Homework 3: rec

`rec` should evaluate its arguments immediately

- Why?

Homework 3: rec

`rec` should evaluate its arguments immediately

- Why?

To match existing programming languages; otherwise it is not call-by-value

Homework 3: rec

`rec` should evaluate its arguments immediately

- Why?

To match existing programming languages; otherwise it is not call-by-value

- Can you tell the difference? (recall the "are these two programs equivalent?" game)

Homework 3: rec

`rec` should evaluate its arguments immediately

- Why?

To match existing programming languages; otherwise it is not call-by-value

- Can you tell the difference? (recall the "are these two programs equivalent?" game)

Nontermination and errors both show the difference in F1WAE; if we had other effects (e.g., exceptions, mutable state) they too would also show the difference

Cost of Substitution

```
(interp {with {x 1}
          {with {y 2}
            {+ 100 {+ 99 {+ 98 ... {+ y x}}}}}})
```

Cost of Substitution

```
(interp {with {x 1}
          {with {y 2}
            {+ 100 {+ 99 {+ 98 ... {+ y x}}}}}})
```

⇒

```
(interp {with {y 2}
          {+ 100 {+ 99 {+ 98 ... {+ y 1}}}}})
```

Cost of Substitution

```
(interp {with {x 1}
          {with {y 2}
            {+ 100 {+ 99 {+ 98 ... {+ y x}}}}}})
```

⇒

```
(interp {with {y 2}
          {+ 100 {+ 99 {+ 98 ... {+ y 1}}}}})
```

⇒

```
(interp {+ 100 {+ 99 {+ 98 ... {+ 2 1}}}})
```

Cost of Substitution

```
(interp {with {x 1}
          {with {y 2}
            {+ 100 {+ 99 {+ 98 ... {+ y x}}}}}})
```

⇒

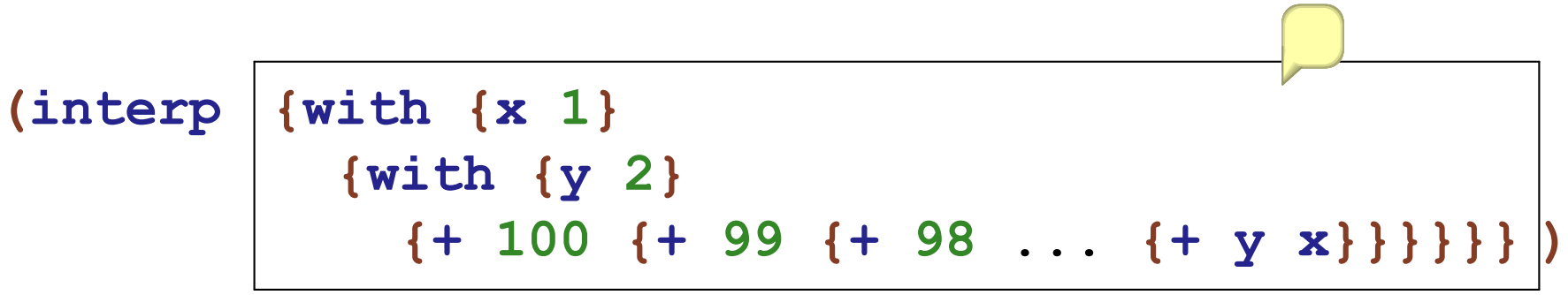
```
(interp {with {y 2}
          {+ 100 {+ 99 {+ 98 ... {+ y 1}}}}})
```

⇒

```
(interp {+ 100 {+ 99 {+ 98 ... {+ 2 1}}}})
```

With **n** variables, evaluation will take $O(\mathbf{n}^2)$ time!

Deferring Substitution



Deferring Substitution

```
(interp {with {x 1}
          {with {y 2}
            {+ 100 {+ 99 {+ 98 ... {+ y x}}}}}})
```

⇒

x = 1

```
(interp {with {y 2}
          {+ 100 {+ 99 {+ 98 ... {+ y x}}}}})
```

Deferring Substitution

```
(interp {with {x 1}
          {with {y 2}
            {+ 100 {+ 99 {+ 98 ... {+ y x}}}}}})
```

