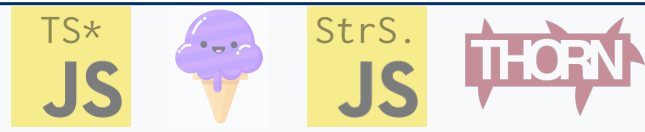




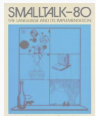
Concrete



limited interop w/ untyped



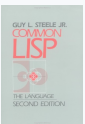
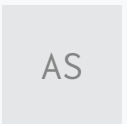
Natural



Transient



Erasure



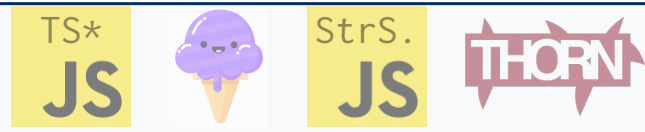
STRONGTALK



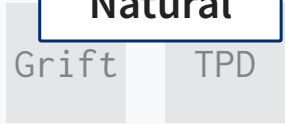
Concrete



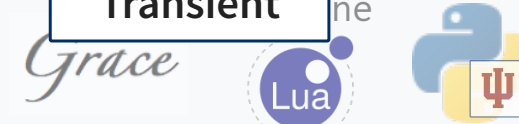
limited interop w/ untyped



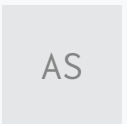
Natural



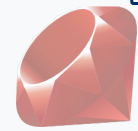
Transient



Erasure



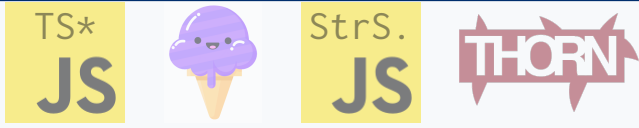
unsound interop



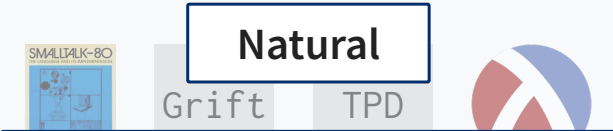
### Concrete



■ limited interop w/ untyped



### Natural



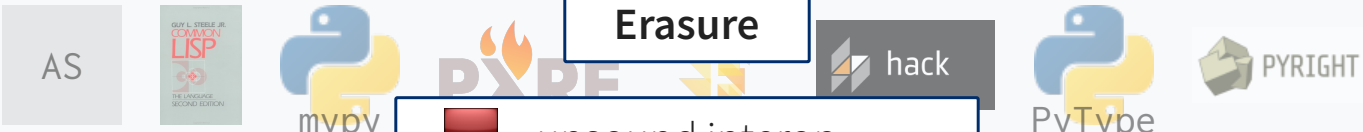
● strong, slow types  
Deep

### Transient



● fast, wrong types  
Shallow

### Erasure



■ unsound interop



# Starting Point

Natural

● strong, slow types  
Deep



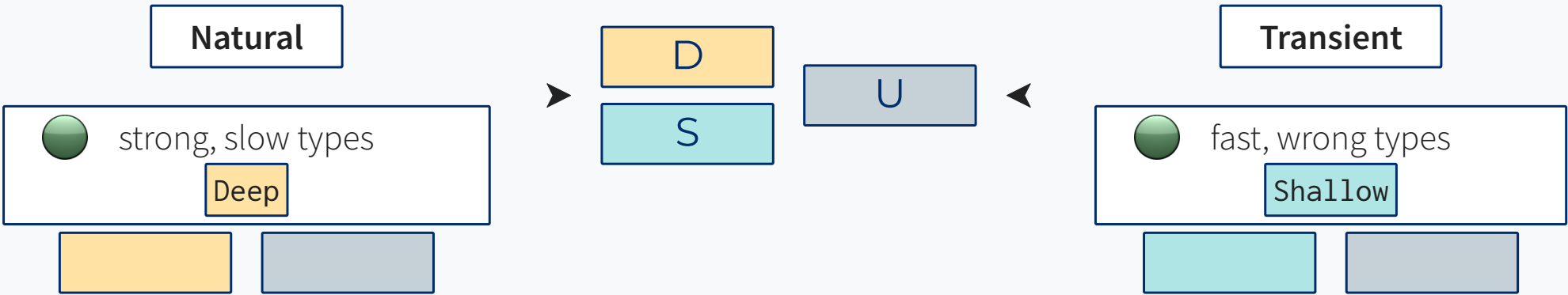
Transient

● fast, wrong types  
Shallow



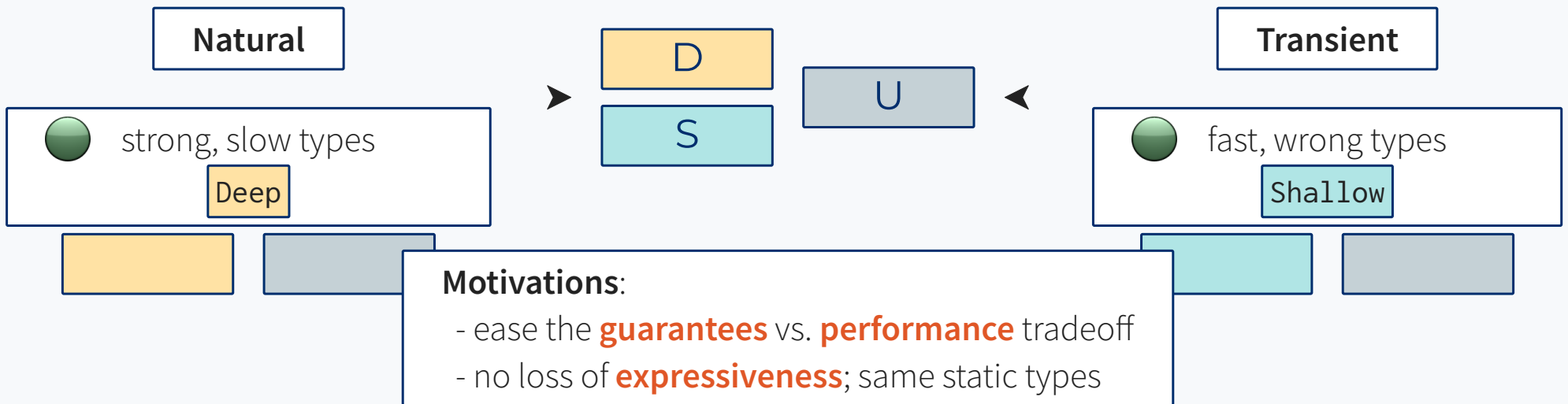
# Starting Point

RQ. Can Natural and Transient interoperate?



# Starting Point

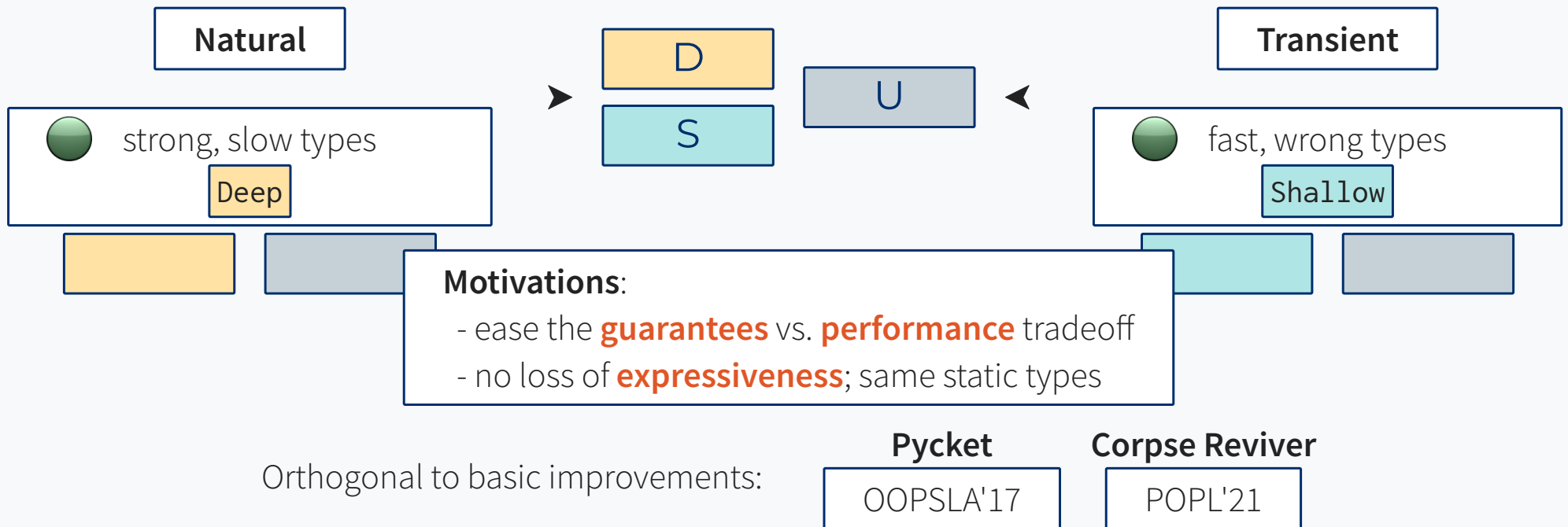
RQ. Can Natural and Transient interoperate?





# Starting Point

RQ. Can Natural and Transient interoperate?



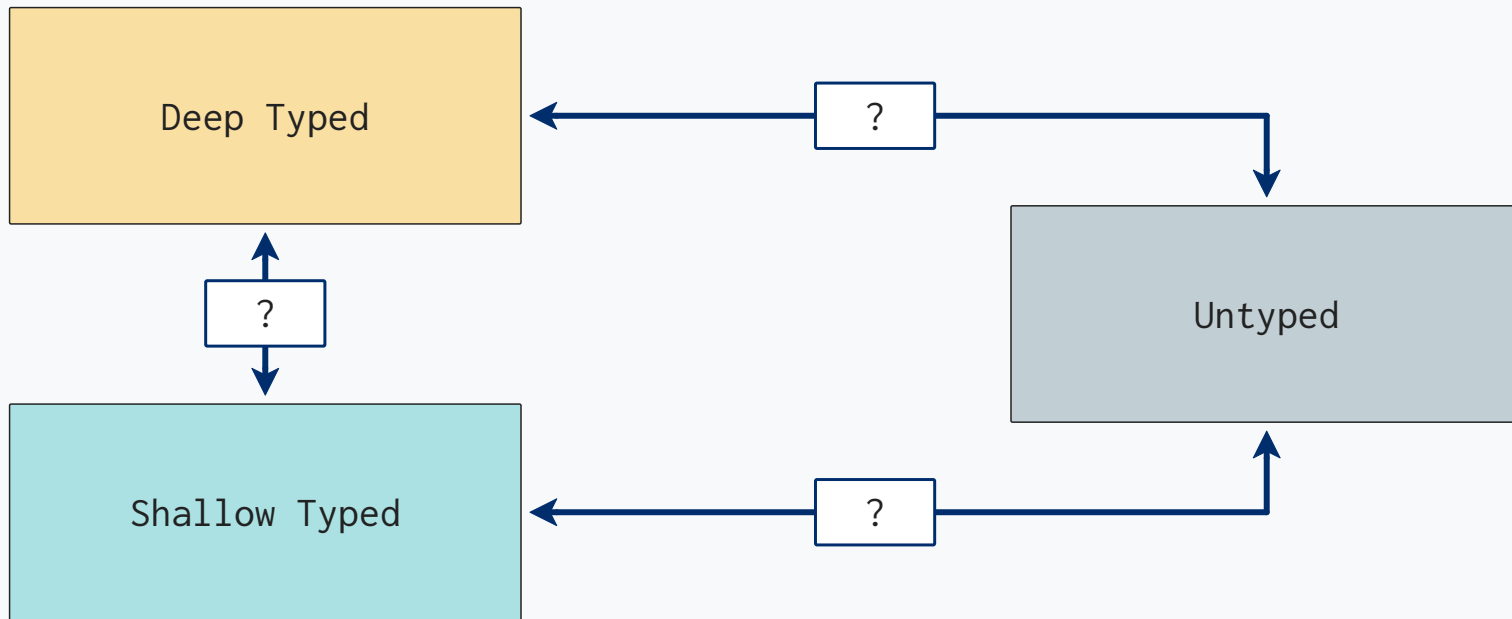
## Key Technical Question: How to Enforce Types at Boundaries?

