

Optimization Coaching for JavaScript

Vincent St-Amour

PLT @ Northwestern University

Shu-yu Guo

Mozilla Research

ECOOP 2015 — July 9th, 2015

```

#lang typed/racket/base
(require racket/math racket/flonum
 (except-in racket/flonum fl->fx fl->f)
 "flonum.rkt"
 "flonum-struct.rkt")
(provide flomap-flip-horizontal flomap-flip-vertical flomap-transpose
 flomap-cw-rotate flomap-ccw-rotate
 (struct-out invertible-2d-function) flomap-transform
 flomap-transform)
(: flomap-flip-horizontal (flomap -> flomap))
(define (flomap-flip-horizontal fm)
 (match-define (flomap vs c w h) fm)
 (define w1 (fx- w 1))
 (inline-build-flomap c w h (lambda (k x y)
 (unsafe-flvector-ref vs (coords->index c w k (fx- w1 x) y))))))
(define (flomap-flip-vertical fm)
 (match-define (flomap vs c w h) fm)
 (define h1 (fx- h 1))
 (inline-build-flomap c w h (lambda (k x y)
 (unsafe-flvector-ref vs (coords->index c w k x (fx- h1 y) y))))))
(define (flomap-transpose fm)
 (match-define (flomap vs c w h) fm)
 (inline-build-flomap c h w (lambda (k x y)
 (unsafe-flvector-ref vs (coords->index c w k x y))))))
(define (flomap-cw-rotate fm)
 (match-define (flomap vs c w h) fm)
 (define h1 (fx- h 1))
 (inline-build-flomap c h w (lambda (k x y)
 (unsafe-flvector-ref vs (coords->index c w k (fx- h1 y) x))))))
(define (flomap-ccw-rotate fm)
 (match-define (flomap vs c w h) fm)
 (define w1 (fx- w 1))
 (inline-build-flomap c w h (lambda (k x y)
 (unsafe-flvector-ref vs (coords->index c w k y (fx- w1 x) y))))))
(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))

```

```

#lang typed/racket/base
(require racket/flonum
 (except-in racket/flonum fl->fx fl->f)
 racket/math racket/math
 "flonum.rkt"
 "flonum-struct.rkt"
 "flonum-stats.rkt")
(provide flomap-lift flomap-lift2 flomap-lift-helper flomap-lift-helper2
 flomag fmagb fmagr fmagin fmagc fmagt flomag fmagp fmagtr fmagin fmagcos fmagt
 forward flvector flscaling fltruncate flzero
 fm- fm- fm/ fm/ fmin fmax
 flomap-normalize flomap-multiply-alpha flomap-divide-alpha)
; =====
: Unary
(: flomap-lift-helper : (flomat -> flomat) -> (flomap -> flomap))
(define (flomap-lift-helper f)
 (lambda (A : (Flonum -> flomap))
 (match-define (flomap vs c w h) fm)
 (flomap (inline-build-flvector (* c w h) (lambda (i) (f (unsafe-flvector-ref vs i))))
 c w h)))
(: flomap-lift ((flomat -> Real) -> (flomap -> flomap))
 (define (flomap-lift op)
 (flomap-lift-helper (lambda (x) (real->double-flonum (op x)))))
 (define fmagb (flomap-lift-helper abs))
 (define fmagr (flomap-lift-helper exp))
 (define fmagin (flomap-lift-helper sin))
 (define fmagc (flomap-lift-helper cos))
 (define fmagt (flomap-lift-helper tan))
 (define flomag (flomap-lift-helper log))
 (define fmagp (flomap-lift-helper exp))
 (define flmagtr (flomap-lift-helper atan))
 (define fmagin (flomap-lift-helper asin))
 (define fmagcos (flomap-lift-helper acos))
 (define fmagt (flomap-lift-helper atan))
 (define forward (flomap-lift-helper round))
 (define flvector (flomap-lift-helper vector))
 (define flscaling (flomap-lift-helper ceiling))
 (define fltruncate (flomap-lift-helper truncate))

```

```

#lang typed/racket/base
(require racket/flonum
 (except-in racket/flonum fx->fl fl->fx)
 racket/math racket/math
 "flonum.rkt"
 "flonum.rkt")
(provide deep-flomap deep-flomap? deep-flomap-argb deep-flomap-z
 deep-flomap-scale-z 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-trin 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-join 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-ctb-superimpose
 deep-flomap-rt-superimpose deep-flomap-rc-superimpose deep-flomap-rtb-superimpose
 deep-flomap-vl-append deep-flomap-vc-append deep-flomap-vr-append
 deep-flomap-hl-append deep-flomap-hc-append deep-flomap-hb-append)
(struct: deep-flomap (argb : flomap) [z : flomap])
#transparent
#guard
(lambda (A (argb-fm z-fm name)
 (match-define (flomap _ 4 w h) argb-fm)
 (match-define (flomap _ 1 zw zh) z-fm)
 (unless (and (= zw w) (= zh h))
 (error "deep-flomap
 "expected flomaps of equal dimension; given dimensions ~e~w and ~e~zh~" 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))

```

Once upon a time in the Racket community...

```

(cond [(c . . . fx c 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
 (lambda (k x y)
 (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)))]))
]
(else
 (define-values (v-min v-max) (flomap-extreme-values fm))
 (define v-size (- v-max v-min))
 (let* [(f (fm (fm- fm v-min)
 (fx (if (= v-size . 0) 0) fm (fm/ fm v-size)))
 fm))
 (define fmdiv/zero
 (flomap-lift-helper2 'fmdiv/zero (lambda (x y) (if (y = . 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
(require racket/match
         (except-in racket/fixnum fl->fx fx->fl)
         racket/match racket/math)
"flomap.rkt"
"flomap-struct.rkt"
"flomap-normalize.rkt"
"flomap-struct.rkt"
"flomap-stats.rkt"
(provide flomap-lift flomap-lift2 flomap-lift-helper flomap-lift-helper2
         fmax fmins fmsqr fmsin fcosn fctan flog fmsq fmsqr fmsin fmaxn fctan
         forward-refloor fceiling ftruncate fzero
         fx fx- fx- fx/ fmin fmax
         flomap-normalize flomap-multiply-alpha flomap-divide-alpha)
; =====
(: Unary
  ( flomap-lift-helper : (float -> float) -> (flomap -> flomap))
  (define (flomap-lift-helper f)
    (lambda (A) (fx : flomap)
              (match-define (flomap vs c w h) A
                            (flomap (inline-build-vecvector (+ c w h) (A i) (f (unsafe-vecvector-ref vs i))))
                                    c w h)))
(: flomap-lift (flomap -> Real) -> (flomap -> flomap))
(define (flomap-lift op)
  (flomap-lift-helper (lambda (x) (real-double-flonum (op x)))))
(define fmax (flomap-lift-helper +))
(define fmin (flomap-lift-helper -))
(define fmsqr (flomap-lift-helper *))
(define fmsin (flomap-lift-helper sin))
(define fcosn (flomap-lift-helper cos))
(define fctan (flomap-lift-helper tan))
(define flog (flomap-lift-helper log))
(define fmsq (flomap-lift-helper exp))
(define fmsqr (flomap-lift-helper exp2))
(define fmsin (flomap-lift-helper exp10))
(define fmaxn (flomap-lift-helper apply))
```

```
(: flomap-tran
  (define flomap
    (case-lambda
      [(fx t)
       (match-define (define x
                       (define y
                         (let [y2]
                           (when (
                               (let:
                                (co
```

```
#lang typed/racket/base
(require racket/flonum
         (except-in racket/fixnum fl->fx fx->fl)
         racket/match racket/math)
"flomap.rkt"
"flomap-struct.rkt"
"flomap-normalize.rkt"
"flomap-struct.rkt"
"flomap-stats.rkt"
(provide flomap-lift flomap-lift2 flomap-lift-helper flomap-lift-helper2
         fmax fmins fmsqr fmsin fcosn fctan flog fmsq fmsqr fmsin fmaxn fctan
         forward-refloor fceiling ftruncate fzero
         fx fx- fx- fx/ fmin fmax
         flomap-normalize flomap-multiply-alpha flomap-divide-alpha)
; =====
(: Unary
  ( flomap-lift-helper : (float -> float) -> (flomap -> flomap))
  (define (flomap-lift-helper f)
    (lambda (A) (fx : flomap)
              (match-define (flomap vs c w h) A
                            (flomap (inline-build-vecvector (+ c w h) (A i) (f (unsafe-vecvector-ref vs i))))
                                    c w h)))
(: flomap-lift (flomap -> Real) -> (flomap -> flomap))
(define (flomap-lift op)
  (flomap-lift-helper (lambda (x) (real-double-flonum (op x)))))
(define fmax (flomap-lift-helper +))
(define fmin (flomap-lift-helper -))
(define fmsqr (flomap-lift-helper *))
(define fmsin (flomap-lift-helper sin))
(define fcosn (flomap-lift-helper cos))
(define fctan (flomap-lift-helper tan))
(define flog (flomap-lift-helper log))
(define fmsq (flomap-lift-helper exp))
(define fmsqr (flomap-lift-helper exp2))
(define fmsin (flomap-lift-helper exp10))
(define fmaxn (flomap-lift-helper apply))
```

```
#lang typed/racket/base
(require racket/flonum
         (except-in racket/fixnum fx->fl fl->fx)
         racket/match racket/math)
"flomap.rkt"
"flomap-struct.rkt"
"flomap-normalize.rkt"
"flomap-struct.rkt"
"flomap-stats.rkt"
(provide flomap-lift flomap-lift2 flomap-lift-helper flomap-lift-helper2
         fmax fmins fmsqr fmsin fcosn fctan flog fmsq fmsqr fmsin fmaxn fctan
         forward-refloor fceiling ftruncate fzero
         fx fx- fx- fx/ fmin fmax
         flomap-normalize flomap-multiply-alpha flomap-divide-alpha)
; =====
(: Unary
  ( flomap-lift-helper : (float -> float) -> (flomap -> flomap))
  (define (flomap-lift-helper f)
    (lambda (A) (fx : flomap)
              (match-define (flomap vs c w h) A
                            (flomap (inline-build-vecvector (+ c w h) (A i) (f (unsafe-vecvector-ref vs i))))
                                    c w h)))
(: flomap-lift (flomap -> Real) -> (flomap -> flomap))
(define (flomap-lift op)
  (flomap-lift-helper (lambda (x) (real-double-flonum (op x)))))
(define fmax (flomap-lift-helper +))
(define fmin (flomap-lift-helper -))
(define fmsqr (flomap-lift-helper *))
(define fmsin (flomap-lift-helper sin))
(define fcosn (flomap-lift-helper cos))
(define fctan (flomap-lift-helper tan))
(define flog (flomap-lift-helper log))
(define fmsq (flomap-lift-helper exp))
(define fmsqr (flomap-lift-helper exp2))
(define fmsin (flomap-lift-helper exp10))
(define fmaxn (flomap-lift-helper apply))
```

1 hour later...

```
(: flomap-tran
  (define flomap
    (case-lambda
      [(fx t)
       (match-define (define x
                       (define y
                         (let [y2]
                           (when (
                               (let:
                                (co
```

```
(: flomap-tran
  (define flomap
    (case-lambda
      [(fx t)
       (match-define (define x
                       (define y
                         (let [y2]
                           (when (
                               (let:
                                (co
```

```
(: flomap-tran
  (define flomap
    (case-lambda
      [(fx t)
       (match-define (define x
                       (define y
                         (let [y2]
                           (when (
                               (let:
                                (co
```



```
#lang typed/racket
(require racket/match
         (except-in r
                    "flonum.rkt")
         "flonop-struct")
(provide flonop-flip
         flonop-cw-r
         (struct-out
          transform-co
          flonop-trans)
         C: flonop-flip-horiz)
(define (flonop-flip
        (match-define (flon
                       (define w1 (f* w
                                       (inline-build-flon
                                        (define (flonop-flip
                                                (match-define (flon
                                                                (define h1 (f* h
                                                                                          (inline-build-flon
                                                                                           (define (flonop-trans
                                                                                                   (match-define (flon
                                                                                                       (inline-build-flon
                                                                                                        (define (flonop-cw-r
                                                                                                                (match-define (flon
                                                                                                                    (define h1 (f* h
                                                                                                                                                (inline-build-flon
                                                                                                                                                 (define (flonop-cw-r
                                                                                                                                                                             (match-define (flon
```

```

#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
            ())

```

```

d' -exec rm
ages/private
ages
```

20 hours later...

```

[let (x-min (r
            [x-max (r
                  [y-min (r
                        [y-max (r
                              (match-define
                               (match-define
                                (define int-x
                                 (define int-y
                                  (define int-z
                                   (define new-w
                                    (define new-h
                                     (define x-offx
                                      (define y-offy
                                       (inline-build-
                                        c new-w new-h
                                        (λ (x y z)
                                         (define-val
                                          (flonop-bil
```

```

#<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
    ...
```

map))

```
#lang typed/racket/base
(require racket/math racket/fixnum
 (except-in racket/fixnum fl->fx fl->f))
"flonum.rkt"
"flonum-struct.rkt"
"flonum-stats.rkt"
(provide flomap-flip-horizontal flomap-flip-vertical flomap-transpose
 flomap-cw-rotate flomap-ccw-rotate
 (struct-out invertible-2d-function) flomap-transform
 flomap-transform)
(: flomap-flip-horizontal (flomap -> flomap))
(define (flomap-flip-horizontal fm)
 (match-define (flomap vs c w h) fm)
 (define w1 (fx - w))
 (inline-build-flomap c w h (lambda (k x y z)
 (unsafe-fvectoref vs (coords->index c w k (fx- w1 x) y))))))
(define (flomap-flip-vertical fm)
 (match-define (flomap vs c w h) fm)
 (define h1 (fx - h))
 (inline-build-flomap c w h (lambda (k x y z)
 (unsafe-fvectoref vs (coords->index c w k x (fx- h1 y) z))))))
(define (flomap-transpose fm)
 (match-define (flomap vs c w h) fm)
 (inline-build-flomap c h w (lambda (k x y z)
 (unsafe-fvectoref vs (coords->index c w k x y (fx- h1 y) z))))))
(define (flomap-cw-rotate fm)
 (match-define (flomap vs c w h) fm)
 (define h1 (fx - h))
 (inline-build-flomap c h w (lambda (k x y z)
 (unsafe-fvectoref vs (coords->index c w k (fx- h1 y) x))))))
(define (flomap-ccw-rotate fm)
 (match-define (flomap vs c w h) fm)
 (define w1 (fx - w))
 (inline-build-flomap c w h (lambda (k x y z)
 (unsafe-fvectoref vs (coords->index c w k y (fx- w1 x) z))))))
(struct: invertible-2d-function ([ f : (Flonum Flonum -> (values Flonum Flonum))]
 [ g : (Flonum Flonum -> (values Flonum Flonum))]))
(: 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 (lambda (x (x : Flonum) y : Flonum)
 (let-values (((x y) (f1 x y))
 ((x1 y1) (f2 x y)))
 (let-values (((x y) (g1 x y))
 ((x2 y2) (g2 x1 y1)))
 (values x2 y2))))))
(: flomap-transform (case-> (flomap flomap-transform -> flomap)
 (flomap flomap-transform Real Real Real -> flomap)))
(define flomap-transform
 (case-lambda
 [(fx 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 (f : Integer R)]
 (when (y f x - h)
 (let [x-loop : Void (f : Integer R)]
 (cond [(x f x - c)
 (define-values (new-x new-y) (f (+ 0.5 (fx->fl x) (+ 0.5 (fx->fl x))))
 (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 ))])
 (y-loop (fx+ y )))))]))
 (flomap-transform fm t x-min x-max y-min y-max))
 [(fx 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
 (lambda (k x y z)
 (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/flonum fl->fx fl->f))
"flonum.rkt"
"flonum-struct.rkt"
"flonum-stats.rkt"
(provide flomap-lift flomap-lift2 flomap-lift-helper flomap-lift-helper2
 flomap-fmaabs flomap-fmin flomap-focus flomap-fmag flomap-fmagrt flomap-fmax flomap-fmax
 flomap-forward flomap-faciling flomap-fruncate flomap-zero
 flomap-fm flomap-fm flomap-fmin flomap-fmax
 flomap-normalize flomap-multiply-alpha flomap-divide-alpha)
"flomap-lift-helper.rkt"
"flomap-lift-helper2.rkt"
"flomap-lift-helper2.rkt"
(define (flomap-lift-helper f)
 (lambda (A (A : (Flonum)
 (match-define (flomap vs c w h) fm)
 (flomap (inline-build-flomap (+ c w h) (A (lambda (f) (unsafe-fvectoref vs i))))
 (lambda (x y z)
 (let-values (((x y) (f x y))
 ((x1 y1) (f x1 y1)))
 (let-values (((x y) (f x y))
 ((x2 y2) (f x1 y1)))
 (values x2 y2)))))))))
(define (flomap-lift2 f)
 (lambda (A (A : (U Real Flonum)
 (let [and (real? fml) (real? fmd)]
 (error name "expected at least one flomap argument, given ~e and ~e" fml fmd))]
 [[real? fml] (let (fml (real->double-flonum fml))
 ((flomap-lift-helper (A v) (f v)) fmd))]
 [[real? fmd] (let (fmd (real->double-flonum fmd))
 ((flomap-lift-helper (A v) (f v)) fml))]
 [else
 (match-define (flomap vs1 c1 w1 h1) fml)
 (match-define (flomap vs2 c2 w2 h2) fmd)
 (cond
 [(not (and (= w1 w2) (= h1 h2)))]
 (error name "expected same-size flomaps; given sizes ~e~e and ~e~e" w1 w2 h1 h2)]
 [(= c1 cx) (define n (+ c1 w1 h1))
 (define res-vs (make-fvectoref 0)
 (flomap (inline-build-flomap n (A (lambda (f) (unsafe-fvectoref vs1 i)
 (unsafe-fvectoref vs2 i))))
 c1 w1 h1)
 (= c1 1) (inline-build-flomap
 c2 w h
 (lambda (k x y i) (if (unsafe-fvectoref vs1 (coords->index 1 w 0 x y))
 (unsafe-fvectoref vs2 i))))))
 (= c2 1) (inline-build-flomap
 c1 w h
 (lambda (k x y i) (if (unsafe-fvectoref vs1 i)
 (unsafe-fvectoref 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 w1 h1 c2 w2 h2)]))])
 (flomap-lift2 fml fmd)))
(define flomap-lift-helper2 name f)
 (let (let: ()
 (A: ((fml : (U Real Flonum)) [fmd : (U Real Flonum)])
 (cond
 [and (real? fml) (real? fmd)]
 (error name "expected at least one flomap argument, given ~e and ~e" fml fmd))]
 [[real? fml] (let (fml (real->double-flonum fml))
 ((flomap-lift-helper (A v) (f v)) fmd))]
 [[real? fmd] (let (fmd (real->double-flonum fmd))
 ((flomap-lift-helper (A v) (f v)) fml))]
 [else
 (match-define (flomap vs1 c1 w1 h1) fml)
 (match-define (flomap vs2 c2 w2 h2) fmd)
 (cond
 [(not (and (= w1 w2) (= h1 h2)))]
 (error name "expected same-size flomaps; given sizes ~e~e and ~e~e" w1 w2 h1 h2)]
 [(= c1 cx) (define n (+ c1 w1 h1))
 (define res-vs (make-fvectoref 0)
 (flomap (inline-build-flomap n (A (lambda (f) (unsafe-fvectoref vs1 i)
 (unsafe-fvectoref vs2 i))))
 c1 w1 h1)
 (= c1 1) (inline-build-flomap
 c2 w h
 (lambda (k x y i) (if (unsafe-fvectoref vs1 (coords->index 1 w 0 x y))
 (unsafe-fvectoref vs2 i))))))
 (= c2 1) (inline-build-flomap
 c1 w h
 (lambda (k x y i) (if (unsafe-fvectoref vs1 i)
 (unsafe-fvectoref 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 w1 h1 c2 w2 h2)]))])
 (flomap-lift2 fml fmd)))

