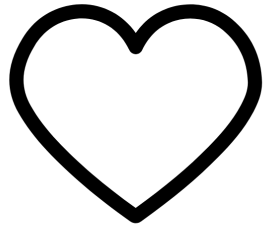
Does Blame Shifting Work?

Lukas Lazarek,
Alexis King, Samanvitha Sundar,
Robby Findler, Christos Dimoulas

PLT @ Northwestern

Contracts and blame research

contracts  $\exists \vdash \Rightarrow \forall$

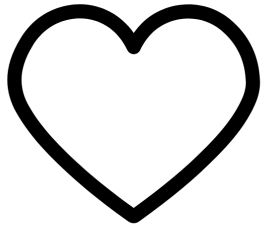
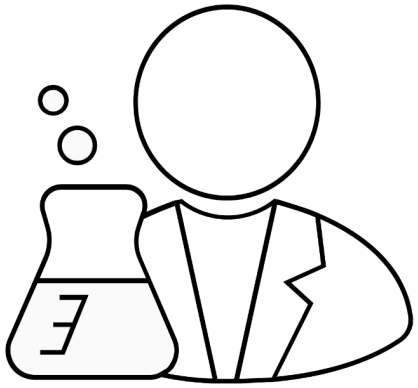
Blame is supposed to help debug

"Correct blame assignment **gets programmers started** with the **debugging process**"
[Dimoulas et al. POPL'11]

"After all, the **purpose of blame** assignment is to **assist programmers with debugging**"
[Strickland & Felleisen IFL'09]

Blame "narrow[s] down the **search for the error**"
[Dimoulas et al. OOPSLA'13]

Does blame actually work?

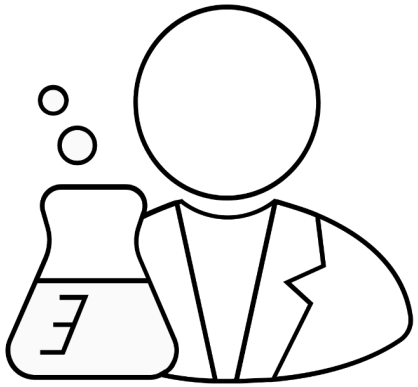
contracts   ?

The intuitive hypothesis

Blame has a systematic relationship

to the location of bugs in programs

Blame does work!



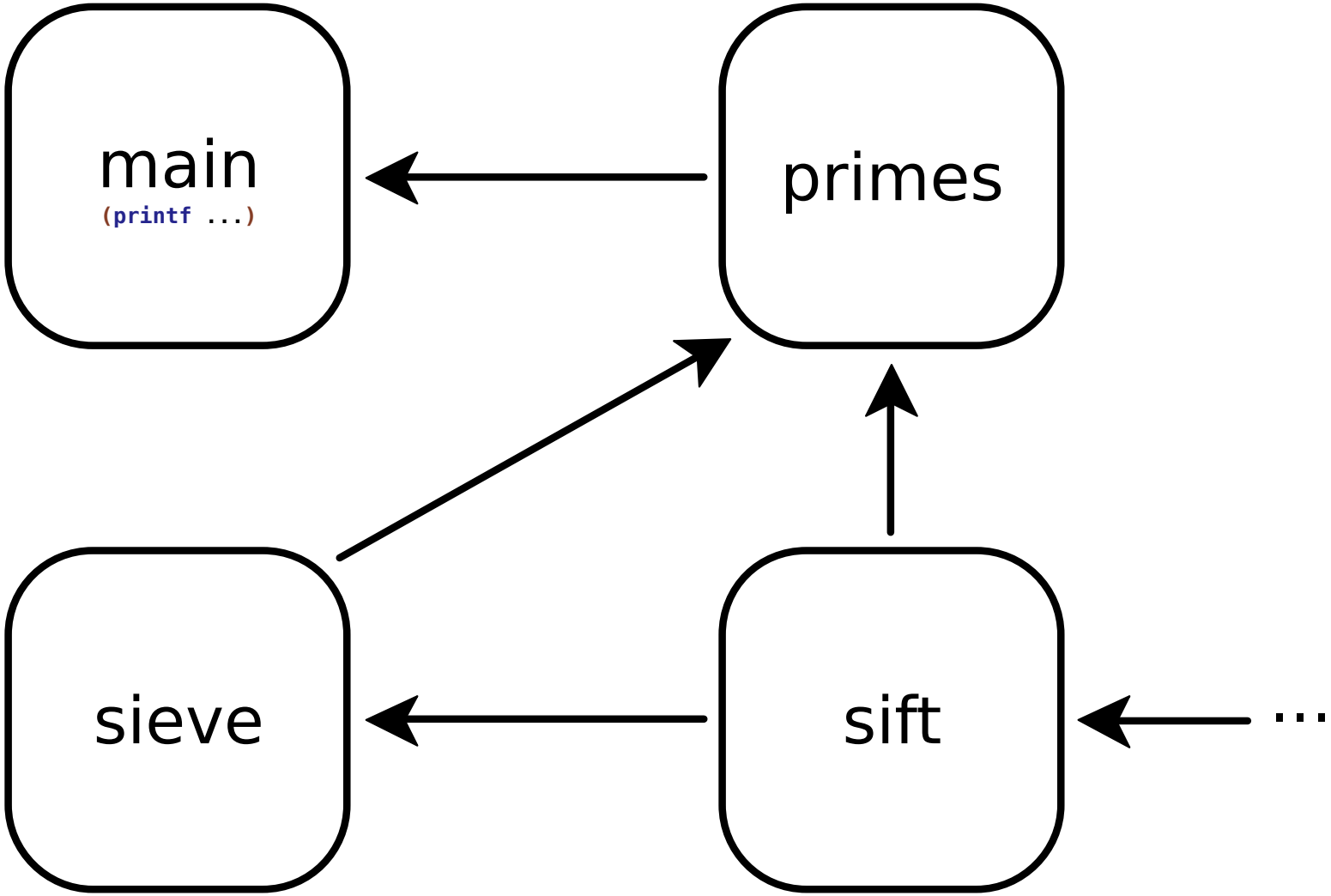
Let’s step back for a minute

Contracts

and

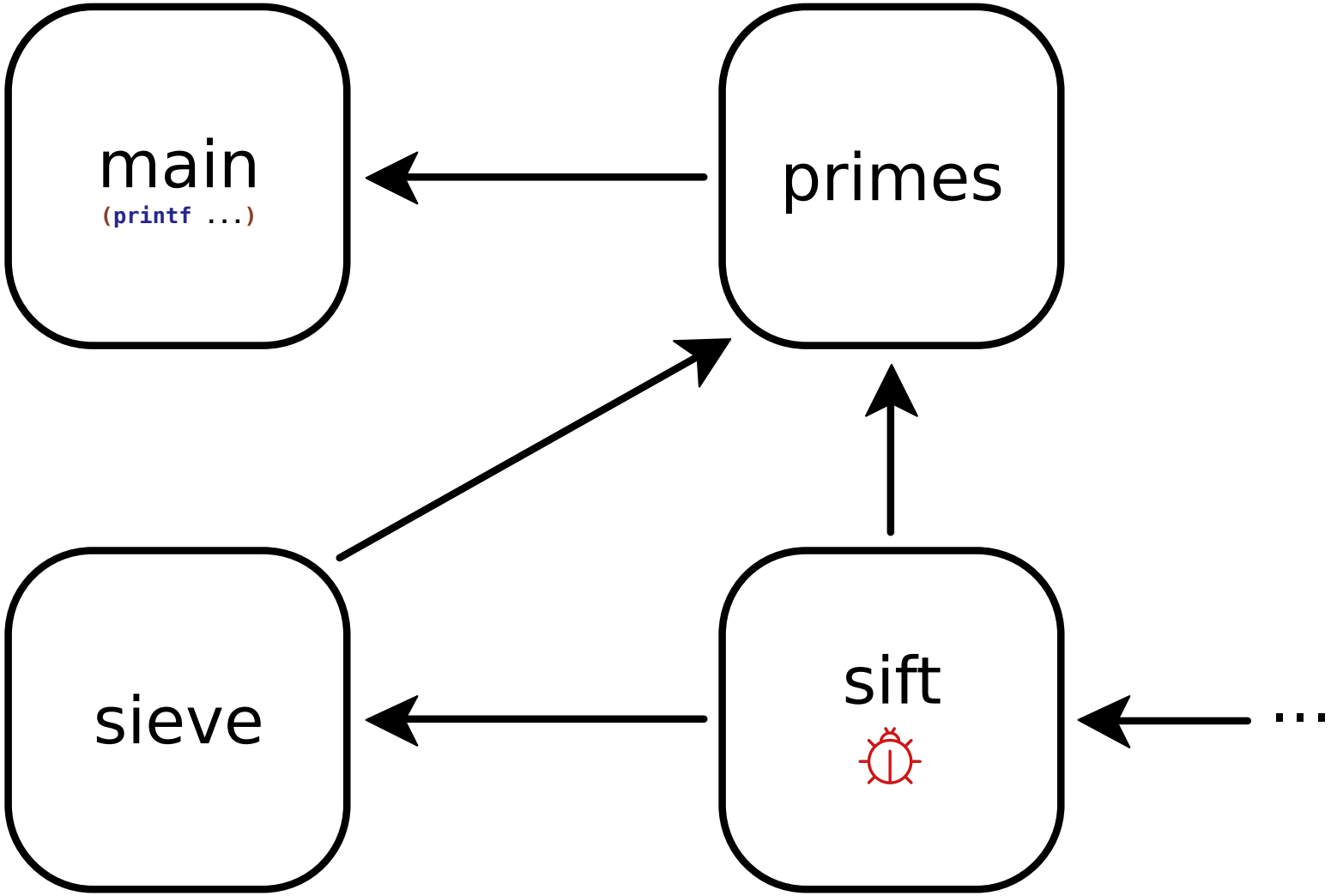
Blame

This is a program



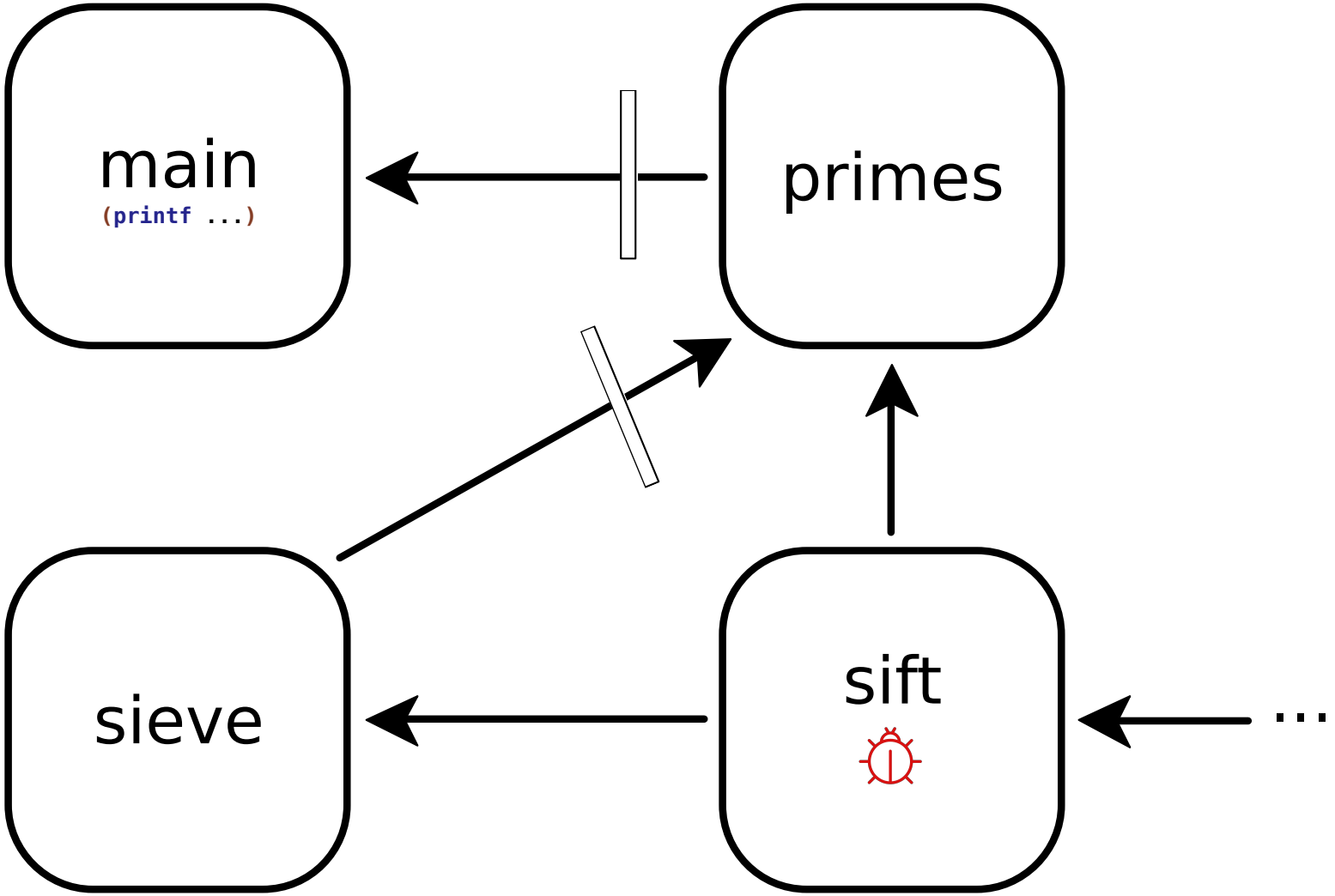
One component has a bug

2 3 4 5 ...



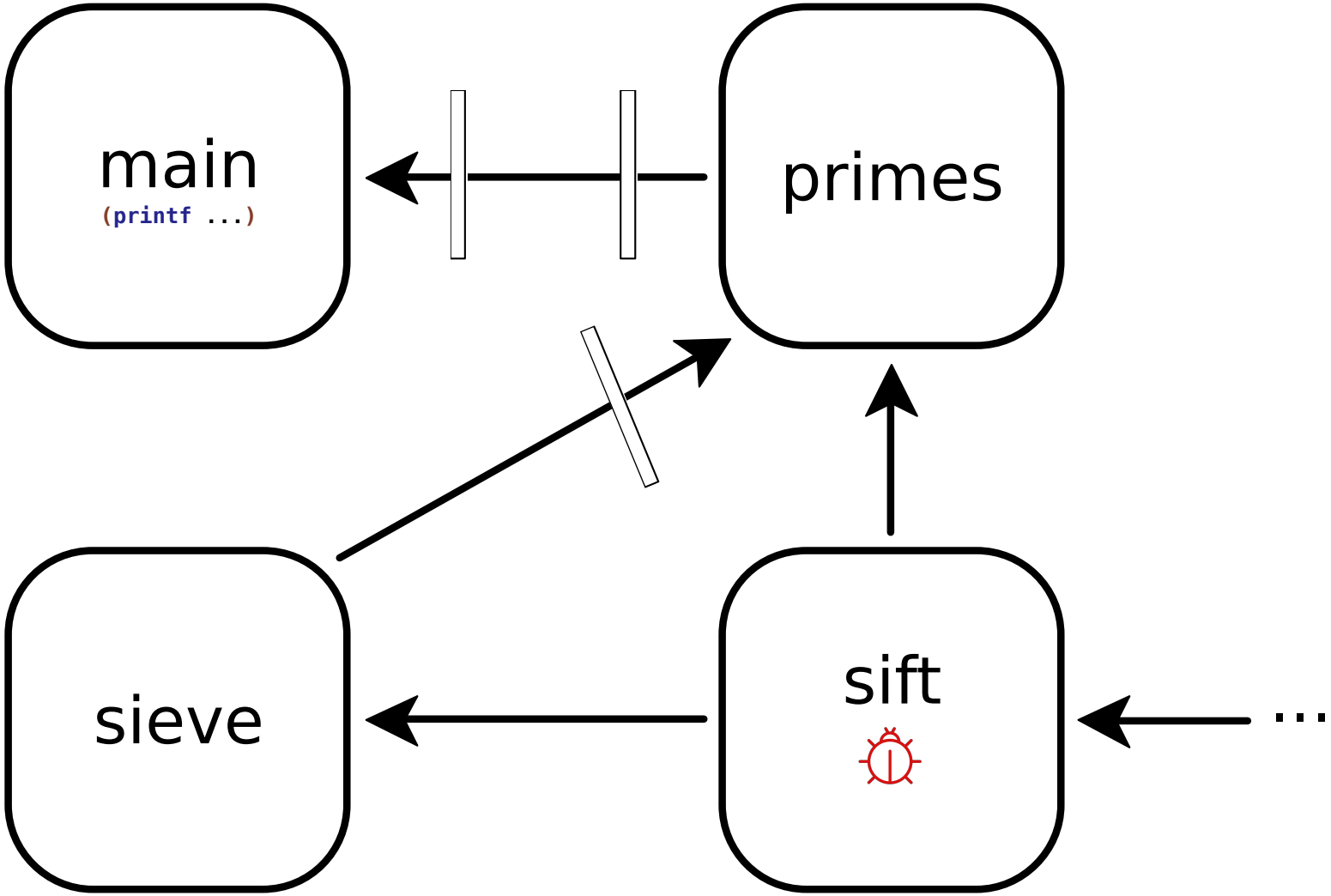
Contracts mediate component interaction

2 3 4 5 ...

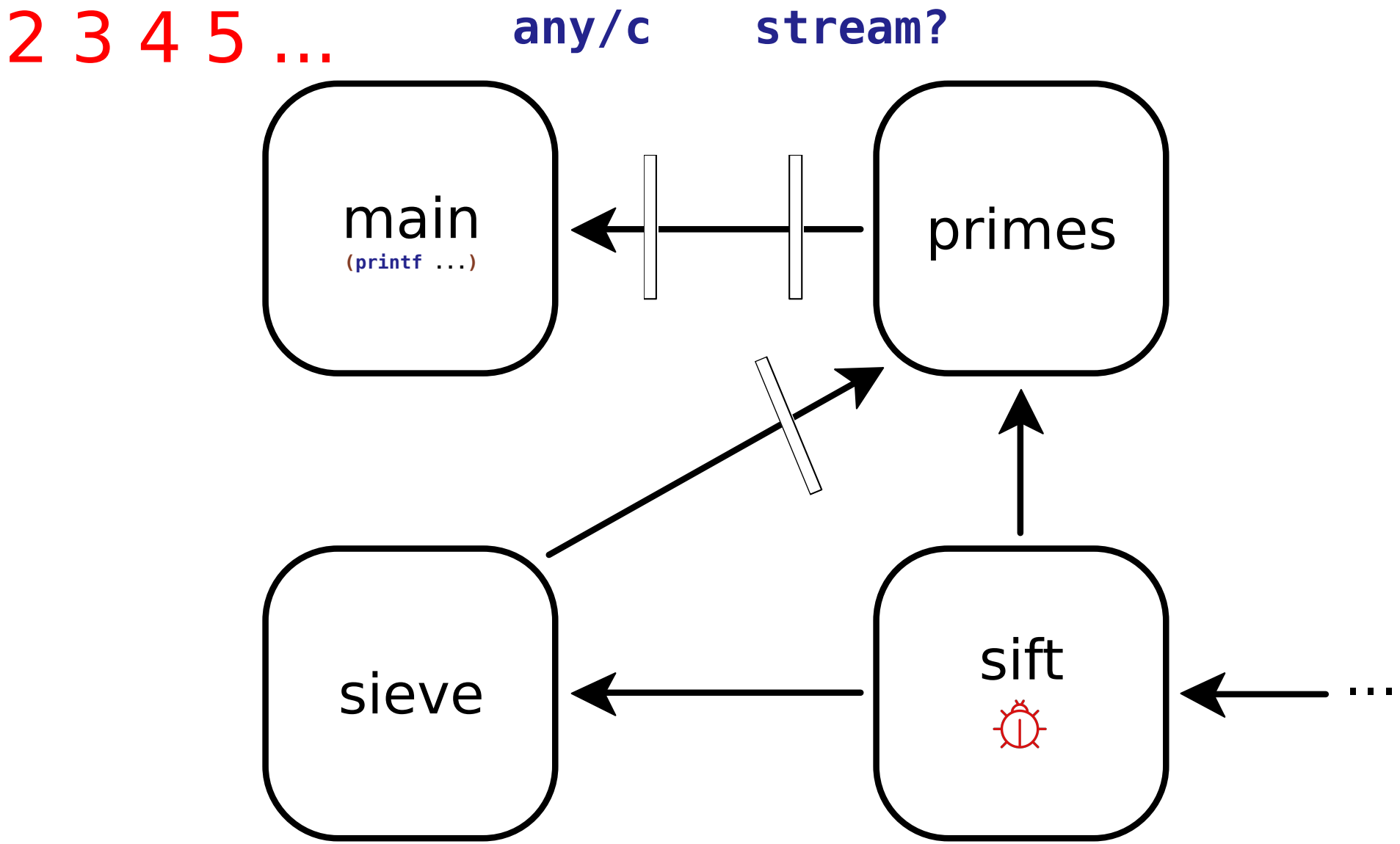


Each component can impose contracts

2 3 4 5 ...