Deep Typed

Shallow Typed

Untyped

# Key Technical Question: How to Enforce Types at Boundaries?

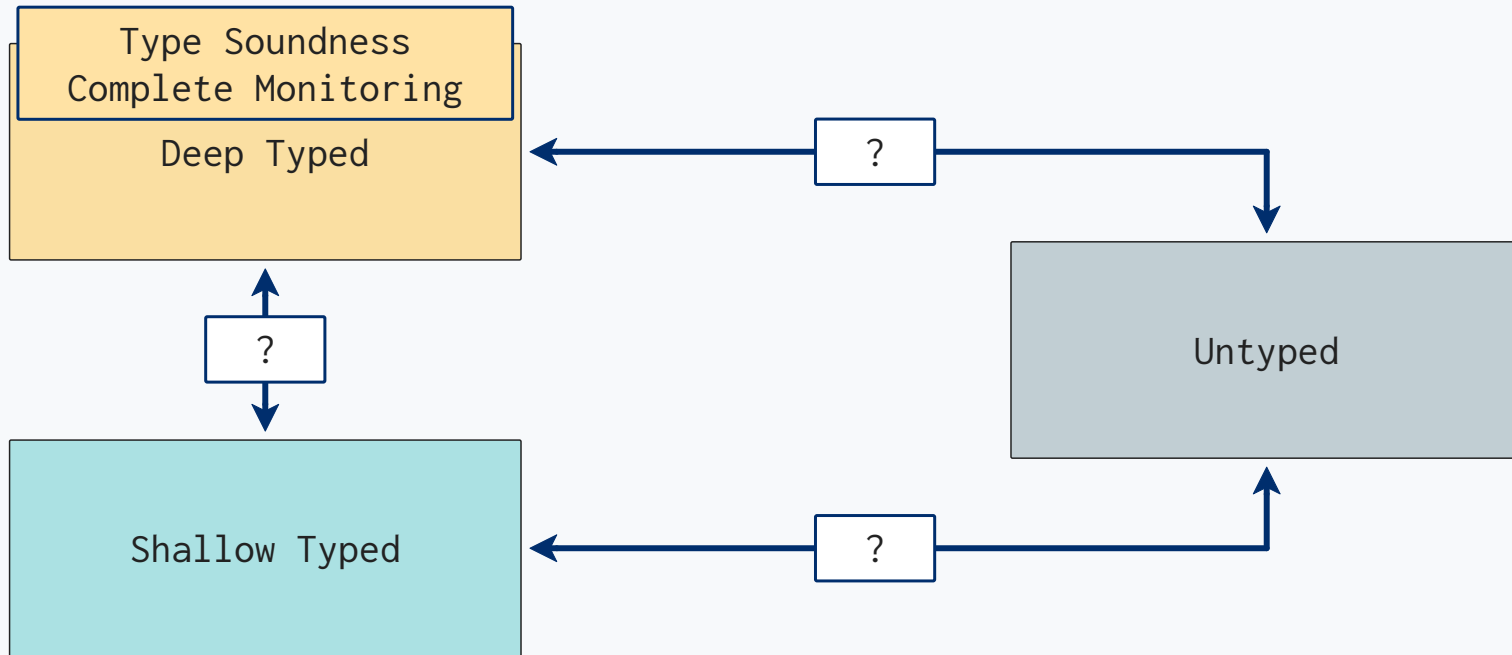


While preserving their formal properties

OOPSLA'19

ICFP'18

# Key Technical Question: How to Enforce Types at Boundaries?

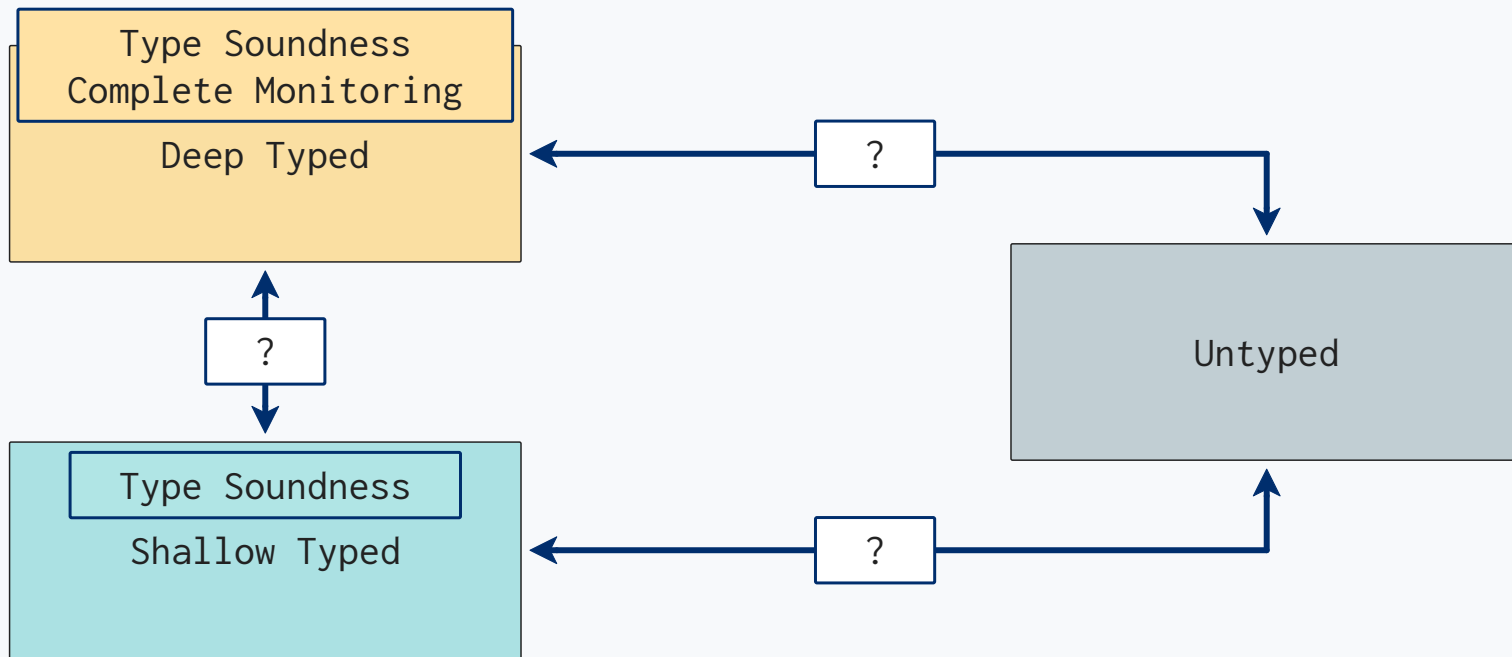


While preserving their formal properties

OOPSLA'19

ICFP'18

# Key Technical Question: How to Enforce Types at Boundaries?

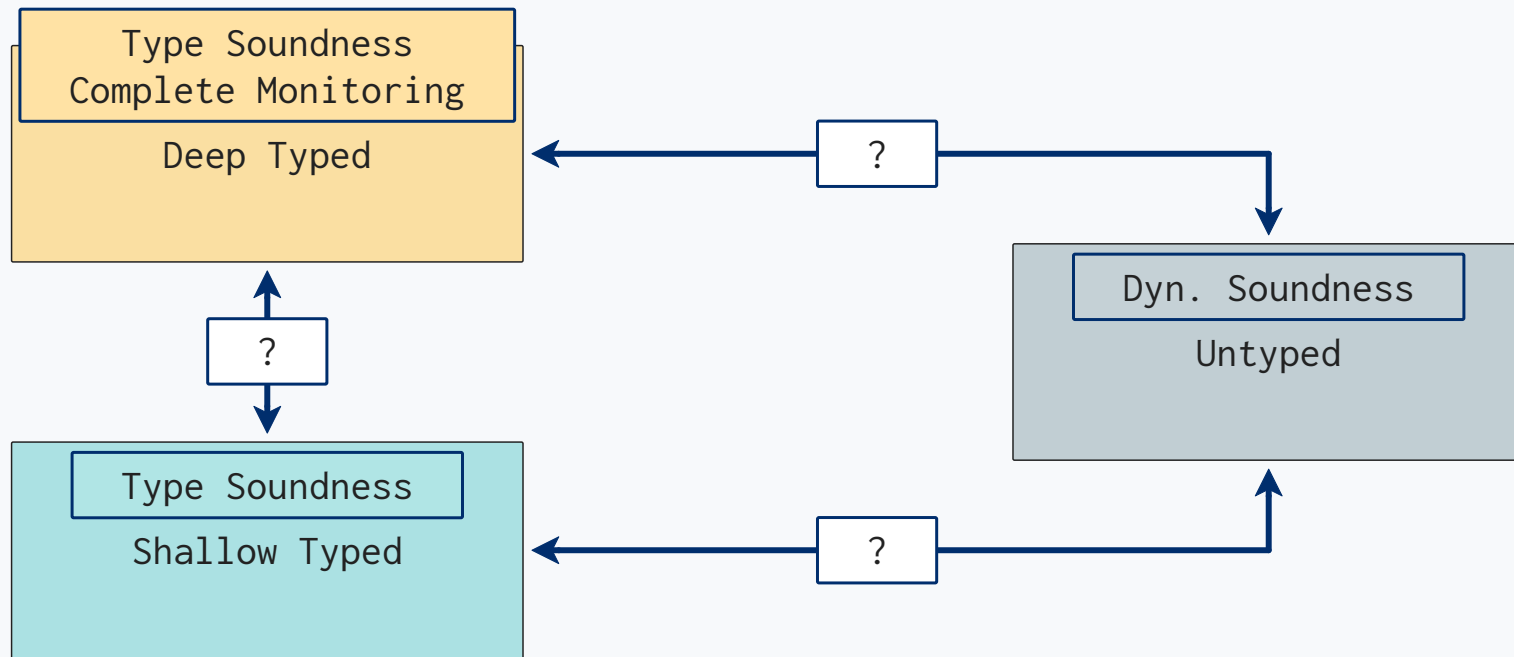


While preserving their formal properties

OOPSLA'19

ICFP'18

# Key Technical Question: How to Enforce Types at Boundaries?



While preserving their formal properties

OOPSLA'19

ICFP'18

**Key Technical Question:**  
How to Enforce Types at Boundaries?

## Key Technical Question:

How to Enforce Types at Boundaries?

First of all:

Q. How does **Natural** enforce **Deep** types?

Q. How does **Transient** enforce **Shallow** types?



Q. How does **Natural** enforce **Deep** types?



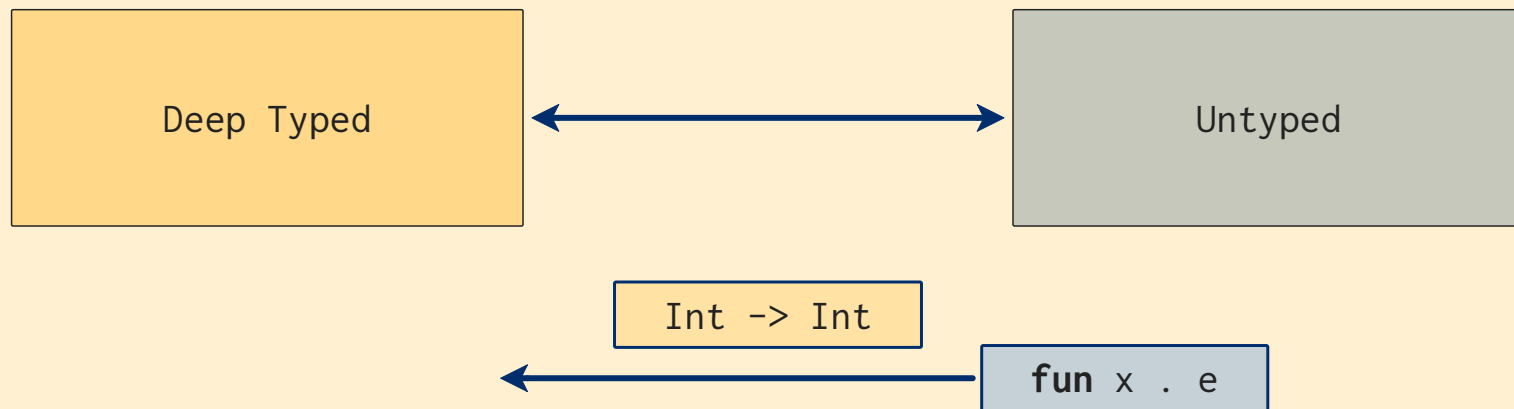
Q. How does **Natural** enforce **Deep** types?

A. Use **wrappers** to guard boundaries



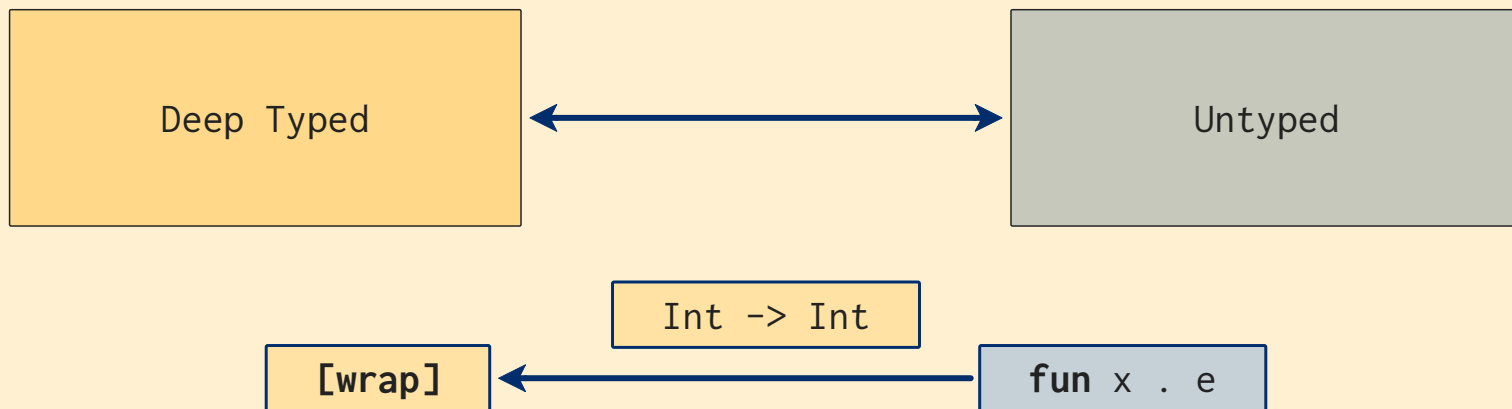
Q. How does **Natural** enforce **Deep** types?