```

```
#lang typed/racket/base
(require racket/flonum
 (except-in racket/flonum fx->fl fl->f))
"flonum.rkt"
"flomap.rkt"
(provide deep-flomap deep-flomap2 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-min deep-flomap-pin
deep-flomap-lt-superimpose deep-flomap-lc-superimpose deep-flomap-lb-superimpose
deep-flomap-ct-superimpose deep-flomap-cs-superimpose deep-flomap-rt-superimpose
deep-flomap-rc-superimpose deep-flomap-cc-superimpose deep-flomap-rb-superimpose
deep-flomap-rl-superimpose deep-flomap-rl-superimpose deep-flomap-rr-superimpose
deep-flomap-bl-superimpose deep-flomap-bl-superimpose deep-flomap-br-superimpose
deep-flomap-br-superimpose
(struct: deep-flomap (argb : flomap) [z : flomap])
#transparent
#guard
(A (argb-fm z-fm name)
 (match-define (flomap _ 4 w h) argb-fm)
 (match-define (flomap _ 1 w h) 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 dfa)
 (define w (flomap-width (deep-flomap-argb dfa)))
 (with-asserts (w nonnegative-fixnum?))
 w)
(: deep-flomap-height (deep-flomap -> Nonnegative-Fixnum))
(define (deep-flomap-height dfa)
 (define h (flomap-height (deep-flomap-argb dfa)))
 (with-asserts (h nonnegative-fixnum?))
 h)
(: deep-flomap-z-min (deep-flomap -> flomap))
(define (deep-flomap-z-min dfa)
 (flomap-min-value (deep-flomap-z dfa)))
(: deep-flomap-z-max (deep-flomap -> flomap))
(define (deep-flomap-z-max dfa)
 (flomap-max-value (deep-flomap-z dfa)))
(: flomap-max-value (deep-flomap-z dfa) -> (values Nonnegative-Fixnum Nonnegative-Fixnum))
(define (deep-flomap-size dfa)
 (values (deep-flomap-width dfa) (deep-flomap-height dfa)))
(: deep-flomap-alpha (deep-flomap -> flomap))
(define (deep-flomap-alpha dfa)
 (flomap-ref-component (deep-flomap-argb dfa) 0))
(: deep-flomap-rgb (deep-flomap -> flomap))
(define (deep-flomap-rgb dfa)
 (flomap-drop-components (deep-flomap-argb dfa) 1))
; =====
: Z adjusters
(: deep-flomap-scale-z (deep-flomap (U Real Flonum) -> deep-flomap))
(define (deep-flomap-scale-z dfa z)
 (match-define (deep-flomap argb-fm z-fm) dfa)
 (deep-flomap argb-fm (fx z-fm z)))
(: deep-flomap-smooth-z (deep-flomap Real -> deep-flomap))
(define (deep-flomap-smooth-z dfa 0)
 (let [(f (exact->inexact 0))]
 (match-define (deep-flomap argb-fm z-fm) dfa)
 (define new-z-fm (flomap-blur z-fm 0))
 (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 0.5 everywhere alpha is 0.0)
(: deep-flomap-raise (deep-flomap (U Real Flonum) -> deep-flomap))
(define (deep-flomap-raise dfa z)
 (match-define (deep-flomap argb-fm z-fm) dfa)
 (define alpha-fm (deep-flomap-alpha dfa))
 (deep-flomap argb-fm (fx z-fm (fx alpha-fm z))))
(: deep-flomap-emboss (deep-flomap Real (U Real Flonum) -> deep-flomap))
(define (deep-flomap-emboss dfa xy-ant z-ant)
 (let [(f (/ xy-ant 3.0))]
 (define z-fm (flomap-normalize (deep-flomap-alpha dfa)))
 (define new-z-fm (flomap-normalize (fx z-fm z)))
 (deep-flomap argb-fm (fx z-fm (fx alpha-fm z))))))

```



```

#lang typed/racket/base
(require racket/math racket/flonum
 (except-in racket/flonum fl->fx fx->fl)
 "flonum.rkt"
 "flonop-struct.rkt")
"flonop-struct.rkt")
(provide flonop-flip-horizontal flonop-flip-vertical flonop-transpose
 flonop-cw-rotate flonop-ccw-rotate
 (struct-out invertible-2d-function) flonop-transform
 flonop-transform)
(: flonop-flip-horizontal (flonop -> flonop))
(define (flonop-flip-horizontal fa)
 (match-define (flonop vs c w h) fa)
 (define w1 (fx- w 1))
 (inline-build-flonop c w h (λ (k x y z)
 (unsafe-flvector-ref vs (coords->index c w k (fx- w1 x) y))))))
(define (flonop-flip-vertical fa)
 (match-define (flonop vs c w h) fa)
 (define h1 (fx- h 1))
 (inline-build-flonop c w h (λ (k x y z)
 (unsafe-flvector-ref vs (coords->index c w k x (fx- h1 y) z))))))
(define (flonop-transpose fa)
 (match-define (flonop vs c w h) fa)
 (inline-build-flonop c h w (λ (k x y z)
 (unsafe-flvector-ref vs (coords->index c w k x y z))))))
(define (flonop-cw-rotate fa)
 (match-define (flonop vs c w h) fa)
 (define h1 (fx- h 1))
 (inline-build-flonop c h w (λ (k x y z)
 (unsafe-flvector-ref vs (coords->index c w k (fx- h1 y) z))))))
(define (flonop-ccw-rotate fa)
 (match-define (flonop vs c w h) fa)
 (define w1 (fx- w 1))
 (inline-build-flonop c w h (λ (k x y z)
 (unsafe-flvector-ref vs (coords->index c w k y (fx- w1 x) z))))))
(struct: invertible-2d-function ((f : (Flonum Flonum -> (values Flonum Flonum))))

```

```

#lang typed/racket/base
(require racket/flonum
 (except-in racket/flonum fl->fx fx->fl)
 racket/math racket/math
 "flonum.rkt"
 "flonop-struct.rkt"
 "flonop-stats.rkt")
(provide flonop-lift flonop-lift2 flonop-lift-helper flonop-lift-helper2
 flonop-fmabs flonop-fmin flonop-fmax flonop-fmround flonop-fmceil flonop-fmtrunc flonop-fmzero
 flonop-fm- / flonop-fm/ flonop-fmmax
 flonop-normalize flonop-multiply-alpha flonop-divide-alpha)
; =====
: Unary
(: flonop-lift-helper : (flonop -> flonop) -> (flonop -> flonop))
(define (flonop-lift-helper f)
 (λ (fa : (flonop -> flonop))
 (match-define (flonop vs c w h) fa)
 (flonop (inline-build-flvector (* c w h) (λ (i) (f (unsafe-flvector-ref vs i))))
 c w h)))
(: flonop-lift ((flonum -> Real) -> (flonop -> flonop))
(define (flonop-lift op)
 (flonop-lift-helper (λ (x) (real->double-flonum (op x)))))
(define flonop (flonop-lift-helper -))
(define flonopabs (flonop-lift-helper abs))
(define flonopexp (flonop-lift-helper exp))
(define flonopln (flonop-lift-helper ln))
(define flonopcos (flonop-lift-helper cos))
(define flonoptan (flonop-lift-helper tan))
(define flonoplog (flonop-lift-helper log))
(define flonopexp2 (flonop-lift-helper exp2))
(define flonopln2 (flonop-lift-helper ln2))
(define flonopasin (flonop-lift-helper asin))
(define flonopacos (flonop-lift-helper acos))
(define flonoptan2 (flonop-lift-helper atan2))
(define flonopround (flonop-lift-helper round))

```

```

#lang typed/racket/base
(require racket/flonum
 (except-in racket/flonum fx->fl fl->fx)
 racket/math racket/math
 "flonum.rkt"
 "flonop.rkt")
(provide deep-flonop deep-flonop? deep-flonop-argb deep-flonop-z
 deep-flonop-width deep-flonop-height deep-flonop-z-min deep-flonop-z-max
 deep-flonop-size deep-flonop-alpha deep-flonop-rgb
 flonop->deep-flonop
 : Sizing
 deep-flonop-inset deep-flonop-trin deep-flonop-scale deep-flonop-resize
 : Z-adjusting
 deep-flonop-scale-z deep-flonop-smooth-z deep-flonop-raise deep-flonop-tilt
 deep-flonop-emboss
 deep-flonop-bulge deep-flonop-bulge-round deep-flonop-bulge-round-rect
 deep-flonop-bulge-spheroid deep-flonop-bulge-horizontal deep-flonop-bulge-vertical
 deep-flonop-bulge-ripple
 ; Compositing
 deep-flonop-pin deep-flonop-pinw
 deep-flonop-lt-superimpose deep-flonop-lc-superimpose deep-flonop-lb-superimpose
 deep-flonop-ct-superimpose deep-flonop-cb-superimpose deep-flonop-cb-superimpose
 deep-flonop-rt-superimpose deep-flonop-rc-superimpose deep-flonop-rb-superimpose
 deep-flonop-vl-append deep-flonop-vc-append deep-flonop-vr-append
 deep-flonop-hl-append deep-flonop-hc-append deep-flonop-hb-append)
(struct: deep-flonop (argb : flonop) [z : flonop])
#transparent
#guard
(λ (argb-fa z-fa name)
 (match-define (flonop _ 4 w h) argb-fa)
 (match-define (flonop _ 1 w h) z-fa)
 (unless (and (≠ w zw) (≠ h zh))
 (error "deep-flonop
 *expected flonops of equal dimension; given dimensions ~w~h and ~zw~zh"))
 (values argb-fa z-fa)))
(: flonop->deep-flonop (flonop -> deep-flonop))
(define (flonop->deep-flonop flonop)

```



This could happen anywhere.

```

(when (new-x < . x-max) (set! x-min 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))]]))
(flonop-transform fa t x-min x-max y-min y-max)
[(fa 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 (flonop vs c w h) fa)
 (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-flonop
 c new-w new-h
 (λ (k x y z)
 (define-values (old-x old-y) (g (+ (fx->fl x) x-offset)
 (+ (fx->fl y) y-offset)))
 (flonop-bilinear-ref fa k old-x old-y))))))

```

```

[ (= c 1) (inline-build-flonop
 c2 w h
 (λ (k x y z) (if (unsafe-flvector-ref vs1 (coords->index 1 w 0 x y))
 (unsafe-flvector-ref vs2 i2)))))]
[ (= c 2) (inline-build-flonop
 c1 w h
 (λ (k x y z) (if (unsafe-flvector-ref vs1 i1)
 (unsafe-flvector-ref vs2 (coords->index 1 w 0 x y)))))]
[else
 (error name (string-append "expected flonops with the same number of components, "
 "or a flonop with 1 component and any same-size flonop; "
 "given flonops with ~e and ~e components")
 c2)]))]]))
(: flonop-lift2 (Symbol (Flonum Flonum -> Real) -> ((U Real flonop) (U Real flonop) -> flonop))
(define (flonop-lift2 name f)
 (flonop-lift-helper2 name (λ (x y) (real->double-flonum (f x y))))
(define fa (flonop-lift-helper2 'fa +))
(define fa- (flonop-lift-helper2 'fa -))
(define fa* (flonop-lift-helper2 'fa *))
(define fa/ (flonop-lift-helper2 'fa /))
(define flonop (flonop-lift-helper2 'flonop))
(define flonop-normalize (flonop -> flonop))
(define (flonop-normalize fa)
 (define-values (v-min v-max) (flonop-extreme-values fa))
 (define v-size (- v-max v-min))
 (let* [(f (fa- fa v-min))
 (f (if (= v-size . 0) fa (fa/ fa v-size)))]
 fa))
(define flonop/zero
 (flonop-lift-helper2 'flonop/zero (λ (x y) (if (y = . 0) 0.0 (/ x y))))))
(: flonop-divide-alpha (flonop -> flonop))
(define (flonop-divide-alpha fa)
 (match-define (flonop c w h) fa)
 (cond [(c .< . 1) fa]
 [else
 (define alpha-fa (flonop-ref-component fa 0))
 (flonop-append-components alpha-fa (flonop-drop-components fa 1) alpha-fa))]]))

```

```

(: deep-flonop-scale-z (deep-flonop (U Real flonop) -> deep-flonop))
(define (deep-flonop-scale-z dfa z)
 (match-define (deep-flonop argb-fa z-fa) dfa)
 (deep-flonop-argb-fa (fa z-fa z)))
(: deep-flonop-smooth-z (deep-flonop Real -> deep-flonop))
(define (deep-flonop-smooth-z dfa o)
 (let [(O (exact->inexact o))]
 (match-define (deep-flonop argb-fa z-fa) dfa)
 (define new-z-fa (flonop-blur z-fa O))
 (deep-flonop-argb-fa new-z-fa))
; deep-flonop-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-flonop-raise (deep-flonop (U Real flonop) -> deep-flonop))
(define (deep-flonop-raise dfa z)
 (match-define (deep-flonop argb-fa z-fa) dfa)
 (define alpha-fa (deep-flonop-alpha dfa))
 (deep-flonop-argb-fa (fa z-fa (fa alpha-fa z))))
(: deep-flonop-emboss (deep-flonop Real (U Real flonop) -> deep-flonop))
(define (deep-flonop-emboss dfa xy-ant z-ant)
 (let [(O (/ xy-ant 3.0))]
 (define z-fa (flonop-normalize (deep-flonop-alpha dfa)))
 (define new-z-fa (fa (flonop-blur z-fa O) z-ant))
 (deep-flonop-raise dfa new-z-fa)))
(: deep-flonop-bulge-helper (deep-flonop (Flonum Flonum -> Flonum) -> deep-flonop))
(define (deep-flonop-bulge-helper dfa f)
 (let O
 (define-values (w h) (deep-flonop-size dfa))
 (define half-x-size (+ 0.5 (fx->fl w) 0.5))
 (define half-y-size (+ 0.5 (fx->fl h) 0.5))
 (define z-fa
 (inline-build-flonop
 1 w h
 (λ (k x y z)
 (f (c (/ (fx->fl x) half-x-size) 1.0)
 (/ (fx->fl y) half-y-size) 1.0))))))
 (deep-flonop-raise dfa z-fa)))
(: deep-flonop-bulge (deep-flonop (Flonum Flonum -> Real) -> deep-flonop))
(define (deep-flonop-bulge dfa f)
 (deep-flonop-bulge-helper dfa
 (deep-flonop-bulge-helper dfa
 (λ (cx cy) (real->double-flonum (f cx cy))))))

```



```
#lang typed/racket/base
(require racket/flonum)
(racket/match racket/math)
"flonum.rkt"
"flonop-struct.rkt"
"flonop-stats.rkt")
(provide flonop-lift flonop-lift2 flonop-lift-helper flonop-lift-helper2
         fmgq fmgqs fmgqr fmgqs fmgqr fmgqs fmgqr fmgqs fmgqr fmgqs fmgqr fmgqs fmgqr
         fmgq fmgqs fmgqr fmgqs fmgqr fmgqs fmgqr fmgqs fmgqr fmgqs fmgqr fmgqs fmgqr fmgqs fmgqr
         fmgq fmgqs fmgqr fmgqs fmgqr fmgqs fmgqr fmgqs fmgqr fmgqs fmgqr fmgqs fmgqr fmgqs fmgqr
         flonop-normalize flonop-mult)
; =====
; Unary
(: flonop-lift-helper : (flonop -> flonop) -> flonop)
(define (flonop-lift-helper f)
  (lambda (x)
    (match-define (flonop vs c w h) f)
    (flonop (inline-build-fivector (+
                                     c w h)))))
(: flonop-lift (flonop -> Real) -> Real)
(define (flonop-lift op)
  (flonop-lift-helper (lambda (x) (real-to-dec
                                     (define fmgqs (flonop-lift-helper abs)
                                     (define fmgqr (flonop-lift-helper sqrt)
                                     (define fmgqs (flonop-lift-helper cos)
                                     (define fmgqr (flonop-lift-helper tan)
                                     (define fmgqs (flonop-lift-helper fill)
                                     (define fmgqr (flonop-lift-helper exp)
                                     (define fmgqs (flonop-lift-helper fl)
                                     (define fmgqr (flonop-lift-helper as)
                                     (define fmgqs (flonop-lift-helper ac)
                                     (define fmgqr (flonop-lift-helper atan)
                                     (define fmgqs (flonop-lift-helper round))
```



```
#lang typed/racket/base
(require racket/flonum)
(racket/match racket/math)
"flonum.rkt"
"flonop.rkt")
(provide deep-flonop? deep-flonop-argb deep-flonop-z
         deep-flonop-width deep-flonop-height deep-flonop-z-min deep-flonop-z-max
         deep-flonop-size deep-flonop-alpha deep-flonop-rgb
         flonop->deep-flonop
         : Sizing
```

This is what happen are.



```
(flonop-transfom
  ((f t x-min x-max
    (let (x-min (re
            [x-max (re
            [y-min (re
            [y-max (re
            (match-define (
            (match-define (
            (define int-x-max (ceiling x-max))
            (define int-y-min (floor y-min))
            (define int-y-max (ceiling y-max))
            (define noww (- int-x-max int-x-min))
            (define nowh (+ int-y-max int-y-min))
            (define width (floor (/ noww 2)))
            (define height (floor (/ nowh 2)))
            (define x-offest)
            (define y-offest)
            )))
```



```
(: deep-flonop-1
  (define (deep-fl
    (match-define
      (deep-flonop z
        (: deep-flonop-1
          (define (deep-fl
            (let ([o ex
                  (match-define
                    (define new-
                      (deep-flonop
                        : deep-flonop-raise and everywhere decrease from to observe an invariant
                        : when z is added, added z must be 0.0 everywhere alpha is 0.0
                        (: deep-flonop-raise (deep-flonop (U Real flonop) -> deep-flonop))
                        (define (deep-flonop-raise dfa z)
                          (match-define (deep-flonop argb-fa z-fa) dfa)
```



```
(fl x) x-offest)
(fl y) y-offest)))
))
```



```
(define alpha-fa (flonop-ref-component fa 0))
(flomop-append-components alpha-fa (fdiv/zero (flonop-drop-components fa 1) alpha-fa)))
(: deep-flonop-raise dfa z-fa))
(: deep-flonop-bulge (deep-flonop (flonop flonum -> Real) -> deep-flonop))
(define (deep-flonop-bulge dfa f)
  (deep-flonop-bulge-helper dfa (lambda (cx cy) (real->double-flonum (f cx cy)))))
```



```

#lang typed/racket/base
(require racket/hasht/racket/math racket/flonum
 (except-in racket/flonum fl->fx fx->fl)
 "flonum.rkt"
 "flonum-struct.rkt")
(provide flomap-flip-horizontal flomap-flip-vertical flomap-transpose
 flomap-cw-rotate flomap-ccw-rotate
 (struct-out invertible-2d-function) flomap-transform
 flomap-transform)
(: flomap-flip-horizontal (flomap -> flomap))
(define (flomap-flip-horizontal fa)
 (match-define (flomap vs c w h) fa)
 (define w1 (fx - w 1))
 (inline-build-flomap c w h (lambda (k x y z)
 (unsafe-flvector-ref vs (coords->index c w k (fx- w1 x) y))))))
(define (flomap-flip-vertical fa)
 (match-define (flomap vs c w h) fa)
 (define h-1 (fx- h 1))
 (inline-build-flomap c w h (lambda (k x y z)
 (unsafe-flvector-ref vs (coords->index c w k x (fx- h-1 y) z))))))
(define (flomap-transpose fa)
 (match-define (flomap vs c w h) fa)
 (inline-build-flomap c h w (lambda (k x y z)
 (unsafe-flvector-ref vs (coords->index c w k x y z))))))
(define (flomap-cw-rotate fa)
 (match-define (flomap vs c w h) fa)
 (define h-1 (fx- h 1))
 (inline-build-flomap c h w (lambda (k x y z)
 (unsafe-flvector-ref vs (coords->index c w k (fx- h-1 y) z))))))
(define (flomap-ccw-rotate fa)
 (match-define (flomap vs c w h) fa)
 (define w1 (fx- w 1))
 (inline-build-flomap c w h (lambda (k x y z)
 (unsafe-flvector-ref vs (coords->index c w k x y (fx- w1 z))))))
(struct: invertible-2d-function ((f : (Flonum Flonum -> (values Flonum Flonum)))

