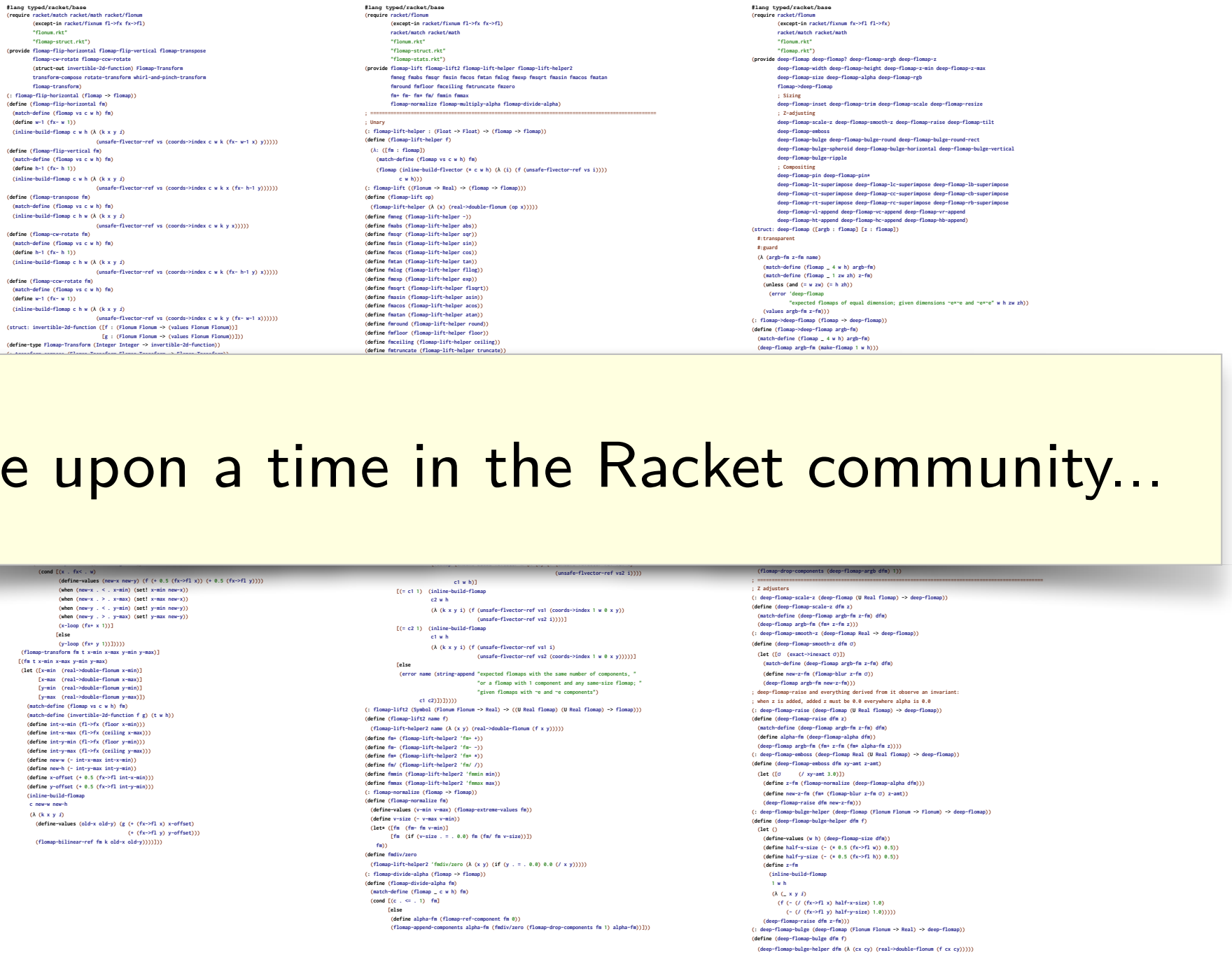
Optimization Coaching for JavaScript

Vincent St-Amour
PLT @ Northwestern University

Shu-yu Guo
Mozilla Research

ECOOP 2015 — July 9th, 2015



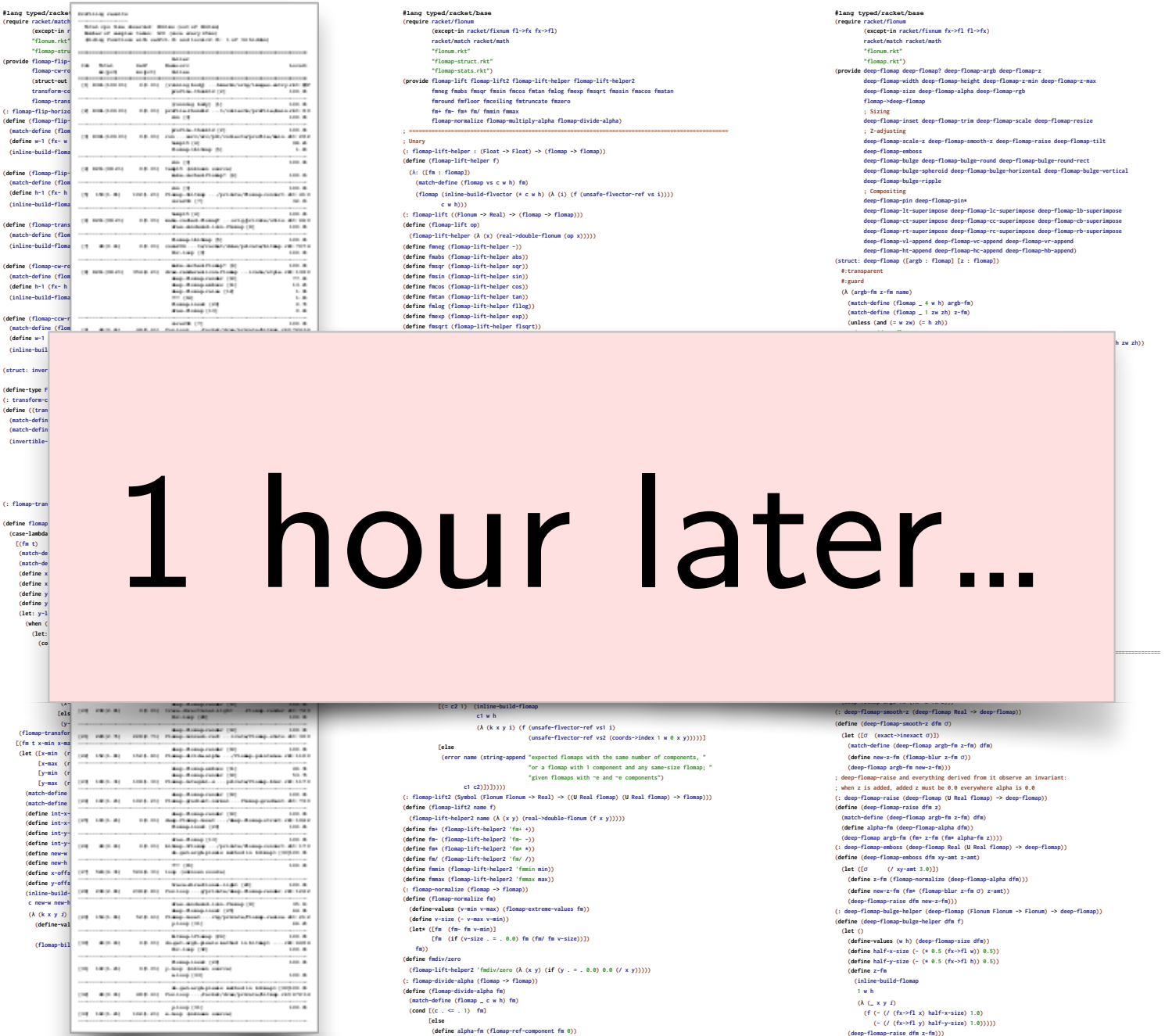
Once upon a time in the Racket community...

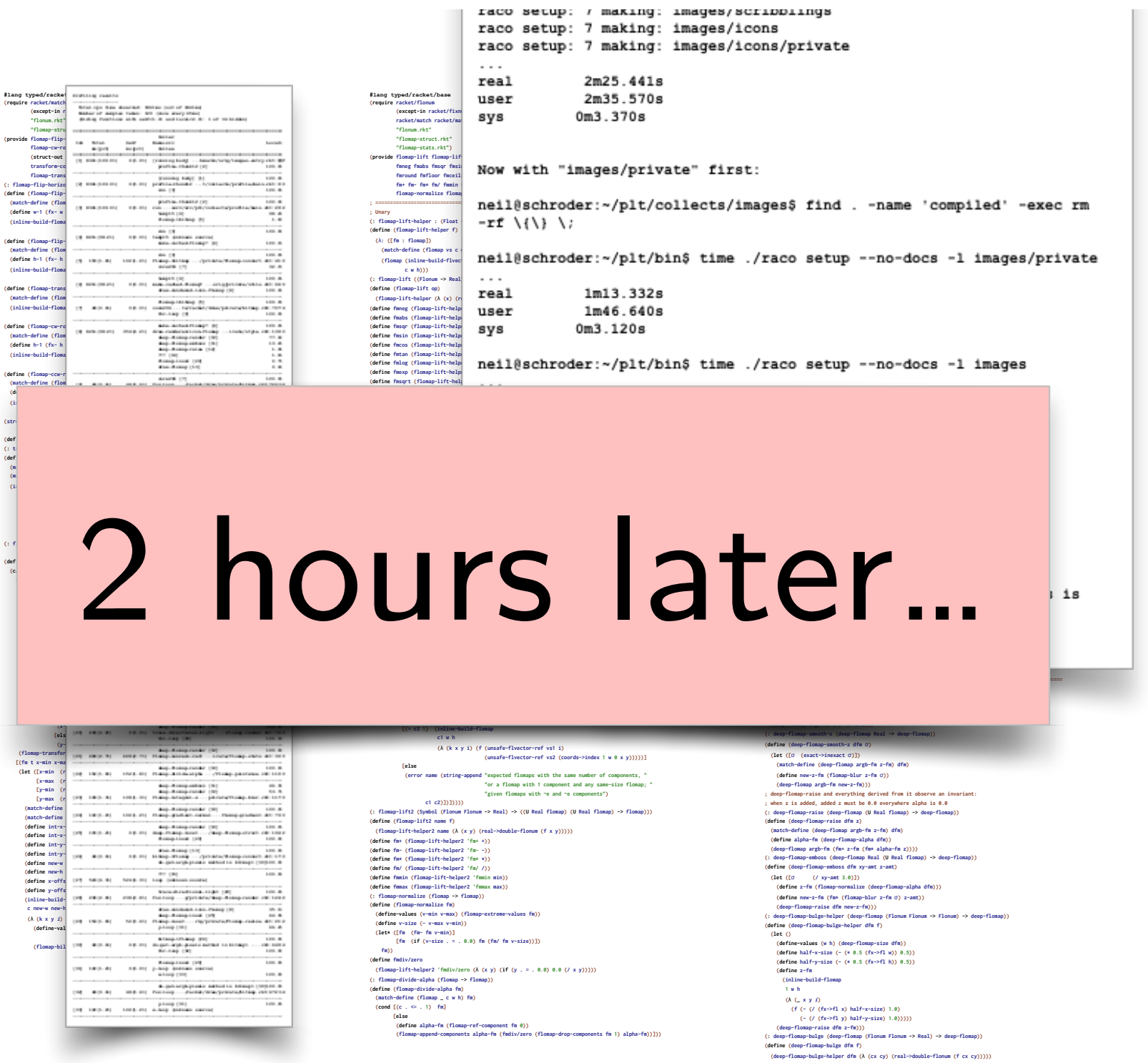

It renders text, too!