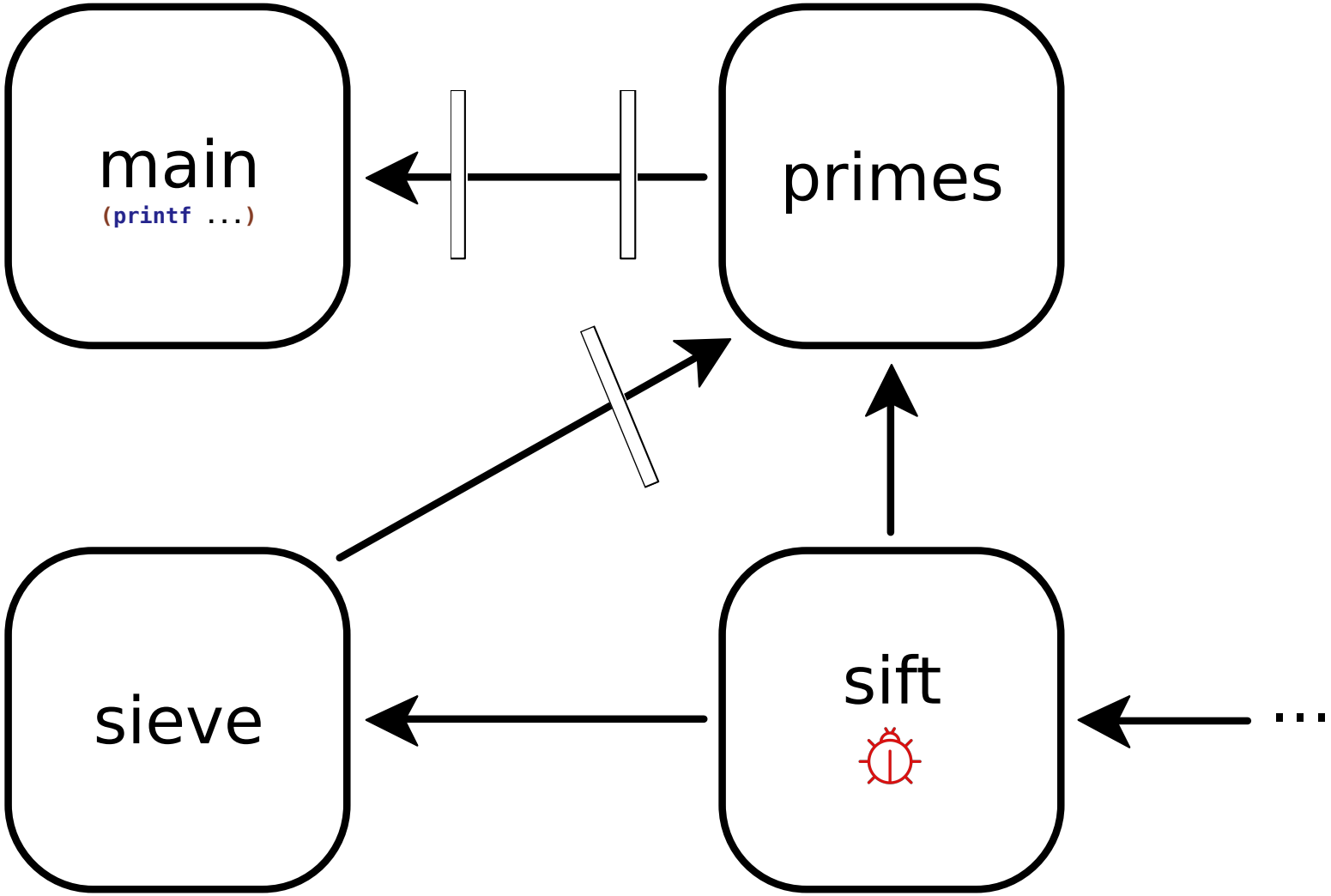
Each component can impose contracts



Contracts can have different strengths

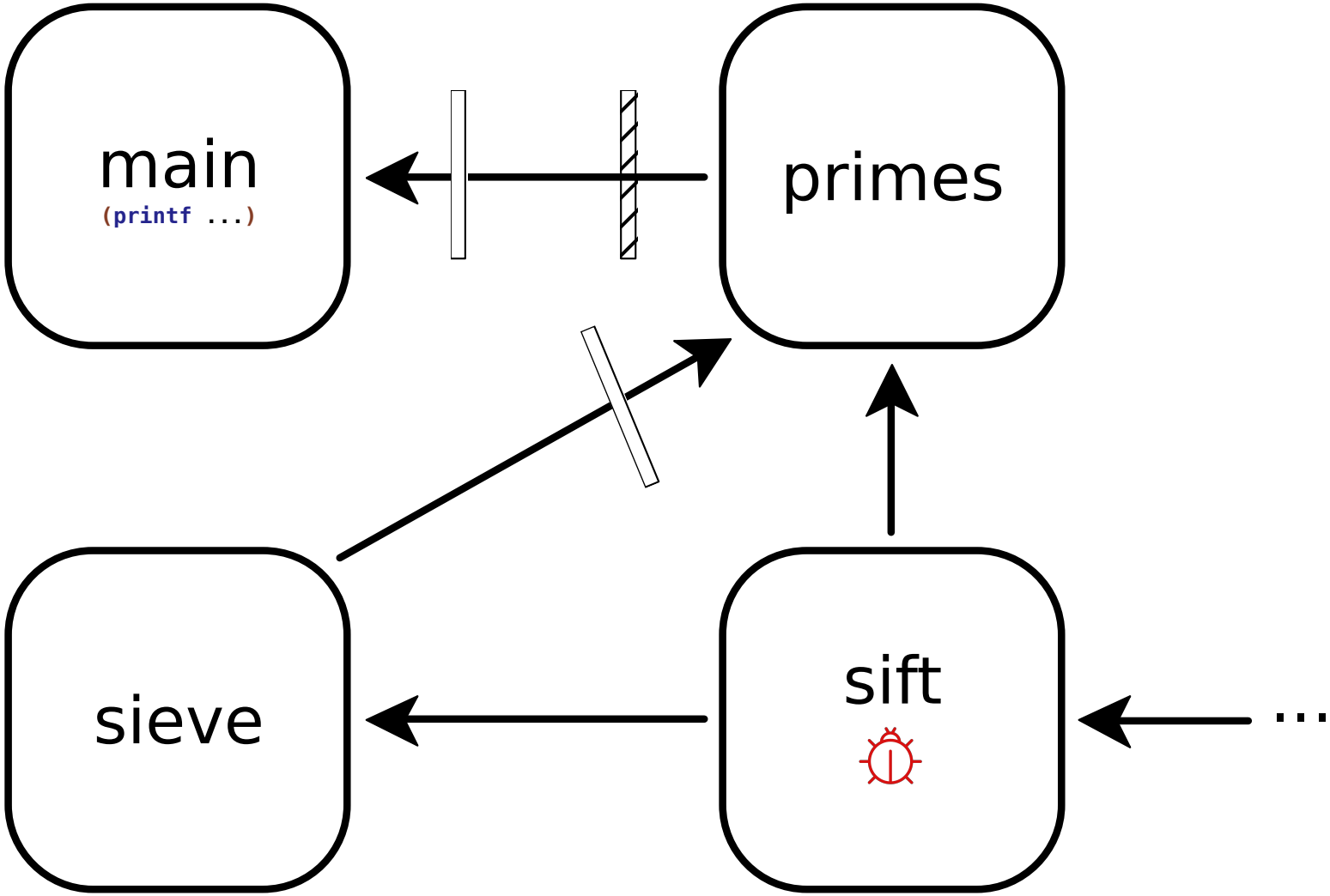
2 3 4 5 ...

stream?

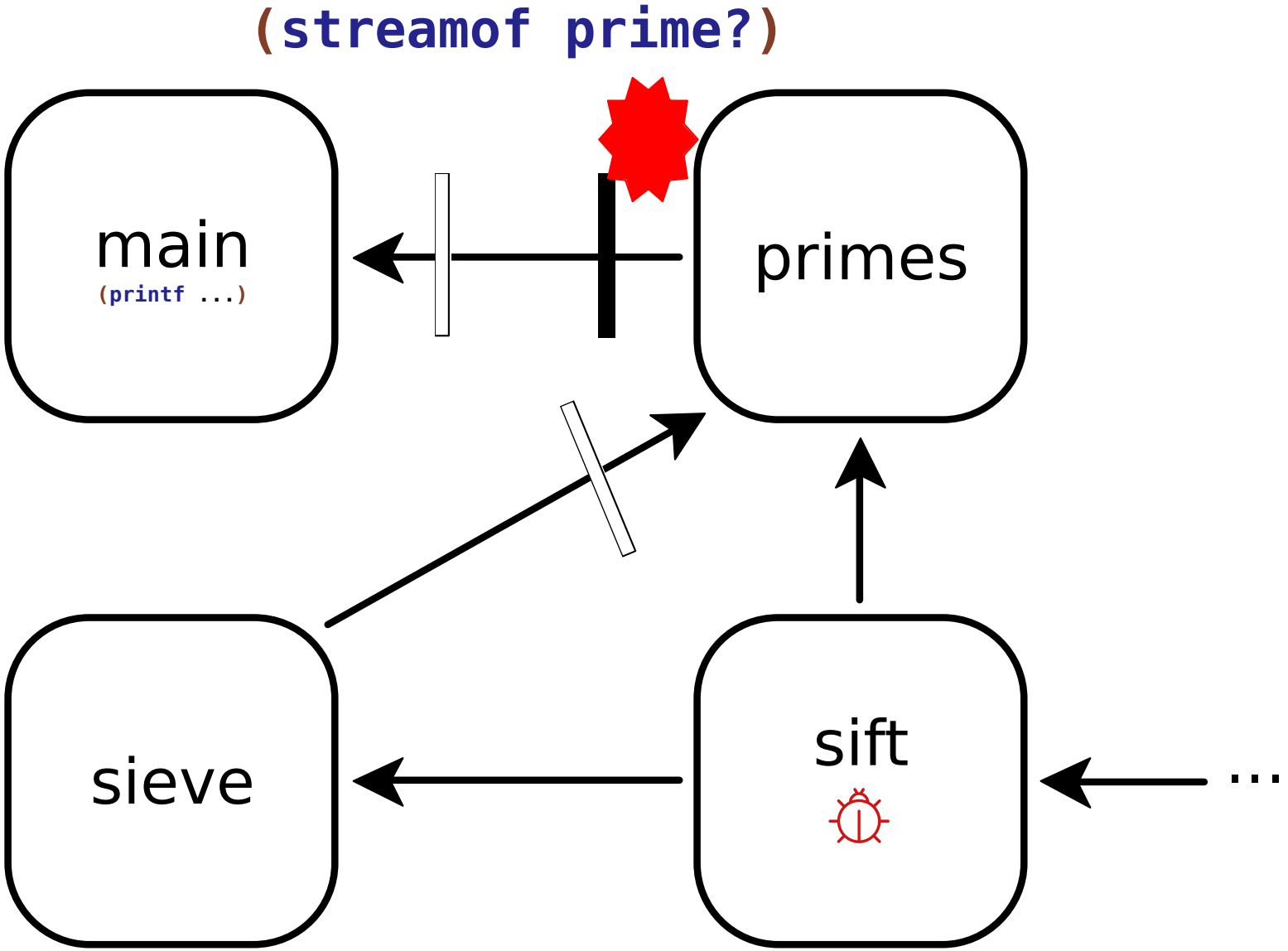


Contracts can have different strengths

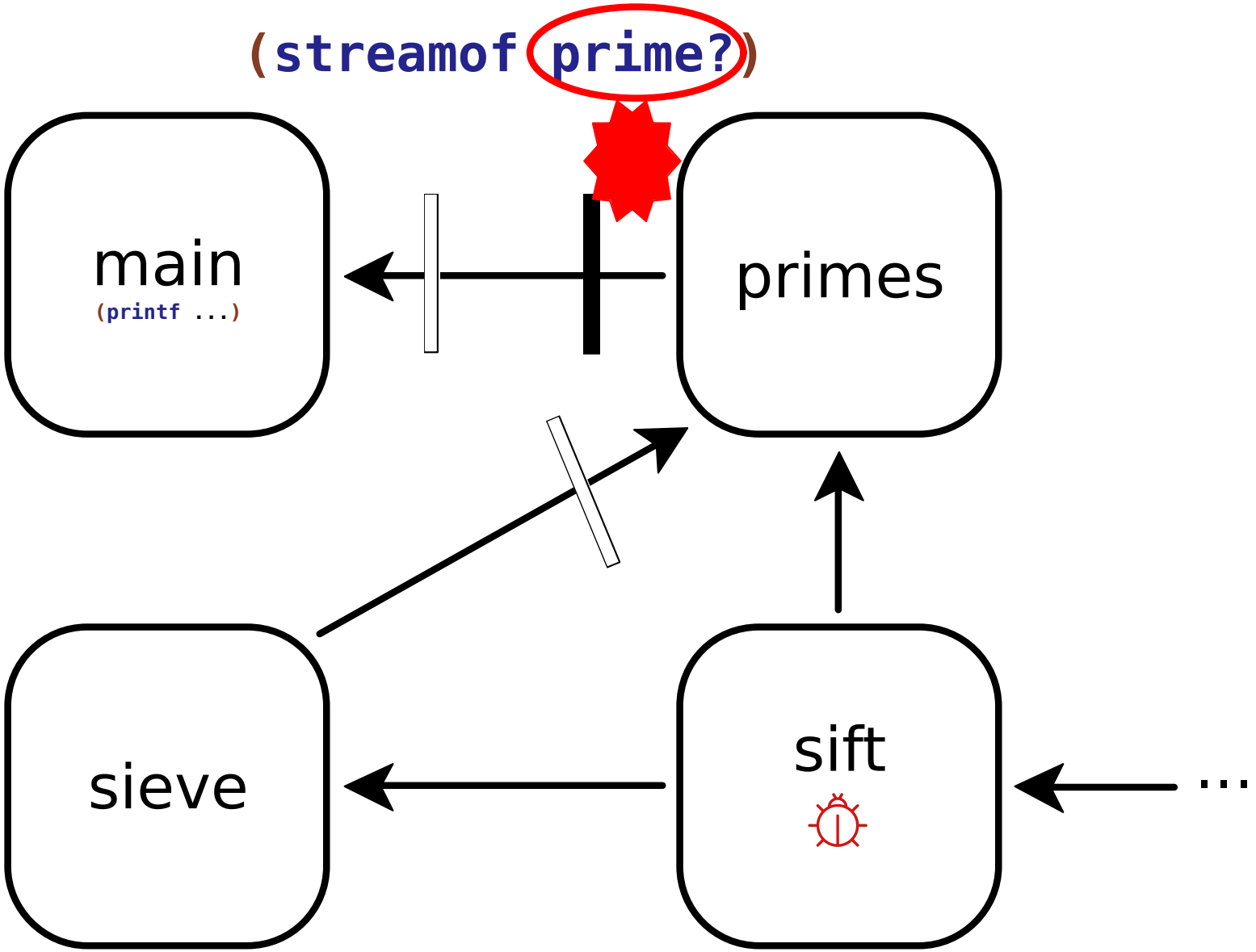
2 3 4 5 ... `(streamof positive-integer?)`



Contracts can have different strengths



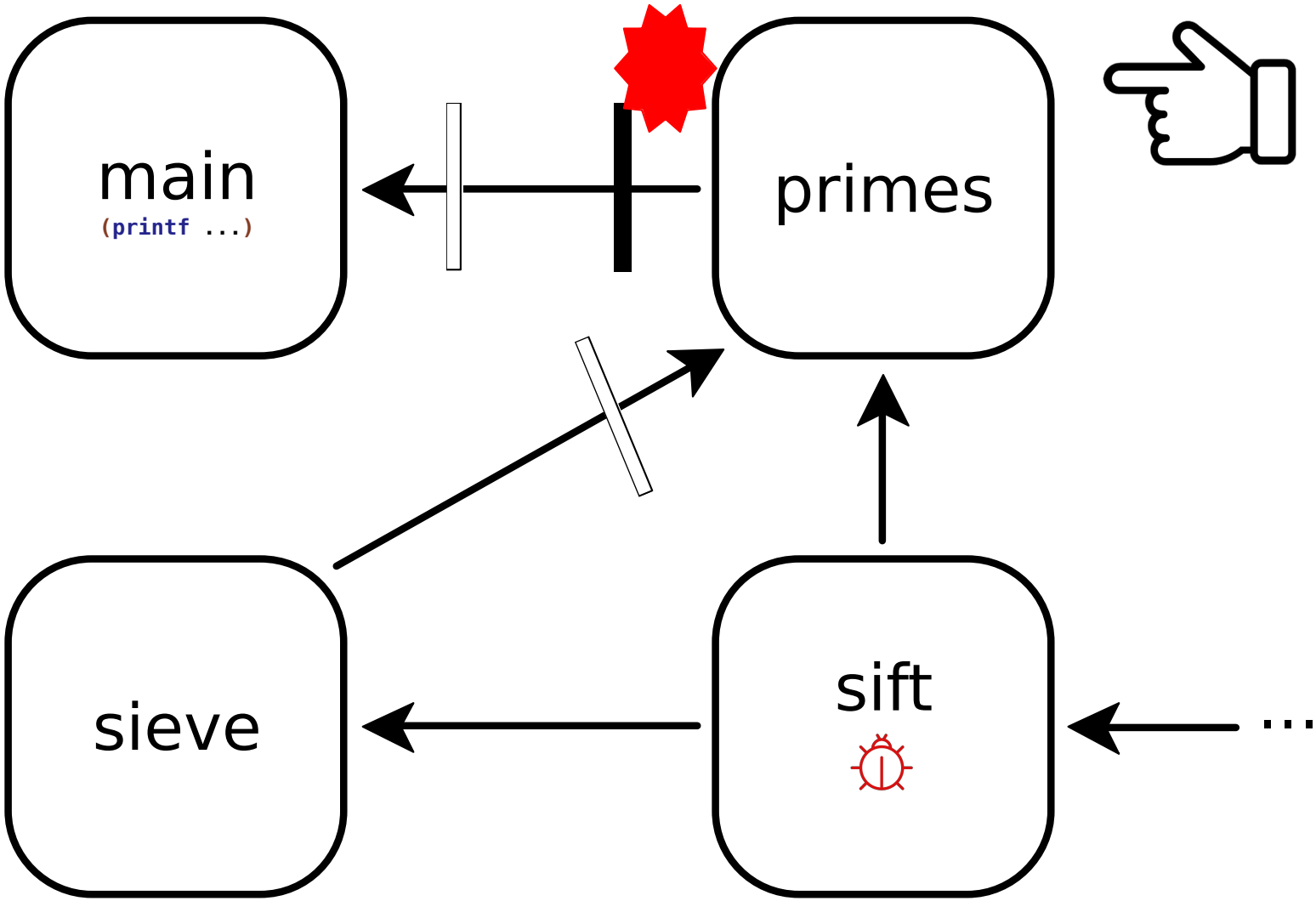
Contracts can have different strengths



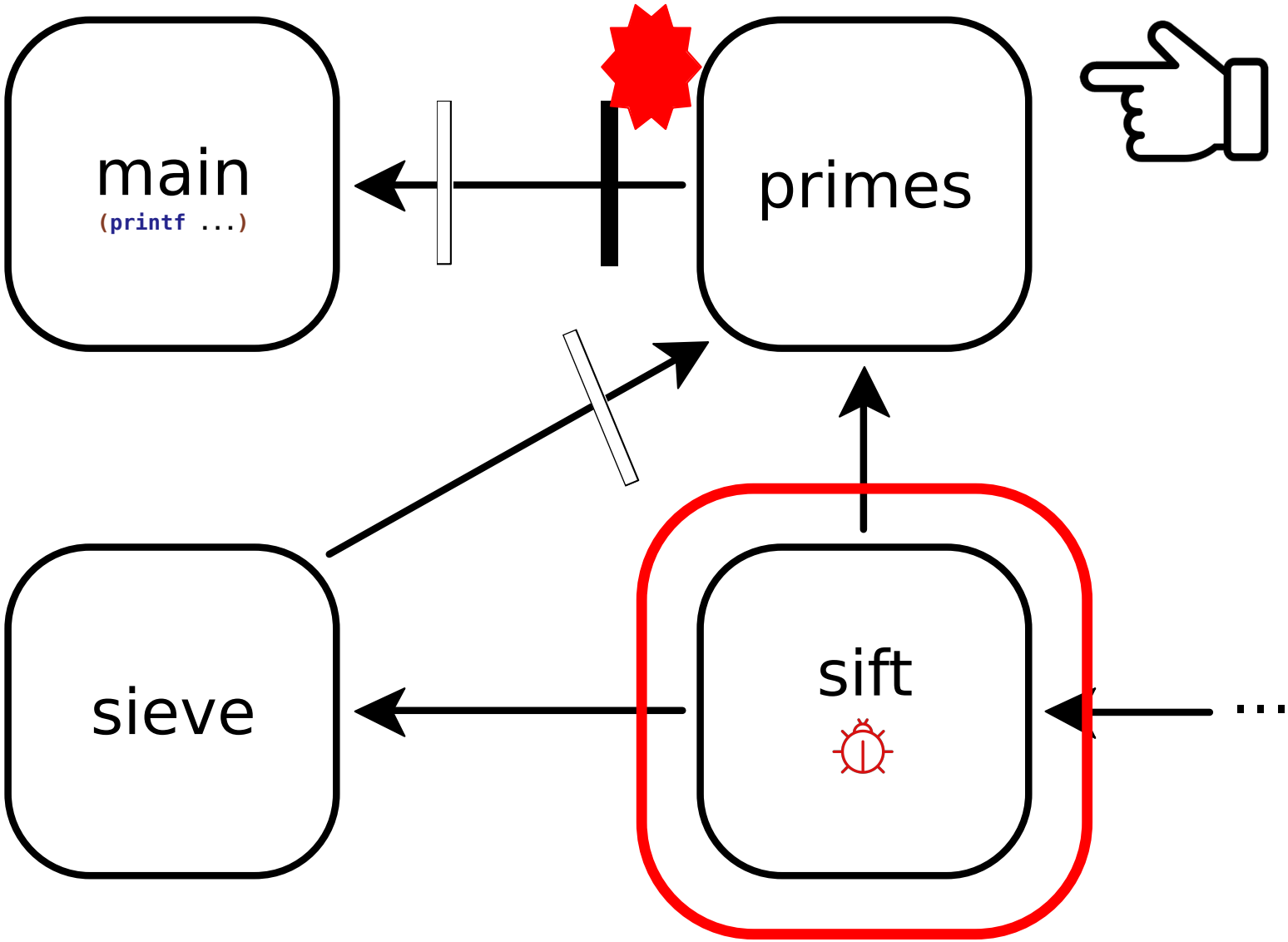
Violations blame a component

```
; primes: contract violation
;   promised: prime?
;   produced: 4
;   in: an element of
;       (streamof prime?)
;   contract from: primes.rkt
;   blaming: primes.rkt
;    (assuming the contract is correct)
```

