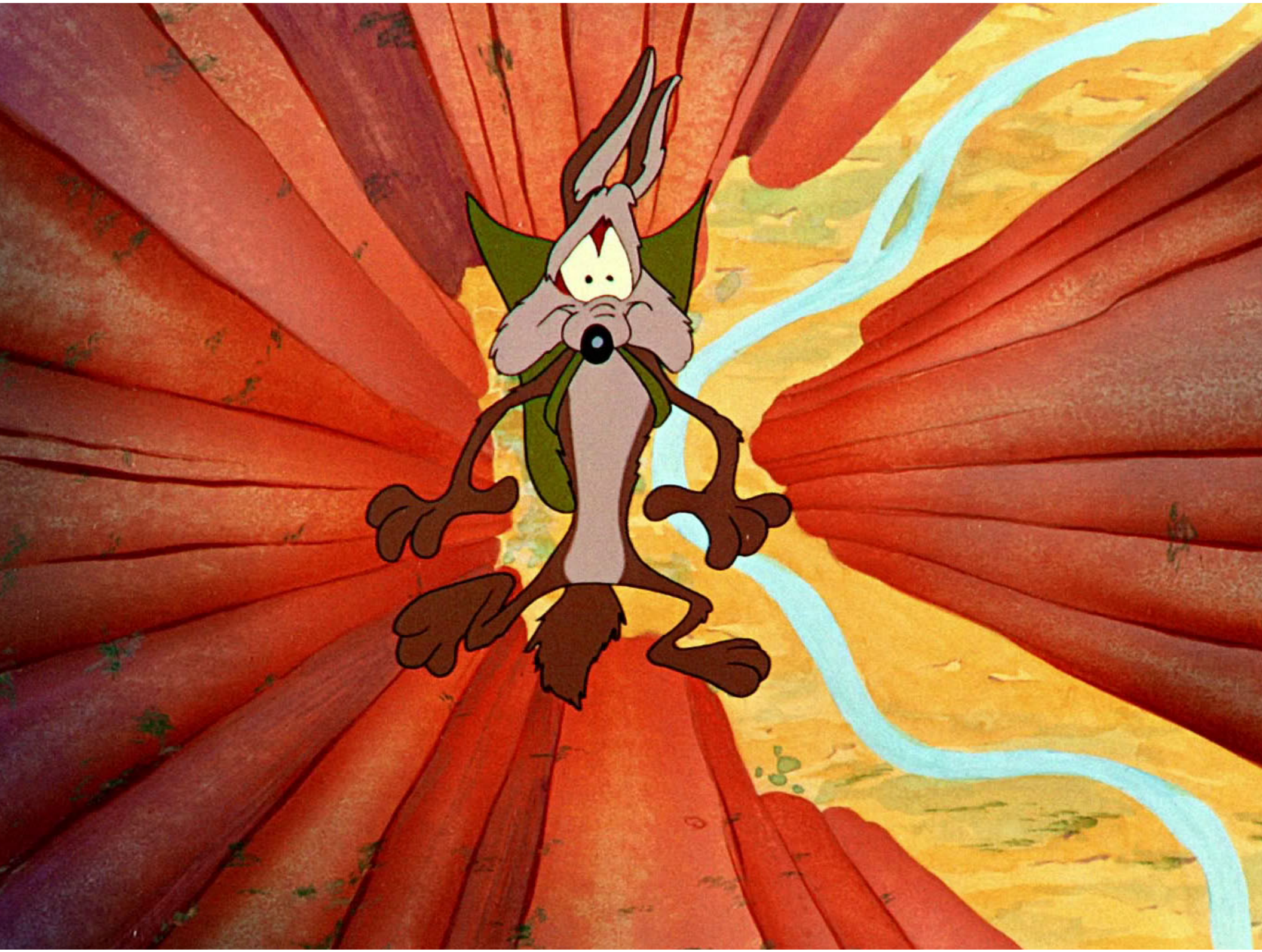
How to Generate Actionable Advice about Performance Problems

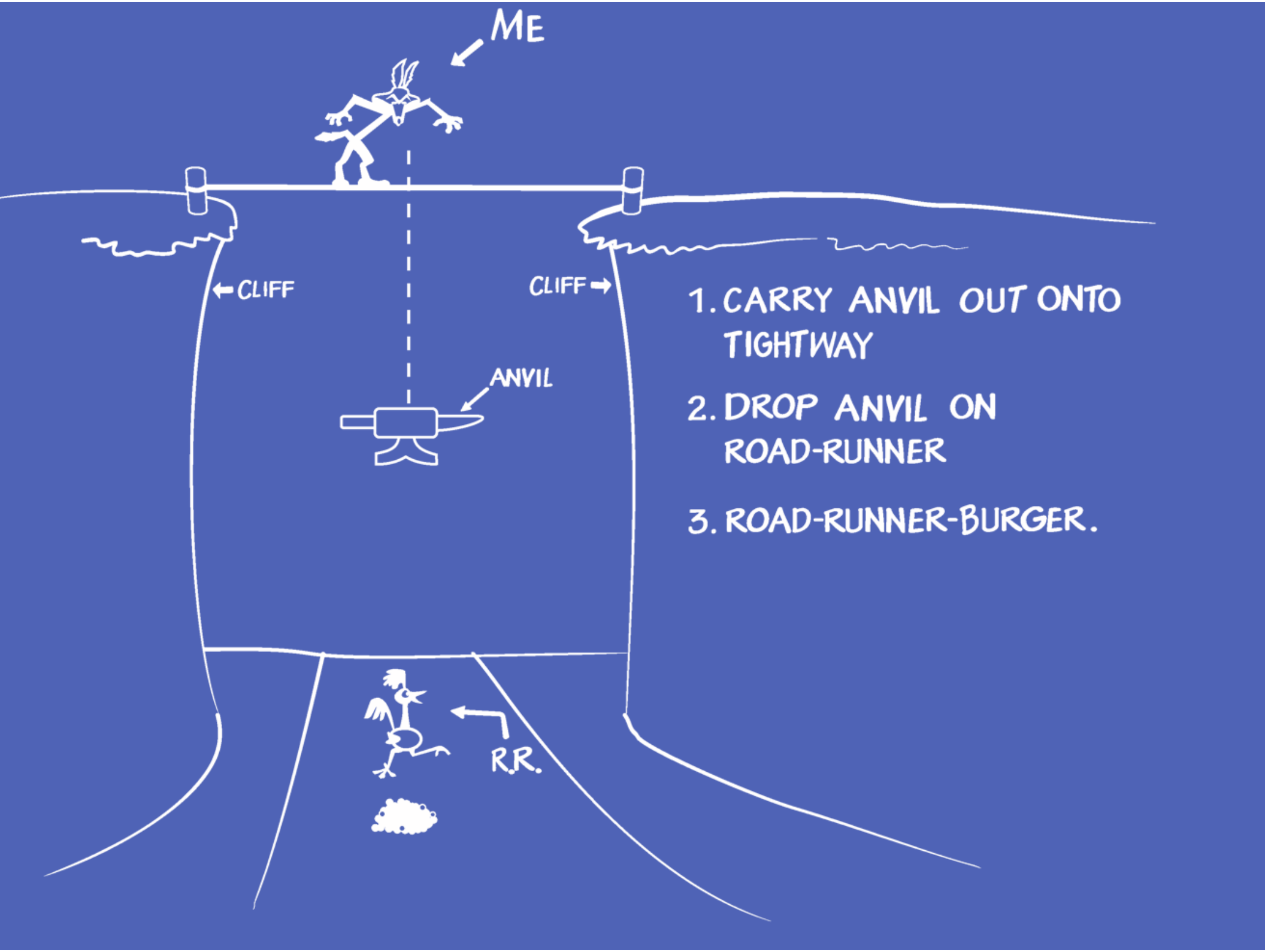
Vincent St-Amour

Northeastern University

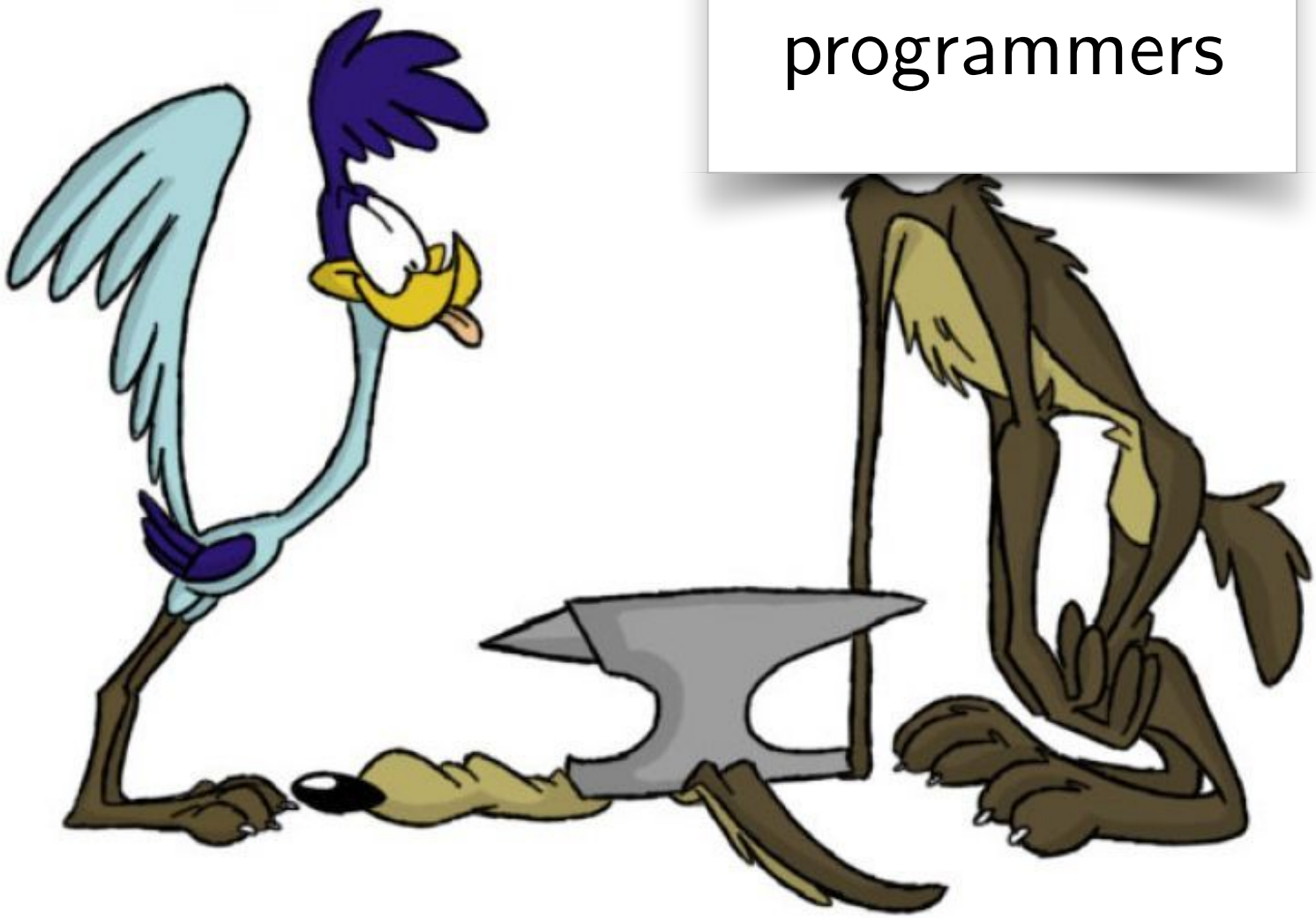
April 24th, 2015


ACME CO.




ME
CLIFF
CLIFF
ANVIL
R.R.
1. CARRY ANVIL OUT ONTO TIGHTWAY
2. DROP ANVIL ON ROAD-RUNNER
3. ROAD-RUNNER-BURGER.