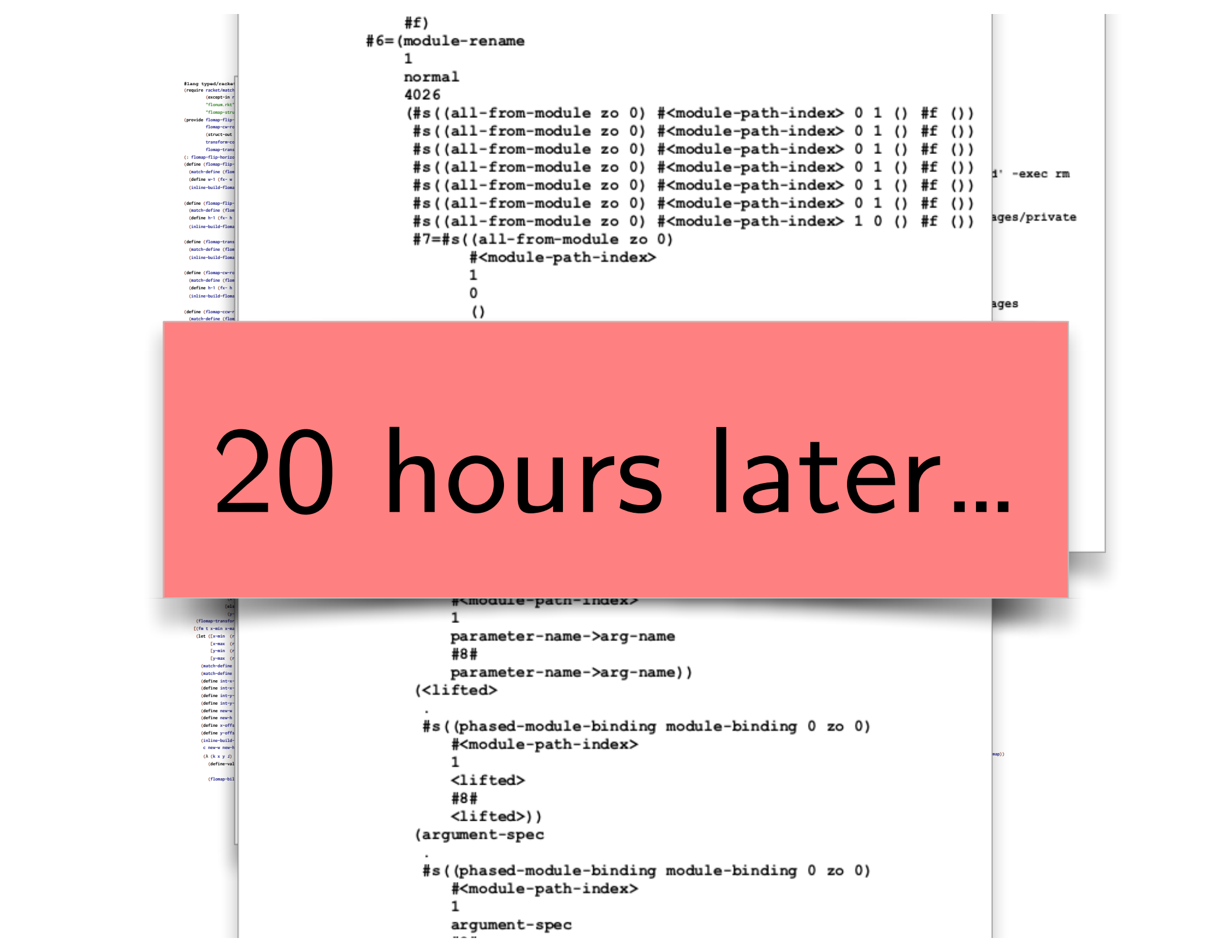
```
#lang typed/racket/base
(require racket/match racket/math racket/flonum
         (except-in racket/fixnum fl->fx fx->fl)
         "flonum.rkt"
         "flomap-struct.rkt")
(provide flomap-flip-horizontal flomap-flip-vertical flomap-transpose
         flomap-cw-rotate flomap-ccw-rotate
         (struct-out invertible-2d-function) flomap-transform
         transform-compose rotate-transform whirl-and-pinch-transform
         flomap-transform)
(: flomap-flip-horizontal (flomap -> flomap))
(define (flomap-flip-horizontal fm)
  (match-define (flomap vs c w h) fm)
  (define w-1 (fx- w 1))
  (inline-build-flomap c w h (λ (k x y i)
                               (unsafe-flvector-ref vs (coords->index c w k (fx- w-1 x) y)))))
(define (flomap-flip-vertical fm)
  (match-define (flomap vs c w h) fm)
  (define h-1 (fx- h 1))
  (inline-build-flomap c w h (λ (k x y i)
                               (unsafe-flvector-ref vs (coords->index c w k x (fx- h-1 y))))))
(define (flomap-transpose fm)
  (match-define (flomap vs c w h) fm)
  (inline-build-flomap c h w (λ (k x y i)
                               (unsafe-flvector-ref vs (coords->index c w k y x)))))
(define (flomap-cw-rotate fm)
  (match-define (flomap vs c w h) fm)
  (define h-1 (fx- h 1))
  (inline-build-flomap c h w (λ (k x y i)
                               (unsafe-flvector-ref vs (coords->index c w k (fx- h-1 y) x)))))
(define (flomap-ccw-rotate fm)
  (match-define (flomap vs c w h) fm)
  (define w-1 (fx- w 1))
  (inline-build-flomap c h w (λ (k x y i)
                               (unsafe-flvector-ref vs (coords->index c w k y (fx- w-1 x))))))
(struct: invertible-2d-function ([f : (Flonum Flonum -> (values Flonum Flonum))]
                                 [g : (Flonum Flonum -> (values Flonum Flonum))]))
(define-type Flomap-Transform (Integer Integer -> invertible-2d-function))
(: transform-compose (Flomap-Transform Flomap-Transform -> Flomap-Transform))
(define ((transform-compose t1 t2) w h)
  (match-define (invertible-2d-function f1 g1) (t1 w h))
  (match-define (invertible-2d-function f2 g2) (t2 w h))
  (invertible-2d-function (λ: ([x : Flonum] [y : Flonum])
                            (let-values ([(x y)  (f2 x y)])
                              (f1 x y)))
                          (λ: ([x : Flonum] [y : Flonum])
                            (let-values ([(x y)  (g1 x y)])
                              (g2 x y)))))
(: flomap-transform (case-> (flomap Flomap-Transform -> flomap)
                            (flomap Flomap-Transform Real Real Real Real -> flomap)))
(define flomap-transform
  (case-lambda
    [(fm t)
     (match-define (flomap vs c w h) fm)
     (match-define (invertible-2d-function f g) (t w h))
     (define x-min +inf.0)
     (define x-max -inf.0)
     (define y-min +inf.0)
     (define y-max -inf.0)
     (let: y-loop : Void ([y : Integer  0])
       (when (y . fx< . h)
         (let: x-loop : Void ([x : Integer  0])
           (cond [(x . fx< . w)
                  (define-values (new-x new-y) (f (+ 0.5 (fx->fl x)) (+ 0.5 (fx->fl y))))
                  (when (new-x . < . x-min) (set! x-min new-x))
                  (when (new-x . > . x-max) (set! x-max new-x))
                  (when (new-y . < . y-min) (set! y-min new-y))
                  (when (new-y . > . y-max) (set! y-max new-y))
                  (x-loop (fx+ x 1))]
                 [else
                  (y-loop (fx+ y 1))]))))
     (flomap-transform fm t x-min x-max y-min y-max)]
    [(fm t x-min x-max y-min y-max)
     (let ([x-min  (real->double-flonum x-min)]
           [x-max  (real->double-flonum x-max)]
           [y-min  (real->double-flonum y-min)]
           [y-max  (real->double-flonum y-max)])
       (match-define (flomap vs c w h) fm)
       (match-define (invertible-2d-function f g) (t w h))
       (define int-x-min (fl->fx (floor x-min)))
       (define int-x-max (fl->fx (ceiling x-max)))
       (define int-y-min (fl->fx (floor y-min)))
       (define int-y-max (fl->fx (ceiling y-max)))
       (define new-w (- int-x-max int-x-min))
       (define new-h (- int-y-max int-y-min))
       (define x-offset (+ 0.5 (fx->fl int-x-min)))
       (define y-offset (+ 0.5 (fx->fl int-y-min)))
       (inline-build-flomap
        c new-w new-h
        (λ (k x y i)
          (define-values (old-x old-y) (g (+ (fx->fl x) x-offset)
                                          (+ (fx->fl y) y-offset)))
          (flomap-bilinear-ref fm k old-x old-y))))]))
```

```
#lang typed/racket/base
(require racket/flonum
         (except-in racket/fixnum fl->fx fx->fl)
         racket/match racket/math
         "flonum.rkt"
         "flomap-struct.rkt"
         "flomap-stats.rkt")
(provide flomap-lift flomap-lift2 flomap-lift-helper flomap-lift-helper2
         fmneg fmabs fmsqr fmsin fmcos fmtan fmlog fmexp fmsqrt fmasin fmacos fmatan
         fmround fmfloor fmceiling fmtruncate fmzero
         fm+ fm- fm* fm/ fmmin fmmax
         flomap-normalize flomap-multiply-alpha flomap-divide-alpha)
; =============================================================================
; Unary
(: flomap-lift-helper : (Float -> Float) -> (flomap -> flomap))
(define (flomap-lift-helper f)
  (λ: ([fm : flomap])
    (match-define (flomap vs c w h) fm)
    (flomap (inline-build-flvector (* c w h) (λ (i) (f (unsafe-flvector-ref vs i))))
            c w h)))
(: flomap-lift ((Flonum -> Real) -> (flomap -> flomap)))
(define (flomap-lift op)
  (flomap-lift-helper (λ (x) (real->double-flonum (op x)))))
(define fmneg (flomap-lift-helper -))
(define fmabs (flomap-lift-helper abs))
(define fmsqr (flomap-lift-helper sqr))
(define fmsin (flomap-lift-helper sin))
(define fmcos (flomap-lift-helper cos))
(define fmtan (flomap-lift-helper tan))
(define fmlog (flomap-lift-helper fllog))
(define fmexp (flomap-lift-helper exp))
(define fmsqrt (flomap-lift-helper flsqrt))
(define fmasin (flomap-lift-helper asin))
(define fmacos (flomap-lift-helper acos))
(define fmatan (flomap-lift-helper atan))
(define fmround (flomap-lift-helper round))
(define fmfloor (flomap-lift-helper floor))
(define fmceiling (flomap-lift-helper ceiling))
(define fmtruncate (flomap-lift-helper truncate))
(define fmzero (flomap-lift-helper (λ (x) (if (x . = . 0.0) 1.0 0.0))))
; ;; =============================================================================
; Bi
(: [illegible]2 : Symbol (Float [illegible] (U Real [illegible])) [illegible])
(define ([illegible]-helper2 name f)
  (λ (fm1 fm2)
    (cond [([illegible]real flomap)] [fm[illegible]])
          [(real? fm1) (real? fm[illegible])
           (error name "expected at l[illegible]ment;[illegible]1 fm2)]
          [(let ([fm1  (r[illegible]m1)])
             ([illegible]-lift[illegible]m1 v)
           (let ([fm2  (r[illegible]m2)])
             ([illegible]-lift[illegible]fm2))
          [else
           (define vs1 (flomap vs1 c1
           (define vs2 (flomap vs2 c2
           (unless (and (= w w2) (= h h2
             (error name "expected same[illegible]size[illegible]w h w2 h2)
             (define [illegible]s c1
             (define [illegible]vs (
             (inline-bu[illegible]ctor-ref vs1 i)
                                       [illegible]lvector-ref vs2 i)
                                       h)]
          [(= c1 1)  (inline-build-flomap
                      c2 w h
                      (λ (k x y i) (f (unsafe-flvector-ref vs1 (coords->index 1 w 0 x y))
                                      (unsafe-flvector-ref vs2 i))))]
          [(= c2 1)  (inline-build-flomap
                      c1 w h
                      (λ (k x y i) (f (unsafe-flvector-ref vs1 i)
                                      (unsafe-flvector-ref vs2 (coords->index 1 w 0 x y)))))]
          [else
           (error name (string-append "expected flomaps with the same number of components, "
                                      "or a flomap with 1 component and any same-size flomap; "
                                      "given flomaps with ~e and ~e components")
                  c1 c2)])))))
(: flomap-lift2 (Symbol (Flonum Flonum -> Real) -> ((U Real flomap) (U Real flomap) -> flomap)))
(define (flomap-lift2 name f)
  (flomap-lift-helper2 name (λ (x y) (real->double-flonum (f x y)))))
(define fm+ (flomap-lift-helper2 'fm+ +))
(define fm- (flomap-lift-helper2 'fm- -))
(define fm* (flomap-lift-helper2 'fm* *))
(define fm/ (flomap-lift-helper2 'fm/ /))
(define fmmin (flomap-lift-helper2 'fmmin min))
(define fmmax (flomap-lift-helper2 'fmmax max))
(: flomap-normalize (flomap -> flomap))
(define (flomap-normalize fm)
  (define-values (v-min v-max) (flomap-extreme-values fm))
  (define v-size (- v-max v-min))
  (let* ([fm  (fm- fm v-min)]
         [fm  (if (v-size . = . 0.0) fm (fm/ fm v-size))])
    fm))
(define fmdiv/zero
  (flomap-lift-helper2 'fmdiv/zero (λ (x y) (if (y . = . 0.0) 0.0 (/ x y)))))
(: flomap-divide-alpha (flomap -> flomap))
(define (flomap-divide-alpha fm)
  (match-define (flomap _ c w h) fm)
  (cond [(c . <= . 1)  fm]
        [else
         (define alpha-fm (flomap-ref-component fm 0))
         (flomap-append-components alpha-fm (fmdiv/zero (flomap-drop-components fm 1) alpha-fm))]))
```

```
#lang typed/racket/base
(require racket/flonum
         (except-in racket/fixnum fx->fl fl->fx)
         racket/match racket/math
         "flonum.rkt"
         "flomap.rkt")
(provide deep-flomap deep-flomap? deep-flomap-argb deep-flomap-z
         deep-flomap-width deep-flomap-height deep-flomap-z-min deep-flomap-z-max
         deep-flomap-size deep-flomap-alpha deep-flomap-rgb
         flomap->deep-flomap
         ; Sizing
         deep-flomap-inset deep-flomap-trim deep-flomap-scale deep-flomap-resize
         ; Z-adjusting
         deep-flomap-scale-z deep-flomap-smooth-z deep-flomap-raise deep-flomap-tilt
         deep-flomap-emboss
         deep-flomap-bulge deep-flomap-bulge-round deep-flomap-bulge-round-rect
         deep-flomap-bulge-spheroid deep-flomap-bulge-horizontal deep-flomap-bulge-vertical
         deep-flomap-bulge-ripple
         ; Compositing
         deep-flomap-pin deep-flomap-pin*
         deep-flomap-lt-superimpose deep-flomap-lc-superimpose deep-flomap-lb-superimpose
         deep-flomap-ct-superimpose deep-flomap-cc-superimpose deep-flomap-cb-superimpose
         deep-flomap-rt-superimpose deep-flomap-rc-superimpose deep-flomap-rb-superimpose
         deep-flomap-vl-append deep-flomap-vc-append deep-flomap-vr-append
         deep-flomap-ht-append deep-flomap-hc-append deep-flomap-hb-append)
(struct: deep-flomap ([argb : flomap] [z : flomap])
  #:transparent
  #:guard
  (λ (argb-fm z-fm name)
    (match-define (flomap _ 4 w h) argb-fm)
    (match-define (flomap _ 1 zw zh) z-fm)
    (unless (and (= w zw) (= h zh))
      (error 'deep-flomap
             "expected flomaps of equal dimension; given dimensions ~ex~e and ~ex~e" w h zw zh))
    (values argb-fm z-fm)))
(: flomap->deep-flomap (flomap -> deep-flomap))
(define (flomap->deep-flomap argb-fm)
  (match-define (flomap _ 4 w h) argb-fm)
  (deep-flomap argb-fm (make-flomap 1 w h)))
(: deep-flomap-width (deep-flomap -> Nonnegative-Fixnum))
(define (deep-flomap-width dfm)
  (flomap-width (deep-flomap-argb dfm)))
(: deep-flomap-height (deep-flomap -> Nonnegative-Fixnum))
(define (deep-flomap-height dfm)
  (flomap-height (deep-flomap-argb dfm)))
(: deep-flomap-size (deep-flomap -> (values Nonnegative-Fixnum Nonnegative-Fixnum)))
(define (deep-flomap-size dfm)
  (flomap-size (deep-flomap-argb dfm)))
(: deep-flomap-alpha (deep-flomap -> flomap))
(define (deep-flomap-alpha dfm)
  (flomap-component (deep-flomap-argb dfm) 0))
(: deep-flomap-rgb (deep-flomap -> flomap))
(define (deep-flomap-rgb dfm)
  (flomap-drop-components (deep-flomap-argb dfm) 1))
; =============================================================================
; Z adjusters
(: deep-flomap-scale-z (deep-flomap (U Real flomap) -> deep-flomap))
(define (deep-flomap-scale-z dfm z)
  (match-define (deep-flomap argb-fm z-fm) dfm)
  (deep-flomap argb-fm (fm* z-fm z)))
(: deep-flomap-smooth-z (deep-flomap Real -> deep-flomap))
(define (deep-flomap-smooth-z dfm σ)
  (let ([σ  (exact->inexact σ)])
    (match-define (deep-flomap argb-fm z-fm) dfm)
    (define new-z-fm (flomap-blur z-fm σ))
    (deep-flomap argb-fm new-z-fm)))
; deep-flomap-raise and everything derived from it observe an invariant:
; when z is added, added z must be 0.0 everywhere alpha is 0.0
(: deep-flomap-raise (deep-flomap (U Real flomap) -> deep-flomap))
(define (deep-flomap-raise dfm z)
  (match-define (deep-flomap argb-fm z-fm) dfm)
  (define alpha-fm (deep-flomap-alpha dfm))
  (deep-flomap argb-fm (fm+ z-fm (fm* alpha-fm z))))
(: deep-flomap-emboss (deep-flomap Real (U Real flomap) -> deep-flomap))
(define (deep-flomap-emboss dfm xy-amt z-amt)
  (let ([σ       (/ xy-amt 3.0)])
    (define z-fm (flomap-normalize (deep-flomap-alpha dfm)))
    (define new-z-fm (fm* (flomap-blur z-fm σ) z-amt))
    (deep-flomap-raise dfm new-z-fm)))
(: deep-flomap-bulge-helper (deep-flomap (Flonum Flonum -> Flonum) -> deep-flomap))
(define (deep-flomap-bulge-helper dfm f)
  (let ()
    (define-values (w h) (deep-flomap-size dfm))
    (define half-x-size (- (* 0.5 (fx->fl w)) 0.5))
    (define half-y-size (- (* 0.5 (fx->fl h)) 0.5))
    (define z-fm
      (inline-build-flomap
       1 w h
       (λ (_ x y i)
         (f (- (/ (fx->fl x) half-x-size) 1.0)
            (- (/ (fx->fl y) half-y-size) 1.0)))))
    (deep-flomap-raise dfm z-fm)))
(: deep-flomap-bulge (deep-flomap (Flonum Flonum -> Real) -> deep-flomap))
(define (deep-flomap-bulge dfm f)
  (deep-flomap-bulge-helper dfm (λ (cx cy) (real->double-flonum (f cx cy)))))
```


1 hour later…

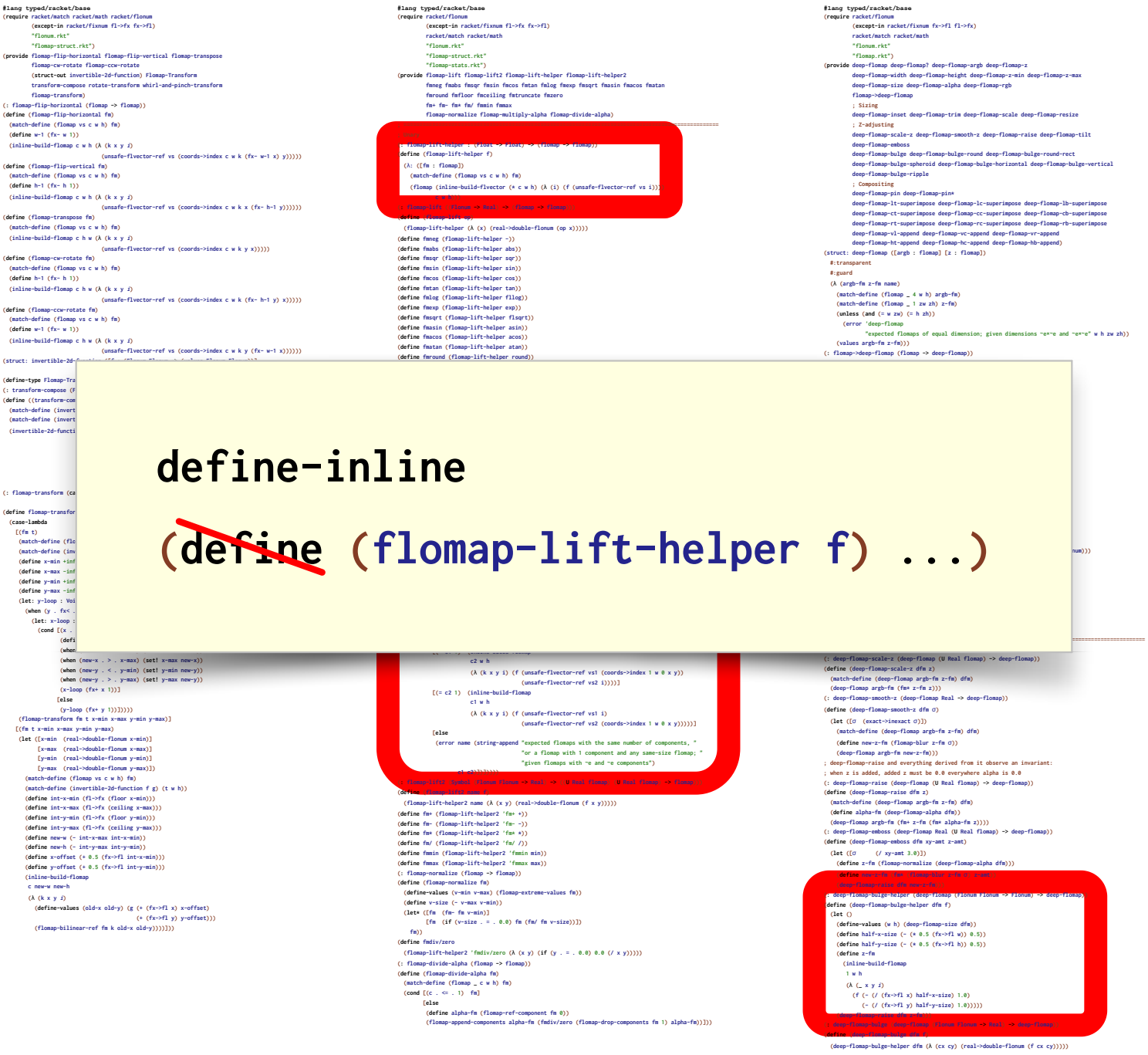

raco setup: 7 making: images/scribblings
raco setup: 7 making: images/icons
raco setup: 7 making: images/icons/private
...
real 2m25.441s
user 2m35.570s
sys 0m3.370s
Now with "images/private" first:
neil@schroder:~/plt/collects/images$ find . -name 'compiled' -exec rm -rf \{\} \;
neil@schroder:~/plt/bin$ time ./raco setup --no-docs -l images/private
...
real 1m13.332s
user 1m46.640s
sys 0m3.120s
neil@schroder:~/plt/bin$ time ./raco setup --no-docs -l images
2 hours later...

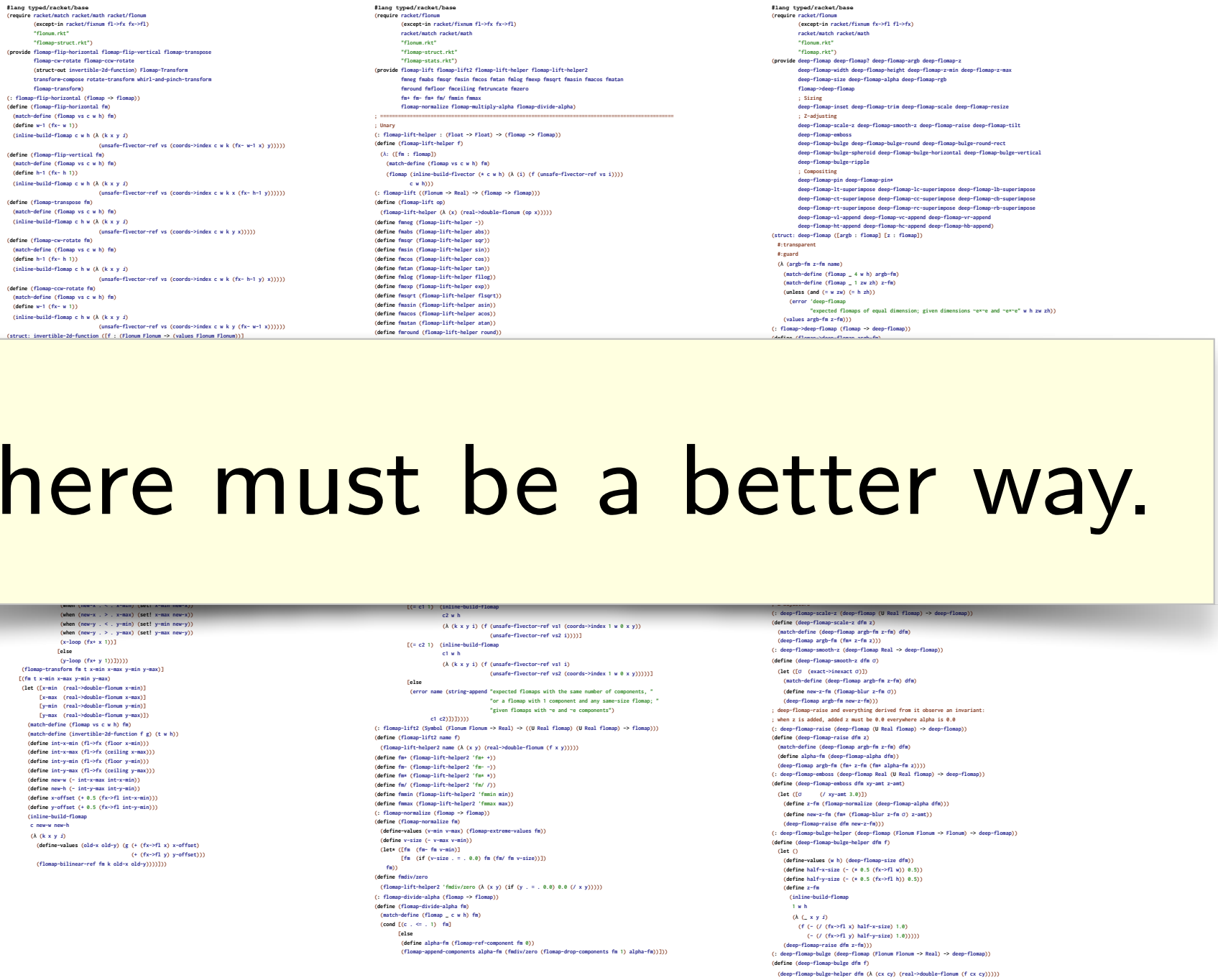

#f)
#6=(module-rename
1
normal
4026
(#s((all-from-module zo 0) #<module-path-index> 0 1 () #f ())
#s((all-from-module zo 0) #<module-path-index> 0 1 () #f ())
#s((all-from-module zo 0) #<module-path-index> 0 1 () #f ())
#s((all-from-module zo 0) #<module-path-index> 0 1 () #f ())
#s((all-from-module zo 0) #<module-path-index> 0 1 () #f ())
#s((all-from-module zo 0) #<module-path-index> 0 1 () #f ())
#s((all-from-module zo 0) #<module-path-index> 1 0 () #f ())
#7=#s((all-from-module zo 0)
#<module-path-index>
1
0
()
20 hours later...
#<module-path-index>
1
parameter-name->arg-name
#8#
parameter-name->arg-name))
(<lifted>
.
#s((phased-module-binding module-binding 0 zo 0)
#<module-path-index>
1
<lifted>
#8#
<lifted>))
(argument-spec
.
#s((phased-module-binding module-binding 0 zo 0)
#<module-path-index>
1
argument-spec

