

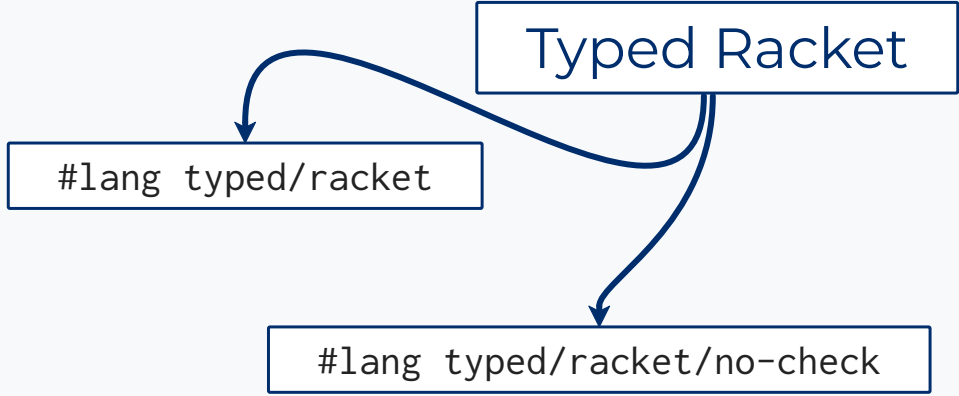
Shallow and **Optional** Types for Typed Racket