Non-expert programmers

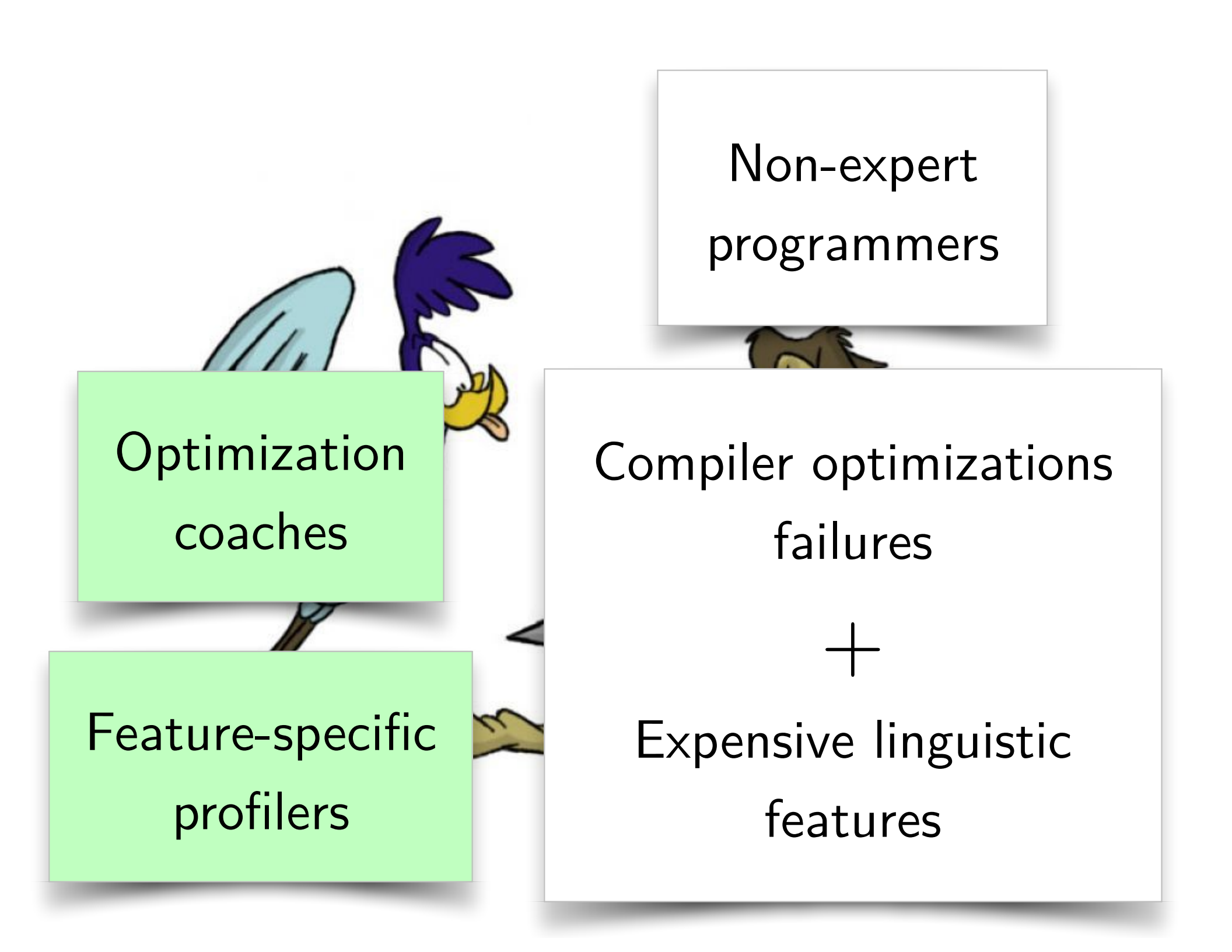


Non-expert programmers

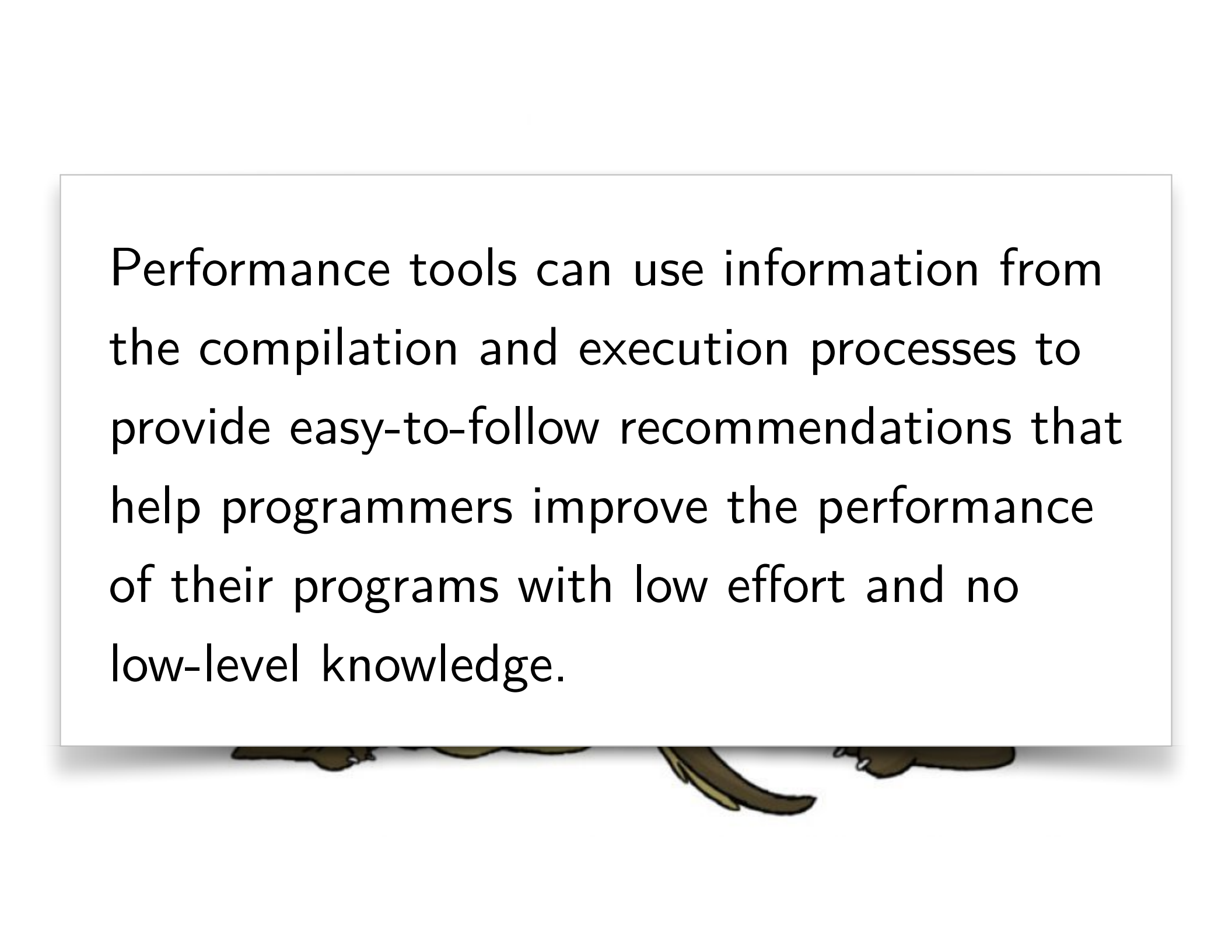
Compiler optimizations failures

+

Expensive linguistic features


Non-expert programmers
Optimization coaches
Compiler optimizations failures
+
Expensive linguistic features
Feature-specific profilers

Performance tools can use information from the compilation and execution processes to provide easy-to-follow recommendations that help programmers improve the performance of their programs with low effort and no low-level knowledge.



Today's menu

Optimization Coaching

- for Racket [OOPSLA 2012]
- for JavaScript [ECOOP 2015]

Feature-Specific Profiling

- for Racket [CC 2015]

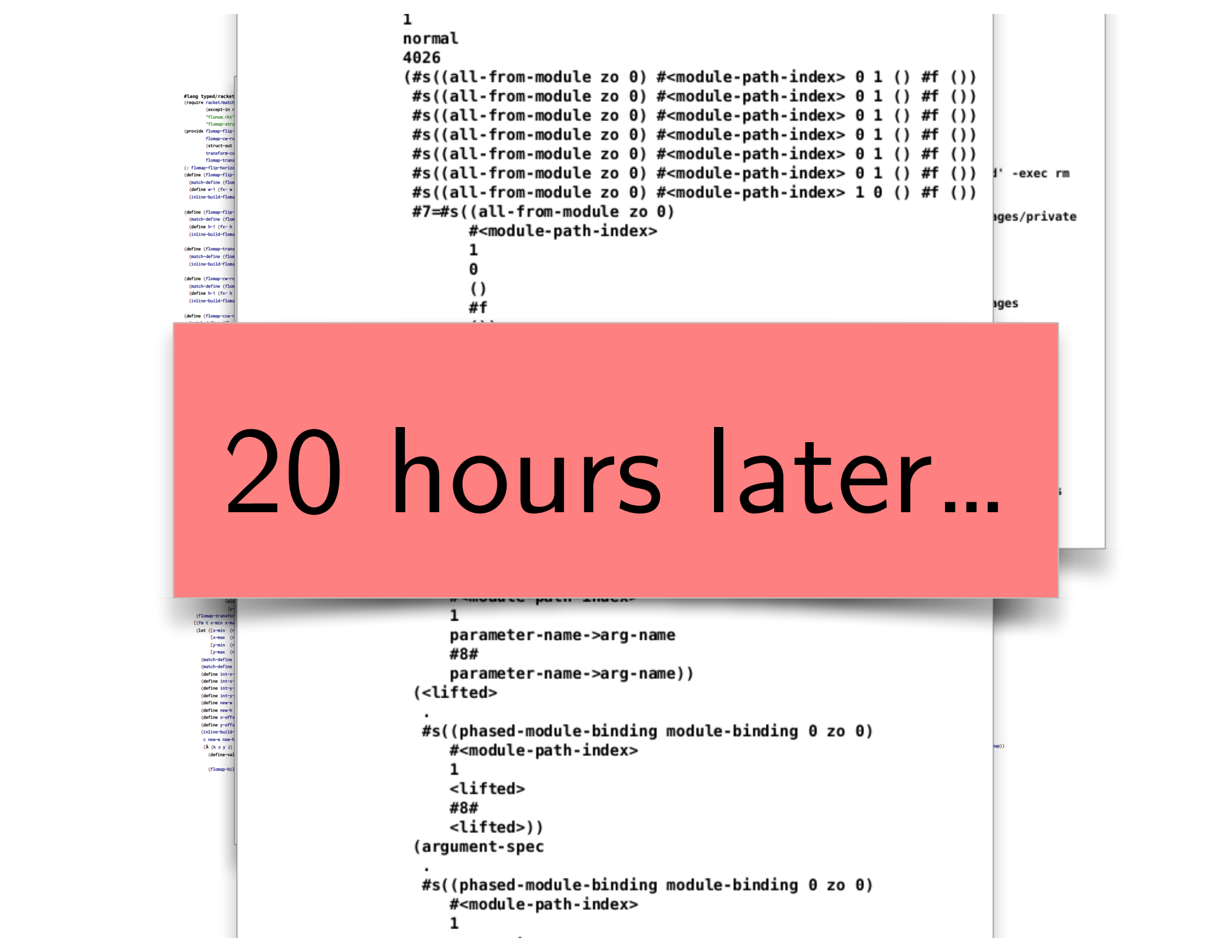
What is Optimization Coaching?


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 ->
(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->
                  (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))))
; ;; ===================================================================================
;  (: 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))]))
```

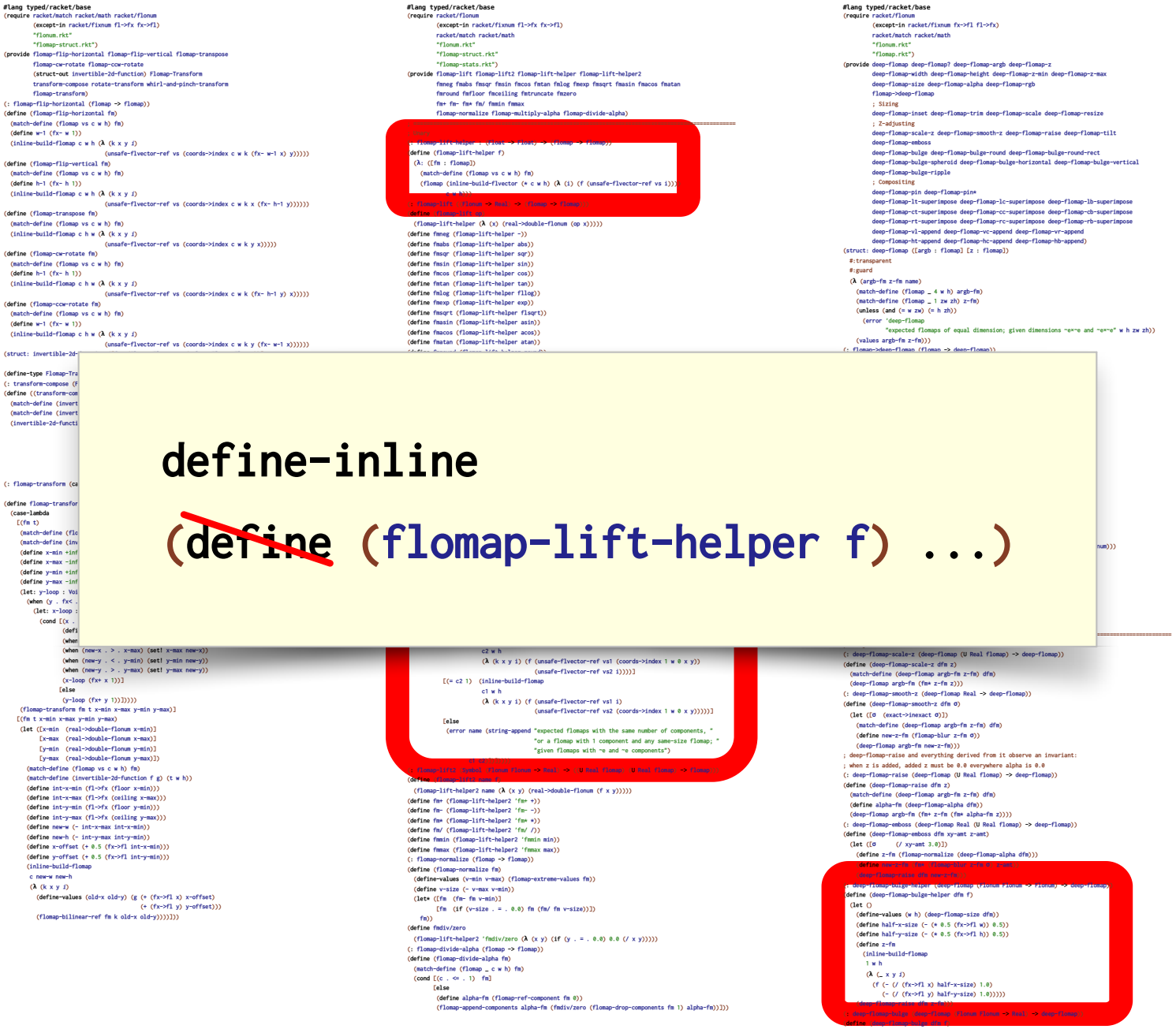
```
#lang typed/racket/base
(require racket/flonum
         (except-in racket/fixnum fx->fl fl->fx)
         racket/match racket/math
         "flonum.rkt"
         "flomap.rkt")
(provide deep-flomap deep-flomap? deep-flomap-argb deep-flomap-z
         deep-flomap-width deep-flomap-height deep-flomap-z-min 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 ~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)))
(: de                                    deep-flomap -> Nonnegative-Fixnum))
(def                                     idth dfm))
                                         dth (deep-flomap-argb dfm)))
                                         onnegative-fixnum?]))