```

```

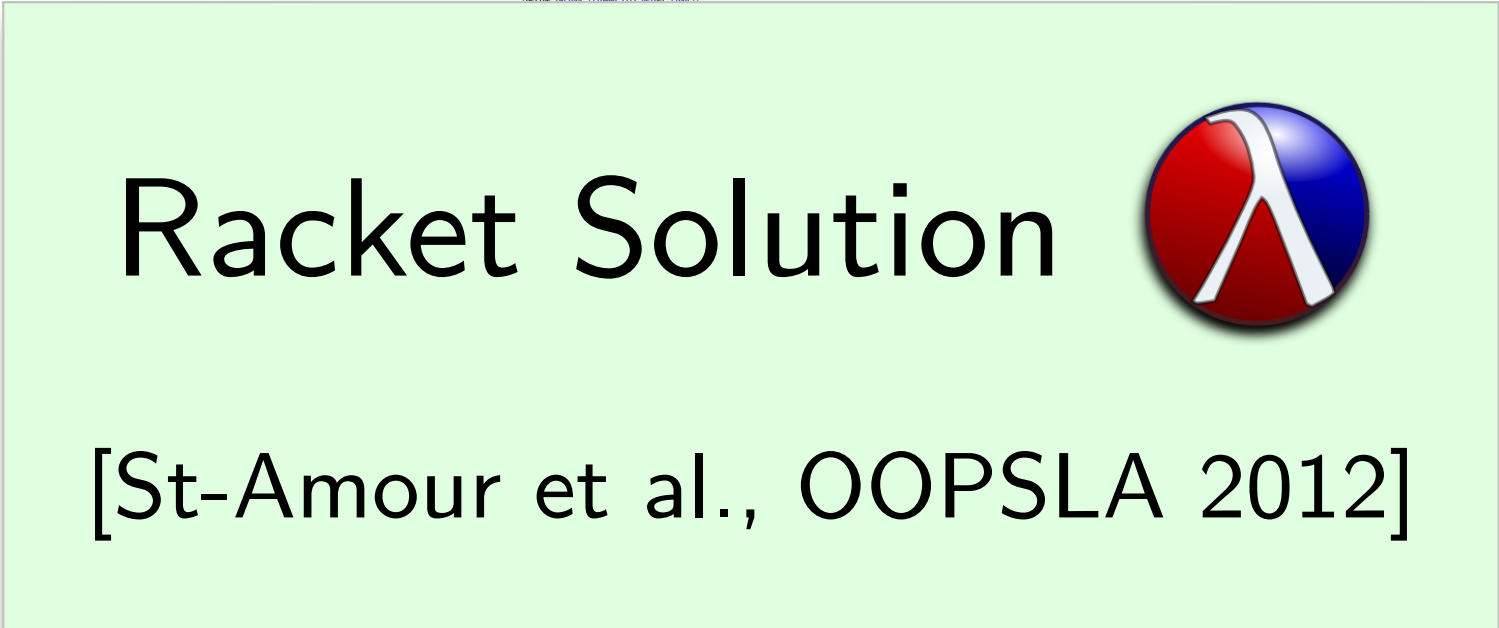
#lang typed/racket/base
(require racket/flonum
 (except-in racket/flonum fl->fx fx->fl)
 racket/math racket/math
 "flonum.rkt"
 "flonum-struct.rkt"
 "flonum-stats.rkt")
(provide flomap-lift flomap-lift2 flomap-lift-helper flomap-lift-helper2
 flomap-fmabs flomap-fmsin flomap-fmcos flomap-fmtan flomap-fmsqrt flomap-fmsin flomap-fmcos flomap-fmtan
 flomap-fmsin flomap-fmcos flomap-fmsqrt flomap-fmsin flomap-fmcos flomap-fmtan
 flomap-fmsin flomap-fmcos flomap-fmsqrt flomap-fmsin flomap-fmcos flomap-fmtan
 flomap-normalize flomap-multiply-alpha flomap-divide-alpha)
; =====
: Unary
(: flomap-lift-helper : (flomat -> flomat) -> (flomap -> flomap))
(define (flomap-lift-helper f)
 (lambda ((fm : flomap))
 (match-define (flomap vs c w h) fm)
 (flomap (inline-build-flvector (+ c w h) (lambda (i) (f (unsafe-flvector-ref vs i))))
 c w h)))
(: flomap-lift ((flomat -> flomat) -> (flomap -> flomap))
(define (flomap-lift op)
 (flomap-lift-helper (lambda (x) (real->double-flonum (op x)))))
(define flomap (flomap-lift-helper -))
(define flomapabs (flomap-lift-helper abs))
(define flomapsin (flomap-lift-helper sin))
(define flomapcos (flomap-lift-helper cos))
(define flomaptan (flomap-lift-helper tan))
(define flomaplog (flomap-lift-helper log))
(define flomapsqrt (flomap-lift-helper sqrt))
(define flomapexp (flomap-lift-helper exp))
(define flomapsqrt (flomap-lift-helper sqrt))
(define flomapasin (flomap-lift-helper asin))
(define flomapsacos (flomap-lift-helper acos))
(define flomaptan (flomap-lift-helper atan))
(define flomapsqrt (flomap-lift-helper sqrt))
(define flomapsqrt (flomap-lift-helper sqrt))
(define flomapsqrt (flomap-lift-helper sqrt))
(define flomapsqrt (flomap-lift-helper sqrt))

```

```

#lang typed/racket/base
(require racket/flonum
 (except-in racket/flonum fx->fl fl->fx)
 racket/math racket/math
 "flonum.rkt"
 "flonum.rkt")
(provide 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-rfb
 flomap->deep-flomap
 : Sizing
 deep-flomap-inset deep-flomap-trin 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-hb-append deep-flomap-hc-append deep-flomap-hb-append)
(struct: deep-flomap (argb : flomap) [z : flomap])
#transparent
#guard
(lambda (argb-fa z-fa name)
 (match-define (flomap _ 4 w h) argb-fa)
 (match-define (flomap _ 1 w h) z-fa)
 (unless (and (= w zw) (= h zh))
 (error 'deep-flomap
 "expected flomaps of equal dimension; given dimensions ~e~w and ~e~h" w h zw zh)))
 (values argb-fa z-fa)))
(: flomap->deep-flomap (flomap -> deep-flomap))
(define (deep-flomap-argb-fa)

```



```

(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
 (lambda (k x y z)
 (define-values (old-x old-y) (let* ((fx->fl x) x x-offset)
 (fx->fl y) y y-offset))
 (flomap-bilinear-ref fa k old-x old-y))))

```

```

(define flomap-max (flomap-lift-helper2 'fmax max))
(: flomap-normalize (flomap -> flomap))
(define (flomap-normalize fa)
 (define-values (v-min v-max) (flomap-extreme-values fa))
 (define v-size (- v-max v-min))
 (let* ((f (lambda (v) (if (= v 0) 0.0 (f (/ v v-size))))))
 (flomap (lambda (x) (f (flomap-lift-helper2 'fdiv zero (lambda (x y) (if (y = 0) 0.0 (/ x y))))))
 c w h)))
(: flomap-divide-alpha (flomap -> flomap))
(define (flomap-divide-alpha fa)
 (match-define (flomap c w h) fa)
 (cond [(c <= -1) fa]
 [else
 (define alpha-fa (flomap-ref-component fa 0))
 (flomap-append-components alpha-fa (fdiv zero (flomap-drop-components fa 1) alpha-fa))]))

```

```

(define z-fa (flomap-normalize (deep-flomap-alpha dfa)))
(define new-z-fa (fmap (flomap-blur z-fa 0) z-ant))
(deep-flomap-raise dfa new-z-fa)))
(: deep-flomap-bulge-helper (deep-flomap (Flonum Flonum -> Flonum) -> deep-flomap))
(define (deep-flomap-bulge-helper dfa f)
 (let ()
 (define-values (w h) (deep-flomap-size dfa))
 (define half-x-size (+ 0.5 (fx->fl w) 0.5))
 (define half-y-size (+ 0.5 (fx->fl h) 0.5))
 (define z-fa
 (inline-build-flomap
 1 w h
 (lambda (k x y z)
 (f (c (/ (fx->fl x) half-x-size) 1.0)
 (/ (fx->fl y) half-y-size) 1.0))))))
 (deep-flomap-raise dfa z-fa)))
(: deep-flomap-bulge (deep-flomap (Flonum Flonum -> Real) -> deep-flomap))
(define (deep-flomap-bulge dfa f)
 (deep-flomap-bulge-helper dfa (lambda (cx cy) (real->double-flonum (f cx cy)))))

```



```

#lang typed/racket/base
(require racket/math racket/fixnum
 (except-in racket/fixnum fl->fx fl->f)
 "flonum.rkt"
 "flonum-struct.rkt")
(provide flonop-flip-horizontal flonop-flip-vertical flonop-transpose
 flonop-cw-rotate flonop-ccw-rotate
 (struct-out invertible-2d-function) flonop-transform
 flonop-transform)
(: flonop-flip-horizontal (flonop -> flonop))
(define (flonop-flip-horizontal f)
 (match-define (flonop vs c w h) f)
 (define w1 (fx - w 1))
 (inline-build-flonop c w h (A (k x y z)
 (unsafe-flvector-ref vs (coords->index c w k (fx - w 1) y))))))
(define (flonop-flip-vertical f)
 (match-define (flonop vs c w h) f)
 (define h1 (fx - h 1))
 (inline-build-flonop c w h (A (k x y z)
 (unsafe-flvector-ref vs (coords->index c w k (fx - h 1) y))))))
(define (flonop-transpose f)
 (match-define (flonop vs c w h) f)
 (inline-build-flonop c h w (A (k x y z)
 (unsafe-flvector-ref vs (coords->index c w k (fx - h 1) y))))))
(define (flonop-cw-rotate f)
 (match-define (flonop vs c w h) f)
 (define h1 (fx - h 1))
 (inline-build-flonop c h w (A (k x y z)
 (unsafe-flvector-ref vs (coords->index c w k (fx - h 1) y))))))
(define (flonop-ccw-rotate f)
 (match-define (flonop vs c w h) f)
 (define w1 (fx - w 1))
 (inline-build-flonop c w h (A (k x y z)
 (unsafe-flvector-ref vs (coords->index c w k (fx - w 1) x))))))
(struct invertible-2d-function ([ f : (Flonum Flonum -> (values Flonum Flonum))])
 #:transparent)
(define-type Flonop Transform (Integer Integer -> invertible-2d-function))
(: transform-compose (Flonop Transform Flonop Transform -> Flonop 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 (A ((x : Flonum) [y : Flonum])
 (let-values (((x y) (f2 x y))
 ((x y) (f1 x y)))
 (A ((x : Flonum) [y : Flonum])
 (let-values (((x y) (g1 x y))
 ((x y) (g2 x y)))))))
 (transform-transform case-> (flonop Flonop Transform -> flonop)
 (flonop Flonop Transform Real Real Real -> flonop))
(define flonop-transform
 (case-lambda
 [(f)
 (match-define (flonop vs c w h) f)
 (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 (f : Integer 0)]
 (when (y f x - h)
 (let [x-loop : Void (f : Integer 0)]
 (cond [(x f x - 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))]
 (y-loop (fx + y 1))))))
 (flonop-transform fa t x-min x-max y-min y-max))
 [(f) 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 (flonop vs c w h) f)
 (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-flonop c new-w new-h
 (A (k x y z)
 (define-values (old-x old-y) (g (+ (fx->fl x) x-offset
 (+ (fx->fl y) y-offset)))
 (flonop-bilinear-ref fa k old-x old-y))))))

```

```

#lang typed/racket/base
(require racket/flonum
 (except-in racket/flonum fl->fx fl->f)
 racket/math racket/math
 "flonum.rkt"
 "flonop-struct.rkt"
 "flonop-stats.rkt")
(provide flonop-lift flonop-lift2 flonop-lift-helper flonop-lift-helper2
 flonop-fmabs flonop-fmin flonop-fmax flonop-fmabs flonop-fmmin flonop-fmmax
 flonop-normalize flonop-normalize-alpha flonop-divide-alpha)
; =====
(: unary
 [F (Flonop-lift-helper : (flonop -> flonop) -> (flonop -> flonop))
 (define (flonop-lift-helper f)
 (A ((f) : flonop)
 (match-define (flonop vs c w h) f)
 (flonop (inline-build-flvector (+ c w h) (A (l) (f (unsafe-flvector-ref vs l))))
 c w h))])
; =====
(: flonop-lift ((flonop -> Real) -> (flonop -> flonop))
(define (flonop-lift op)
 (flonop-lift-helper (A (x) (real->double-flonum (op x))))))
(define flonop (flonop-lift-helper -))
(define flonop-abs (flonop-lift-helper abs))
(define flonop-exp (flonop-lift-helper exp))
(define flonop-sin (flonop-lift-helper sin))
(define flonop-cos (flonop-lift-helper cos))
(define flonop-tan (flonop-lift-helper tan))
(define flonop-log (flonop-lift-helper log))
(define flonop-exp2 (flonop-lift-helper exp2))
(define flonop-atan (flonop-lift-helper atan))
(define flonop-round (flonop-lift-helper round))
(define flonop-floor (flonop-lift-helper floor))
(define flonop-ceil (flonop-lift-helper ceil))
(define flonop-truncate (flonop-lift-helper truncate))
(define flonop-zero (flonop-lift-helper (A (x) (if (x = . 0.0) 1.0 0.0))))
; =====
(: binary
 [F (Flonop-lift-helper : Symbol (flonop flonop -> flonop) -> ((U Real Flonop) (U Real Flonop) -> Flonop))
 (define (flonop-lift-helper2 name f)
 (let ()
 (A ((f1 : (U Real Flonop)) (f2 : (U Real Flonop)))
 (cond
 [and (real? f1) (real? f2)]
 (error name "expected at least one flonop argument, given '~e and '~e" f1 f2)]
 [(real? f1) (let [(f1 (real->double-flonum f1))]
 ((flonop-lift-helper (A (v) (f f1 v))) f2))]
 [(real? f2) (let [(f2 (real->double-flonum f2))]
 ((flonop-lift-helper (A (v) (f v f2))) f1))]
 [else
 (match-define (flonop vs1 c1 w1 h1) f1)
 (match-define (flonop vs2 c2 w2 h2) f2)
 (cond
 [(not (and (= w1 w2) (= h1 h2)))]
 (error name "expected same-size flonops; given sizes '~e and '~e" w1 h1 w2 h2)]
 [(= c1 c2) (define x (+ c1 w1 h1))
 (define res-vs (make-flvector 0))
 (flonop (inline-build-flvector n (A (l) (f (unsafe-flvector-ref vs1 l)
 (unsafe-flvector-ref vs2 l))))
 c1 w1 h1)
 [(= c1 1) (inline-build-flonop c2 w1 h1
 (A (k x y z) (f (unsafe-flvector-ref vs1 (coords->index 1 w 0 x y))
 (unsafe-flvector-ref vs2 z)))))]
 [(= c2 1) (inline-build-flonop c1 w2 h2
 (A (k x y z) (f (unsafe-flvector-ref vs1 z)
 (unsafe-flvector-ref vs2 (coords->index 1 w 0 x y)))))]
 [else
 (error name (string-append "expected flonops with the same number of components, "
 "or a flonop with 1 component and any same-size flonop; "
 "given flonops with '~e and '~e components")
 c1 c2)]))])
; =====
(: flonop-lift2 (Symbol (Flonum Flonum -> Real) -> ((U Real Flonop) (U Real Flonop) -> flonop))
(define (flonop-lift2 name f)
 (flonop-lift-helper2 name (A (x y) (real->double-flonum (f x y))))))
(define flonop (flonop-lift-helper2 'fm +))
(define flonop (flonop-lift-helper2 'fm -))
(define flonop (flonop-lift-helper2 'fm *)
 (define flonop (flonop-lift-helper2 'fm /))
 (define flonop (flonop-lift-helper2 'fmin min))
 (define flonop (flonop-lift-helper2 'fmax max))
 (: flonop-normalize (flonop -> flonop))
 (define (flonop-normalize f)
 (define-values (v-min v-max) (flonop-extreme-values f))
 (define v-size (- v-max v-min))
 (let* [(f (f - f v-min))
 (let* [(f (if (= v-size . 0) f) (f (f v-size)))
 f))
 (define flonop-zero
 (flonop-lift-helper2 'fdiv/zero (A (x y) (if (y = . 0.0) 0.0 (f x y))))
 (: flonop-divide-alpha (flonop -> flonop))
 (define (flonop-divide-alpha f)
 (match-define (flonop c w h) f)
 (cond [(c <= - 1) f]
 [else
 (define alpha-fn (flonop-ref-component f 0))
 (flonop-append-components alpha-fn (fdiv/zero (flonop-drop-components f) 1) alpha-fn))]]