A. Use **wrappers** to guard boundaries



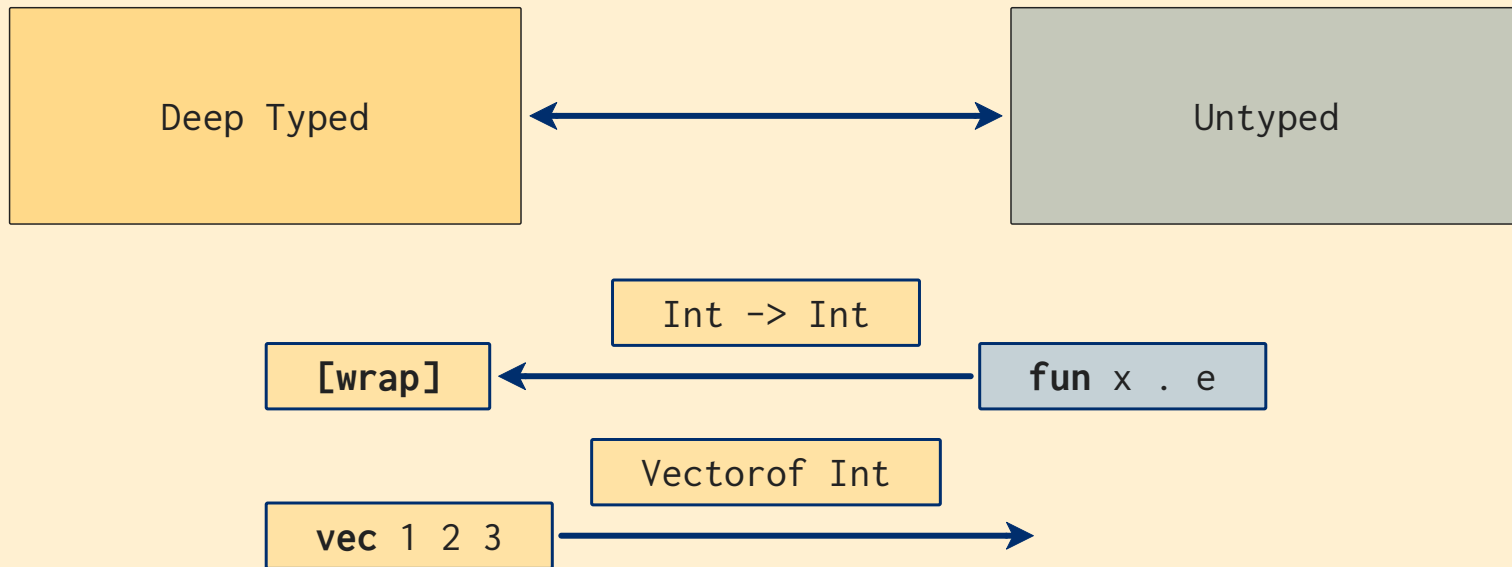
Q. How does **Natural** enforce **Deep** types?

A. Use **wrappers** to guard boundaries



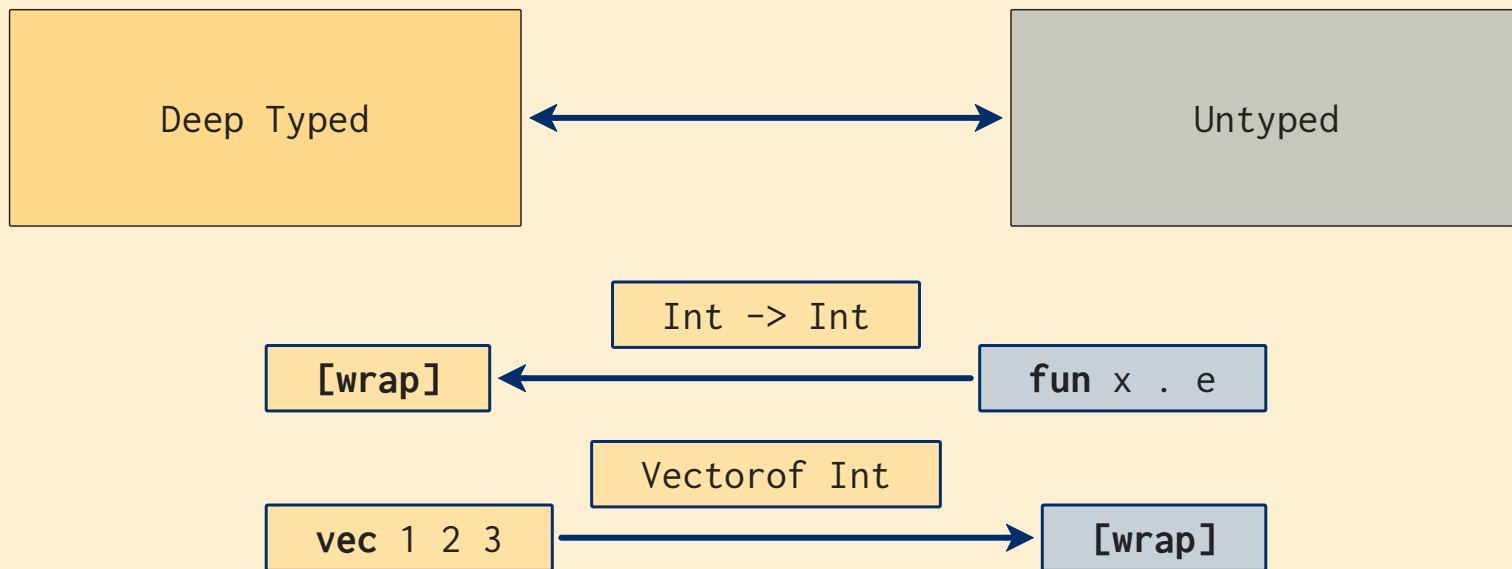
Q. How does **Natural** enforce **Deep** types?

A. Use **wrappers** to guard boundaries



Q. How does **Natural** enforce **Deep** types?

A. Use **wrappers** to guard boundaries



Q. How does **Transient** enforce **Shallow** types?



Q. How does **Transient** enforce **Shallow** types?

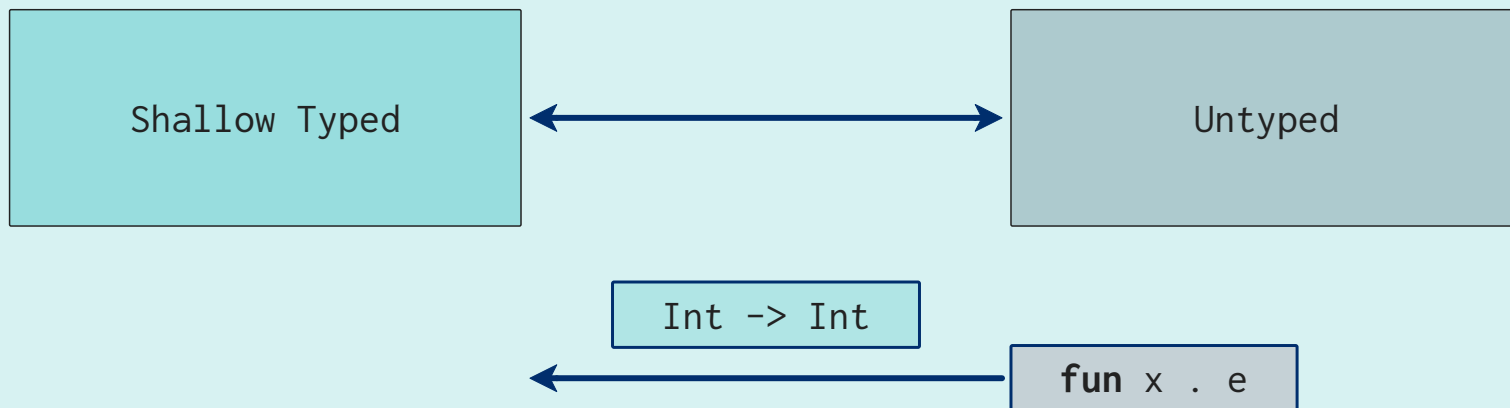
A. With **no wrappers** but many tiny **shape checks**





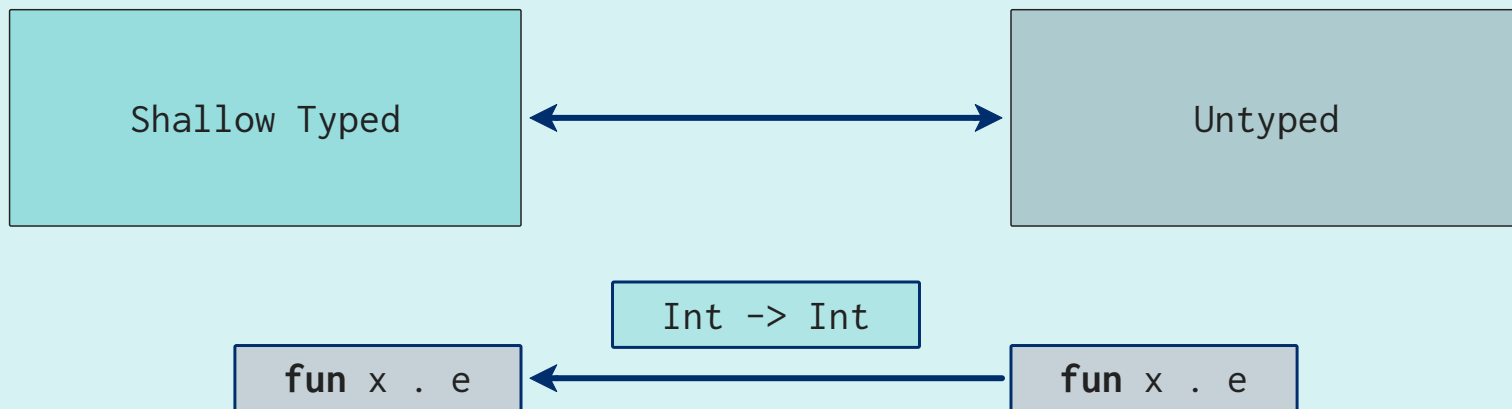
Q. How does **Transient** enforce **Shallow** types?

A. With **no wrappers** but many tiny **shape checks**



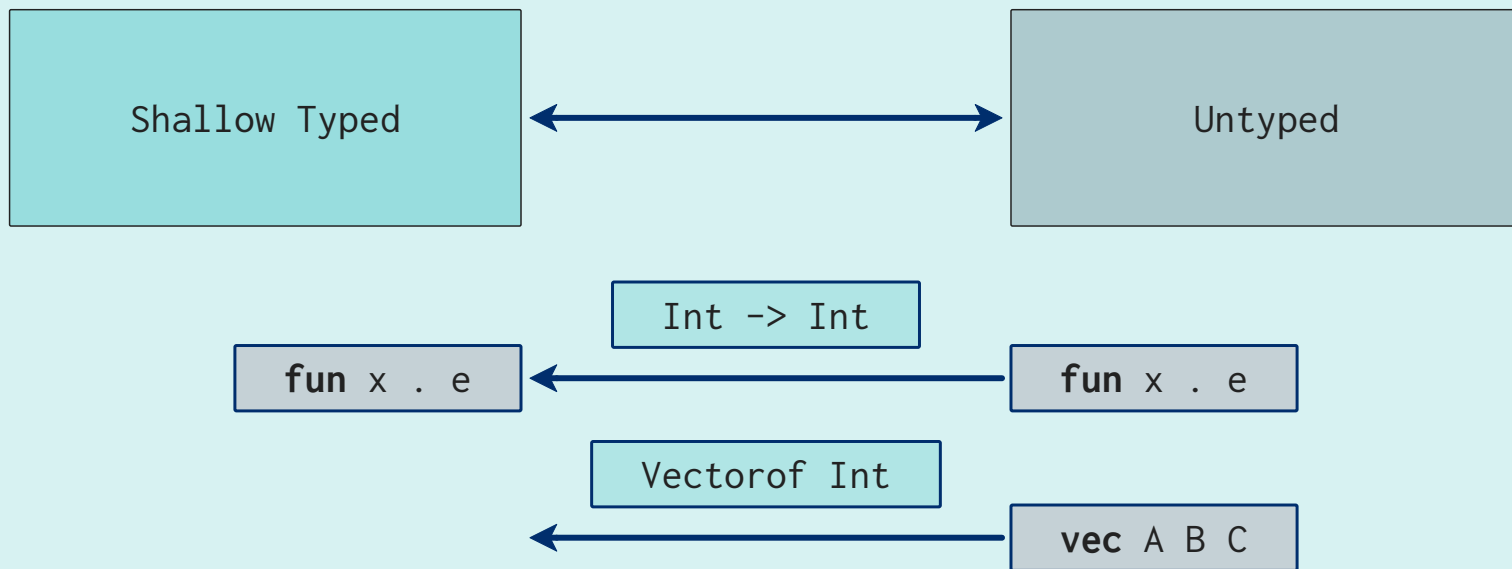
Q. How does **Transient** enforce **Shallow** types?

A. With **no wrappers** but many tiny **shape checks**



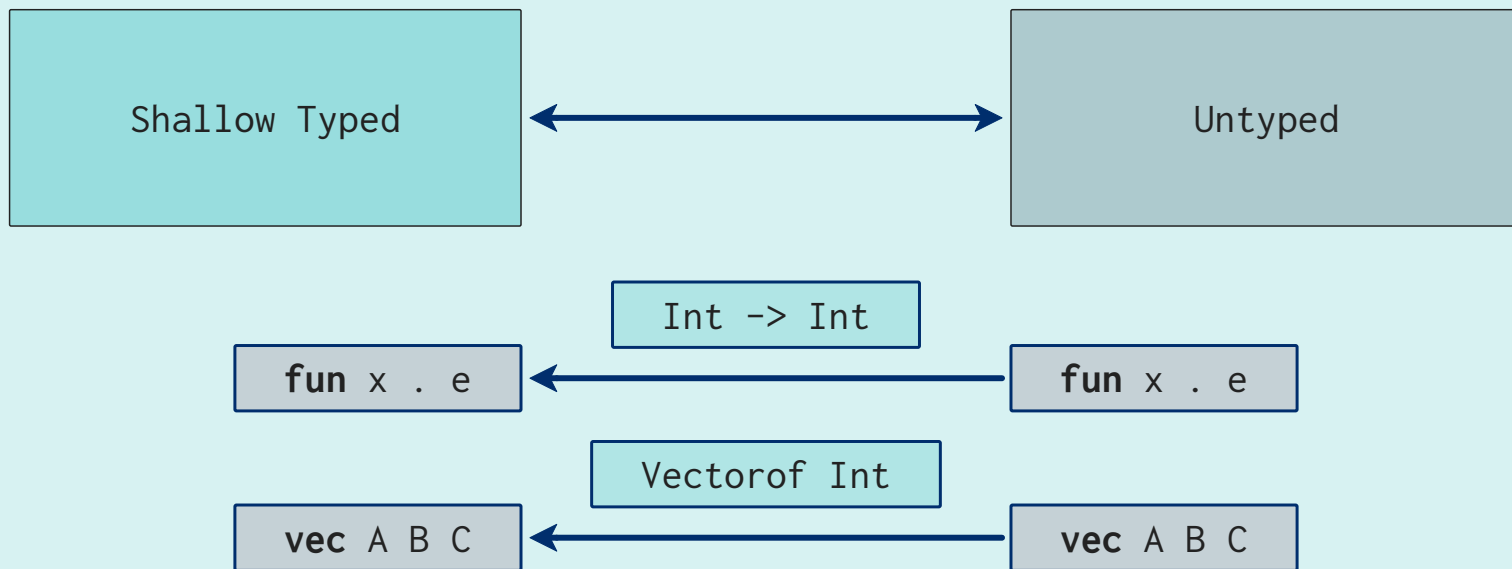
Q. How does **Transient** enforce **Shallow** types?