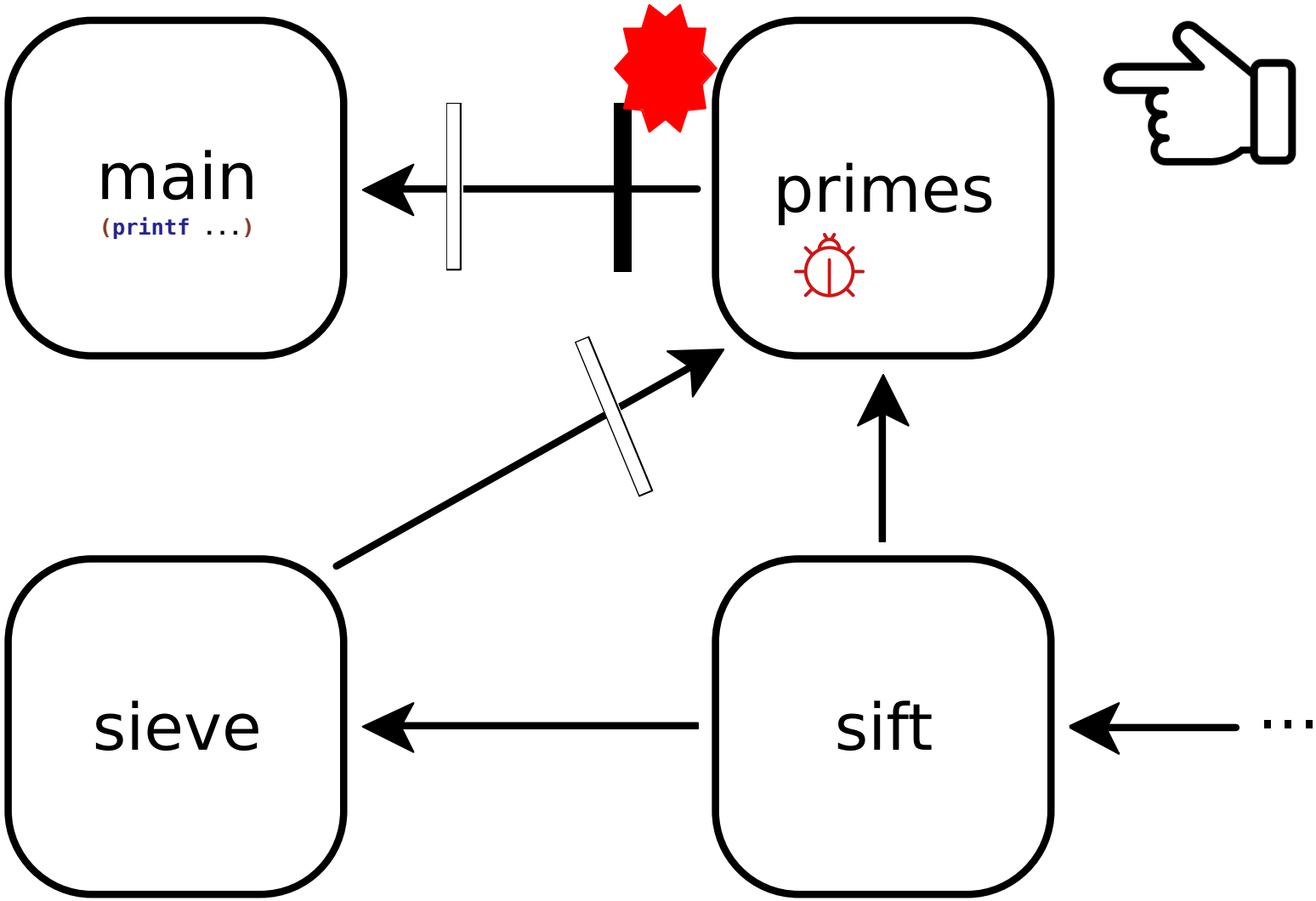
Violations blame a component



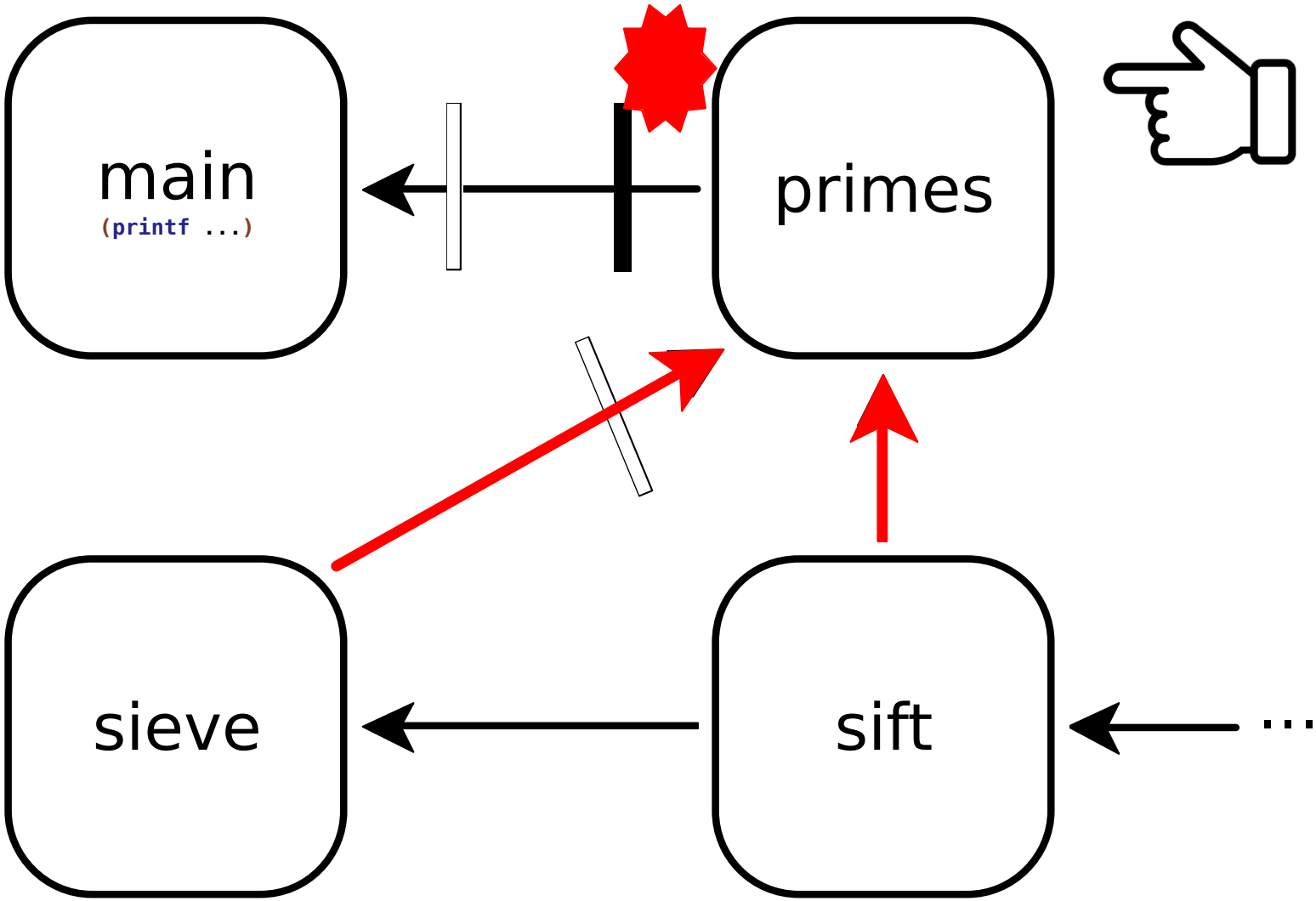
But blame may be far from the bug



Interpreting blame



Interpreting blame



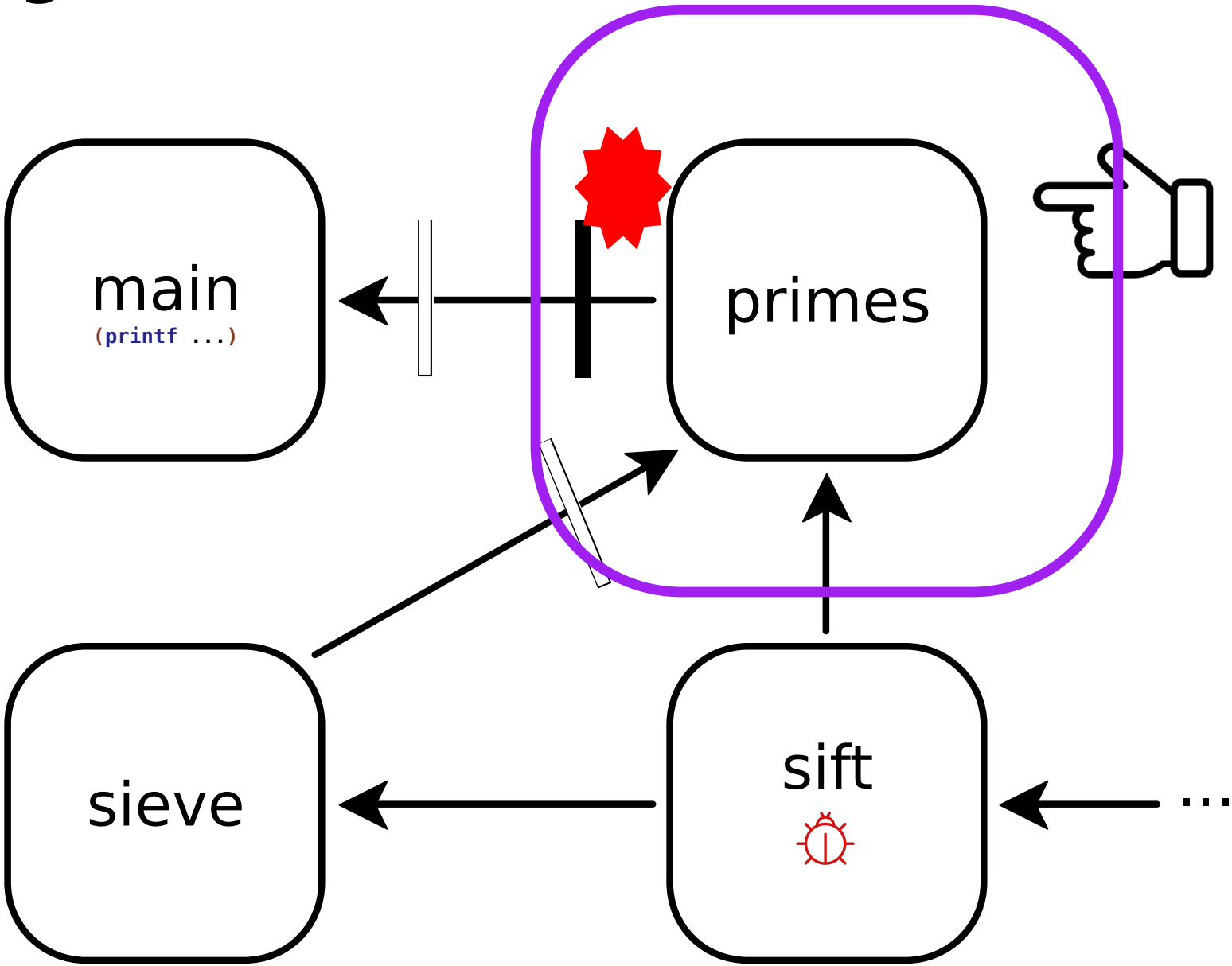
How does blame help?

“Correct blame assignment **gets programmers started** with the **debugging process**”
[Dimoulas et al. POPL'11]

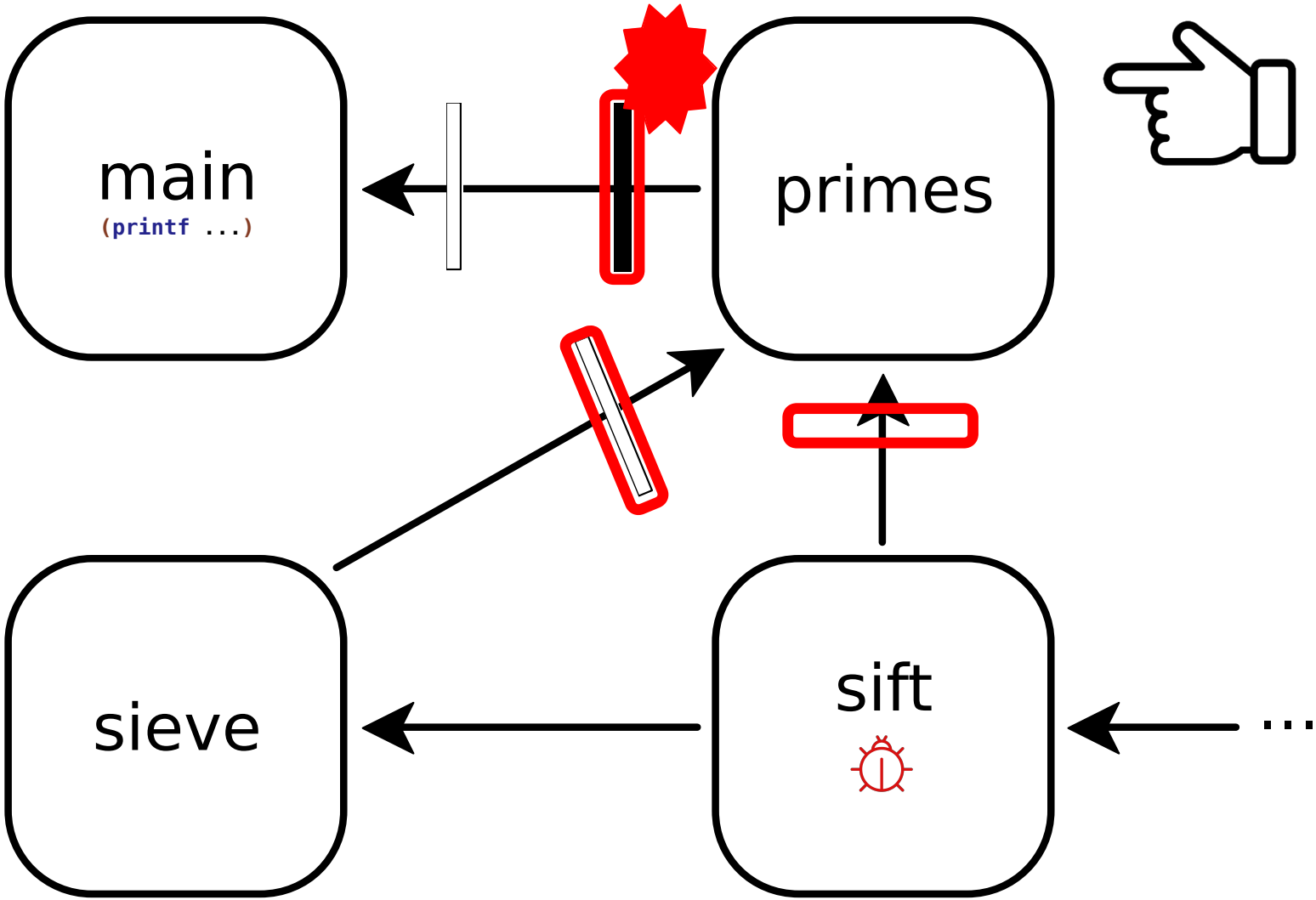
“After all, the **purpose of blame** assignment is to **assist programmers with debugging**”
[Strickland & Felleisen IFL'09]

Blame “narrow[s] down the **search for the error**”
[Dimoulas et al. OOPSLA'13]