```

```

#lang typed/racket/base
(require racket/flonum
 (except-in racket/fixnum fx->fl fl->f)
 racket/math racket/math
 "flonum.rkt"
 "flonop.rkt")
(provide deep-flonop deep-flonop? deep-flonop-argb deep-flonop-z
 deep-flonop-width deep-flonop-height deep-flonop-z-max
 deep-flonop-size deep-flonop-alpha deep-flonop-rgb
 flonop->deep-flonop
 : Sizing
 deep-flonop-inset deep-flonop-trim deep-flonop-scale deep-flonop-resize
 : Z-adjusting
 deep-flonop-scale-z deep-flonop-smooth-z deep-flonop-raise deep-flonop-tilt
 deep-flonop-emboss
 deep-flonop-bulge deep-flonop-bulge-round deep-flonop-bulge-round-rect
 deep-flonop-bulge-spheroid deep-flonop-bulge-horizontal deep-flonop-bulge-vertical
 deep-flonop-bulge-ripple
 ; Compositing
 deep-flonop-min deep-flonop-pin
 deep-flonop-lt-superimpose deep-flonop-lc-superimpose deep-flonop-lb-superimpose
 deep-flonop-ct-superimpose deep-flonop-cs-superimpose deep-flonop-rt-superimpose
 deep-flonop-rl-superimpose deep-flonop-rc-superimpose deep-flonop-rb-superimpose
 deep-flonop-vl-append deep-flonop-vc-append deep-flonop-vr-append
 deep-flonop-hl-append deep-flonop-hc-append deep-flonop-hb-append)
(struct deep-flonop (argb : flonop) [z : flonop])
#:transparent
#guard
(A (argb-fn z-fn name)
 (match-define (flonop _ 4 w h) argb-fn)
 (match-define (flonop _ 1 w h) z-fn)
 (unless (and (w > 0) (h > 0))
 (error "deep-flonop
 "expected flonops of equal dimension; given dimensions '~e and '~e" w h z h))
 (values argb-fn z-fn))
(: flonop->deep-flonop (flonop -> deep-flonop))
(define (flonop->deep-flonop argb-fn)
 (match-define (flonop _ 4 w h) argb-fn)
 (deep-flonop argb-fn (make-flonop 1 w h)))
(: deep-flonop-width (deep-flonop -> Nonnegative-Flonum))
(define (deep-flonop-width dfn)
 (define w (flonop-width (deep-flonop-argb dfn)))
 (with-asserts (l= nonnegative-flonum?)
 w))
(: deep-flonop-height (deep-flonop -> Nonnegative-Flonum))
(define (deep-flonop-height dfn)
 (define h (flonop-height (deep-flonop-argb dfn)))
 (with-asserts (l= nonnegative-flonum?)
 h))
(: deep-flonop-z-min (deep-flonop -> flonop))
(define (deep-flonop-z-min dfn)
 (flonop-min-value (deep-flonop-z dfn)))
(: deep-flonop-z-max (deep-flonop -> flonop))
(define (deep-flonop-z-max dfn)
 (flonop-max-value (deep-flonop-z dfn)))
(: flonop-max-value (deep-flonop-z dfn))
(: deep-flonop-size (deep-flonop -> (values Nonnegative-Flonum Nonnegative-Flonum))
(define (deep-flonop-size dfn)
 (values (deep-flonop-width dfn) (deep-flonop-height dfn)))
(: deep-flonop-alpha (deep-flonop -> flonop))
(define (deep-flonop-alpha dfn)
 (flonop-ref-component (deep-flonop-argb dfn) 0))
(: deep-flonop-argb (deep-flonop -> flonop))
(define (deep-flonop-argb dfn)
 (flonop-drop-components (deep-flonop-argb dfn) 1))
; =====
; Z adjusters
(: deep-flonop-scale-z (deep-flonop (U Real Flonop) -> deep-flonop))
(define (deep-flonop-scale-z dfn z)
 (match-define (deep-flonop argb-fn z-fn) dfn)
 (deep-flonop argb-fn (fx z-fn z)))
(: deep-flonop-smooth-z (deep-flonop Real -> deep-flonop))
(define (deep-flonop-smooth-z dfn o)
 (let [(O (exact->inexact O))]
 (match-define (deep-flonop argb-fn z-fn) dfn)
 (define new-z-fn (flonop-blur z-fn O))
 (deep-flonop argb-fn new-z-fn))
; Deep-flonop-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-flonop-raise (deep-flonop (U Real Flonop) -> deep-flonop))
(define (deep-flonop-raise dfn z)
 (match-define (deep-flonop argb-fn z-fn) dfn)
 (define alpha-fn (deep-flonop-alpha dfn))
 (deep-flonop argb-fn (fx z-fn (fx alpha-fn z))))
(: deep-flonop-emboss (deep-flonop Real (U Real Flonop) -> deep-flonop))
(define (deep-flonop-emboss dfn xy-ant z-ant)
 (let [(O (/ xy-ant 3.0))]
 (define z-fn (flonop-normalize (deep-flonop-alpha dfn))
 (define new-z-fn (f (flonop-blur z-fn O) z-ant))
 (deep-flonop-raise dfn new-z-fn))
; =====
(: deep-flonop-bulge-helper (deep-flonop (Flonum Flonum -> flonop) -> deep-flonop))
(define (deep-flonop-bulge-helper dfn f)
 (let ()
 (define-values (w h) (deep-flonop-size dfn))
 (define half-x-size (+ 0.5 (fx->fl w) 0.5))
 (define half-y-size (+ 0.5 (fx->fl h) 0.5))
 (define z-fn
 (inline-build-flonop
 1 w h
 (A (k x y z)
 (f (c (/ (fx->fl x) half-x-size) 1.0)
 (/ (fx->fl y) half-y-size) 1.0))))
 (deep-flonop-raise dfn z-fn))
(: deep-flonop-bulge (deep-flonop (Flonum Flonum -> Real) -> deep-flonop))
(define (deep-flonop-bulge dfn f)
 (deep-flonop-bulge-helper dfn (A (cx cy) (real->double-flonum (f cx cy))))

```

```
#lang typed/racket/base
(require racket/math racket/flonum
 (except-in racket/flonum fl->fx fx->fl)
 "flonum.rkt"
 "flonap-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 fa)
 (match-define (flomap vs c w h) fa)
 (define w1 (fx- w 1))
 (inline-build-flomap c w h (λ (k x y)
 (unsafe-flvector-ref vs (coords->index c w k (fx- w1 x) y))))))
(define flomap-flip-vertical fa)
 (match-define (flomap vs c w h) fa)
 (define h-1 (fx- h 1))
 (inline-build-flomap c w h (λ (k x y)
 (unsafe-flvector-ref vs (coords->index c w k x (fx- h-1 y))))))
(define flomap-transpose fa)
 (match-define (flomap vs c w h) fa)
 (inline-build-flomap c h w (λ (k x y)
 (unsafe-flvector-ref vs (coords->index c w k x y))))))
```

```
#lang typed/racket/base
(require racket/flonum
 (except-in racket/flonum fl->fx fx->fl)
 racket/math racket/math
 "flonum.rkt"
 "flonap-struct.rkt"
 "flonap-stats.rkt")
(provide flomap-lift flomap-lift2 flomap-lift-helper flomap-lift-helper2
 flmag fmagb fmagc fmagd fmagf fmagg fmagh fmagi fmagj fmagk fmagl fmagm
 fmag n fmag o fmag p fmag q fmag r fmag s fmag t fmag u fmag v fmag w fmag x
 fmag y fmag z fmag aa fmag ab fmag ac fmag ad fmag ae fmag af fmag ag fmag ah
 fmag ai fmag aj fmag ak fmag al fmag am fmag an fmag ao fmag ap fmag aq fmag ar
 fmag as fmag at fmag au fmag av fmag aw fmag ax fmag ay fmag az fmag ba fmag bb
 fmag bc fmag bd fmag be fmag bf fmag bg fmag bh fmag bi fmag bj fmag bk fmag bl
 fmag bm fmag bn fmag bo fmag bp fmag bq fmag br fmag bs fmag bt fmag bu fmag bv
 fmag bw fmag bx fmag by fmag bz fmag ca fmag cb fmag cc fmag cd fmag ce fmag cf
 fmag cg fmag ch fmag ci fmag cj fmag ck fmag cl fmag cm fmag cn fmag co fmag cp
 fmag cq fmag cr fmag cs fmag ct fmag cu fmag cv fmag cw fmag cx fmag cy fmag cz
 fmag da fmag db fmag dc fmag dd fmag de fmag df fmag dg fmag dh fmag di fmag dj
 fmag dk fmag dl fmag dm fmag dn fmag do fmag dp fmag dq fmag dr fmag ds fmag dt
 fmag du fmag dv fmag dw fmag dx fmag dy fmag dz fmag ea fmag eb fmag ec fmag ed
 fmag ee fmag ef fmag eg fmag eh fmag ei fmag ej fmag ek fmag el fmag em fmag en
 fmag eo fmag ep fmag eq fmag er fmag es fmag et fmag eu fmag ev fmag ew fmag ex
 fmag ey fmag ez fmag fa fmag fb fmag fc fmag fd fmag fe fmag ff fmag fg fmag fh
 fmag fi fmag fj fmag fk fmag fl fmag fm fmag fn fmag fo fmag fp fmag fq fmag fr
 fmag fs fmag ft fmag fu fmag fv fmag fw fmag fx fmag fy fmag fz fmag ga fmag gb
 fmag gc fmag gd fmag ge fmag gf fmag gg fmag gh fmag gi fmag gj fmag gk fmag gl
 fmag gm fmag gn fmag go fmag gp fmag gq fmag gr fmag gs fmag gt fmag gu fmag gv
 fmag gw fmag gx fmag gy fmag gz fmag ha fmag hb fmag hc fmag hd fmag he fmag hf
 fmag hg fmag hh fmag hi fmag hj fmag hk fmag hl fmag hm fmag hn fmag ho fmag hp
 fmag hq fmag hr fmag hs fmag ht fmag hu fmag hv fmag hw fmag hx fmag hy fmag hz
 fmag ia fmag ib fmag ic fmag id fmag ie fmag if fmag ig fmag ih fmag ii fmag ij
 fmag ik fmag il fmag im fmag in fmag io fmag ip fmag iq fmag ir fmag is fmag it
 fmag iu fmag iv fmag iw fmag ix fmag iy fmag iz fmag ja fmag jb fmag jc fmag jd
 fmag je fmag jf fmag jg fmag jh fmag ji fmag jj fmag jk fmag jl fmag jm fmag jn
 fmag jo fmag jp fmag jq fmag jr fmag js fmag jt fmag ju fmag jv fmag jw fmag jx
 fmag jy fmag jz fmag ka fmag kb fmag kc fmag kd fmag ke fmag kf fmag kg fmag kh
 fmag ki fmag kj fmag kk fmag kl fmag km fmag kn fmag ko fmag kp fmag kq fmag kr
 fmag ks fmag kt fmag ku fmag kv fmag kw fmag kx fmag ky fmag kz fmag la fmag lb
 fmag lc fmag ld fmag le fmag lf fmag lg fmag lh fmag li fmag lj fmag lk fmag ll
 fmag lm fmag ln fmag lo fmag lp fmag lq fmag lr fmag ls fmag lt fmag lu fmag lv
 fmag lw fmag lx fmag ly fmag lz fmag ma fmag mb fmag mc fmag md fmag me fmag mf
 fmag mg fmag mh fmag mi fmag mj fmag mk fmag ml fmag mm fmag mn fmag mo fmag mp
 fmag mq fmag mr fmag ms fmag mt fmag mu fmag mv fmag mw fmag mx fmag my fmag mz
 fmag na fmag nb fmag nc fmag nd fmag ne fmag nf fmag ng fmag nh fmag ni fmag nj
 fmag nk fmag nl fmag nm fmag nn fmag no fmag np fmag nq fmag nr fmag ns fmag nt
 fmag nu fmag nv fmag nw fmag nx fmag ny fmag nz fmag oa fmag ob fmag oc fmag od
 fmag oe fmag of fmag og fmag oh fmag oi fmag oj fmag ok fmag ol fmag om fmag on
 fmag oo fmag op fmag oq fmag or fmag os fmag ot fmag ou fmag ov fmag ow fmag ox
 fmag oy fmag oz fmag pa fmag pb fmag pc fmag pd fmag pe fmag pf fmag pg fmag ph
 fmag pi fmag pj fmag pk fmag pl fmag pm fmag pn fmag po fmag pp fmag pq fmag pr
 fmag ps fmag pt fmag pu fmag pv fmag pw fmag px fmag py fmag pz fmag qa fmag qb
 fmag qc fmag qd fmag qe fmag qf fmag qg fmag qh fmag qi fmag qj fmag qk fmag ql
 fmag qm fmag qn fmag qo fmag qp fmag qq fmag qr fmag qs fmag qt fmag qu fmag qv
 fmag qw fmag qx fmag qy fmag qz fmag ra fmag rb fmag rc fmag rd fmag re fmag rf
 fmag rg fmag rh fmag ri fmag rj fmag rk fmag rl fmag rm fmag rn fmag ro fmag rp
 fmag rq fmag rr fmag rs fmag rt fmag ru fmag rv fmag rw fmag rx fmag ry fmag rz
 fmag sa fmag sb fmag sc fmag sd fmag se fmag sf fmag sg fmag sh fmag si fmag sj
 fmag sk fmag sl fmag sm fmag sn fmag so fmag sp fmag sq fmag sr fmag ss fmag st
 fmag su fmag sv fmag sw fmag sx fmag sy fmag sz fmag ta fmag tb fmag tc fmag td
 fmag te fmag tf fmag tg fmag th fmag ti fmag tj fmag tk fmag tl fmag tm fmag tn
 fmag to fmag tp fmag tq fmag tr fmag ts fmag tt fmag tu fmag tv fmag tw fmag tx
 fmag ty fmag tz fmag ua fmag ub fmag uc fmag ud fmag ue fmag uf fmag ug fmag uh
 fmag ui fmag uj fmag uk fmag ul fmag um fmag un fmag uo fmag up fmag uq fmag ur
 fmag us fmag ut fmag uu fmag uv fmag uw fmag ux fmag uy fmag uz fmag va fmag vb
 fmag vc fmag vd fmag ve fmag vf fmag vg fmag vh fmag vi fmag vj fmag vk fmag vl
 fmag vm fmag vn fmag vo fmag vp fmag vq fmag vr fmag vs fmag vt fmag vu fmag vv
 fmag vw fmag vx fmag vy fmag vz fmag wa fmag wb fmag wc fmag wd fmag we fmag wf
 fmag wg fmag wh fmag wi fmag wj fmag wk fmag wl fmag wm fmag wn fmag wo fmag wp
 fmag wq fmag wr fmag ws fmag wt fmag wu fmag wv fmag ww fmag wx fmag wy fmag wz
 fmag xa fmag xb fmag xc fmag xd fmag xe fmag xf fmag xg fmag xh fmag xi fmag xj
 fmag xk fmag xl fmag xm fmag xn fmag xo fmag xp fmag xq fmag xr fmag xs fmag xt
 fmag xu fmag xv fmag xw fmag xx fmag xy fmag xz fmag ya fmag yb fmag yc fmag yd
 fmag ye fmag yf fmag yg fmag yh fmag yi fmag yj fmag yk fmag yl fmag ym fmag yn
 fmag yo fmag yp fmag yq fmag yr fmag ys fmag yt fmag yu fmag yv fmag yw fmag yx
 fmag yy fmag yz fmag za fmag zb fmag zc fmag zd fmag ze fmag zf fmag zg fmag zh
 fmag zi fmag zj fmag zk fmag zl fmag zm fmag zn fmag zo fmag zp fmag zq fmag zr
 fmag zs fmag zt fmag zu fmag zv fmag zw fmag zx fmag zy fmag zz)
 flomap-normalize flomap-multiply-alpha flomap-divide-alpha)
; =====
; Unary
[: flomap-lift-helper (flomat -> flomat -> (flomap -> flomap))
 (define (flomap-lift-helper f)
 (λ (A (f# : flomap))
 (match-define (flomap vs c w h) fa)
 (flomap (inline-build-flvector (+ c w h) (A (λ (f (unsafe-flvector-ref vs i))))
 c w h))))))
(: flomap-lift ((flomat -> Real) -> (flomap -> flomap))
 (define (flomap-lift op)
 (flomap-lift-helper (λ (x) (real->double-flonum (op x)))))
 (define fmag (flomap-lift-helper -))
 (define fmagb (flomap-lift-helper abs))
```

```
#lang typed/racket/base
(require racket/flonum
 (except-in racket/flonum fx->fl fl->fx)
 racket/math racket/math
 "flonum.rkt"
 "flonap.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-pinw
 deep-flomap-lt-superimpose deep-flomap-lc-superimpose deep-flomap-lb-superimpose
 deep-flomap-ct-superimpose deep-flomap-cc-superimpose deep-flomap-ch-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-hl-append deep-flomap-hc-append deep-flomap-hb-append)
(structural deep-flomap (flarg : flomat) (r : flomat))
```

```
(let (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) fa)
 (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 min-w (- int-x-max int-x-min))
 (define max-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 min-w max-h
 (λ (k x y)
 (define-values (old-x old-y) (g (+ (fx->fl x) x-offset
 (+ (fx->fl y) y-offset))))
 (flomap-bitlinear-ref fa k old-x old-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 'x' and 'y' components")
 ct 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 f# (flomap-lift-helper2 'f# +))
(define f#- (flomap-lift-helper2 'f# -))
(define f#* (flomap-lift-helper2 'f# *))
(define f#/ (flomap-lift-helper2 'f# /))
(define f#min (flomap-lift-helper2 'f#min min))
(define f#max (flomap-lift-helper2 'f#max max))
(: flomap-normalize (flomap -> flomap))
(define (flomap-normalize fa)
 (define-values (v-min v-max) (flomap-extreme-values fa))
 (define v-size (- v-max v-min))
 (let* ((f# (f#- fa v-min))
 (f# (if (= v-size . 0) f# (f# f# v-size))))
 f#))
(define f#div/zero
 (flomap-lift-helper2 'f#div/zero (λ (x y) (if (y = . 0) 0.0 (/ x y))))))
(: flomap-divide-alpha (flomap -> flomap))
(define (flomap-divide-alpha fa)
 (match-define (flomap c w h) fa)
 (cond ((c <= - 1) fa)
 [else
 (define alpha-f# (flomap-ref-component fa 0))
 (flomap-append-components alpha-f# (f#div/zero (flomap-drop-components fa 1) alpha-f#)))]))
```

```
(match-define (deep-flomap argb-fa z-fa) dfa)
(define new-z-fa (flomap-blur z-fa 0))
(define (deep-flomap-argb-fa new-z-fa))
; 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 dfa z)
 (match-define (deep-flomap argb-fa z-fa) dfa)
 (define alpha-f# (deep-flomap-alpha dfa))
 (deep-flomap-argb-fa (fa z-fa (fa alpha-f# z))))
(: deep-flomap-emboss (deep-flomap Real (U Real flomap) -> deep-flomap))
(define (deep-flomap-emboss dfa xy-ant z-ant)
 (let ((z (/ xy-ant 3.0)))
 (define z-f# (flomap-normalize (deep-flomap-alpha dfa)))
 (define new-z-f# (fa (flomap-blur z-f# 0) z-ant))
 (deep-flomap-raise dfa new-z-f#)))
(: deep-flomap-bulge-helper (deep-flomap (flonum flonum -> flomat) -> deep-flomap))
(define (deep-flomap-bulge-helper dfa f)
 (let ()
 (define-values (w h) (deep-flomap-size dfa))
 (define half-x-size (+ 0.5 (fx->fl w) 0.5))
 (define half-y-size (+ 0.5 (fx->fl h) 0.5))
 (define z-f#
 (inline-build-flomap
 1 w h
 (λ (x y)
 (f (c (/ (fx->fl x) half-x-size) 1.0)
 (/ (fx->fl y) half-y-size) 1.0))))))
 (deep-flomap-raise dfa z-f#)))
(: deep-flomap-bulge (deep-flomap (flonum flonum -> Real) -> deep-flomap))
(define (deep-flomap-bulge dfa f)
 (deep-flomap-bulge-helper dfa (λ (x y) (real->double-flonum (f x y)))))
```

```

#lang typed/racket/base
(require racket/math racket/flonum
 (except-in racket/flonum fl->fx fx->fl)
 "flonum.rkt"
 "flonop-struct.rkt")
"flonop.rkt"
"flonop-struct.rkt"
"flonop-stats.rkt")
(provide flonop-flip-horizontal flonop-flip-vertical flonop-transpose
 flonop-cw-rotate flonop-ccw-rotate
 (struct-out invertible-2d-function) flonop-transform
 transform-compose rotate-transform whirl-and-pinch-transform
 flonop-transform)
(: flonop-flip-horizontal (flonop -> flonop))
(define (flonop-flip-horizontal fa)
 (match-define (flonop vs c w h) fa)
 (define w1 (fx- w 1))
 (inline-build-flonop c w h (λ (k x y z)
 (unsafe-flvector-ref vs (coords->index c w k (fx- w1 x) y))))))
(define (flonop-flip-vertical fa)
 (match-define (flonop vs c w h) fa)
 (define h-1 (fx- h 1))
 (inline-build-flonop c w h (λ (k x y z)
 (unsafe-flvector-ref vs (coords->index c w k x (fx- h-1 y) z))))))
(define (flonop-transpose fa)
 (match-define (flonop vs c w h) fa)
 (inline-build-flonop c h w (λ (k x y z)
 (unsafe-flvector-ref vs (coords->index c w k y x z))))))
(define (flonop-cw-rotate fa)