(: d                                     (deep-flomap -> Nonnegative-Fixnum))
(de                                      eight dfm))
                                         ight (deep-flomap-argb dfm)))
                                         onnegative-fixnum?]))

(: d                                     (deep-flomap -> Flonum))
(de                                      z-min dfm))
                                         (deep-flomap -> Flonum))
/de                                      z-max dfm))
                                         (deep-flomap -> (values Nonnegative-Fixnum Nonnegative-Fixnum)))
                                         -size dfm))
                                         ap-width dfm) (deep-flomap-height dfm)))
                                         e (deep-flomap -> flomap))
                                         -alpha dfm))
                                         ent (deep-flomap-argb dfm) 0))
                                         (deep-flomap -> 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)))))
```

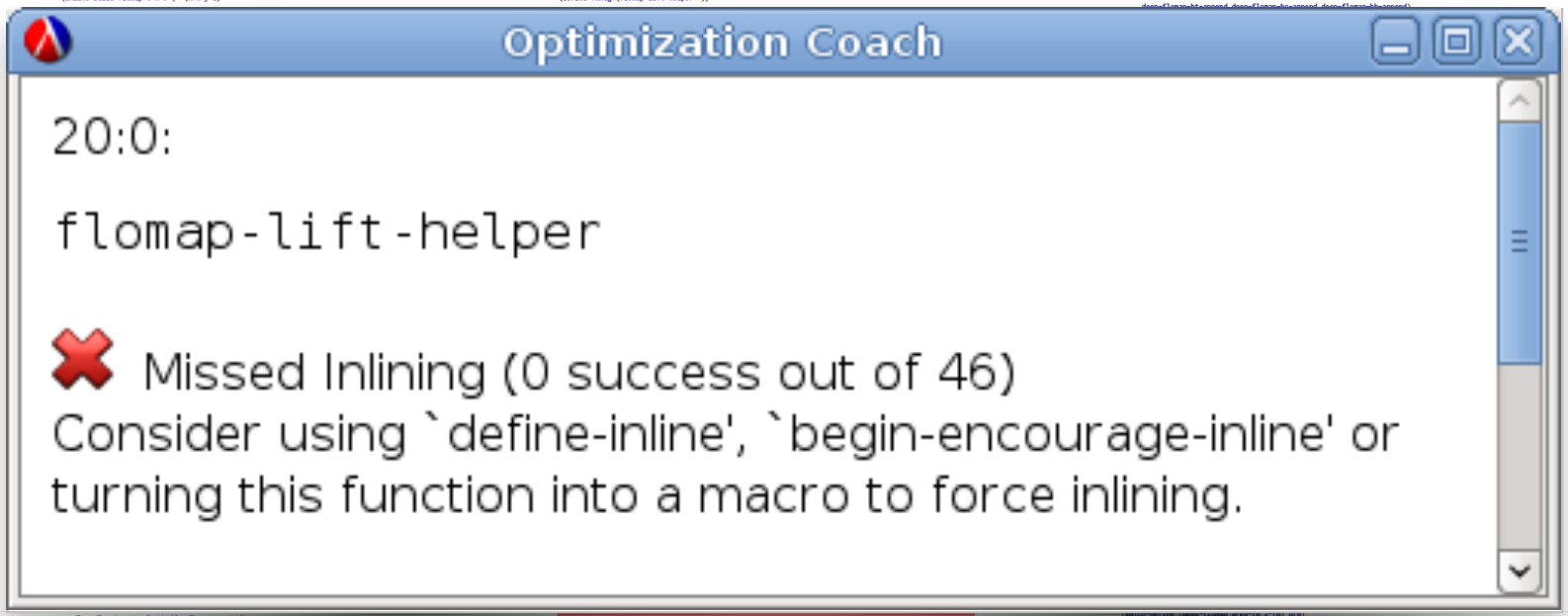

20 hours later...

```
#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))))]))
```

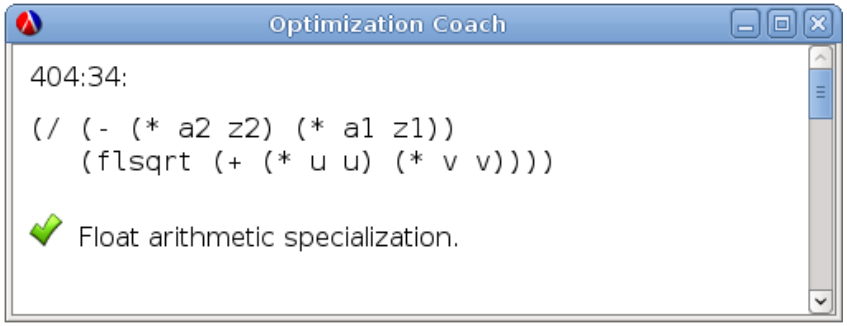
```
#lang typed/racket/base
(require racket/flonum
         (except-in racket/fixnum fl->fx fx->fl)
         racket/match racket/math
         "flonum.rkt"
         "flomap-struct.rkt"
         "flomap-stats.rkt")
(provide flomap-lift flomap-lift2 flomap-lift-helper flomap-lift-helper2
         fmneg 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)

;; ---------------------------------------------------------------------------------------------------
; 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 x0)
  (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))))

;; ---------------------------------------------------------------------------------------------------
; 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 x)
  (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 smoothed-fm (flomap-blur z-fm σ))
    (deep-flomap-raise dfm (fm* z-fm (fm- z-fm smoothed-fm)))))
; deep-flomap-bulge-helper : (deep-flomap (flomap flomap -> flomap) -> 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 cx cy)
  (deep-flomap-bulge-helper dfm (λ (cx cy) (real->double-flonum (f cx cy)))))
```

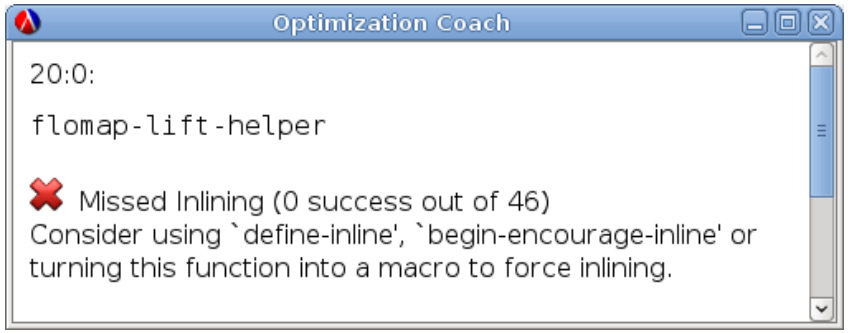

define-inline
(define (flomap-lift-helper f) ...)


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.

Dialog between compilers and programmers

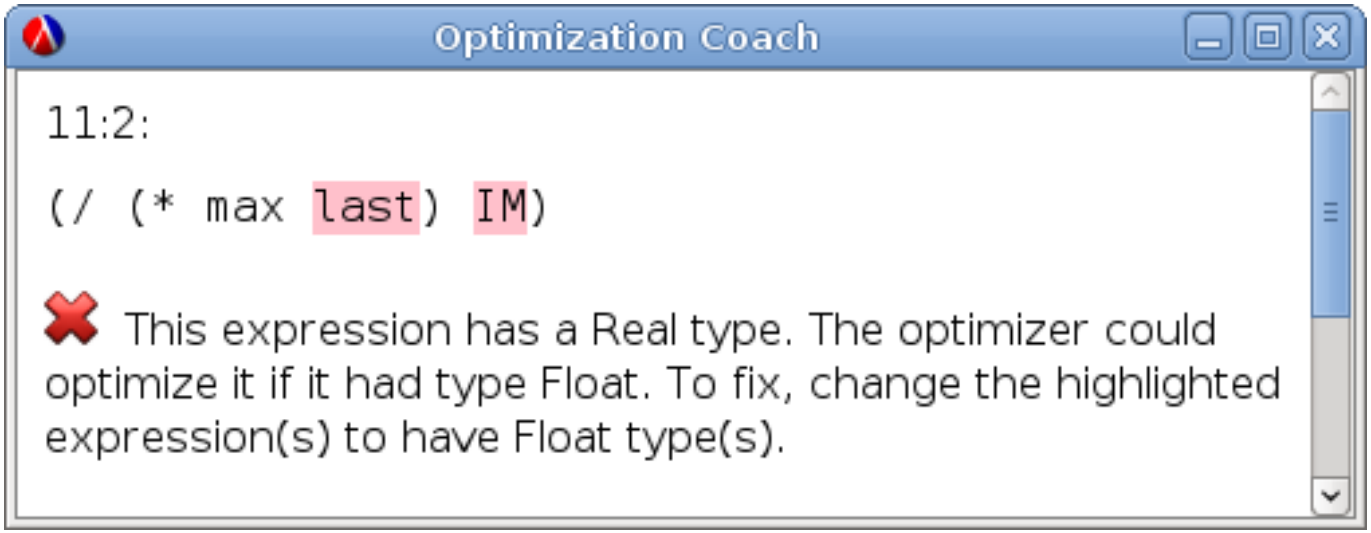


Successes



Near misses + Recommendations

Compilers must be conservative



Recommendations can **change semantics**!

```
(/  1    0)    ➡   /: division by zero
(/  1.0  0.0)  ➡   +inf.0
```

Coaching for Racket

Type-Driven Specialization

```
#lang typed/racket

(define IM   139968)
(define IA     3877)
(define IC    29573)

(define last 42)
(define min  35.3)
(define max  156.8)

(define (gen-random)
  (set! last (modulo (+ (* last IA) IC) IM))
  (+ (/ (* (- max min) last) IM) min))
```

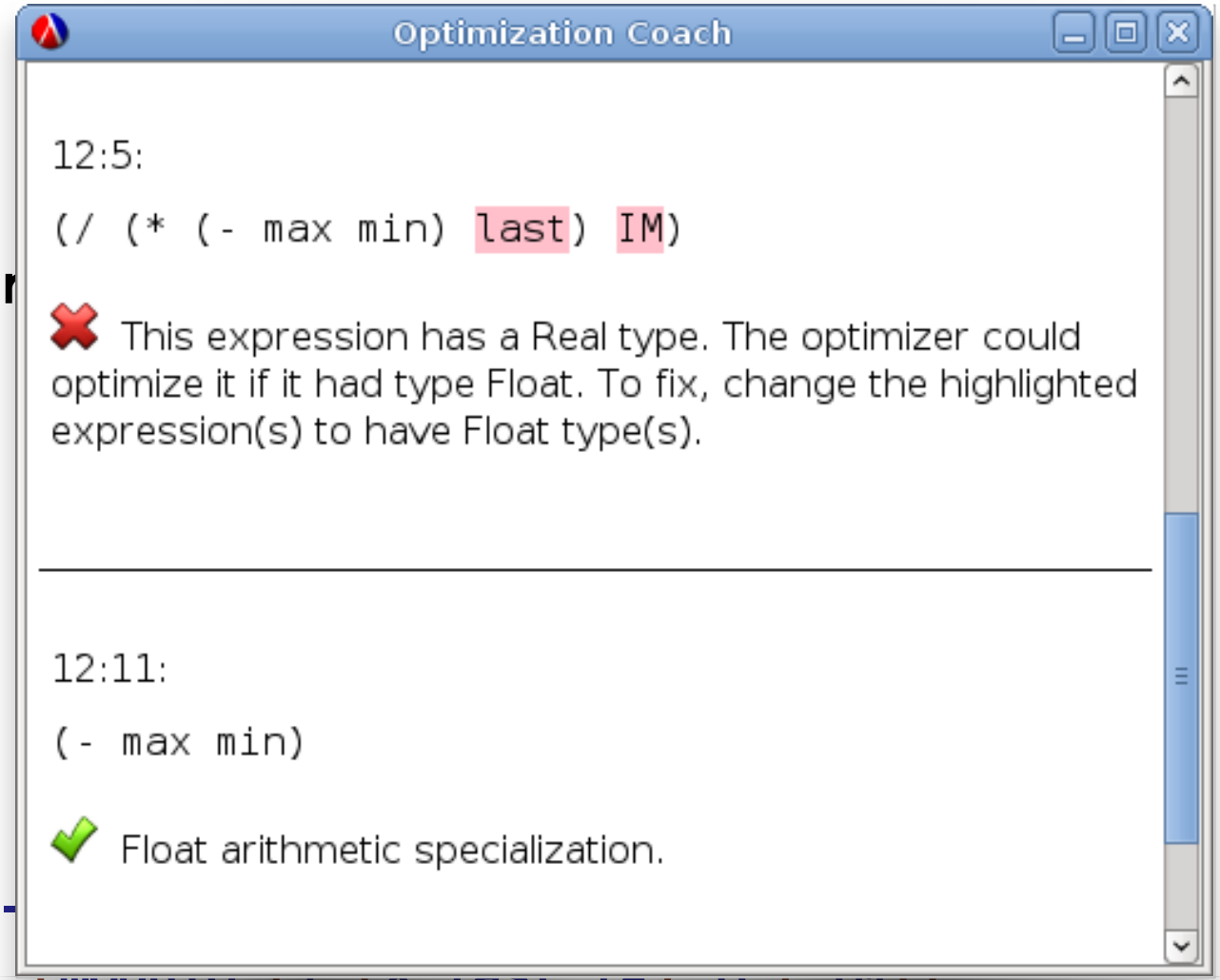
Type-Driven Specialization