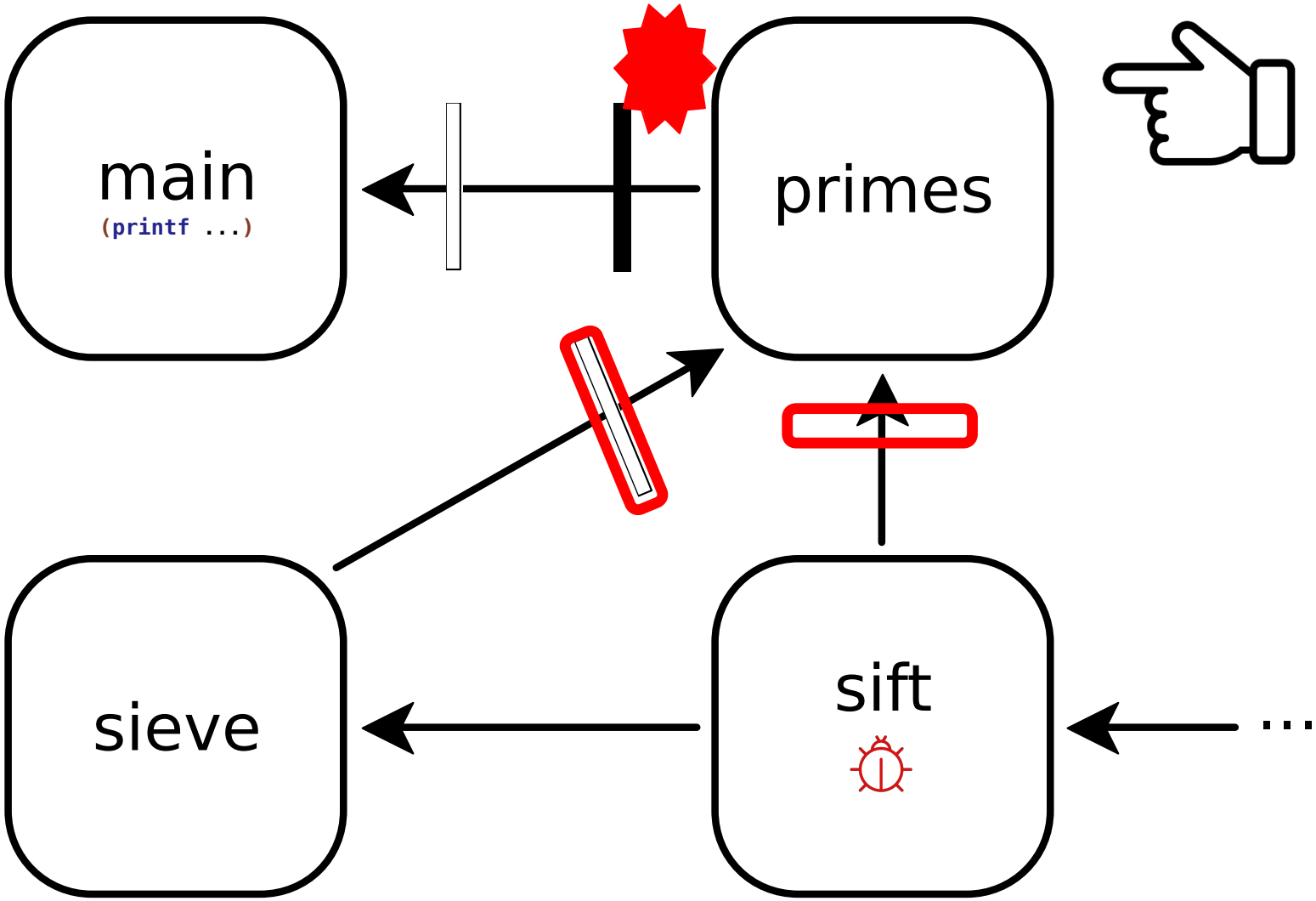
Using blame to focus our attention



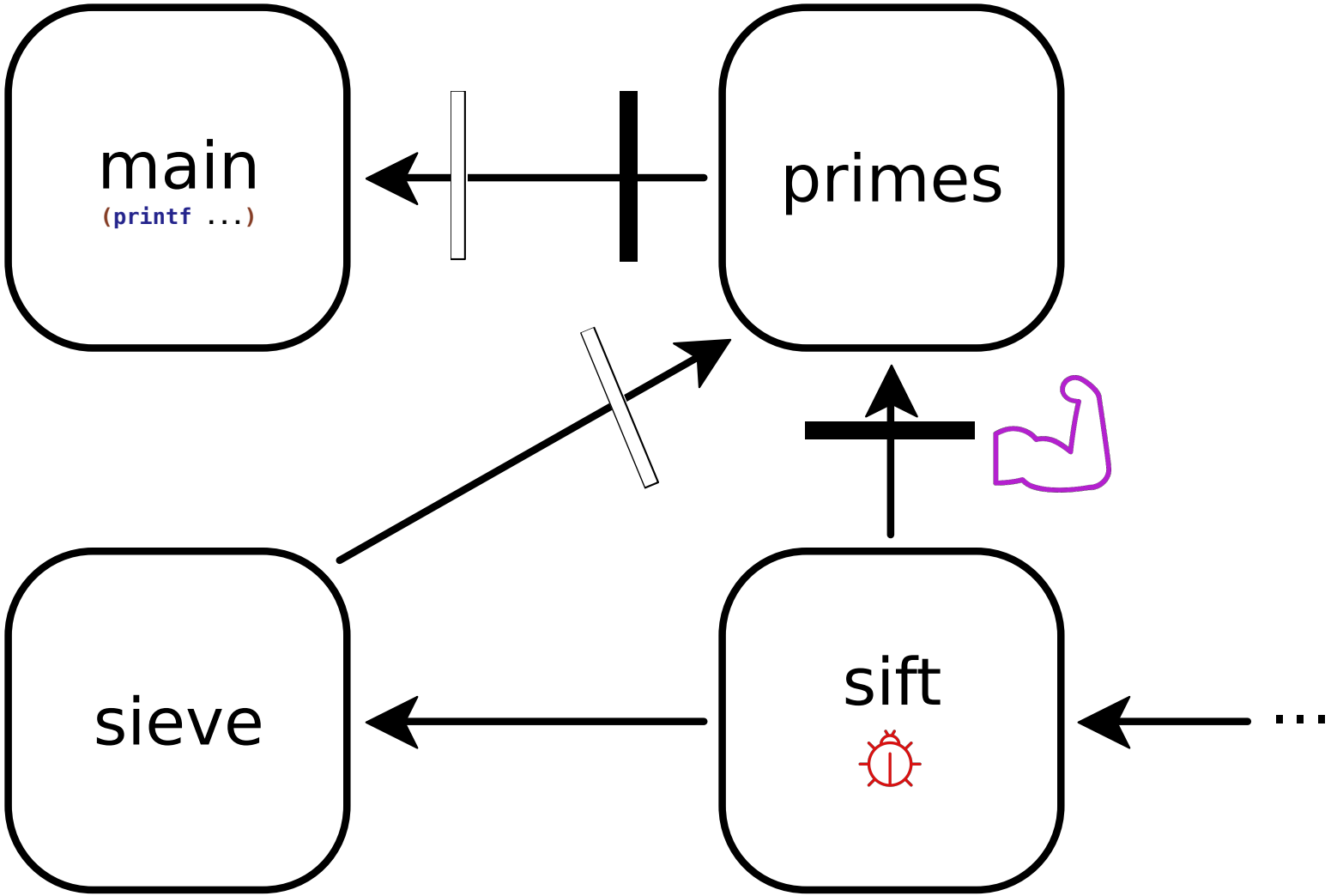
Contracts around a component



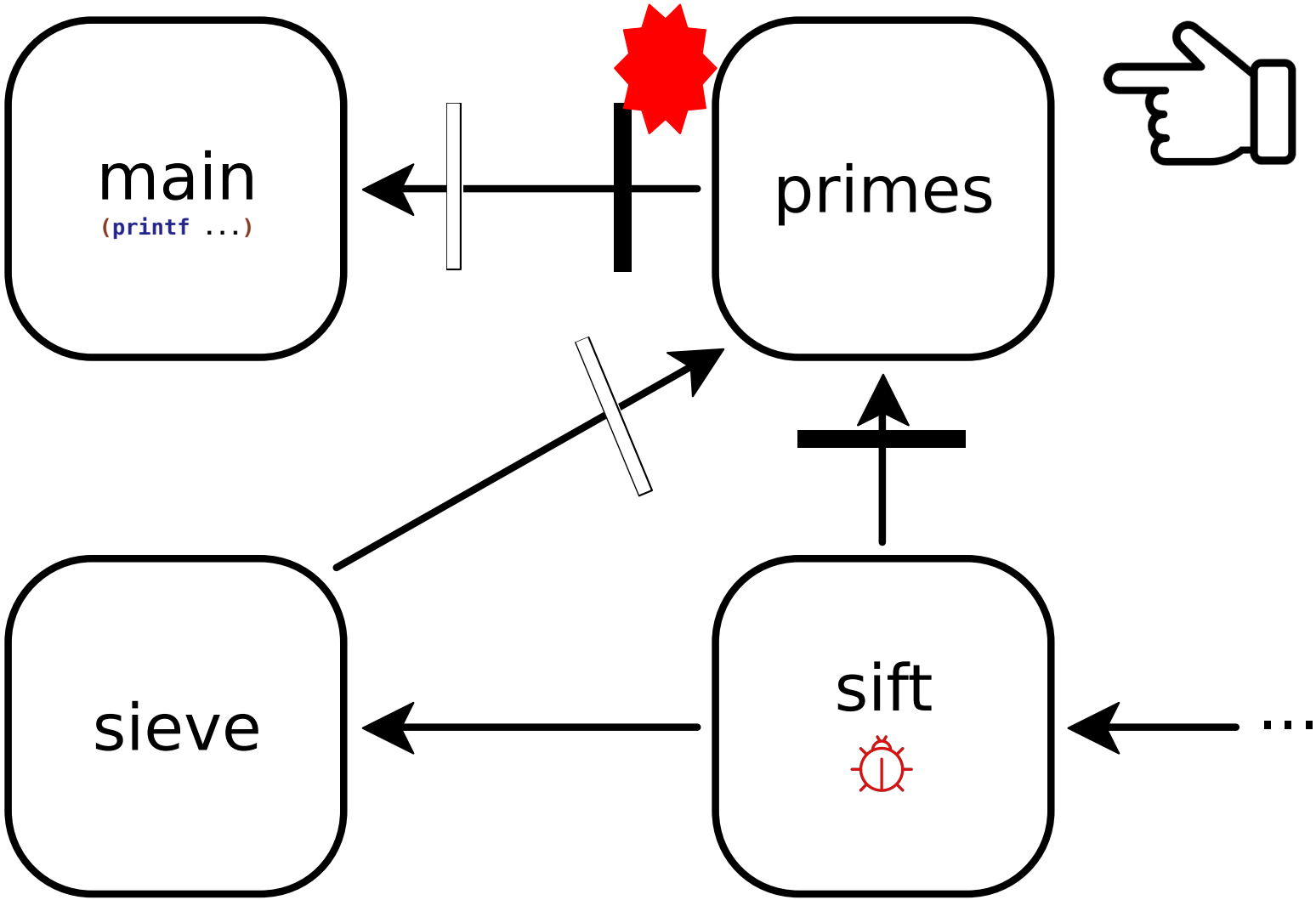
Contracts around a component



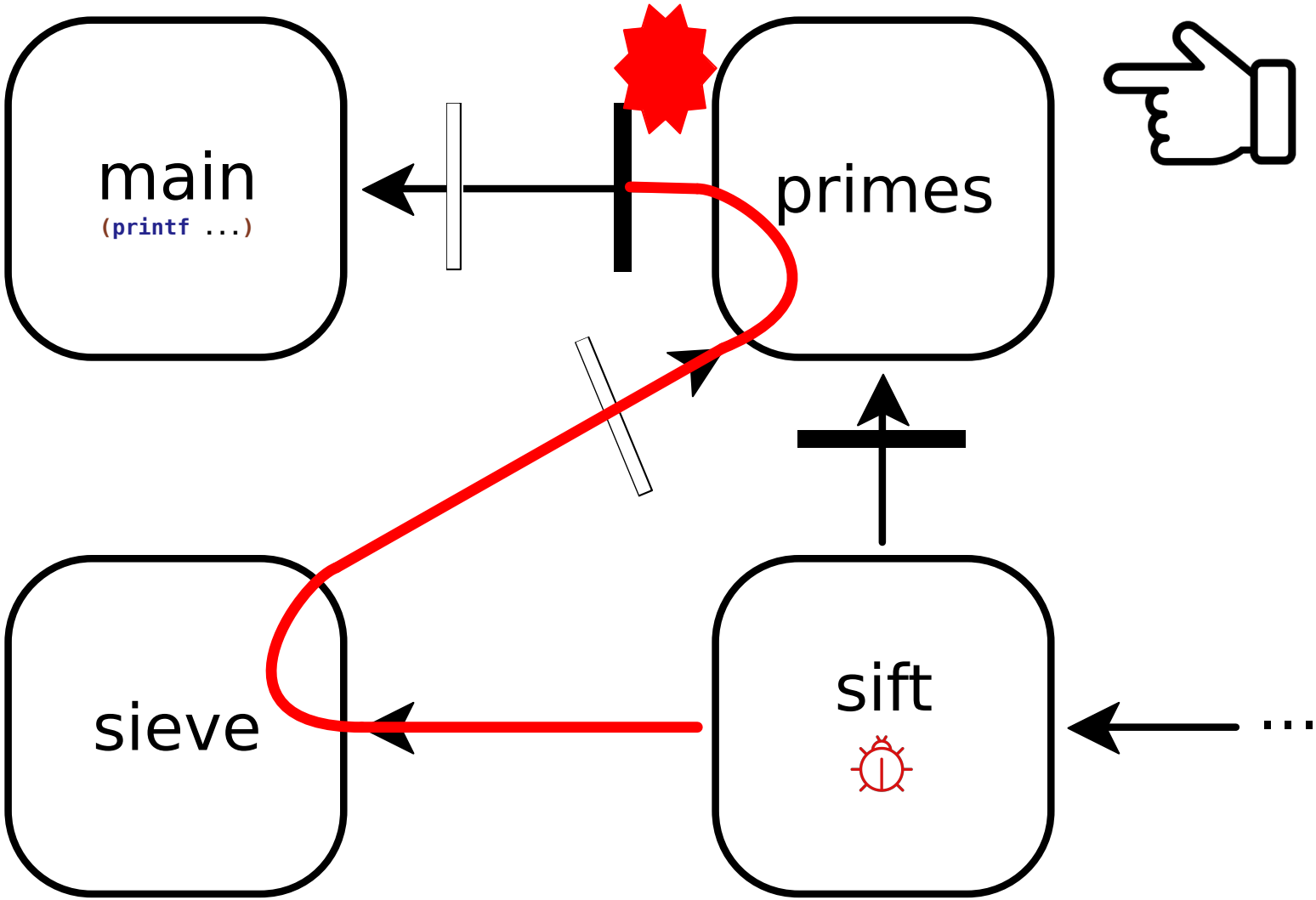
Trying to shift blame



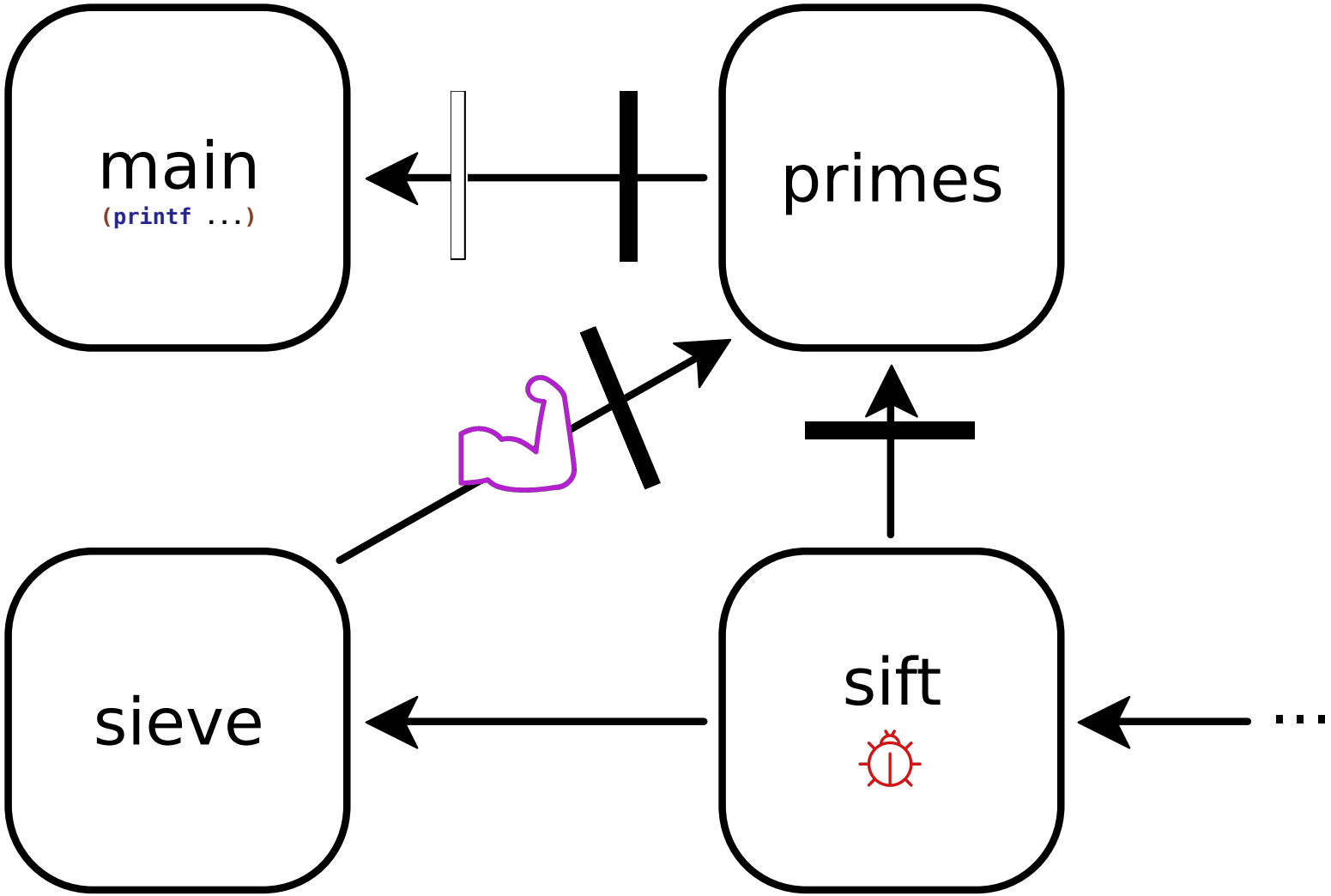
Trying to shift blame



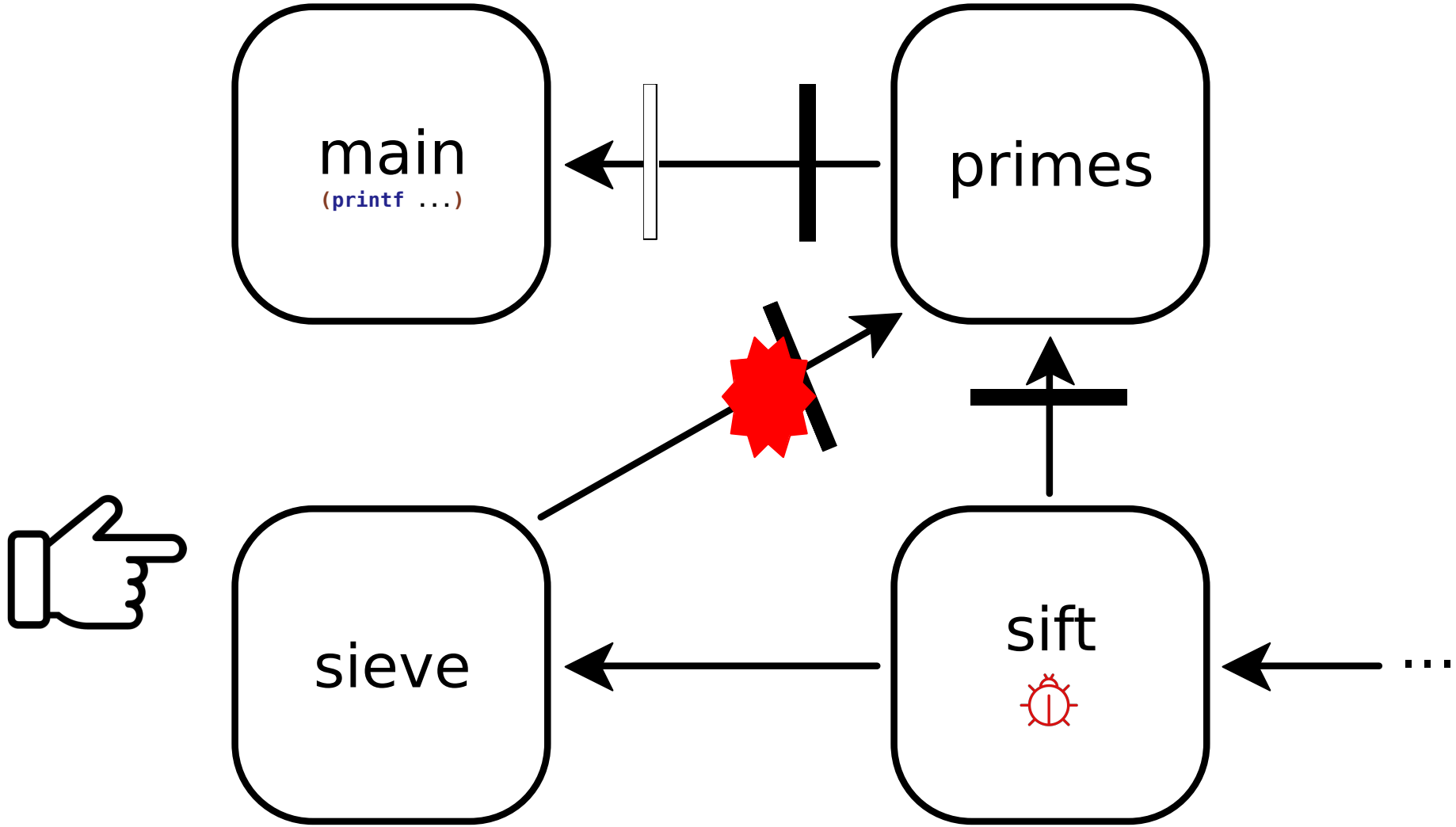
Trying to shift blame



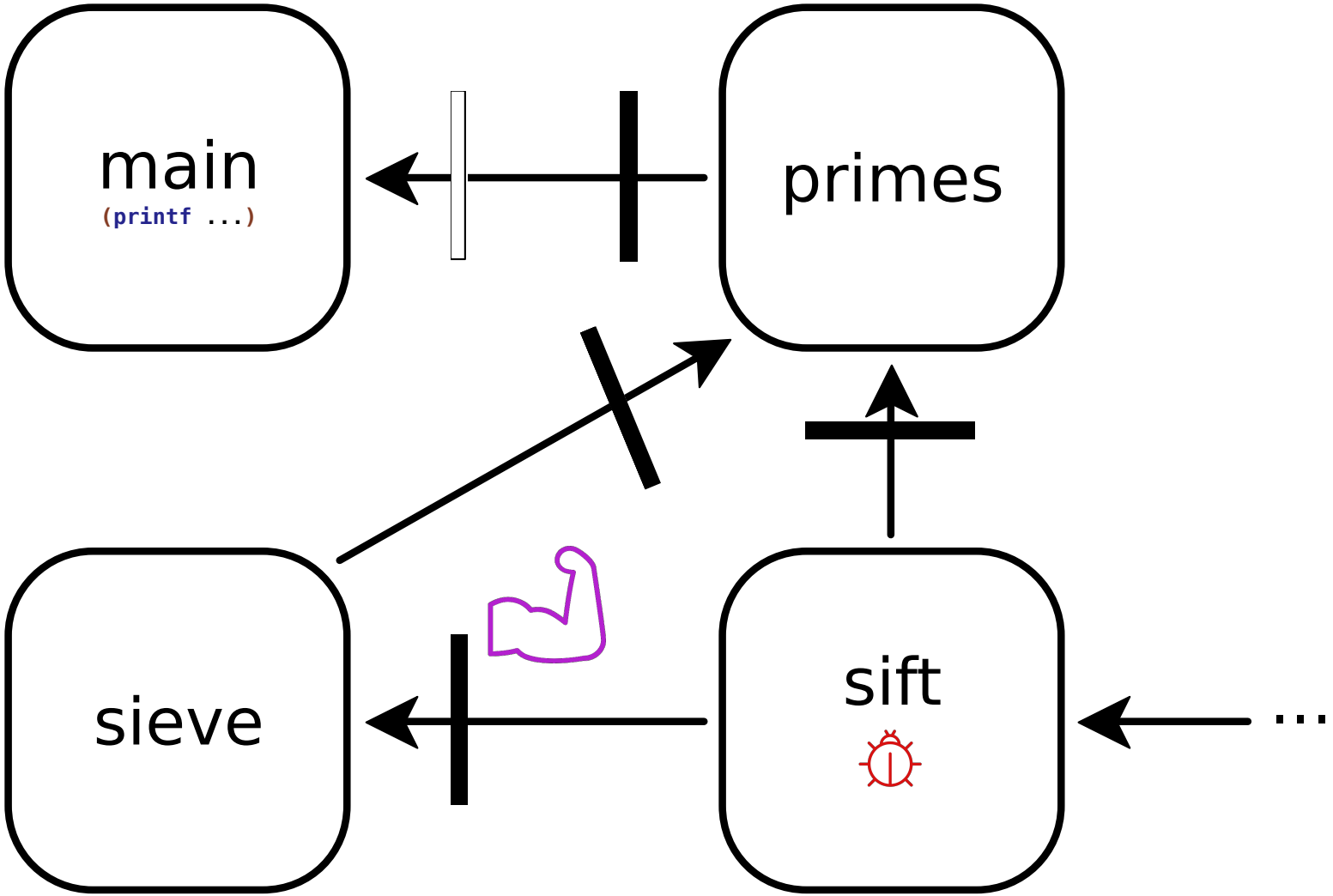
Trying to shift blame



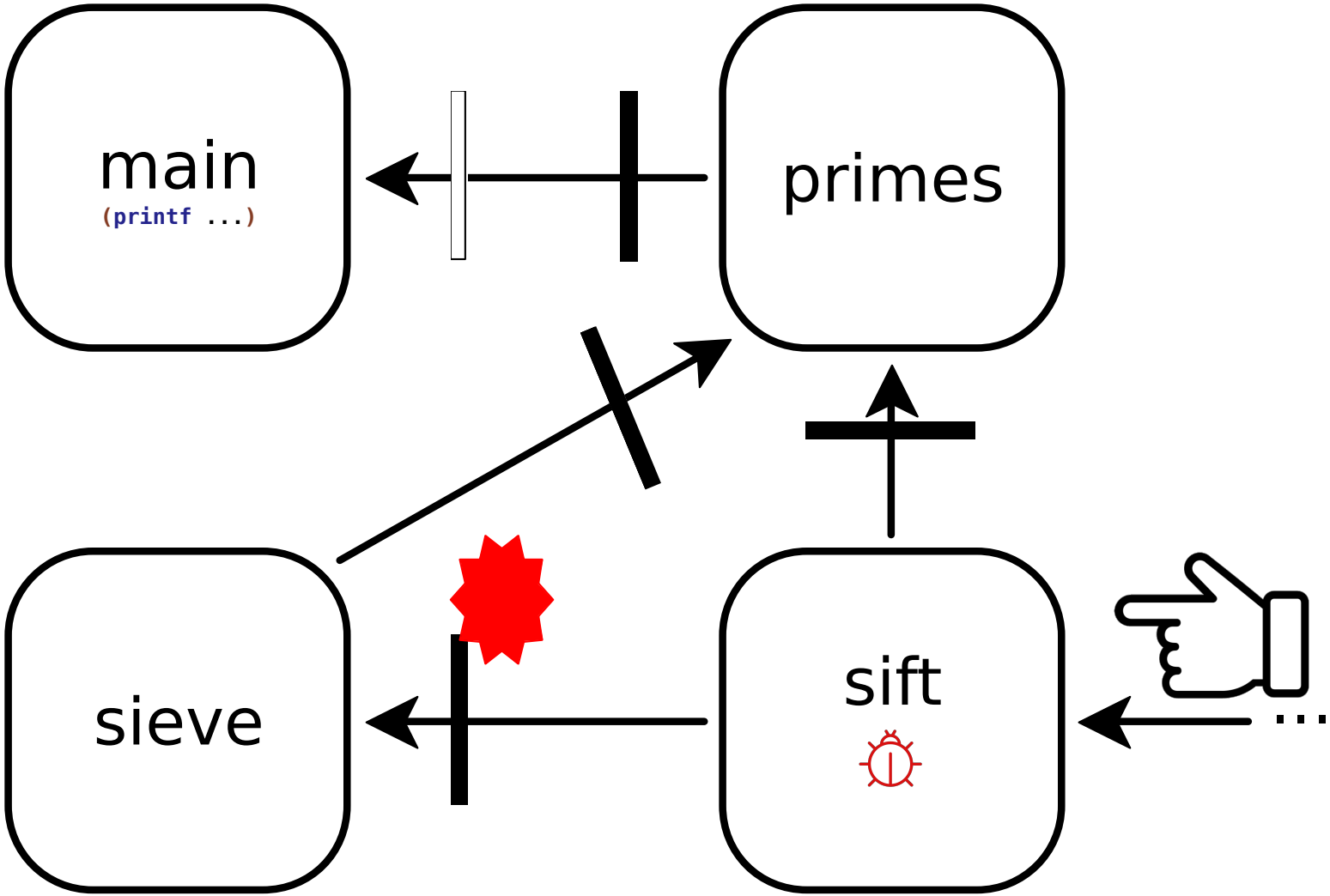
Trying to shift blame



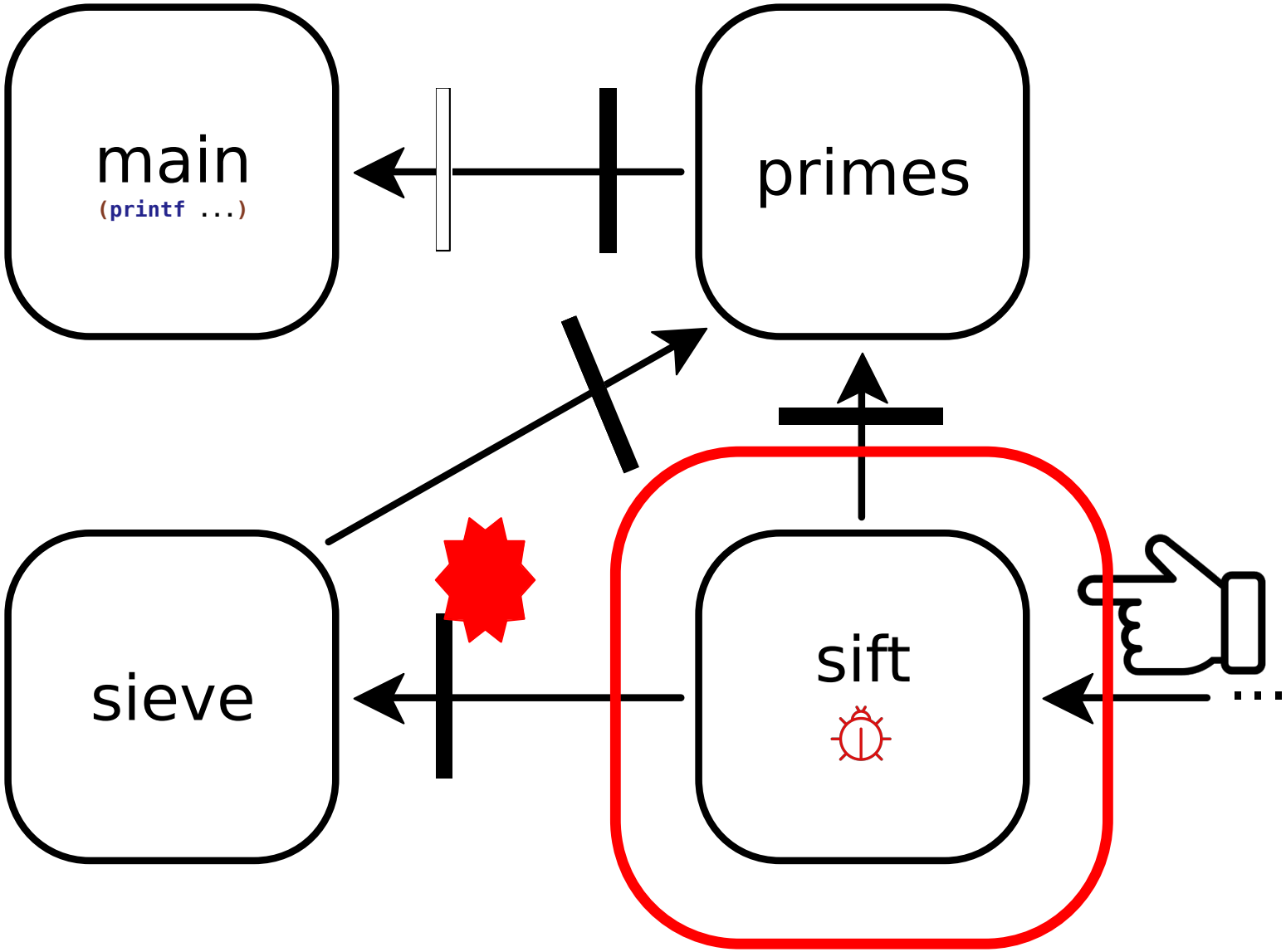
Trying to shift blame



Trying to shift blame



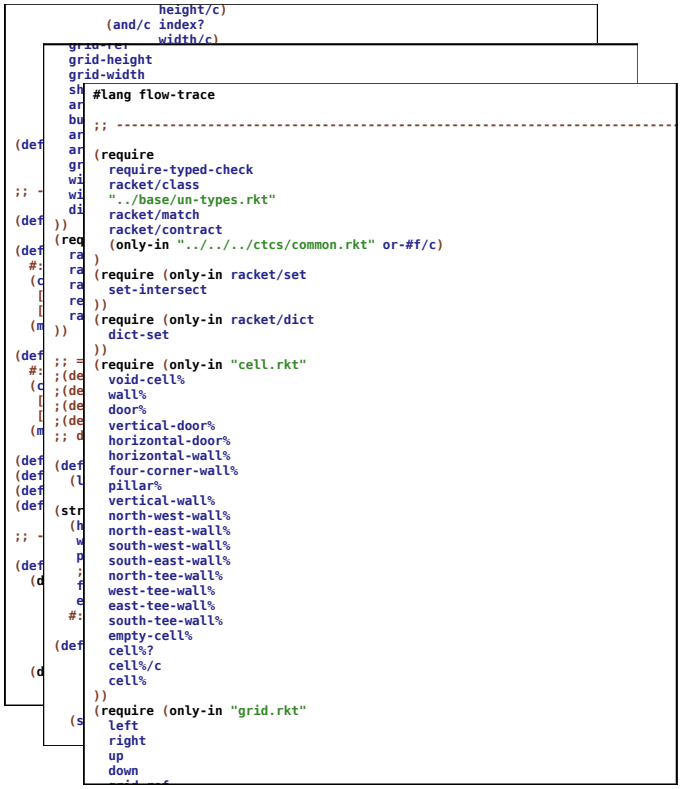
Blame shifting should lead us to the bug



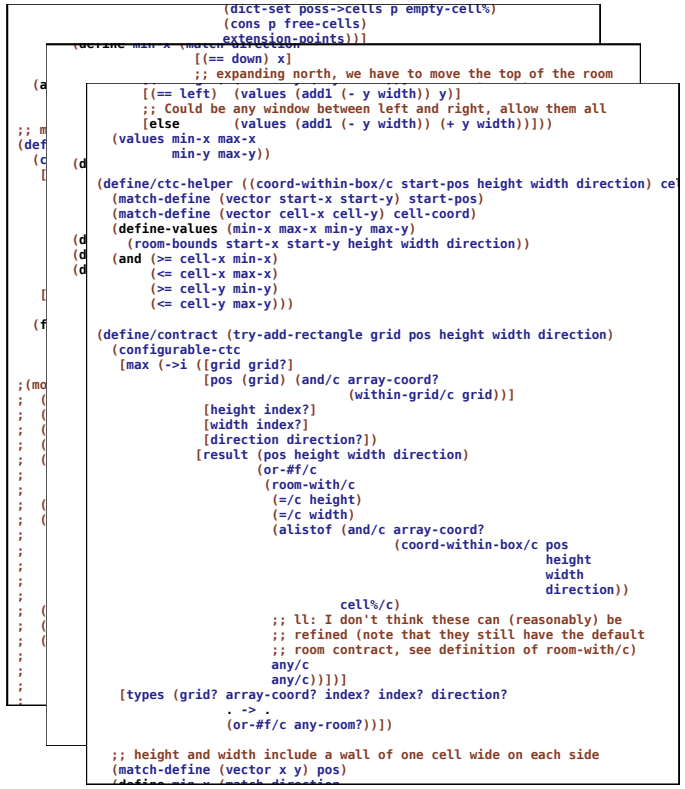
The hypothesis, a little more precisely

Blame can be systematically translated into the location of the bug via blame shifting

Empirically evaluating blame shifting

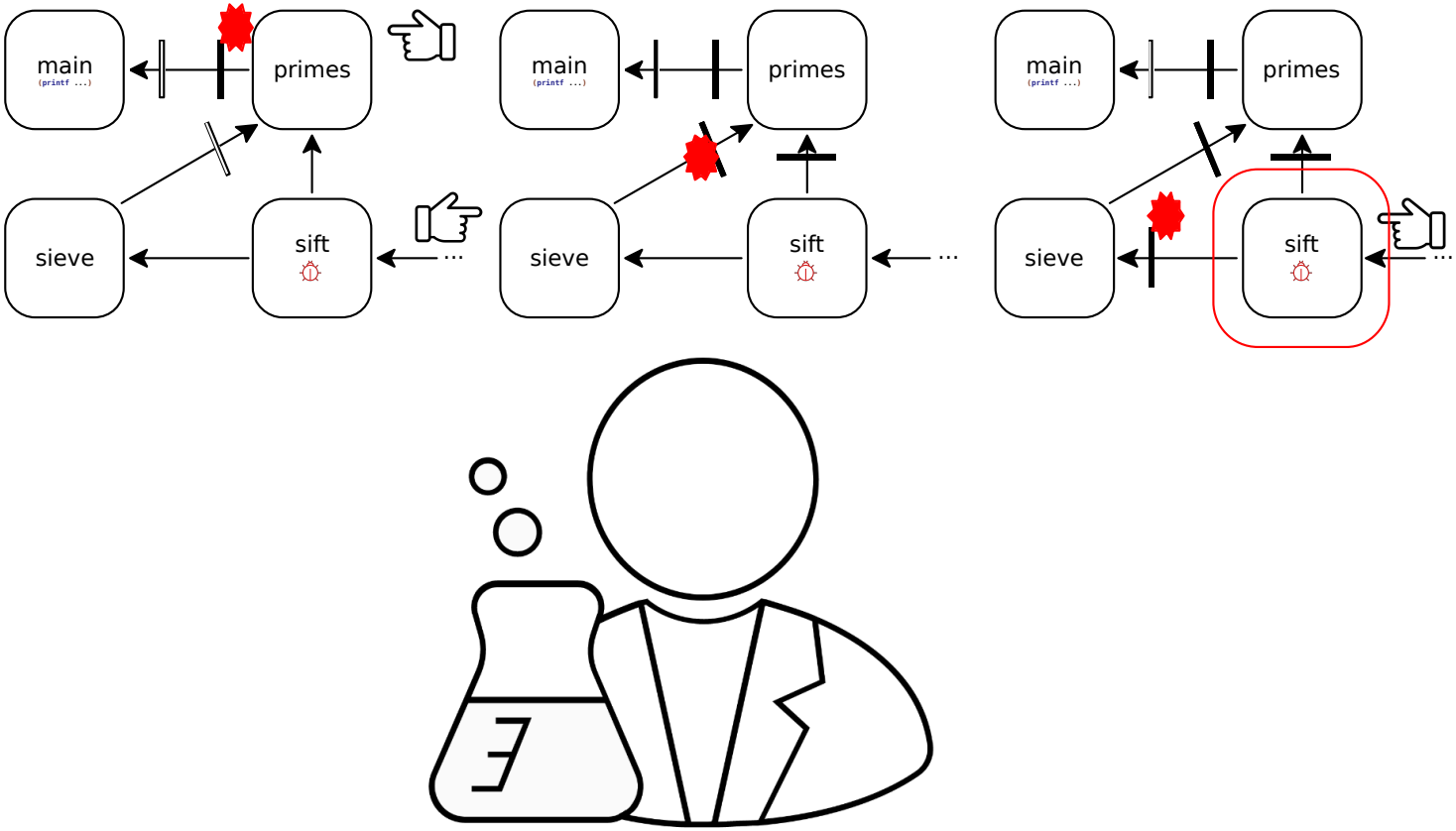


```
#lang flow-trace

;; ------------------------------------------------------------------------

(require
  require-typed-check
  racket/class
  "../base/un-types.rkt"
  racket/match
  racket/contract
  (only-in "../../../ctcs/common.rkt" or-#f/c)
)
(require (only-in racket/set
  set-intersect
))
(require (only-in racket/dict
  dict-set
))
(require (only-in "cell.rkt"
  void-cell%
  wall%
  door%
  vertical-door%
  horizontal-door%
  horizontal-wall%
  four-corner-wall%
  pillar%
  vertical-wall%
  north-west-wall%
  north-east-wall%
  south-west-wall%
  south-east-wall%
  north-tee-wall%
  west-tee-wall%
  east-tee-wall%
  south-tee-wall%
  empty-cell%
  cell%?
  cell%/c
  cell%
))
(require (only-in "grid.rkt"
  left
  right
  up
  down
```

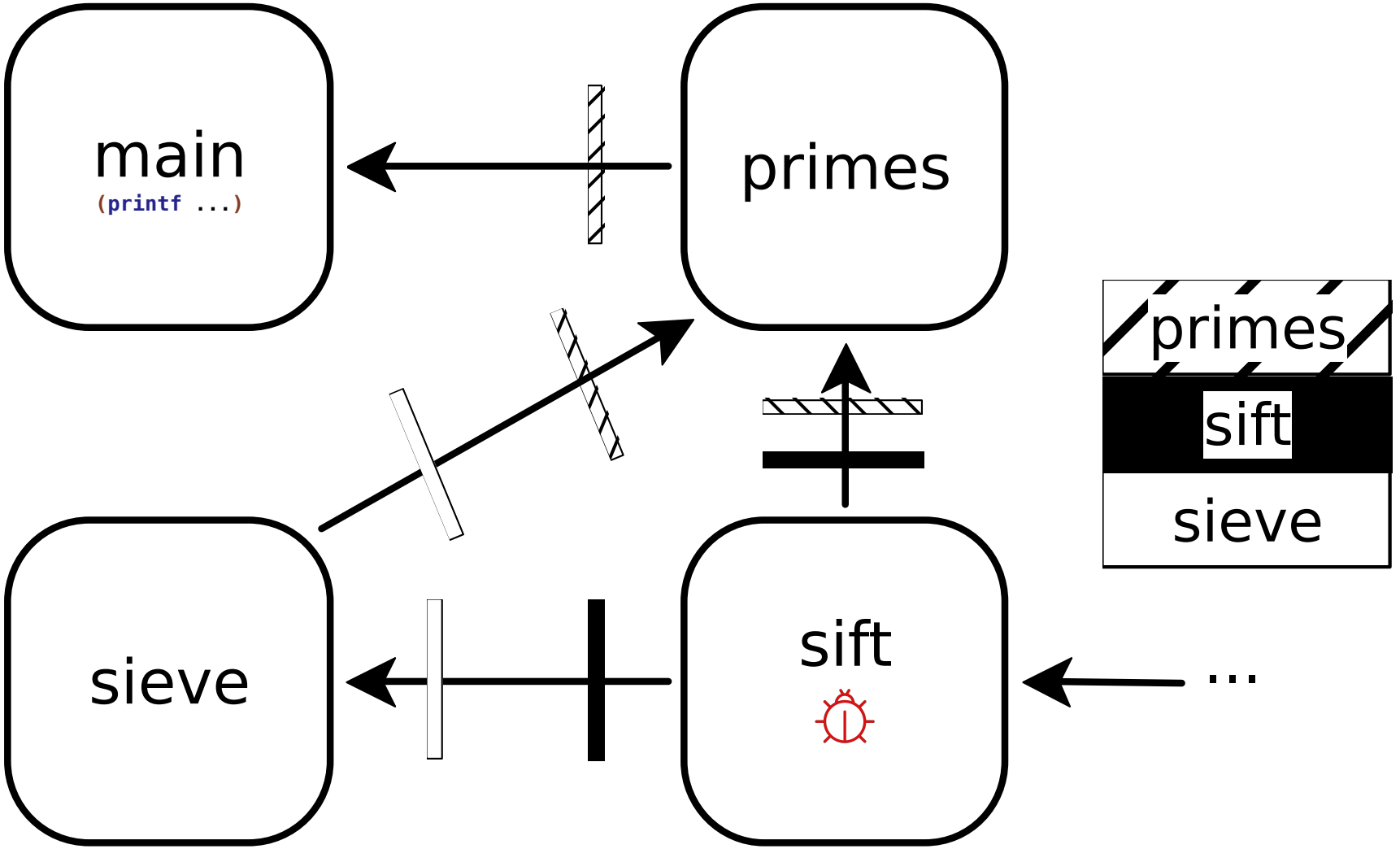


```
      [(== left)  (values (add1 (- y width)) y)]
      ;; Could be any window between left and right, allow them all