⇒

```
(interp {with {y 2}
          {+ 100 {+ 99 {+ 98 ... {+ y x}}}}})
```

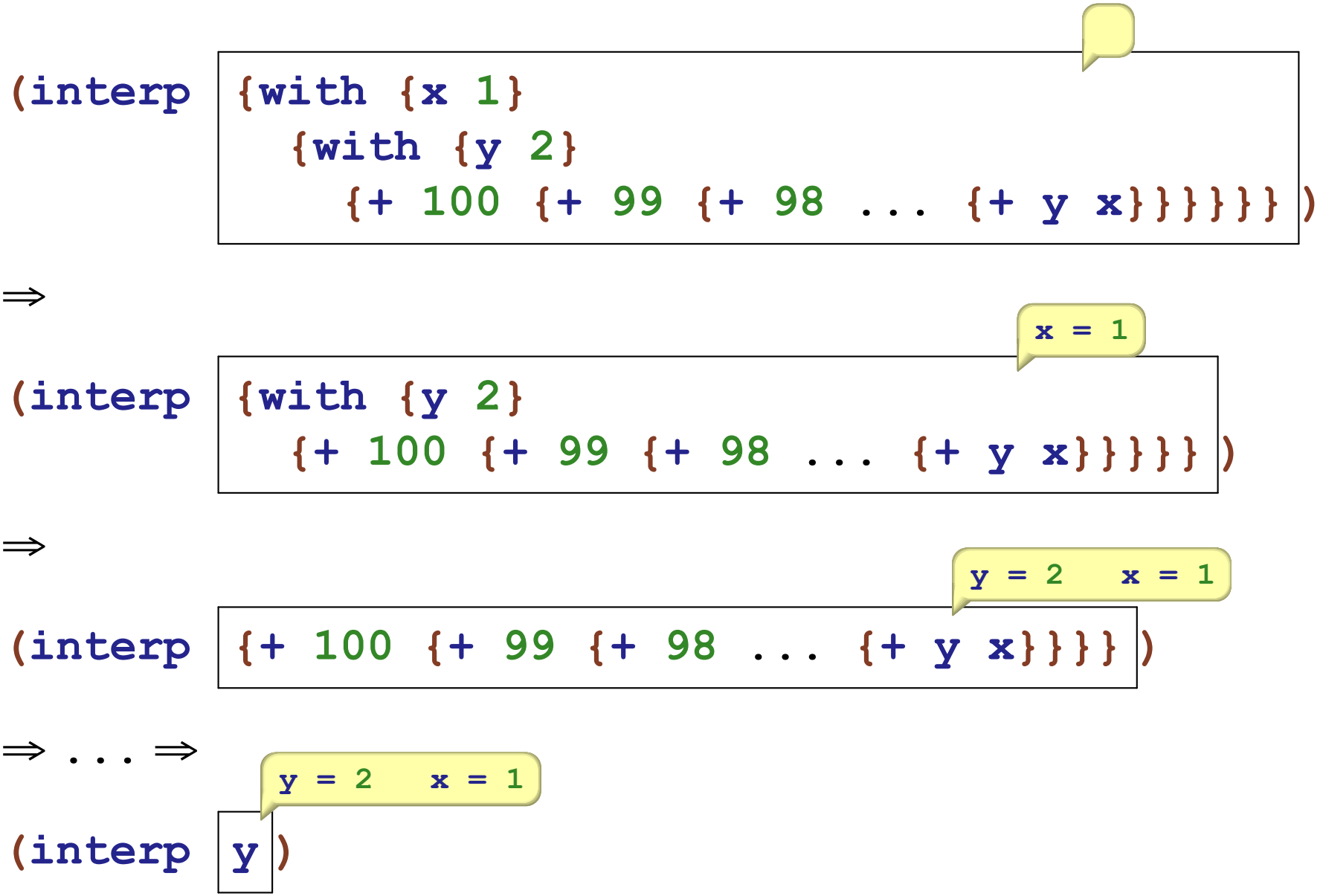
x = 1

⇒

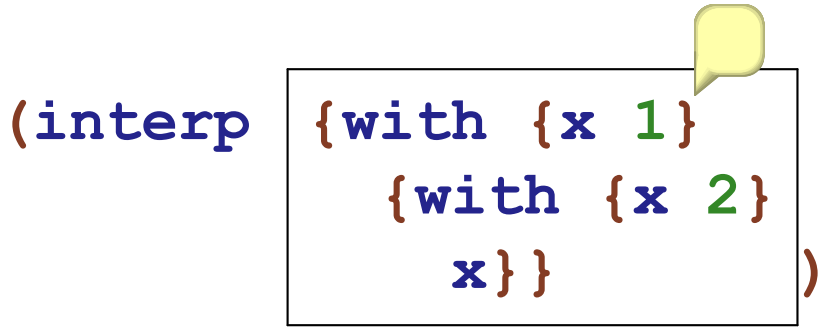
```
(interp {+ 100 {+ 99 {+ 98 ... {+ y x}}}})
```