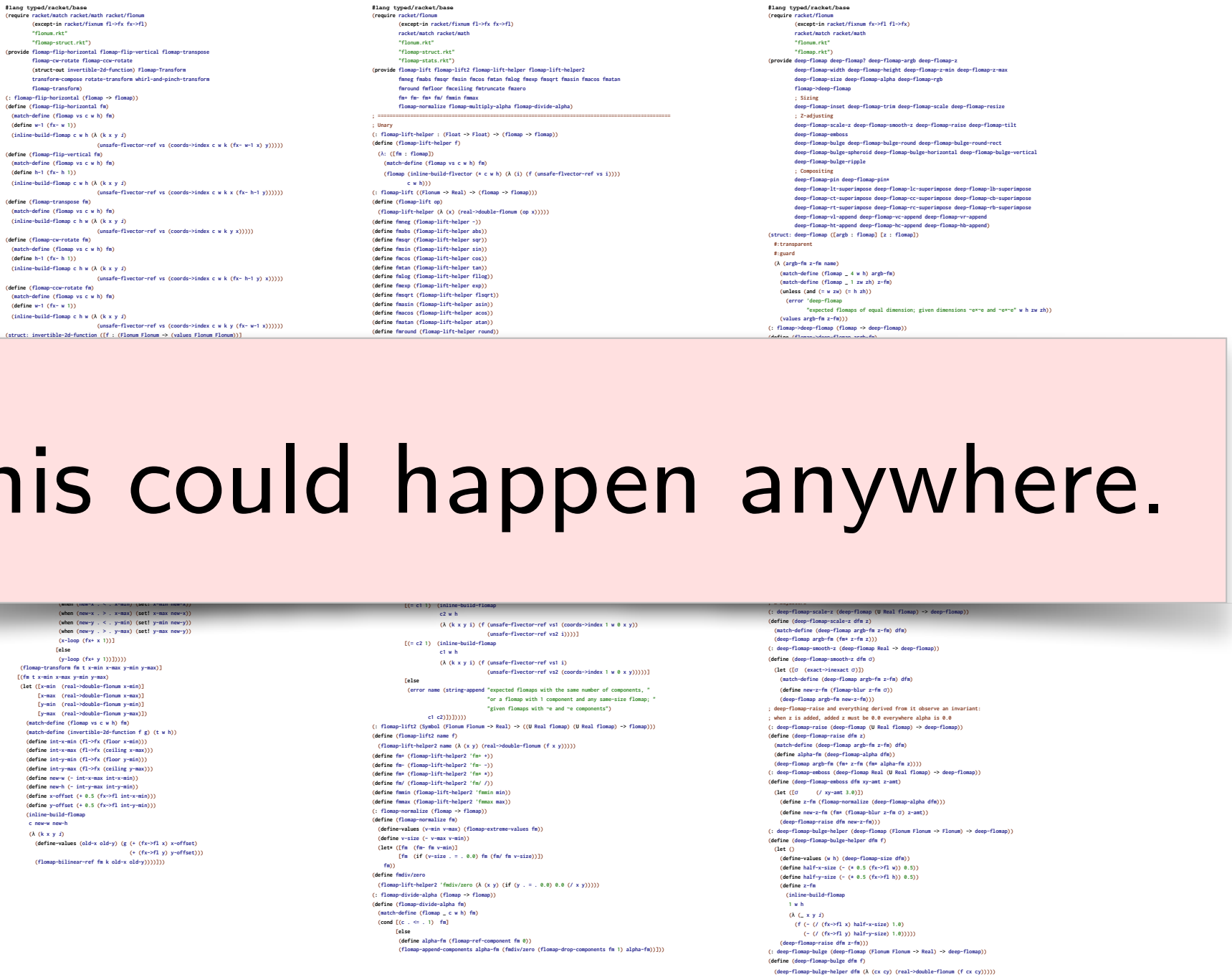
```
#lang typed/racket/base
(require racket/match racket/math racket/flonum
         (except-in racket/fixnum fl->fx fx->fl)
         "flonum.rkt"
         "flomap-struct.rkt")
(provide flomap-flip-horizontal flomap-flip-vertical flomap-transpose
         flomap-cw-rotate flomap-ccw-rotate
         (struct-out invertible-2d-function) flomap-transform
         transform-compose rotate-transform whirl-and-pinch-transform
         flomap-transform)
(: flomap-flip-horizontal (flomap -> flomap))
(define (flomap-flip-horizontal fm)
  (match-define (flomap vs c w h) fm)
  (define w-1 (fx- w 1))
  (inline-build-flomap c w h (λ (k x y i)
                               (unsafe-flvector-ref vs (coords->index c w k (fx- w-1 x) y)))))
(define (flomap-flip-vertical fm)
  (match-define (flomap vs c w h) fm)
  (define h-1 (fx- h 1))
  (inline-build-flomap c w h (λ (k x y i)
                               (unsafe-flvector-ref vs (coords->index c w k x (fx- h-1 y))))))
(define (flomap-transpose fm)
  (match-define (flomap vs c w h) fm)
  (inline-build-flomap c h w (λ (k x y i)
                               (unsafe-flvector-ref vs (coords->index c w k y x)))))
(define (flomap-cw-rotate fm)
  (match-define (flomap vs c w h) fm)
  (define h-1 (fx- h 1))
  (inline-build-flomap c h w (λ (k x y i)
                               (unsafe-flvector-ref vs (coords->index c w k (fx- h-1 y) x)))))
(define (flomap-ccw-rotate fm)
  (match-define (flomap vs c w h) fm)
  (define w-1 (fx- w 1))
  (inline-build-flomap c h w (λ (k x y i)
                               (unsafe-flvector-ref vs (coords->index c w k y (fx- w-1 x))))))
(struct: invertible-2d-function ([f : (Flonum Flonum -> (values Flonum Flonum))]
                                 [g : (Flonum Flonum -> (values Flonum Flonum))]))
(define-type Flomap-Transform (Integer Integer -> invertible-2d-function))
(: transform-compose (Flomap-Transform Flomap-Transform -> Flomap-Transform))
(define ((transform-compose t1 t2) w h)
  (match-define (invertible-2d-function f1 g1) (t1 w h))
  (match-define (invertible-2d-function f2 g2) (t2 w h))
  (invertible-2d-function (λ: ([x : Flonum] [y : Flonum])
                            (let-values ([(x y)  (f2 x y)])
                              (f1 x y)))
                          (λ: ([x : Flonum] [y : Flonum])
                            (let-values ([(x y)  (g1 x y)])
                              (g2 x y)))))
(: flomap-transform (case-> (flomap Flomap-Transform -> flomap)
                            (flomap Flomap-Transform Real Real Real Real -> flomap)))
(define flomap-transform
  (case-lambda
    [(fm t)
     (match-define (flomap vs c w h) fm)
     (match-define (invertible-2d-function f g) (t w h))
     (define x-min +inf.0)
     (define x-max -inf.0)
     (define y-min +inf.0)
     (define y-max -inf.0)
     (let: y-loop : Void ([y : Integer  0])
       (when (y . fx< . h)
         (let: x-loop : Void ([x : Integer  0])
           (cond [(x . fx< . w)
                  (define-values (new-x new-y) (f (+ 0.5 (fx->fl x)) (+ 0.5 (fx->fl y))))
                  (when (new-x . < . x-min) (set! x-min new-x))
                  (when (new-x . > . x-max) (set! x-max new-x))
                  (when (new-y . < . y-min) (set! y-min new-y))
                  (when (new-y . > . y-max) (set! y-max new-y))
                  (x-loop (fx+ x 1))]
                 [else
                  (y-loop (fx+ y 1))]))))
     (flomap-transform fm t x-min x-max y-min y-max)]
    [(fm t x-min x-max y-min y-max)
     (let ([x-min  (real->double-flonum x-min)]
           [x-max  (real->double-flonum x-max)]
           [y-min  (real->double-flonum y-min)]
           [y-max  (real->double-flonum y-max)])
       (match-define (flomap vs c w h) fm)
       (match-define (invertible-2d-function f g) (t w h))
       (define int-x-min (fl->fx (floor x-min)))
       (define int-x-max (fl->fx (ceiling x-max)))
       (define int-y-min (fl->fx (floor y-min)))
       (define int-y-max (fl->fx (ceiling y-max)))
       (define new-w (- int-x-max int-x-min))
       (define new-h (- int-y-max int-y-min))
       (define x-offset (+ 0.5 (fx->fl int-x-min)))
       (define y-offset (+ 0.5 (fx->fl int-y-min)))
       (inline-build-flomap
        c new-w new-h
        (λ (k x y i)
          (define-values (old-x old-y) (g (+ (fx->fl x) x-offset)
                                          (+ (fx->fl y) y-offset)))
          (flomap-bilinear-ref fm k old-x old-y))))]))
```

```
#lang typed/racket/base
(require racket/flonum
         (except-in racket/fixnum fl->fx fx->fl)
         racket/match racket/math
         "flonum.rkt"
         "flomap-struct.rkt"
         "flomap-stats.rkt")
(provide flomap-lift flomap-lift2 flomap-lift-helper flomap-lift-helper2
         fmneg fmabs fmsqr fmsin fmcos fmtan fmlog fmexp fmsqrt fmasin fmacos fmatan
         fmround fmfloor fmceiling fmtruncate fmzero
         fm+ fm- fm* fm/ fmmin fmmax
         flomap-normalize flomap-multiply-alpha flomap-divide-alpha)
;; ===================================================================================================
;; Lifting
(: flomap-lift-helper : (Float -> Float) -> (flomap -> flomap))
(define (flomap-lift-helper f)
  (λ: ([fm : flomap])
    (match-define (flomap vs c w h) fm)
    (flomap (inline-build-flvector (* c w h) (λ (i) (f (unsafe-flvector-ref vs i))))
            c w h)))
(: flomap-lift : Flonum -> Real -> (flomap -> flomap))
(define (flomap-lift sp)
  (flomap-lift-helper (λ (x) (real->double-flonum (op x)))))
(define fmneg (flomap-lift-helper -))
(define fmabs (flomap-lift-helper abs))
(define fmsqr (flomap-lift-helper sqr))
(define fmsin (flomap-lift-helper sin))
(define fmcos (flomap-lift-helper cos))
(define fmtan (flomap-lift-helper tan))
(define fmlog (flomap-lift-helper fllog))
(define fmexp (flomap-lift-helper exp))
(define fmsqrt (flomap-lift-helper flsqrt))
(define fmasin (flomap-lift-helper asin))
(define fmacos (flomap-lift-helper acos))
(define fmatan (flomap-lift-helper atan))
(define fmround (flomap-lift-helper round))
(define fmfloor (flomap-lift-helper floor))
(define fmceiling (flomap-lift-helper ceiling))
(define fmtruncate (flomap-lift-helper truncate))
(define fmzero (flomap-lift-helper (λ (x) (if (x . = . 0.0) 1.0 0.0))))
;; ===================================================================================================
;; Binary operations
(: flomap-lift-helper2 : Symbol (Float Float -> Float) -> ((U Real flomap) (U Real flomap) -> flomap))
(define (flomap-lift-helper2 name f)
  (let: ()
    (λ: ([fm1 : (U Real flomap)] [fm2 : (U Real flomap)])
      (cond
        [(and (real? fm1) (real? fm2))
         (error name "expected at least one flomap argument; given ~e and ~e" fm1 fm2)]
        [(real? fm1)  (let ([fm1  (real->double-flonum fm1)])
                        ((flomap-lift-helper (λ (v) (f fm1 v))) fm2))]
        [(real? fm2)  (let ([fm2  (real->double-flonum fm2)])
                        ((flomap-lift-helper (λ (v) (f v fm2))) fm1))]
        [else
         (match-define (flomap vs1 c1 w h) fm1)
         (match-define (flomap vs2 c2 w2 h2) fm2)
         (cond
           [(not (and (= w w2) (= h h2)))
            (error name "expected same-size flomaps; given sizes ~e×~e and ~e×~e" w h w2 h2)]
           [(= c1 c2)  (define n (* c1 w h))
                       (define res-vs (make-flvector n))
                       (flomap (inline-build-flvector n (λ (i) (f (unsafe-flvector-ref vs1 i)
                                                                  (unsafe-flvector-ref vs2 i))))
                               c1 w h)]
           [(= c1 1)  (inline-build-flomap
                       c2 w h
                       (λ (k x y i) (f (unsafe-flvector-ref vs1 (coords->index 1 w 0 x y))
                                       (unsafe-flvector-ref vs2 i))))]
           [(= c2 1)  (inline-build-flomap
                       c1 w h
                       (λ (k x y i) (f (unsafe-flvector-ref vs1 i)
                                       (unsafe-flvector-ref vs2 (coords->index 1 w 0 x y)))))]
           [else
            (error name (string-append "expected flomaps with the same number of components, "
                                       "or a flomap with 1 component and any same-size flomap; "
                                       "given flomaps with ~e and ~e components")
                   c1 c2)])])))
(: flomap-lift2 : Symbol Flonum Flonum -> Real -> (U Real flomap) (U Real flomap) -> flomap)
(define (flomap-lift2 name sp)
  (flomap-lift-helper2 name (λ (x y) (real->double-flonum (f x y)))))
(define fm+ (flomap-lift-helper2 'fm+ +))
(define fm- (flomap-lift-helper2 'fm- -))
(define fm* (flomap-lift-helper2 'fm* *))
(define fm/ (flomap-lift-helper2 'fm/ /))
(define fmmin (flomap-lift-helper2 'fmmin min))
(define fmmax (flomap-lift-helper2 'fmmax max))
(: flomap-normalize (flomap -> flomap))
(define (flomap-normalize fm)
  (define-values (v-min v-max) (flomap-extreme-values fm))
  (define v-size (- v-max v-min))
  (let* ([fm  (fm- fm v-min)]
         [fm  (if (v-size . = . 0.0) fm (fm/ fm v-size))])
    fm))
(define fmdiv/zero
  (flomap-lift-helper2 'fmdiv/zero (λ (x y) (if (y . = . 0.0) 0.0 (/ x y)))))
(: flomap-divide-alpha (flomap -> flomap))
(define (flomap-divide-alpha fm)
  (match-define (flomap _ c w h) fm)
  (cond [(c . <= . 1)  fm]
        [else
         (define alpha-fm (flomap-ref-component fm 0))
         (flomap-append-components alpha-fm (fmdiv/zero (flomap-drop-components fm 1) alpha-fm))]))
```

```
#lang typed/racket/base
(require racket/flonum
         (except-in racket/fixnum fx->fl fl->fx)
         racket/match racket/math
         "flonum.rkt"
         "flomap.rkt")
(provide deep-flomap deep-flomap? deep-flomap-argb deep-flomap-z
         deep-flomap-width deep-flomap-height deep-flomap-z-min deep-flomap-z-max
         deep-flomap-size deep-flomap-alpha deep-flomap-rgb
         flomap->deep-flomap
         ; Sizing
         deep-flomap-inset deep-flomap-trim deep-flomap-scale deep-flomap-resize
         ; Z-adjusting
         deep-flomap-scale-z deep-flomap-smooth-z deep-flomap-raise deep-flomap-tilt
         deep-flomap-emboss
         deep-flomap-bulge deep-flomap-bulge-round deep-flomap-bulge-round-rect
         deep-flomap-bulge-spheroid deep-flomap-bulge-horizontal deep-flomap-bulge-vertical
         deep-flomap-bulge-ripple
         ; Compositing
         deep-flomap-pin deep-flomap-pin*
         deep-flomap-lt-superimpose deep-flomap-lc-superimpose deep-flomap-lb-superimpose
         deep-flomap-ct-superimpose deep-flomap-cc-superimpose deep-flomap-cb-superimpose
         deep-flomap-rt-superimpose deep-flomap-rc-superimpose deep-flomap-rb-superimpose
         deep-flomap-vl-append deep-flomap-vc-append deep-flomap-vr-append
         deep-flomap-ht-append deep-flomap-hc-append deep-flomap-hb-append)
(struct: deep-flomap ([argb : flomap] [z : flomap])
  #:transparent
  #:guard
  (λ (argb-fm z-fm name)
    (match-define (flomap _ 4 w h) argb-fm)
    (match-define (flomap _ 1 zw zh) z-fm)
    (unless (and (= w zw) (= h zh))
      (error 'deep-flomap
             "expected flomaps of equal dimension; given dimensions ~e×~e and ~e×~e" w h zw zh))
    (values argb-fm z-fm)))
(: flomap->deep-flomap (flomap -> deep-flomap))
(define (flomap->deep-flomap argb-fm)
  (match-define (flomap _ 4 w h) argb-fm)
  (deep-flomap argb-fm (make-flomap 1 w h)))
(: deep-flomap-width (deep-flomap -> Nonnegative-Fixnum))
(define (deep-flomap-width dfm)
  (define w (flomap-width (deep-flomap-argb dfm)))
  (with-asserts ([w  nonnegative-fixnum?])
    w))
(: deep-flomap-height (deep-flomap -> Nonnegative-Fixnum))
(define (deep-flomap-height dfm)
  (define h (flomap-height (deep-flomap-argb dfm)))
  (with-asserts ([h  nonnegative-fixnum?])
    h))
(: deep-flomap-z-min (deep-flomap -> Flonum))
(define (deep-flomap-z-min dfm)
  (flomap-min-value (deep-flomap-z dfm)))
(: deep-flomap-z-max (deep-flomap -> Flonum))
(define (deep-flomap-z-max dfm)
  (flomap-max-value (deep-flomap-z dfm)))
(: deep-flomap-size (deep-flomap -> (values Nonnegative-Fixnum Nonnegative-Fixnum)))
(define (deep-flomap-size dfm)
  (values (deep-flomap-width dfm) (deep-flomap-height dfm)))
(: deep-flomap-alpha (deep-flomap -> flomap))
(define (deep-flomap-alpha dfm)
  (flomap-ref-component (deep-flomap-argb dfm) 0))
(: deep-flomap-rgb (deep-flomap -> flomap))
(define (deep-flomap-rgb dfm)
  (flomap-drop-components (deep-flomap-argb dfm) 1))
;; ===================================================================================================
;; Z adjusters
(: deep-flomap-scale-z (deep-flomap (U Real flomap) -> deep-flomap))
(define (deep-flomap-scale-z dfm z)
  (match-define (deep-flomap argb-fm z-fm) dfm)
  (deep-flomap argb-fm (fm* z-fm z)))
(: deep-flomap-smooth-z (deep-flomap Real -> deep-flomap))
(define (deep-flomap-smooth-z dfm σ)
  (let ([σ  (exact->inexact σ)])
    (match-define (deep-flomap argb-fm z-fm) dfm)
    (define new-z-fm (flomap-blur z-fm σ))
    (deep-flomap argb-fm new-z-fm)))
; deep-flomap-raise and everything derived from it observe an invariant:
; when z is added, added z must be 0.0 everywhere alpha is 0.0
(: deep-flomap-raise (deep-flomap (U Real flomap) -> deep-flomap))
(define (deep-flomap-raise dfm z)
  (match-define (deep-flomap argb-fm z-fm) dfm)
  (define alpha-fm (deep-flomap-alpha dfm))
  (deep-flomap argb-fm (fm+ z-fm (fm* alpha-fm z))))
(: deep-flomap-emboss (deep-flomap Real (U Real flomap) -> deep-flomap))
(define (deep-flomap-emboss dfm xy-amt z-amt)
  (let ([σ       (/ xy-amt 3.0)])
    (define z-fm (flomap-normalize (deep-flomap-alpha dfm)))
    (define new-z-fm (fm- (flomap-blur z-fm σ) z-amt))
    (deep-flomap-raise dfm new-z-fm)))
(: deep-flomap-bulge-helper (deep-flomap (Flonum Flonum -> Flonum) -> deep-flomap))
(define (deep-flomap-bulge-helper dfm f)
  (let ()
    (define-values (w h) (deep-flomap-size dfm))
    (define half-x-size (- (* 0.5 (fx->fl w)) 0.5))
    (define half-y-size (- (* 0.5 (fx->fl h)) 0.5))
    (define z-fm
      (inline-build-flomap
       1 w h
       (λ (_ x y i)
         (f (- (/ (fx->fl x) half-x-size) 1.0)
            (- (/ (fx->fl y) half-y-size) 1.0)))))
    (deep-flomap-raise dfm z-fm)))
(: deep-flomap-bulge (deep-flomap Flonum Flonum -> Real  -> deep-flomap))
(define (deep-flomap-bulge dfm fx fy)
  (deep-flomap-bulge-helper dfm (λ (cx cy) (real->double-flonum (f cx cy)))))
```


define-inline
(define (flomap-lift-helper f) ...)


There must be a better way.



