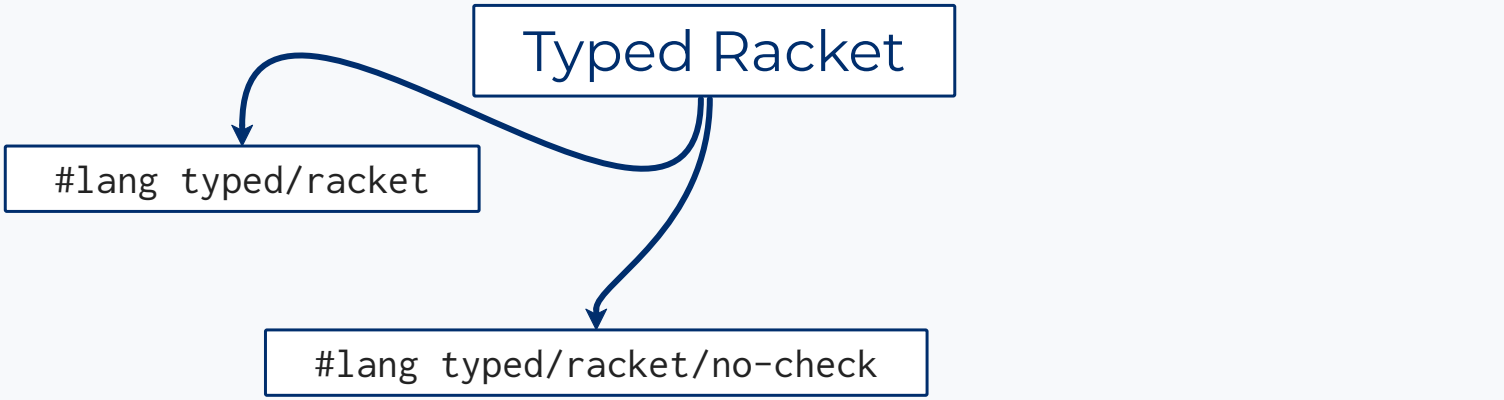
Shallow and **Optional** Types for Typed Racket

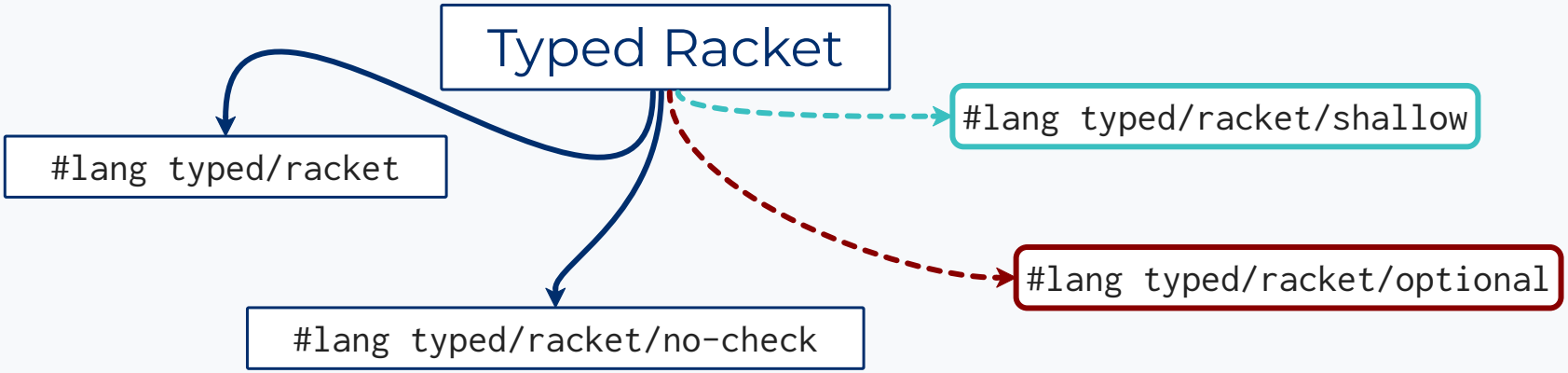
in Racket 8.7

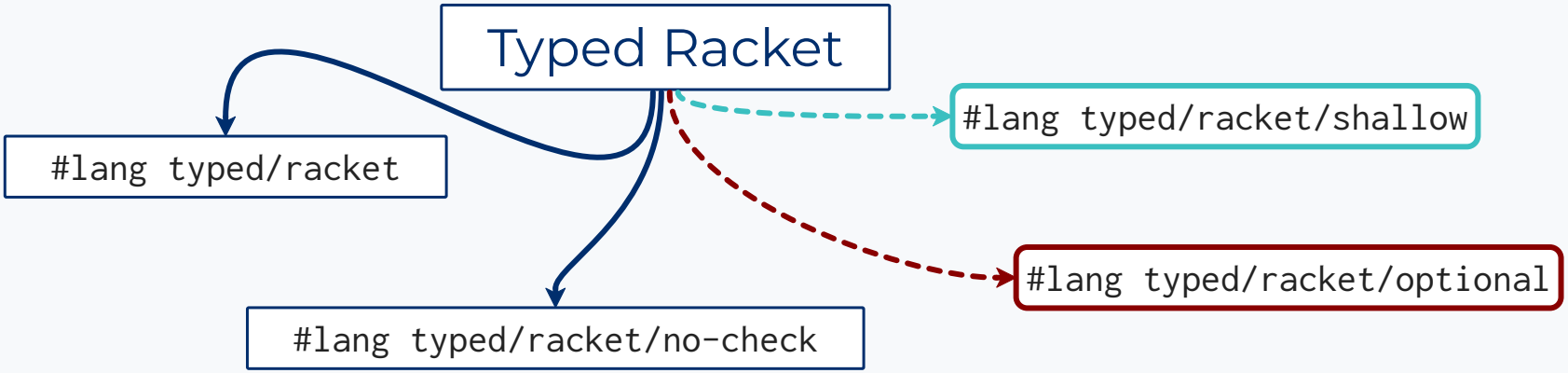
Ben Greenman

RacketCon 2022

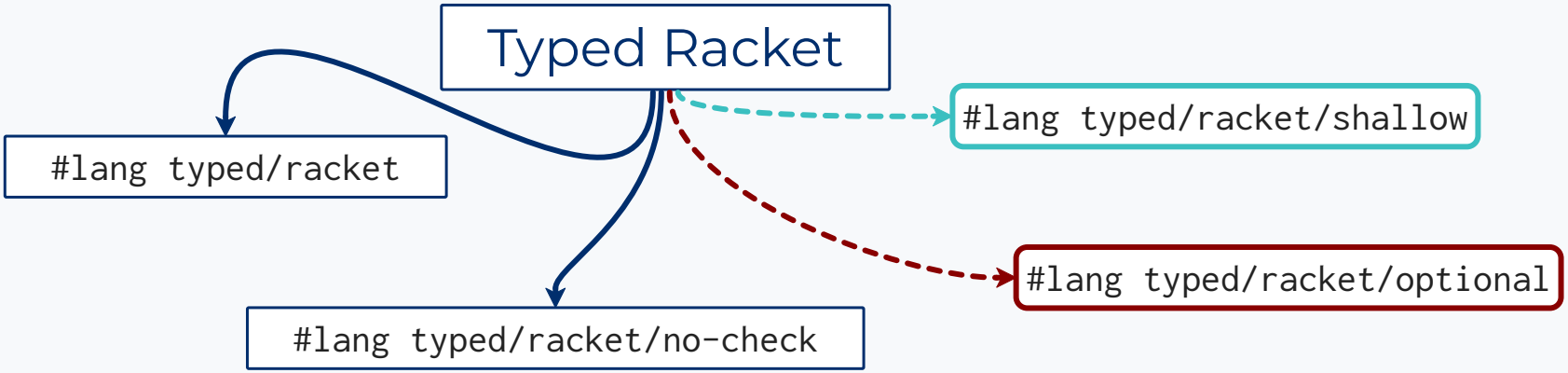
Typed Racket

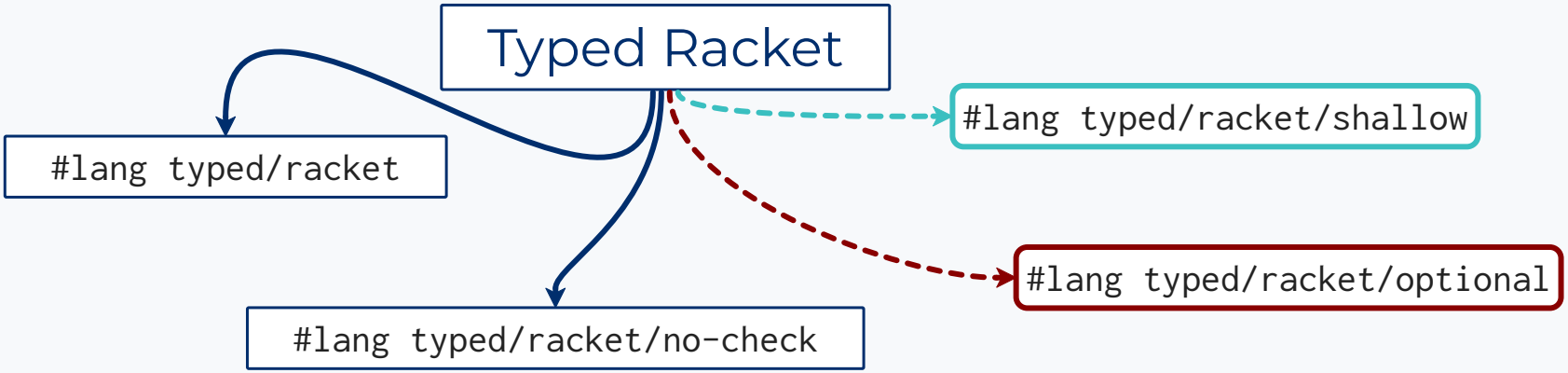

Typed Racket
#lang typed/racket
#lang typed/racket/no-check


Typed Racket
#lang typed/racket
#lang typed/racket/shallow
#lang typed/racket/optional
#lang typed/racket/no-check