```

```

#lang typed/racket/base
(require racket/flonum
 (except-in racket/flonum fl->fx fx->fl)
 racket/math racket/math
 "flonum.rkt"
 "flonop-struct.rkt"
 "flonop-stats.rkt")
(provide flonop-lift flonop-lift2 flonop-lift-helper flonop-lift-helper2
 flonop-fmabs flonop-fmmin flonop-fmcos flonop-fmsin flonop-fmtan
 flonop-fmround flonop-fmceil flonop-fmtruncate flonop-fmzero
 flonop-fm- fa- fa- fa/ flonop-fmmax
 flonop-normalize flonop-multiply-alpha flonop-divide-alpha)
; =====
: Unary
(: flonop-lift-helper : (flonop -> flonop) -> (flonop -> flonop))
(define (flonop-lift-helper f)
 (λ (fa : (flonop -> flonop))
 (match-define (flonop vs c w h) fa)
 (flonop (inline-build-flvector (* c w h) (λ (i) (f (unsafe-flvector-ref vs i))))
 c w h)))
(: flonop-lift2 (flonop -> flonop) -> (flonop -> flonop))
(define (flonop-lift2 fa)
 (flonop-lift-helper (λ (x) (real->double-flonum (op x)))))
(: flonop-lift-helper2 (flonop -> flonop) -> (flonop -> flonop))
(define flonop-lift-helper2 fa)
 (define flonop-lift-helper2 (λ (x) (real->double-flonum (op x))))
(define flonop-lift-helper2 (λ (x) (real->double-flonum (op x))))
(define flonop-lift-helper2 (λ (x) (real->double-flonum (op x))))
(define flonop-lift-helper2 (λ (x) (real->double-flonum (op x))))

```

```

#lang typed/racket/base
(require racket/flonum
 (except-in racket/flonum fl->fx fx->fl)
 racket/math racket/math
 "flonum.rkt"
 "flonop.rkt")
(provide deep-flonop? deep-flonop-argb deep-flonop-z
 deep-flonop-width deep-flonop-height deep-flonop-z-min deep-flonop-z-max
 deep-flonop-size deep-flonop-alpha deep-flonop-rgb
 flonop->deep-flonop
 : Sizing
 deep-flonop-inset deep-flonop-trin deep-flonop-scale deep-flonop-resize
 : Z-adjusting
 deep-flonop-scale-z deep-flonop-smooth-z deep-flonop-raise deep-flonop-tilt
 deep-flonop-emboss
 deep-flonop-bulge deep-flonop-bulge-round deep-flonop-bulge-round-rect
 deep-flonop-bulge-spheroid deep-flonop-bulge-bulge-vertical
 deep-flonop-bulge-ripple
 ; Compositing
 deep-flonop-pin deep-flonop-pinw
 deep-flonop-lt-superimpose deep-flonop-lc-superimpose deep-flonop-lb-superimpose
 deep-flonop-rt-superimpose deep-flonop-rc-superimpose deep-flonop-rb-superimpose
 deep-flonop-rl-superimpose deep-flonop-rr-superimpose deep-flonop-rlb-superimpose
 deep-flonop-vl-append deep-flonop-vc-append deep-flonop-vr-append
 deep-flonop-hl-append deep-flonop-hc-append deep-flonop-hb-append)
(struct: deep-flonop (argb : flonop) [z : flonop])
#transparent

```

JavaScript Solution

JS

This talk

```

; or a flonop with 1 component and any same-size flonop; "
; given flonops with "e" and "o" components"
(: flonop-lift2 (Symbol (flonop flonum -> Real) -> ((U Real flonop) (U Real flonop) -> flonop))
 (define (flonop-lift2 name f)
 (flonop-lift-helper2 name (λ (x y) (real->double-flonum (f x y))))
 (define fa (flonop-lift-helper2 'fa +))
 (define fb (flonop-lift-helper2 'fa -))
 (define fc (flonop-lift-helper2 'fa *)
 (define fd (flonop-lift-helper2 'fa /))
 (define fmin (flonop-lift-helper2 'fmin min))
 (define fmax (flonop-lift-helper2 'fmax max))
 (: flonop-normalize (flonop -> flonop))
 (define (flonop-normalize fa)
 (define-values (v-min v-max) (flonop-extreme-values fa))
 (define v-size (- v-max v-min))
 (let* ((fa (fa- fa v-min))
 (fb (if (= v-size 0) 0.0) fa (fa/ fa v-size)))
 fa))
(define fndiv/zero
 (flonop-lift-helper2 'fndiv/zero (λ (x y) (if (y . = . 0.0) 0.0 (/ x y))))
 (: flonop-divide-alpha (flonop -> flonop))
 (define (flonop-divide-alpha fa)
 (match-define (flonop c w h) fa)
 (cond [(c . <= . 1) fa]
 [else
 (define alpha-fa (flonop-ref-component fa 0))
 (flonop-append-components alpha-fa (fndiv/zero (flonop-drop-components fa 1) alpha-fa))]))

```

```

(deep-flonop-argb-fa msw-z-fa)))
; deep-flonop-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-flonop-raise (deep-flonop (U Real flonop) -> deep-flonop))
(define (deep-flonop-raise dfa z)
 (match-define (deep-flonop argb-fa z-fa) dfa)
 (define alpha-fa (deep-flonop-alpha dfa))
 (deep-flonop-argb-fa (fa z-fa (fa alpha-fa z))))
(: deep-flonop-emboss (deep-flonop Real (U Real flonop) -> deep-flonop))
(define (deep-flonop-emboss dfa xy-ant z-ant)
 (let ((/ (xy-ant 3.0)))
 (define z-fa (flonop-normalize (deep-flonop-alpha dfa)))
 (define msw-z-fa (fa (flonop-blur z-fa 0) z-ant))
 (deep-flonop-raise dfa msw-z-fa)))
(: deep-flonop-bulge-helper (deep-flonop (flonop flonum -> flonum) -> deep-flonop))
(define (deep-flonop-bulge-helper dfa f)
 (let ()
 (define-values (w h) (deep-flonop-size dfa))
 (define half-x-size (+ 0.5 (fx->fl 0) 0.5))
 (define half-y-size (+ 0.5 (fx->fl h) 0.5))
 (define z-fa
 (inline-build-flonop
 1 w h
 (λ (k x y z)
 (f (c (/ (fx->fl x) half-x-size) 1.0)
 (/ (fx->fl y) half-y-size) 1.0))))))
 (deep-flonop-raise dfa z-fa)))
(: deep-flonop-bulge (deep-flonop (flonop flonum -> Real) -> deep-flonop))
(define (deep-flonop-bulge dfa f)
 (deep-flonop-bulge-helper dfa (λ (cx cy) (real->double-flonum (f cx cy)))))

```


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

```
badness: 2023  
raytrace.js:432:12  
property: isHit
```

Can't inline assignment.

This operation may add a new property to objects.

Initialize the property in the constructor to enable optimizations.

...

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

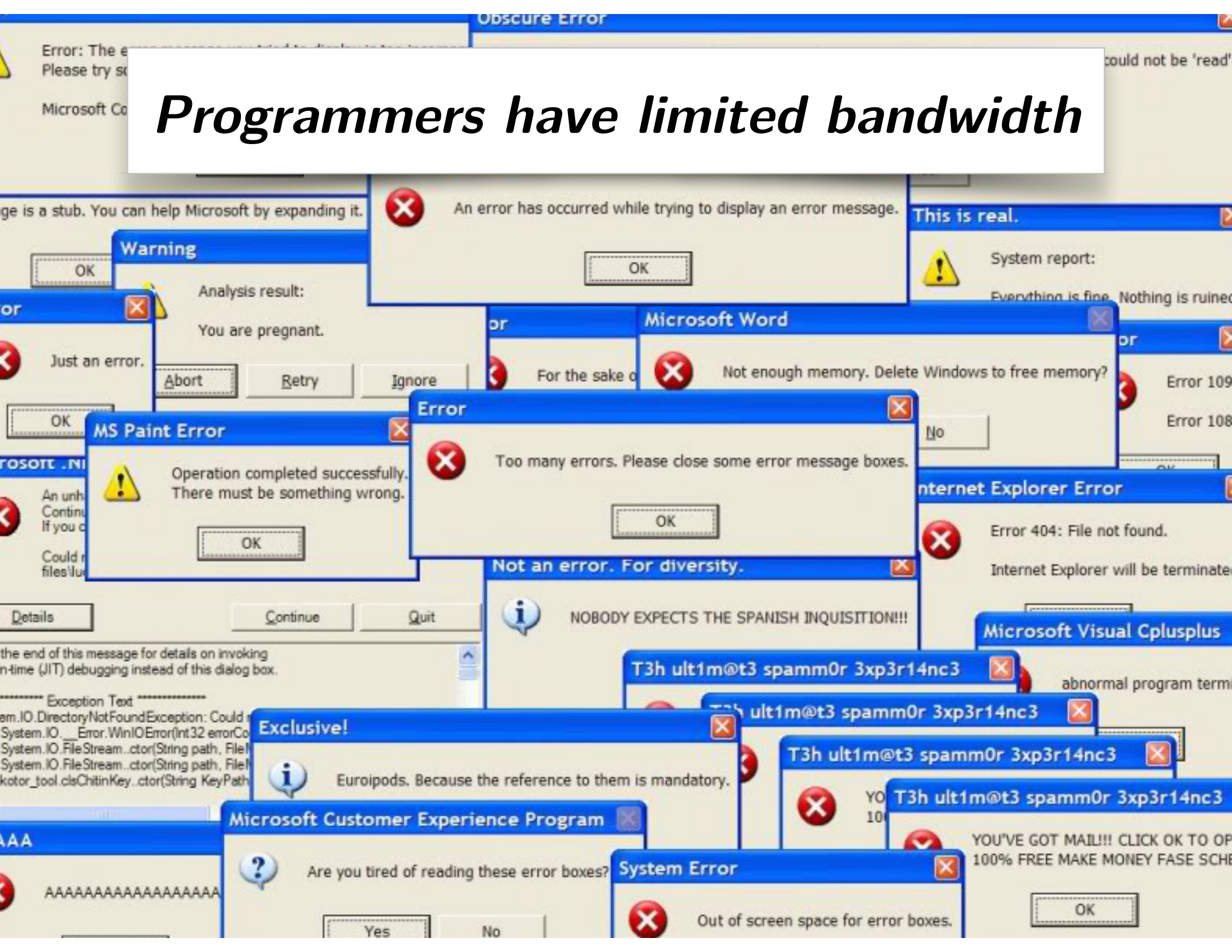
```
saturn_V = {height: 110, stages: 3}
ariane_5 = {stages: 2, height: 52}
→ ariane_5 = {height: 52, stages: 2}
```

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



Programmers have limited bandwidth



Why JS for coaching?

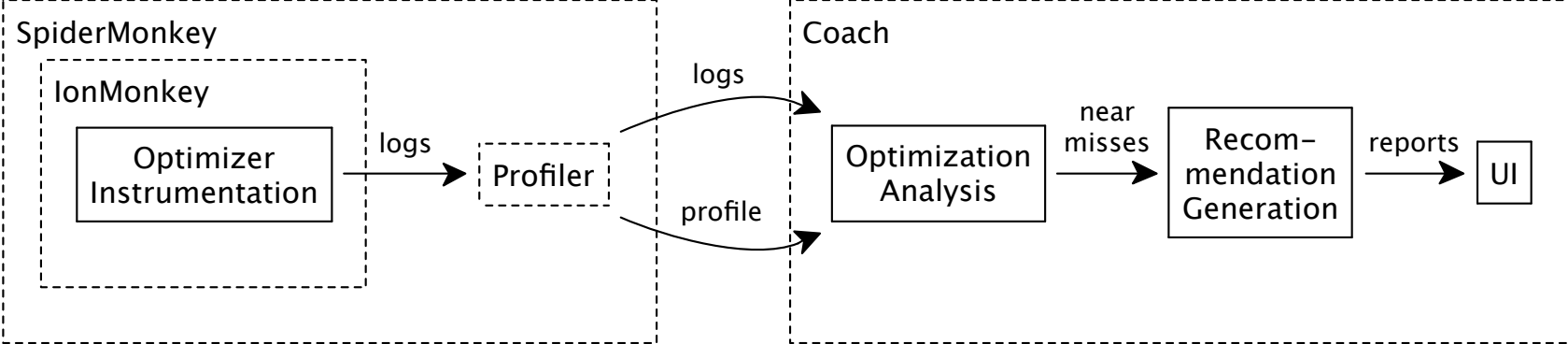
- Can it work beyond Racket?
- Different compilation model (JIT)
- Different language (OO)

Why coaching for JS?

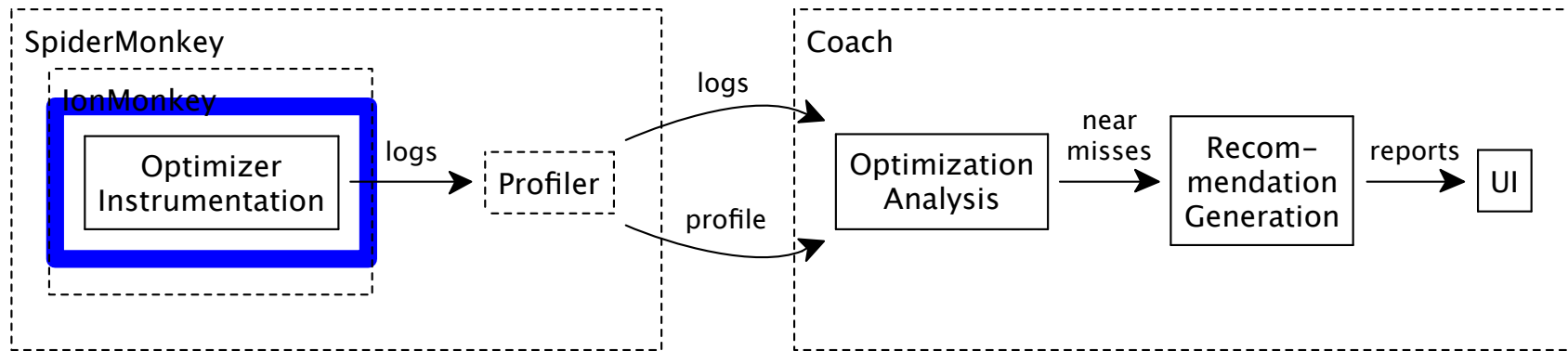
- Hard to write performant code
- Performance matters
- Non-experts / multi-language programmers

How does it work?

Architecture

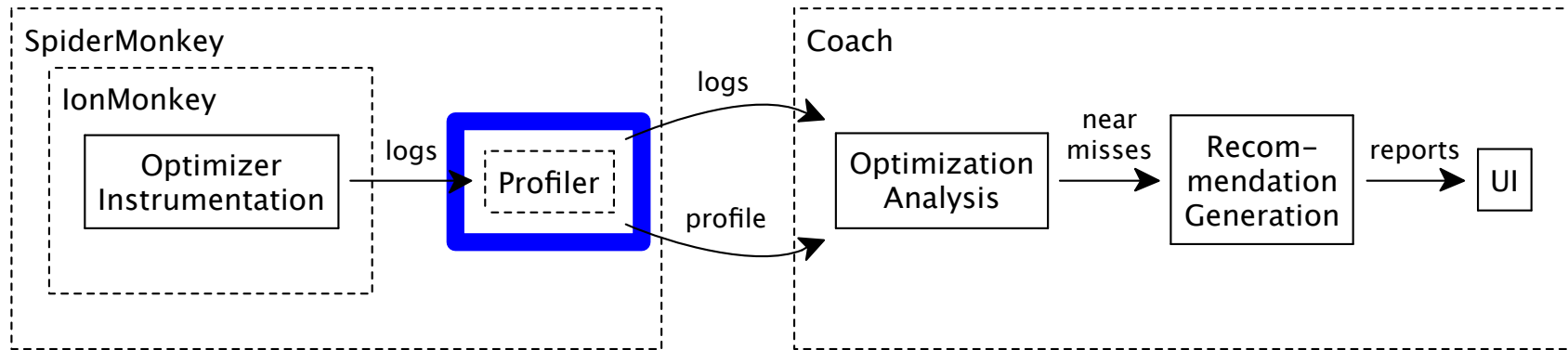


Architecture



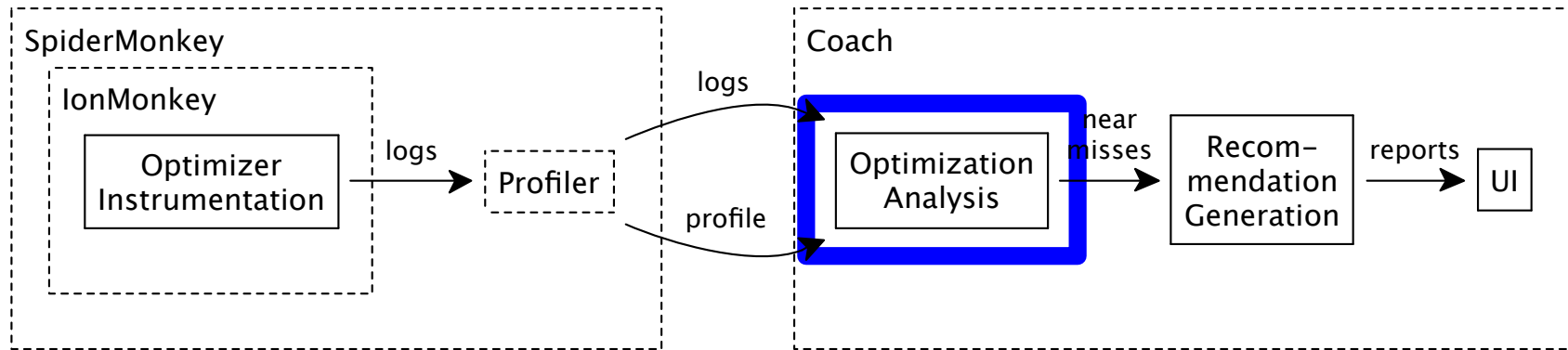
Log optimization decisions
(attempts, successes, failures)