y = 2 x = 1

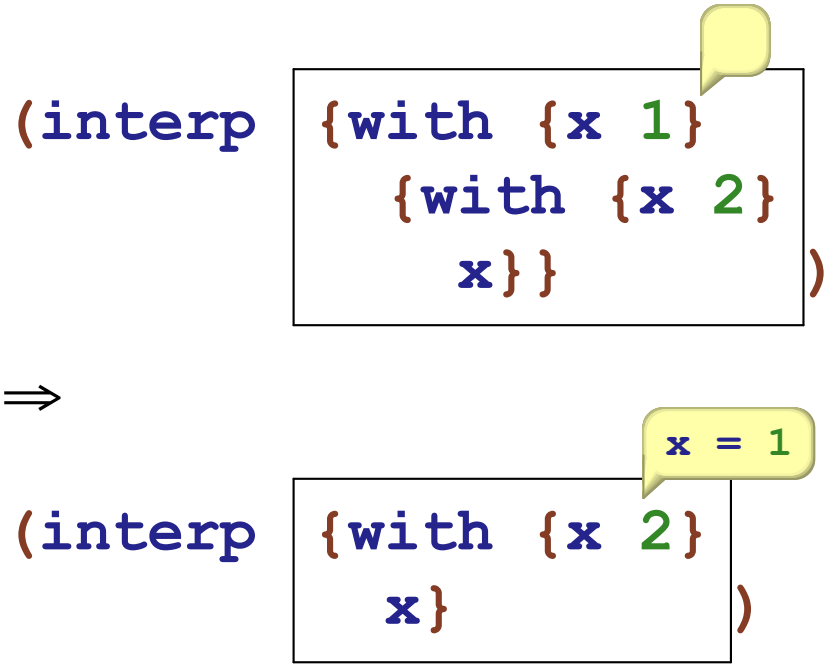
Deferring Substitution



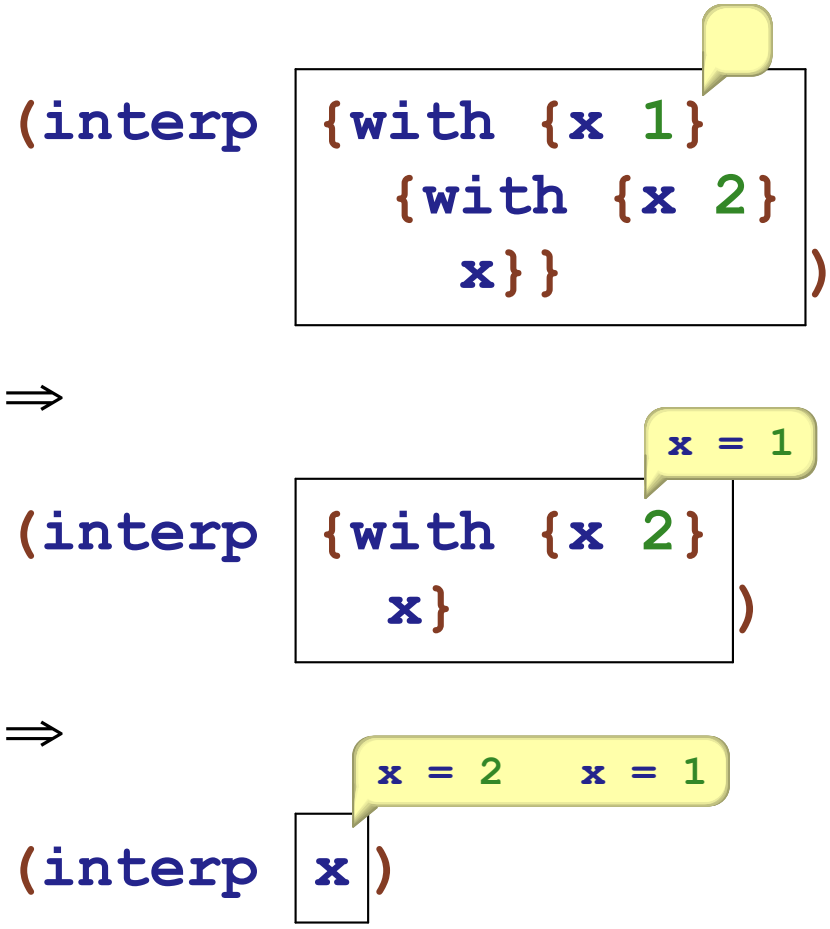
Deferring Substitution with the Same Identifier