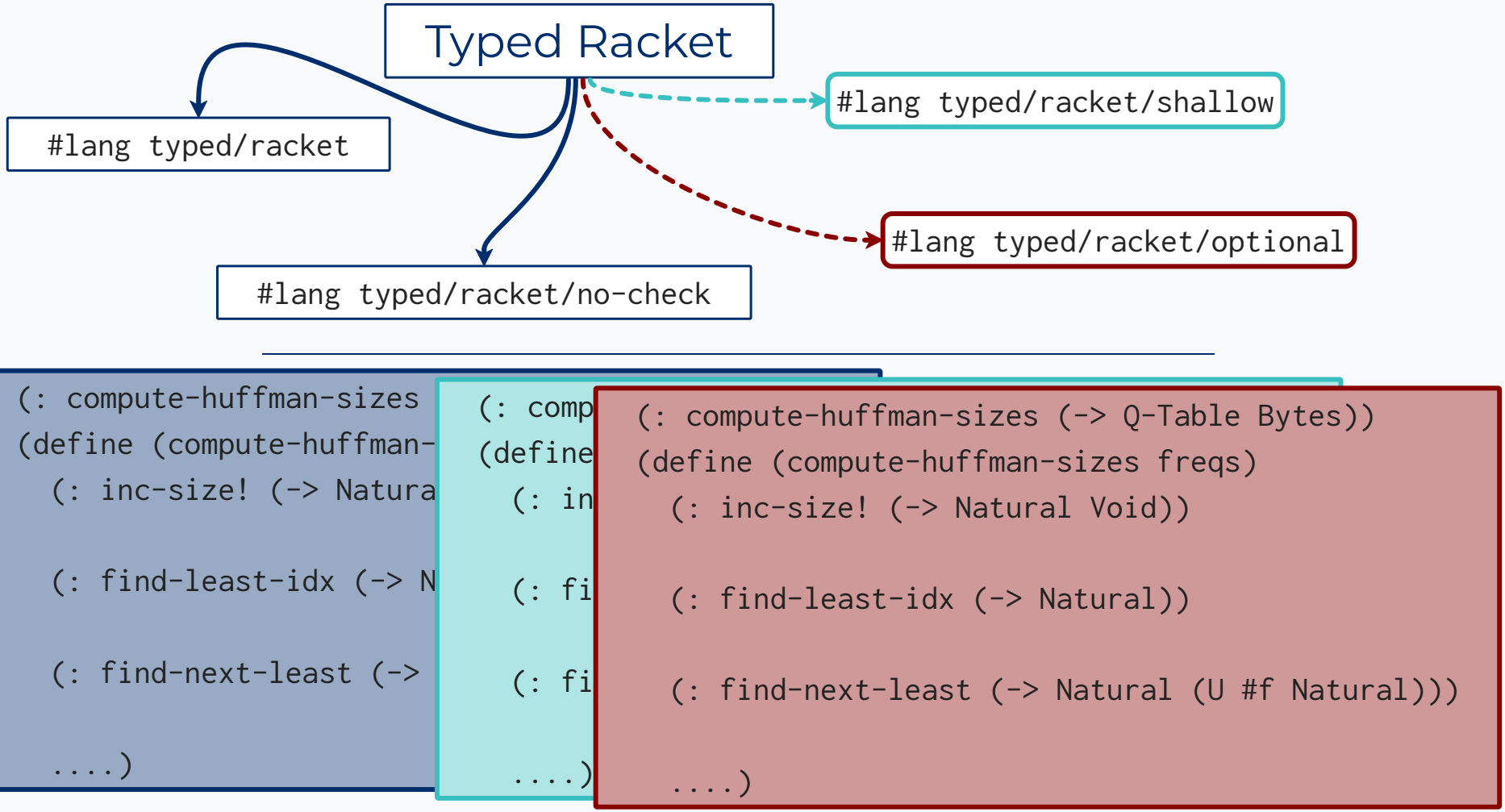


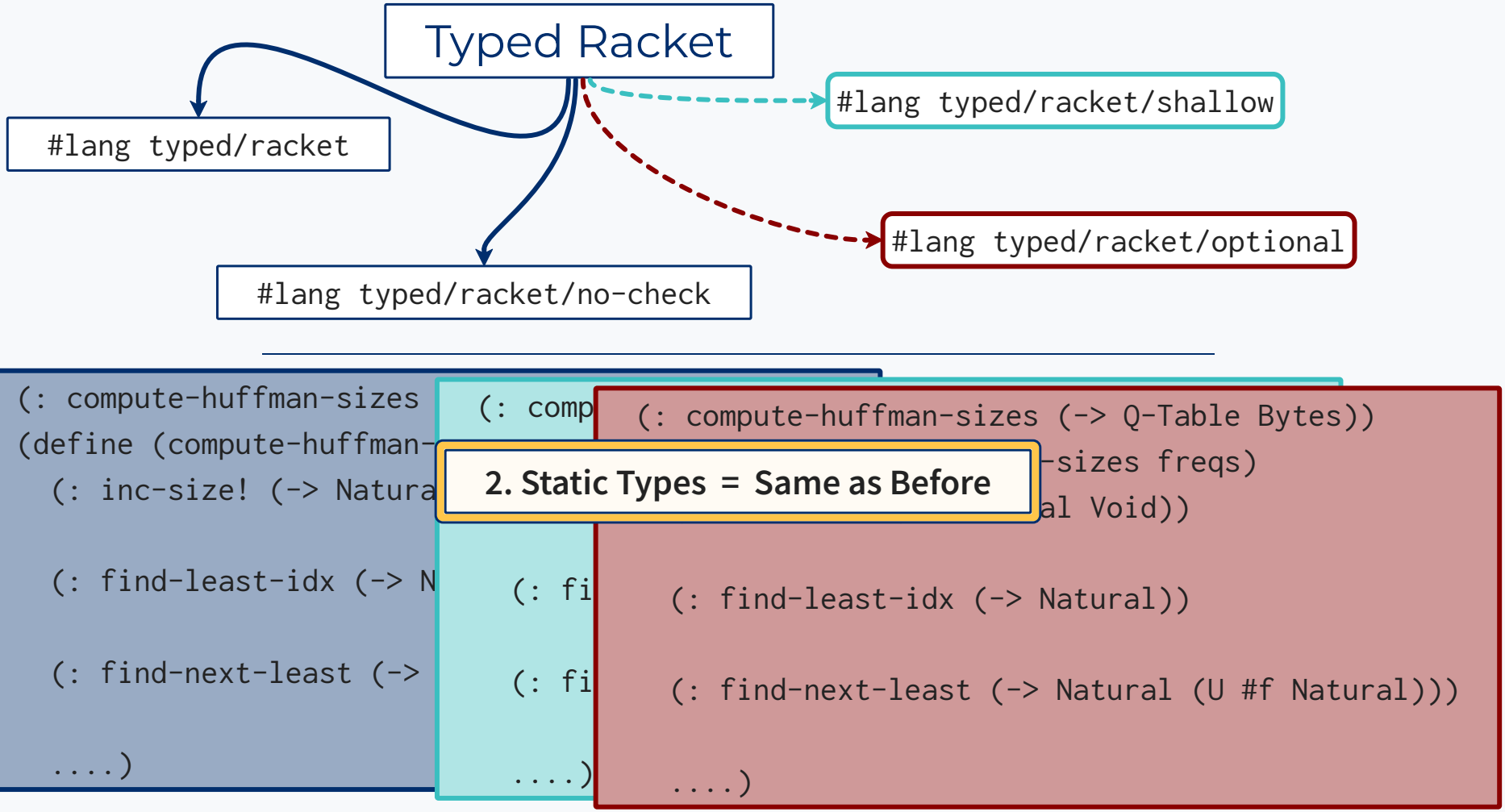
1. Two New Languages

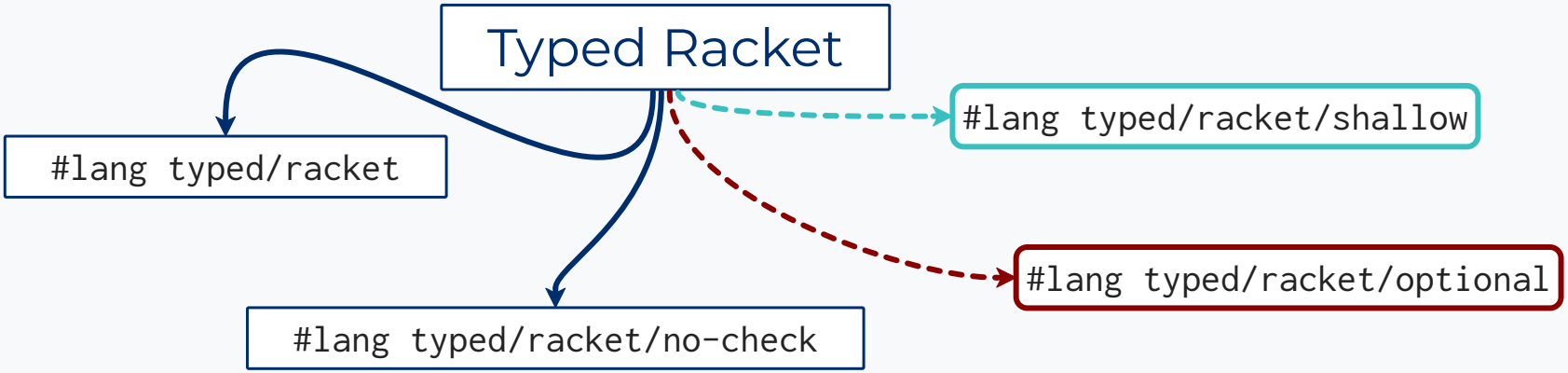

Typed Racket
#lang typed/racket
#lang typed/racket/shallow
#lang typed/racket/optional
#lang typed/racket/no-check