      [else       (values (add1 (- y width)) (+ y width))]))
  (values min-x max-x
          min-y max-y))

(define/ctc-helper ((coord-within-box/c start-pos height width direction) ce
  (match-define (vector start-x start-y) start-pos)
  (match-define (vector cell-x cell-y) cell-coord)
  (define-values (min-x max-x min-y max-y)
    (room-bounds start-x start-y height width direction))
  (and (>= cell-x min-x)
       (<= cell-x max-x)
       (>= cell-y min-y)
       (<= cell-y max-y)))

(define/contract (try-add-rectangle grid pos height width direction)
  (configurable-ctc
   [max (->i ([grid grid?]
              [pos (grid) (and/c array-coord?
                                 (within-grid/c grid))]
              [height index?]
              [width index?]
              [direction direction?])
             [result (pos height width direction)
                     (or-#f/c
                      (room-with/c
                       (=/c height)
                       (=/c width)
                       (alistof (and/c array-coord?
                                       (coord-within-box/c pos
                                                           height
                                                           width
                                                           direction))
                                cell%/c)
                       ;; ll: I don't think these can (reasonably) be
                       ;; refined (note that they still have the default
                       ;; room contract, see definition of room-with/c)
                       any/c
                       any/c))])]
   [types (grid? array-coord? index? index? direction?
           . -> .
           (or-#f/c any-room?))])

  ;; height and width include a wall of one cell wide on each side
  (match-define (vector x y) pos)
```