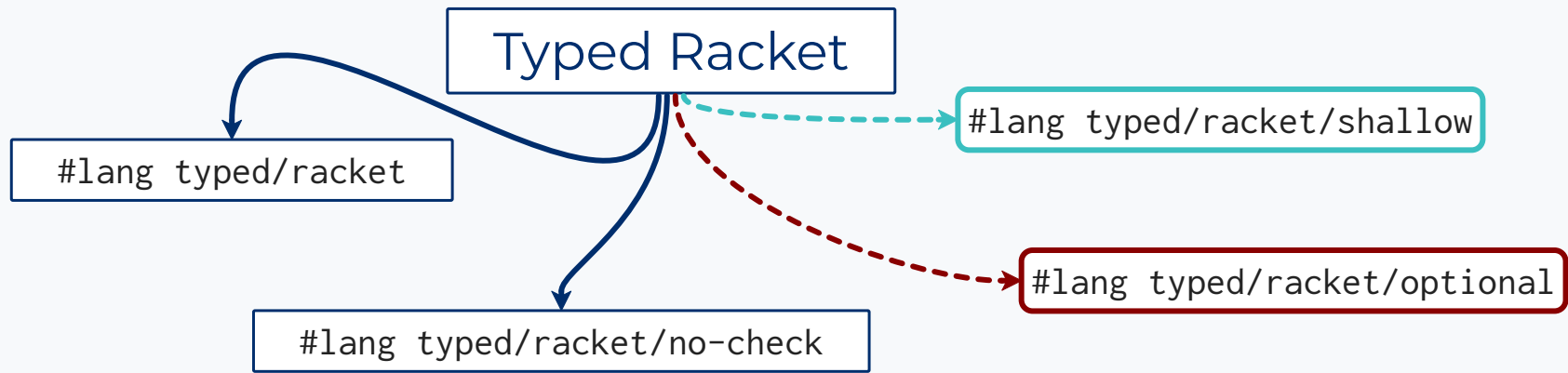


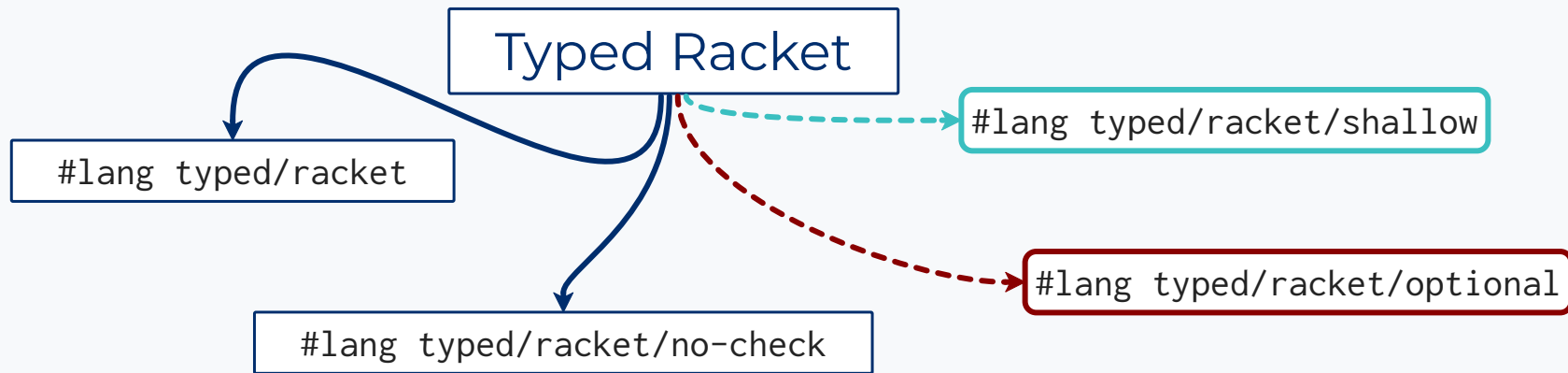
Ben Greenman

RacketCon 2022

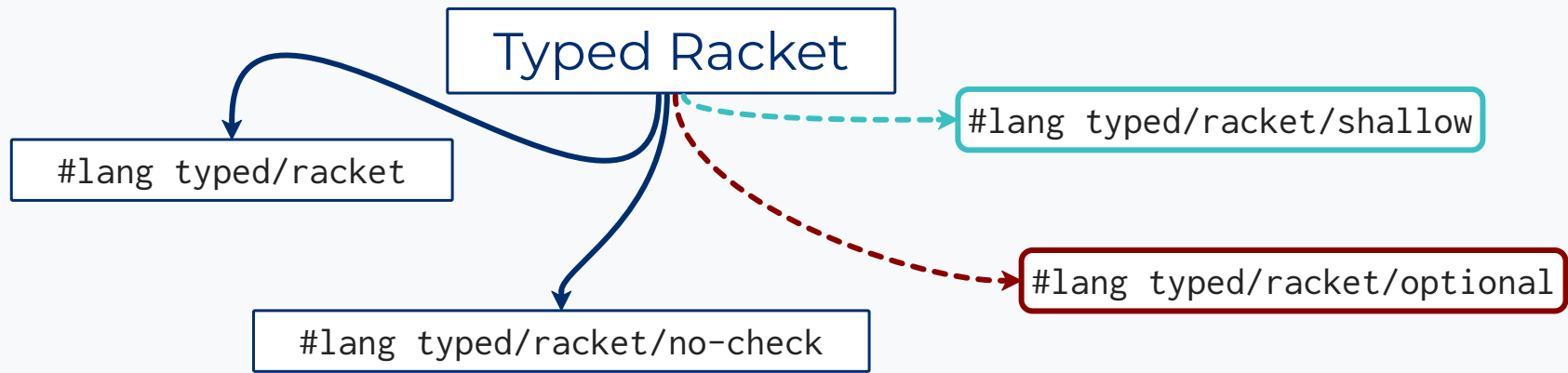
Typed Racket

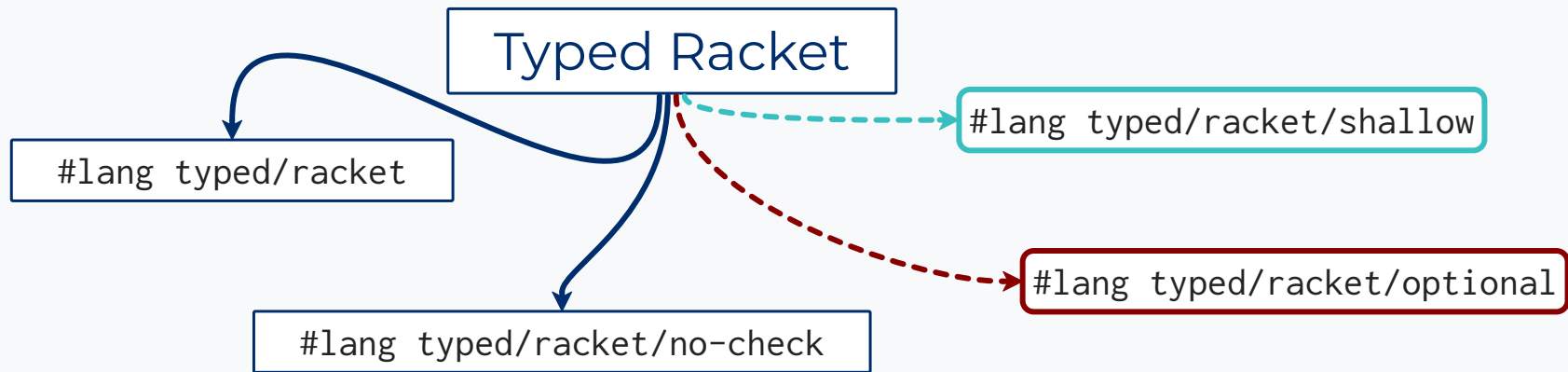






1. Two New Languages

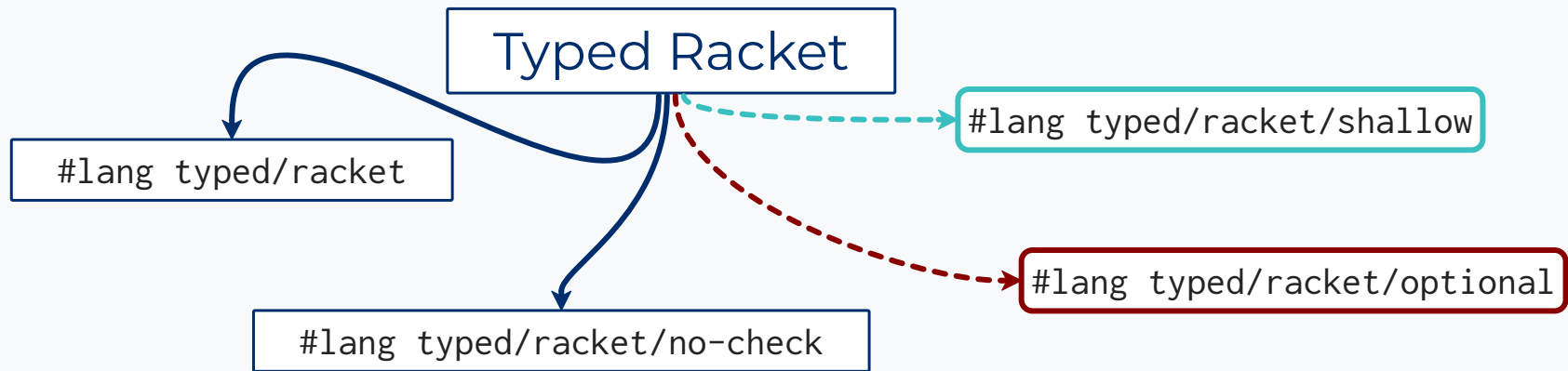




```
(: compute-huffman-sizes (-> Q-Table Bytes))
(define (compute-huffman-sizes freqs)