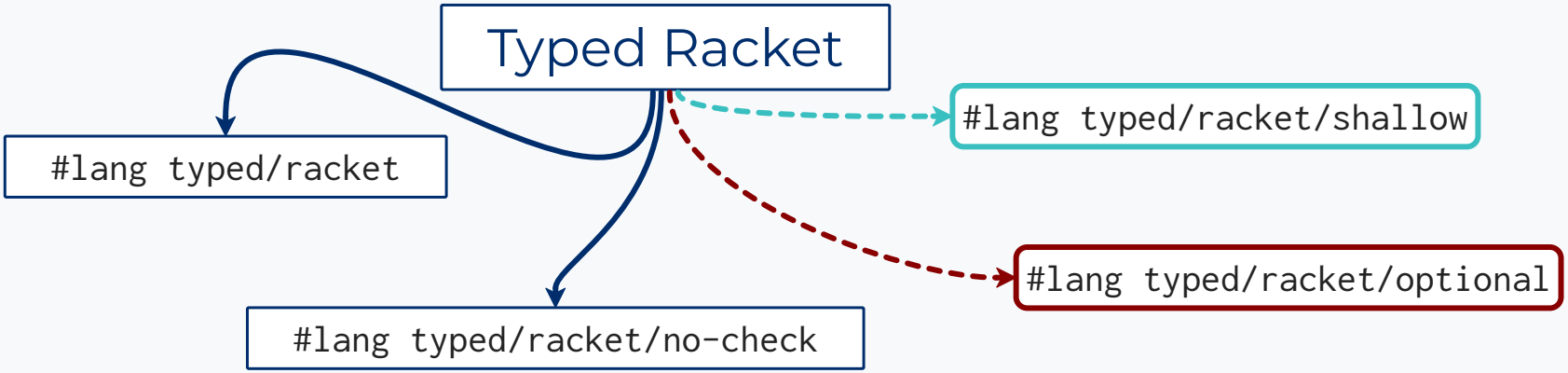



```
(: compute-huffman-sizes (-> Q-Table Bytes))
(define (compute-huffman-sizes freqs)
  (: inc-size! (-> Natural Void))

  (: find-least-idx (-> Natural))

  (: find-next-least (-> Natural (U #f Natural)))

  ....)
```

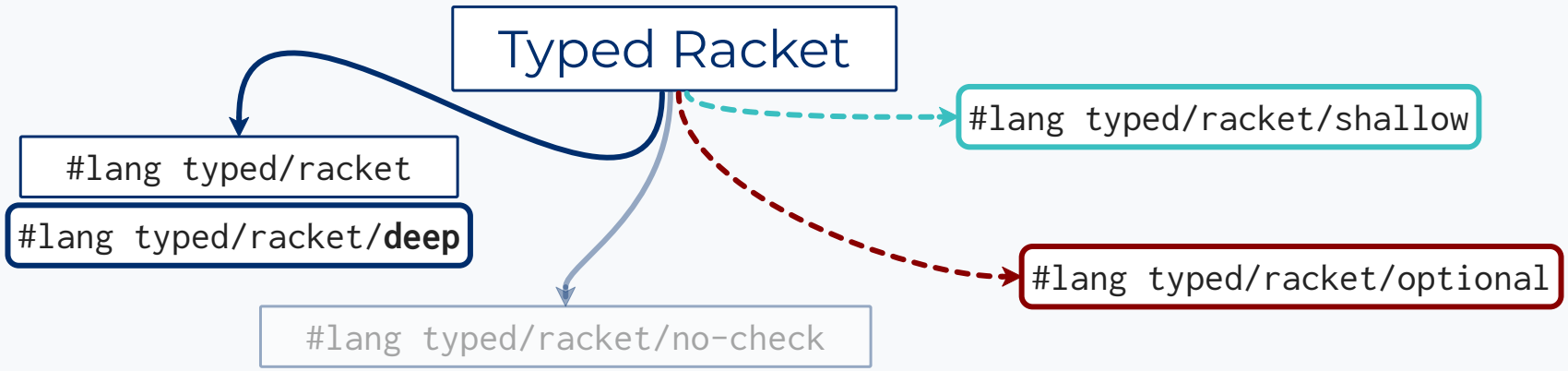

Typed Racket
#lang typed/racket
#lang typed/racket/shallow
#lang typed/racket/optional
#lang typed/racket/no-check
(: compute-huffman-sizes
(define (compute-huffman-
(: inc-size! (-> Natura
(: find-least-idx (-> N
(: find-next-least (->
....)
(: comp
(define
(: in
(: fi
(: fi
....)
(: compute-huffman-sizes (-> Q-Table Bytes))
(define (compute-huffman-sizes freqs)
(: inc-size! (-> Natural Void))
(: find-least-idx (-> Natural))
(: find-next-least (-> Natural (U #f Natural)))
....)


Typed Racket
#lang typed/racket
#lang typed/racket/shallow
#lang typed/racket/optional
#lang typed/racket/no-check
(: compute-huffman-sizes
(define (compute-huffman-
(: inc-size! (-> Natura
(: find-least-idx (-> N
(: find-next-least (->
....)
(: comp
(: fi
(: fi
....)
(: compute-huffman-sizes (-> Q-Table Bytes))
-sizes freqs)
al Void))
(: find-least-idx (-> Natural))
(: find-next-least (-> Natural (U #f Natural)))
....)
2. Static Types = Same as Before

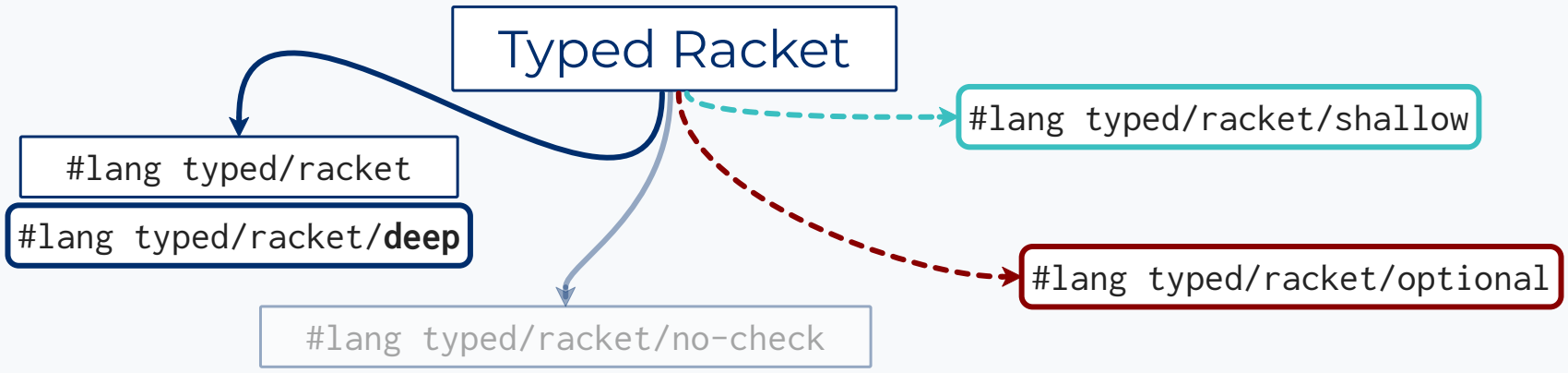

Typed Racket
#lang typed/racket
#lang typed/racket/shallow
#lang typed/racket/optional
#lang typed/racket/no-check



3. Better Performance at Type Boundaries

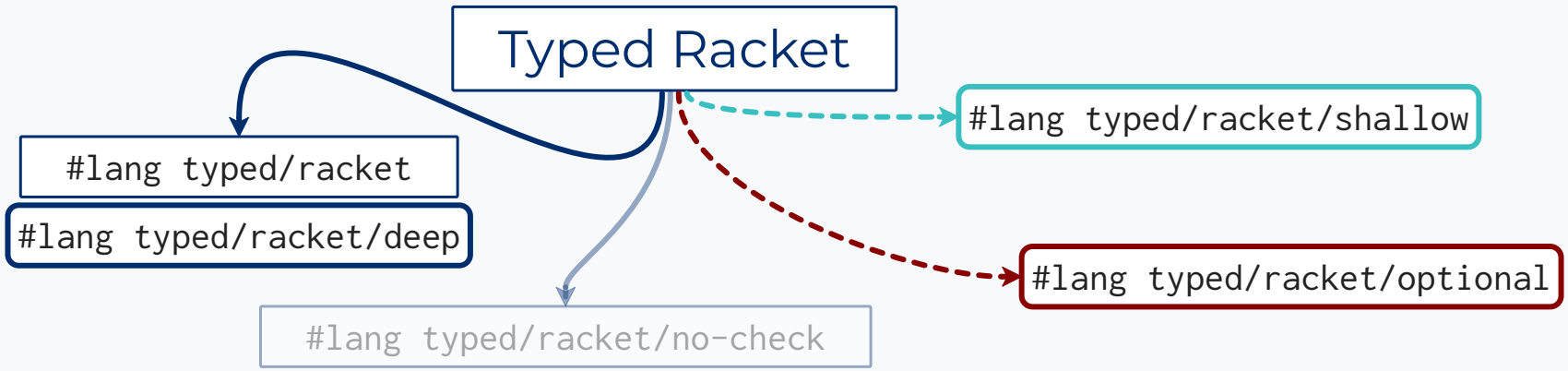


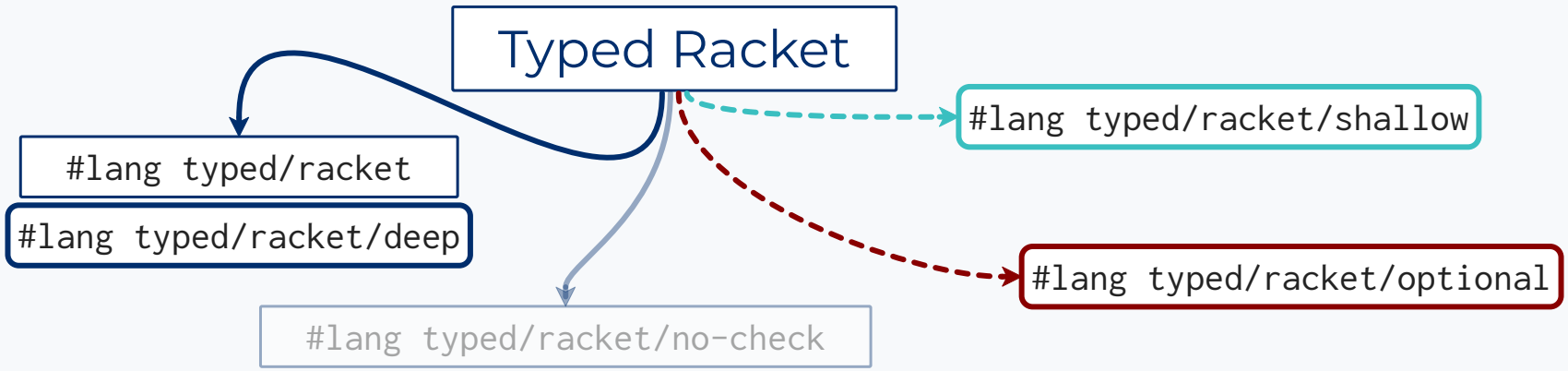
3. Better Performance at Type Boundaries