A. With **no wrappers** but many tiny **shape checks**



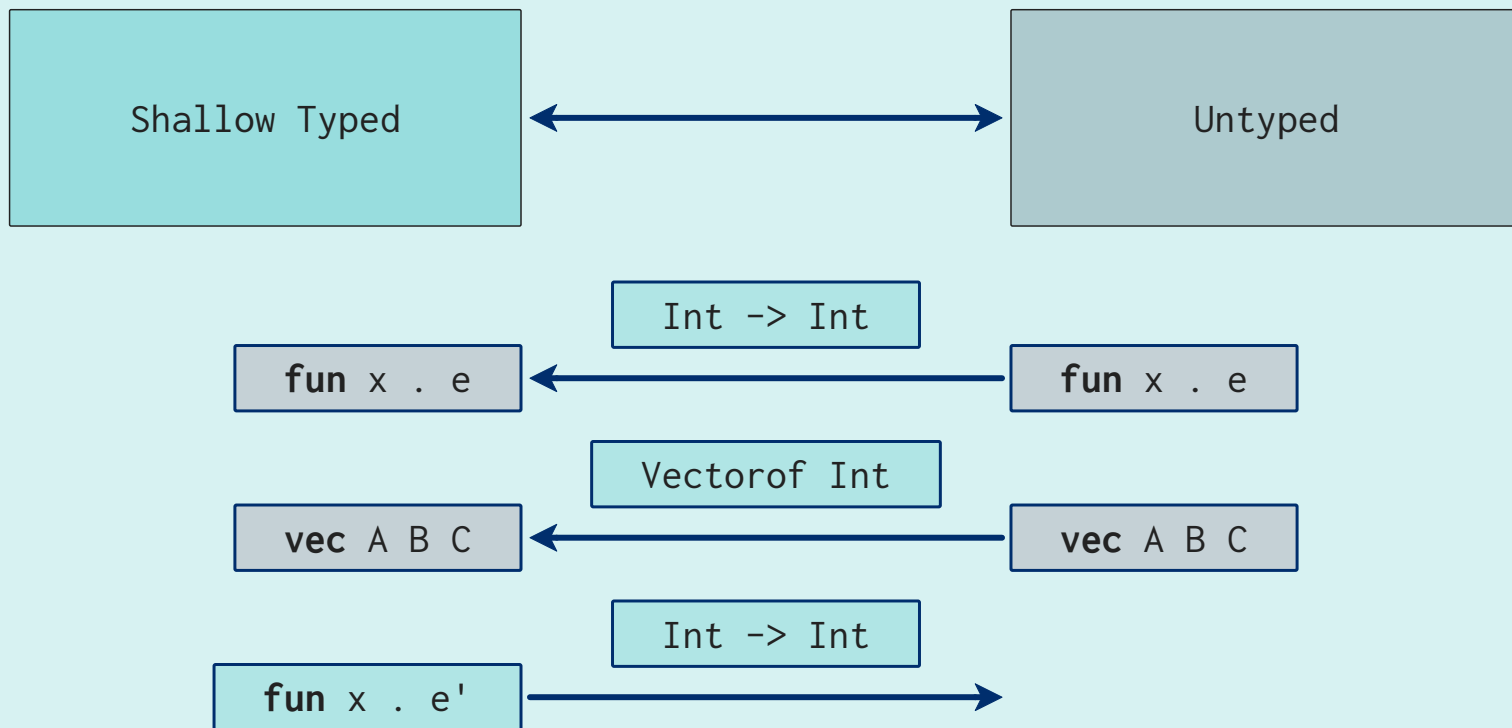
Q. How does **Transient** enforce **Shallow** types?

A. With **no wrappers** but many tiny **shape checks**



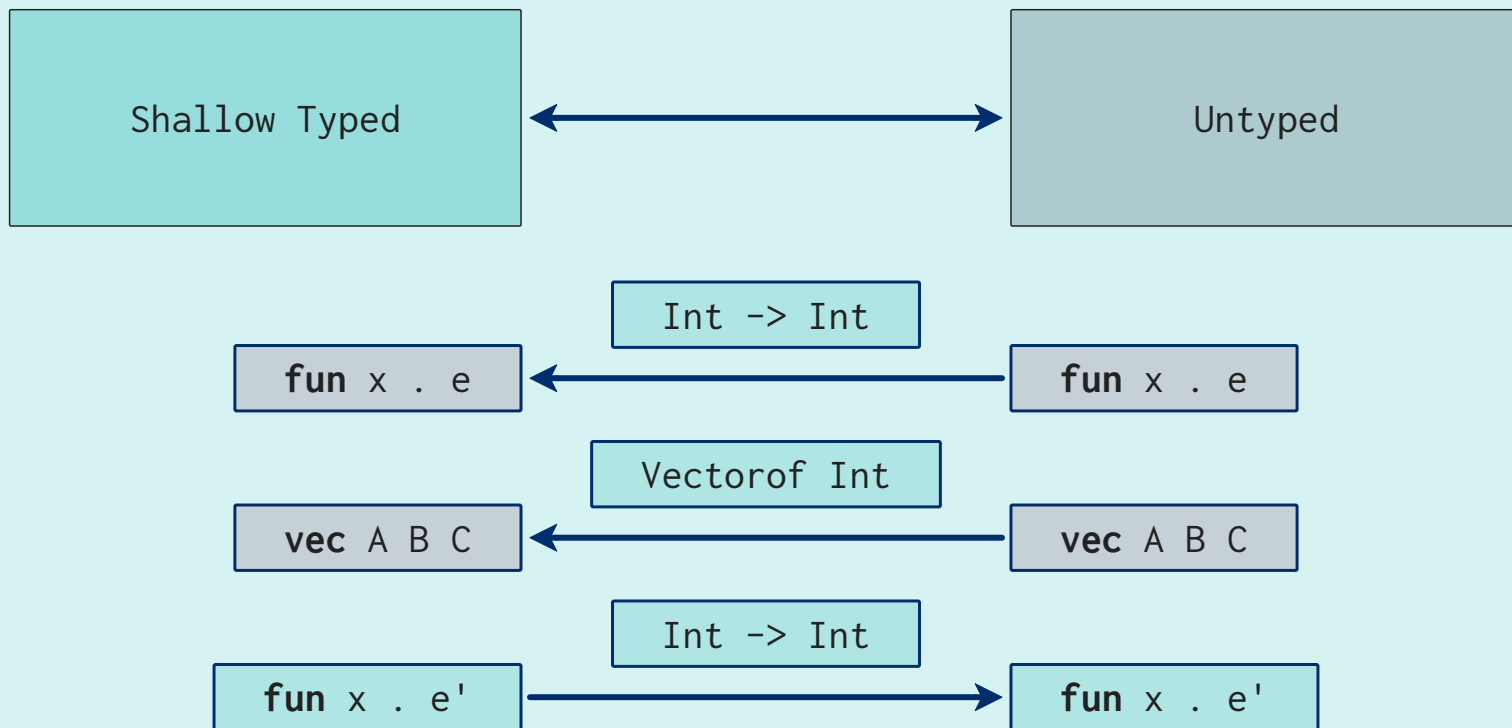
Q. How does **Transient** enforce **Shallow** types?

A. With **no wrappers** but many tiny **shape checks**



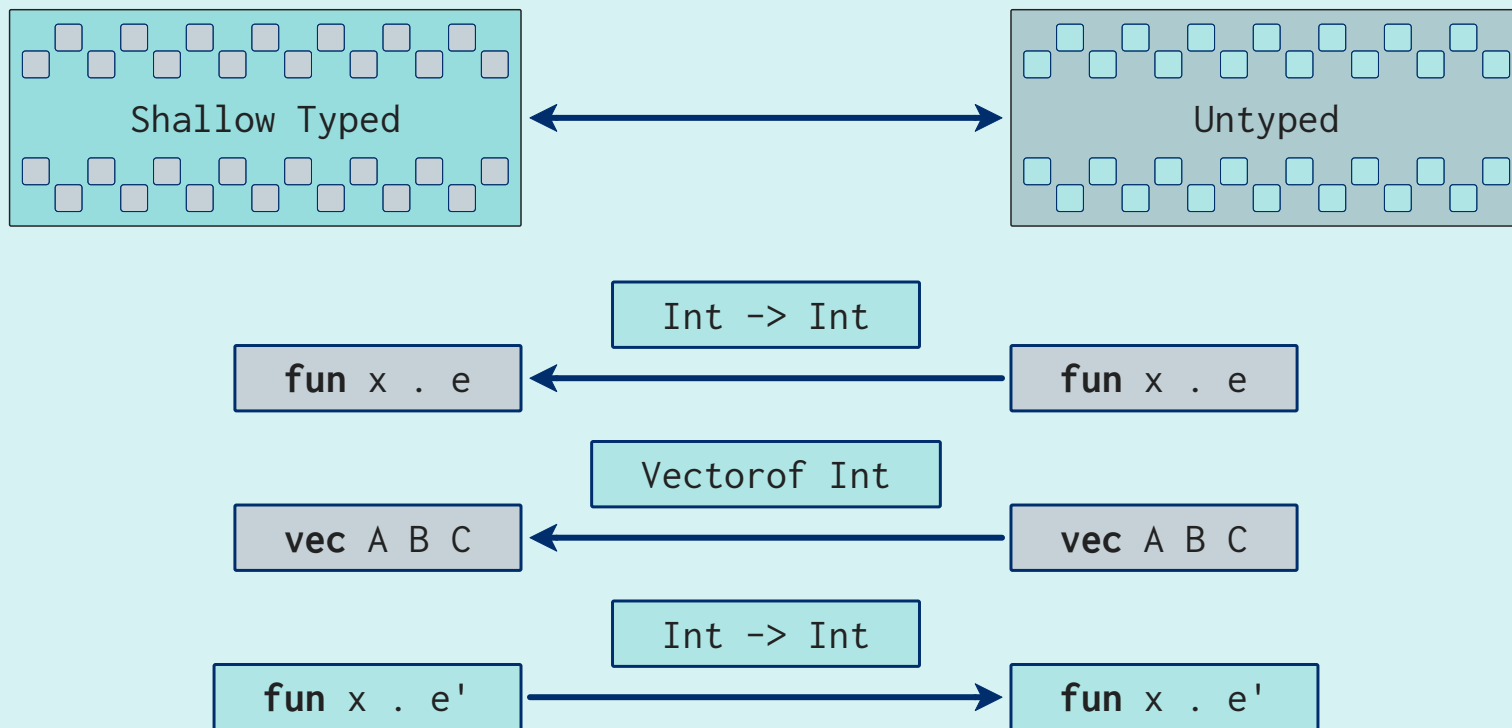
Q. How does **Transient** enforce **Shallow** types?

A. With **no wrappers** but many tiny **shape checks**



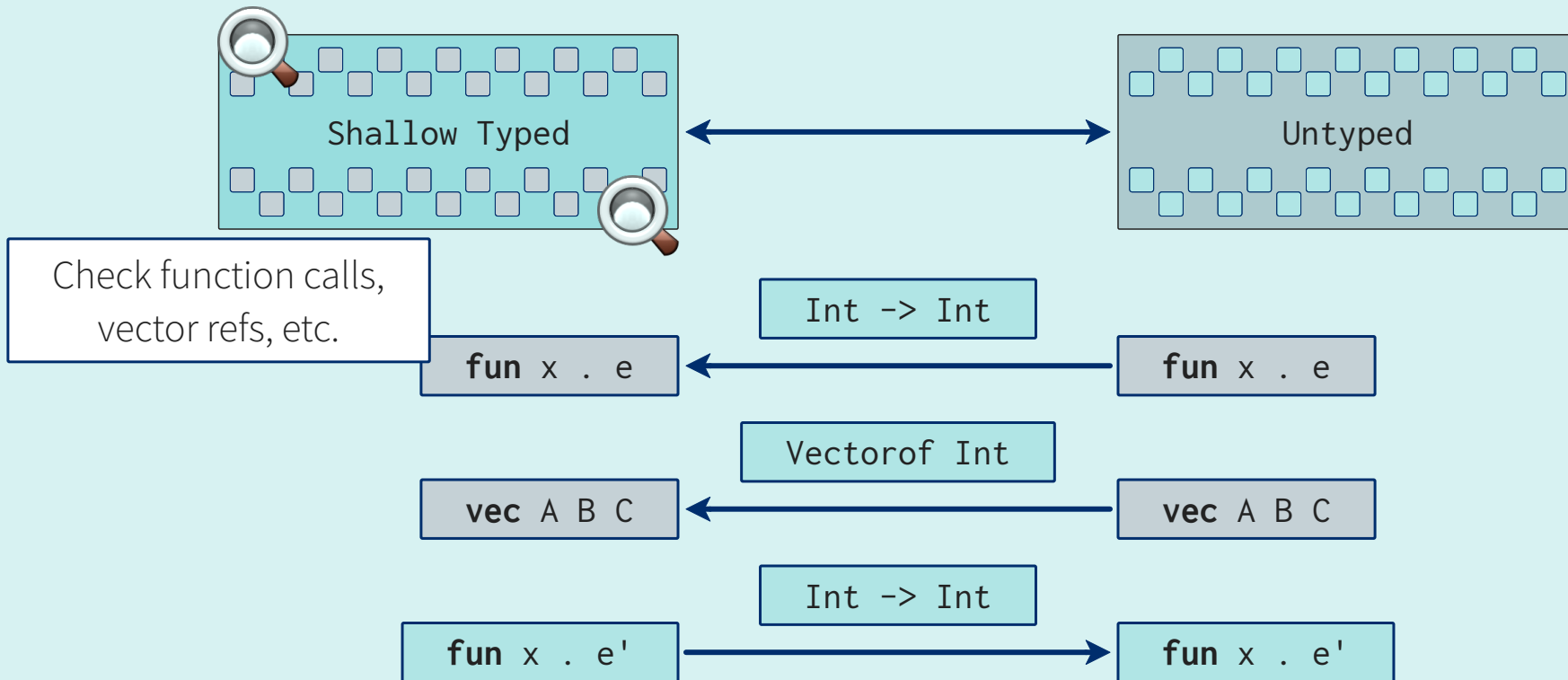
Q. How does **Transient** enforce **Shallow** types?