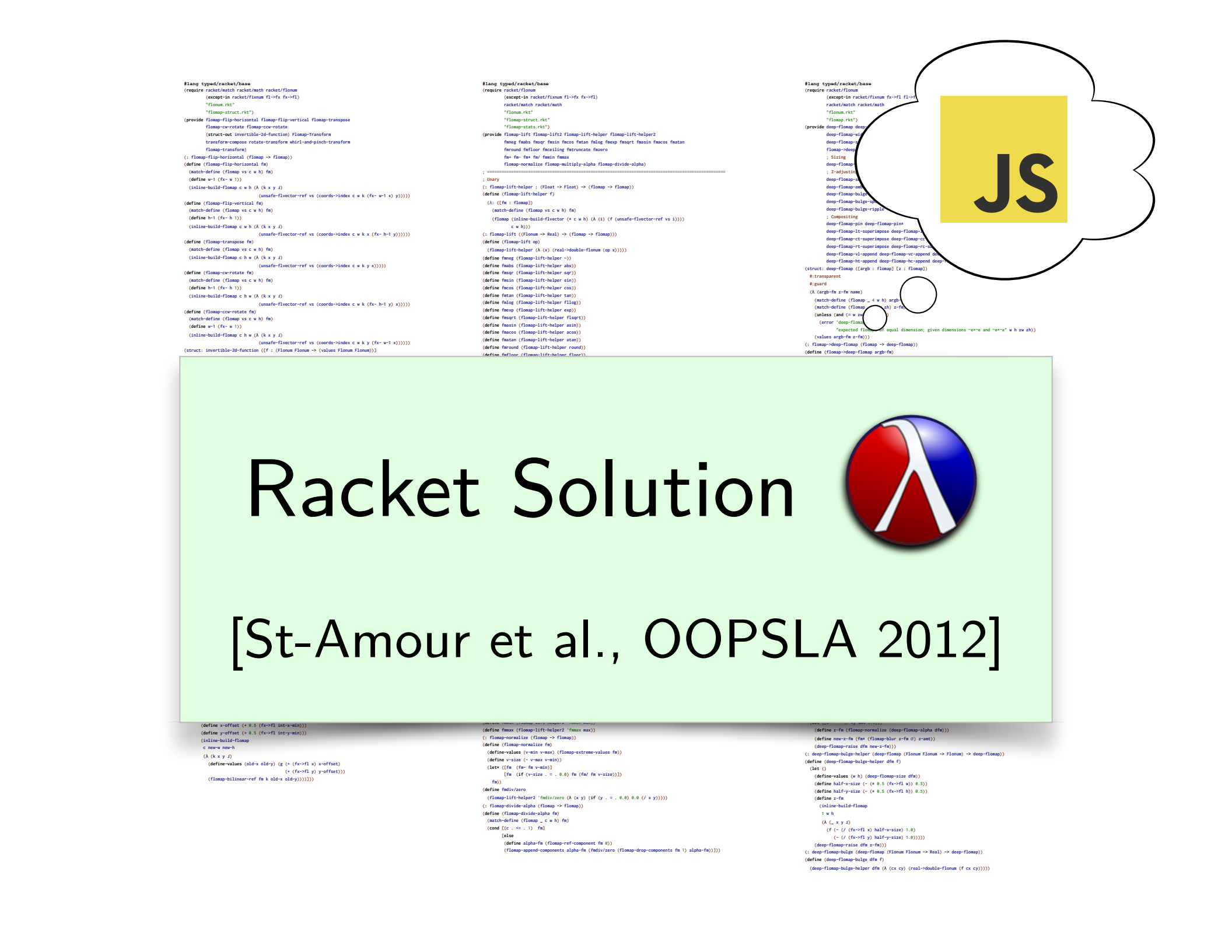
This could happen anywhere.

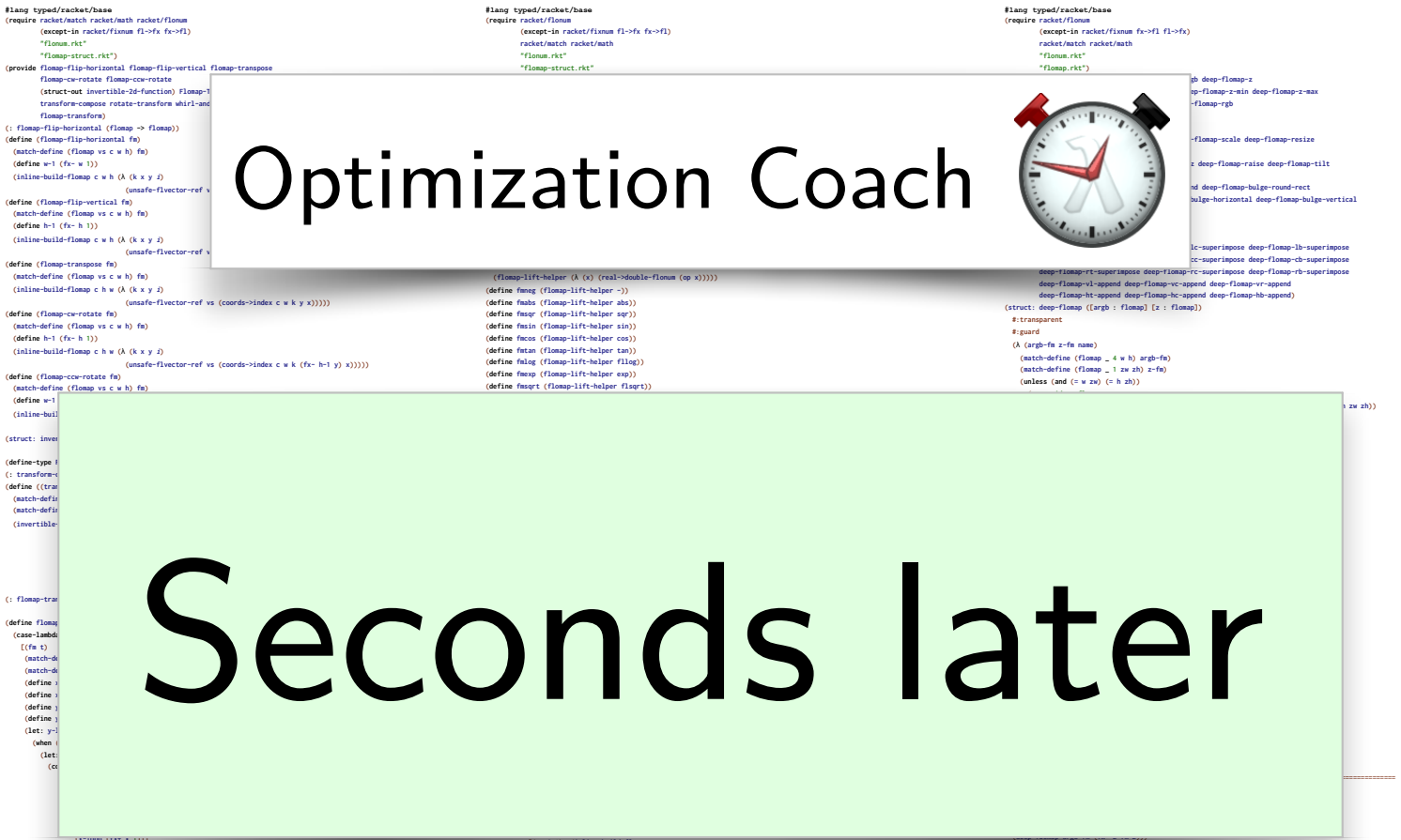

Java™
pypy
LLVM
JS
Microsoft®
.NET
This happen ar e.

This could happen anywhere.

Racket Solution

[St-Amour et al., OOPSLA 2012]

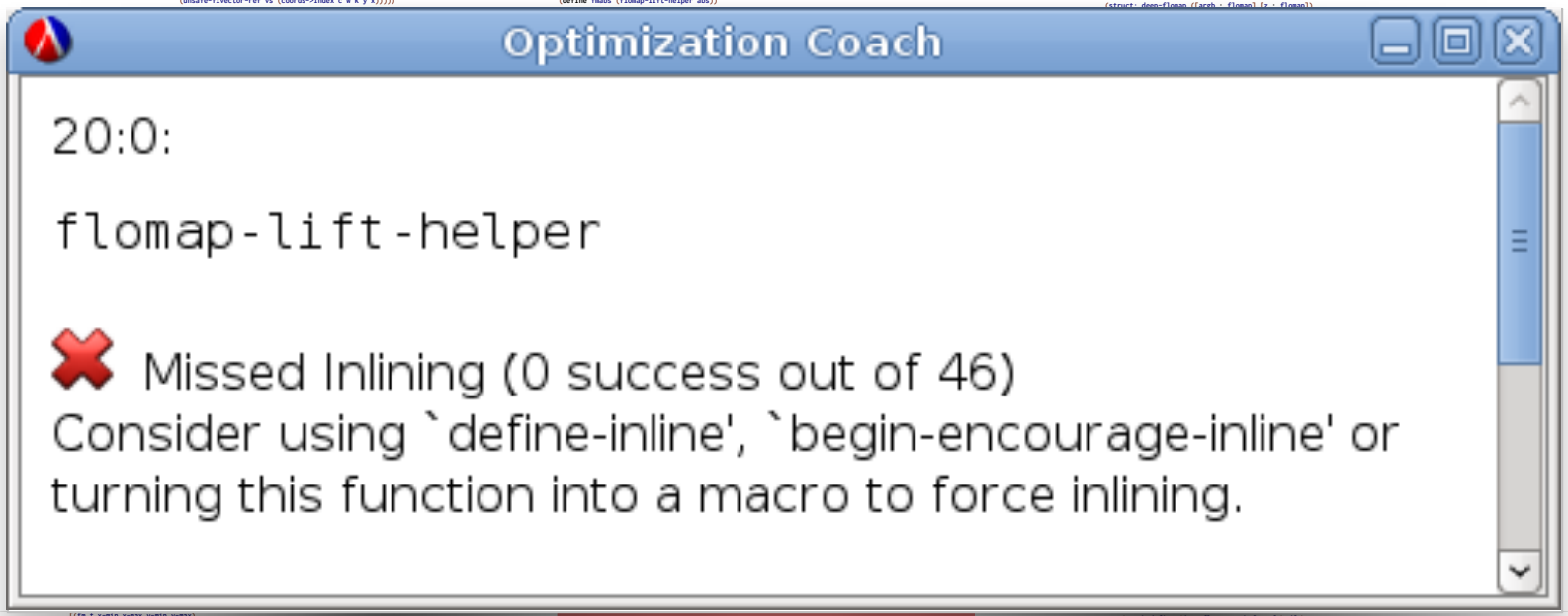

JS
Racket Solution
[St-Amour et al., OOPSLA 2012]


Optimization Coach
Seconds later

```
#lang typed/racket/base
(require racket/match racket/math racket/flonum
         (except-in racket/fixnum fl->fx fx->fl)
         "flonum.rkt"
         "flomap-struct.rkt")
(provide flomap-flip-horizontal flomap-flip-vertical flomap-transpose
         flomap-cw-rotate flomap-ccw-rotate
         (struct-out invertible-2d-function) flomap-transform
         transform-compose rotate-transform whirl-and-pinch-transform
         flomap-transform)
(: flomap-flip-horizontal (flomap -> flomap))
(define (flomap-flip-horizontal fm)
  (match-define (flomap vs c w h) fm)
  (define w-1 (fx- w 1))
  (inline-build-flomap c w h (λ (k x y i)
                               (unsafe-flvector-ref vs (coords->index c w k (fx- w-1 x) y)))))
(define (flomap-flip-vertical fm)
  (match-define (flomap vs c w h) fm)
  (define h-1 (fx- h 1))
  (inline-build-flomap c w h (λ (k x y i)
                               (unsafe-flvector-ref vs (coords->index c w k x (fx- h-1 y))))))
(define (flomap-transpose fm)
  (match-define (flomap vs c w h) fm)
  (inline-build-flomap c h w (λ (k x y i)
                               (unsafe-flvector-ref vs (coords->index c w k y x)))))
(define (flomap-cw-rotate fm)
  (match-define (flomap vs c w h) fm)
  (define h-1 (fx- h 1))
  (inline-build-flomap c h w (λ (k x y i)
                               (unsafe-flvector-ref vs (coords->index c w k (fx- h-1 y) x)))))
(define (flomap-ccw-rotate fm)
  (match-define (flomap vs c w h) fm)
  (define w-1 (fx- w 1))
  (inline-build-flomap c h w (λ (k x y i)
                               (unsafe-flvector-ref vs (coords->index c w k y (fx- w-1 x))))))
(struct: invertible-2d-function ([f : (Flonum Flonum -> (values Flonum Flonum))]
                                 [g : (Flonum Flonum -> (values Flonum Flonum))]))
(define-type Flomap-Transform (Integer Integer -> invertible-2d-function))
(: transform-compose (Flomap-Transform Flomap-Transform -> Flomap-Transform))
(define ((transform-compose t1 t2) w h)
  (match-define (invertible-2d-function f1 g1) (t1 w h))
  (match-define (invertible-2d-function f2 g2) (t2 w h))
  (invertible-2d-function (λ: ([x : Flonum] [y : Flonum])
                            (let-values ([(x y)  (f2 x y)])
                              (f1 x y)))
                          (λ: ([x : Flonum] [y : Flonum])
                            (let-values ([(x y)  (g1 x y)])
                              (g2 x y)))))
(: flomap-transform (case-> (flomap Flomap-Transform -> flomap)
                            (flomap Flomap-Transform Real Real Real Real -> flomap)))
(define flomap-transform
  (case-lambda
    [(fm t)
     (match-define (flomap vs c w h) fm)
     (match-define (invertible-2d-function f g) (t w h))
     (define x-min +inf.0)
     (define x-max -inf.0)
     (define y-min +inf.0)
     (define y-max -inf.0)
     (let: y-loop : Void ([y : Integer  0])
       (when (y . fx< . h)
         (let: x-loop : Void ([x : Integer  0])
           (cond [(x . fx< . w)
                  (define-values (new-x new-y) (f (+ 0.5 (fx->fl x)) (+ 0.5 (fx->fl y))))
                  (when (new-x . < . x-min) (set! x-min new-x))
                  (when (new-x . > . x-max) (set! x-max new-x))
                  (when (new-y . < . y-min) (set! y-min new-y))
                  (when (new-y . > . y-max) (set! y-max new-y))
                  (x-loop (fx+ x 1))]
                 [else
                  (y-loop (fx+ y 1))]))))
     (flomap-transform fm t x-min x-max y-min y-max)]
    [(fm t x-min x-max y-min y-max)
     (let ([x-min  (real->double-flonum x-min)]
           [x-max  (real->double-flonum x-max)]
           [y-min  (real->double-flonum y-min)]
           [y-max  (real->double-flonum y-max)])
       (match-define (flomap vs c w h) fm)
       (match-define (invertible-2d-function f g) (t w h))
       (define int-x-min (fl->fx (floor x-min)))
       (define int-x-max (fl->fx (ceiling x-max)))
       (define int-y-min (fl->fx (floor y-min)))
       (define int-y-max (fl->fx (ceiling y-max)))
       (define new-w (- int-x-max int-x-min))
       (define new-h (- int-y-max int-y-min))
       (define x-offset (+ 0.5 (fx->fl int-x-min)))
       (define y-offset (+ 0.5 (fx->fl int-y-min)))
       (inline-build-flomap
        c new-w new-h
        (λ (k x y i)
          (define-values (old-x old-y) (g (+ (fx->fl x) x-offset)
                                          (+ (fx->fl y) y-offset)))
          (flomap-bilinear-ref fm k old-x old-y))))]))
```

```
#lang typed/racket/base
(require racket/flonum
         (except-in racket/fixnum fl->fx fx->fl)
         racket/match racket/math
         "flonum.rkt"
         "flomap-struct.rkt"
         "flomap-stats.rkt")
(provide flomap-lift flomap-lift2 flomap-lift-helper flomap-lift-helper2
         fmneg fmabs fmsqr fmsin fmcos fmtan fmlog fmexp fmsqrt fmasin fmacos fmatan
         fmround fmfloor fmceiling fmtruncate fmzero
         fm+ fm- fm* fm/ fmmin fmmax
         flomap-normalize flomap-multiply-alpha flomap-divide-alpha)
; =============================================================================
; Unary
(: flomap-lift-helper : (Float -> Float) -> (flomap -> flomap))
(define (flomap-lift-helper f)
  (λ: ([fm : flomap])
    (match-define (flomap vs c w h) fm)
    (flomap (inline-build-flvector (* c w h) (λ (i) (f (unsafe-flvector-ref vs i))))
            c w h)))
(: flomap-lift ((Flonum -> Real) -> (flomap -> flomap)))
(define (flomap-lift op)
  (flomap-lift-helper (λ (x) (real->double-flonum (op x)))))
(define fmneg (flomap-lift-helper -))
(define fmabs (flomap-lift-helper abs))
(define fmsqr (flomap-lift-helper sqr))
(define fmsin (flomap-lift-helper sin))
(define fmcos (flomap-lift-helper cos))
(define fmtan (flomap-lift-helper tan))
(define fmlog (flomap-lift-helper fllog))
(define fmexp (flomap-lift-helper exp))
(define fmsqrt (flomap-lift-helper flsqrt))
(define fmasin (flomap-lift-helper asin))
(define fmacos (flomap-lift-helper acos))
(define fmatan (flomap-lift-helper atan))
(define fmround (flomap-lift-helper round))
(define fmfloor (flomap-lift-helper floor))
(define fmceiling (flomap-lift-helper ceiling))
(define fmtruncate (flomap-lift-helper truncate))
(define fmzero (flomap-lift-helper (λ (x) (if (x . = . 0.0) 1.0 0.0))))
; ;; =============================================================================
; Binary
(: flomap-lift-helper2 : Symbol (Float Float -> Float) -> ((U Real flomap) (U Real flomap) -> flomap))
(define (flomap-lift-helper2 name f)
  (let: ()
    (λ: ([fm1 : (U Real flomap)] [fm2 : (U Real flomap)])
      (cond
        [(and (real? fm1) (real? fm2))
         (error name "expected at least one flomap argument; given ~e and ~e" fm1 fm2)]
        [(real? fm1)  (let ([fm1  (real->double-flonum fm1)])
                        ((flomap-lift-helper (λ (v) (f fm1 v))) fm2))]
        [(real? fm2)  (let ([fm2  (real->double-flonum fm2)])
                        ((flomap-lift-helper (λ (v) (f v fm2))) fm1))]
        [else
         (match-define (flomap vs1 c1 w h) fm1)
         (match-define (flomap vs2 c2 w2 h2) fm2)
         (cond
           [(not (and (= w w2) (= h h2)))
            (error name "expected same-size flomaps; given sizes ~ex~e and ~ex~e" w h w2 h2)]
           [(= c1 c2)  (define n (* c1 w h))
                       (define res-vs (make-flvector n))
                       (flomap (inline-build-flvector n (λ (i) (f (unsafe-flvector-ref vs1 i)
                                                                 (unsafe-flvector-ref vs2 i))))
                               c1 w h)]
           [(= c1 1)  (inline-build-flomap
                       c2 w h
                       (λ (k x y i) (f (unsafe-flvector-ref vs1 (coords->index 1 w 0 x y))
                                       (unsafe-flvector-ref vs2 i))))]
           [(= c2 1)  (inline-build-flomap
                       c1 w h
                       (λ (k x y i) (f (unsafe-flvector-ref vs1 i)
                                       (unsafe-flvector-ref vs2 (coords->index 1 w 0 x y)))))]
           [else
            (error name (string-append "expected flomaps with the same number of components, "
                                       "or a flomap with 1 component and any same-size flomap; "
                                       "given flomaps with ~e and ~e components")
                   c1 c2))])]))))
(: flomap-lift2 (Symbol (Flonum Flonum -> Real) -> ((U Real flomap) (U Real flomap) -> flomap)))
(define (flomap-lift2 name f)
  (flomap-lift-helper2 name (λ (x y) (real->double-flonum (f x y)))))
(define fm+ (flomap-lift-helper2 'fm+ +))
(define fm- (flomap-lift-helper2 'fm- -))
(define fm* (flomap-lift-helper2 'fm* *))
(define fm/ (flomap-lift-helper2 'fm/ /))
(define fmmin (flomap-lift-helper2 'fmmin min))
(define fmmax (flomap-lift-helper2 'fmmax max))
(: flomap-normalize (flomap -> flomap))
(define (flomap-normalize fm)
  (define-values (v-min v-max) (flomap-extreme-values fm))
  (define v-size (- v-max v-min))
  (let* ([fm  (fm- fm v-min)]
         [fm  (if (v-size . = . 0.0) fm (fm/ fm v-size))])
    fm))
(define fmdiv/zero
  (flomap-lift-helper2 'fmdiv/zero (λ (x y) (if (y . = . 0.0) 0.0 (/ x y)))))
(: flomap-divide-alpha (flomap -> flomap))
(define (flomap-divide-alpha fm)
  (match-define (flomap _ c w h) fm)
  (cond [(c . <= . 1)  fm]
        [else
         (define alpha-fm (flomap-ref-component fm 0))
         (flomap-append-components alpha-fm (fmdiv/zero (flomap-drop-components fm 1) alpha-fm))]))
```

```
#lang typed/racket/base
(require racket/flonum
         (except-in racket/fixnum fx->fl fl->fx)
         racket/match racket/math
         "flonum.rkt"
         "flomap.rkt")
(provide deep-flomap deep-flomap? deep-flomap-argb deep-flomap-z
         deep-flomap-width deep-flomap-height deep-flomap-z-min deep-flomap-z-max
         deep-flomap-size deep-flomap-alpha deep-flomap-rgb
         flomap->deep-flomap
         ; Sizing
         deep-flomap-inset deep-flomap-trim deep-flomap-scale deep-flomap-resize
         ; Z-adjusting
         deep-flomap-scale-z deep-flomap-smooth-z deep-flomap-raise deep-flomap-tilt
         deep-flomap-emboss
         deep-flomap-bulge deep-flomap-bulge-round deep-flomap-bulge-round-rect
         deep-flomap-bulge-spheroid deep-flomap-bulge-horizontal deep-flomap-bulge-vertical
         deep-flomap-bulge-ripple
         ; Compositing
         deep-flomap-pin deep-flomap-pin*
         deep-flomap-lt-superimpose deep-flomap-lc-superimpose deep-flomap-lb-superimpose
         deep-flomap-ct-superimpose deep-flomap-cc-superimpose deep-flomap-cb-superimpose
         deep-flomap-rt-superimpose deep-flomap-rc-superimpose deep-flomap-rb-superimpose
         deep-flomap-vl-append deep-flomap-vc-append deep-flomap-vr-append
         deep-flomap-ht-append deep-flomap-hc-append deep-flomap-hb-append)
(struct: deep-flomap ([argb : flomap] [z : flomap])
  #:transparent
  #:guard
  (λ (argb-fm z-fm name)
    (match-define (flomap _ 4 w h) argb-fm)
    (match-define (flomap _ 1 zw zh) z-fm)
    (unless (and (= w zw) (= h zh))
      (error 'deep-flomap
             "expected flomaps of equal dimension; given dimensions ~ex~e and ~ex~e" w h zw zh))
    (values argb-fm z-fm)))
(: flomap->deep-flomap (flomap -> deep-flomap))
(define (flomap->deep-flomap argb-fm)
  (match-define (flomap _ 4 w h) argb-fm)
  (deep-flomap argb-fm (make-flomap 1 w h)))
(: deep-flomap-width (deep-flomap -> Nonnegative-Fixnum))
(define (deep-flomap-width dfm)
  (define w (flomap-width (deep-flomap-argb dfm)))
  (with-asserts ([w  nonnegative-fixnum?])
    w))
(: deep-flomap-height (deep-flomap -> Nonnegative-Fixnum))
(define (deep-flomap-height dfm)
  (define h (flomap-height (deep-flomap-argb dfm)))
  (with-asserts ([h  nonnegative-fixnum?])
    h))
(: deep-flomap-z-min (deep-flomap -> Flonum))
(define (deep-flomap-z-min dfm)
  (flomap-min-value (deep-flomap-z dfm)))
(: deep-flomap-z-max (deep-flomap -> Flonum))
(define (deep-flomap-z-max dfm)
  (flomap-max-value (deep-flomap-z dfm)))
(: deep-flomap-size (deep-flomap -> (values Nonnegative-Fixnum Nonnegative-Fixnum)))
(define (deep-flomap-size dfm)
  (values (deep-flomap-width dfm) (deep-flomap-height dfm)))
(: deep-flomap-alpha (deep-flomap -> flomap))
(define (deep-flomap-alpha dfm)
  (flomap-ref-component (deep-flomap-argb dfm) 0))
(: deep-flomap-rgb (deep-flomap -> flomap))
(define (deep-flomap-rgb dfm)
  (flomap-drop-components (deep-flomap-argb dfm) 1))
; =============================================================================
; Z adjusters
(: deep-flomap-scale-z (deep-flomap (U Real flomap) -> deep-flomap))
(define (deep-flomap-scale-z dfm z)
  (match-define (deep-flomap argb-fm z-fm) dfm)
  (deep-flomap argb-fm (fm* z-fm z)))
(: deep-flomap-smooth-z (deep-flomap Real -> deep-flomap))
(define (deep-flomap-smooth-z dfm σ)
  (let ([σ  (exact->inexact σ)])
    (match-define (deep-flomap argb-fm z-fm) dfm)
    (define new-z-fm (flomap-blur z-fm σ))
    (deep-flomap argb-fm new-z-fm)))
; deep-flomap-raise and everything derived from it observe an invariant:
; when z is added, added z must be 0.0 everywhere alpha is 0.0
(: deep-flomap-raise (deep-flomap (U Real flomap) -> deep-flomap))
(define (deep-flomap-raise dfm z)
  (match-define (deep-flomap argb-fm z-fm) dfm)
  (define alpha-fm (deep-flomap-alpha dfm))
  (deep-flomap argb-fm (fm+ z-fm (fm* alpha-fm z))))
(: deep-flomap-emboss (deep-flomap Real (U Real flomap) -> deep-flomap))
(define (deep-flomap-emboss dfm xy-amt z-amt)
  (let ([σ       (/ xy-amt 3.0)])
    (define z-fm (flomap-normalize (deep-flomap-alpha dfm)))
    (define new-z-fm (fm* (flomap-blur z-fm σ) z-amt))
    (deep-flomap-raise dfm new-z-fm)))
(: deep-flomap-bulge-helper (deep-flomap (Flonum Flonum -> Flonum) -> deep-flomap))
(define (deep-flomap-bulge-helper dfm f)
  (let ()
    (define-values (w h) (deep-flomap-size dfm))
    (define half-x-size (- (* 0.5 (fx->fl w)) 0.5))
    (define half-y-size (- (* 0.5 (fx->fl h)) 0.5))
    (define z-fm
      (inline-build-flomap
       1 w h
       (λ (_ x y i)
         (f (- (/ (fx->fl x) half-x-size) 1.0)
            (- (/ (fx->fl y) half-y-size) 1.0)))))
    (deep-flomap-raise dfm z-fm)))
(: deep-flomap-bulge (deep-flomap (Flonum Flonum -> Real) -> deep-flomap))
(define (deep-flomap-bulge dfm f)
  (deep-flomap-bulge-helper dfm (λ (cx cy) (real->double-flonum (f cx cy)))))
```