3. Better Performance at Type Boundaries

`deep`	Check everything
`shallow`	Check top-level shapes
`optional`	Check nothing


Typed Racket
#lang typed/racket
#lang typed/racket/deep
#lang typed/racket/shallow
#lang typed/racket/optional
#lang typed/racket/no-check



1. Two New Languages
2. Static Types = Same as Before
3. Better Performance at Type Boundaries

Typed Racket

1. Two New Languages
2. Static Types = Same as Before
3. Better Performance at Type Boundaries

Typed Racket

1. Two New Languages
2. Static Types = Same as Before
3. Better Performance at Type Boundaries

Typed Racket

Strong Guarantees,
High Overhead

Weak Guarantees,
Low Overhead

---

1. Two New Languages

2. Static Types = Same as Before

3. Better Performance at Type Boundaries

Example 1 D

Example 1 D

```
(: says-moo? (-> (Listof Symbol)
                 Boolean))
(define (says-moo? lst)
  (cond
   [(null? lst)
    #false]
   [(eq? 'moo (car lst))
    #true]
   [else
    (says-moo? (cdr lst))]))
```

Example 1 D

```
(: says-moo? (-> (Listof Symbol)
                 Boolean))
(define (says-moo? lst)
  (cond
   [(null? lst)
    #false]
   [(eq? 'moo (car lst))
    #true]
   [else
    (says-moo? (cdr lst))]))
```

Example 1 D

```
(: says-moo? (-> (Listof Symbol)
                 Boolean))
(define (says-moo? lst)
  (cond
   [(null? lst)
    #false]
   [(eq? 'moo (car lst))
    #true]
   [else
    (says-moo? (cdr lst))]))
```

Example 1 D

deep Types

```
(: says-moo? (-> (Listof Symbol)
                 Boolean))
(define (says-moo? lst)
  (cond
   [(null? lst)
    #false]
   [(eq? 'moo (car lst))
    #true]
   [else
    (says-moo? (cdr lst))]))
```

Example 1 D

deep Types

```
(: says-moo? (-> (Listof Symbol)
                 Boolean))
(define (says-moo? lst)
  (cond
   [(null? lst)
    #false]
   [(eq? 'moo (car lst))
    #true]
   [else
    (says-moo? (cdr lst))]))
```

Untyped Code

```
(says-moo? '(A B moo C D E F G))
```

Example 1 D

deep Types

```
(: says-moo? (-> (Listof Symbol)
                 Boolean))
(define (says-moo? lst)
  (cond
   [(null? lst)
    #false]
   [(eq? 'moo (car lst))
    #true]
   [else
    (says-moo? (cdr lst))]))
```

Untyped Code

```
(says-moo? '(A B moo C D E F G))
```

Example 1 D

deep Types

```
(: says-moo? (-> (Listof Symbol)
                 Boolean))
(define (says-moo? lst)
  (cond
   [(null? lst)
    #false]
   [(eq? 'moo (car lst))
    #true]
   [else
    (says-moo? (cdr lst))]))
```

Guarantee: list of symbols

Cost: check the **entire list**

Untyped Code

```
(says-moo? '(A B moo C D E F G))
```

Example 1 D

deep Types

```
(: says-moo? (-> (Listof Symbol)
                 Boolean))
(define (says-moo? lst)
  (cond
   [(null? lst)
    #false]
   [(eq? 'moo (car lst))
    #true]
   [else
    (says-moo? (cdr lst))]))
```

Guarantee: list of symbols
Cost: check the **entire list**

Untyped Code

```
(says-moo? '(A B moo C D E F G))
```

deep : contracts at boundaries

Example 1S

`shallow` Types

```
(: says-moo? (-> (Listof Symbol)
                 Boolean))
(define (says-moo? lst)
  (cond
   [(null? lst)
    #false]
   [(eq? 'moo (car lst))
    #true]
   [else
    (says-moo? (cdr lst))]))
```

Untyped Code

```
(says-moo? '(A B moo C D E F G))
```

Example 1S

shallow Types

```
(: says-moo? (-> (Listof Symbol)
                 Boolean))
(define (says-moo? lst)
  (cond
   [(null? lst)
    #false]
   [(eq? 'moo (car lst))
    #true]
   [else
    (says-moo? (cdr lst))]))
```

Untyped Code

```
(says-moo? '(A B moo C D E F G))
```

Example 1S

`shallow` Types

```
(: says-moo? (-> (Listof Symbol)
                 Boolean))
(define (says-moo? lst)
  (cond
   [(null? lst)
    #false]
   [(eq? 'moo (car lst))
    #true]
   [else
    (says-moo? (cdr lst))]))
```

Guarantee: a list
Cost: `list?`

Untyped Code

```
(says-moo? '(A B moo C D E F G))
```

Example 1S

`shallow` Types

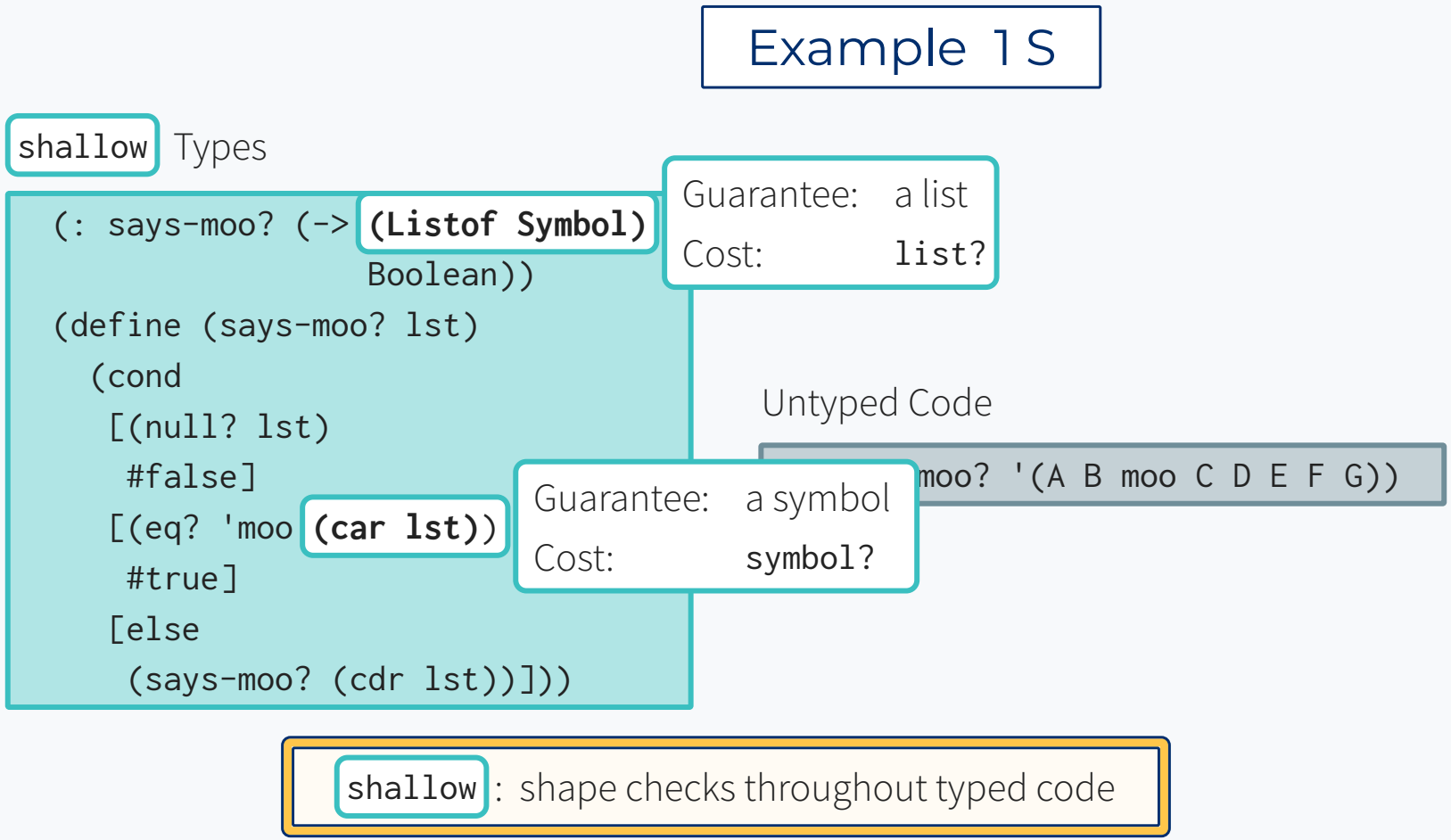
```
(: says-moo? (-> (Listof Symbol)
                 Boolean))
(define (says-moo? lst)
  (cond
   [(null? lst)
    #false]
   [(eq? 'moo (car lst))
    #true]
   [else
    (says-moo? (cdr lst))]))
```