```
#lang typed/racket

(define IM
(define IA
(define IC

(define last 42)
(define min  35.3)
(define max  156.8)

(define (gen-random)
  (set! last (modulo (+ (* last IA) IC) IM))
  (+ (/ (* (- max min) last) IM) min))
```

```
fl-
(- <Float> <Float>)
```

Float Float

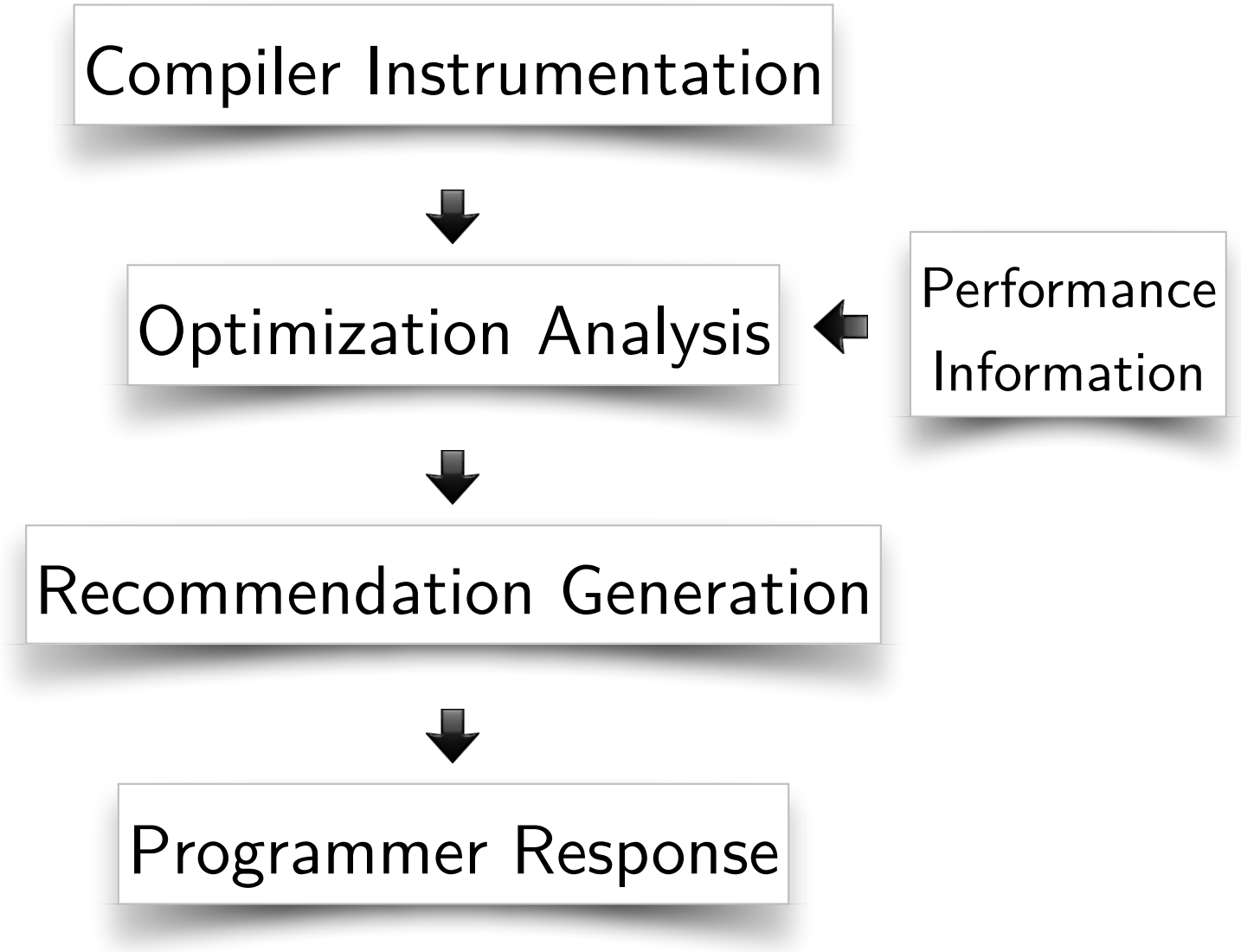
Type-Driven Specialization




```
#lang typed/r

(define IM
(define IA
(define IC

(define last
(define min
(define max

(define (gen-
  (set! last
  (+ (/ (* (- max min) last) IM) min))
```

Float Integer Integer

Coach Architecture



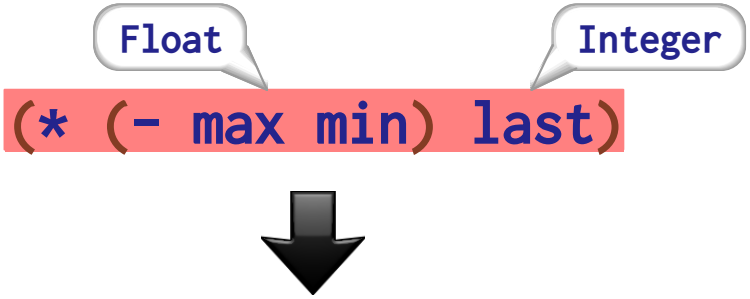
Compiler Instrumentation

```
fl-
(+ <Float> <Float>)
```

```
(fl- max min)
```

```
TR opt: prng-example.rkt 12:11
  (- max min)
  Float Float
  binary float subtraction
```

Compiler Instrumentation




```
(* <Number> <Number>) ; no change
```

```
(* (- max min) last)
```

```
TR opt failure: prng-example.rkt 12:8
  (* (- max min) last)
  Float Integer
  generic multiplication
```

Optimization Analysis

Optimization-agnostic techniques
+ Optimization-specific heuristics

Pruning

Incomprehensible failure pruning
Irrelevant failure pruning
Optimization proximity
Harmless failure pruning
Partial success short-circuiting
Profiling-based pruning

Merging

Causality merging
Locality merging
Temporal merging
Same-property analysis
By-solution merging
By-constructor merging

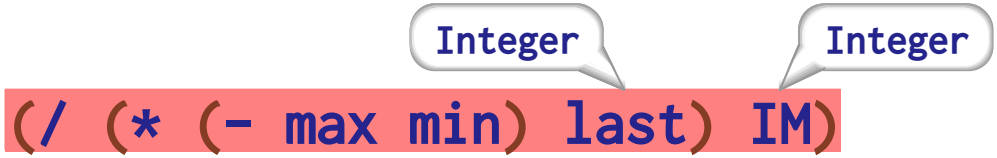
Targeting

Local reporting
Non-local reporting
Solution-site inference

Ranking

Static badness
Profiling-based badness

Recommendation Generation



Float

Float ~~Integer~~

Irritant

12:5:

```
(/ (* (- max min) last) IM)
```

This expression has a Real type. The optimizer could optimize it if it had type Float. To fix, change the highlighted expression(s) to have Float type(s).

Programmer Response

```
                    (->fl last)  (->fl IM)
(+ (/ (* (- max min) last) IM) min)
```

12:5:

```
(/ (* (- max min) last) IM)
```

This expression has a Real type. The optimizer could optimize it if it had type Float. To fix, change the highlighted expression(s) to have Float type(s).

How well does it work?

Hypothesis: Coaching improves performance

Experiment

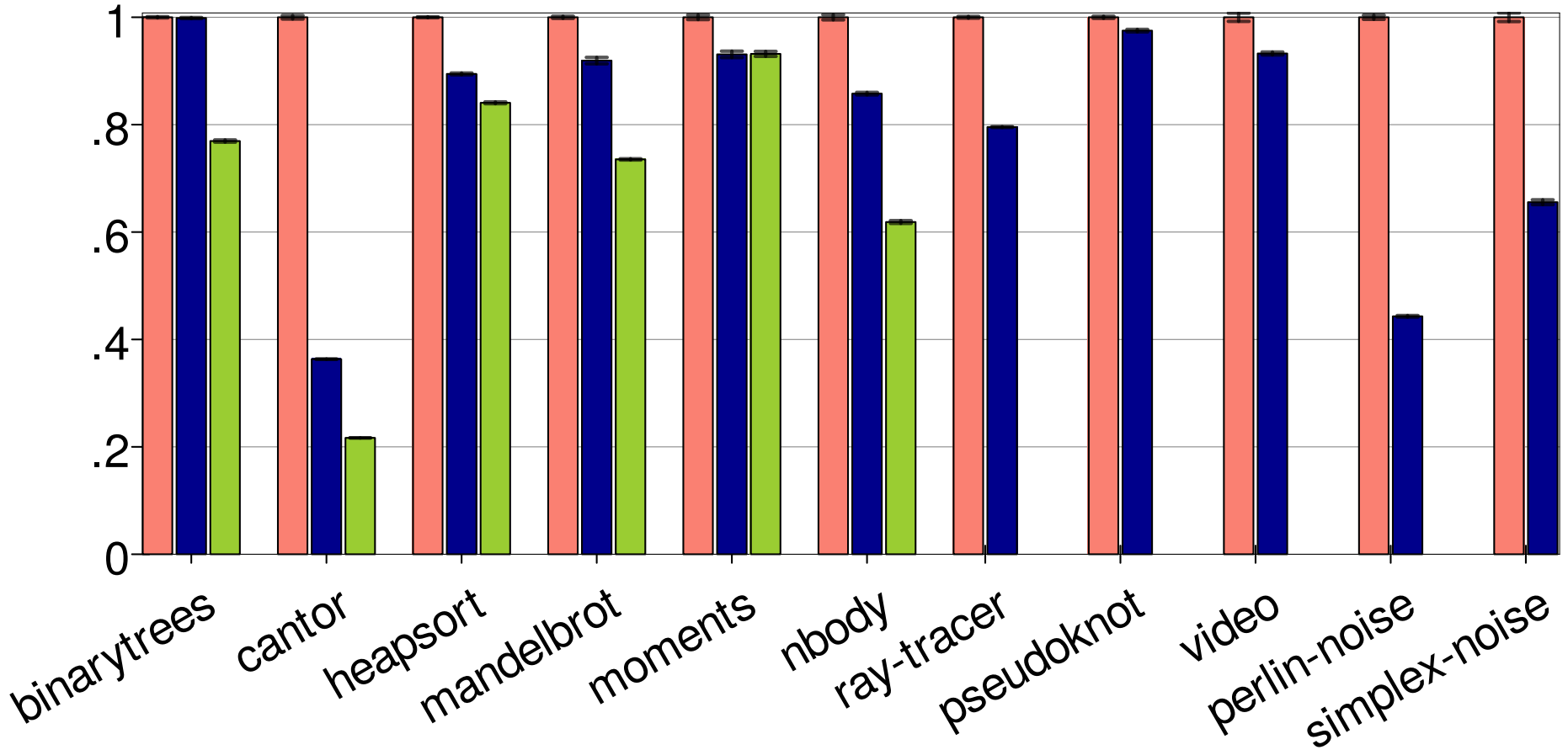
- Take existing Racket programs
- Run Optimization Coach
- Follow recommendations
- Measure performance impact (running time)
- Compare with versions hand-optimized by experts

50%	Expert speedups
1%	Work
0%	Email traffic

Hypothesis: Coaching improves performance

Baseline:	Non-optimized	
Coached:	Followed recommendations	(Minutes of work)
Gold standard:	Hand-optimized by experts	(Days of work)

Execution time, lower is better



Optimization Coach today

- Plugin for the DrRacket IDE
- **`raco pkg install optimization-coach`**
- "I have bug reports, therefore I exist."

  – Matthias Felleisen

Coaching for JavaScript

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

Property Access Optimizations

`object.prop` 

 Dereference static offset

 Polymorphic inline cache

 Virtual machine call

Failure causes

- Non-uniform initialization

```
rocket.height = 100
rocket.speed  = 20
```

vs

```
rocket.speed  = 20
rocket.height = 100
```

- Inconsistent property location (next slide)
- Polymorphism
- ...

Inconsistent Property Location

```
// constructor
function IntersectionInfo () {
   this.color = "black"
}

IntersectionInfo.prototype = {
   isHit: false,
   position: null,
   ...
}

...  if (D > 0) info.isHit = true;  ...

...  if (info.isHit) return info.color;  ...
```

Inconsistent Property Location

```
// constructor
function IntersectionInfo () {
   this.color = "black"
}

IntersectionInfo.prototype = {
   isHit: false,
   position: null,
   ...
}

...  if (D > 0) info.isHit = true;  ...