  (: inc-size! (-> Natural Void))

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

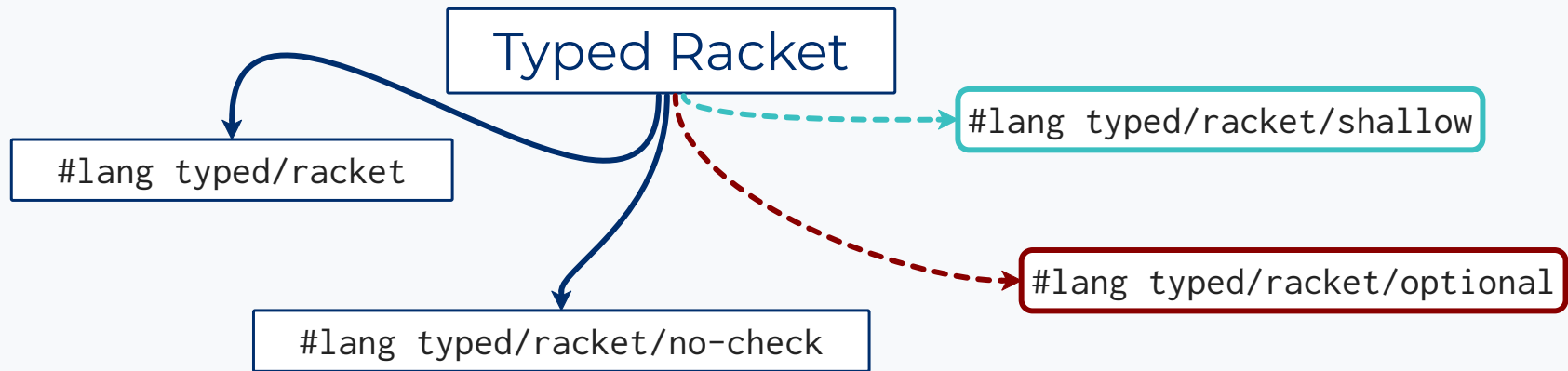
  (: find-next-least (-> Natural (U #f Natural)))
  ....)
```



```
(: compute-huffman-sizes
(define (compute-huffman-
  (: inc-size! (-> Natural)
  (: find-least-idx (-> Natural)
  (: find-next-least (-> Natural)
  ....))
```