Guarantee: a list
Cost: `list?`

Guarantee: a symbol
Cost: `symbol?`

Untyped Code

```
moo? '(A B moo C D E F G))
```


Example 1S
shallow Types
(: says-moo? (-> (Listof Symbol) Boolean))
(define (says-moo? lst)
(cond
[(null? lst)
#false]
[(eq? 'moo (car lst))
#true]
[else
(says-moo? (cdr lst))]))
Guarantee: a list
Cost: list?
Untyped Code
moo? '(A B moo C D E F G))
Guarantee: a symbol
Cost: symbol?
shallow : shape checks throughout typed code

Example 10

`optional` Types

```
(: says-moo? (-> (Listof Symbol)
                 Boolean))
(define (says-moo? lst)
  (cond
   [(null? lst)
    #false]
   [(eq? 'moo (car lst))
    #true]
   [else
    (says-moo? (cdr lst))]))
```

Untyped Code

```
(says-moo? '(A B moo C D E F G))
```

Example 10

`optional` Types

```
(: says-moo? (-> (Listof Symbol)
                 Boolean))
(define (says-moo? lst)
  (cond
   [(null? lst)
    #false]
   [(eq? 'moo (car lst))
    #true]
   [else
    (says-moo? (cdr lst))]))
```

Guarantee: any value
Cost: **free**

Untyped Code

```
(says-moo? '(A B moo C D E F G))
```

Example 10

`optional` Types

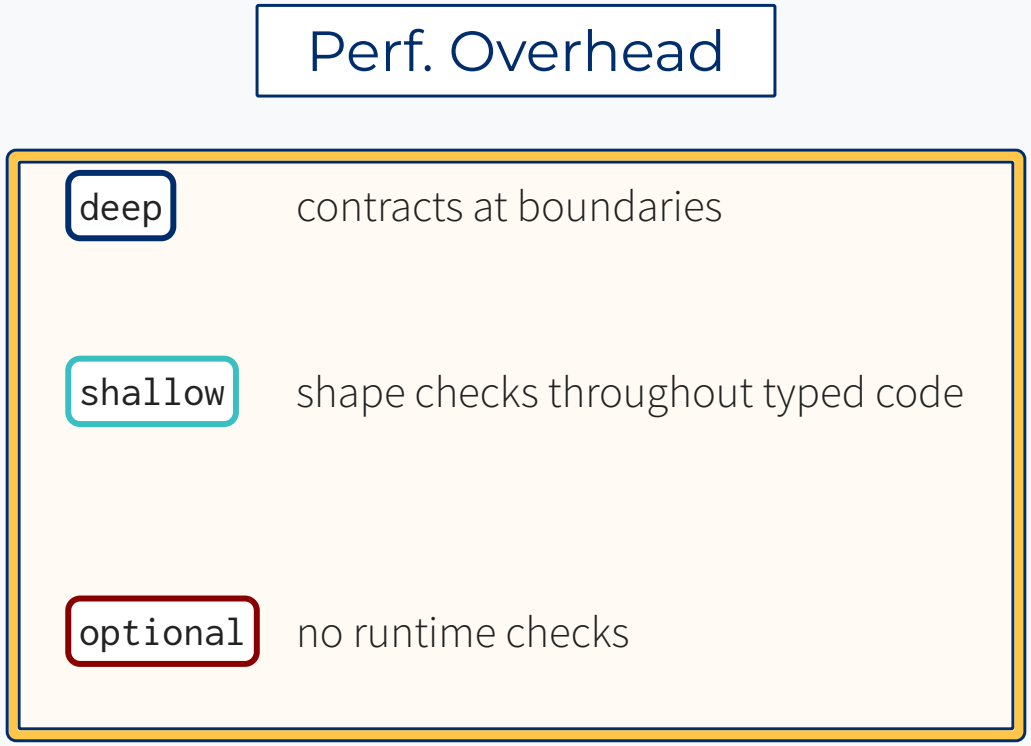
```
(: says-moo? (-> (Listof Symbol)
                 Boolean))
(define (says-moo? lst)
  (cond
   [(null? lst)
    #false]
   [(eq? 'moo (car lst))
    #true]
   [else
    (says-moo? (cdr lst))]))
```

Guarantee: any value
Cost: **free**

Untyped Code

```
(says-moo? '(A B moo C D E F G))
```

`optional`: no runtime checks


Perf. Overhead
deep
contracts at boundaries
shallow
shape checks throughout typed code
optional
no runtime checks

Perf. Overhead

`deep` contracts at boundaries

- can be expensive: `->`, `HashTable`, ...

`shallow` shape checks throughout typed code

`optional` no runtime checks

Perf. Overhead

`deep` contracts at boundaries

- can be expensive: `->`, `HashTable`, ...

`shallow` shape checks throughout typed code

- often cheap, but add up!