...  if (info.isHit) return info.color;  ...
```

Inconsistent Property Location

```
// constructor
function Intersec
   this.color = "
}

IntersectionInfo.
   isHit: false,
   position: null
   ...
}

...  if (D > 0) info.isHit = true;  ...

...  if (info.isHit) return info.color;  ...
```

badness: 2596

affected property: isHit

This operation needs to walk the prototype chain to find the property.

Try putting the property in the same location for all objects.

Inconsistent Property Location

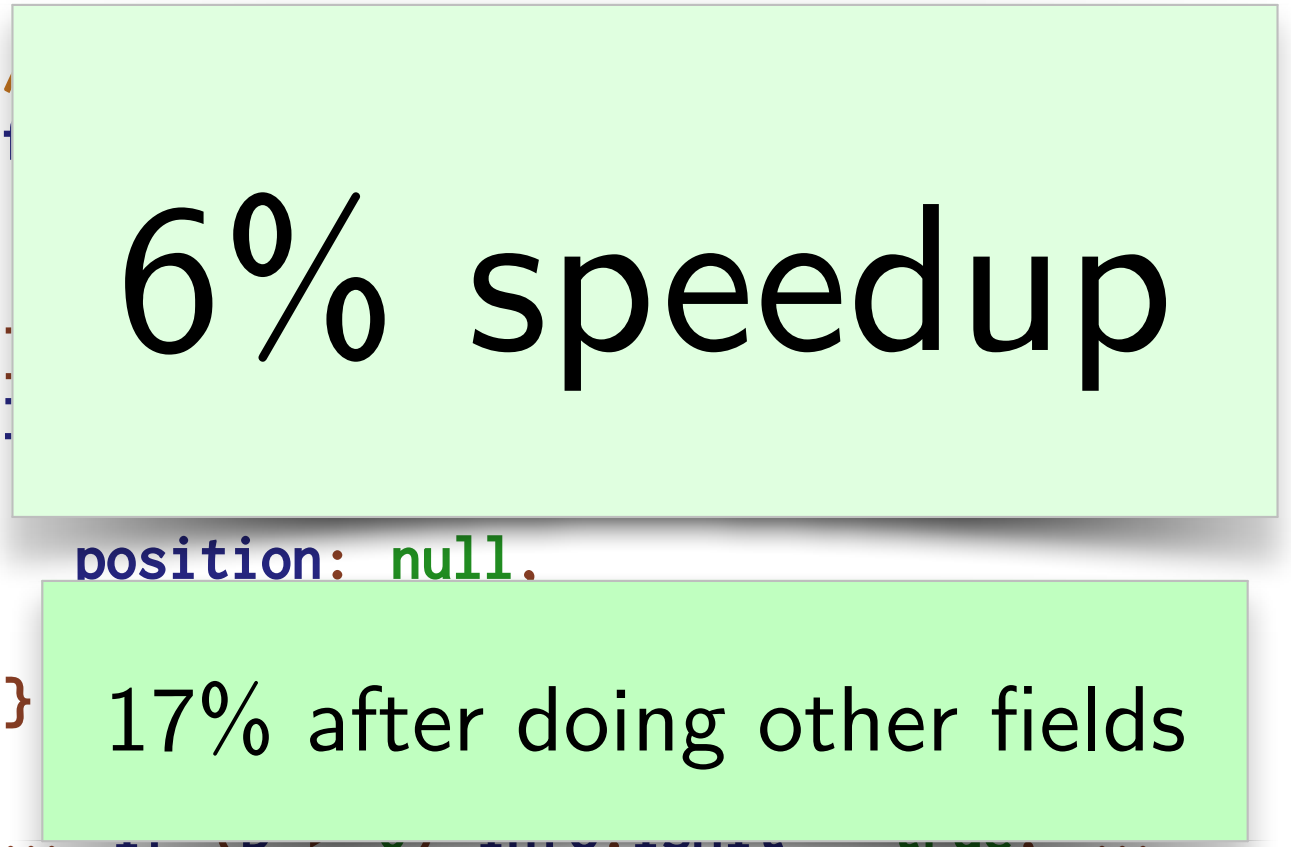
```
// constructor
function IntersectionInfo () {
   this.color = "black"
   this.isHit = false;
}
IntersectionInfo.prototype = {
   isHit: false,
   position: null,
   ...
}

...  if (D > 0) info.isHit = true;  ...

...  if (info.isHit) return info.color;  ...
```

Inconsistent Property Location

6% speedup

```
    position: null,
    ...
}

...  if (D > 0) info.isHit = true;  ...