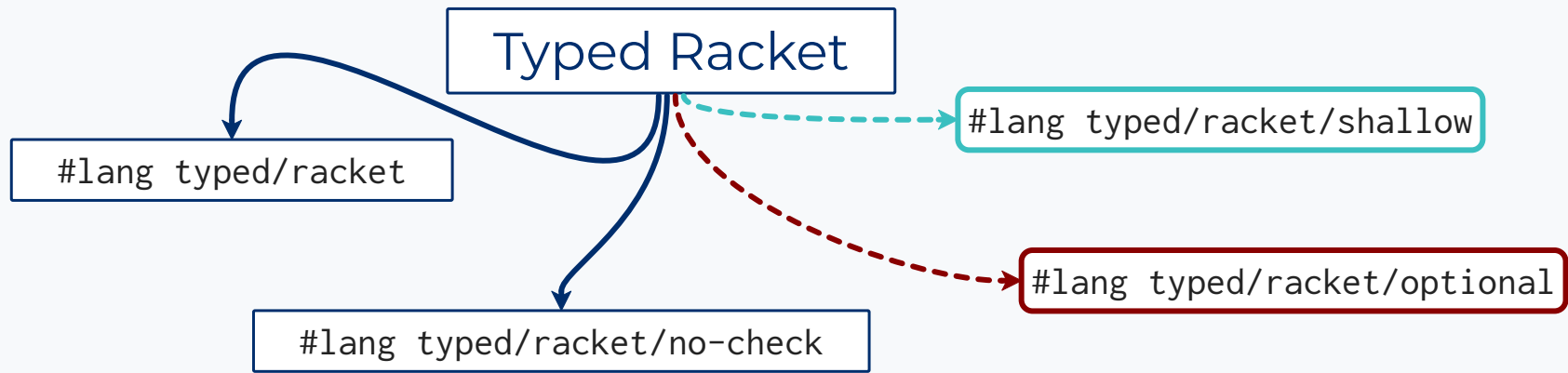
```
(: compute-huffman-sizes
(define (compute-huffman-
  (: inc-size! (-> Natural)
  (: find-least-idx (-> Natural)
  (: find-next-least (-> Natural)
  ....))
```

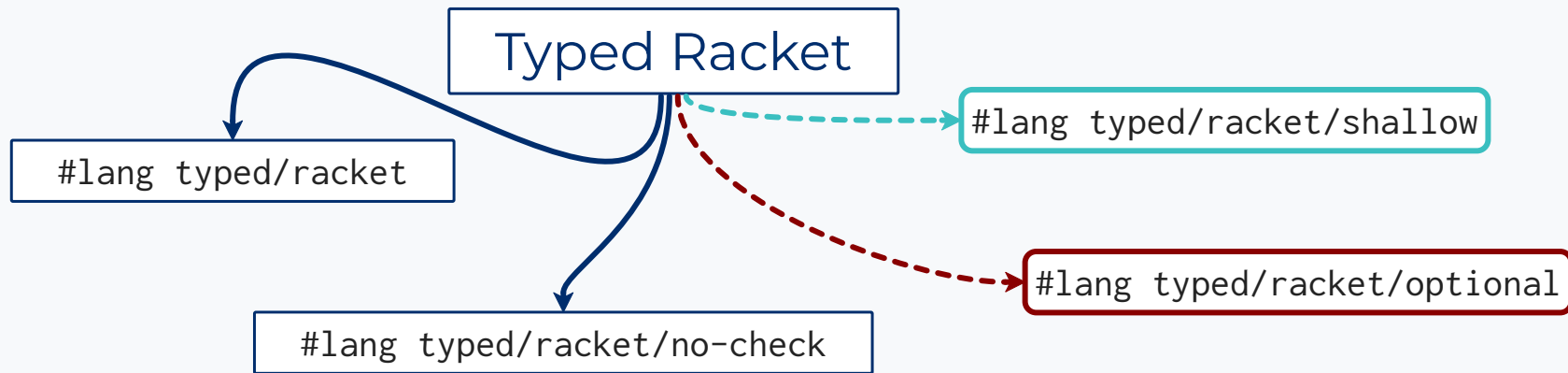
```
(: compute-huffman-sizes (-> Q-Table Bytes))
(define (compute-huffman-sizes freqs)
  (: inc-size! (-> Natural Void))
  (: find-least-idx (-> Natural))
  (: find-next-least (-> Natural (U #f Natural)))
  ....)
```