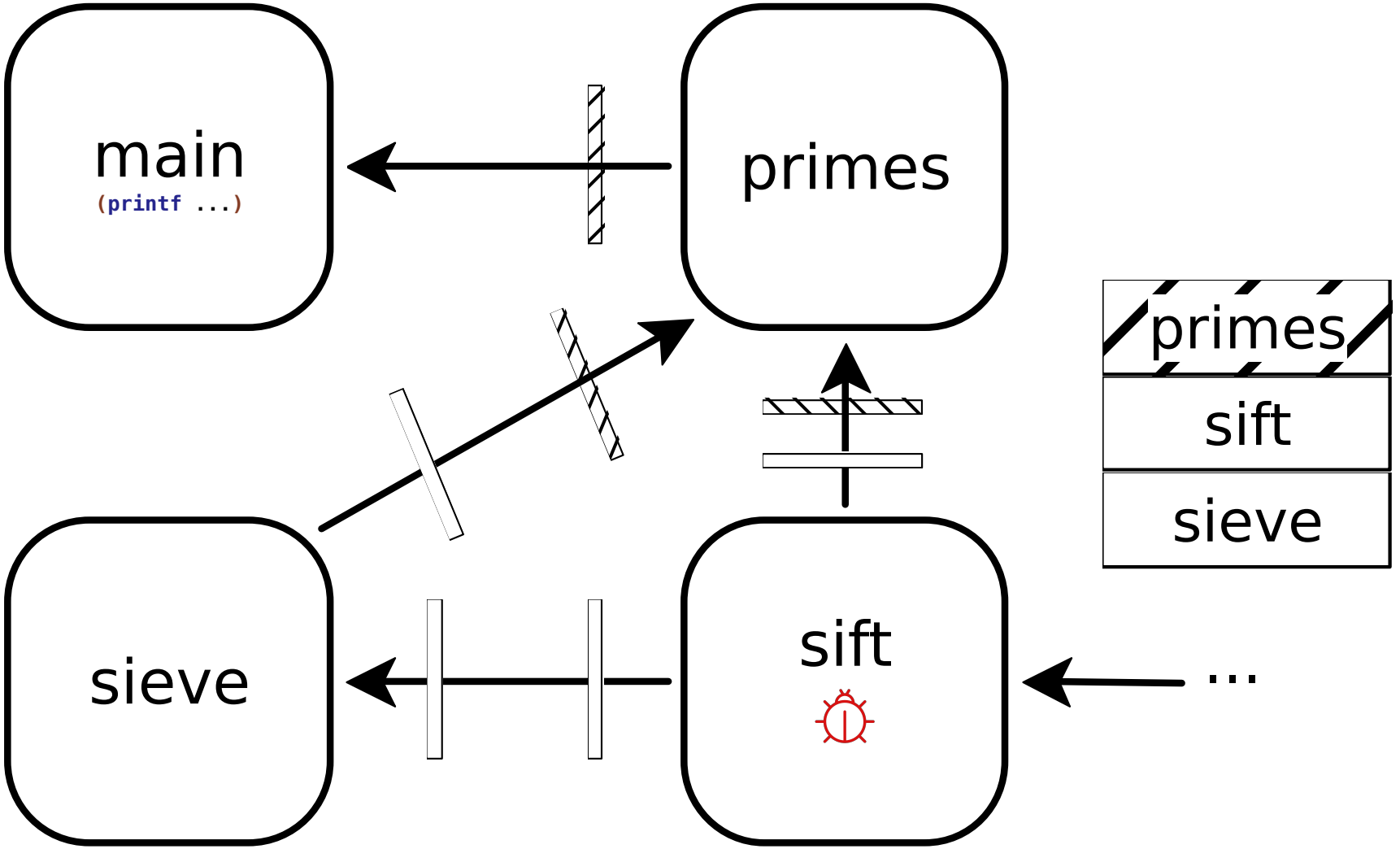
Empirically evaluating blame shifting



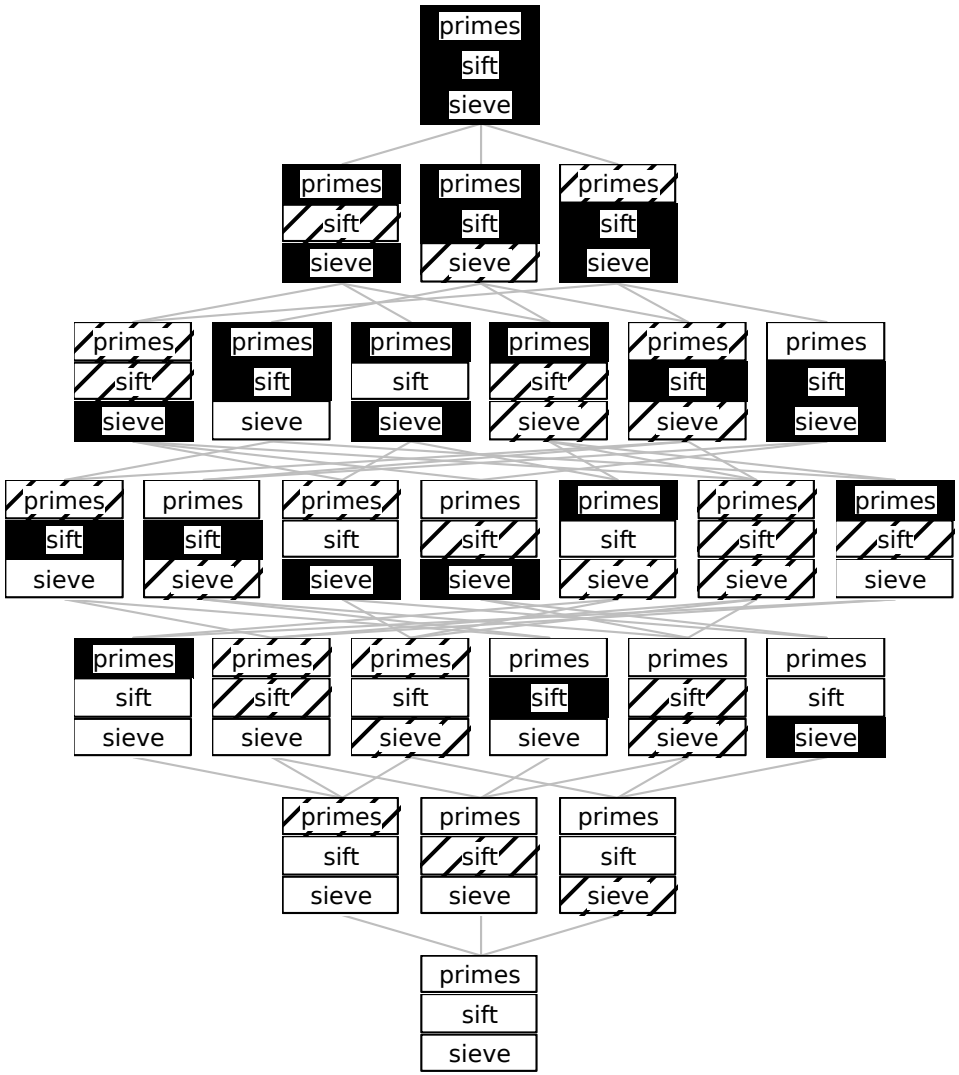
Configuring the program with contracts



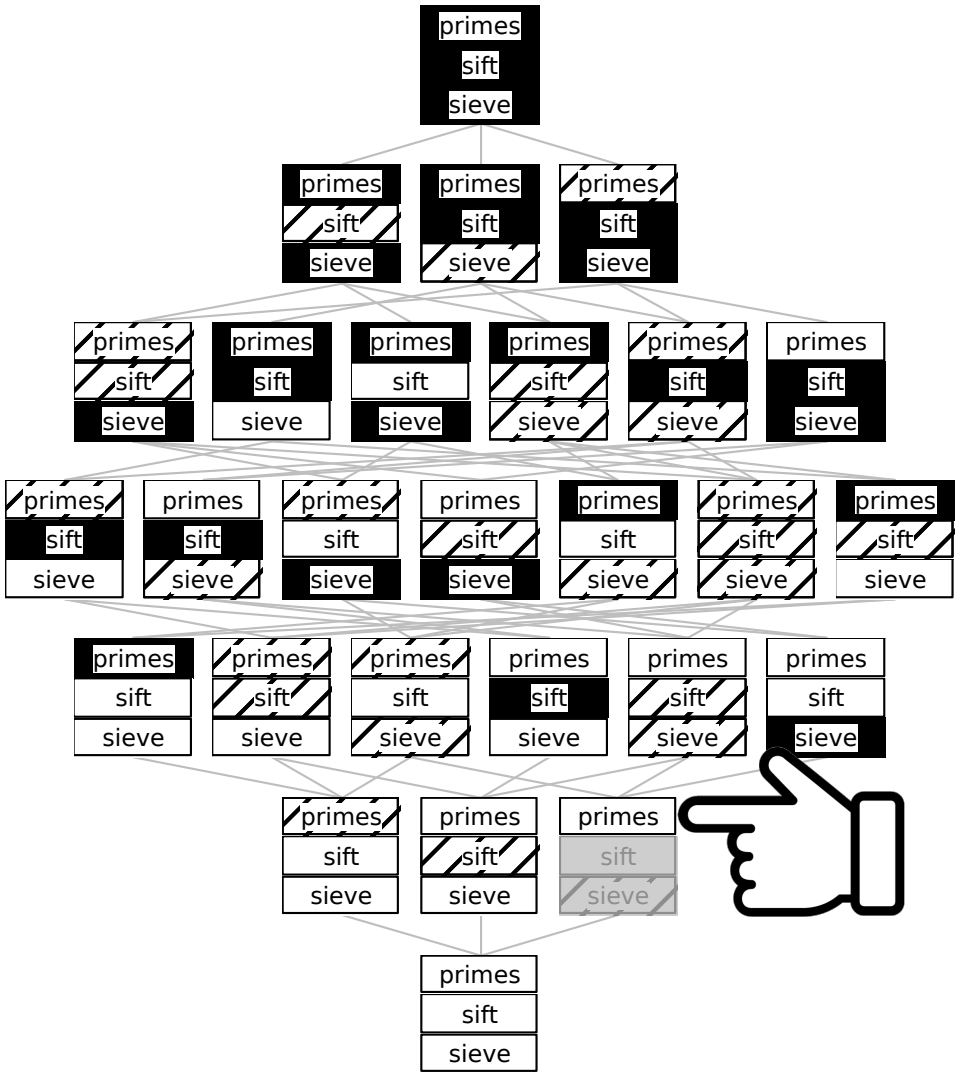
Configuring the program with contracts



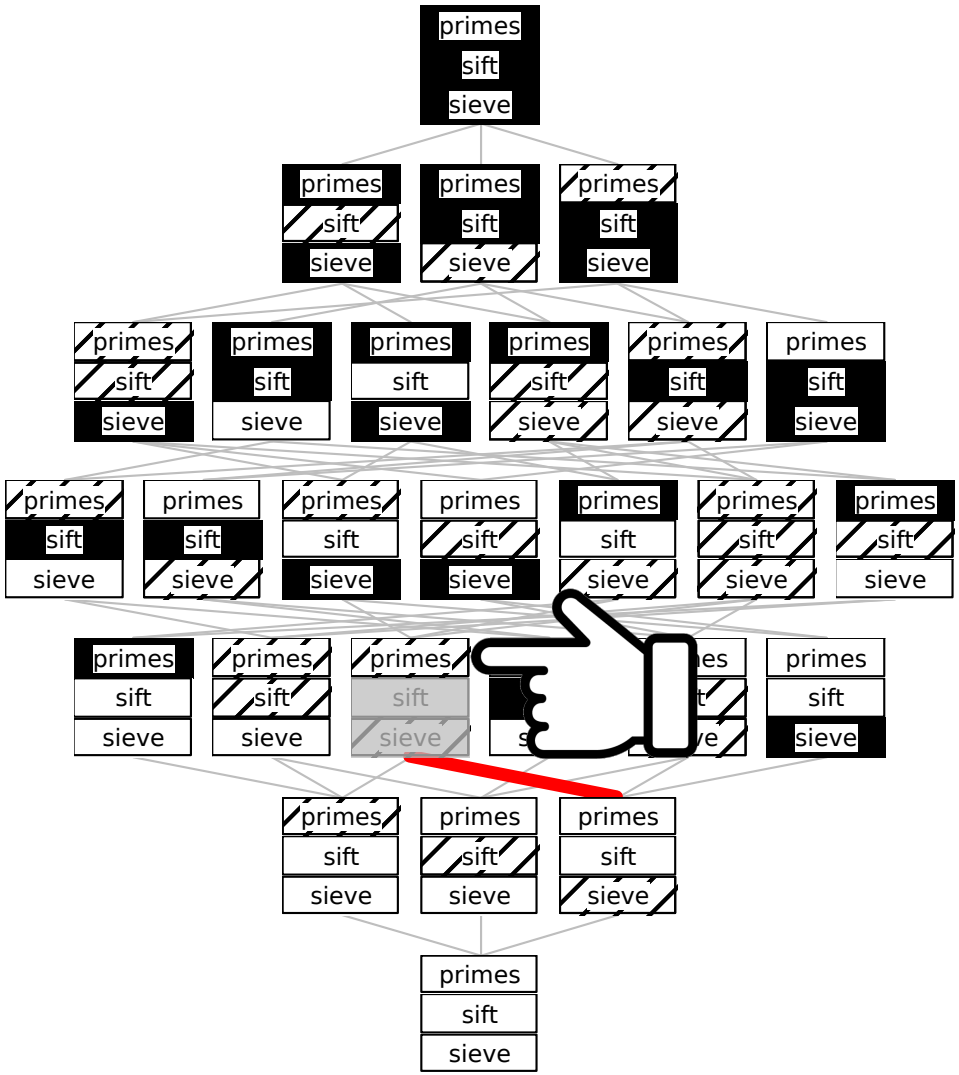
Configurations form a lattice



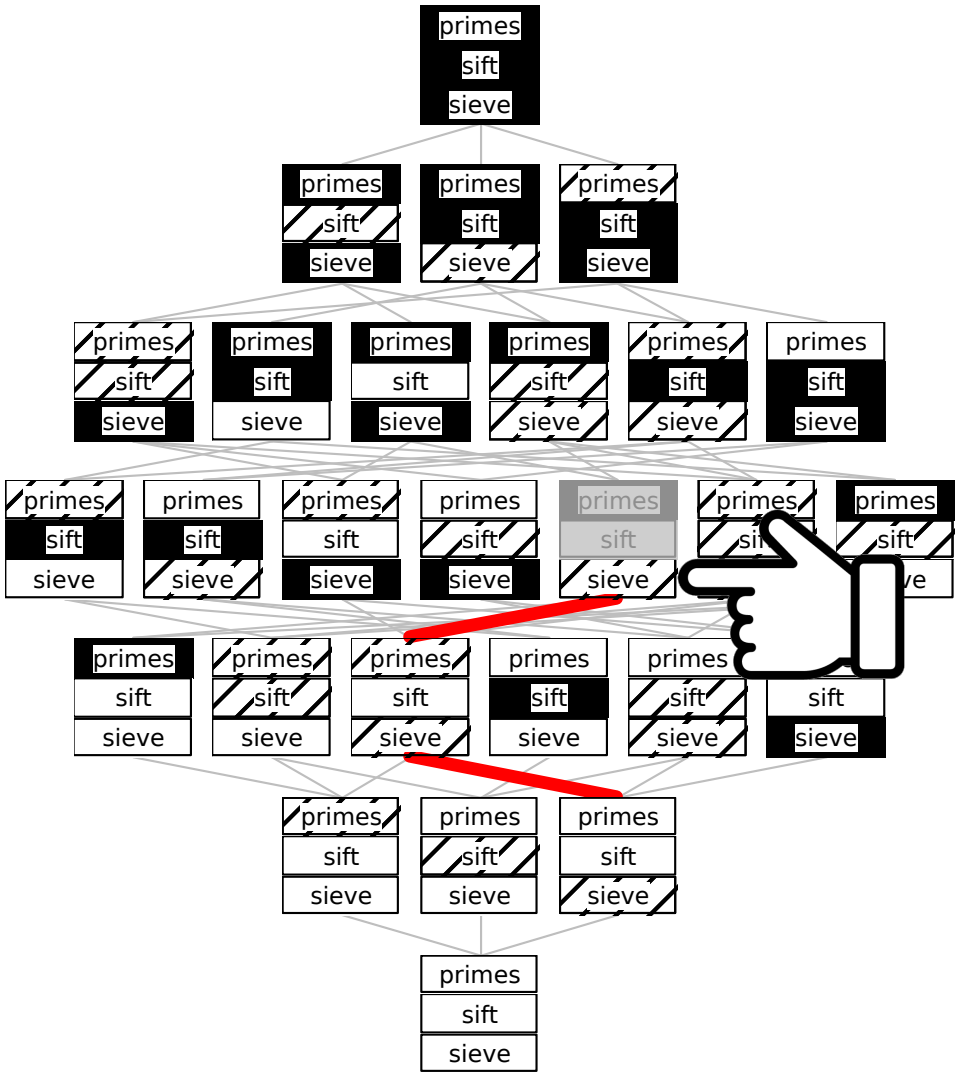
Blame shifting forms a blame trail



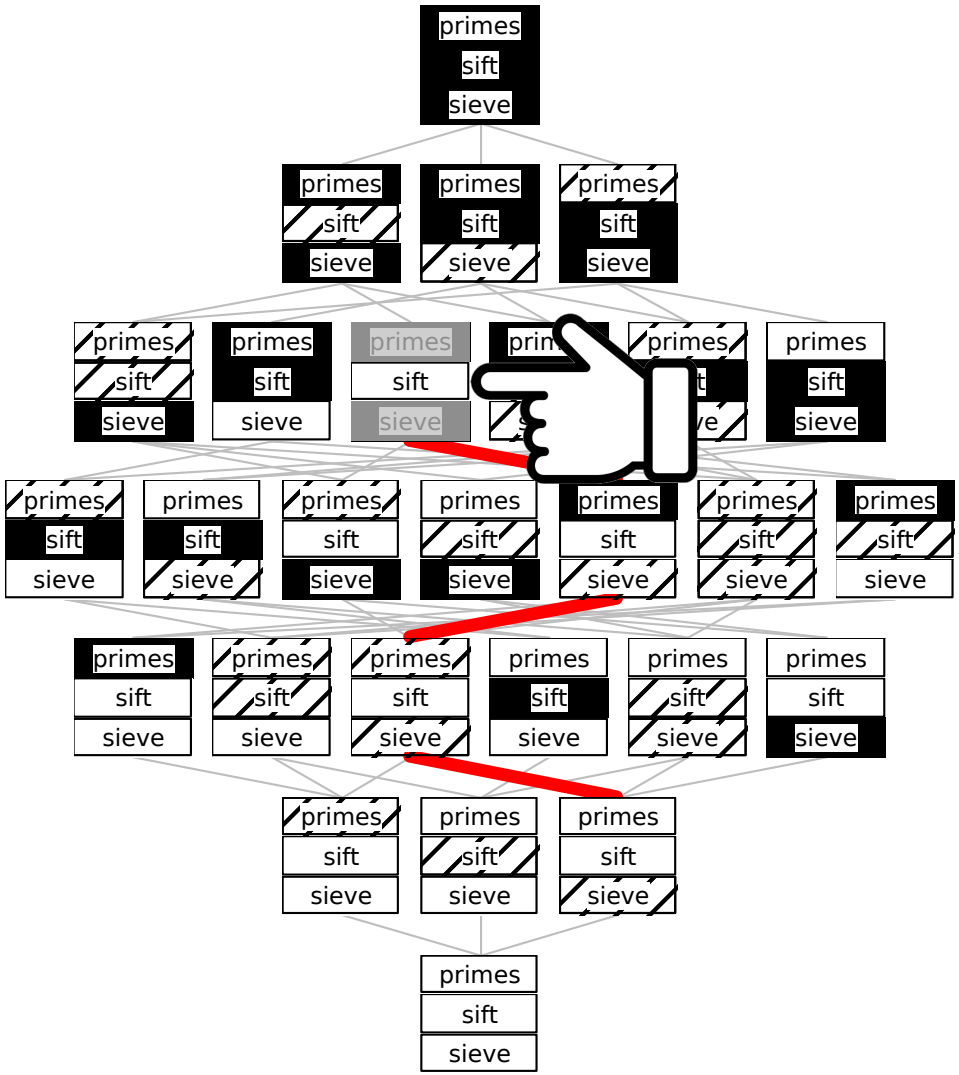
Blame shifting forms a blame trail



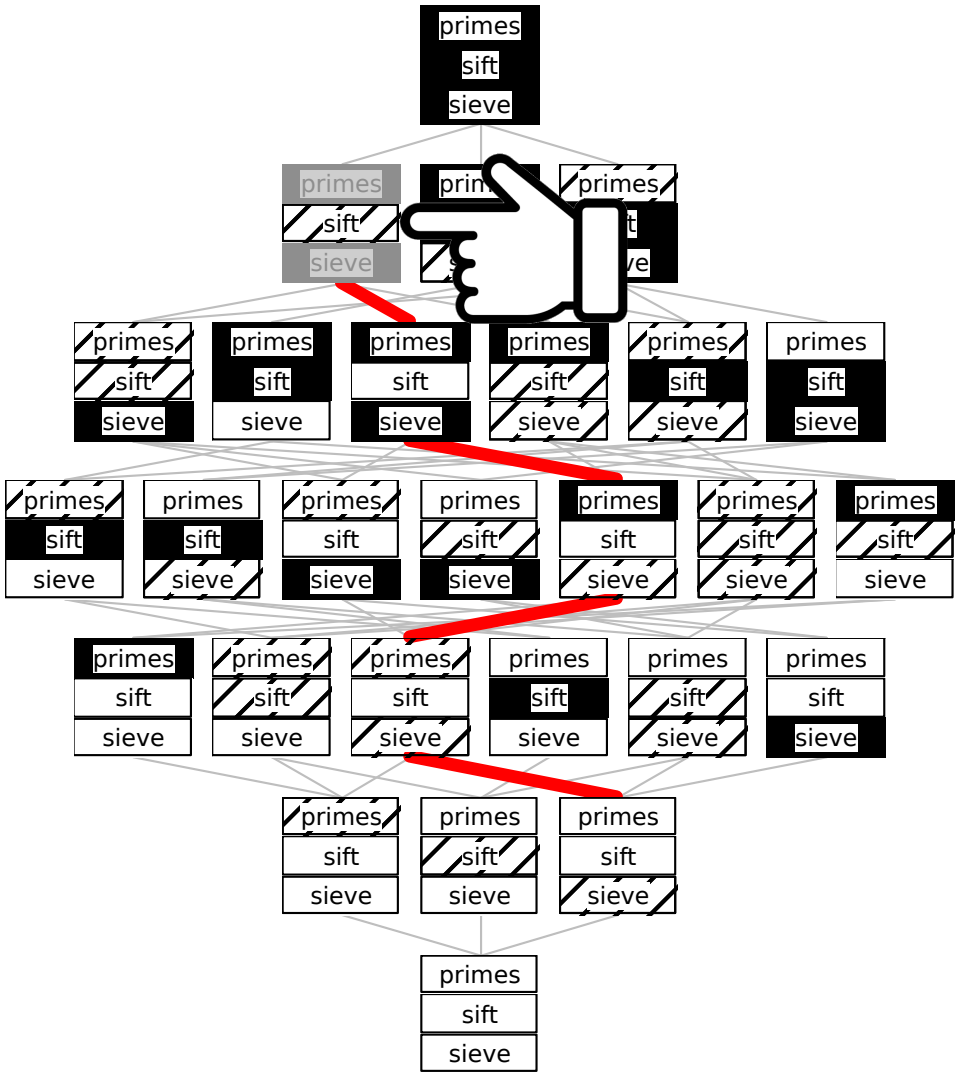
Blame shifting forms a blame trail



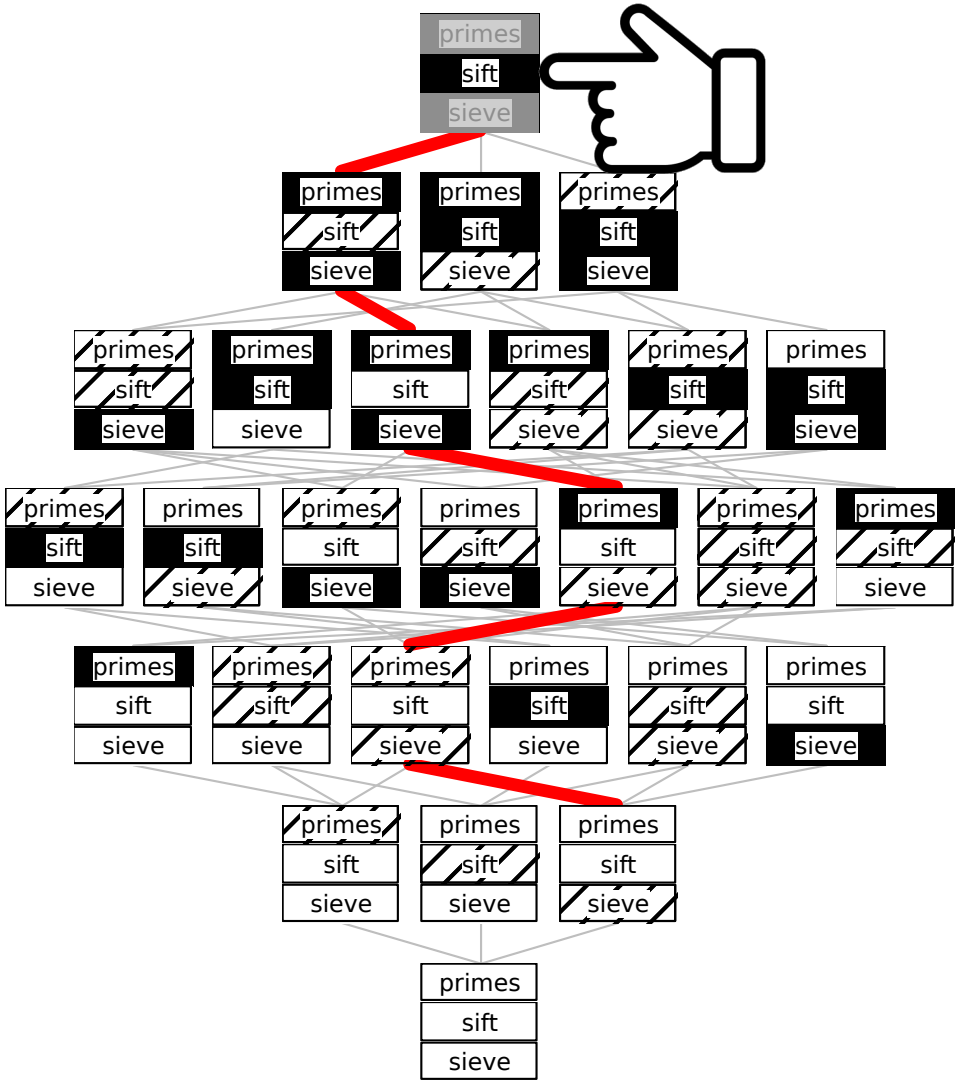
Blame shifting forms a blame trail



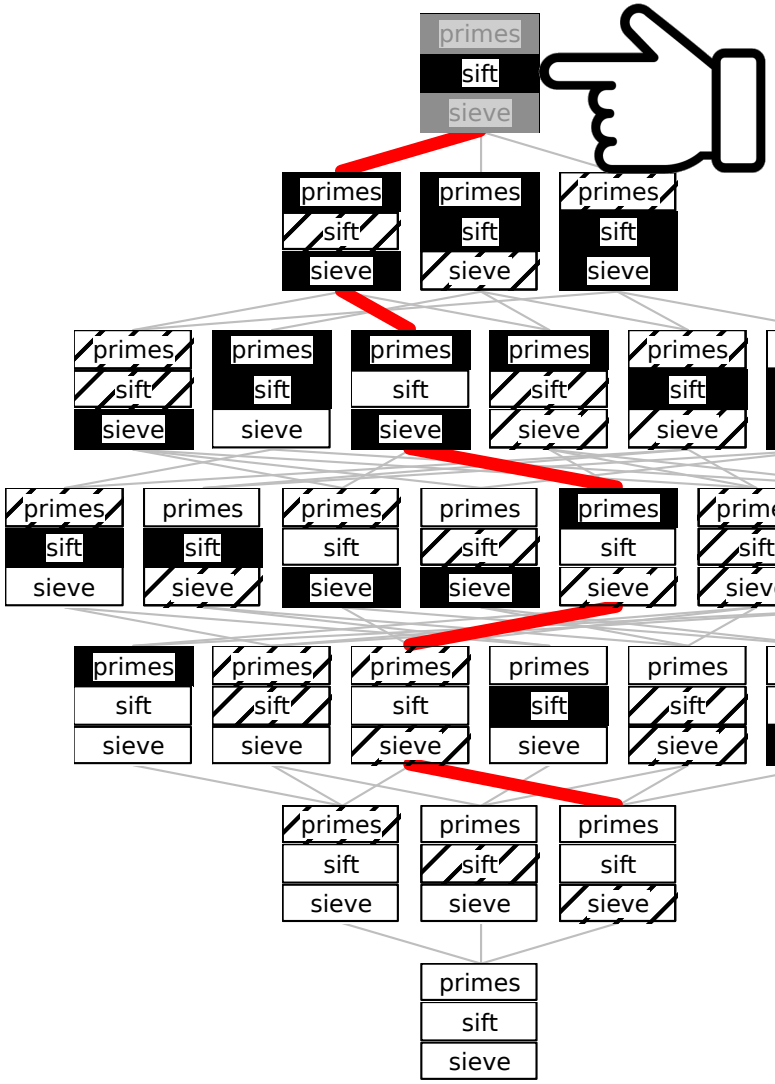
Blame shifting forms a blame trail



Blame shifting forms a blame trail



What we want to know



sift contains bug?