A. With **no wrappers** but many tiny **shape checks**

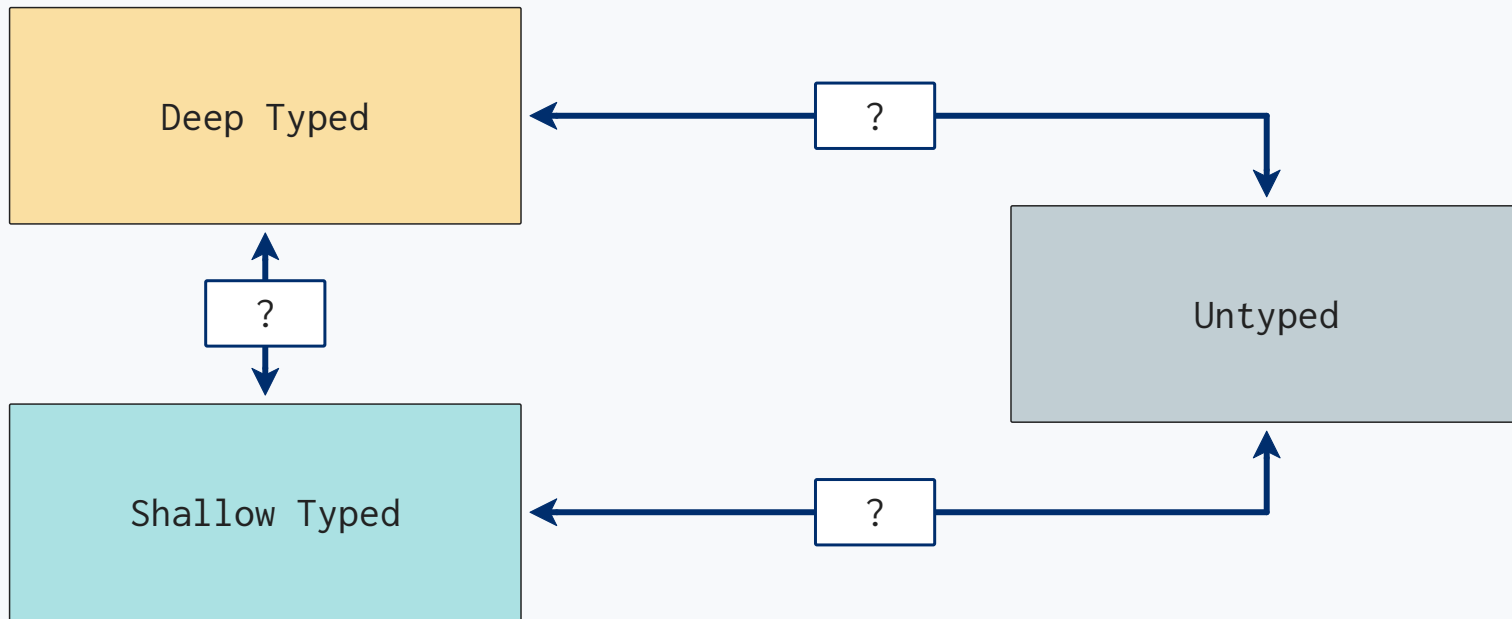


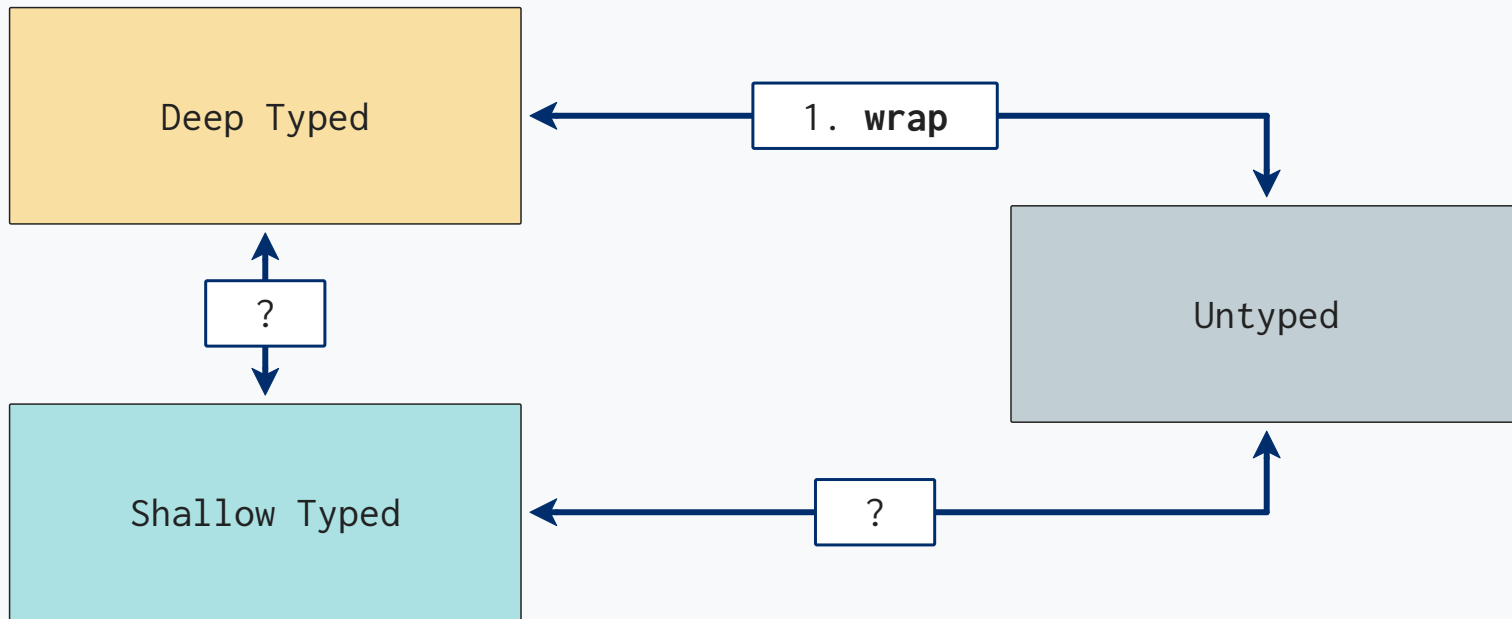
Q. How does **Transient** enforce **Shallow** types?

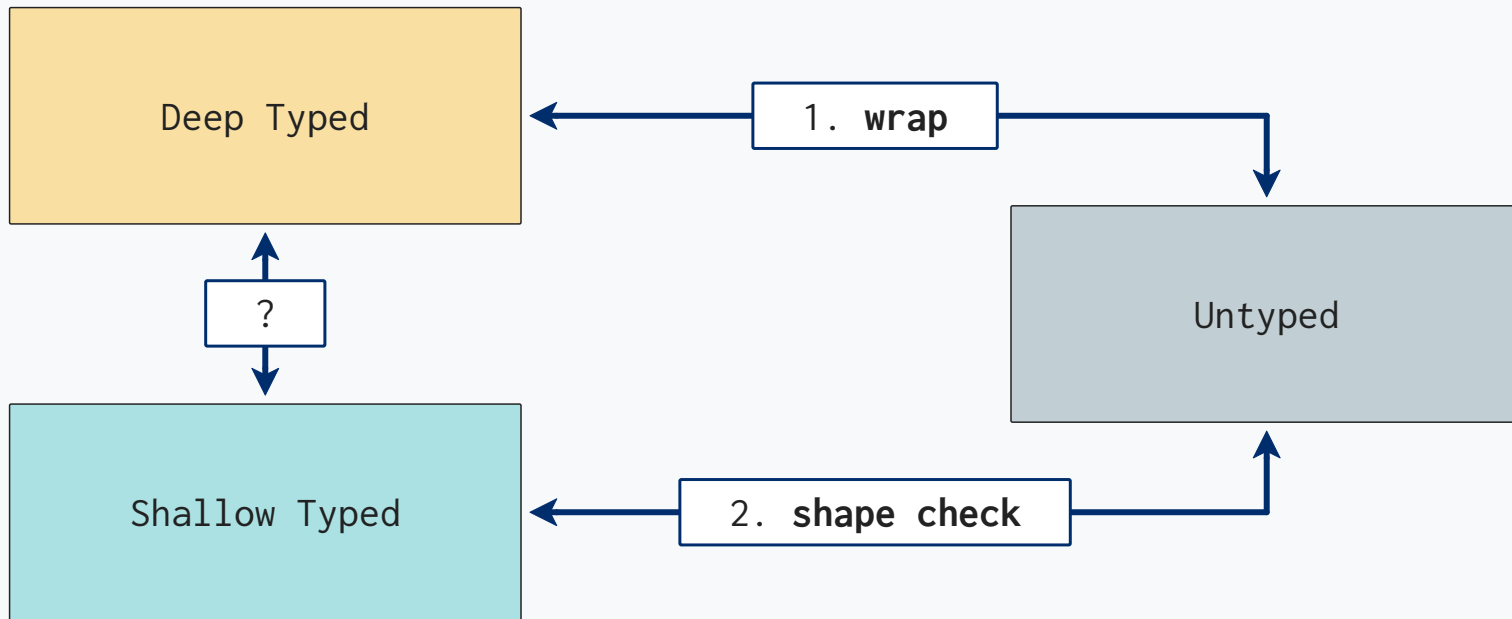
A. With **no wrappers** but many tiny **shape checks**