Architecture



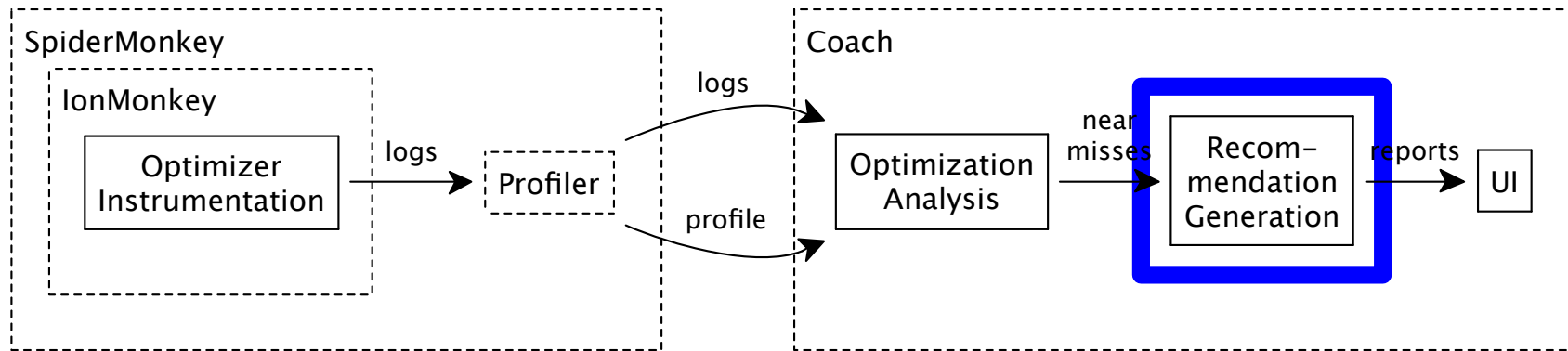
Emit profile events
(get logs out of the engine)

Architecture



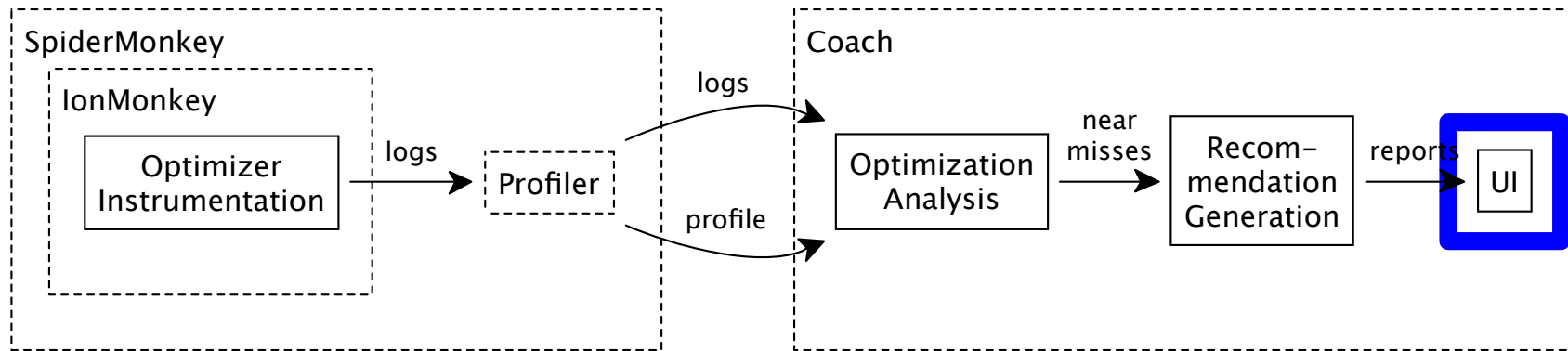
Produce near miss reports
(pruning, merging, ranking)

Architecture



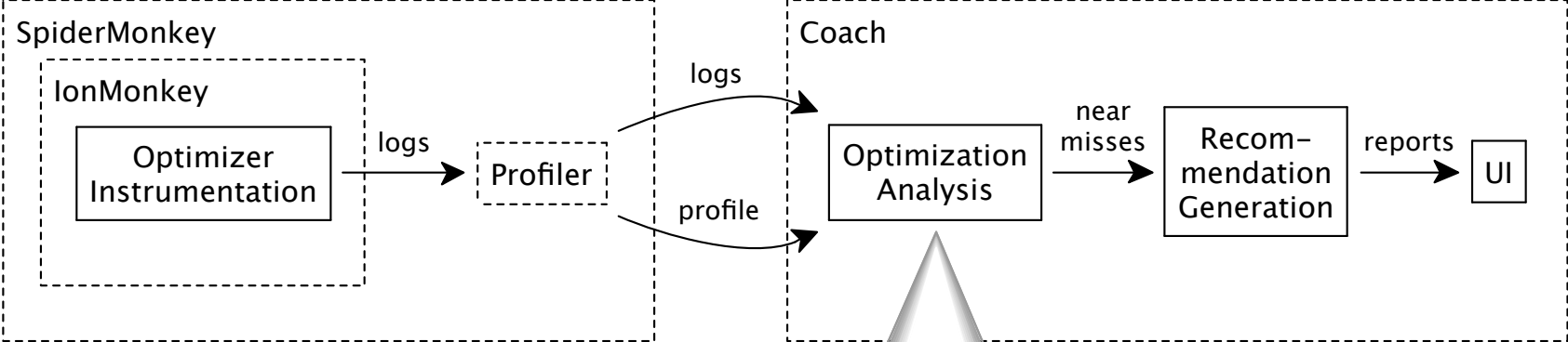
Fill recommendation templates
(general advice + targeted info)

Architecture



Show reports and recommendations
(consumed by programmers)

Architecture

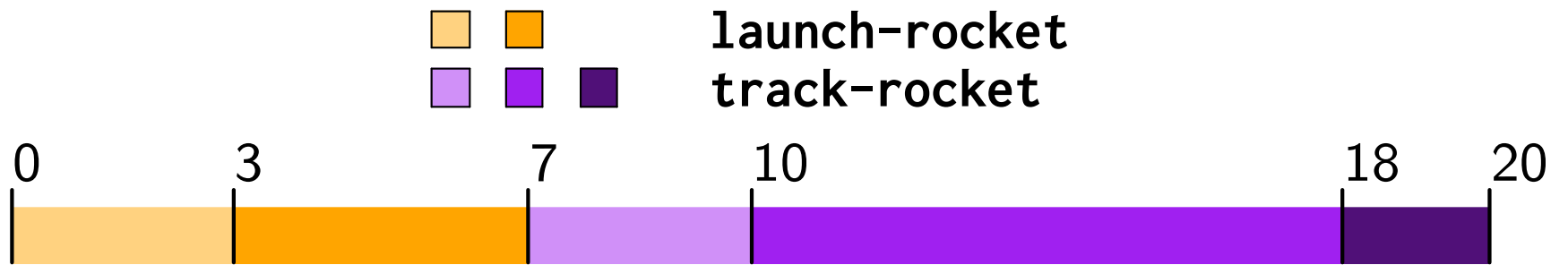


...

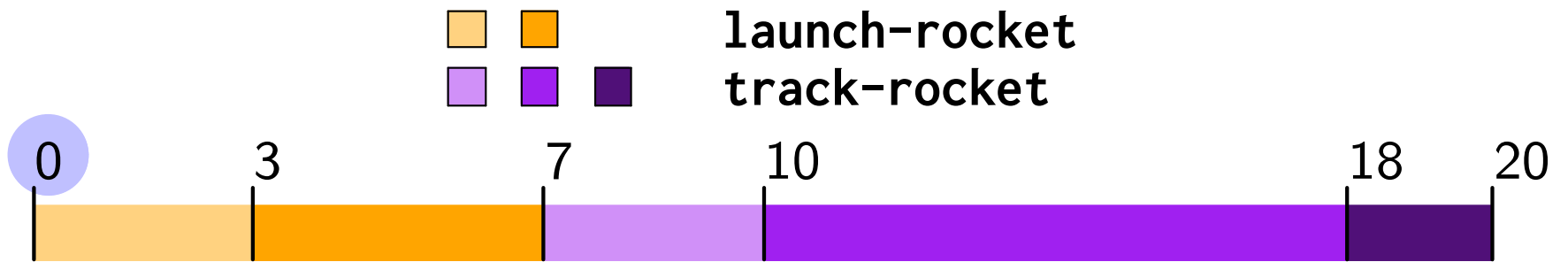
Profiling-Based Badness } JIT
Temporal Merging }

...