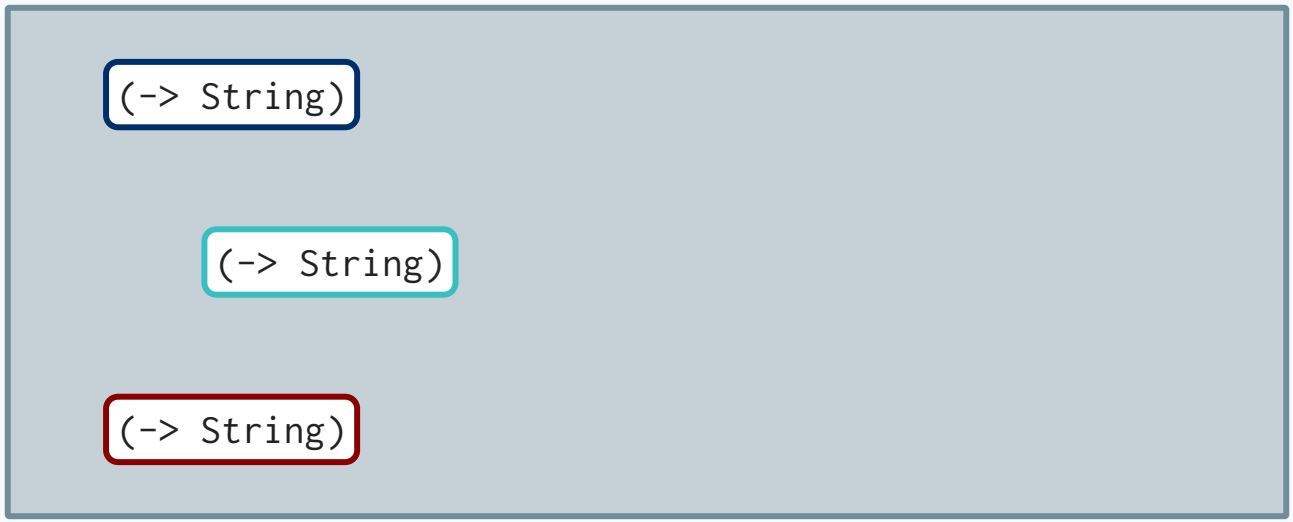
`optional` no runtime checks

Example 2

Untyped Codebase

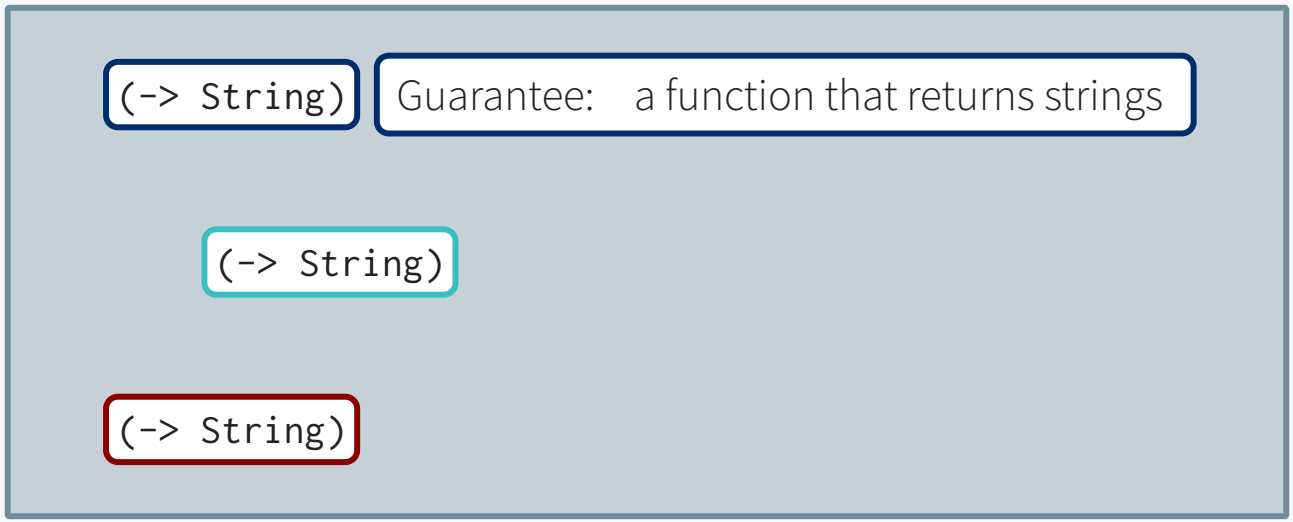
Example 2

Untyped Codebase ... with a few Types



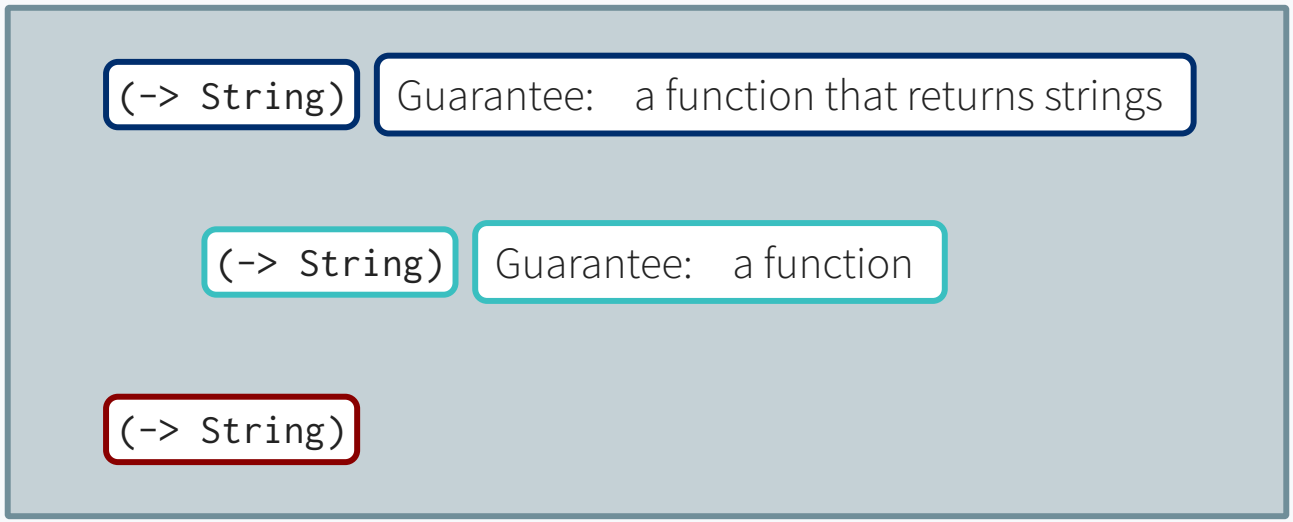
Example 2

Untyped Codebase ... with a few Types



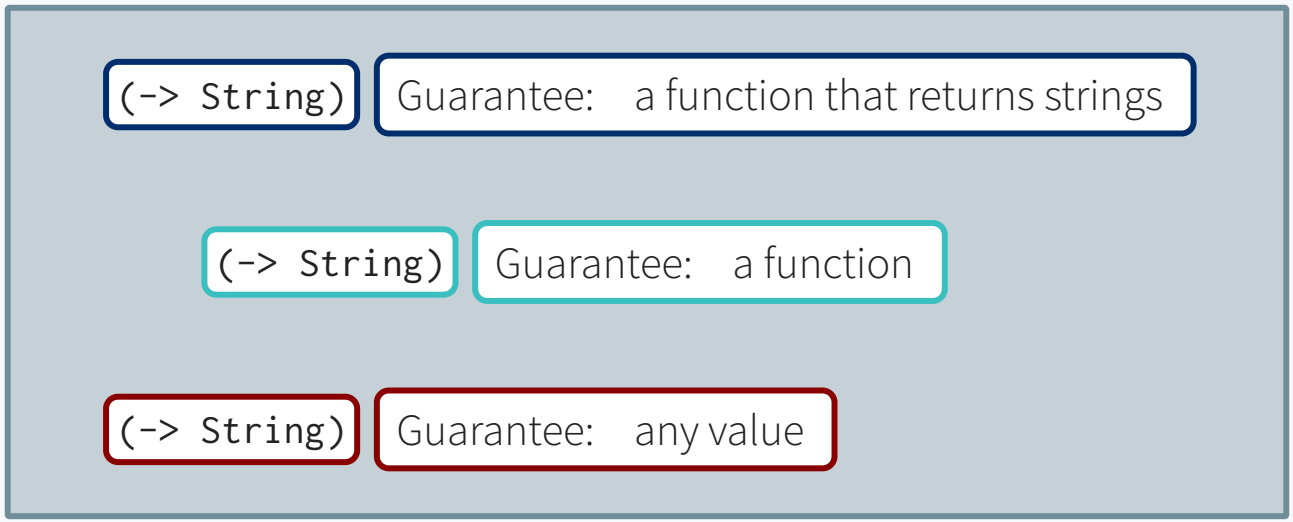
Example 2

Untyped Codebase ... with a few Types



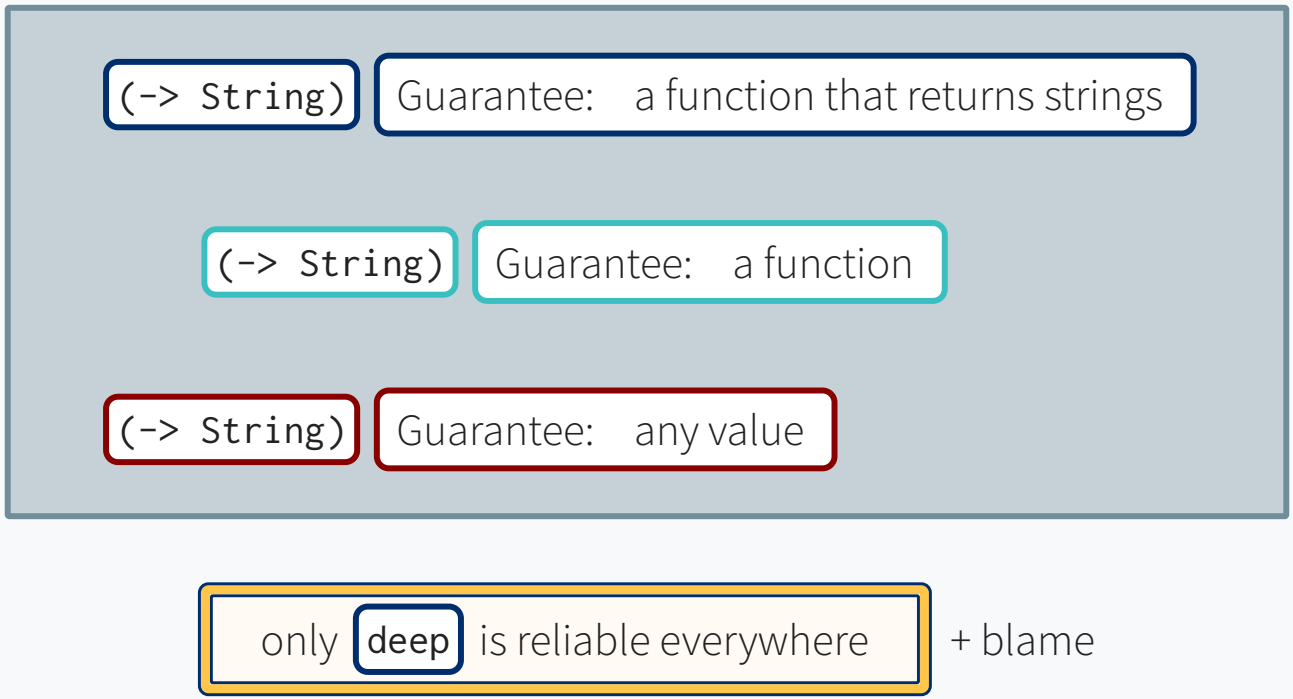
Example 2

Untyped Codebase ... with a few Types



Example 2

Untyped Codebase ... with a few Types

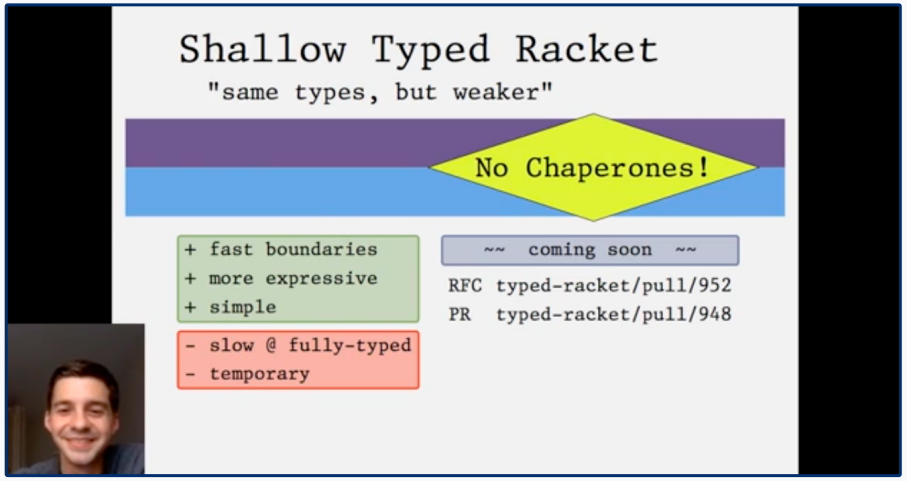


only `deep` is reliable everywhere + blame

Typed Racket


in
Racket 8.7

Announced in 2020 ...



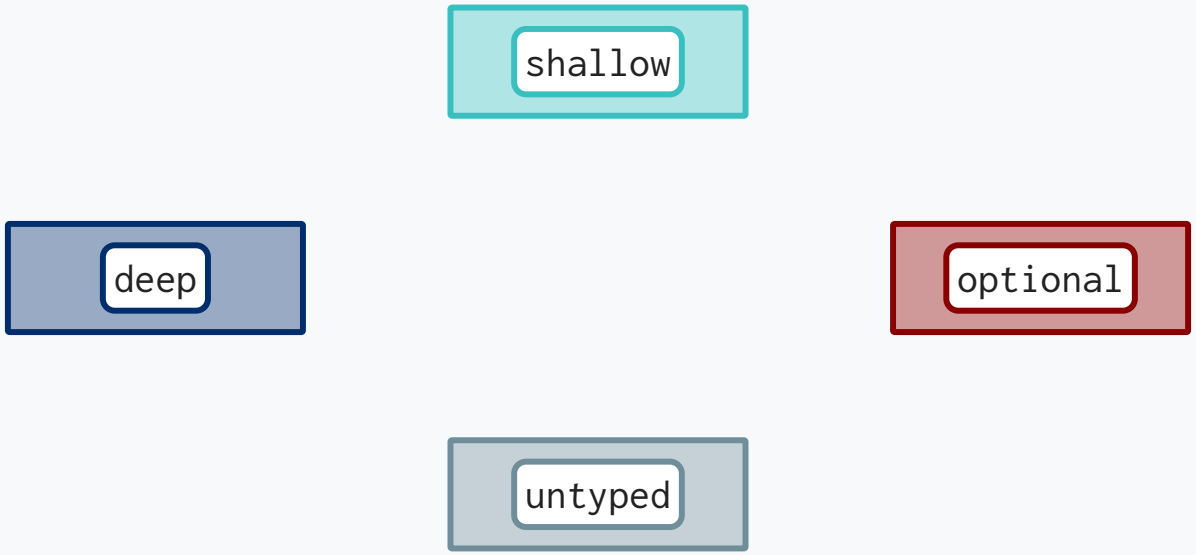
What took so long?