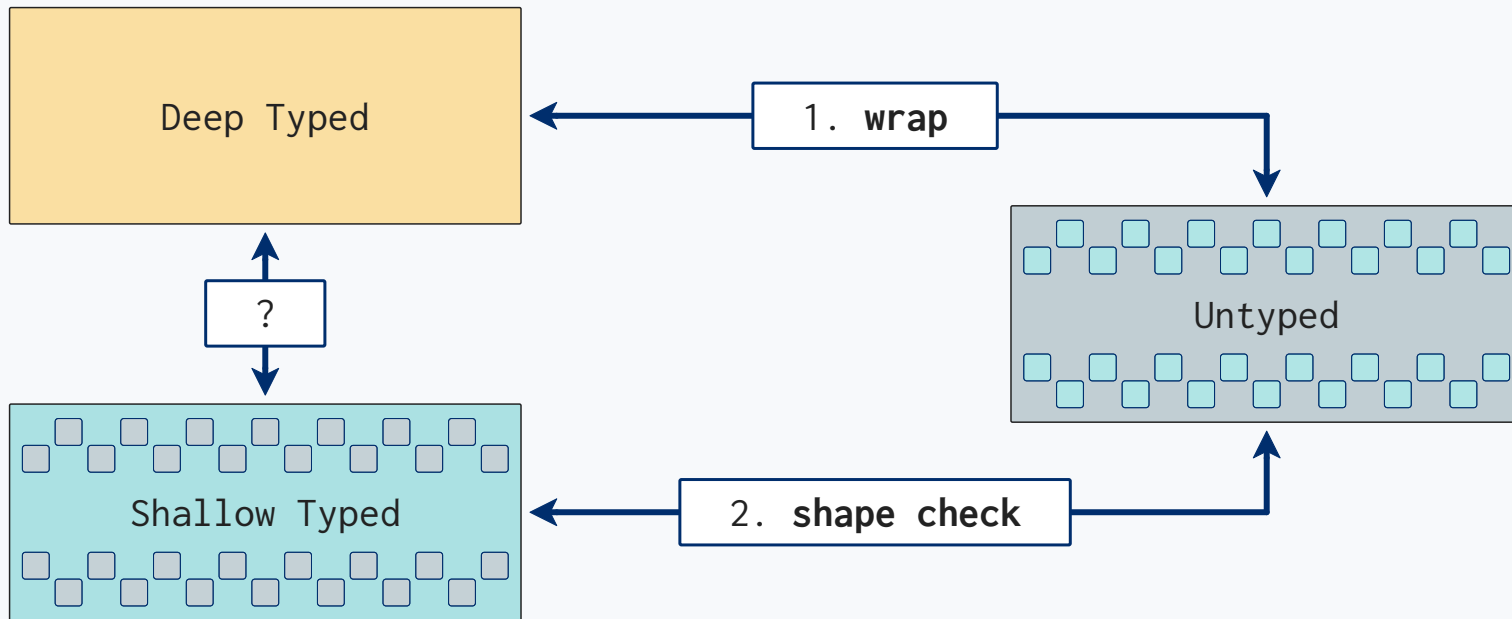


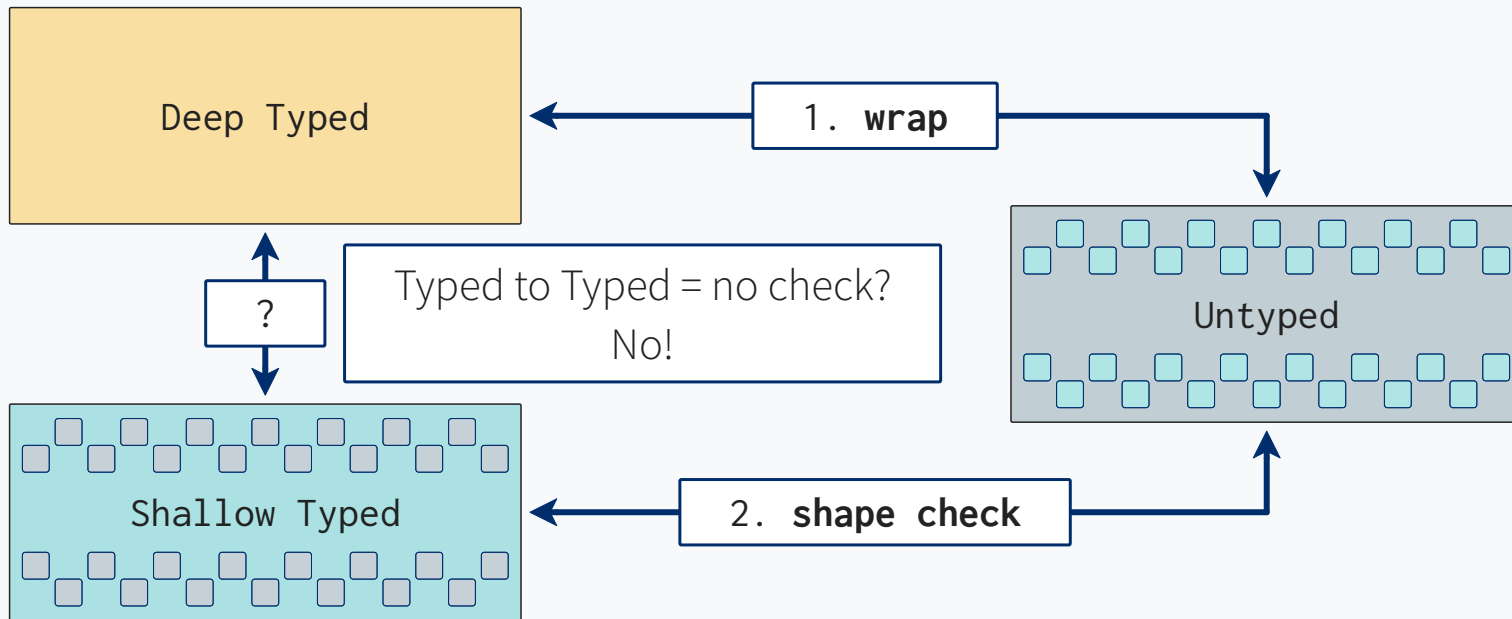












## What If: No Checks Between Deep and Shallow

Example 1:

Deep code cannot trust Shallow types because they are lazily enforced

Shallow makes a function,

```
def f0(n : Int):  
  n + 2
```

## What If: No Checks Between Deep and Shallow

Example 1:

Deep code cannot trust Shallow types because they are lazily enforced

Shallow makes a function,

```
def f0(n : Int):  
  n + 2
```

sends it to untyped code ...

```
def f1 = f0
```

## What If: No Checks Between Deep and Shallow

Example 1:

Deep code cannot trust Shallow types because they are lazily enforced

Shallow makes a function,

```
def f0(n : Int):  
  n + 2
```

sends it to untyped code ...

```
def f1 = f0
```

and back, with a new type.

```
f2 : Str -> Str  
def f2 = f1
```



## What If: No Checks Between Deep and Shallow

Example 1:

Deep code cannot trust Shallow types because they are lazily enforced

Shallow makes a function,

```
def f0(n : Int):  
  n + 2
```

sends it to untyped code ...

```
def f1 = f0
```

and back, with a new type.

```
f2 : Str -> Str  
def f2 = f1
```

**Types say** f2 : Str -> Str  
**Checks say** f2 is a function



## What If: No Checks Between Deep and Shallow

Example 1:

Deep code cannot trust Shallow types because they are lazily enforced

**Types say** `f2 : Str -> Str`  
**Checks say** `f2` is a function

Shallow makes a function,

```
def f0(n : Int):  
  n + 2
```

sends it to untyped code ...

```
def f1 = f0
```

and back, with a new type.

```
f2 : Str -> Str  
def f2 = f1
```

Deep gets a 'bad' function

```
f3 : Str -> Str  
def f3 = f2
```

## What If: No Checks Between Deep and Shallow

Deep makes a function,

```
def g0(h : Int -> Int):  
  h(3)
```

Example 2:

Shallow can send a Deep value to  
Untyped code

## What If: No Checks Between Deep and Shallow

Deep makes a function,

```
def g0(h : Int -> Int):  
  h(3)
```

Example 2:

Shallow can send a Deep value to  
Untyped code

sends it to Shallow,

```
g1 : (Int -> Int) -> Int  
def g1 = g0
```

## What If: No Checks Between Deep and Shallow

Example 2:  
Shallow can send a Deep value to  
Untyped code

Deep makes a function,

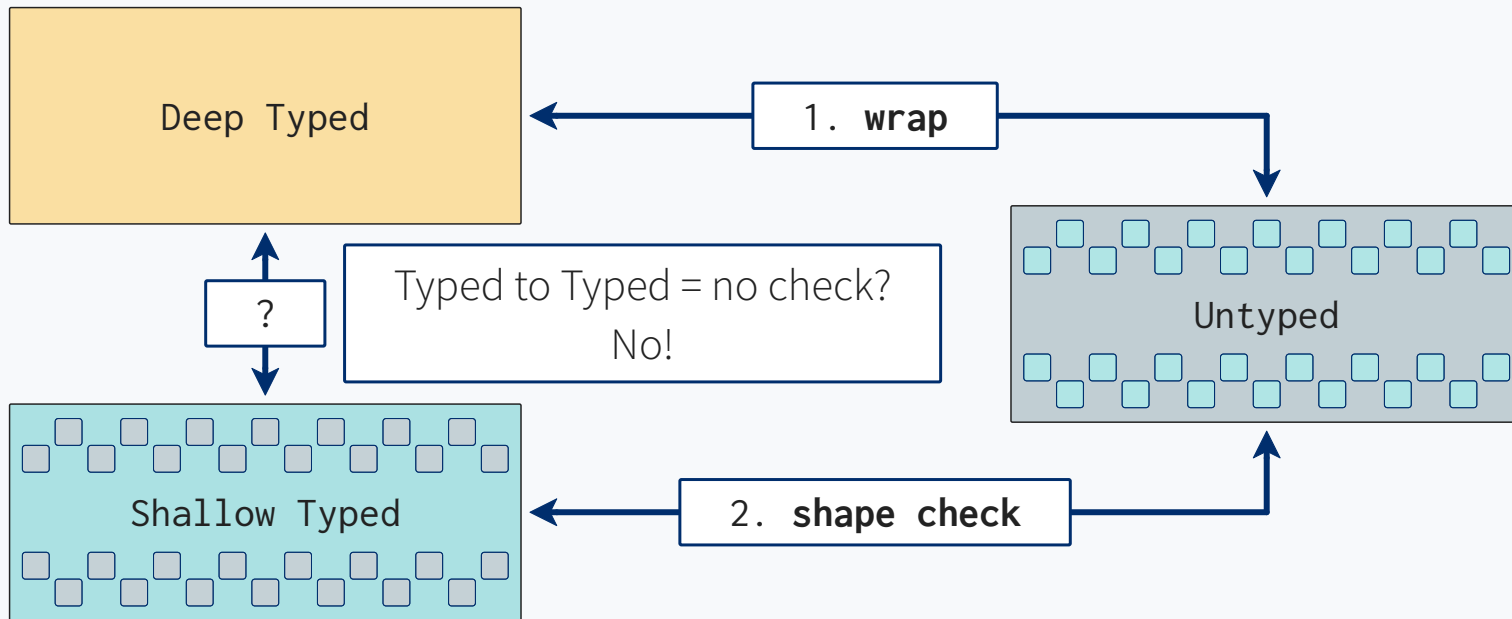
```
def g0(h : Int -> Int):  
  h(3)
```

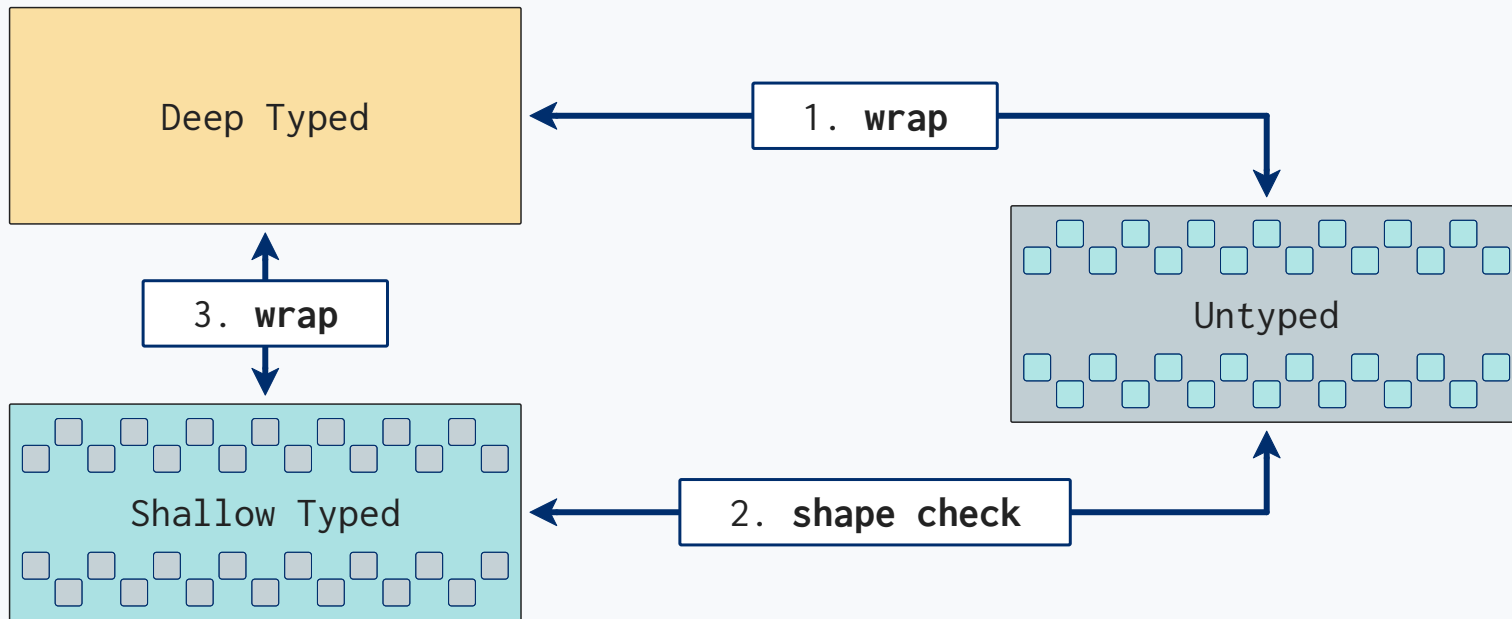
sends it to Shallow,

```
g1 : (Int -> Int) -> Int  
def g1 = g0
```

which sends it to untyped

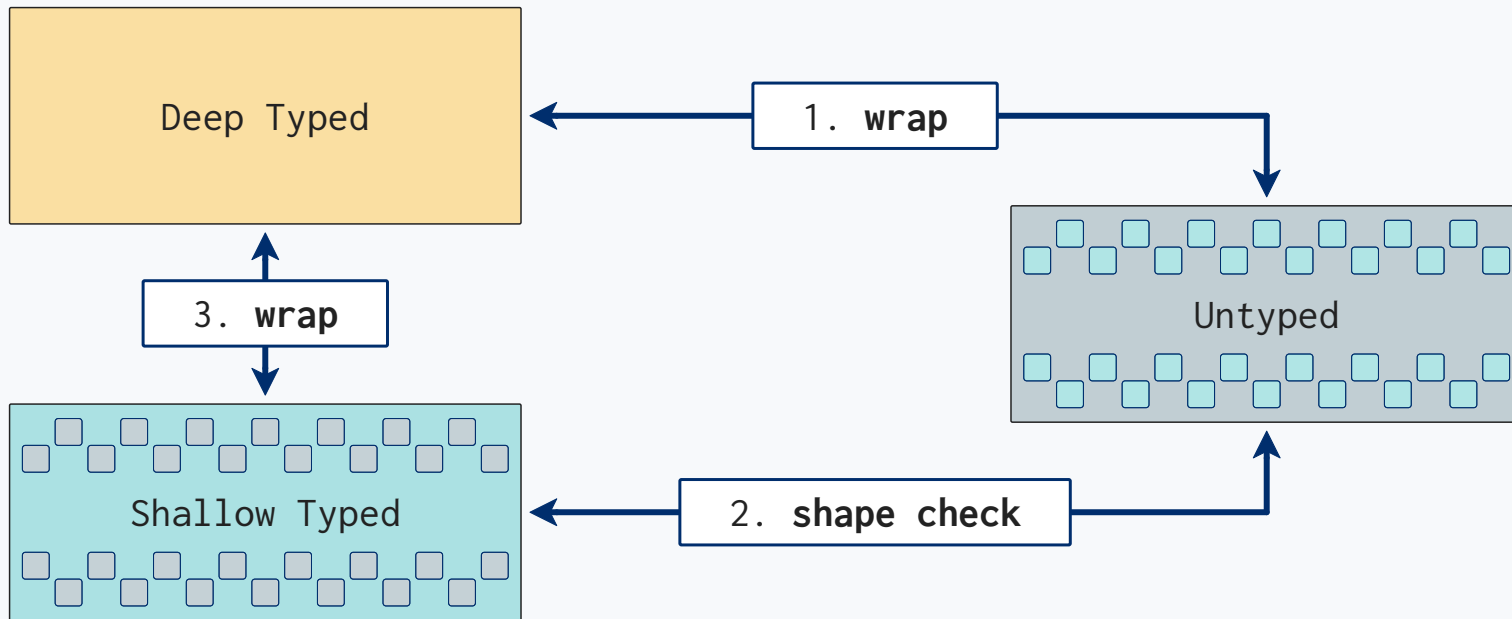
```
def g2 = g1  
g2("not a function")
```