...  if (info.isHit) return info.color;  ...
```

Inconsistent Property Location



JIT vs AOT

- Compile / execute interleaved
- Code compiled multiple times
- Optimizations change over time

- **Profiler-driven instrumentation**
- **Temporal merging**
- **Profile-based ranking**

OO vs Functional

- Non-local failures
- Constructor near miss clusters

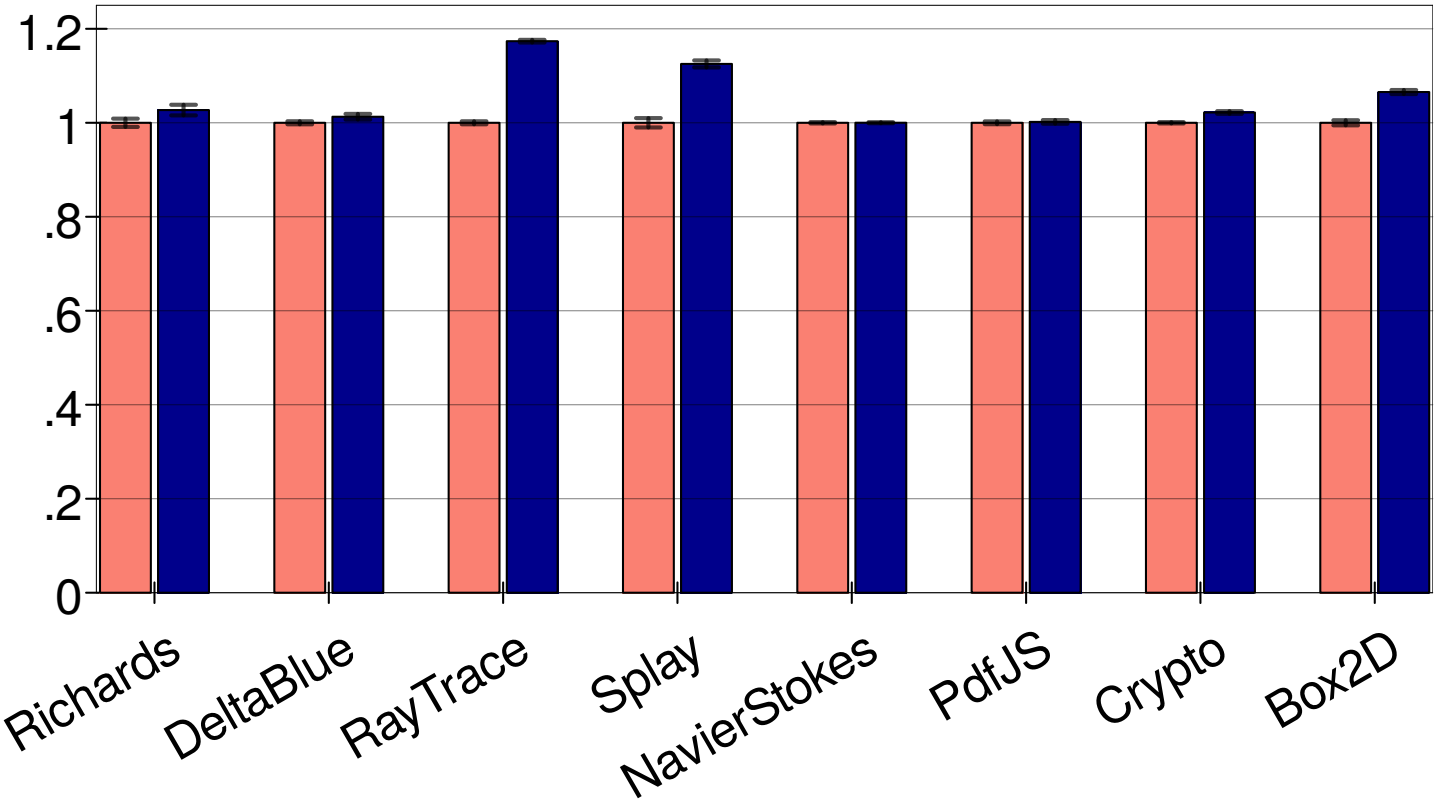
- **Solution-site inference**
- **By-solution merging**
- **By-constructor merging**

Hypothesis: Coaching improves performance

Baseline: Non-optimized

Coached: Followed recommendations (Minutes of work)

Octane score, higher is better

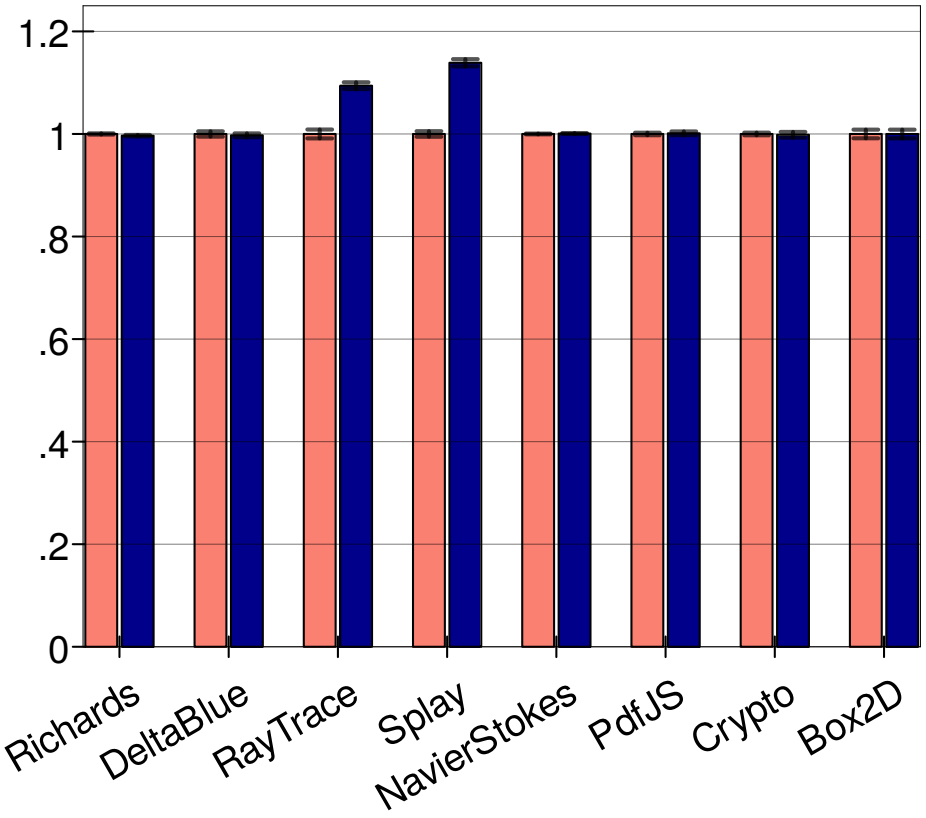


Hypothesis: Coaching improves performance

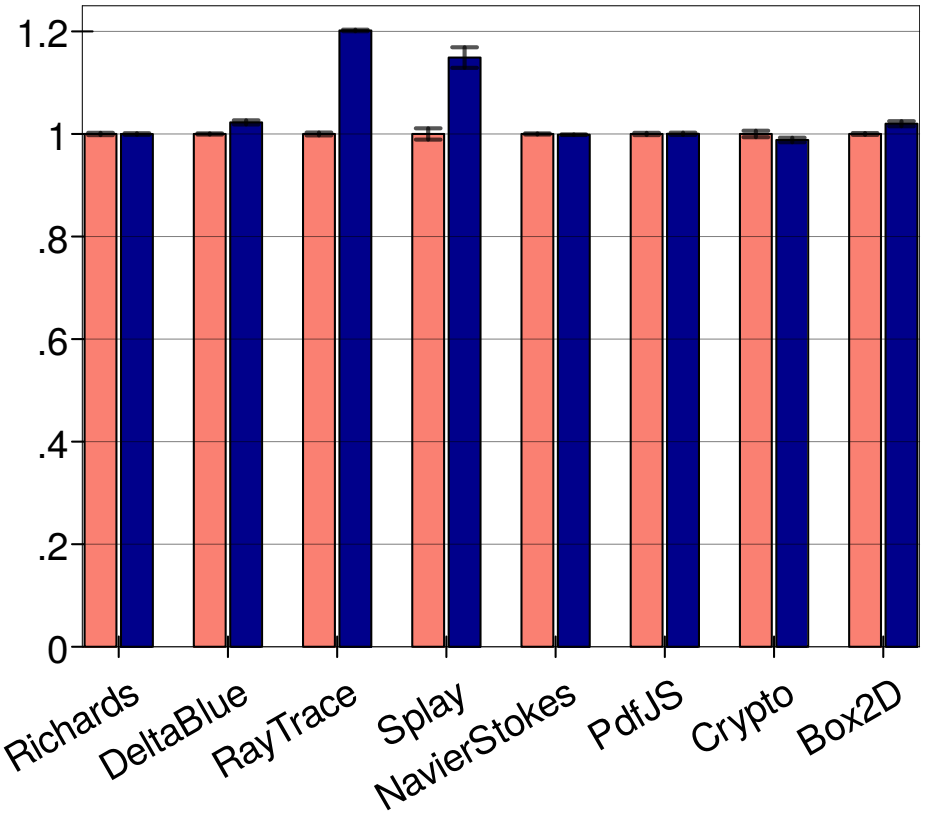
Baseline: Non-optimized

Coached: Followed recommendations (Minutes of work)

Octane score, higher is better



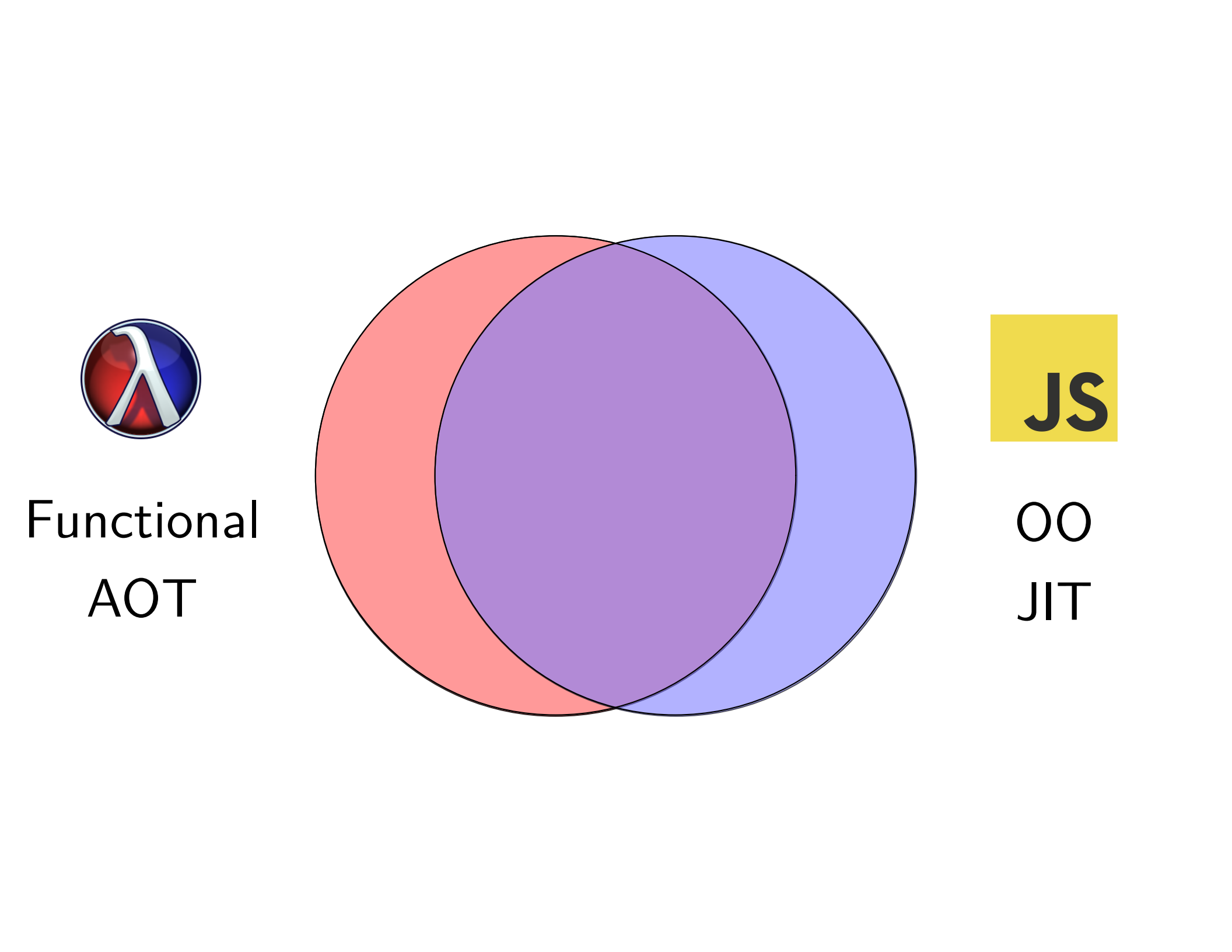
V8



JavaScriptCore

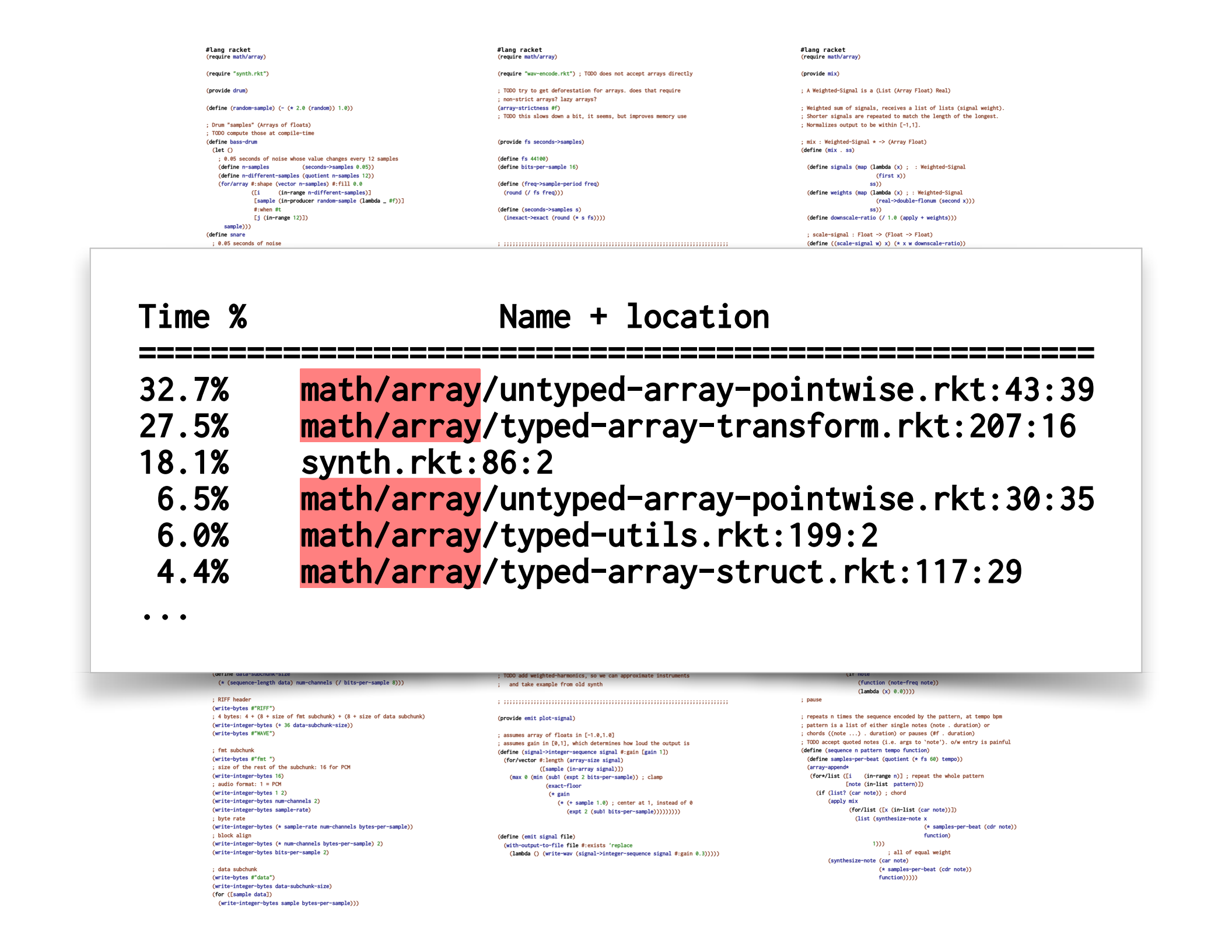
The SpiderMonkey coach today

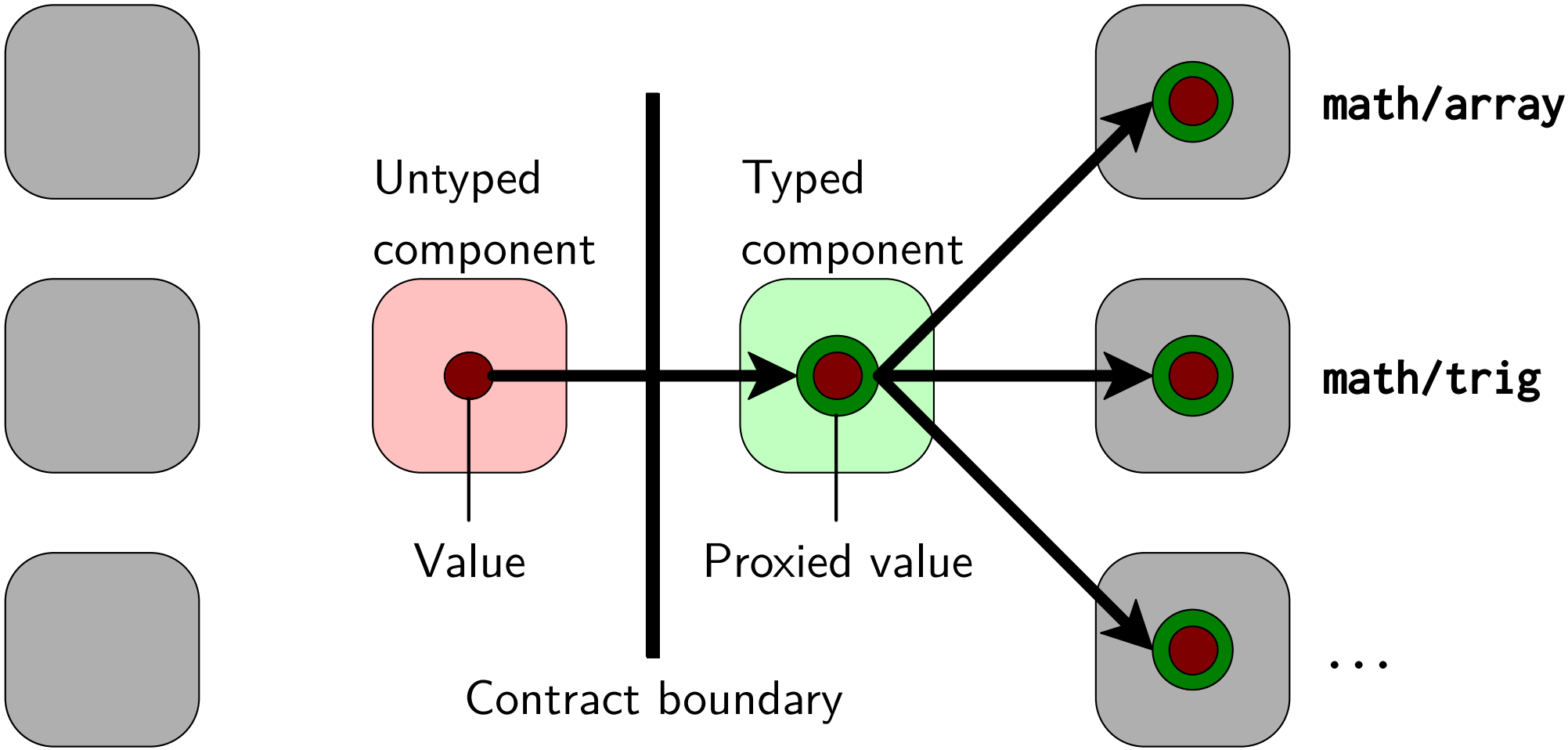
- **`github.com/stamourv/jit-coach/`**
- Command-line prototype
- Ongoing at Mozilla: Firefox integration

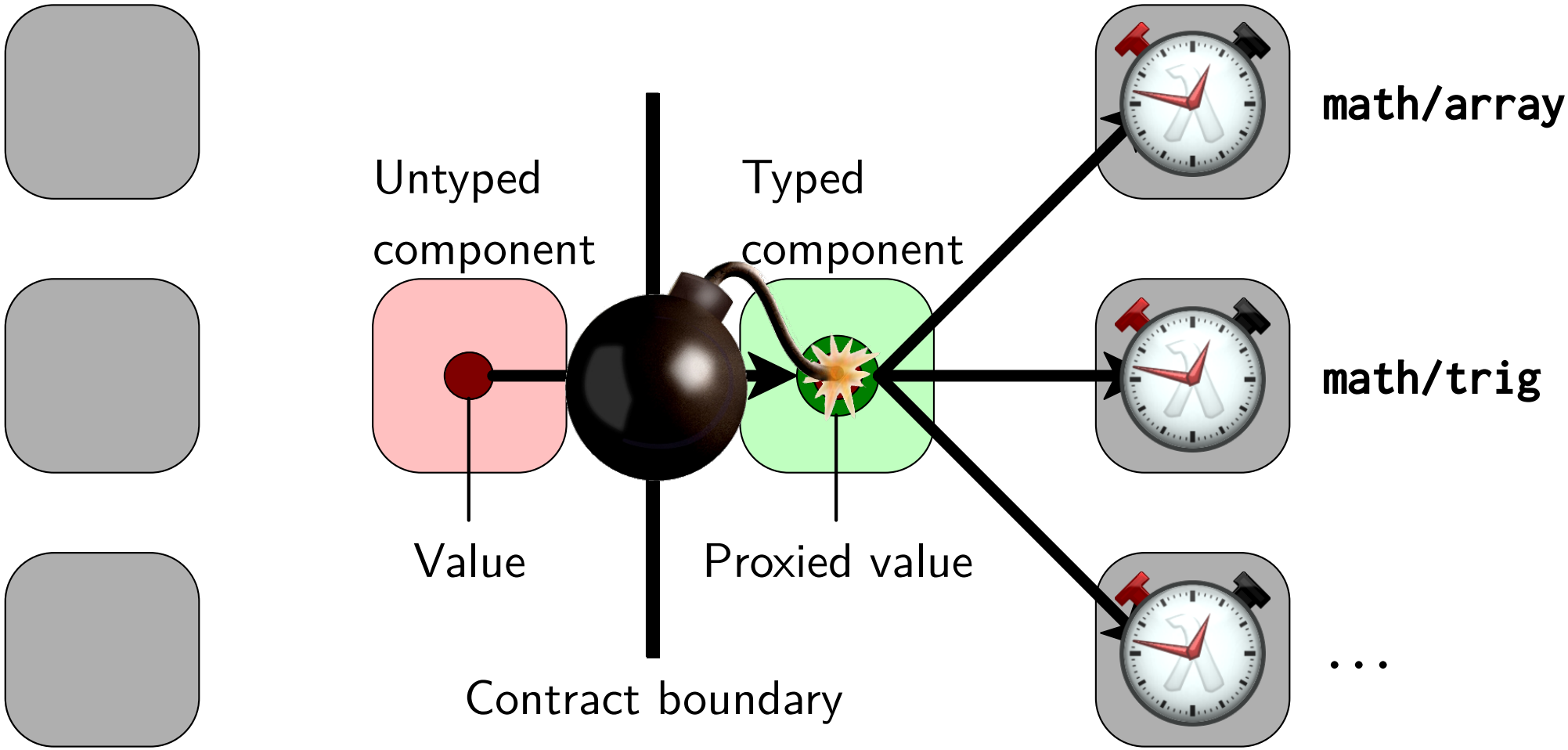

Functional
AOT
JS
OO
JIT

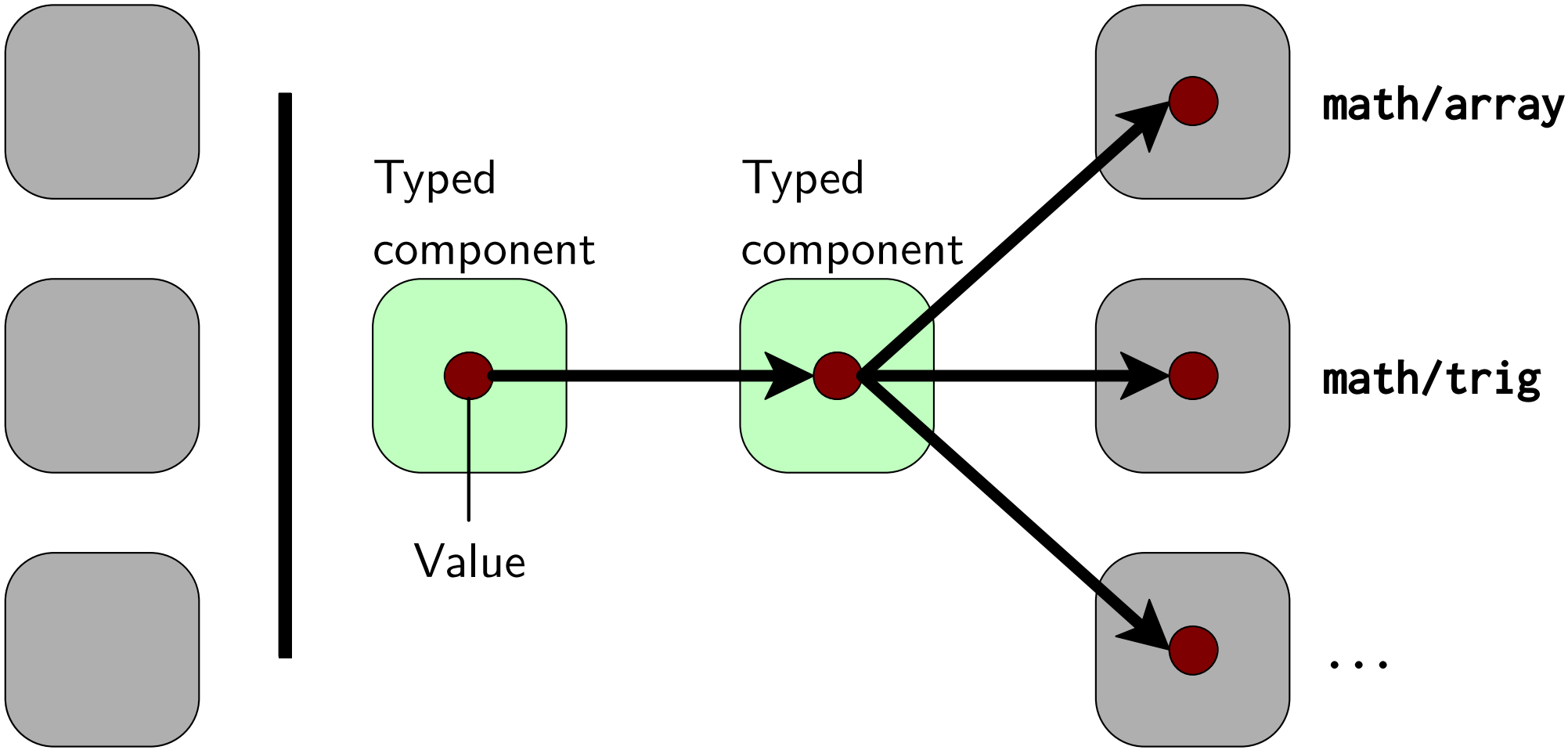
Feature-Specific Profiling