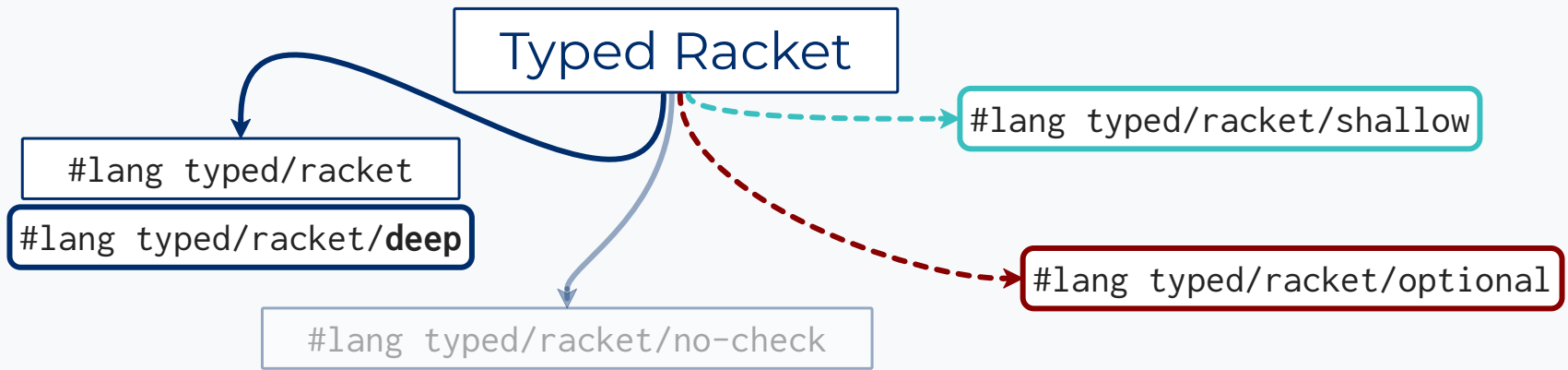
<pre>(: compute-huffman-sizes (define (compute-huffman- (: inc-size! (-> Natural) (: find-least-idx (-> Natural) (: find-next-least (-> Natural) )</pre>	<pre>(: compute-huffman-sizes (define (compute-huffman- (: inc-size! (-> Natural) (: find-least-idx (-> Natural) (: find-next-least (-> Natural) )</pre>	<pre>(: compute-huffman-sizes (-> Q-Table Bytes)) (define (compute-huffman- (-sizes freqs) (al Void)) (: find-least-idx (-> Natural)) (: find-next-least (-> Natural (U #f Natural))) )</pre>
---	---	--