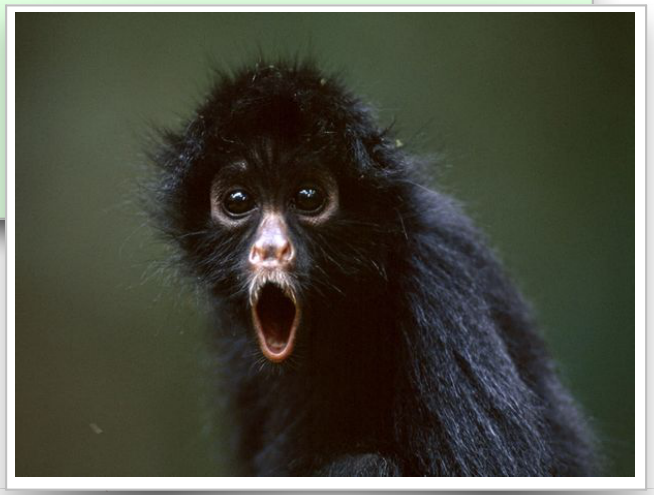

Optimization Coach
20:0:
flomap-lift-helper
Missed Inlining (0 success out of 46)
Consider using `define-inline', `begin-encourage-inline' or
turning this function into a macro to force inlining.

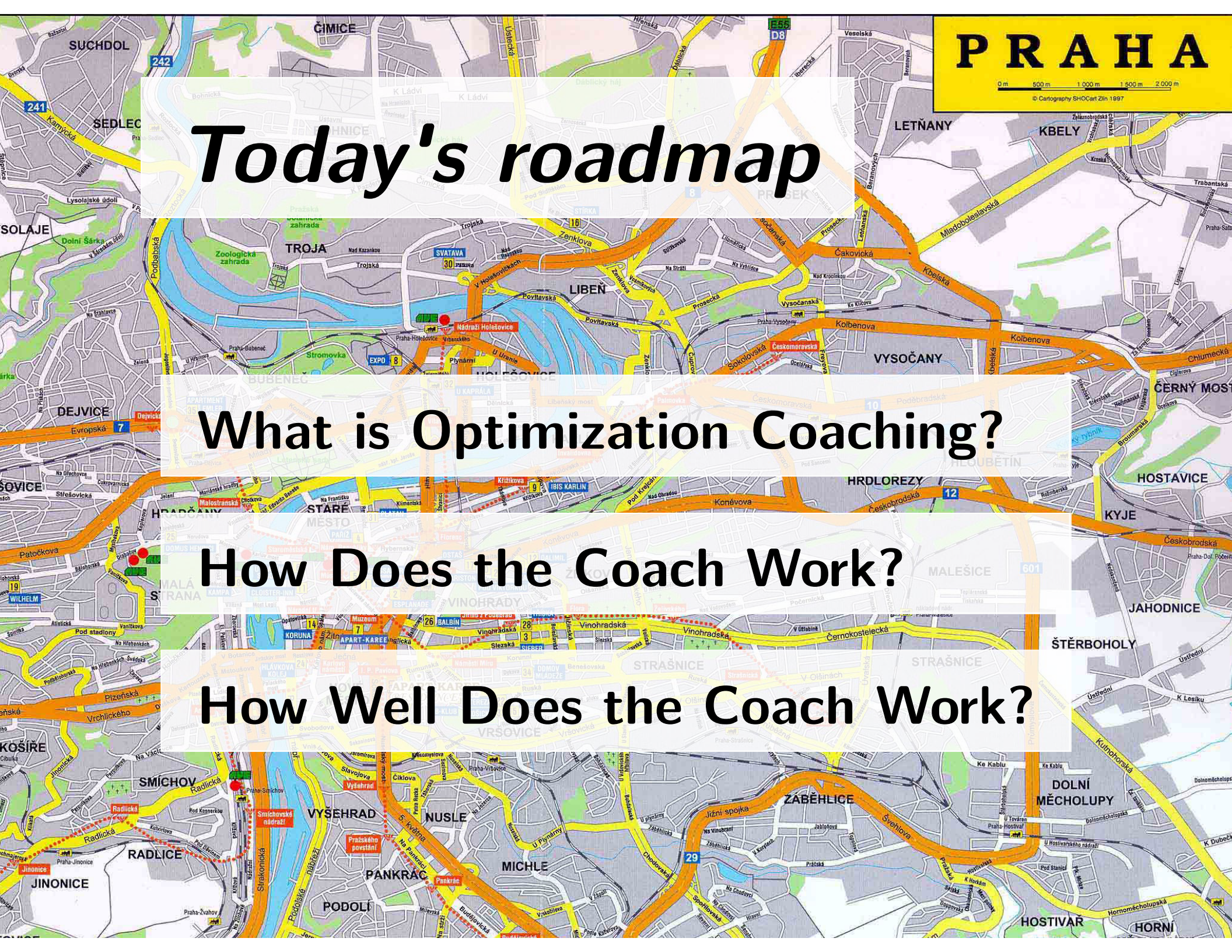
JavaScript Solution

This talk

JavaScript Solution

This talk





Today's roadmap

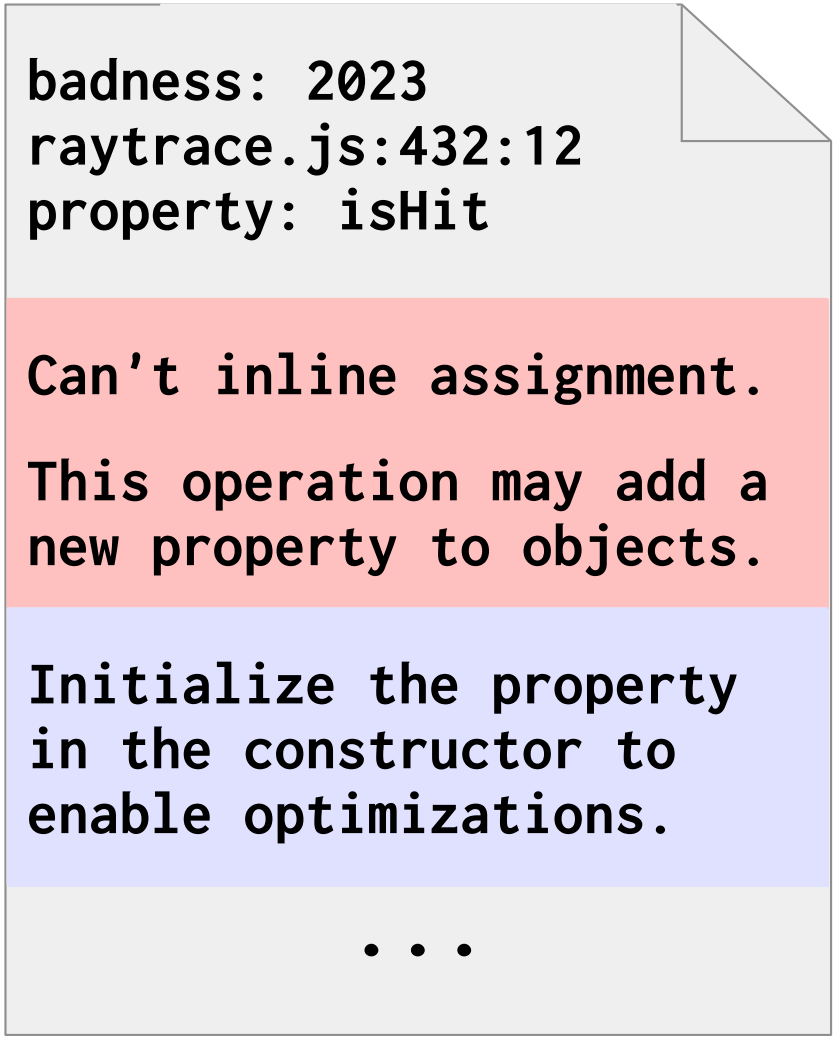
What is Optimization Coaching?

How Does the Coach Work?

How Well Does the Coach Work?

What is Optimization Coaching?

Dialog between compilers and programmers



Near misses

+

Recommendations

Compilers must be conservative

```
saturn_V =
  {height: 110,
   stages: 3}
```

```
ariane_5 =
  {stages: 2,
   height: 52}
```

Compilers must be conservative

```
saturn_V =
  {height: 110,
   stages: 3}
```

```
ariane_5 =
  {stages: 2,
   height: 52}
```

```
ariane_5 =
  {height: 52,
   stages: 2}
```

Uniformity → Optimizations!

Compilers must be conservative

```
saturn_V =
  {height: 110,
   stages: 3}
```

```
ariane_5 =
  {stages: 2,
   height: 52}
```

≠

```
ariane_5 =
  {height: 52,
   stages: 2}
```

```
print("My rocket has:")
for (var p in rocket)
    print(p, rocket[p])
```

```
My rocket has:
height 52
stages 2
```

≠

```
My rocket has:
stages 2
height 52
```

Compilers must be conservative

```
saturn_V =
  {height: 110,
   stages: 3}
```

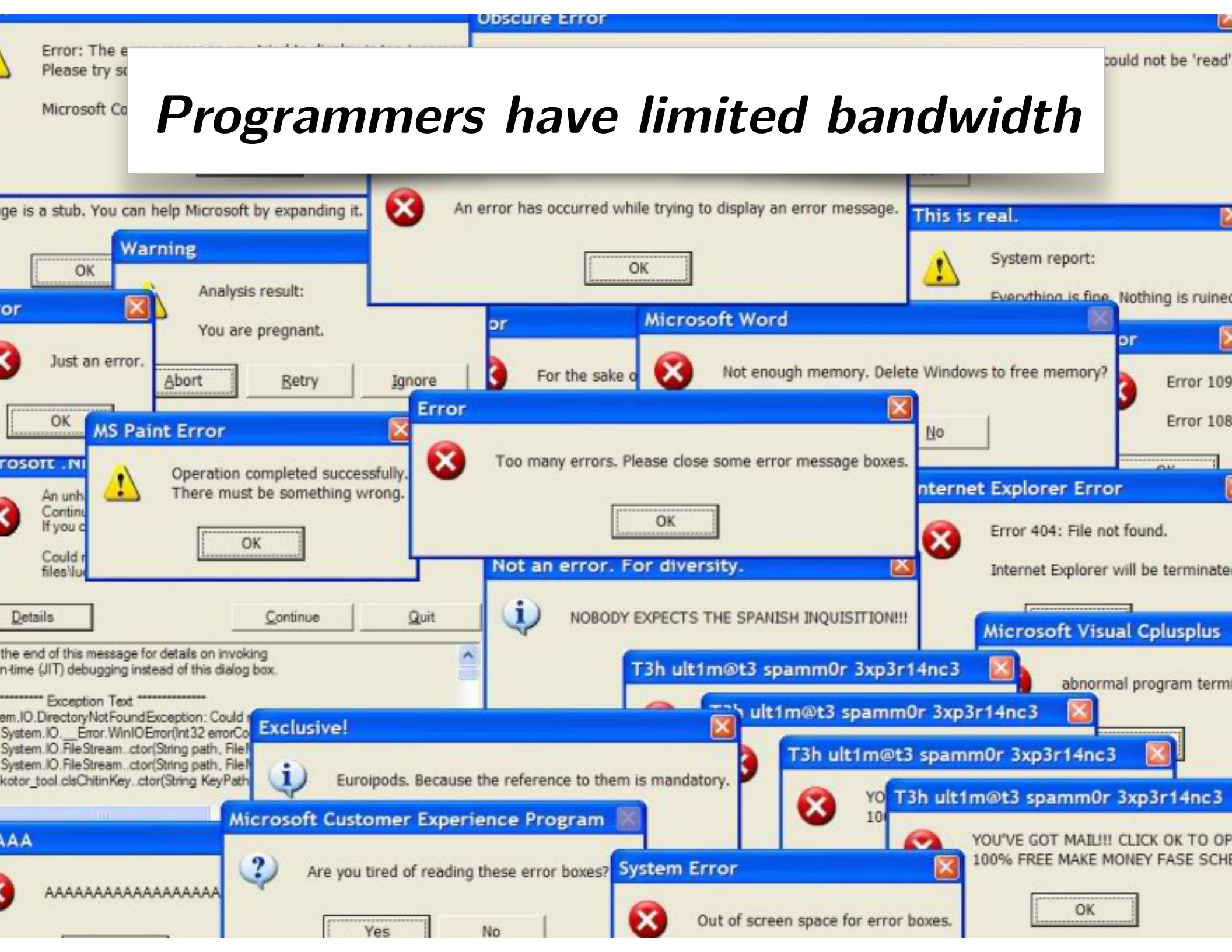
```
ariane_5 =
  {stages: 2,
   height: 52}
```

```
ariane_5 =
  {height: 52,
   stages: 2}
```

These properties are not always in the same location.

Try to always initialize them in the same order.

Recommendations can **change semantics**!


Programmers have limited bandwidth
Obscure Error
An error has occurred while trying to display an error message.
OK
Warning
Analysis result:
You are pregnant.
Abort
Retry
Ignore
This is real.
System report:
Microsoft Word
Not enough memory. Delete Windows to free memory?
No
Just an error.
MS Paint Error
Operation completed successfully.
There must be something wrong.
Error
Too many errors. Please close some error message boxes.
Not an error. For diversity.
NOBODY EXPECTS THE SPANISH INQUISITION!!!
Internet Explorer Error
Error 404: File not found.
Details
Continue
Quit
Microsoft Visual Cplusplus
abnormal program termi
T3h ult1m@t3 spamm0r 3xp3r14nc3
Exclusive!
Euroipods. Because the reference to them is mandatory.
Microsoft Customer Experience Program
Are you tired of reading these error boxes?
Yes
No
System Error
Out of screen space for error boxes.
YOU'VE GOT MAIL!!! CLICK OK TO OP
100% FREE MAKE MONEY FASE SCHE

Programmers have limited bandwidth

Why JS for coaching?

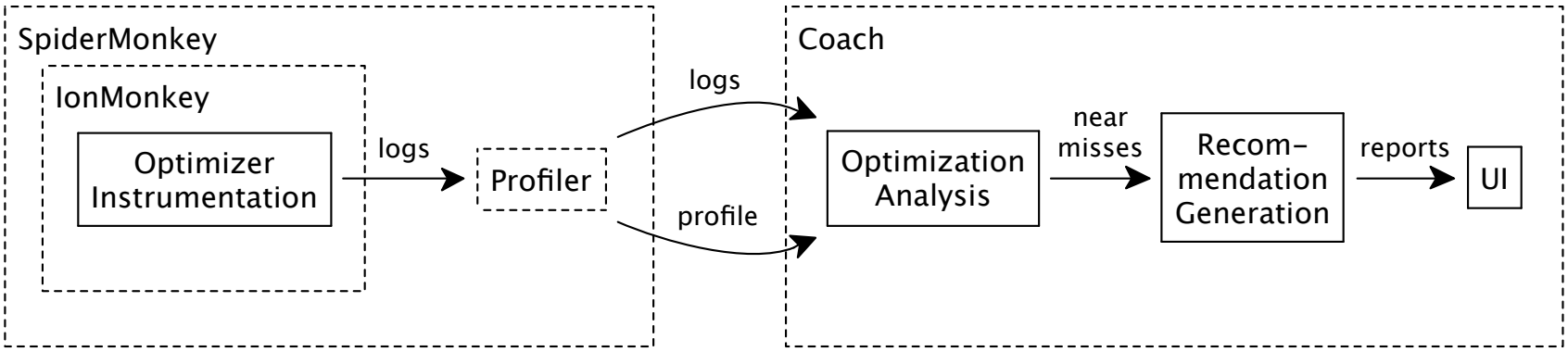
- Can it work beyond Racket?
- Different compilation model (JIT)
- Different language (OO)

Why coaching for JS?