```
(emit
 (sequence
   sawtooth-wave #:bpm 380
   [(C 5) #f (C 5) #f (A# 4) #f (C 5) ...])
 "funky-town.wav")
```


#lang racket
(require math/array)
(require "synth.rkt")
(provide drum)
(define (random-sample) (- (* 2.0 (random)) 1.0))
; Drum "samples" (Arrays of floats)
; TODO compute those at compile-time
(define bass-drum
(define snare
#lang racket
(require math/array)
(require "wav-encode.rkt") ; TODO does not accept arrays directly
; TODO try to get deforestation for arrays. does that require
; non-strict arrays? lazy arrays?
(array-strictness #f)
; TODO this slows down a bit, it seems, but improves memory use
(provide fs seconds->samples)
(define fs 44100)
(define bits-per-sample 16)
(define (freq->sample-period freq)
(round (/ fs freq)))
(define (seconds->samples s)
(inexact->exact (round (* s fs))))
#lang racket
(require math/array)
(provide mix)
; A Weighted-Signal is a (List (Array Float) Real)
; Weighted sum of signals, receives a list of lists (signal weight).
; Shorter signals are repeated to match the length of the longest.
; Normalizes output to be within [-1,1].
; mix : Weighted-Signal * -> (Array Float)
(define (mix . ss)
(define signals (map (lambda (x) ; : Weighted-Signal
(first x))
ss))
(define weights (map (lambda (x) ; : Weighted-Signal
(real->double-flonum (second x)))
ss))
(define downscale-ratio (/ 1.0 (apply + weights)))
; scale-signal : Float -> (Float -> Float)
(define ((scale-signal w) x) (* x w downscale-ratio))
Time %
Name + location
32.7%
math/array/untyped-array-pointwise.rkt:43:39
27.5%
math/array/typed-array-transform.rkt:207:16
18.1%
synth.rkt:86:2
6.5%
math/array/untyped-array-pointwise.rkt:30:35
6.0%
math/array/typed-utils.rkt:199:2
4.4%
math/array/typed-array-struct.rkt:117:29
...
(* (sequence-length data) num-channels (/ bits-per-sample 8)))
; RIFF header
(write-bytes #"RIFF")
; 4 bytes: 4 + (8 + size of fmt subchunk) + (8 + size of data subchunk)
(write-integer-bytes (+ 36 data-subchunk-size))
(write-bytes #"WAVE")
; fmt subchunk
(write-bytes #"fmt ")
; size of the rest of the subchunk: 16 for PCM
(write-integer-bytes 16)
; audio format: 1 = PCM
(write-integer-bytes 1 2)
(write-integer-bytes num-channels 2)
(write-integer-bytes sample-rate)
; byte rate
(write-integer-bytes (* sample-rate num-channels bytes-per-sample))
; block align
(write-integer-bytes (* num-channels bytes-per-sample) 2)
(write-integer-bytes bits-per-sample 2)
; data subchunk
(write-bytes #"data")
(write-integer-bytes data-subchunk-size)
(for ([sample data])
(write-integer-bytes sample bytes-per-sample)))
; TODO add weighted-harmonics, so we can approximate instruments
; and take example from old synth
(provide emit plot-signal)
; assumes array of floats in [-1.0,1.0]
; assumes gain in [0,1], which determines how loud the output is
(define (signal->integer-sequence signal #:gain [gain 1])
(for/vector #:length (array-size signal)
([sample (in-array signal)])
(max 0 (min (sub1 (expt 2 bits-per-sample)) ; clamp
(exact-floor
(* gain
(* (+ sample 1.0) ; center at 1, instead of 0
(expt 2 (sub1 bits-per-sample)))))))))
(define (emit signal file)
(with-output-to-file file #:exists 'replace
(lambda () (write-wav (signal->integer-sequence signal #:gain 0.3)))))
(function (note-freq note))
(lambda (x) 0.0))))
; pause
; repeats n times the sequence encoded by the pattern, at tempo bpm
; pattern is a list of either single notes (note . duration) or
; chords ((note ...) . duration) or pauses (#f . duration)
; TODO accept quoted notes (i.e. args to 'note'). o/w entry is painful
(define (sequence n pattern tempo function)
(define samples-per-beat (quotient (* fs 60) tempo))
(array-append*
(for*/list ([i (in-range n)] ; repeat the whole pattern
[note (in-list pattern)])
(if (list? (car note)) ; chord
(apply mix
(for/list ([x (in-list (car note))])
(list (synthesize-note x
(* samples-per-beat (cdr note))
function)
1)))
; all of equal weight
(synthesize-note (car note)
(* samples-per-beat (cdr note))
function)))))


Untyped
component
Typed
component
math/array
math/trig
...
Value
Proxied value
Contract boundary


Untyped
component
Typed
component
Value
Proxied value
Contract boundary
math/array
math/trig
...


Typed
component
Typed
component
Value
math/array
math/trig
...

```
Contracts account for 73.77% of running time
     (17568 / 23816 ms)

  6210 ms : Array-unsafe-proc
            (-> Array (-> (vectorof Int) any))
  3110 ms : array-append*
            (->* ((listof Array)) (Int) Array)
  2776 ms : unsafe-build-array
            (-> (vectorof Int) [...] Array)
  ...


Generic sequences account for 0.04% of running time
     (10 / 23816 ms)

  10 ms : wav-encode.rkt:51:16
```

Reporting costs per feature instance

```
<linguistic feature> : <total cost>
    <cost> : <instance>
    <cost> : <instance>
    ...
```

E.g.

Output	Generic sequences
Casts	Security checks
Marketplace processes	Contracts
Pattern matching	Method dispatch
Keyword arguments	Backtracking

<insert your new feature here>

Reporting costs per feature instance

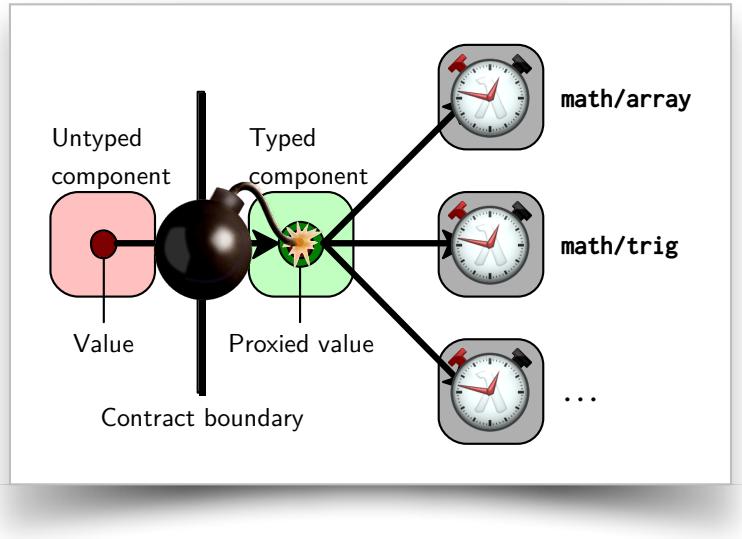
```
Pattern Matching : 1000ms
    600ms : sequencer.rkt:23
    200ms : drum.rkt:52
    ...
```

```
(define (sawtooth-wave ...)
  ...
  (match signal
    [<pattern>
     ... (harmonics ...)]
    ...))
```

Instance ~ Source location

Reporting costs per feature instance

```
Contracts : 2400ms
    1300ms : make-waveform
    500ms  : generate-chord
    ...
```



1 instance: Costs in N locations

Reporting costs per feature instance

```
Marketplace Processes : 1300ms
     800ms : (tcp-serve 53588)
     400ms : (tcp-serve 53587)
     ...
```

```
(define (tcp-serve ...)
  ...)

(spawn 53587
  (tcp-serve)
  ...)