1. Life Stuff

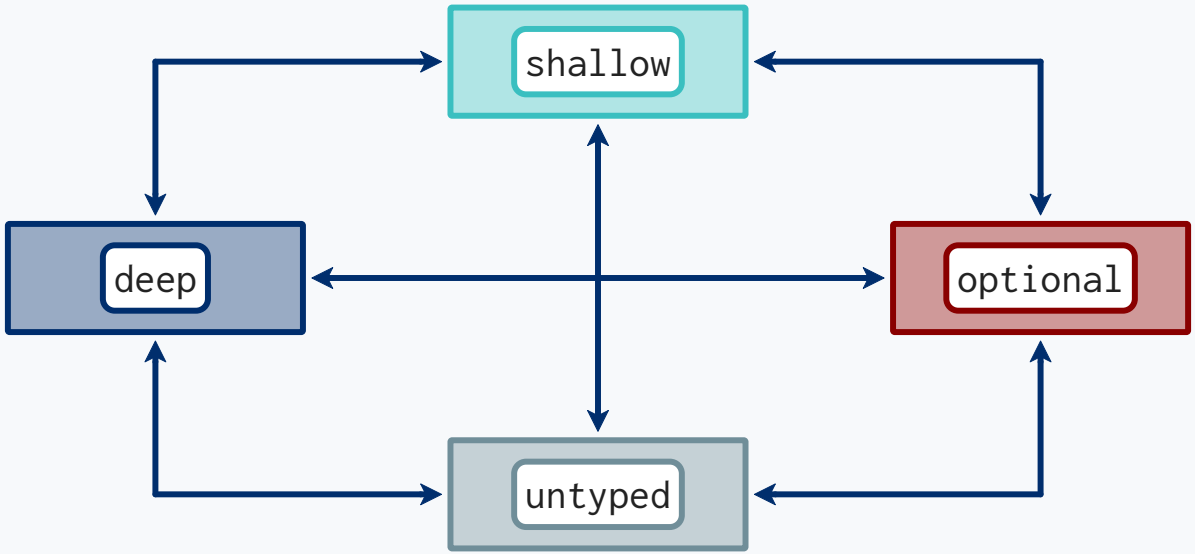
BROWN

1. Life Stuff

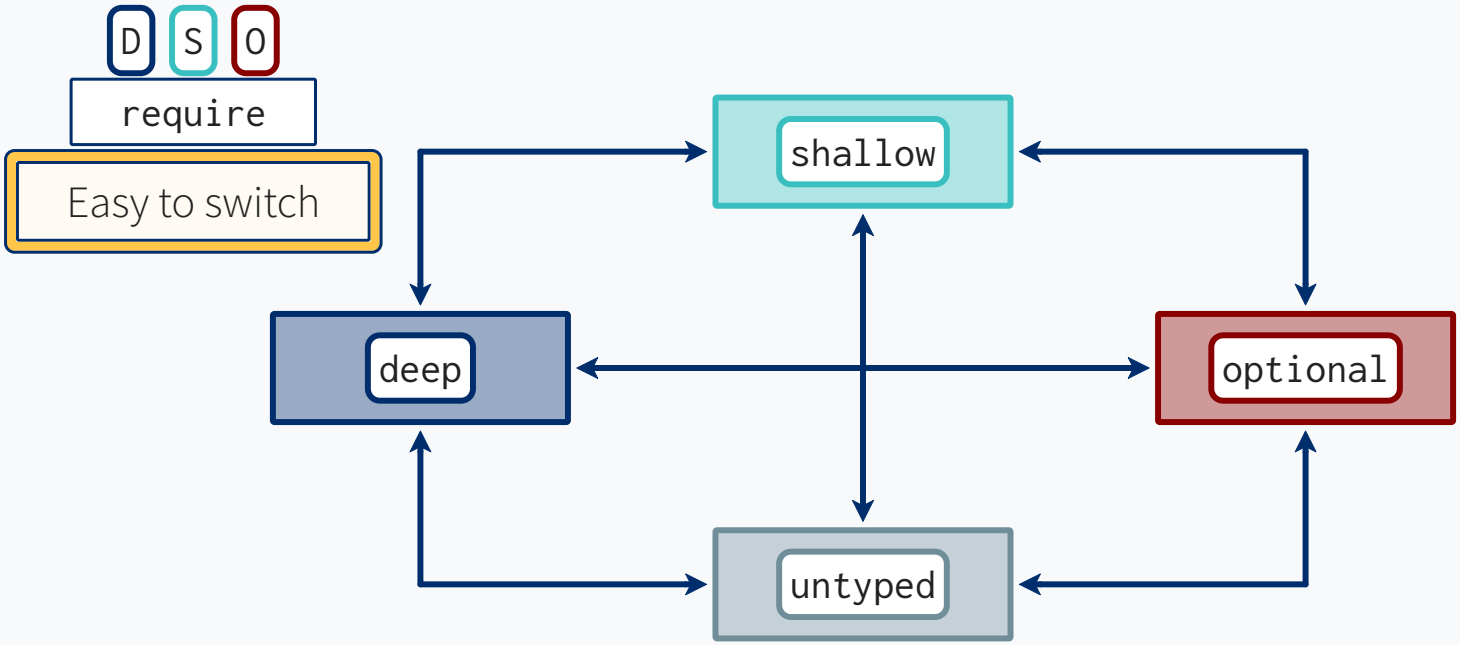
2. Well-Typed Interactions



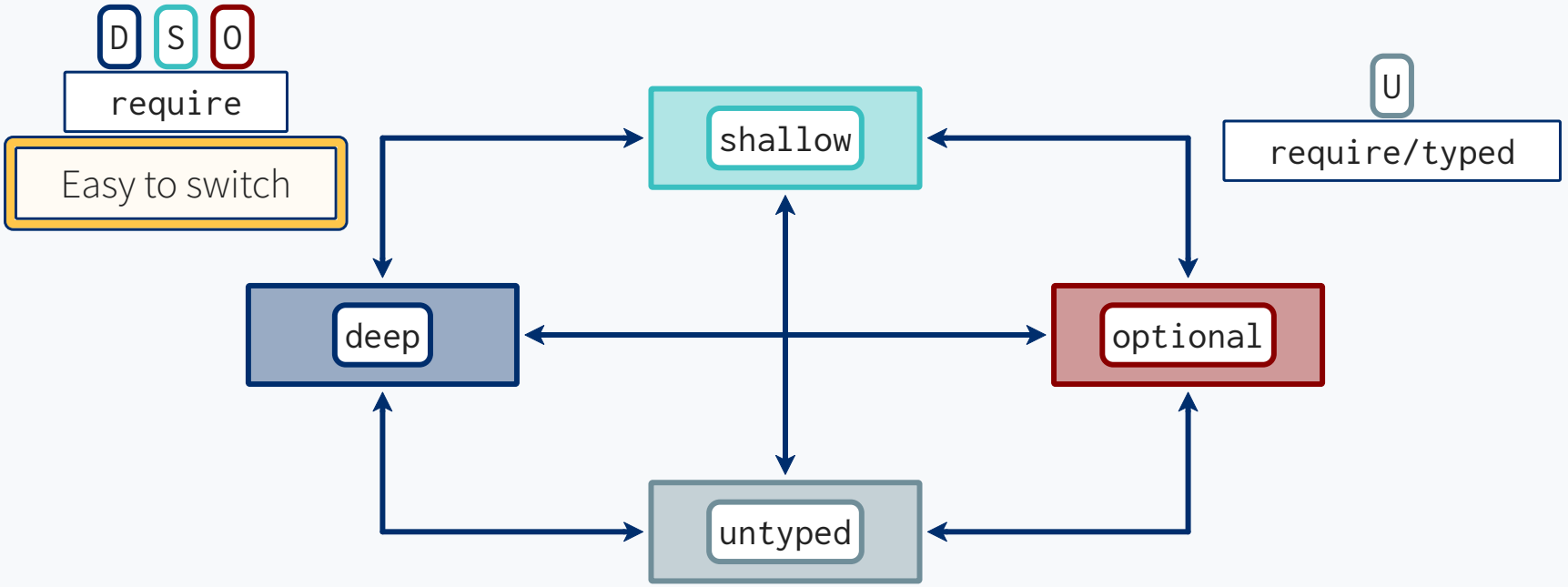
2. Well-Typed Interactions



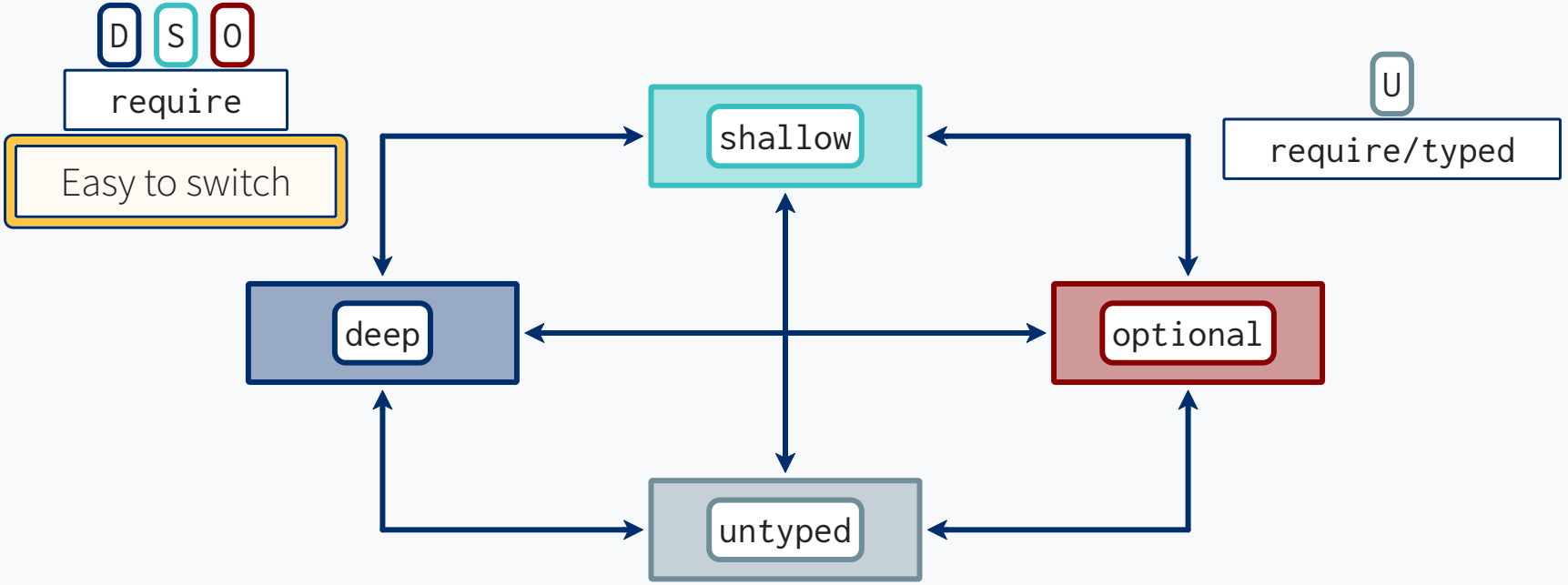
2. Well-Typed Interactions



2. Well-Typed Interactions



2. Well-Typed Interactions



PLDI '22

* macros, define-typed/untyped-id, ...