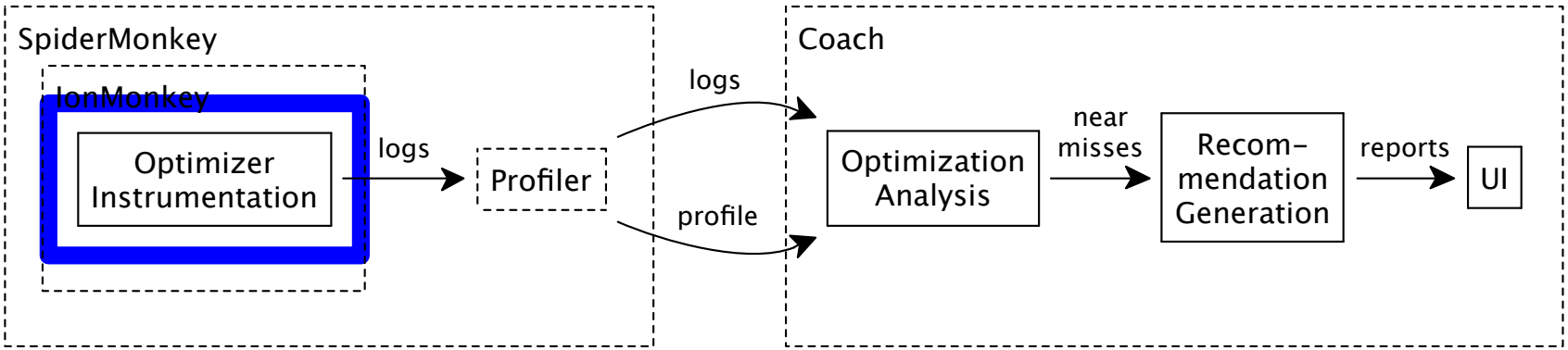
- Hard to write performant code
- Performance matters
- Non-experts / multi-language programmers

How does it work?

Architecture



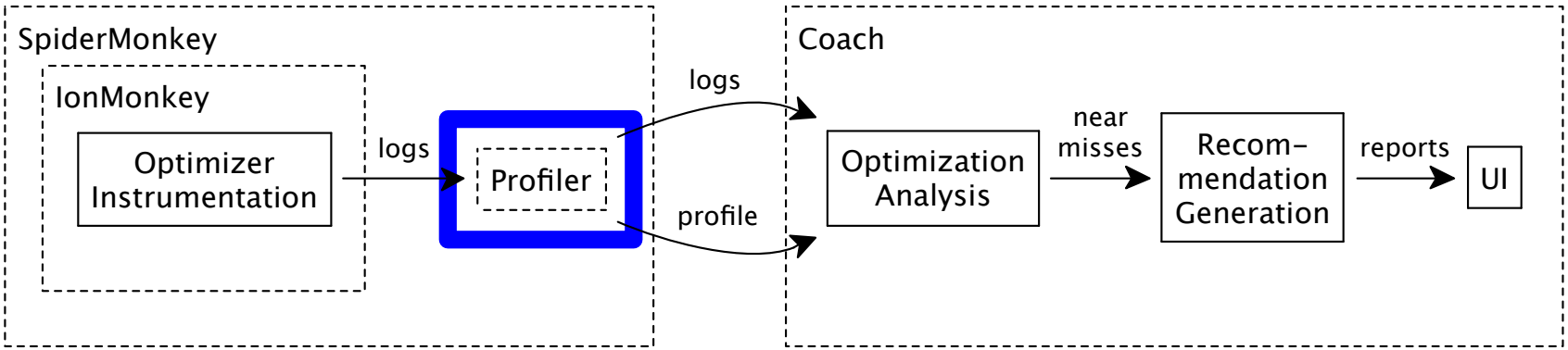
Architecture



Log optimization decisions

(attempts, successes, failures)

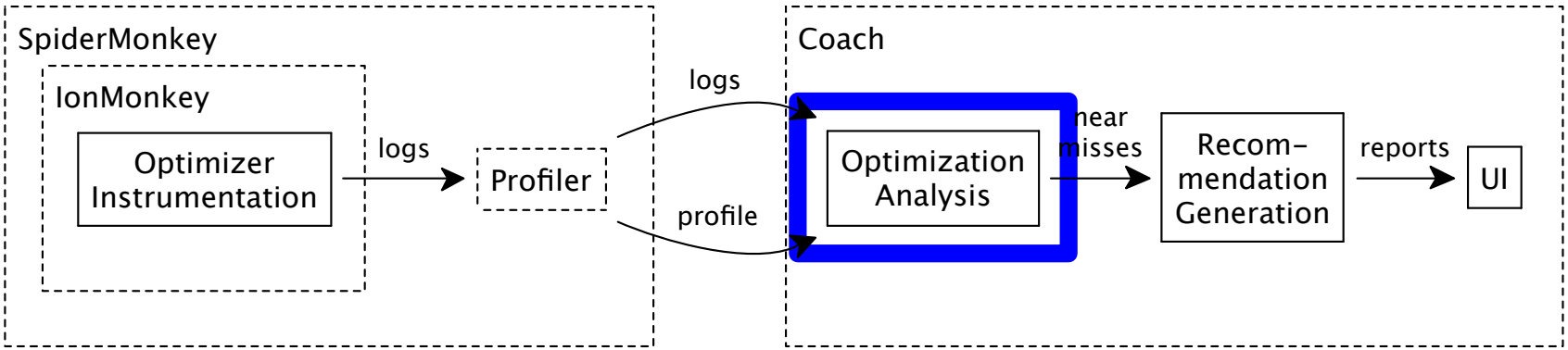
Architecture



Emit profile events

(get logs out of the engine)

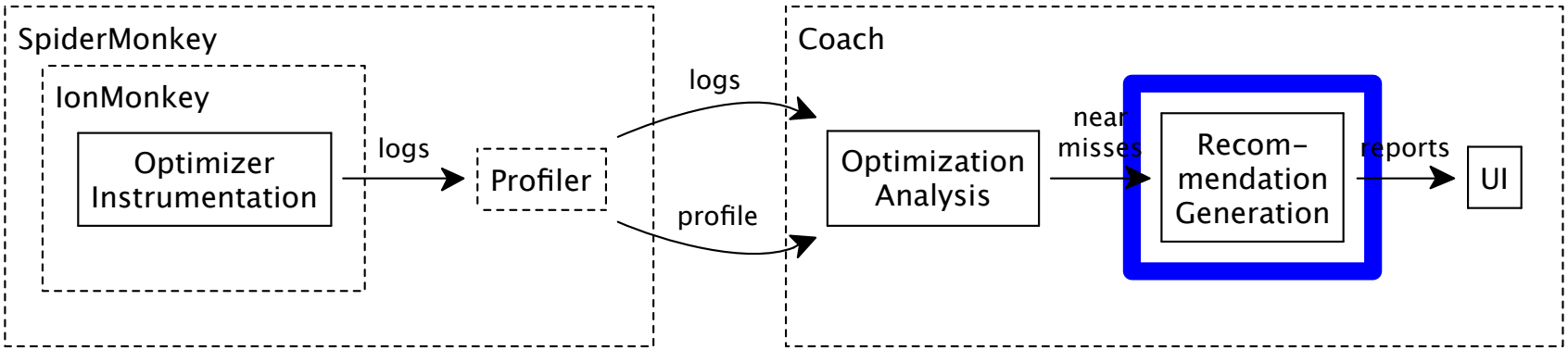
Architecture



Produce near miss reports

(pruning, merging, ranking)

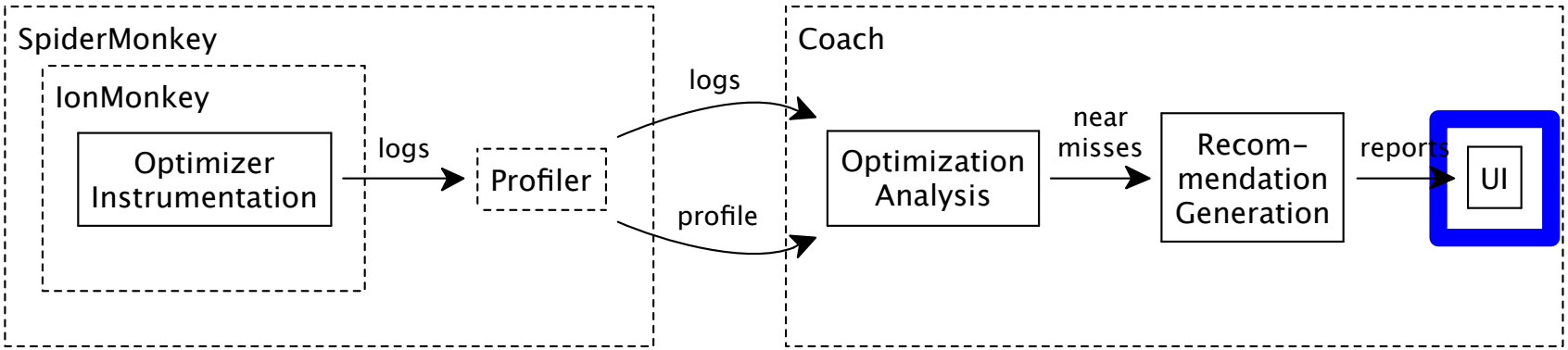
Architecture



Fill recommendation templates

(general advice + targeted info)

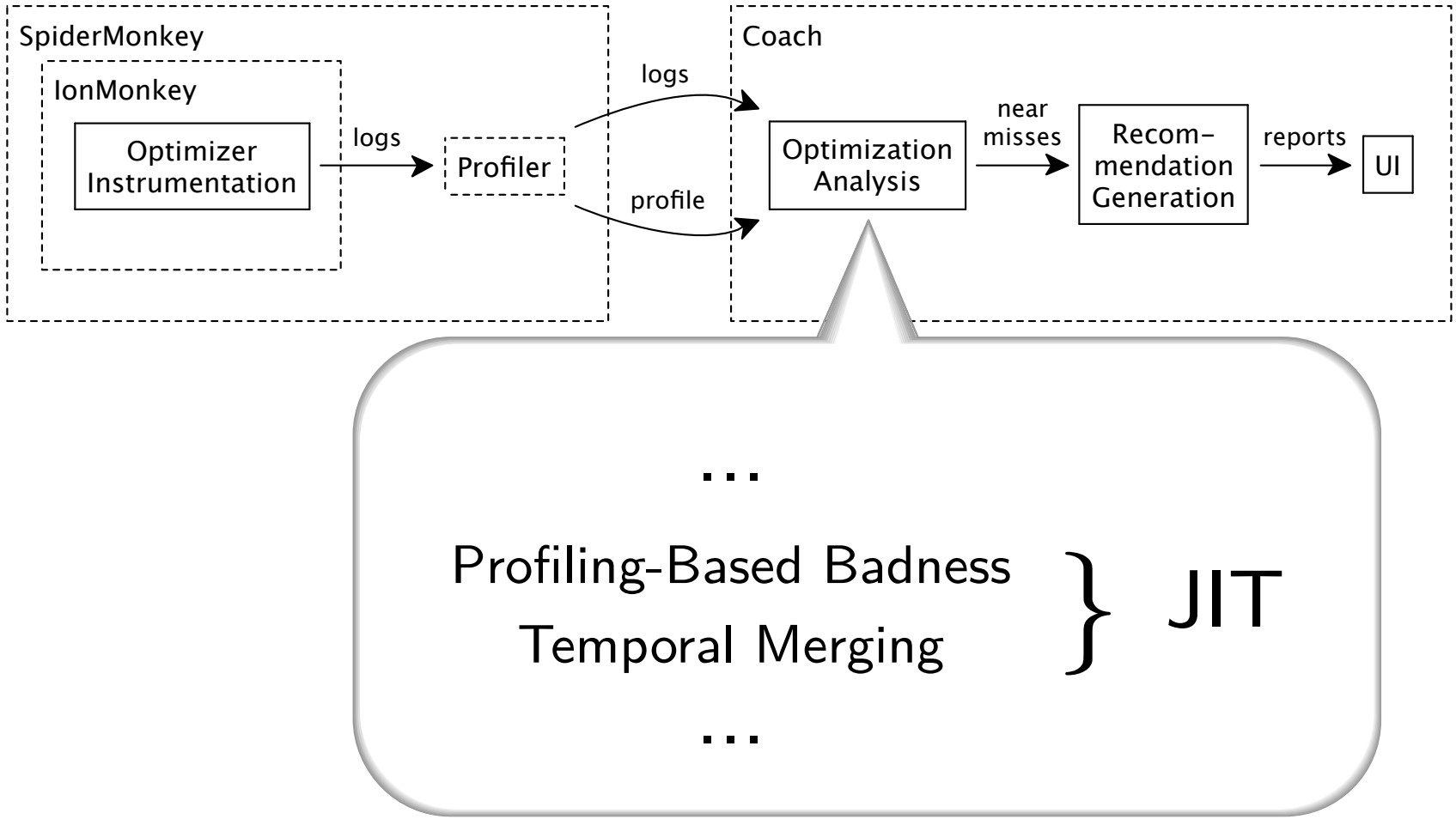
Architecture



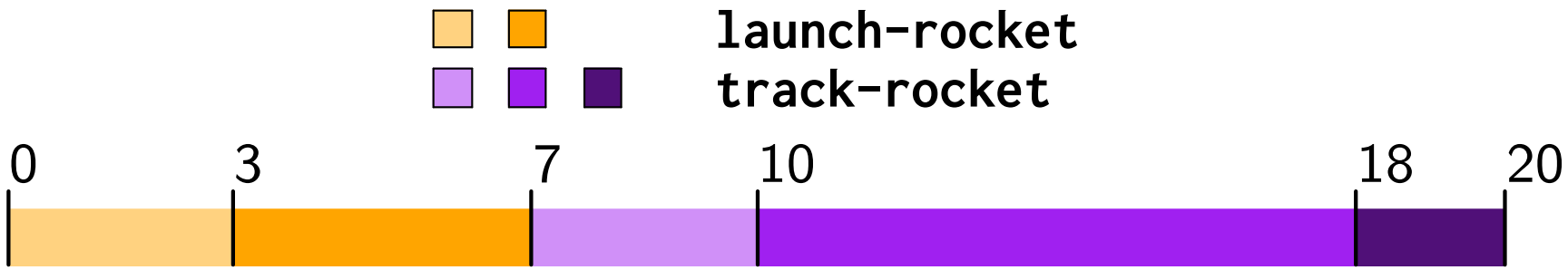
Show reports and recommendations

(consumed by programmers)