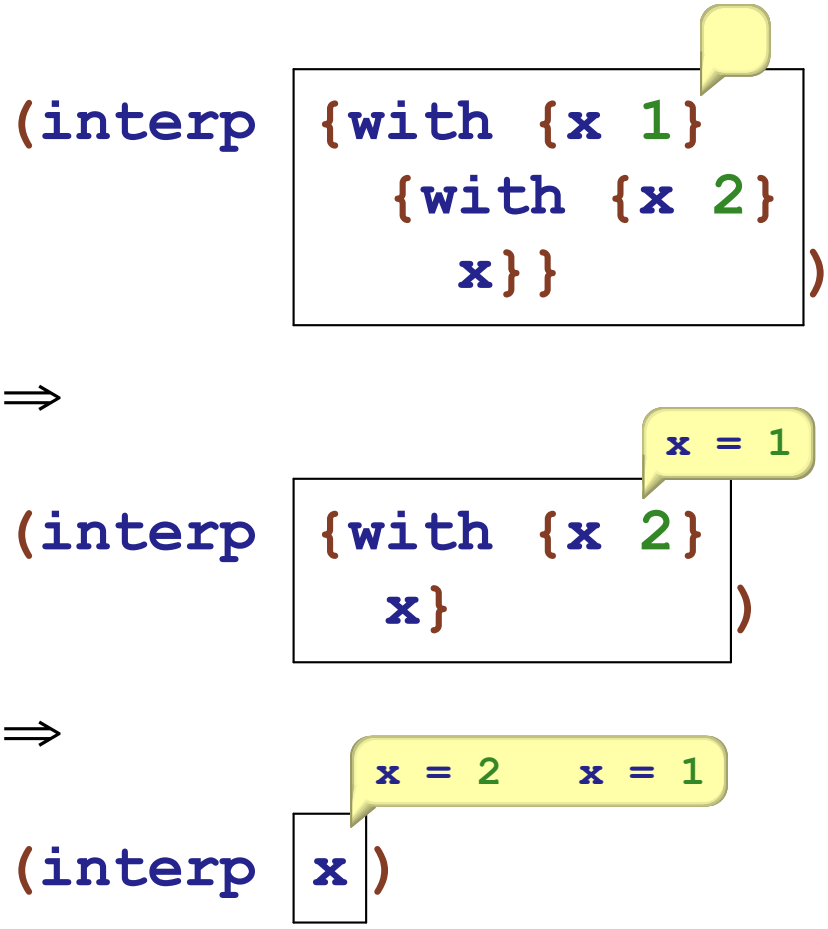
Deferring Substitution with the Same Identifier