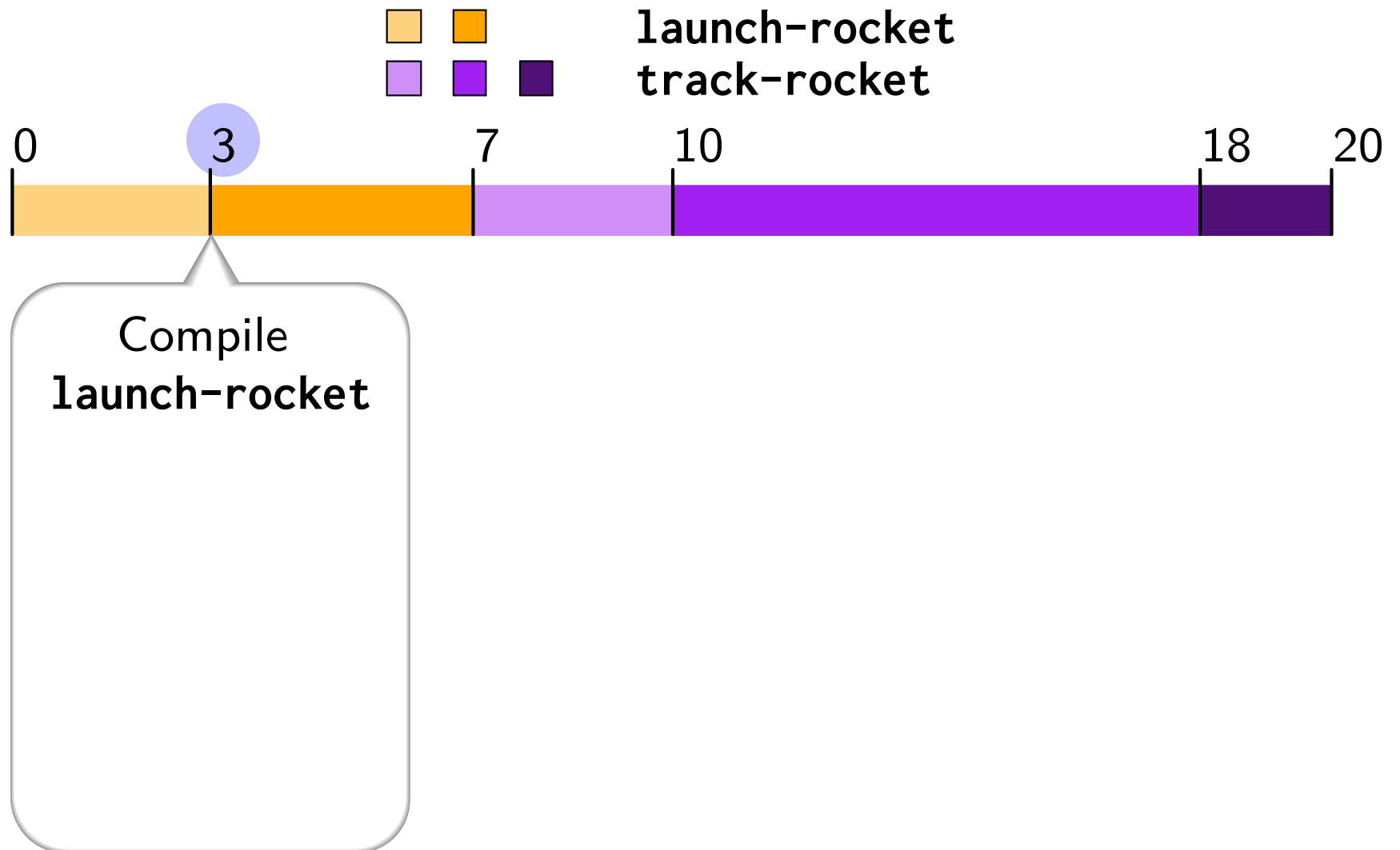
Profiling-Based Badness



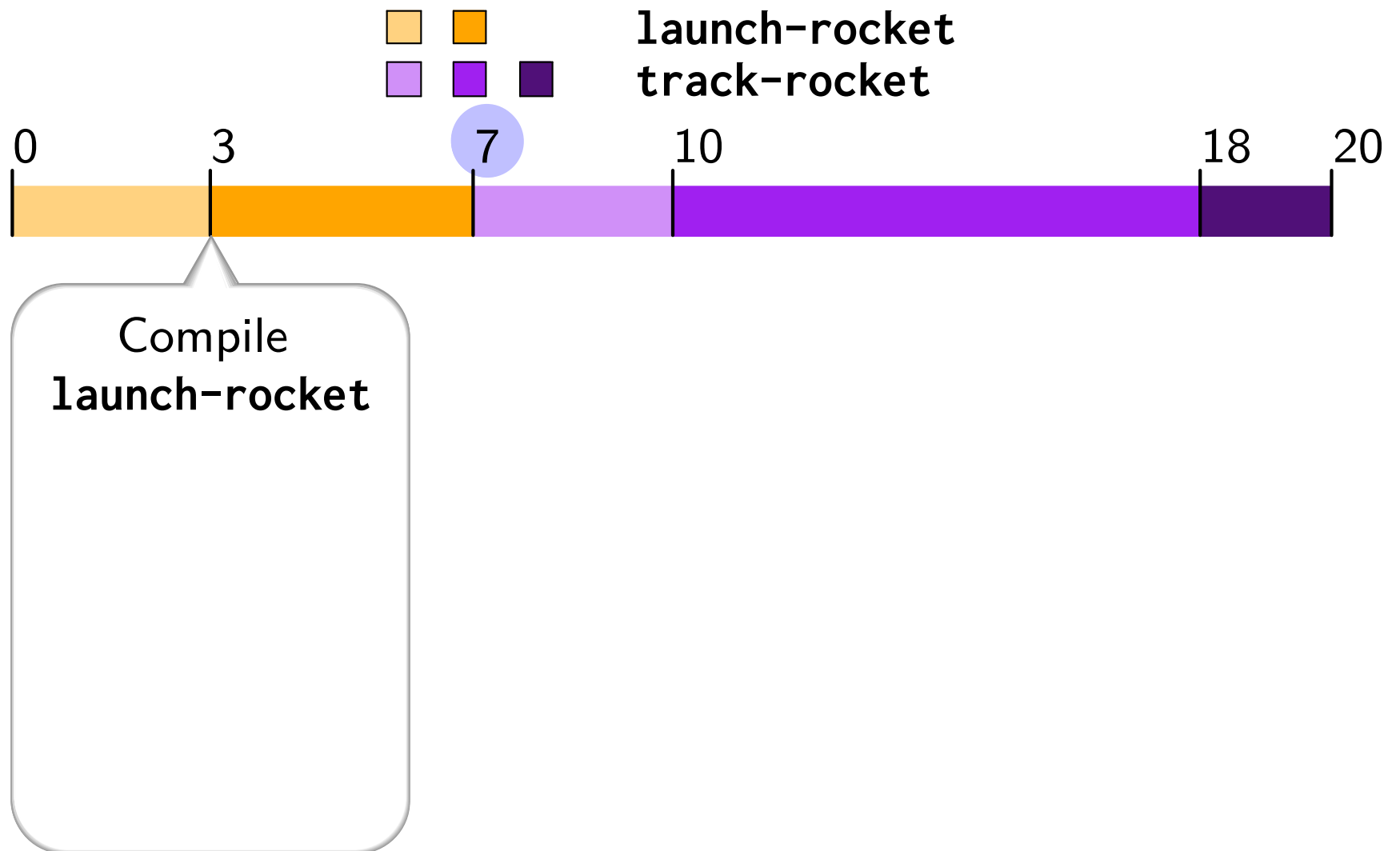
Profiling-Based Badness



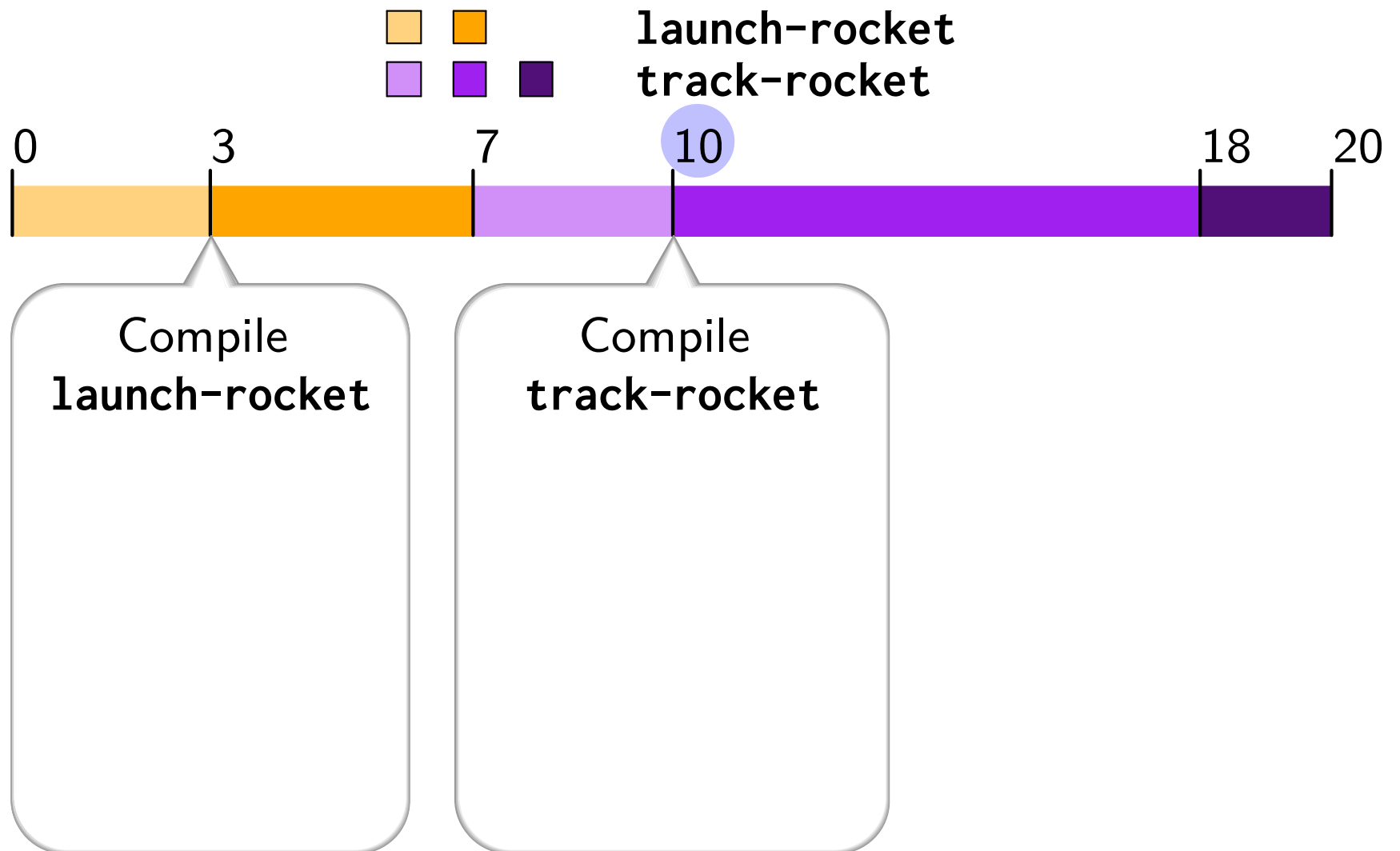
Profiling-Based Badness



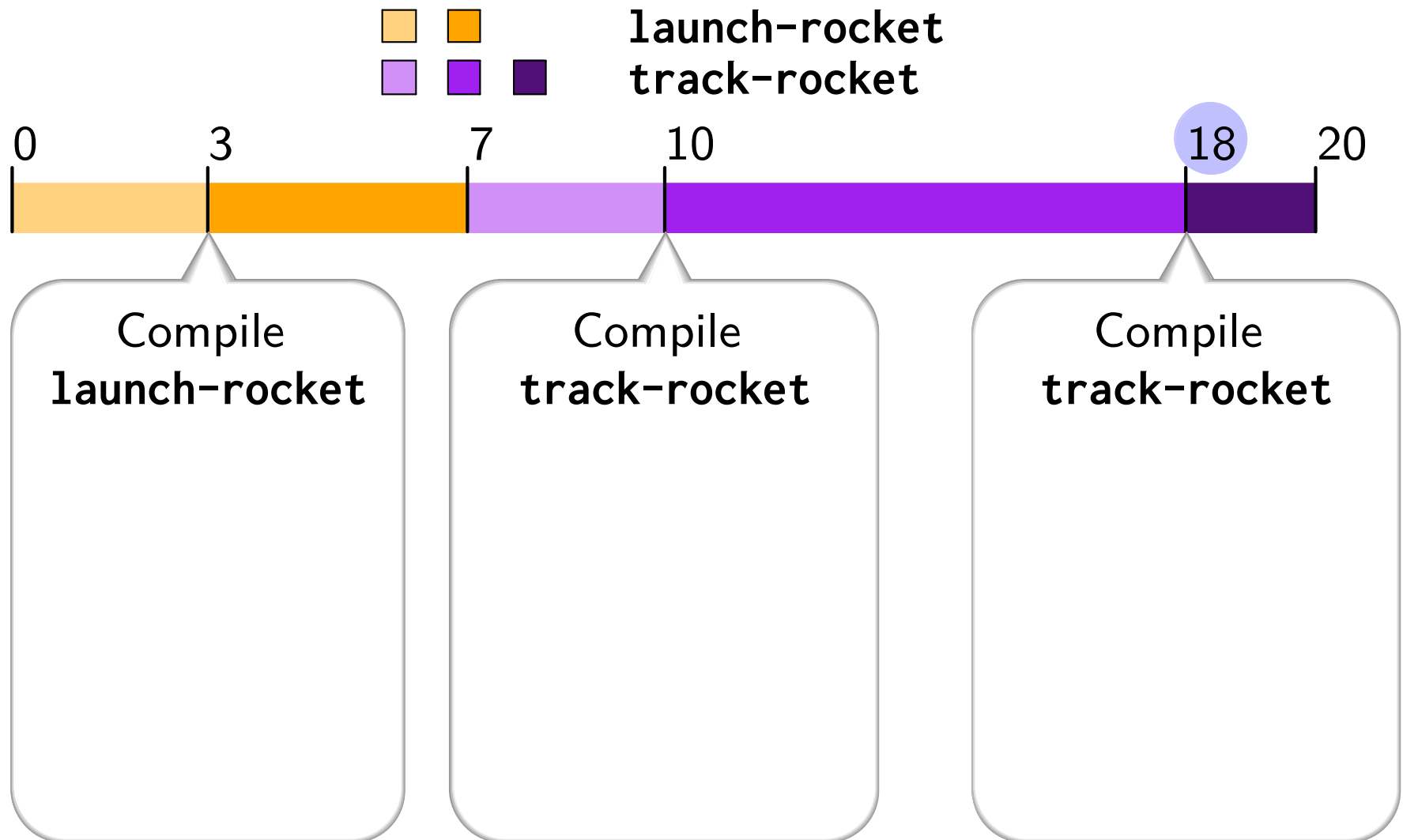
Profiling-Based Badness



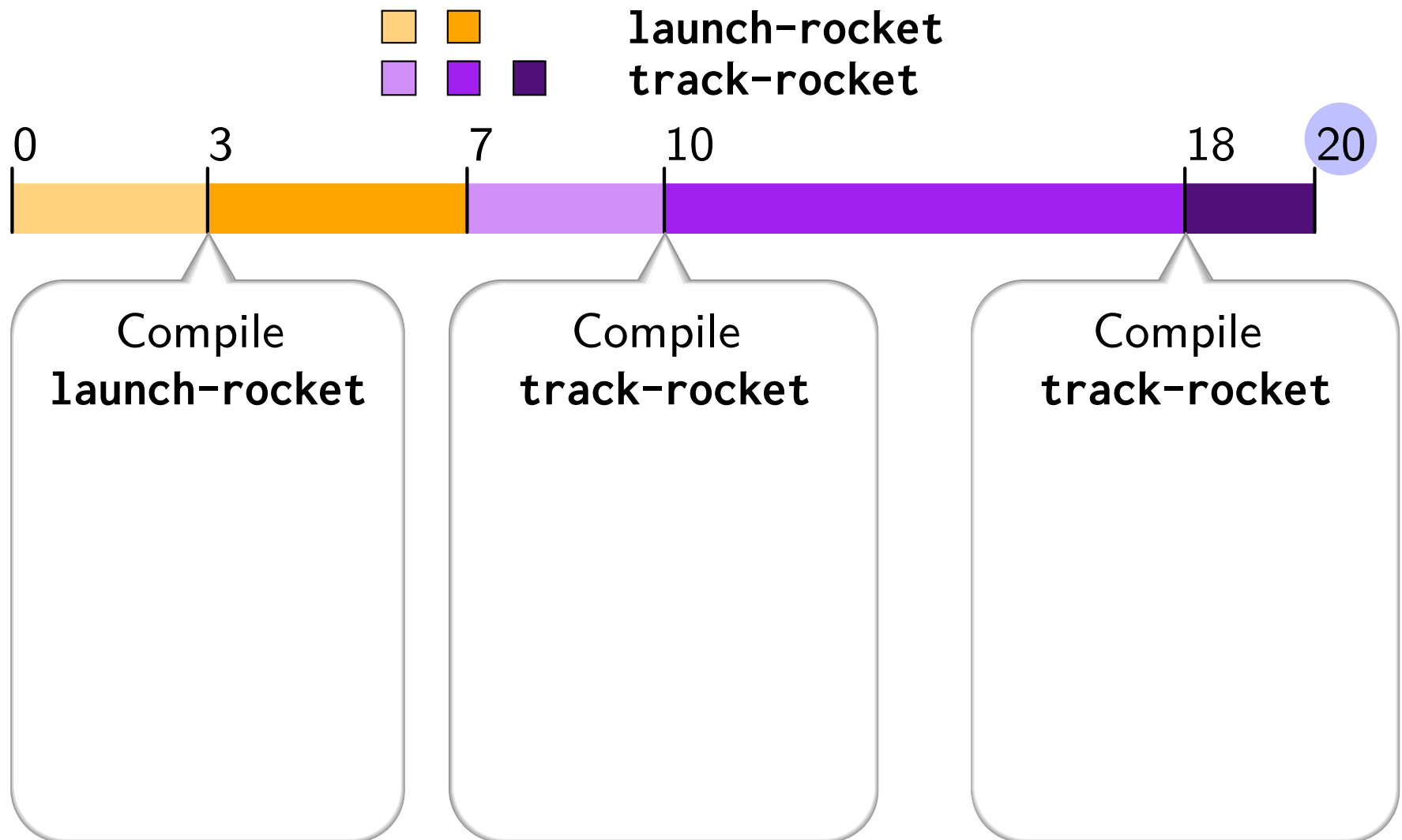
Profiling-Based Badness



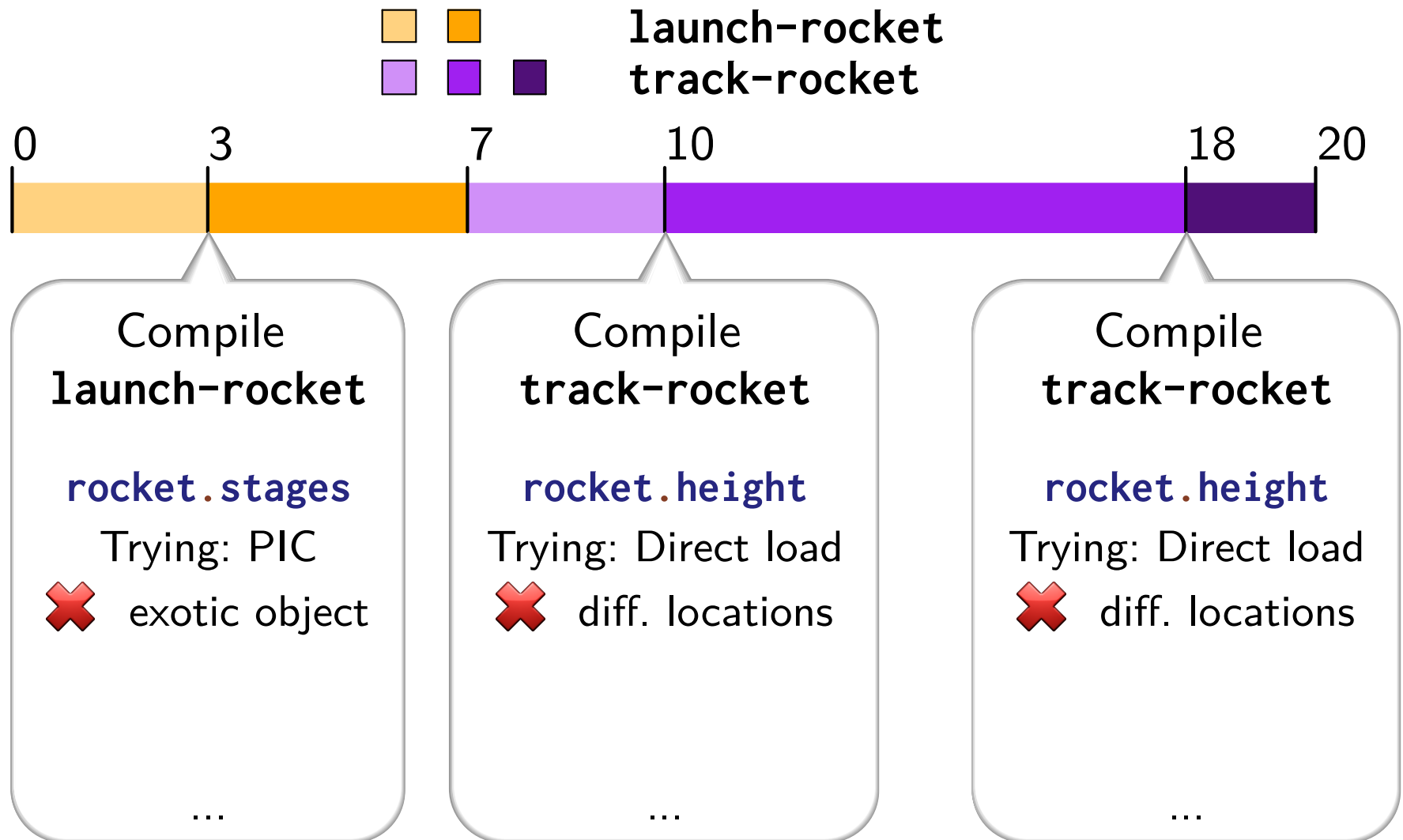
Profiling-Based Badness



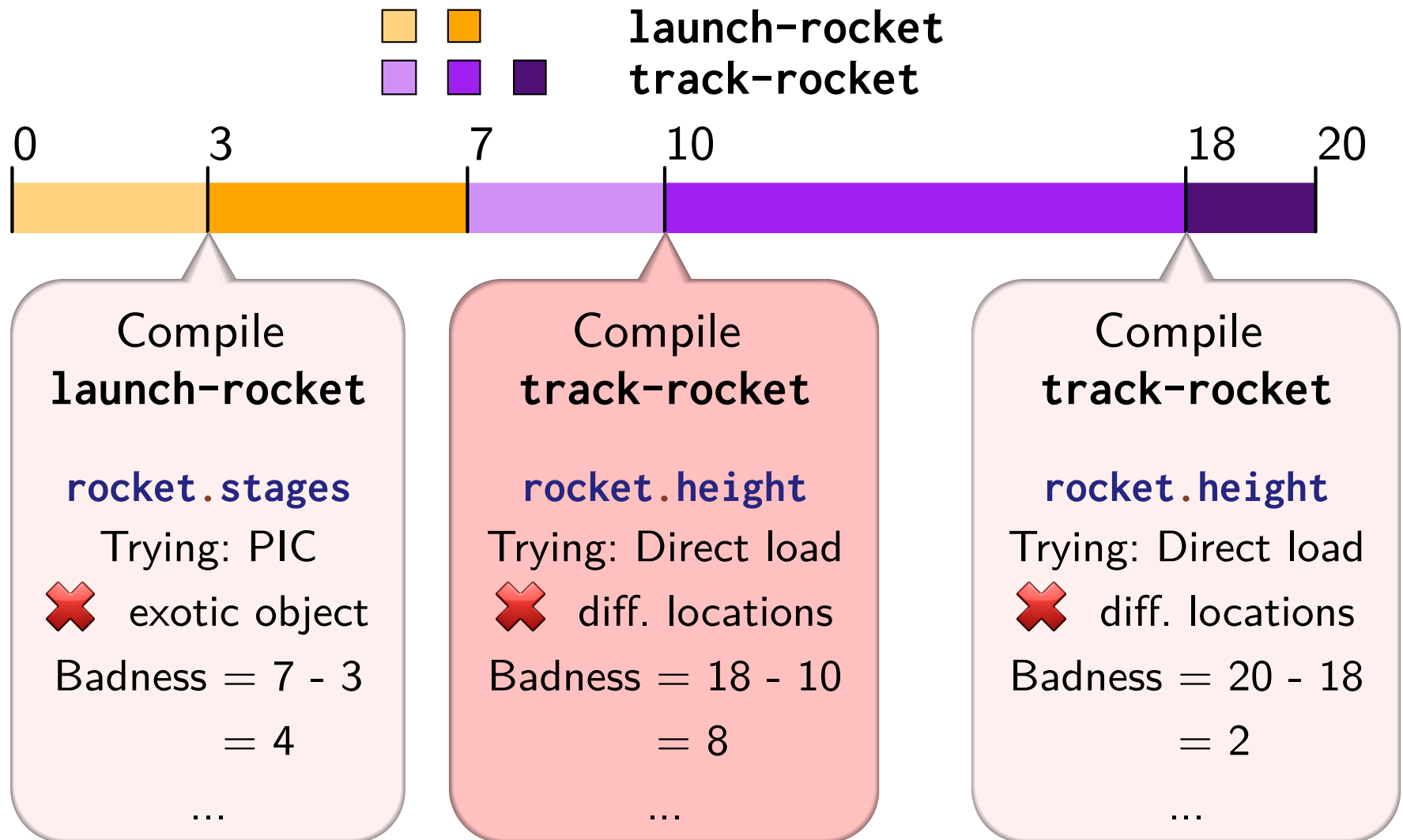
Profiling-Based Badness



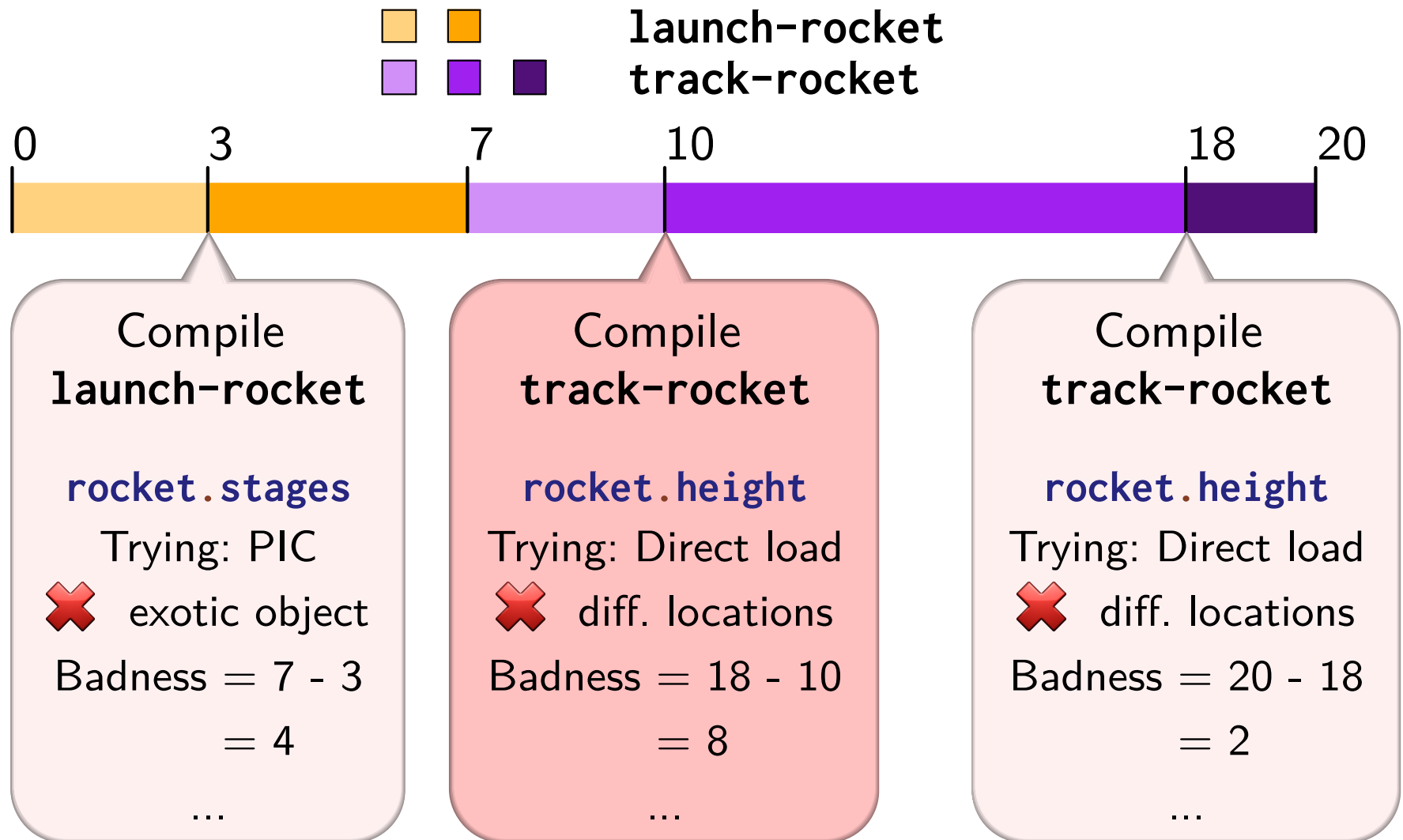
Profiling-Based Badness



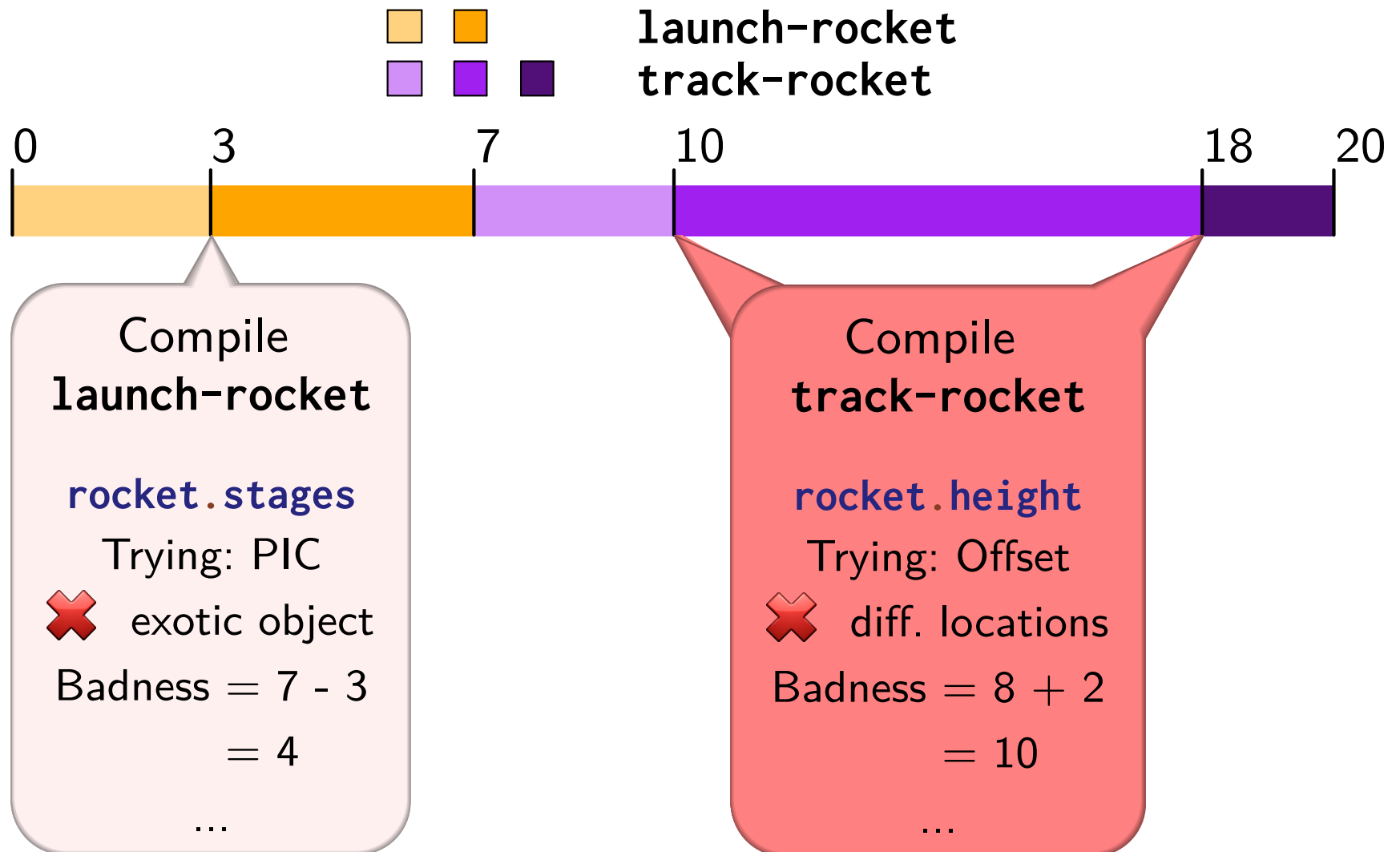
Profiling-Based Badness



Temporal Merging



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

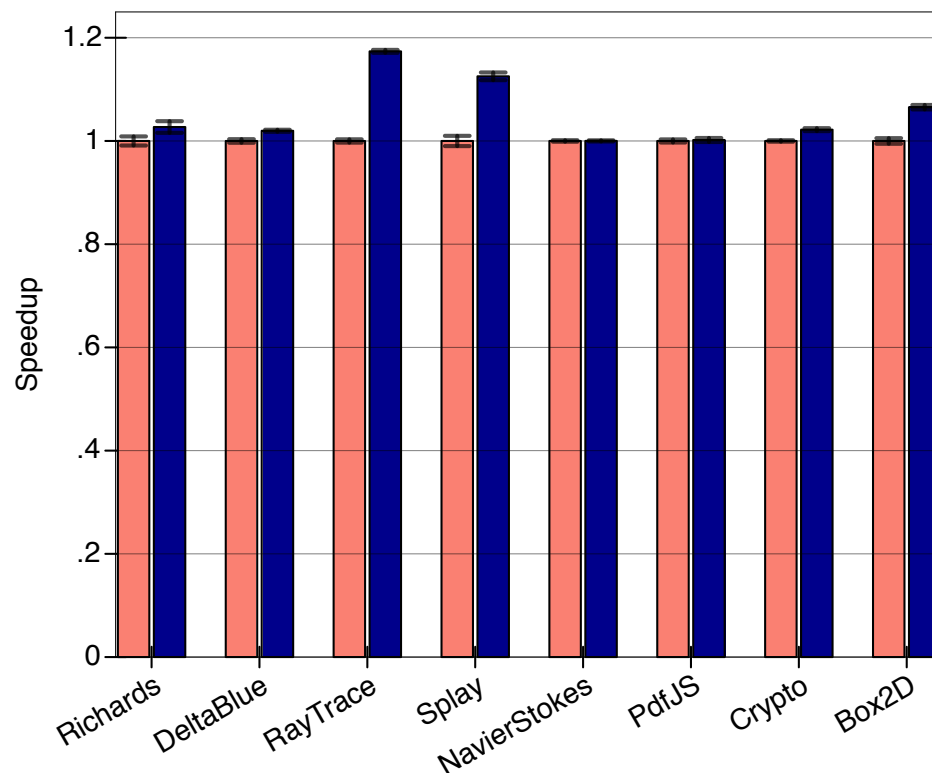
■ Baseline: Non-optimized

■ Coached: Followed recommendations (Minutes of work)

Speedup, higher is better

Experiment

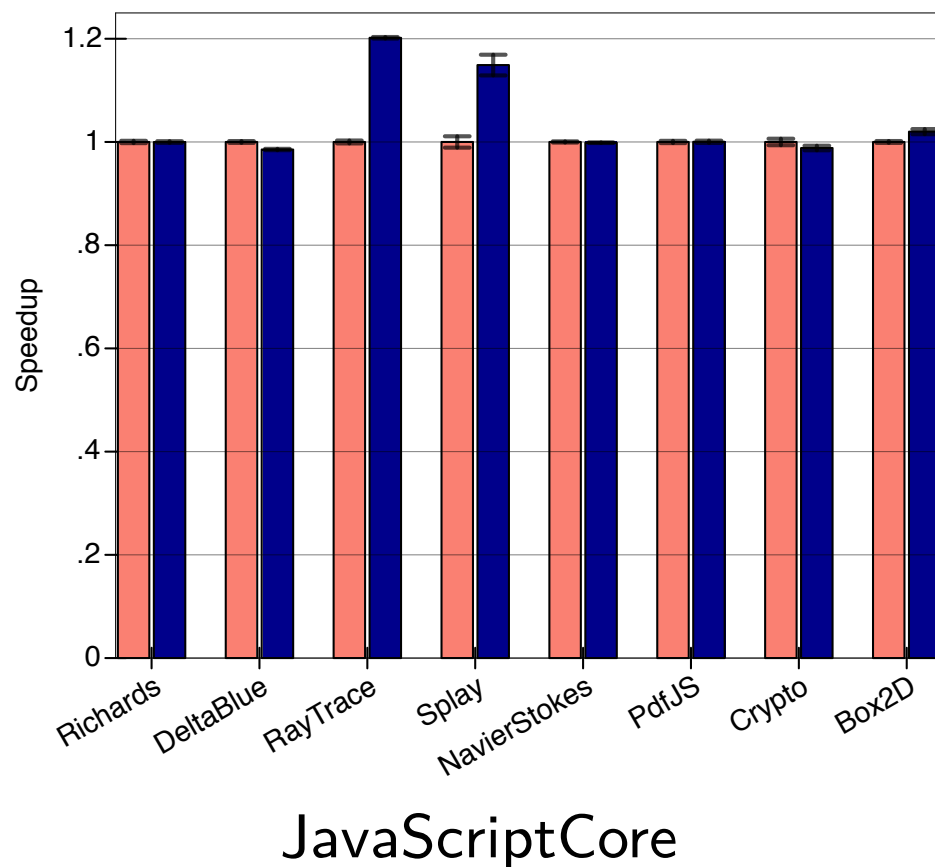
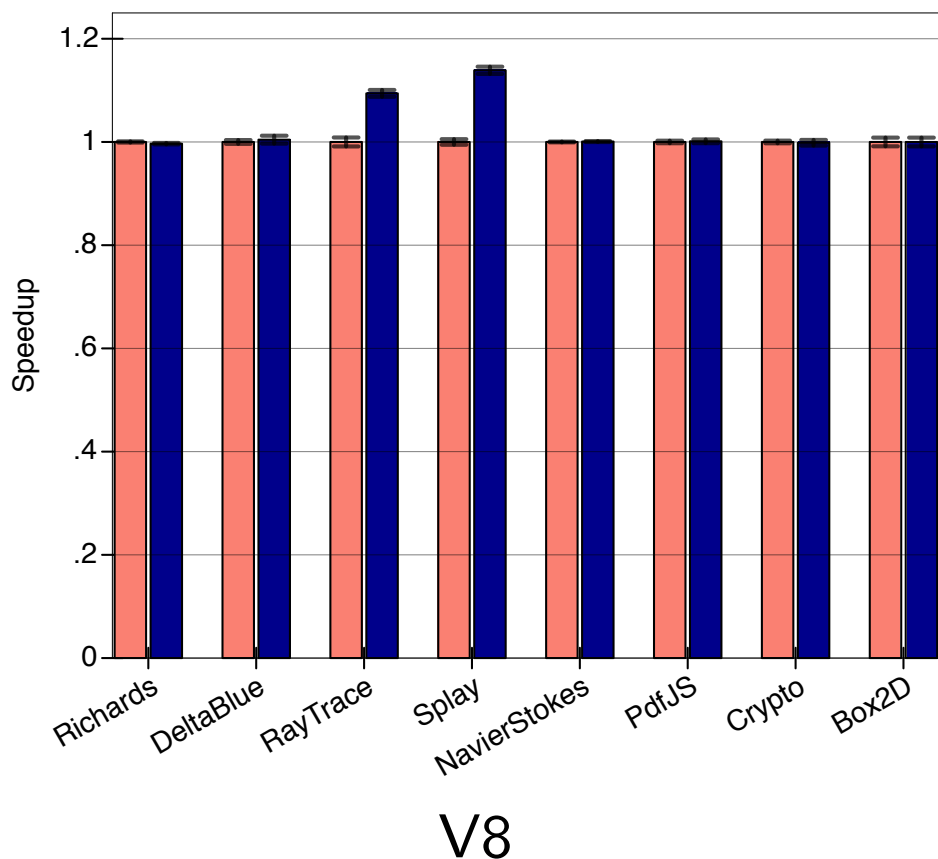
- Take Octane benchmarks
- Run the coach
- Follow recommendations
- Measure performance impact (Octane score)