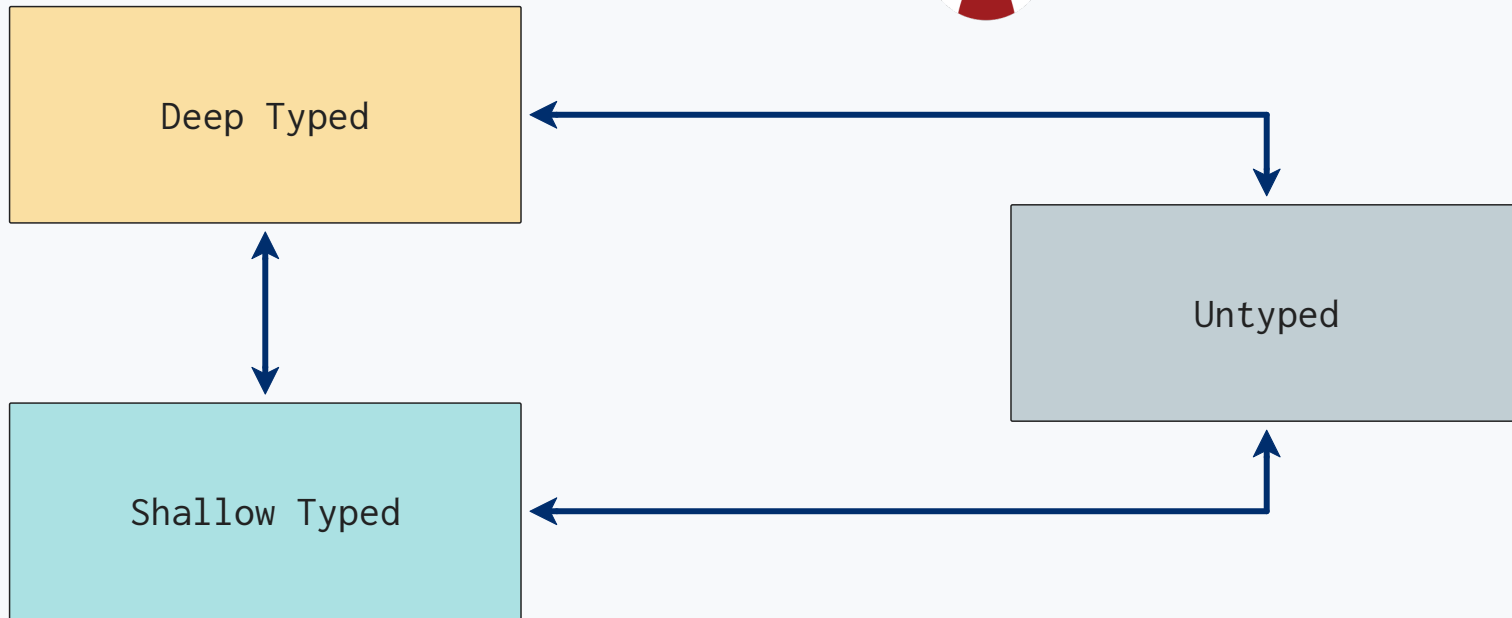
In paper: model, type soundness, complete monitoring





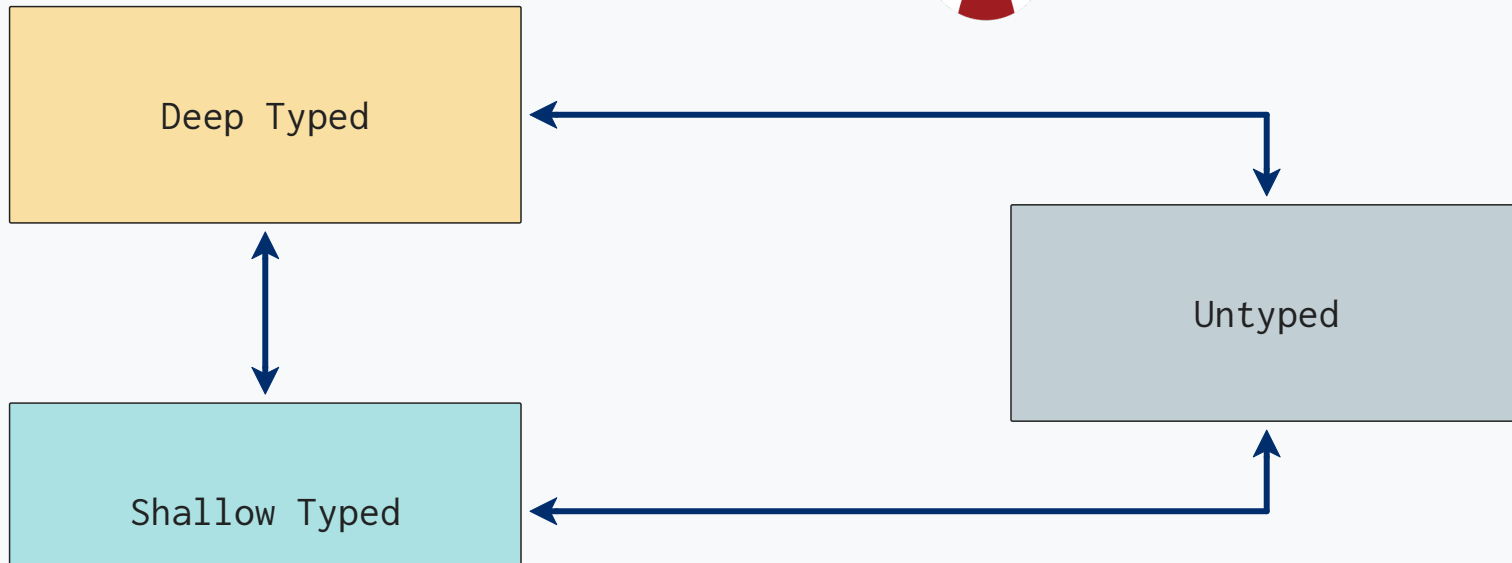
# Implementation

Typed Racket



# Implementation

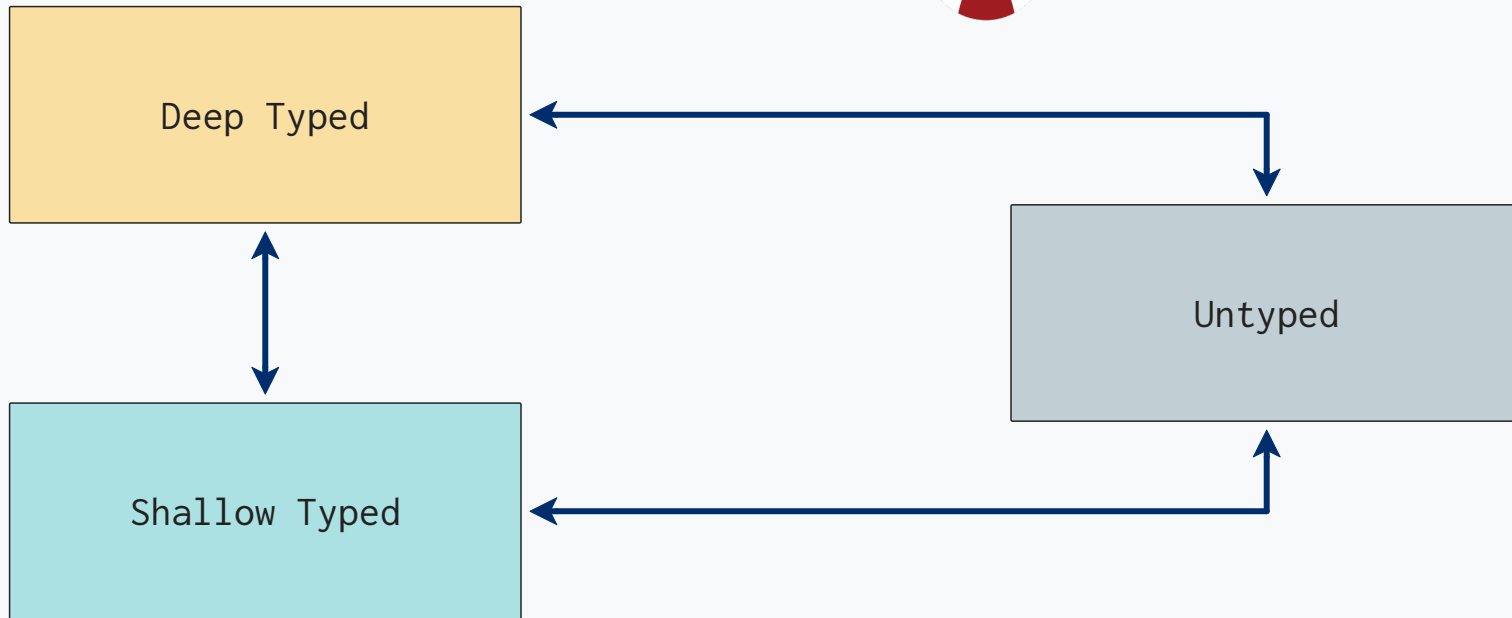
Typed Racket



**A Transient Semantics for Typed Racket**  
Programming'22

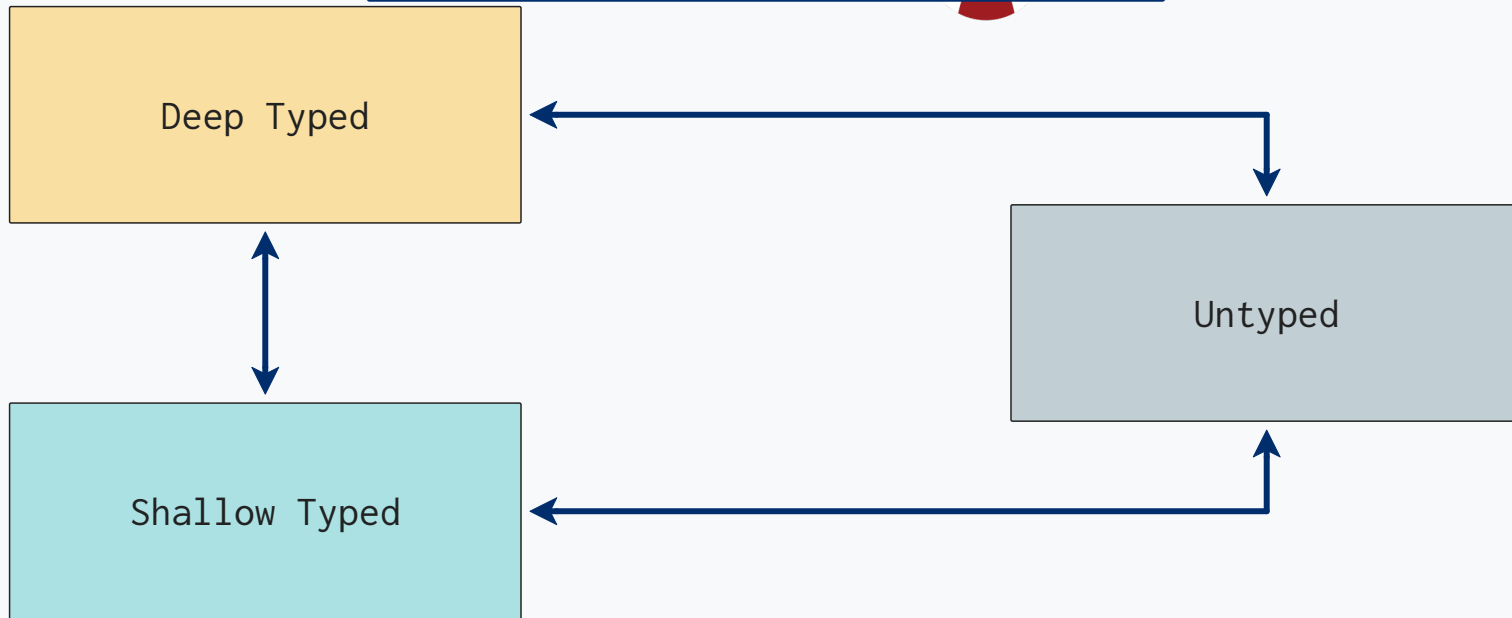
# Implementation

Typed Racket



# Implementation

In paper: general lessons (**no macros**)



# Evaluation

**Guarantees** vs. **Performance** vs. **Expressiveness**

# Evaluation

**Guarantees** vs. **Performance** vs. **Expressiveness**





# GTP Benchmarks

## 21 programs

...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](https://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.

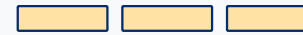
0. admin.rkt	3. board.rkt	6. state.rkt	9. .../base/untyped.rkt
1. auxiliaries.rkt	4. main.rkt	7. strategy.rkt	
2. basics.rkt	5. player.rkt	8. tree.rkt	

[docs.racket-lang.org/gtp-benchmarks](https://docs.racket-lang.org/gtp-benchmarks)



## GTP Benchmarks 21 programs

Ex: One program with 3 components



...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](https://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	3. board.rkt	6. state.rkt	9. .../base/untyped.rkt
1. auxiliaries.rkt	4. main.rkt	7. strategy.rkt	
2. basics.rkt	5. player.rkt	8. tree.rkt	

[docs.racket-lang.org/gtp-benchmarks](https://docs.racket-lang.org/gtp-benchmarks)





# GTP Benchmarks

21 programs

...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](https://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.