(spawn 53588
  (tcp-serve)
  ...)
```

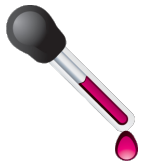
1 location: N instances

How does it work?

Architecture

Instrumentation inside libraries/DSLs

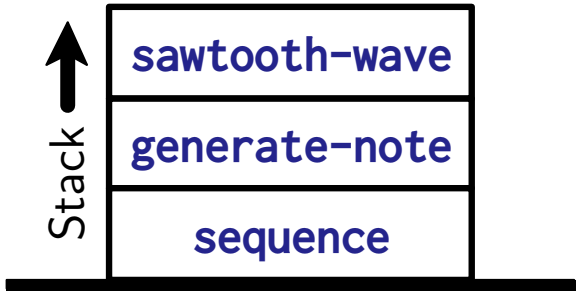
Sampling thread



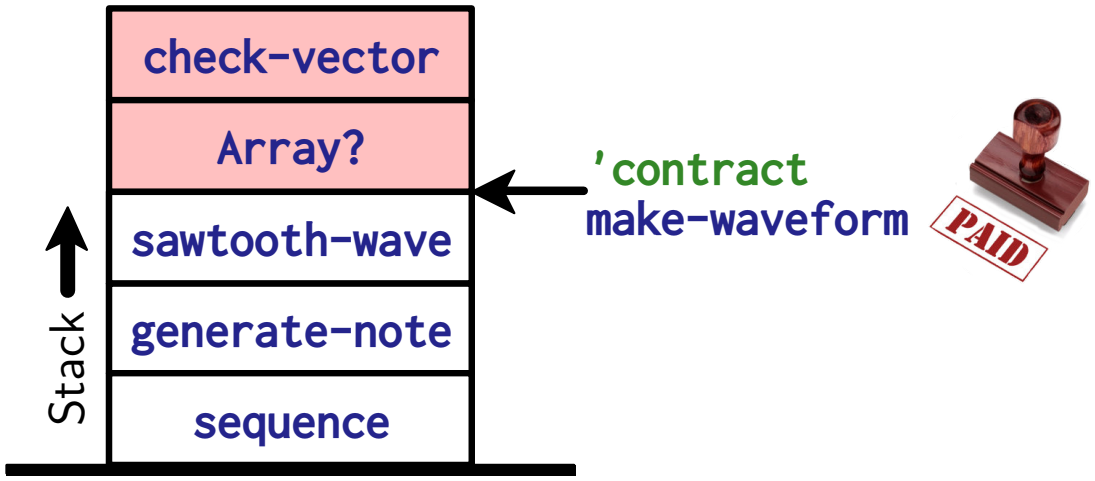
Protocol

Offline analysis

Observing Feature Code



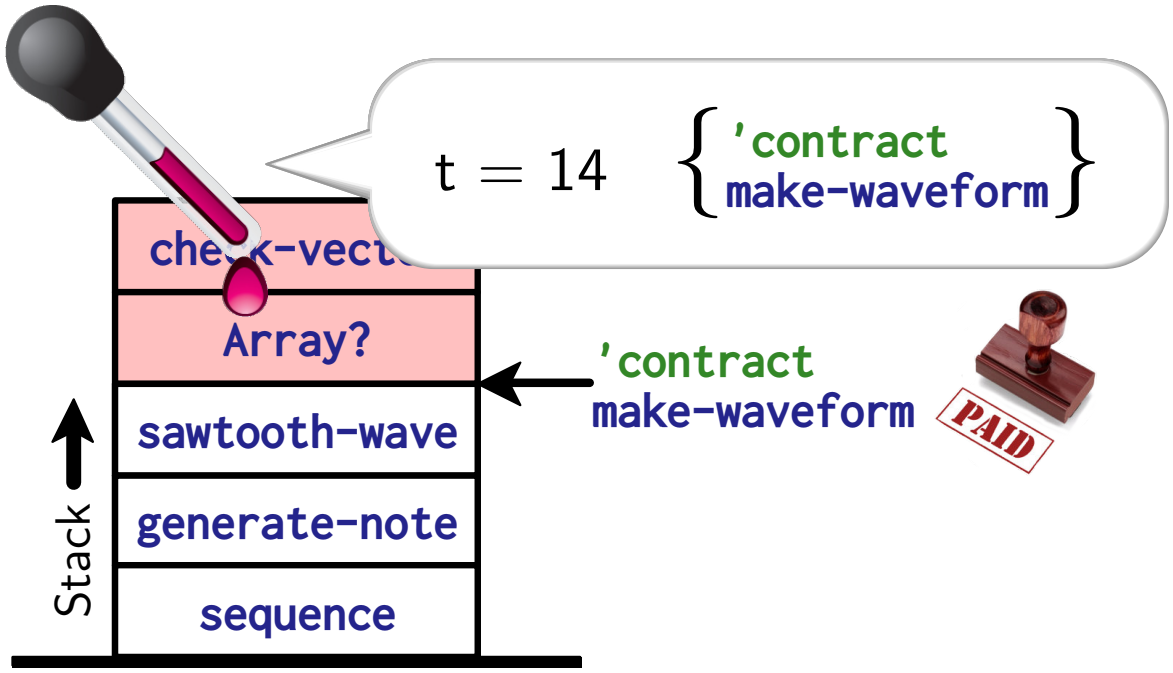
Observing Feature Code




```
(define (sawtooth-wave ...)
  (make-waveform ...)
  ...)
```

Mark present = Feature code is running

Observing Feature Code




```
(define (sawtooth-wave ...)
  (make-waveform ...)
  ...)
```

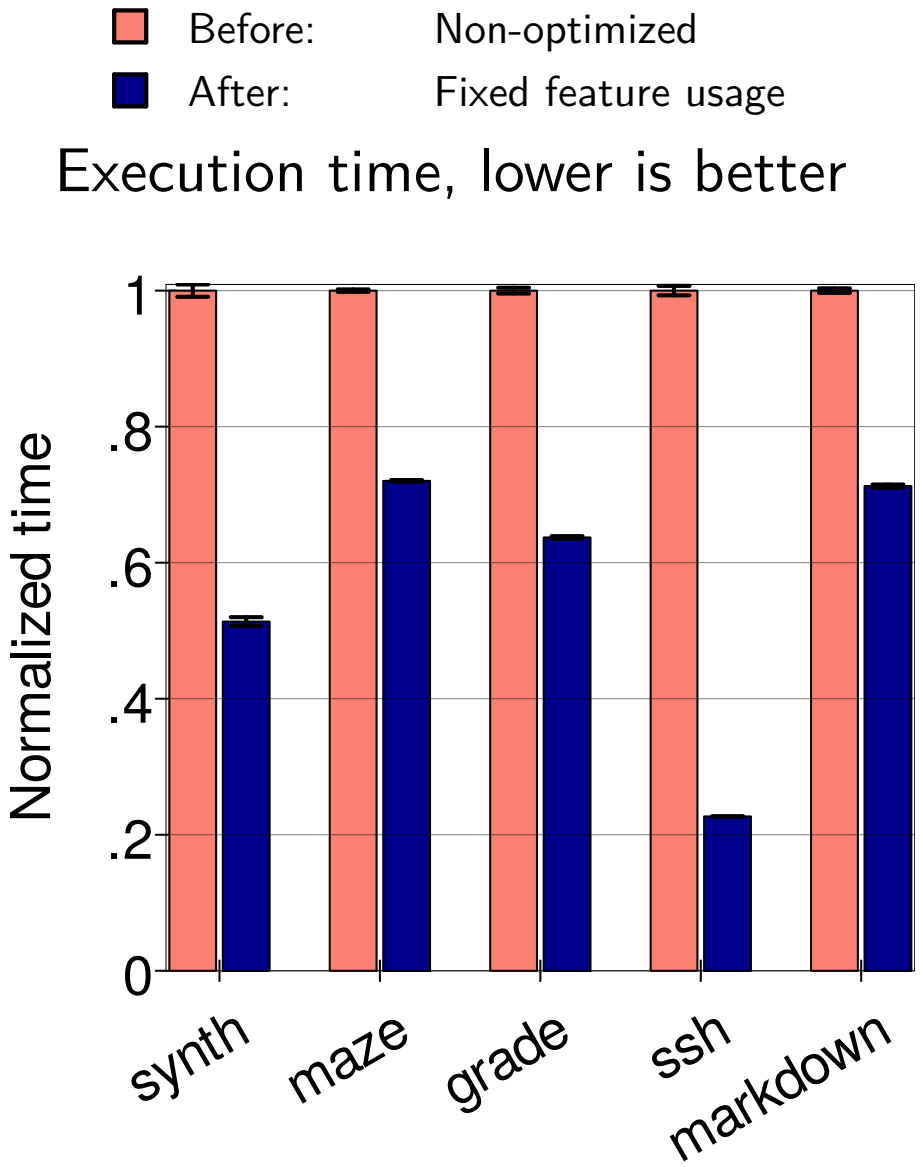
Mark present = Feature code is running

How well does it work?

Performance Impact

Experiment

- Take existing Racket programs
- Run the feature-specific profiler
- Fix uses of features mentioned in the report
- Measure performance impact (running time)



Instrumentation Effort

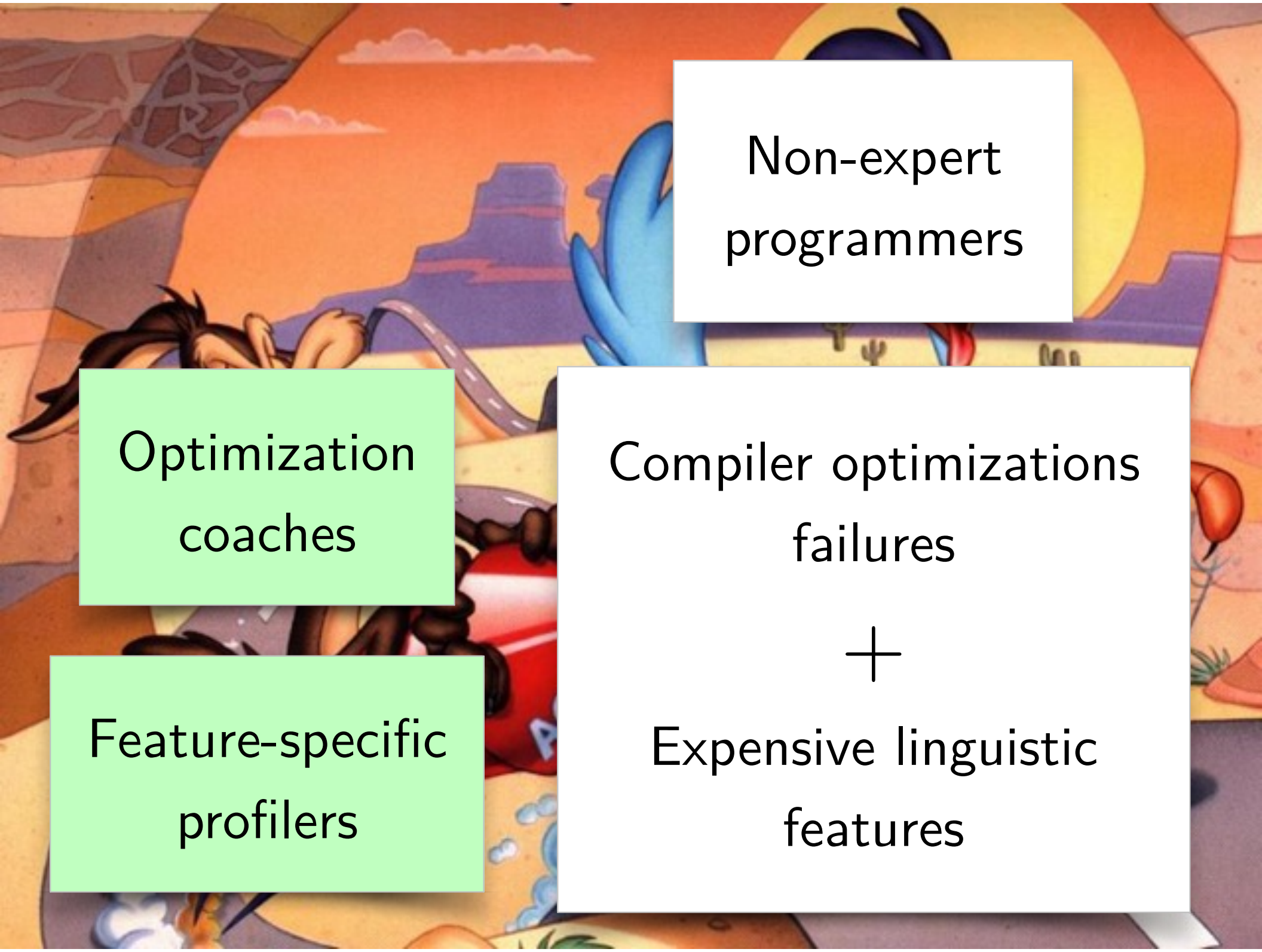
Feature	**LOC**
Contracts	183
Output	11
Generic sequences	18
Casts and assertions	37
Parser backtracking	18
Security policies	23
Marketplace processes	7
Pattern matching	18
Method dispatch	12
Keyword arguments	50

Reasonable for library creators

35 minutes for creator!
(+ 40 for extra analysis)

Wrapping up


ACME


Non-expert programmers
Optimization coaches
Feature-specific profilers
Compiler optimizations failures
+
Expensive linguistic features


MERRIE MELODIES
REG U.S. PAT. OFF.
"That's all Folks!"
A WARNER BROS. CARTOON
A VITAPHONE® RELEASE