2. Static Types = Same as Before

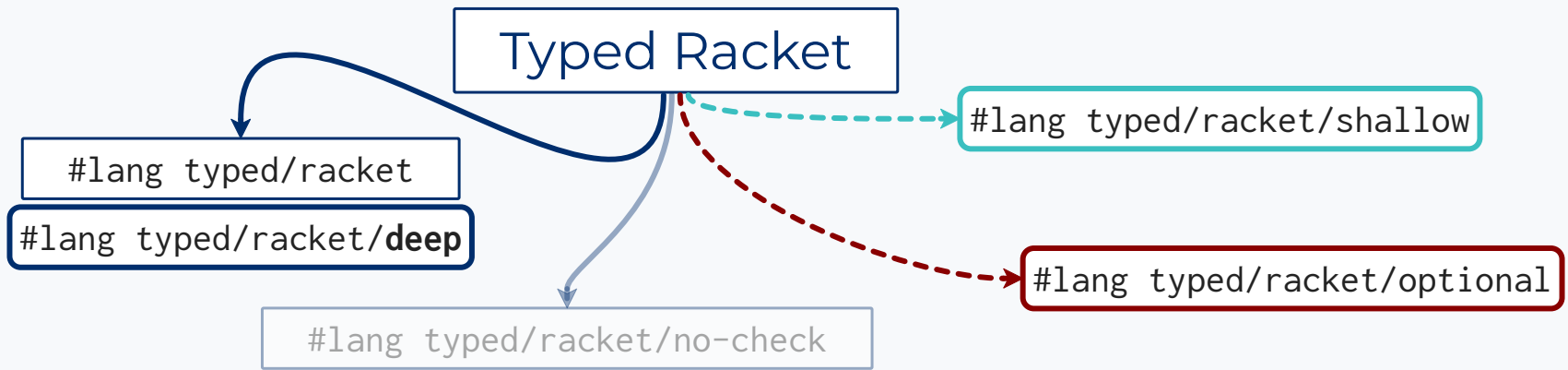




3. Better Performance at Type Boundaries

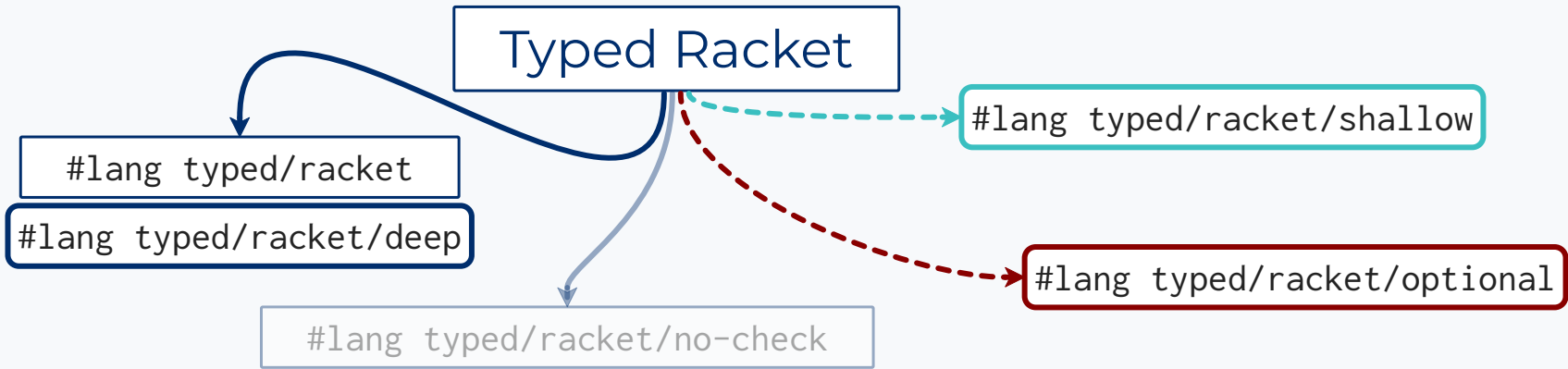


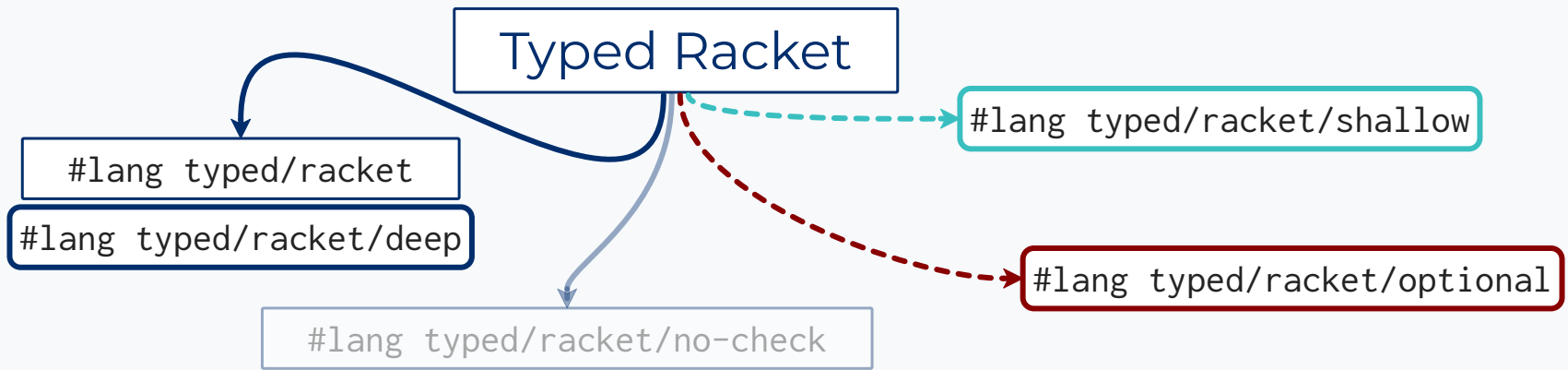
3. Better Performance at Type Boundaries



3. Better Performance at Type Boundaries

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





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

deep

shallow

optional

1. Two New Languages

2. Static Types = Same as Before

3. Better Performance at Type Boundaries

Typed Racket

Strong Guarantees,
High Overhead

deep

shallow

optional

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 1 S

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 1 S

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 1 S

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 1 S

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

Guarantee: a symbol
Cost: symbol?

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

Example 1 S

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

Guarantee: a symbol
Cost: symbol?

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

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

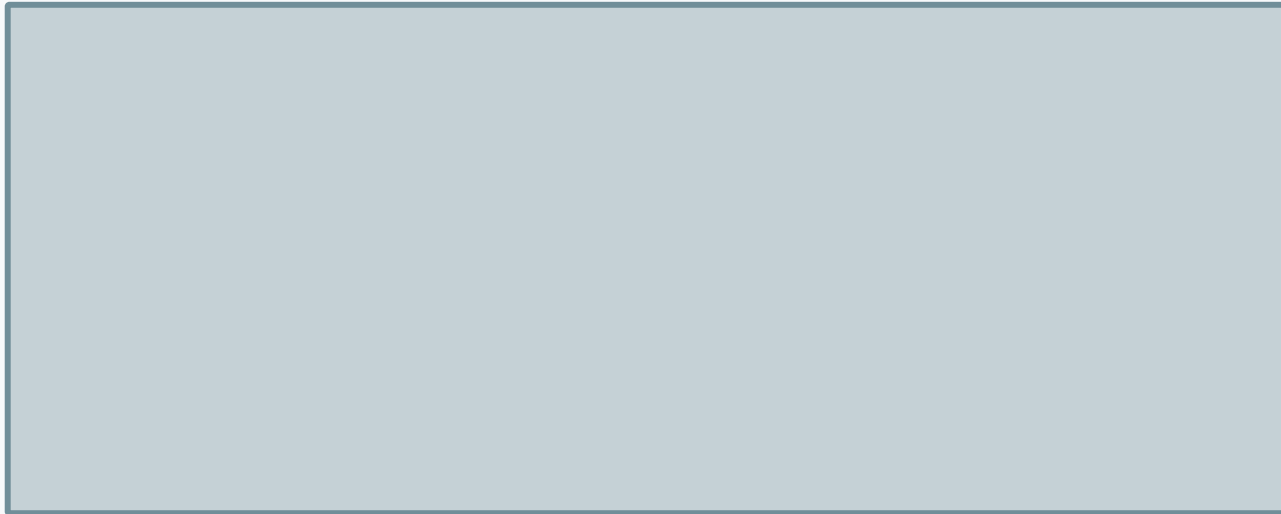
- often cheap, but add up!

optional

no runtime checks

Example 2

Untyped Codebase



Example 2

Untyped Codebase ... with a few Types

`(-> String)`

`(-> String)`

`(-> String)`

Example 2

Untyped Codebase ... with a few Types

`(-> String)`

Guarantee: a function that returns strings

`(-> String)`

`(-> String)`

Example 2

Untyped Codebase ... with a few Types

`(-> String)`

Guarantee: a function that returns strings

`(-> String)`

Guarantee: a function

`(-> String)`

Example 2

Untyped Codebase ... with a few Types

`(-> String)`

Guarantee: a function that returns strings

`(-> String)`

Guarantee: a function

`(-> String)`

Guarantee: any value

Example 2

Untyped Codebase ... with a few Types

`(-> String)`

Guarantee: a function that returns strings

`(-> String)`

Guarantee: a function

`(-> String)`

Guarantee: any value

only `deep` is reliable everywhere + blame

Typed Racket

Strong Guarantees,
High Overhead

deep



shallow



optional

Weak Guarantees,
Low Overhead





Announced in 2020 ...

A screenshot of a presentation slide titled 'Shallow Typed Racket' with the subtitle '"same types, but weaker"'. The slide features a yellow diamond with the text 'No Chaperones!' inside. Below this, there are three colored boxes: a green box with '+ fast boundaries', '+ more expressive', and '+ simple'; a red box with '- slow @ fully-typed' and '- temporary'; and a blue box with '~~ coming soon ~~'. To the right of these boxes, the text 'RFC typed-racket/pull/952' and 'PR typed-racket/pull/948' is visible. A small video inset of a man's face is in the bottom-left corner of the slide.

Shallow Typed Racket
"same types, but weaker"

No Chaperones!

+ fast boundaries
+ more expressive
+ simple

- slow @ fully-typed
- temporary

~~ coming soon ~~
RFC typed-racket/pull/952
PR typed-racket/pull/948

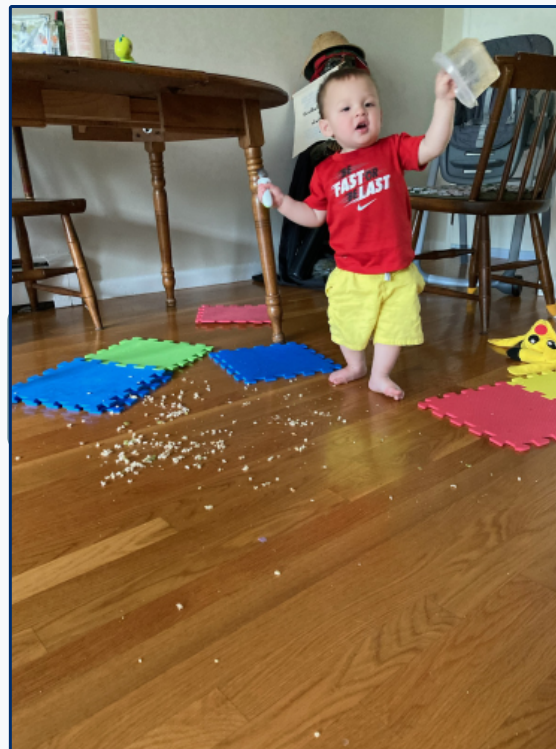


What took so long?