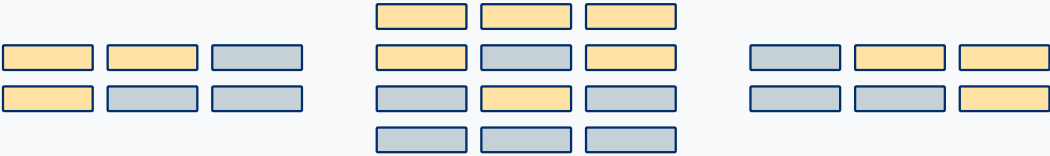
0. admin.rkt    3. board.rkt    6. state.rkt    9. .../base/untyped.rkt  
1. auxiliaries.rkt    4. main.rkt    7. strategy.rkt  
2. basics.rkt    5. player.rkt    8. tree.rkt

[docs.racket-lang.org/gtp-benchmarks](https://docs.racket-lang.org/gtp-benchmarks)

Ex: One program with 3 components



8 Typed / Untyped points ( $2^N$ )





# GTP Benchmarks

21 programs

...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](https://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.

```

graph TD
    9[9] --- 0[0]
    9 --- 1[1]
    9 --- 2[2]
    9 --- 3[3]
    9 --- 4[4]
    9 --- 5[5]
    9 --- 6[6]
    9 --- 7[7]
    9 --- 8[8]
    0 --- 1
    0 --- 2
    0 --- 3
    0 --- 4
    0 --- 5
    0 --- 6
    0 --- 7
    0 --- 8
    1 --- 2
    1 --- 3
    1 --- 4
    1 --- 5
    1 --- 6
    1 --- 7
    1 --- 8
    2 --- 3
    2 --- 4
    2 --- 5
    2 --- 6
    2 --- 7
    2 --- 8
    3 --- 4
    3 --- 5
    3 --- 6
    3 --- 7
    3 --- 8
    4 --- 5
    4 --- 6
    4 --- 7
    4 --- 8
    5 --- 6
    5 --- 7
    5 --- 8
    6 --- 7
    6 --- 8
    7 --- 8
    
```

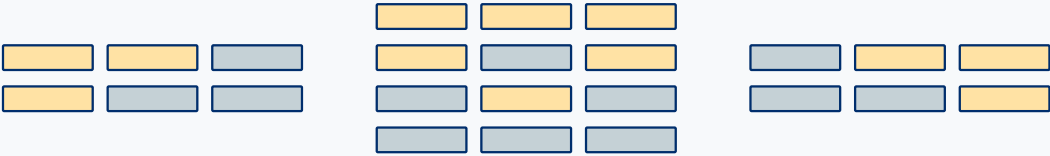
0. admin.rkt	3. board.rkt	6. state.rkt	9. .../base/untyped.rkt
1. auxiliaries.rkt	4. main.rkt	7. strategy.rkt	
2. basics.rkt	5. player.rkt	8. tree.rkt	

[docs.racket-lang.org/gtp-benchmarks](https://docs.racket-lang.org/gtp-benchmarks)

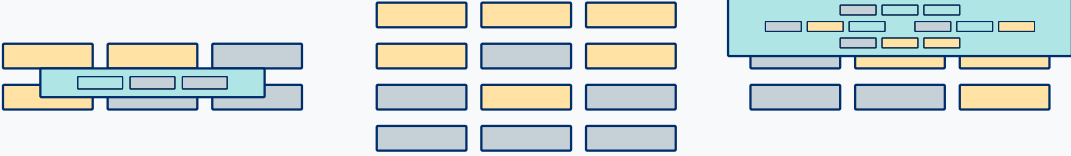
Ex: One program with 3 components



8 Typed / Untyped points (2^N)



27 Deep / Shallow / Untyped points (3^N)

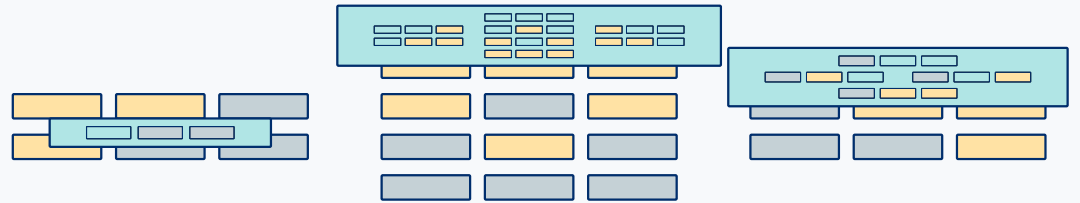


## Better Performance

Q. How many points run fastest with a Deep + Shallow mix?

## Better Performance

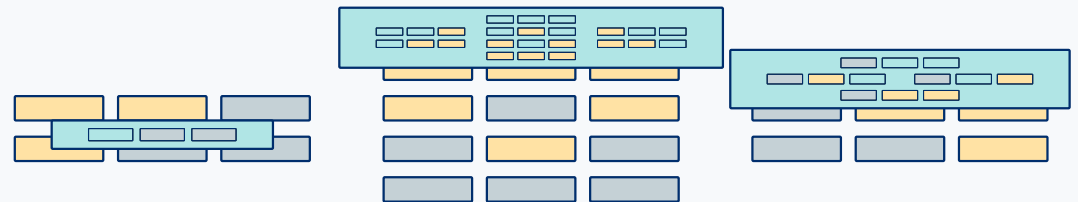
Q. How many points run fastest with a Deep + Shallow mix?



## Better Performance

Q. How many points run fastest with a Deep + Shallow mix?

forth	12%	zordoz	<b>47%</b>
fsm	38%	lnm	<b>66%</b>
fsmoo	31%	suffixtree	<b>48%</b>
mbta	19%	kcfa	<b>55%</b>
morsecode	25%	snake	<b>46%</b>
zombie	6%	take5	36%
dungeon	31%	acquire	<b>64%</b>
jpeg	38%	tetris	<b>62%</b>



## Better Performance

Q. What is the worst-case overhead?

Deep

or

Shallow

## Better Performance

Q. What is the worst-case overhead?

Deep

or

Shallow

sieve	2.97x	suffixtree	5.8x
forth	5.43x	kcfa	1.24x
fsm	1.91x	snake	7.61x
fsmoo	4.25x	take5	2.97x
mbta	1.71x	acquire	1.42x
morsecode	1.3x	tetris	5.44x
zombie	<b>31x</b>	synth	4.2x
dungeon	3.16x	gregor	1.51x
jpeg	1.56x	quadT	7.23x
zordoz	2.58x	quadU	7.45x
lnm	1.17x		

## Better Performance

Q. What is the worst-case overhead?

Deep

or

Shallow

sieve	2.97x	16x	4.36x	suffixtree	5.8x	31x	5.8x
forth	5.43x	5800x	5.51x	kcfa	1.24x	4.33x	1.24x
fsm	1.91x	2.24x	2.38x	snake	7.61x	12x	7.67x
fsmoo	4.25x	420x	4.28x	take5	2.97x	44x	2.99x
mbta	1.71x	1.91x	1.74x	acquire	1.42x	4.22x	1.42x
morsecode	1.3x	1.57x	2.77x	tetris	5.44x	13x	9.93x
zombie	31x	46x	31x	synth	4.2x	47x	4.2x
dungeon	3.16x	1500x	4.97x	gregor	1.51x	1.72x	1.59x
jpeg	1.56x	23x	1.66x	quadT	7.23x	26x	7.39x
zordoz	2.58x	2.63x	2.75x	quadU	7.45x	55x	7.57x
lnm	1.17x	1.23x	1.21x				



## Better Performance

Q. What is the worst-case overhead?

Deep

or

Shallow

sieve	2.97x	16x	4.36x	suffixtree	5.8x	31x	5.8x
forth	5.43x	5800x	5.51x	kcfa	1.24x	4.33x	1.24x
fsm	1.91x	2.24x	2.38x	snake	7.61x	12x	7.67x
fsmoo	4.25x	420x	4.28x	take5	2.97x	44x	2.99x
mbta	1.71x	1.91x	1.74x	acquire	1.42x	4.22x	1.42x
morsecode	1.3x	1.57x	2.77x	tetris	5.44x	13x	9.93x
zombie	31x	46x	31x	synth	4.2x	47x	4.2x
dungeon	3.16x	1500x	4.97x	gregor	1.51x	1.72x	1.59x
jpeg	1.56x	23x	1.66x	quadT	7.23x	26x	7.39x
zordoz	2.58x	2.63x	2.75x	quadU	7.45x	55x	7.57x
lnm	1.17x	1.23x	1.21x				

H.O. values and many elim. forms



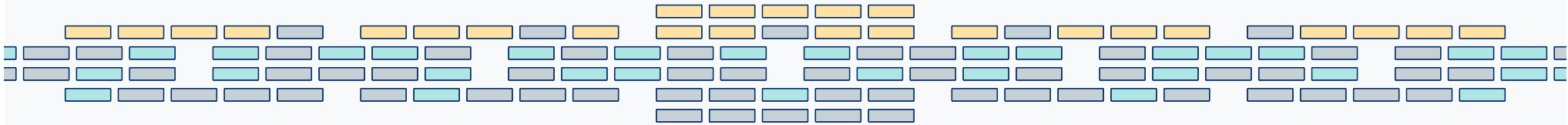
# Better Performance

# Better Performance

Overall: switching between Deep and Shallow can avoid perf. bottlenecks

**Deep** near the top,  
to maximize the benefits of types

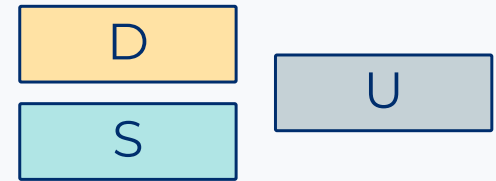
**Shallow** in the middle,  
to minimize the cost of boundaries



# Conclusion

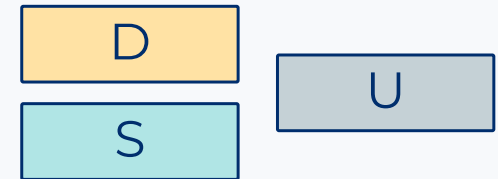
**Context:**

Different GT strategies exist  
(for good reason!)



**Context:**

Different GT strategies exist  
(for good reason!)



**Inquiry:**

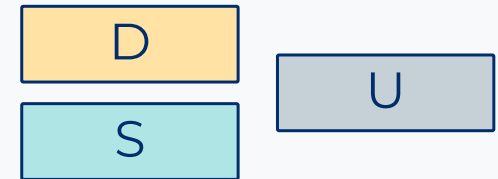
Can two extreme strategies interoperate?

**Deep** types via **Natural** (wrappers)

**Shallow** types via **Transient** (no wrappers)

**Context:**

Different GT strategies exist  
(for good reason!)



**Inquiry:**

Can two extreme strategies interoperate?

**Deep** types via **Natural** (wrappers)

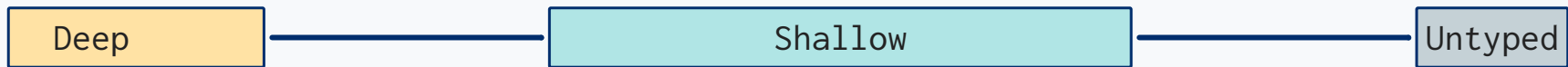
**Shallow** types via **Transient** (no wrappers)

**Contribution:**

Yes! In a way that:

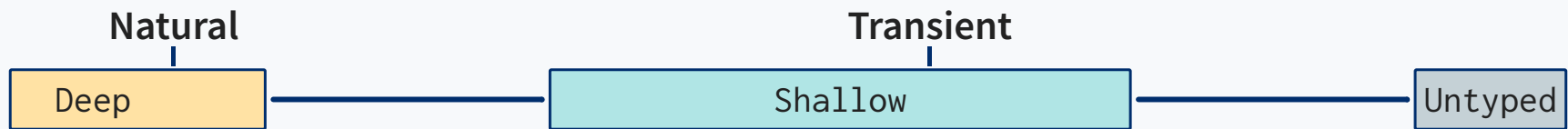
- preserves their formal **guarantees**
- leads to better overall **performance**
- lets TR **express** additional programs

## A New Dimension for Gradual Typing

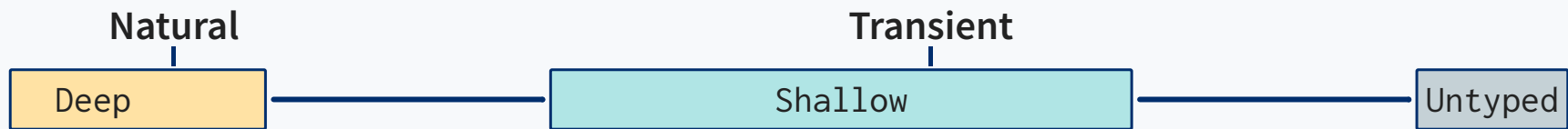




## A New Dimension for Gradual Typing

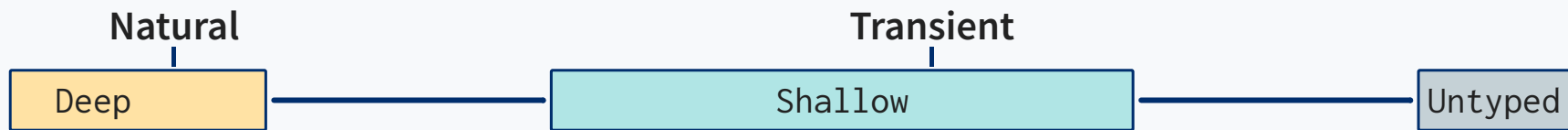


## A New Dimension for Gradual Typing



Q. More regions along the spectrum?

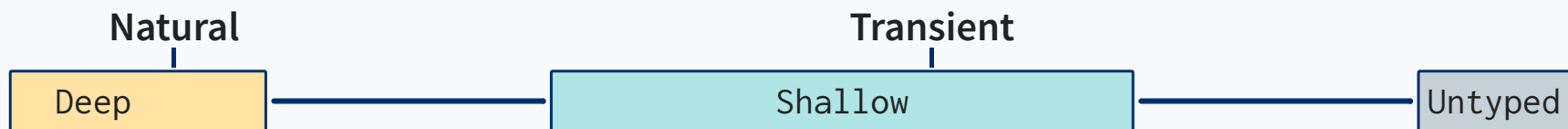
## A New Dimension for Gradual Typing



Q. More regions along the spectrum?

Q. Better cooperation b/w Deep and Shallow?

## A New Dimension for Gradual Typing

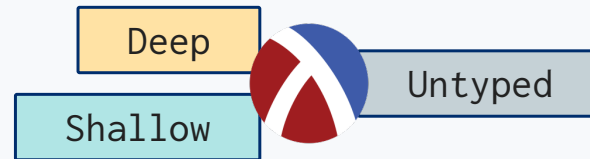


Q. More regions along the spectrum?

Q. Better cooperation b/w Deep and Shallow?

Q. Solve the  $N^2$  interop problem?

# The End



Coming soon to Racket  
<https://racket-lang.org>

Pull Request <https://github.com/racket/typed-racket/pull/948>

Research Repo <https://github.com/bennn/g-pldi-2022>