3. Faster Shallow

Better base types, fewer shape checks

3. Faster Shallow

Better base types, fewer shape checks

Problem: every function call **might** need a shape check

3. Faster Shallow

Better base types, fewer shape checks

Problem: every function call **might** need a shape check

```
(: says-moo? (-> (Listof Symbol)
                 Boolean))
(define (says-moo? lst)
  (cond
   [(null? lst)
    #false]
   [(eq? 'moo (car lst))
    #true]
   [else
    (says-moo? (cdr lst))]))
```

3. Faster Shallow

Better base types, fewer shape checks

Problem: every function call **might** need a shape check

```
(: says-moo? (-> (Listof Symbol)
                 Boolean))
(define (says-moo? lst)
  (cond
   [(null? lst)
    #false]
   [(eq? 'moo (car lst))
    #true]
   [else
    (says-moo? (cdr lst))]))
```

Yes check

3. Faster Shallow

Better base types, fewer shape checks

Problem: every function call **might** need a shape check

```
(: says-moo? (-> (Listof Symbol)
                 Boolean))
(define (says-moo? lst)
  (cond
   [(null? lst)
    #false]
   [(eq? 'moo (car lst))
    #true]
   [else
    (says-moo? (cdr lst))]))
```

No check

Yes check

No check

3. Faster Shallow

Better base types, fewer shape checks

Problem: every function call **might** need a shape check

```
(: says-moo? (-> (Listof Symbol)
                 Boolean))
(define (says-moo? lst)
  (cond
    [(null? lst) #false]
    [(eq? 'moo (car lst)) #true]
    [else (says-moo? (cdr lst))]))
```

No check

Yes check

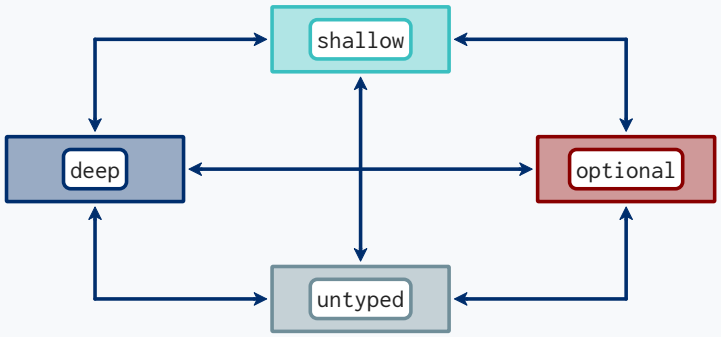
No check

Old Solution: trust specific IDs

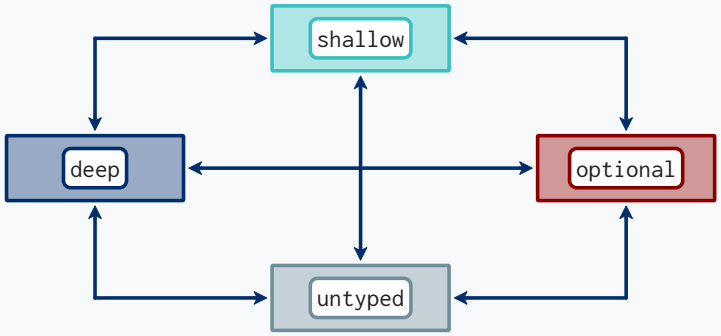
New Solution: type-based, compositional

++ Well-Typed Interactions

++ Faster Shallow



++ Well-Typed Interactions
++ Faster Shallow



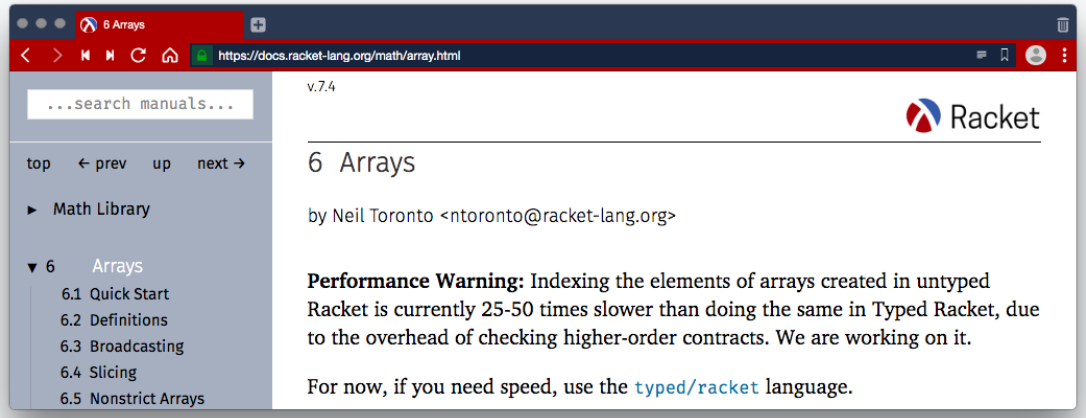
More to Come

- D/S Cooperation
- Even Faster Shallow
- Macro Reuse
- Occurrence Type Boundaries

Faster Math?

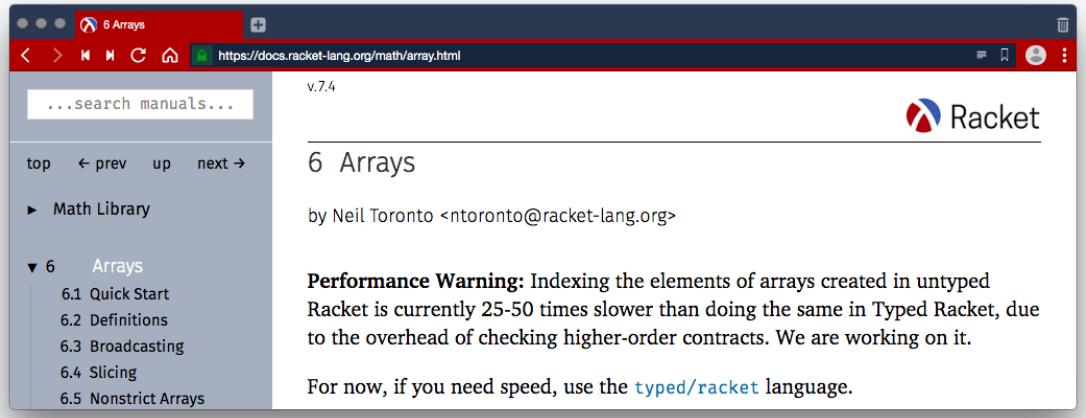
Faster Math?

Untyped slowdown: **25x to 50x**



Faster Math?

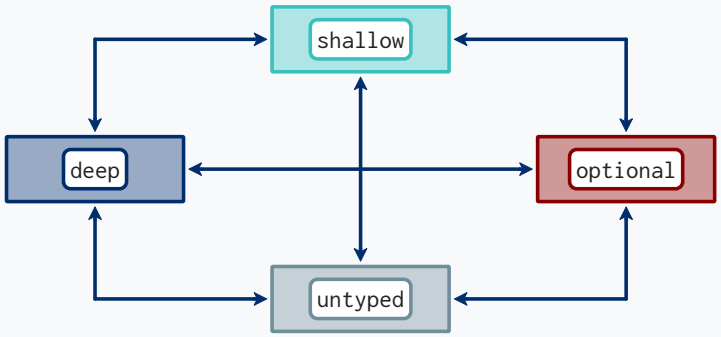
Untyped slowdown: **25x to 50x**



Not easy, but there's hope: `github.com/racket/math/issues/75`

++ Well-Typed Interactions

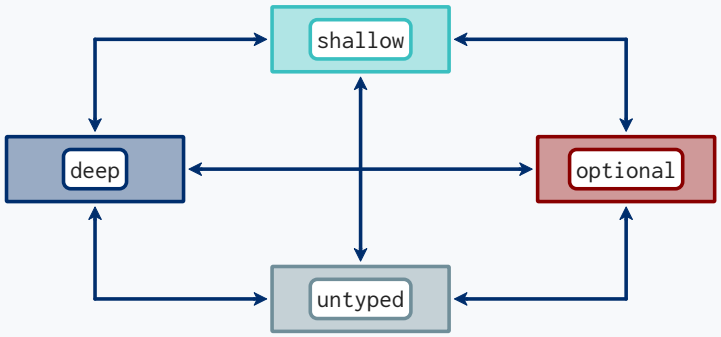
++ Faster Shallow



in Racket 8.7

++ Well-Typed Interactions

++ Faster Shallow



Shallow and **Optional** Types for Typed Racket

1. Two New Languages
2. Static Types = Same as Before
3. Better Performance at Type Boundaries

THE UNIVERSITY OF UTAH