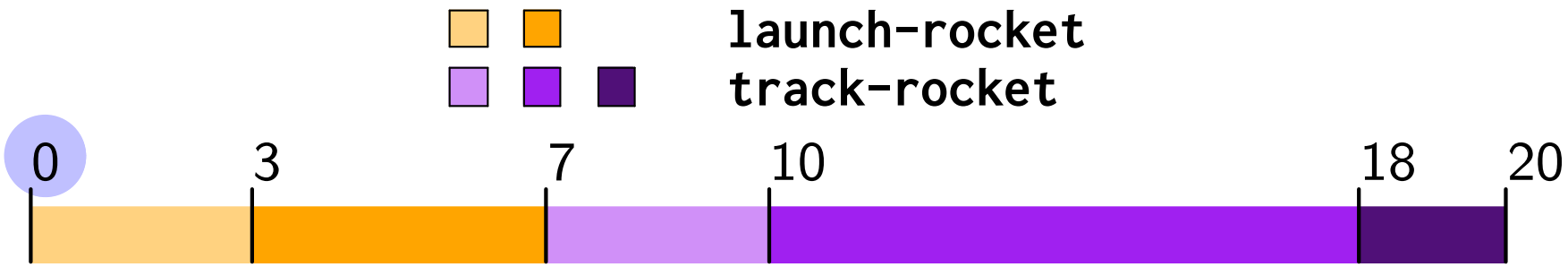
Architecture



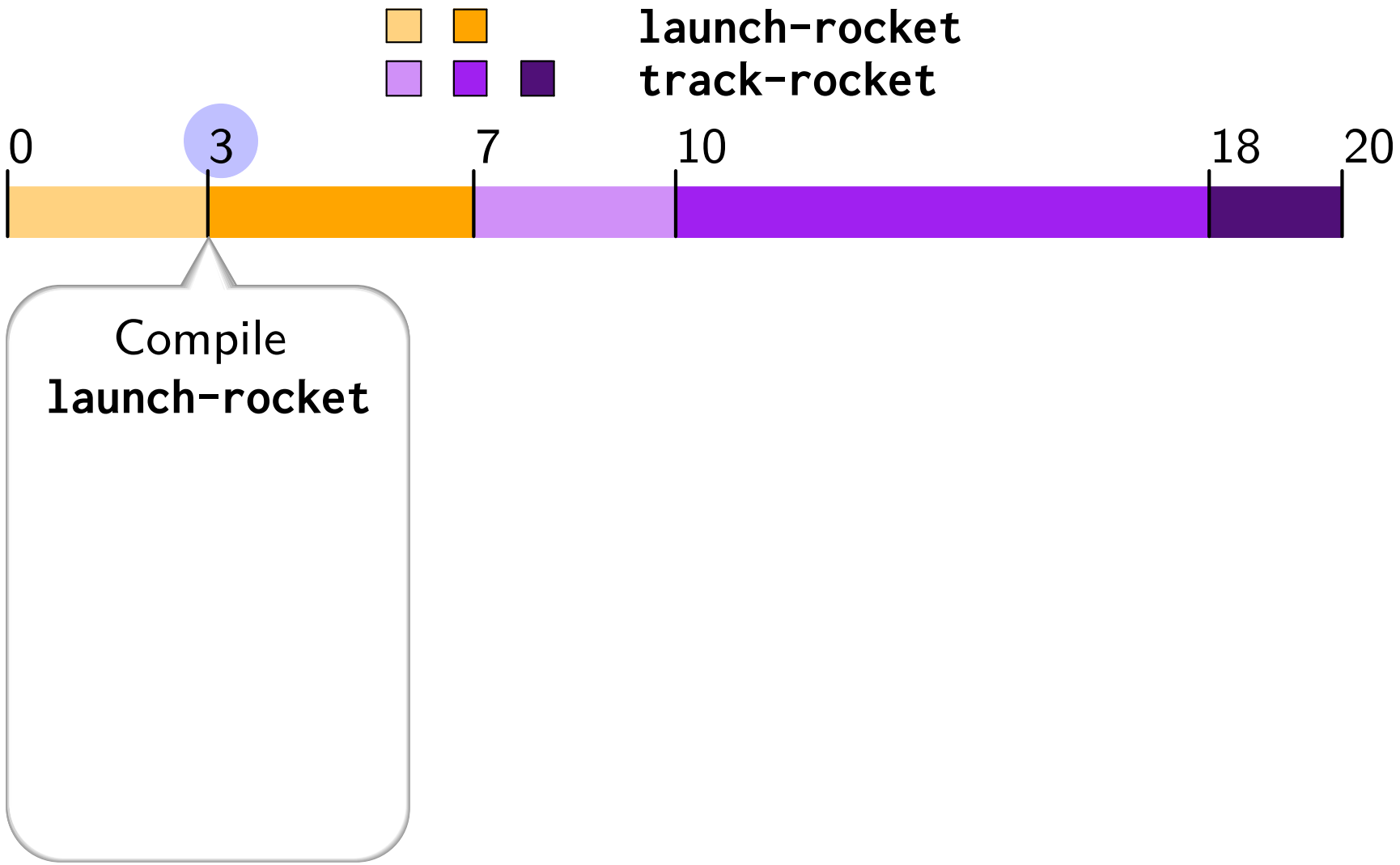
Profiling-Based Badness



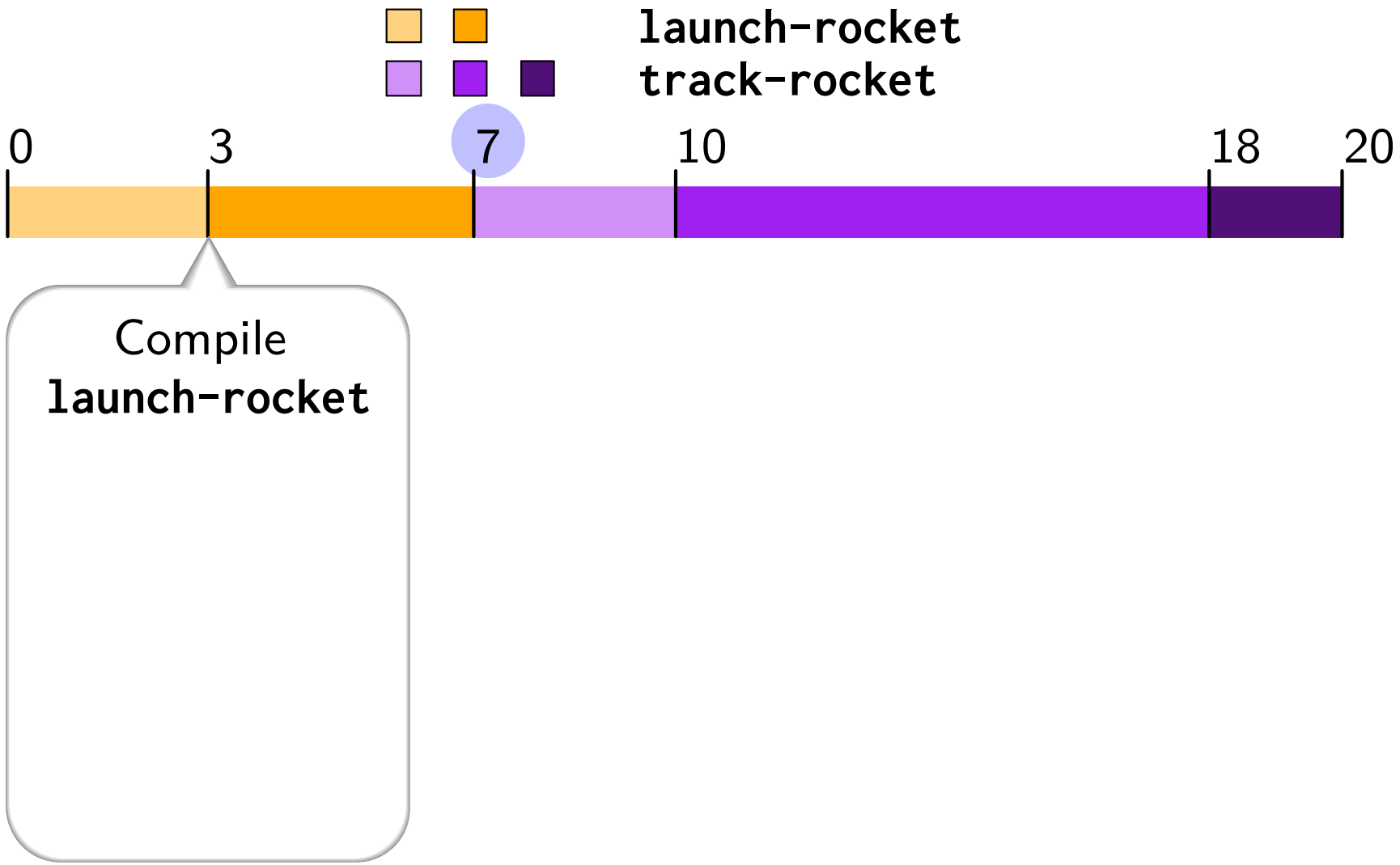
Profiling-Based Badness



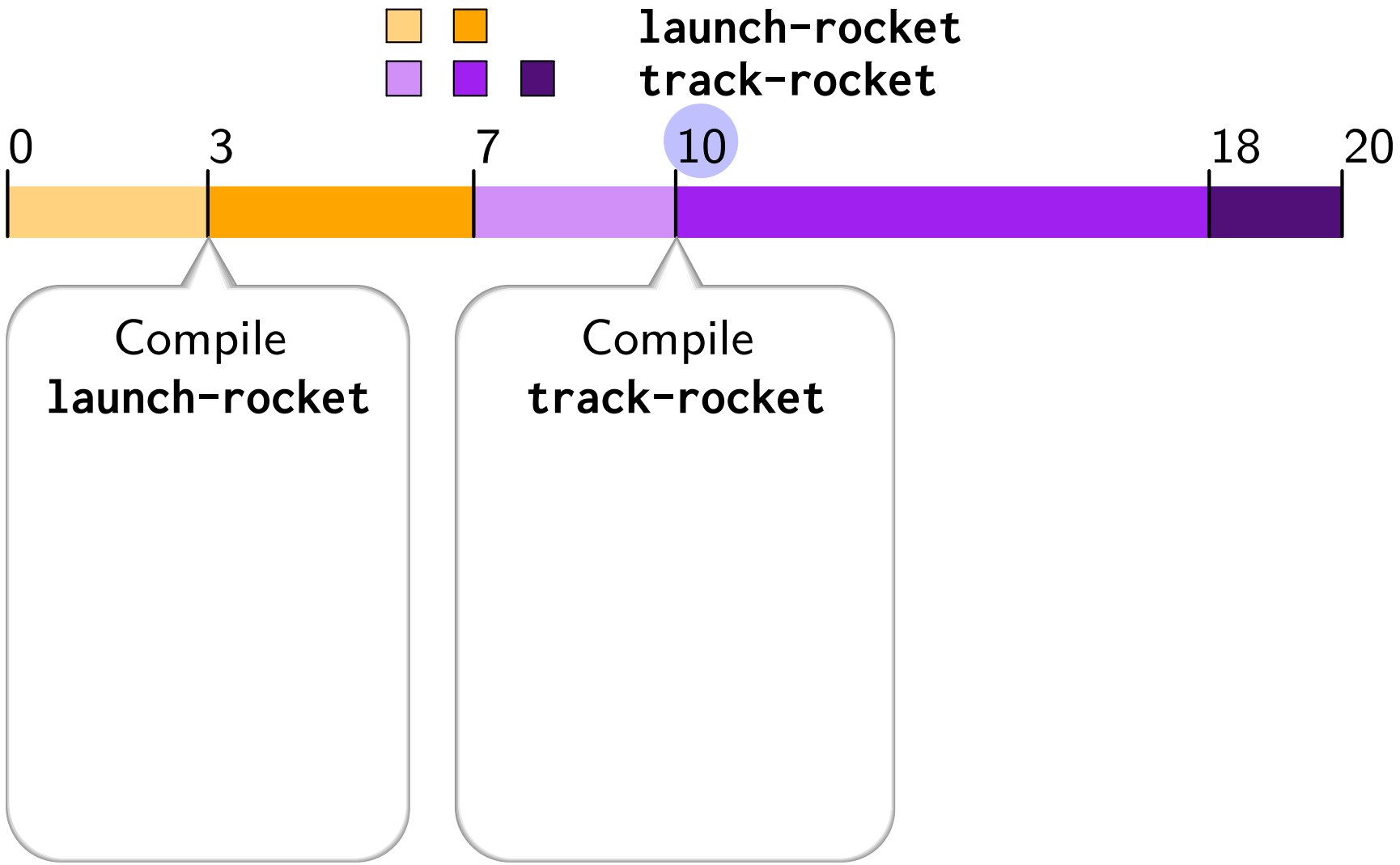
Profiling-Based Badness



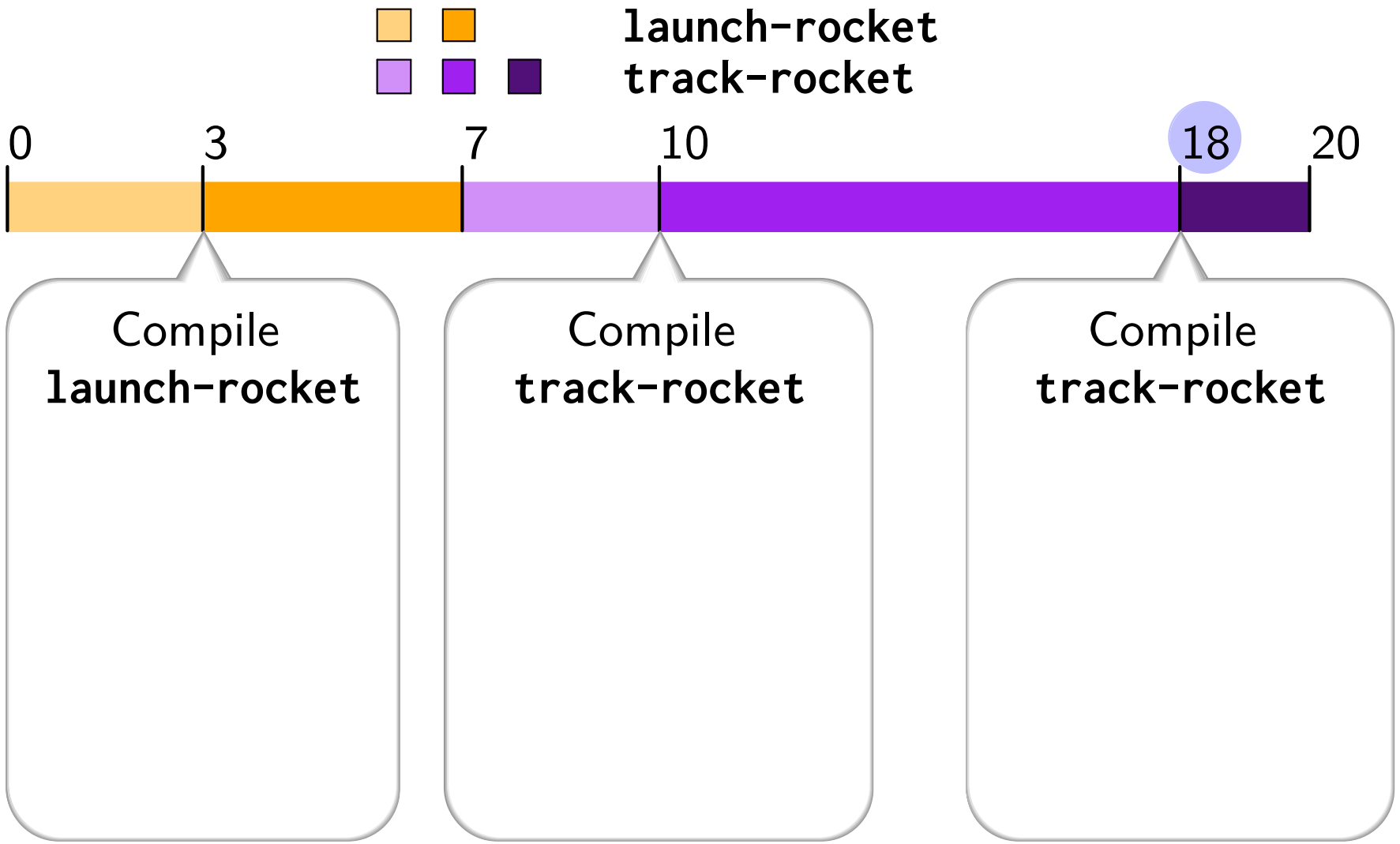
Profiling-Based Badness



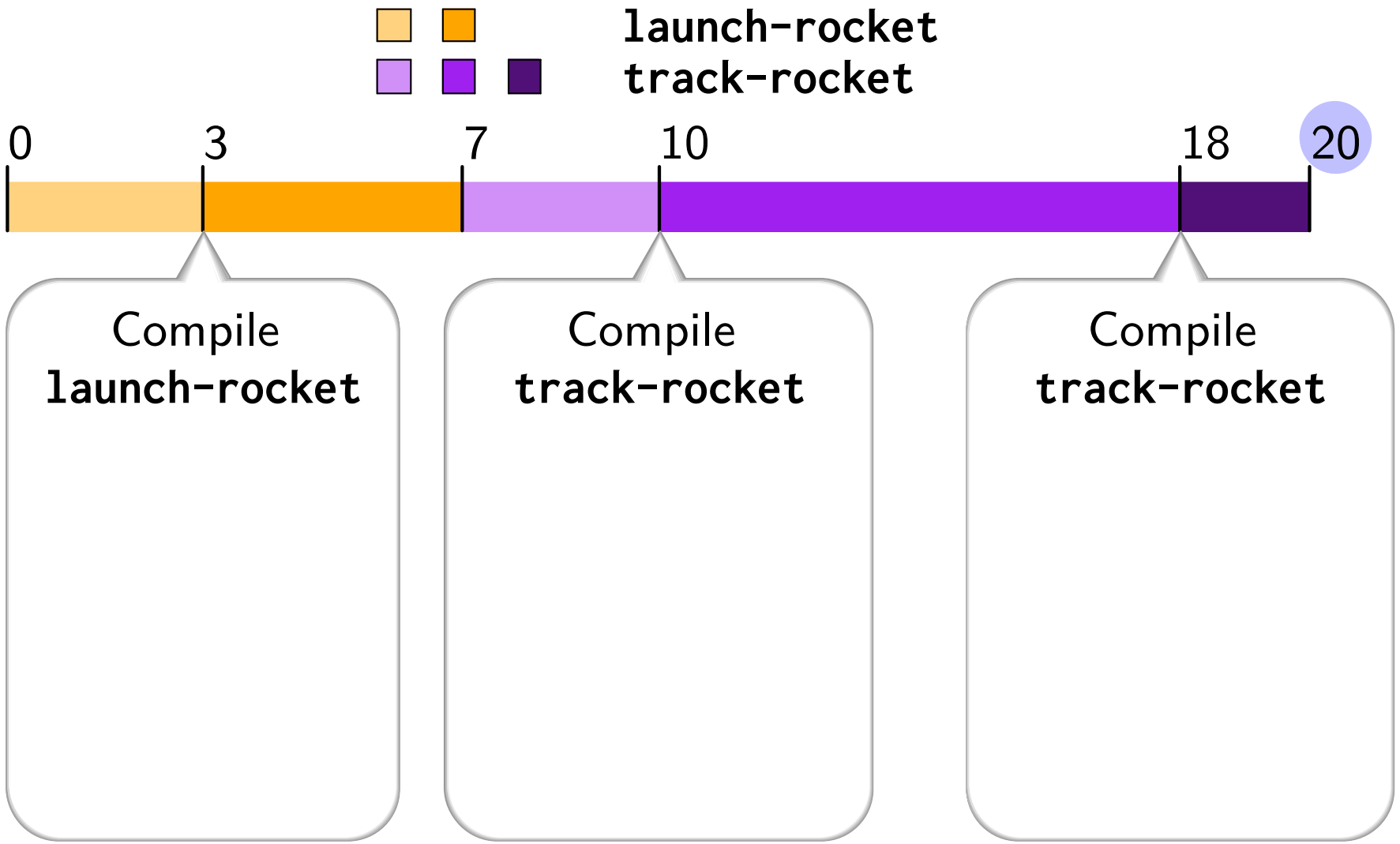
Profiling-Based Badness



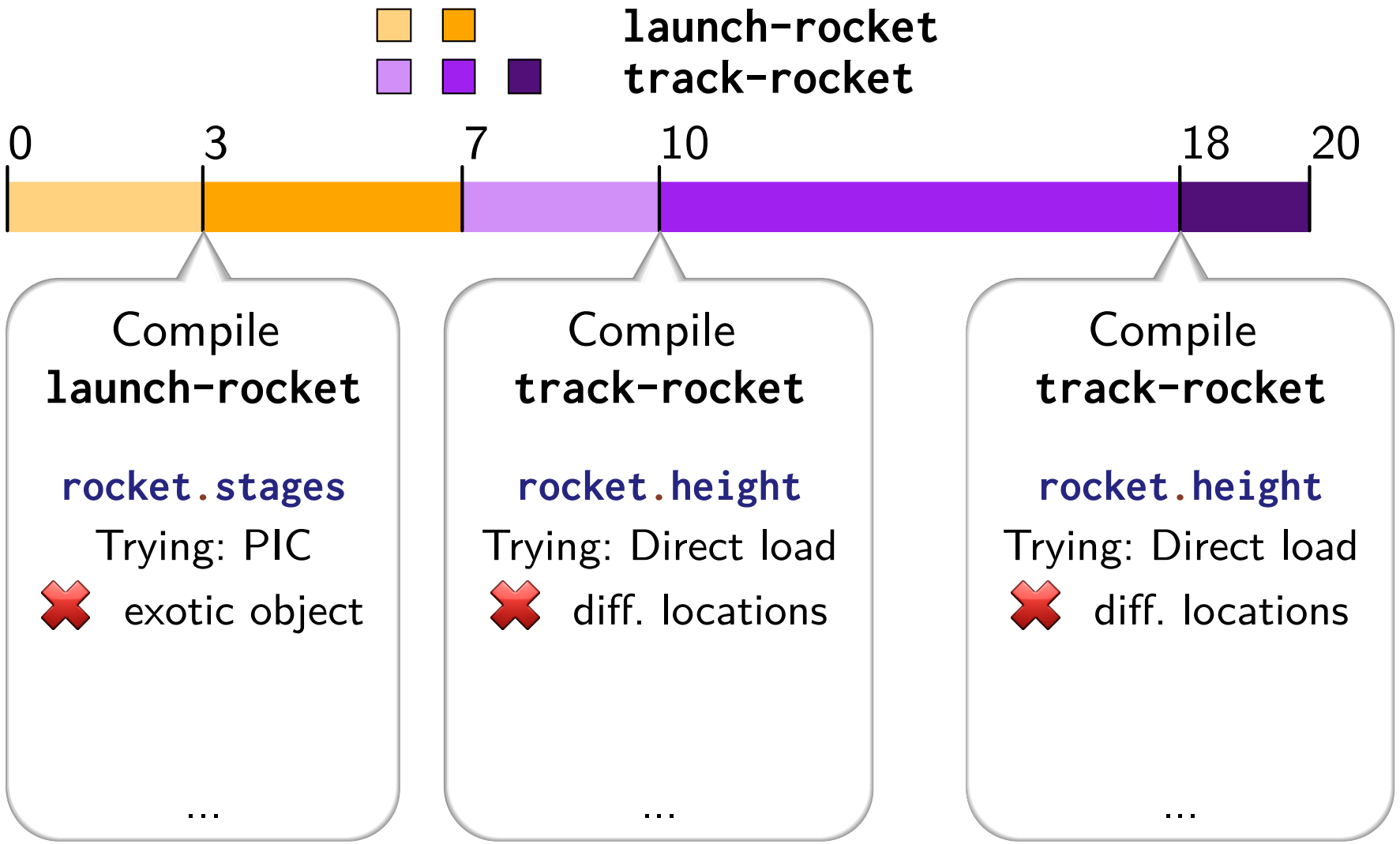
Profiling-Based Badness



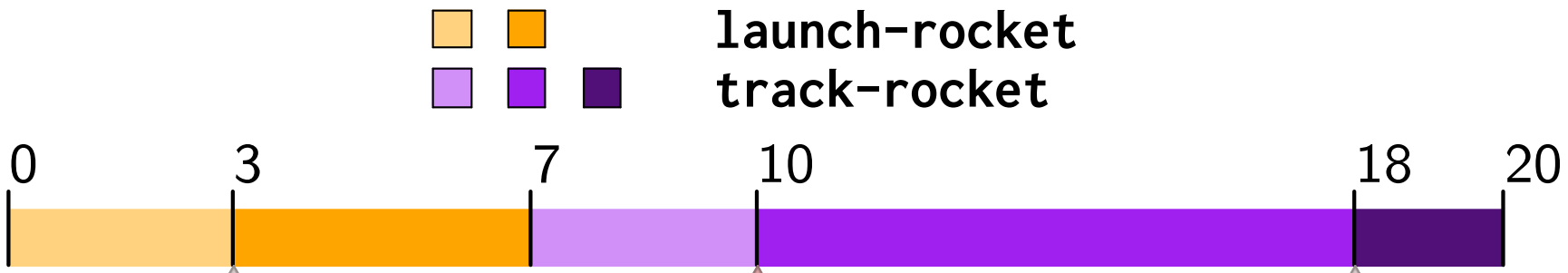
Profiling-Based Badness



Profiling-Based Badness



Profiling-Based Badness



Compile
`launch-rocket`

`rocket.stages`

Trying: PIC

✖ exotic object

Badness = 7 - 3
= 4

...

Compile
`track-rocket`

`rocket.height`

Trying: Direct load

✖ diff. locations

Badness = 18 - 10
= 8

...

Compile
`track-rocket`

`rocket.height`

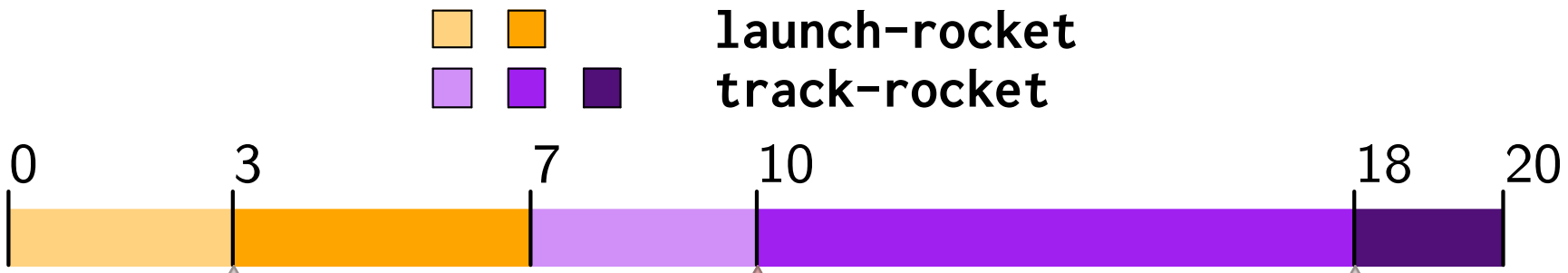
Trying: Direct load

✖ diff. locations

Badness = 20 - 18
= 2

...

Temporal Merging



Compile
launch-rocket

rocket.stages
Trying: PIC
✖ exotic object
Badness = 7 - 3
= 4
...