1. Life Stuff



1. Life Stuff



2. Well-Typed Interactions

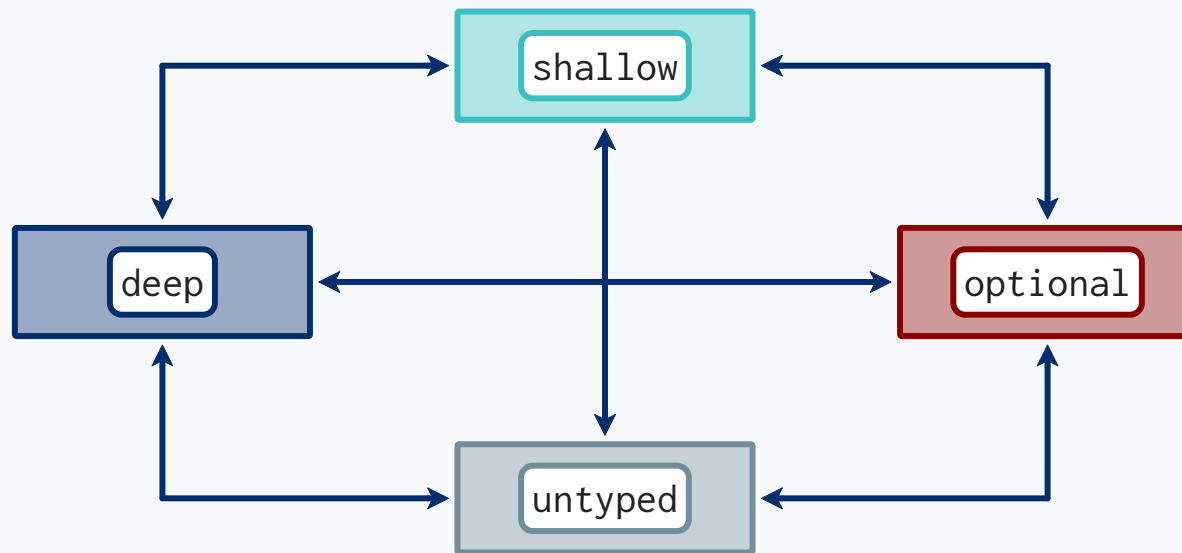
shallow

deep

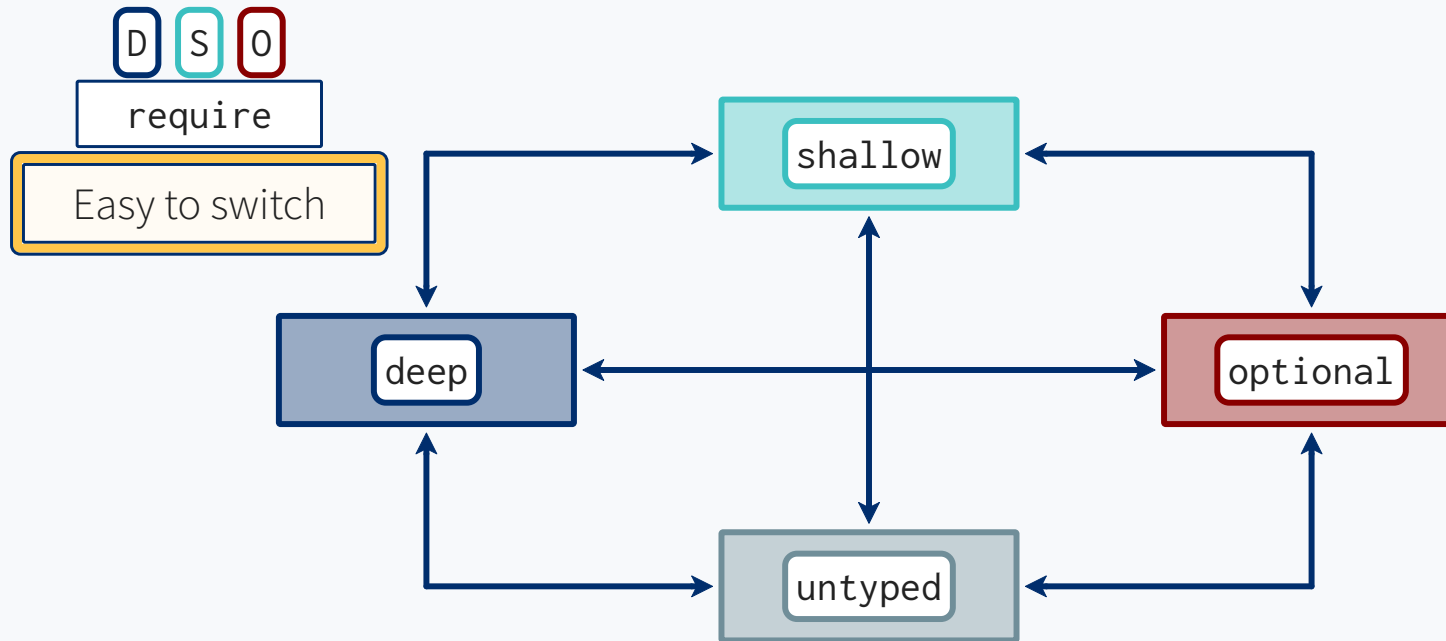
optional

untyped

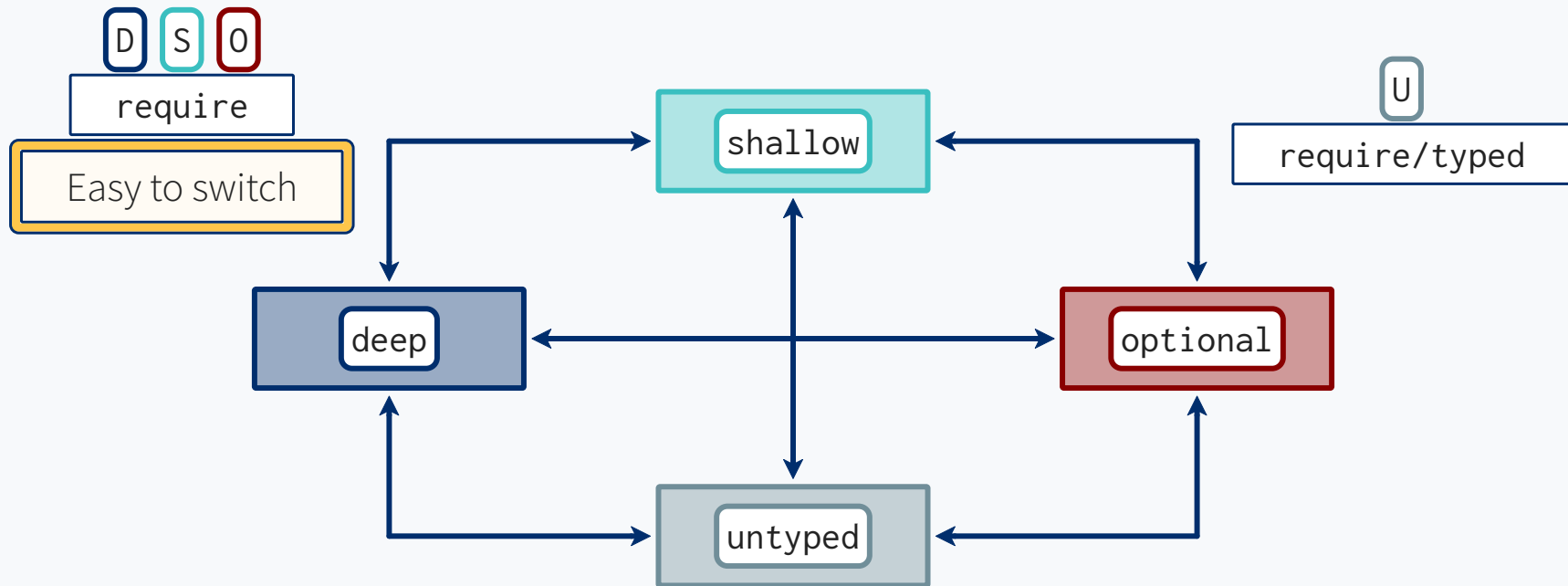
2. Well-Typed Interactions



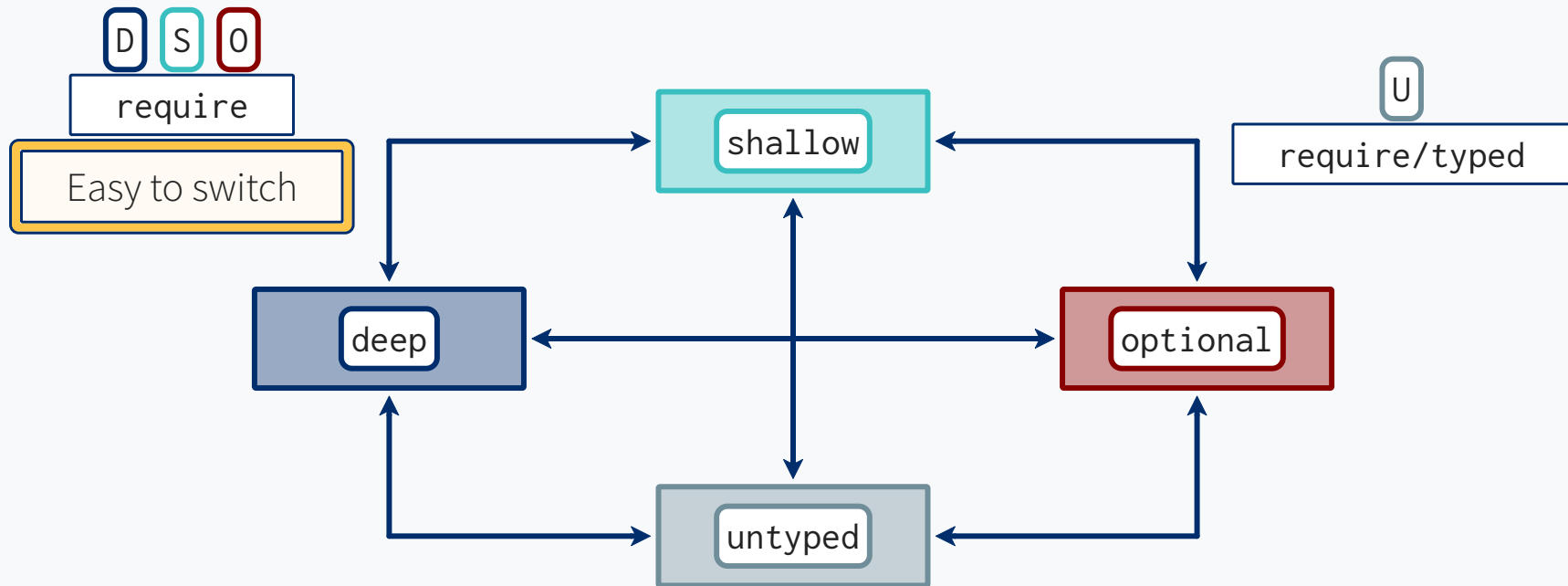
2. Well-Typed Interactions



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)) Yes check
     #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) No check
     #false]
    [(eq? 'moo (car lst)) Yes check
     #true]
    [else
     (says-moo? (cdr lst)) 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) No check
     #false]
    [(eq? 'moo (car lst)) Yes check
     #true]
    [else
     (says-moo? (cdr lst)) No check])
```

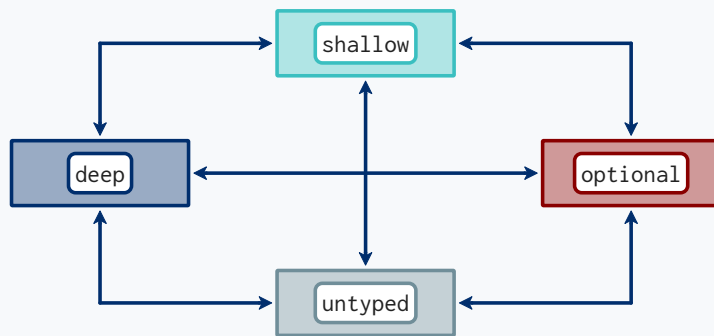
Old Solution: trust specific IDs

New Solution: type-based, compositional



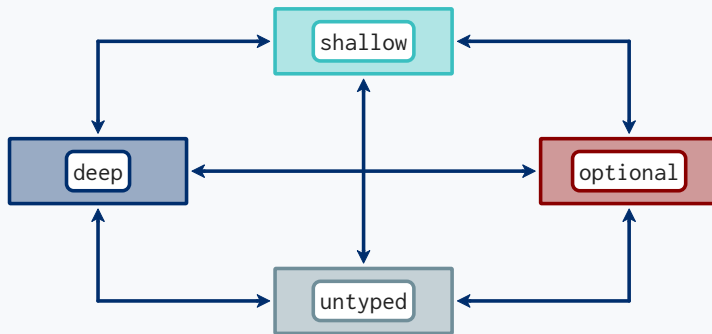
++ Well-Typed Interactions

++ Faster Shallow





- ++ Well-Typed Interactions
- ++ Faster Shallow



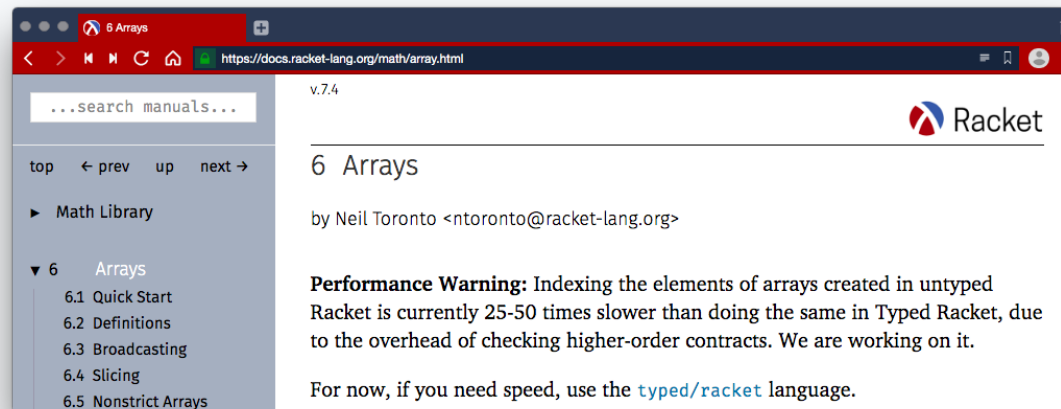
More to Come

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

Faster Math?

Faster Math?