The hypothesis, precisely

Every blame trail ends on a configuration that blames the buggy component

Applying the experiment to Racket

- 7 programs
  [Takikawa et al. POPL'16, Greenman et al. JFP'19]
- contracts at 3 strength levels
  (none, types, full)
- 228 viable mutants
  e.g. `(+ 4 2)` ⟶ `(- 4 2)`
- lattice size from 59k to 14 octillion (10^{27})
- 43k blame trails across all programs

Applying the experiment to Racket

- **7 programs**
  [Takikawa et al. POPL'16, Greenman et al. JFP'19]
- contracts at 3 strength levels
  (none, types, full)
- 228 viable mutants
  e.g. `(+ 4 2)` ⟶ `(- 4 2)`
- lattice size from 59k to 14 octillion (10^{27})
- 43k blame trails across all programs

Applying the experiment to Racket

- 7 programs
  [Takikawa et al. POPL'16, Greenman et al. JFP'19]
- contracts at **3 strength levels**
  (none, types, full)
- 228 viable mutants
  e.g. `(+ 4 2)` ⟶ `(- 4 2)`
- lattice size from 59k to 14 octillion (10^{27})
- 43k blame trails across all programs

Applying the experiment to Racket

- 7 programs
  [Takikawa et al. POPL'16, Greenman et al. JFP'19]
- contracts at 3 strength levels
  (none, types, full)
- **228 viable mutants**
  e.g. `(+ 4 2)` ⟶ `(- 4 2)`
- lattice size from 59k to 14 octillion (10^{27})
- 43k blame trails across all programs

Applying the experiment to Racket

- 7 programs
  [Takikawa et al. POPL'16, Greenman et al. JFP'19]
- contracts at 3 strength levels
  (none, types, full)
- 228 viable mutants
  e.g. `(+ 4 2)` ⟶ `(- 4 2)`
- lattice size from **59k to 14 octillion** (10^{27})
- 43k blame trails across all programs

Applying the experiment to Racket

- 7 programs
  [Takikawa et al. POPL'16, Greenman et al. JFP'19]
- contracts at 3 strength levels
  (none, types, full)
- 228 viable mutants
  e.g. `(+ 4 2)` ⟶ `(- 4 2)`
- lattice size from 59k to 14 octillion (10^{27})
- **43k blame trails** across all programs

Blame works

blame shifting works

43466

Blame works (mostly)

Blame works (mostly)

blame shifting works

43466

blame doesn't settle on bug

276 ⟶ need protocol contracts

Blame works (mostly)

blame shifting works

43466

blame doesn't settle on bug

276 ⟶ need protocol contracts

blame doesn't always move closer to bug

3

Blame works (mostly)

blame shifting works

43466

blame doesn't settle on bug

276 ⟶ need protocol contracts

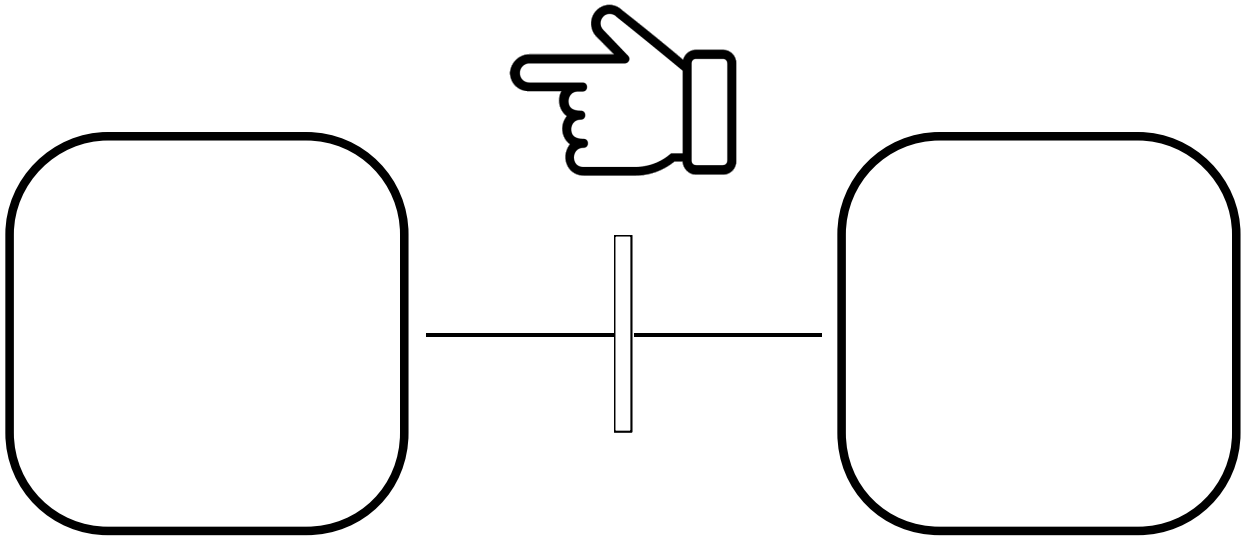
blame doesn't always move closer to bug

3 ⟶ need to tame violations during contract checking

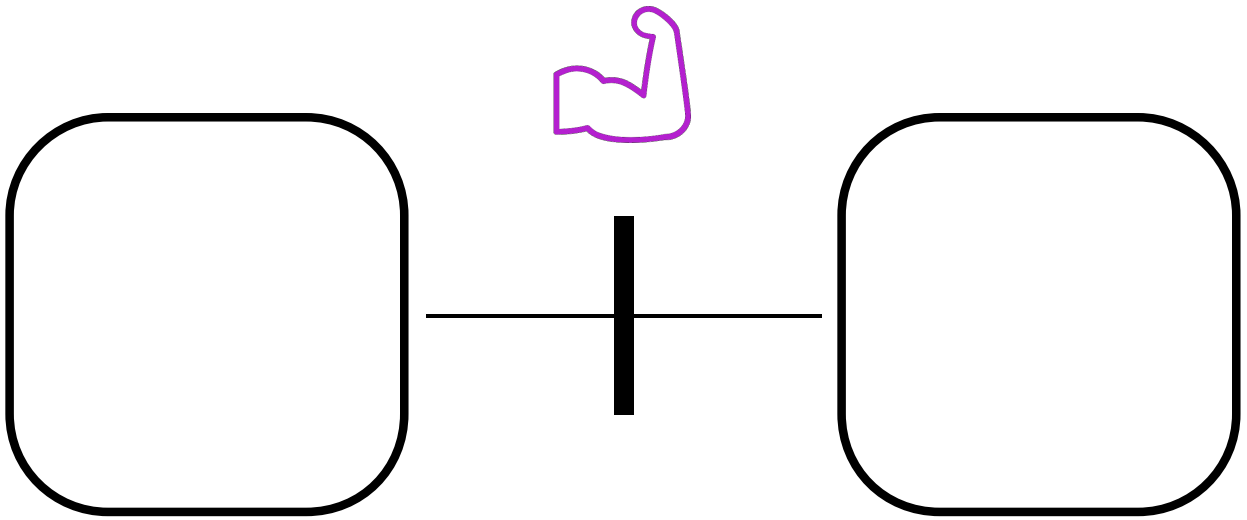
state in contracts

state in contracts ⟶ blame shifts

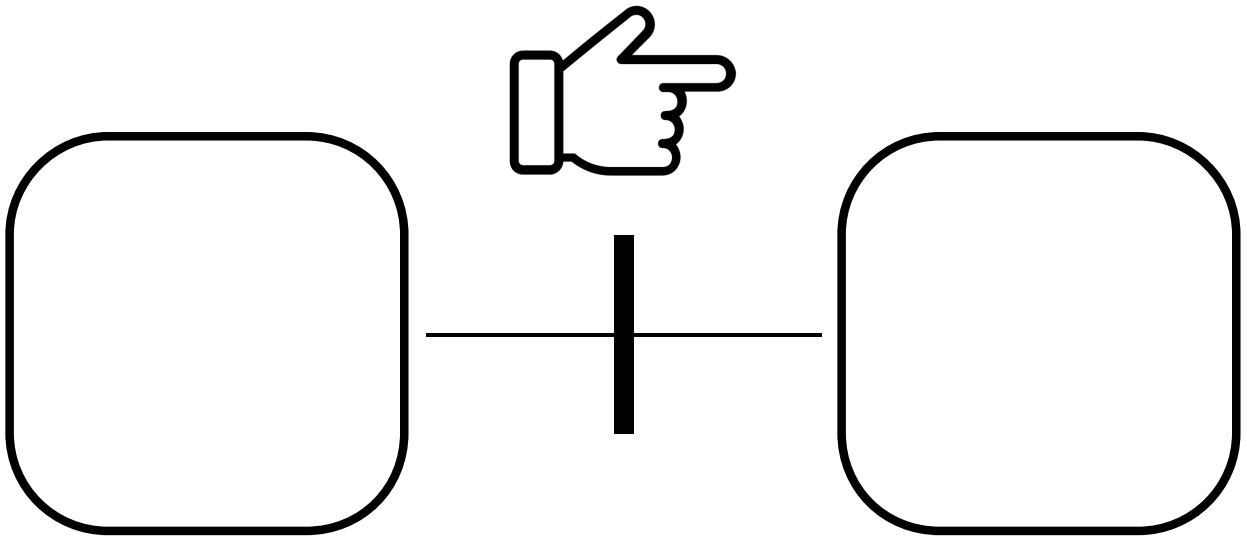
Unexpected blame shifting?



Unexpected blame shifting?



Unexpected blame shifting?



Adding an innocuous contract

```
#lang racket

(require require-typed-check
         racket/set
         "structs.rkt"
         "benv.rkt"
         "denotable.rkt"
         "time.rkt"
         (only-in racket/string string-join)
         "ai.rkt")

(define (summarize states)
  (for/fold ([store empty-store])
    ([state (in-set states)])
    (store-join (State-store state) store)))

(define (f v)



  (for/fold ([mono-store empty-mono-store])
    ([(b vs) (in-hash store)])
    (hash-update mono-store
                 (Binding-var b)
                 (update-lam vs)
                 default-lam)))

(define (analyze exp)
  (define init-state (State exp empty-benv empty-store time-zero))
  (define states (explore (set)
                          (list init-state)))
  mono-store)



(for ([x (in-set a-set)])
  (displayln x))
```

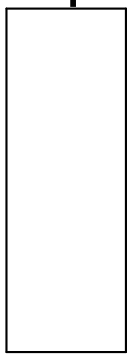
output

```
1
2
3
4
```

Adding an innocuous contract

```
#lang racket

(require require-typed-check
         racket/set
         "structs.rkt"
         "benv.rkt"
         "denotable.rkt"
         "time.rkt"
         (only-in racket/string string-join)
         "ai.rkt")

(define (summarize states)
  (for/fold ([store empty-store])
    ([state (in-set states)])
    (store-join (State-store state) store)))

(define/contract (f v)
  (-> (λ (x) (set-member? ok-vs x))
      number?)
  (for/fold ([mono-store empty-mono-store])
    ([(b vs) (in-hash store)])
    (hash-update mono-store
                 (Binding-var b)
                 (update-lam vs)
                 default-lam)))

(define (analyze exp)
  (define init-state (State exp empty-benv empty-store time-zero))
  (define states (explore (set)
                          (list init-state)))
  mono-store)



(for ([x (in-set a-set)])
  (displayln x))
```

output



Adding an innocuous contract

```
#lang racket

(require require-typed-check
         racket/set
         "structs.rkt"
         "benv.rkt"
         "denotable.rkt"
         "time.rkt"
         (only-in racket/string string-join)
         "ai.rkt")

(define (summarize states)
  (for/fold ([store empty-store])
    ([state (in-set states)])
    (store-join (State-store state) store)))

(define/contract (f v)
  (-> (λ (x) (set-member? ok-vs x))
      number?)
  (for/fold ([mono-store empty-mono-store])
    ([(b vs) (in-hash store)])
    (hash-update mono-store
                 (Binding-var b)
                 (update-lam vs)
                 default-lam)))

(define (analyze exp)
  (define init-state (State exp empty-benv empty-store time-zero))
  (define states (explore (set)
                          (list init-state)))
  mono-store)



(for ([x (in-set a-set)])
  (displayln x))
```