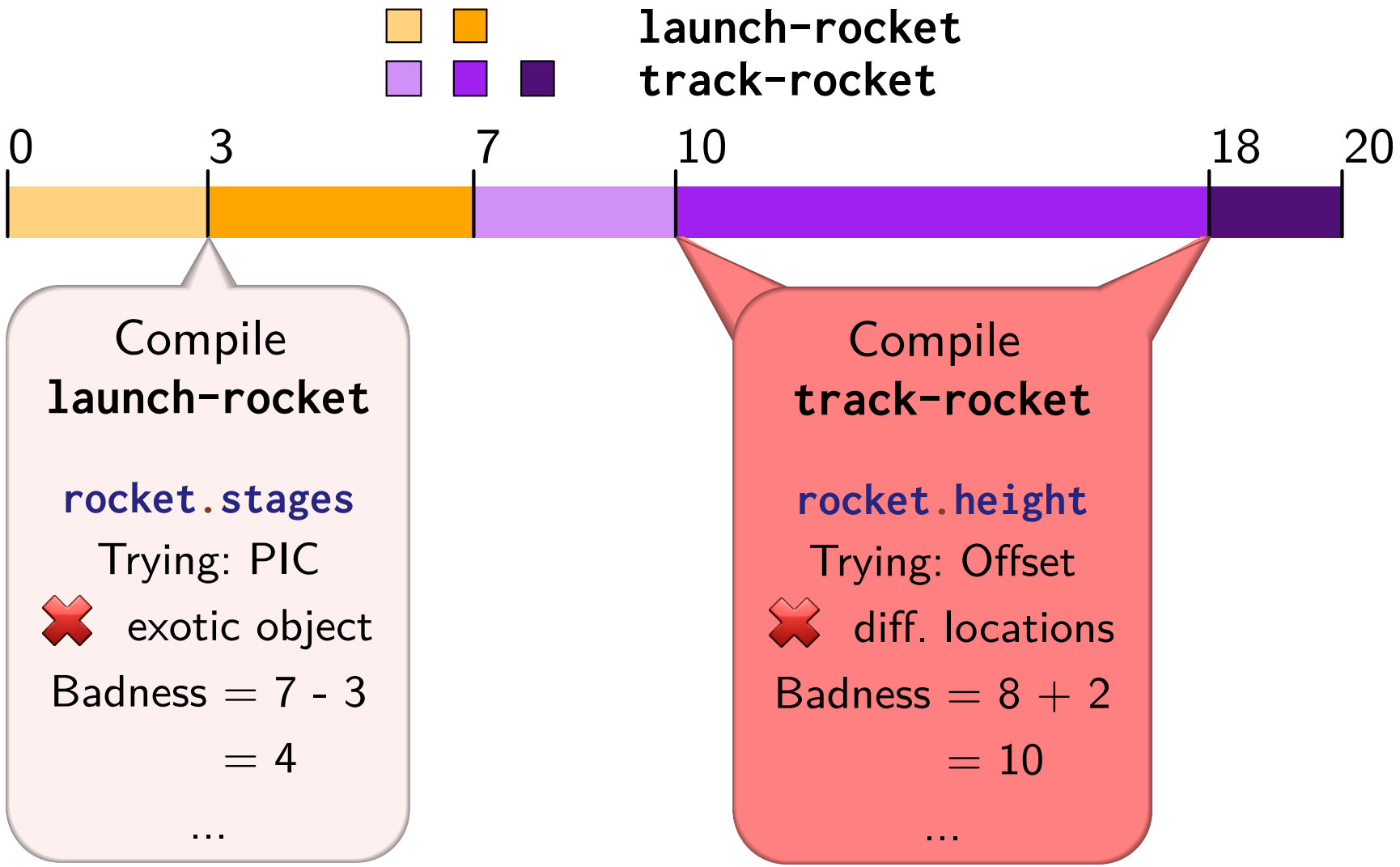
Compile
track-rocket

rocket.height
Trying: Direct load
✖ diff. locations
Badness = 18 - 10
= 8
...

Compile
track-rocket

rocket.height
Trying: Direct load
✖ diff. locations
Badness = 20 - 18
= 2
...

Temporal Merging



Further Sightseeing

(In the Paper)

- Solution Site Inference
- Same-Property Analysis
- By-Solution Merging
- By-Constructor Merging

} OO

- Profiler-Driven Instrumentation
- Profiling-Based Badness
- Temporal Merging

} JIT

- Irrelevant Failure Pruning
- Partial Success Shortcircuiting

How well does it work?

Hypothesis: Coaching improves performance

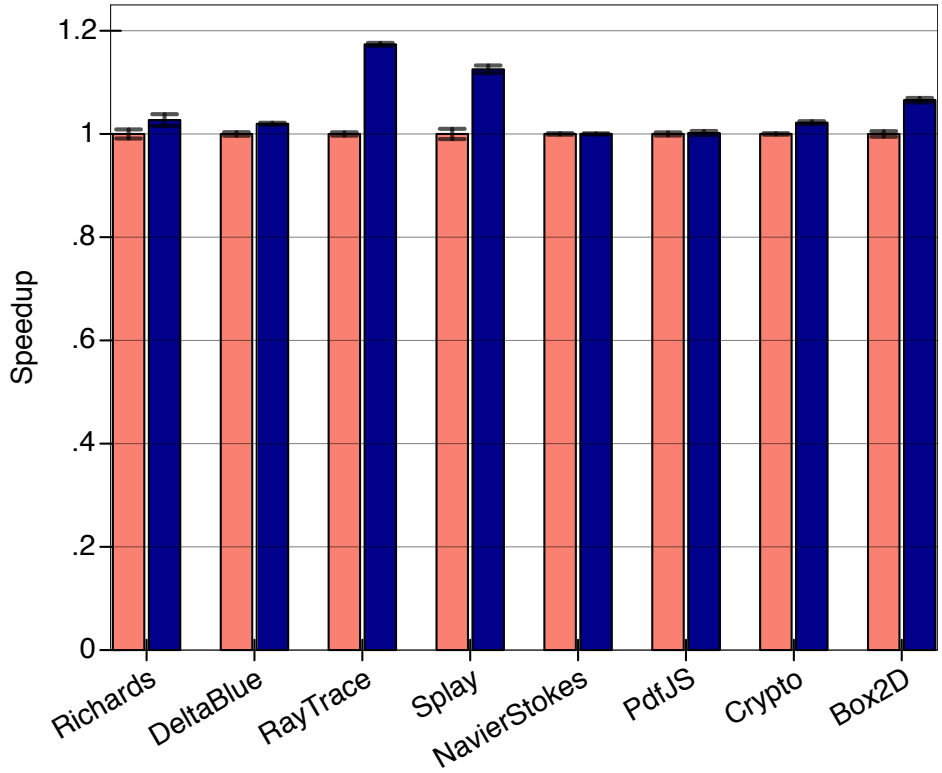
Baseline: Non-optimized

Coached: Followed recommendations (Minutes of work)

Speedup, higher is better

Experiment

- Take Octane benchmarks
- Run the coach
- Follow recommendations
- Measure performance impact (Octane score)

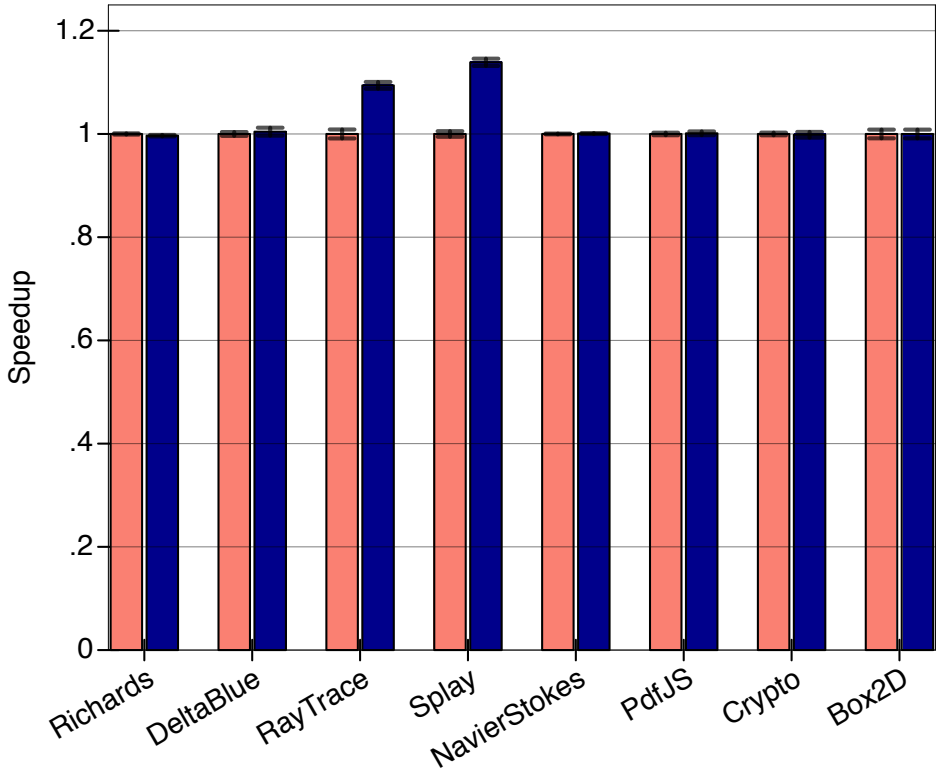


Hypothesis: Coaching improves performance

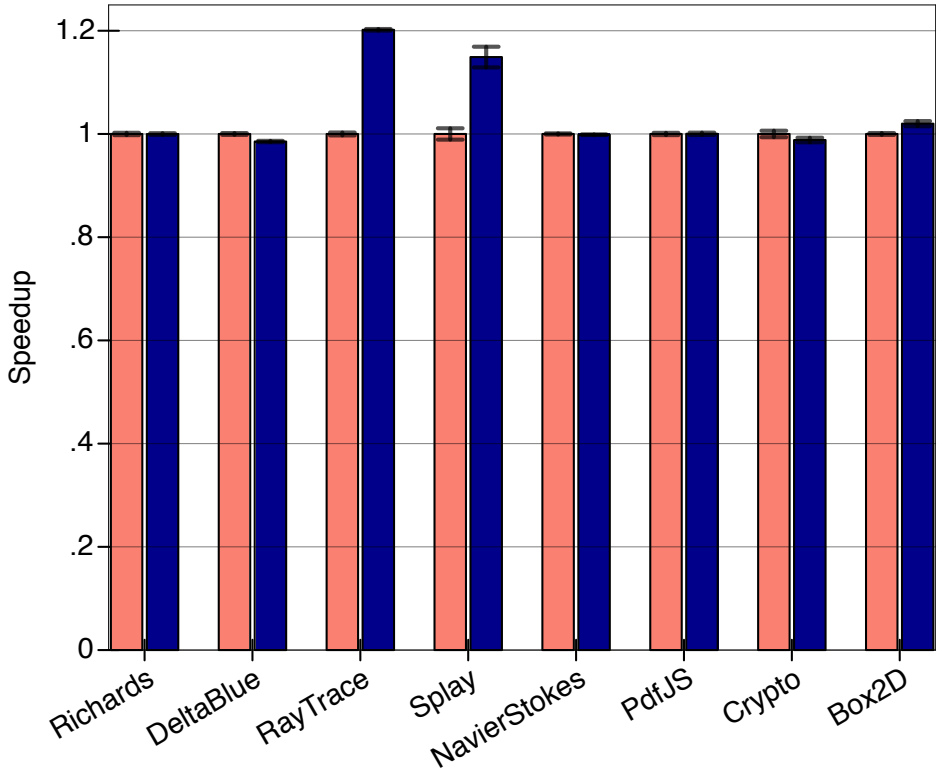
Baseline: Non-optimized

Coached: Followed recommendations (Minutes of work)

Speedup, higher is better

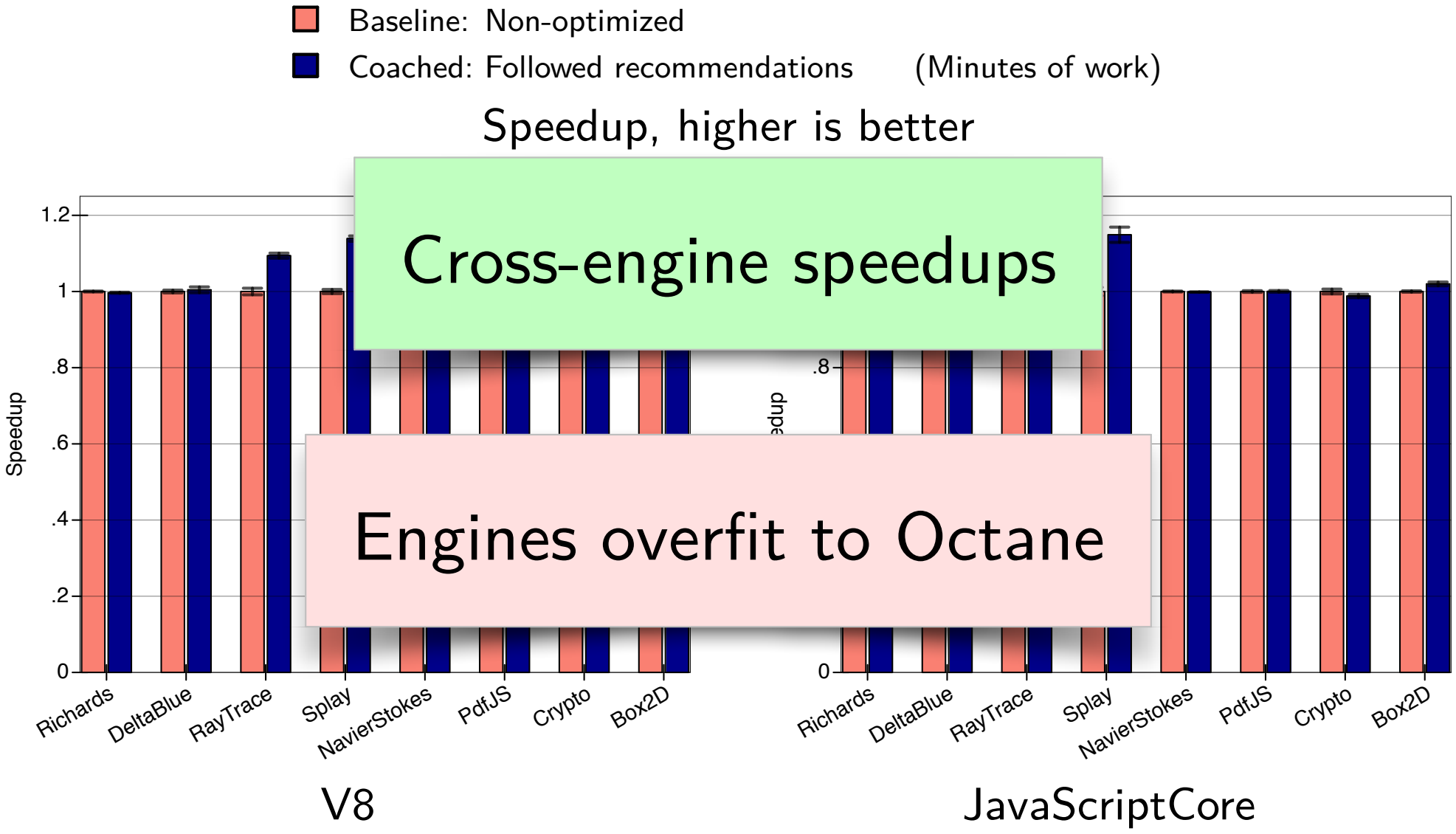


V8



JavaScriptCore

Hypothesis: Coaching improves performance



Hypothesis: Recommendations are low-effort

Benchmark	Total LOC	LOC Added	LOC Deleted	LOC Edited
Richards	538	1	5	0
DeltaBlue	881	12	6	24
RayTrace	903	10	11	0
Splay	422	3	3	0
NavierStokes	415	0	0	4
PdfJS	33,053	2	1	0
Crypto	1,698	2	0	1
Box2D	10,970	8	0	0

← At most 42 LOC!

Simple, mechanical changes!

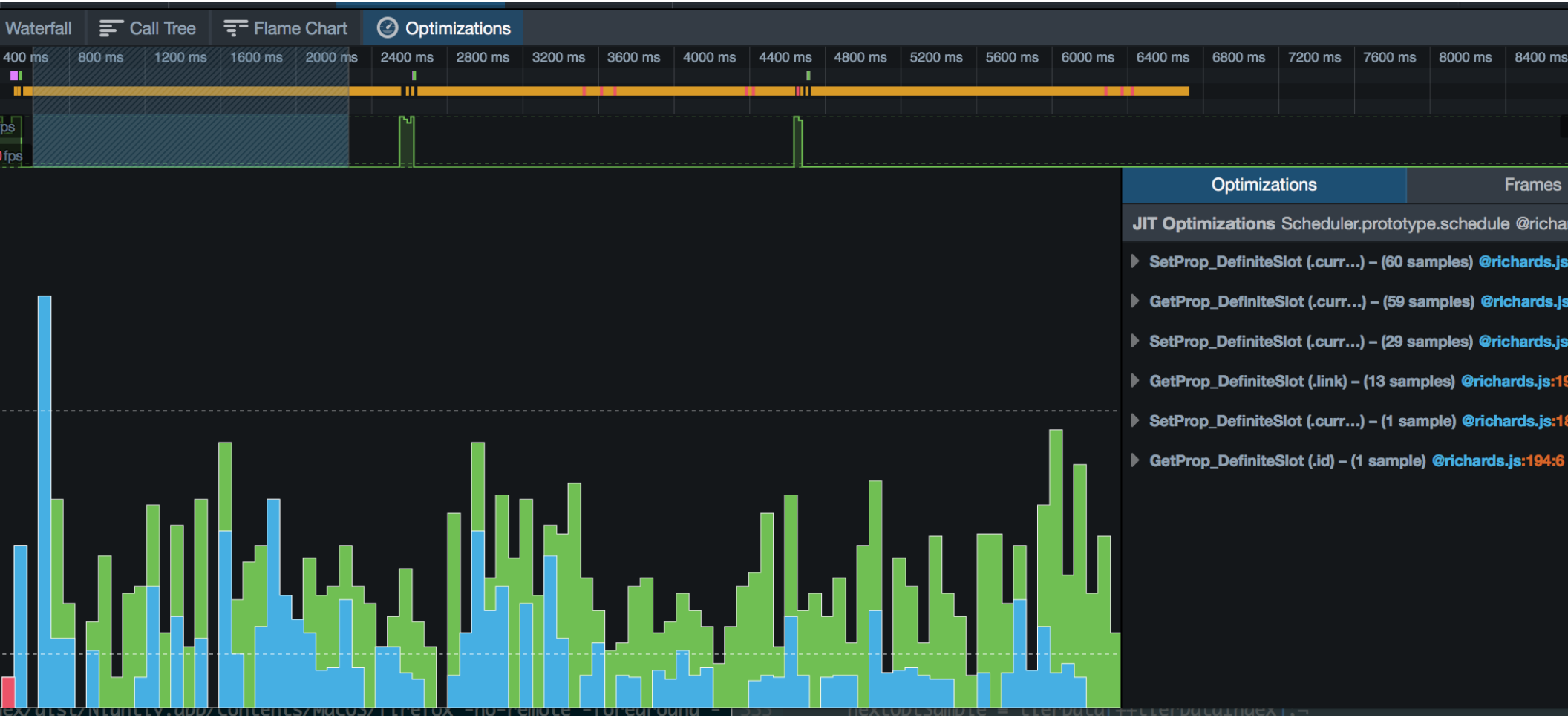
Further Sightseeing

(In the Paper)

- Recommendation quality
- Discussion of individual recommendations

Wrapping Up

Coming soon to a browser near you



This Talk

Coaching works

- beyond Racket
- with JIT compilation
- with OO optimizations

The Coaching Philosophy

Compilers are great, but they can fail

Compilers gather tons of information

Liberate it, and show it to programmers!

They may succeed where compilers fail

The Coaching Philosophy

_____ are great, but they can fail
(noun)

_____ gather tons of information
(noun)

Liberate it, and show it to programmers!

They may succeed where _____ fail
(noun)

The Coaching Philosophy

Runtimes (noun) are great, but they can fail

Runtimes (noun) gather tons of information

Liberate it, and show it to programmers!

They may succeed where Runtimes (noun) fail

The Coaching Philosophy

OSes (noun) are great, but they can fail

OSes (noun) gather tons of information

Liberate it, and show it to programmers!

They may succeed where OSes (noun) fail

The Coaching Philosophy

? are great, but they can fail
(noun)

? gather tons of information
(noun)

Liberate it, and show it to programmers!

They may succeed where ? fail
(noun)

The Coaching Philosophy

? (noun) are great, but they can fail

? (noun) gather tons of information

Liberate it, and show it to programmers!

They may succeed where ? (noun) fail

Thank you!