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



The screenshot shows a web browser window with the address bar displaying `https://docs.racket-lang.org/math/array.html`. The page title is "6 Arrays" and it is authored by Neil Toronto. A prominent "Performance Warning" is displayed, stating that indexing elements in untyped Racket is 25-50 times slower than in Typed Racket. The page also includes a navigation sidebar on the left and the Racket logo in the top right corner.

...search manuals...

top ← prev up next →

► Math Library

▼ 6 Arrays

- 6.1 Quick Start
- 6.2 Definitions
- 6.3 Broadcasting
- 6.4 Slicing
- 6.5 Nonstrict Arrays

v.7.4

Racket

6 Arrays

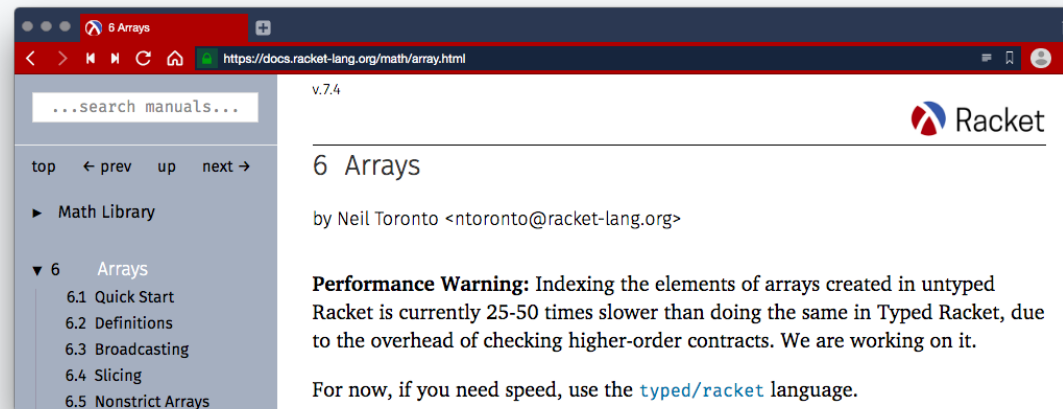
by Neil Toronto <ntoronto@racket-lang.org>

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.

Faster Math?

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



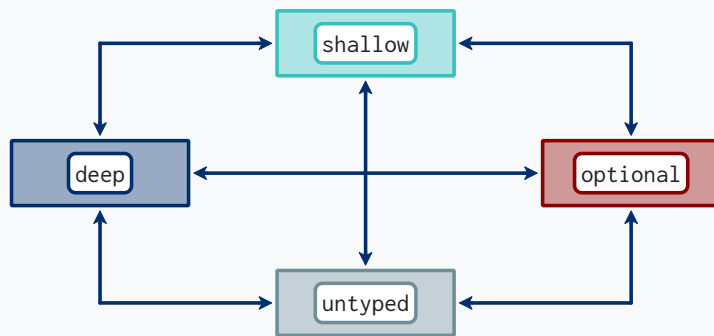
Not easy, but there's hope:

github.com/racket/math/issues/75



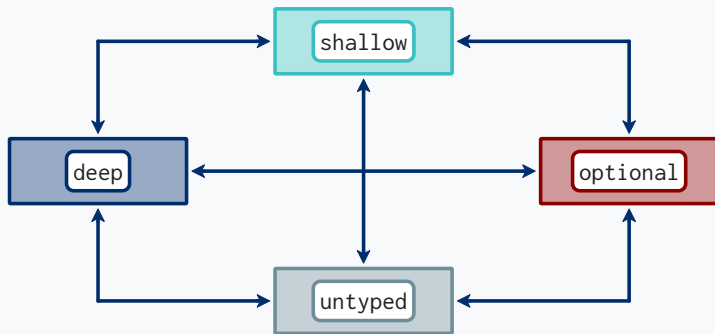
++ Well-Typed Interactions

++ Faster Shallow





- ++ 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