output

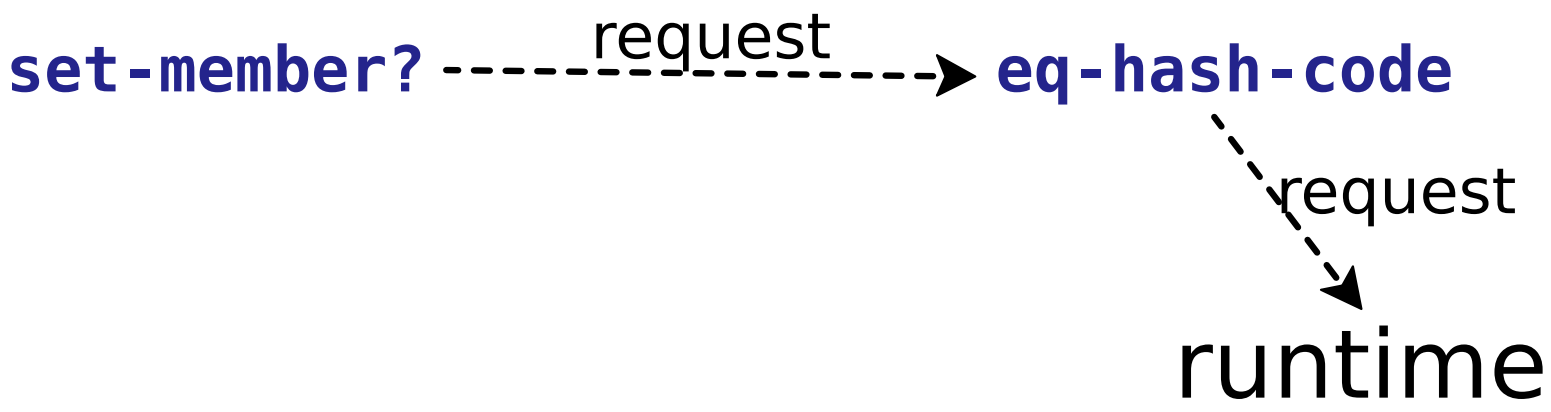
```
2
4
3
1
```

The runtime is working against you

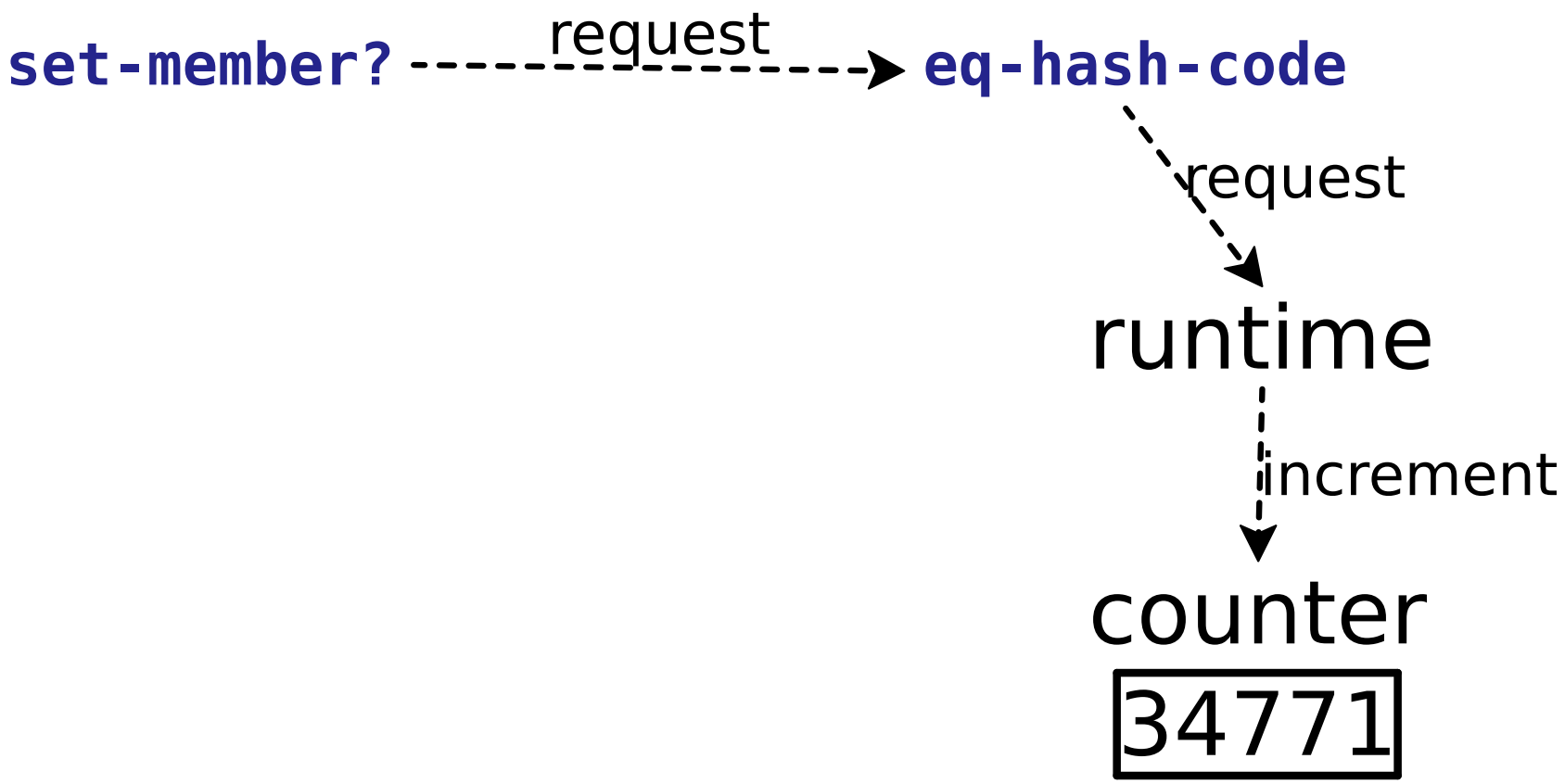
```
set-member?
```

The runtime is working against you

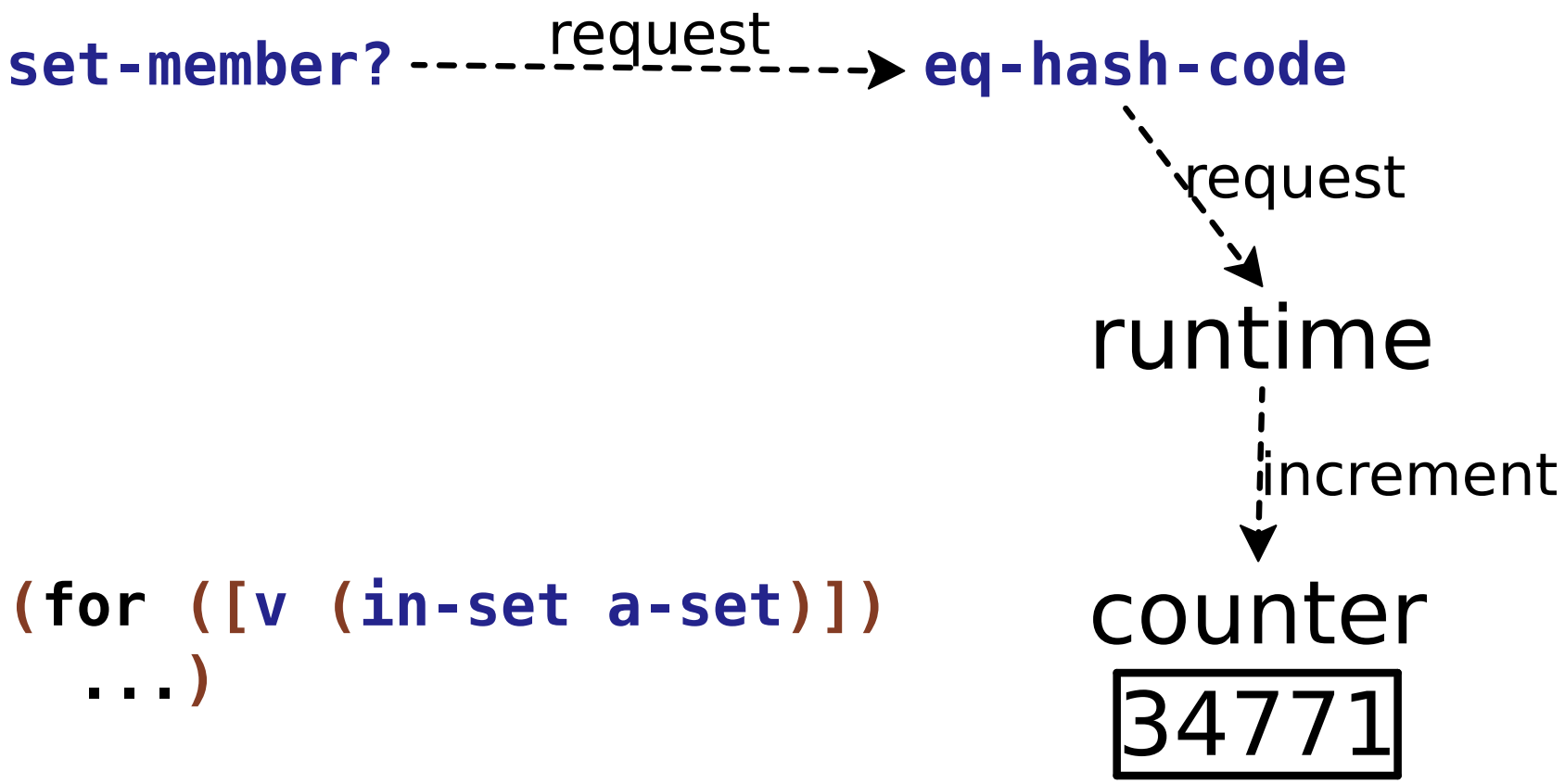
The runtime is working against you



The runtime is working against you

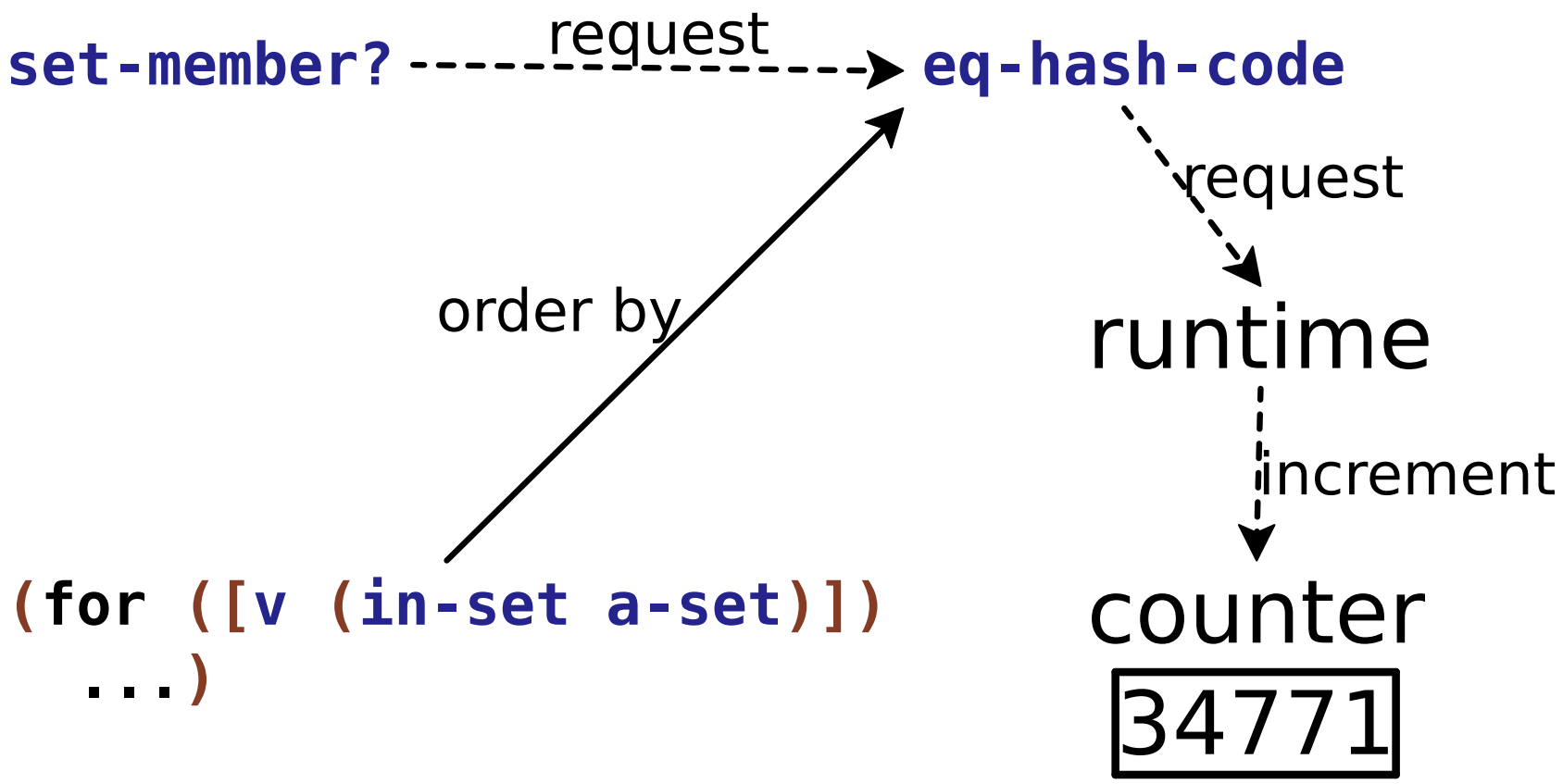


The runtime is working against you

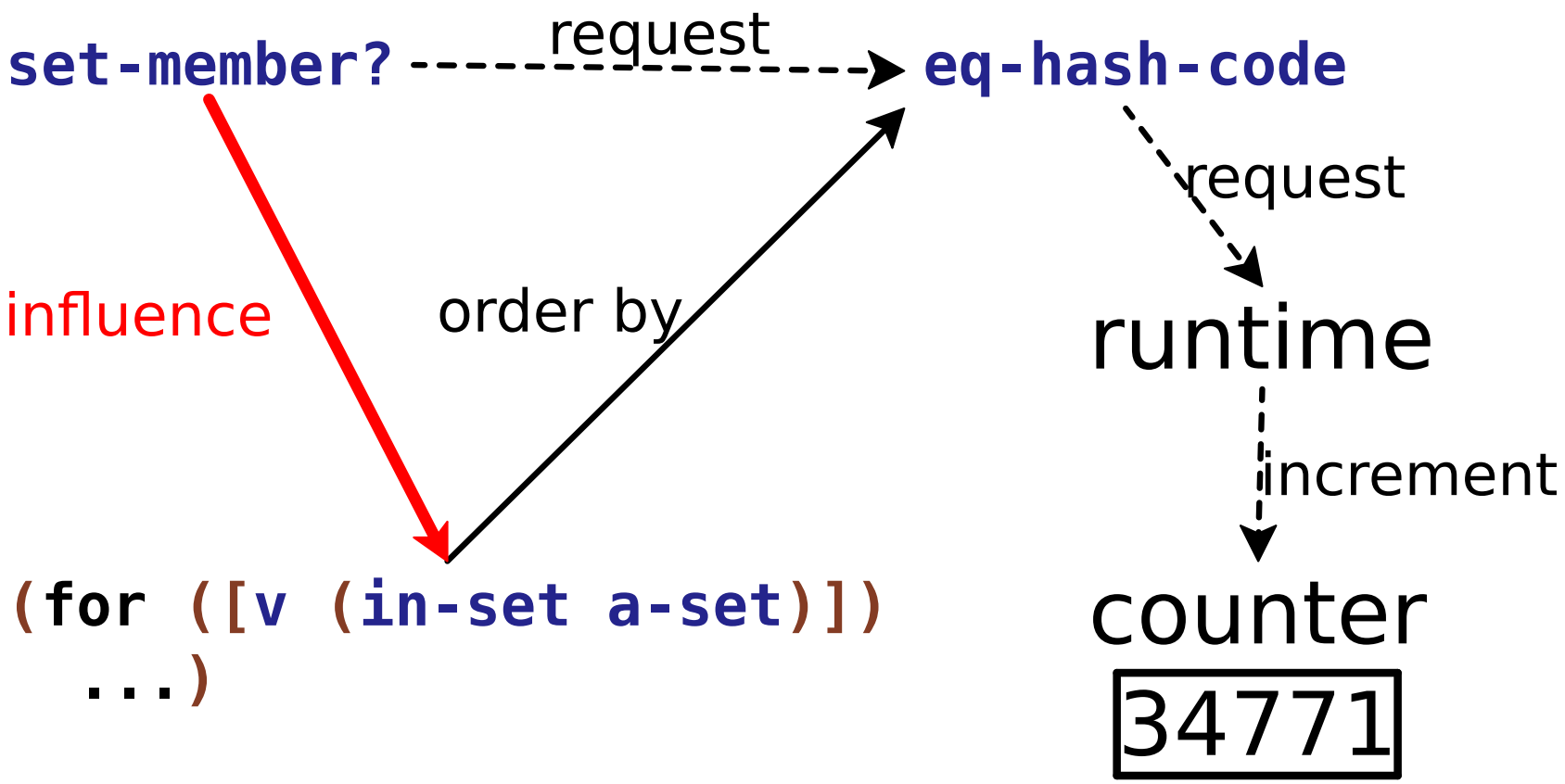



```
(for ([v (in-set a-set)])
  ...)
```

The runtime is working against you



The runtime is working against you



This can affect blame

```
#lang racket

(require require-typed-check
         racket/set
         "structs.rkt"
         "benv.rkt"
         "denotable.rkt"
         "time.rkt"
         (only-in racket/string string-join)
         "ai.rkt")

(define (summarize states)
  (for/fold ([store empty-store])
    ([state (in-set states)])
    (store-join (State-store state) store)))

(define/contract (f v)
  (-> (λ (x) (set-member? ok-vs x))
      number?)
  (for/fold ([mono-store empty-mono-store])
    ([(b vs) (in-hash store)])
    (hash-update mono-store
                 (Binding-var b)
                 (update-lam vs)
                 default-lam)))

(define (analyze exp)
  (define init-state (State exp empty-benv empty-store time-zero))
  (define states (explore (set)
                          (list init-state)))
  mono-store)



(for ([x (in-set a-set)])
  (displayln x))
```

This can affect blame

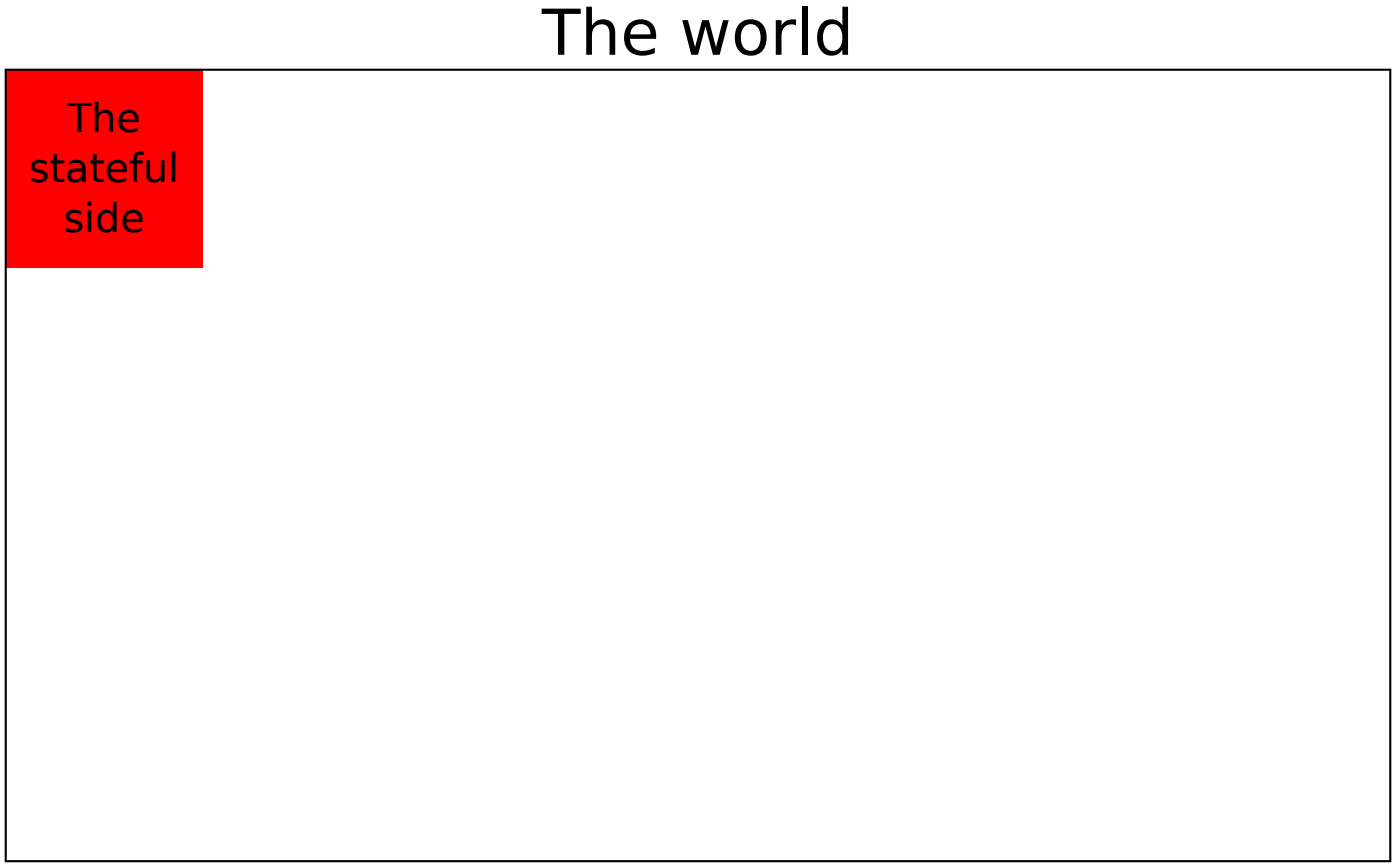
```
#lang racket

(require require-typed-check
         racket/set
         "structs.rkt"
         "benv.rkt"
         "denotable.rkt"
         "time.rkt"
         (only-in racket/string string-join)
         "ai.rkt")

(define (summarize states)
  (for/fold ([store empty-store])
    ([state (in-set states)])
    (store-join (State-store state) store)))

(define/contract (f v)
  (-> (λ (x) (set-member? ok-vs x))
      number?)
  (for/fold ([mono-store empty-mono-store])
    ([(b vs) (in-hash store)])
    (hash-update mono-store
                 (Binding-var b)
                 (update-lam vs)
                 default-lam)))

(define (analyze exp)
  (define init-state (State exp empty-benv empty-store time-zero))
  (define states (explore (set)
                          (list init-state)))
  mono-store)



(for ([a-function (in-set a-set)])
  (a-function))
```

This can affect blame

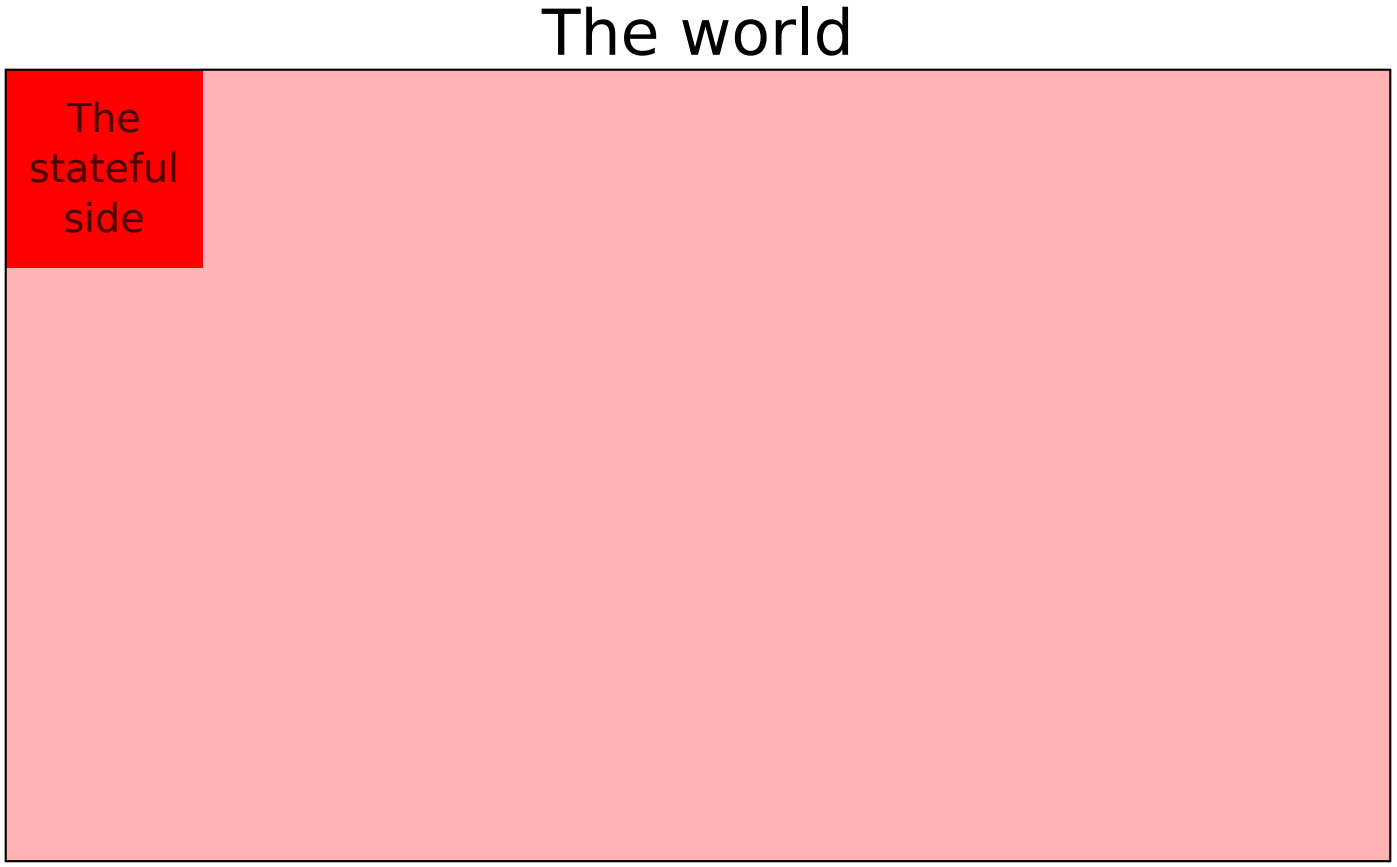
KCFA

```
#lang racket

(require require-typed-check
         racket/set
         "structs.rkt"
         "benv.rkt"
         "denotable.rkt"
         "time.rkt"
         (only-in racket/string string-join)
         "ai.rkt")

(define (summarize states)
  (for/fold ([store empty-store])
    ([state (in-set states)])
    (store-join (State-store state) store)))

(define/contract (f v)
  (-> (λ (x) (set-member? ok-vs x))
      number?)
  (for/fold ([mono-store empty-mono-store])
    ([(b vs) (in-hash store)])
    (hash-update mono-store
                 (Binding-var b)
                 (update-lam vs)
                 default-lam)))

(define (analyze exp)
  (define init-state (State exp empty-benv empty-store time-zero))
  (define states (explore (set)
                          (list init-state)))
  mono-store)



(for ([a-function (in-set a-set)])
  (a-function))
```

We knew state is a problem



But state is pervasive



Closing the loop

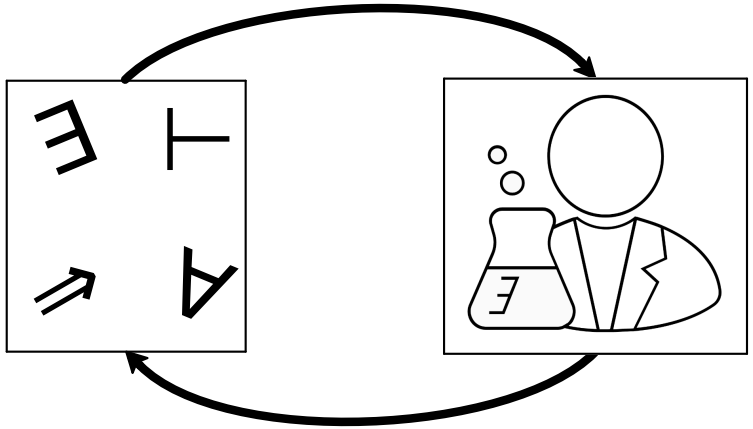
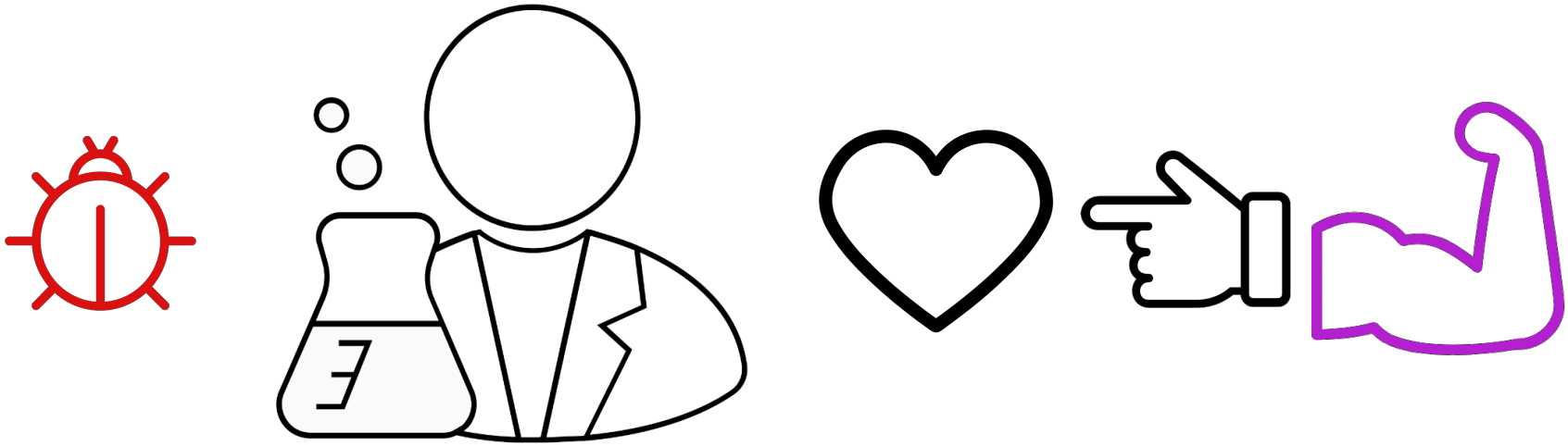


Image credits



The Noun Project: alvianwijaya, ID; parkjisun; BomSymbols, TH; Vectors Point, PK; Vectors Market