Hypothesis: Coaching improves performance

- Baseline: Non-optimized
- Coached: Followed recommendations (Minutes of work)

Speedup, higher is better



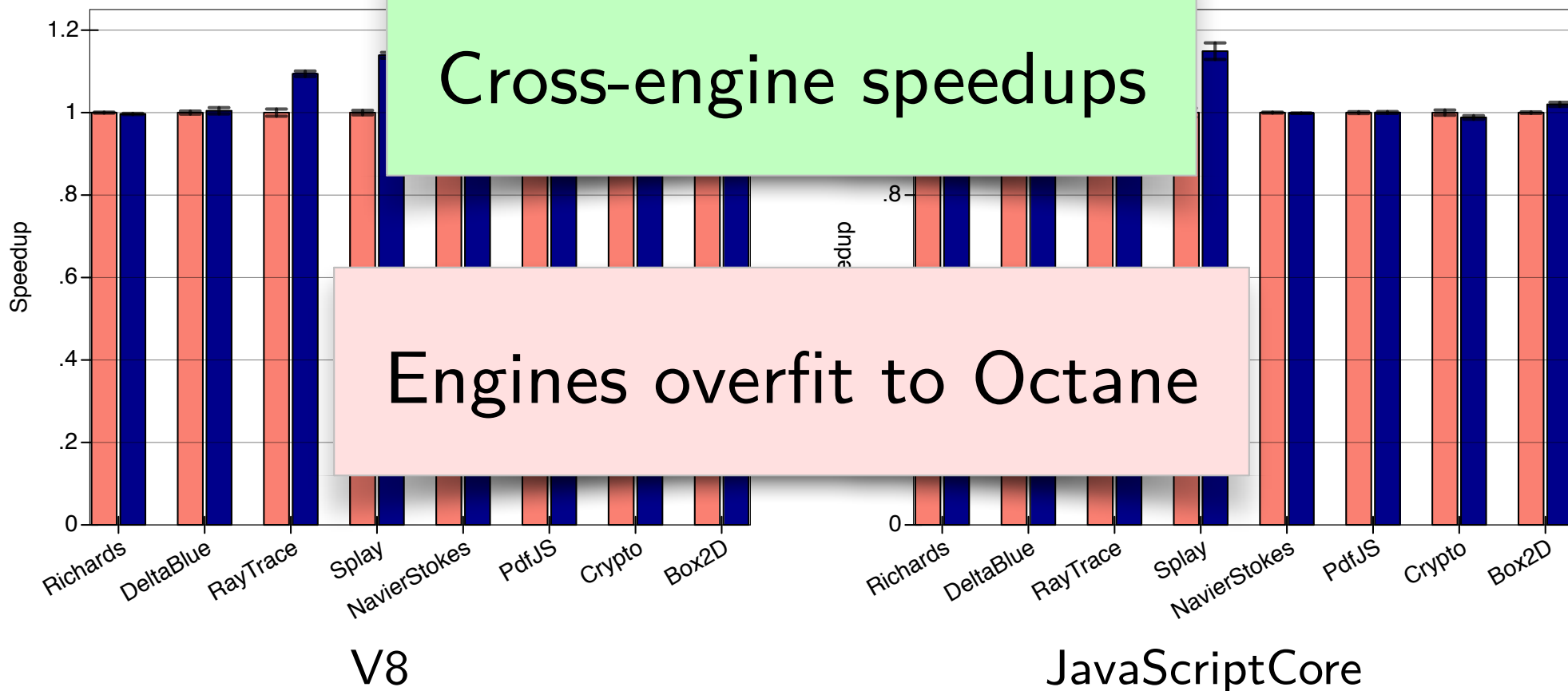
Hypothesis: Coaching improves performance

- Baseline: Non-optimized
- Coached: Followed recommendations (Minutes of work)

Speedup, higher is better

Cross-engine speedups

Engines overfit to Octane



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

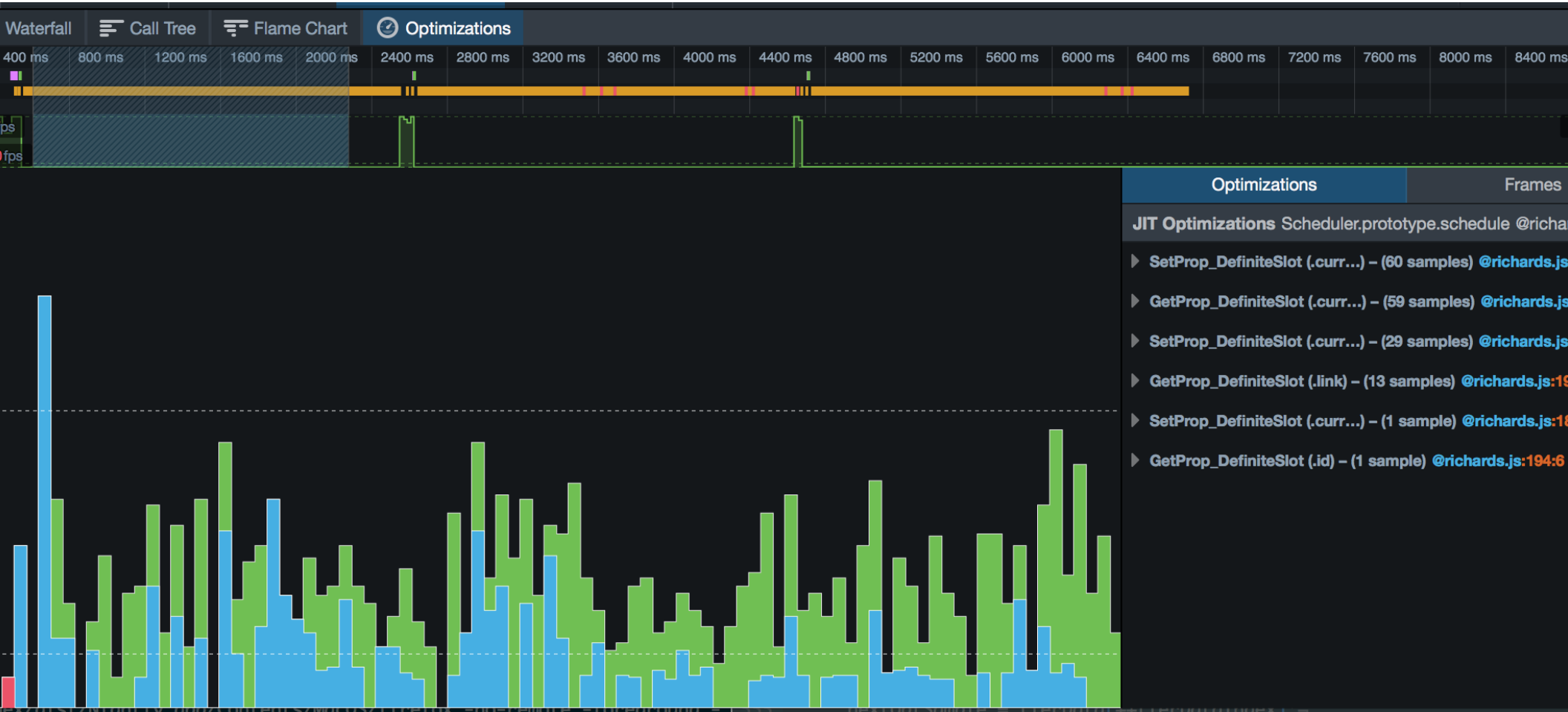
(In the Paper)

- 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 are great, but they can fail
(noun)

Runtimes gather tons of information
(noun)

Liberate it, and show it to programmers!

They may succeed where Runtimes fail
(noun)



The Coaching Philosophy

OSes are great, but they can fail
(noun)

OSes gather tons of information
(noun)

Liberate it, and show it to programmers!

They may succeed where OSes 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 ? 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 ? fail

(noun)

Thank you!