Deferring Substitution with the Same Identifier

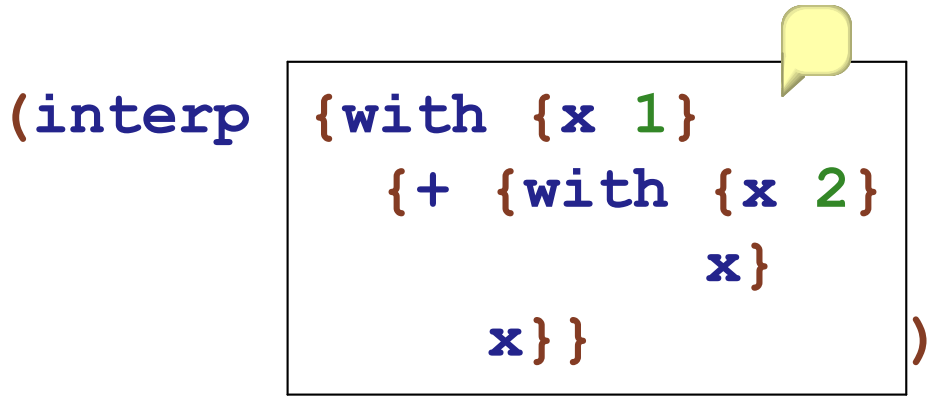


Deferring Substitution with the Same Identifier



Always add to start, then always check from start

Deferring Substitution with the Same Identifier



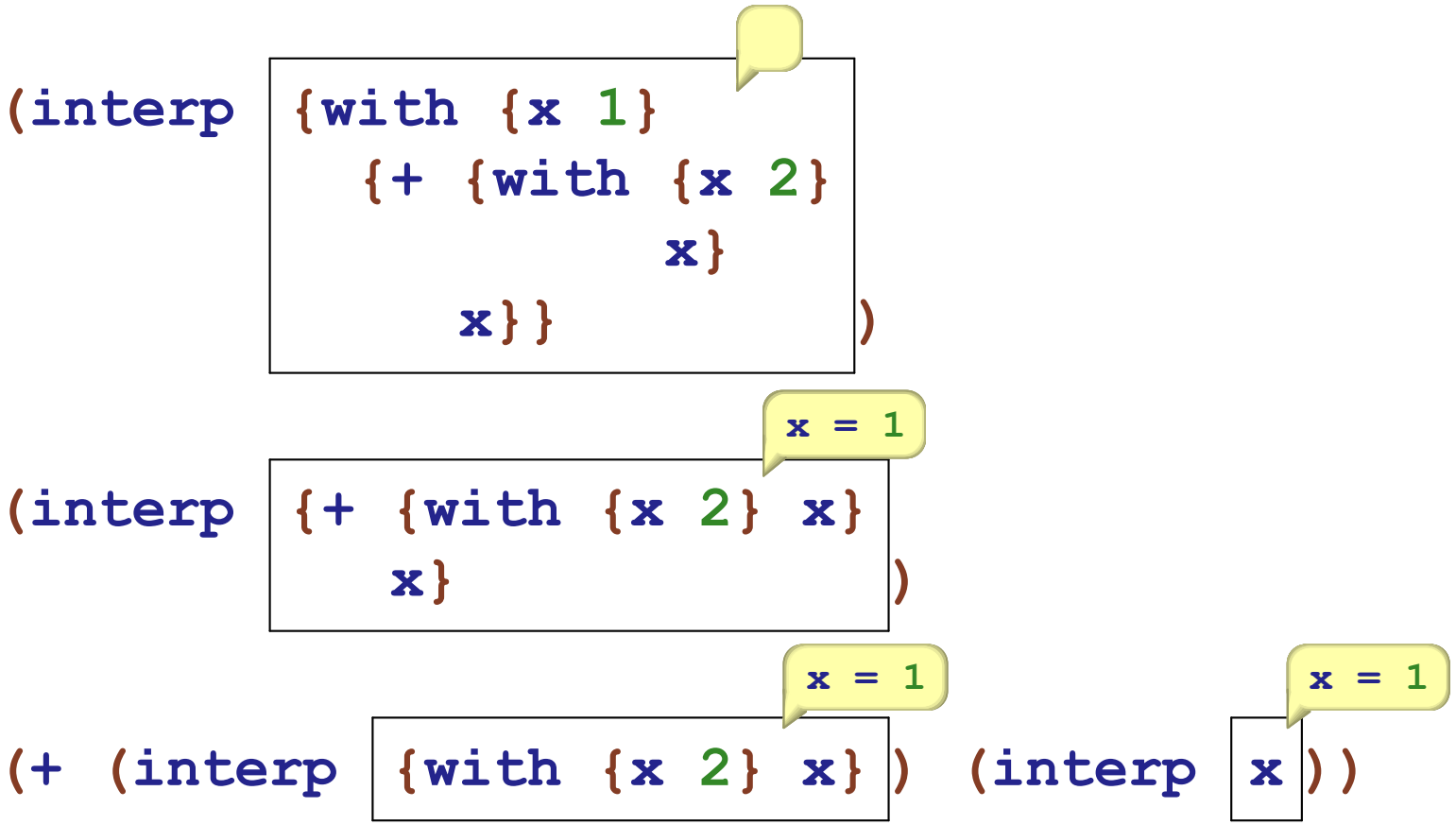
Deferring Substitution with the Same Identifier

```
(interp {with {x 1}
          {+ {with {x 2}
               x}
             x}})
```

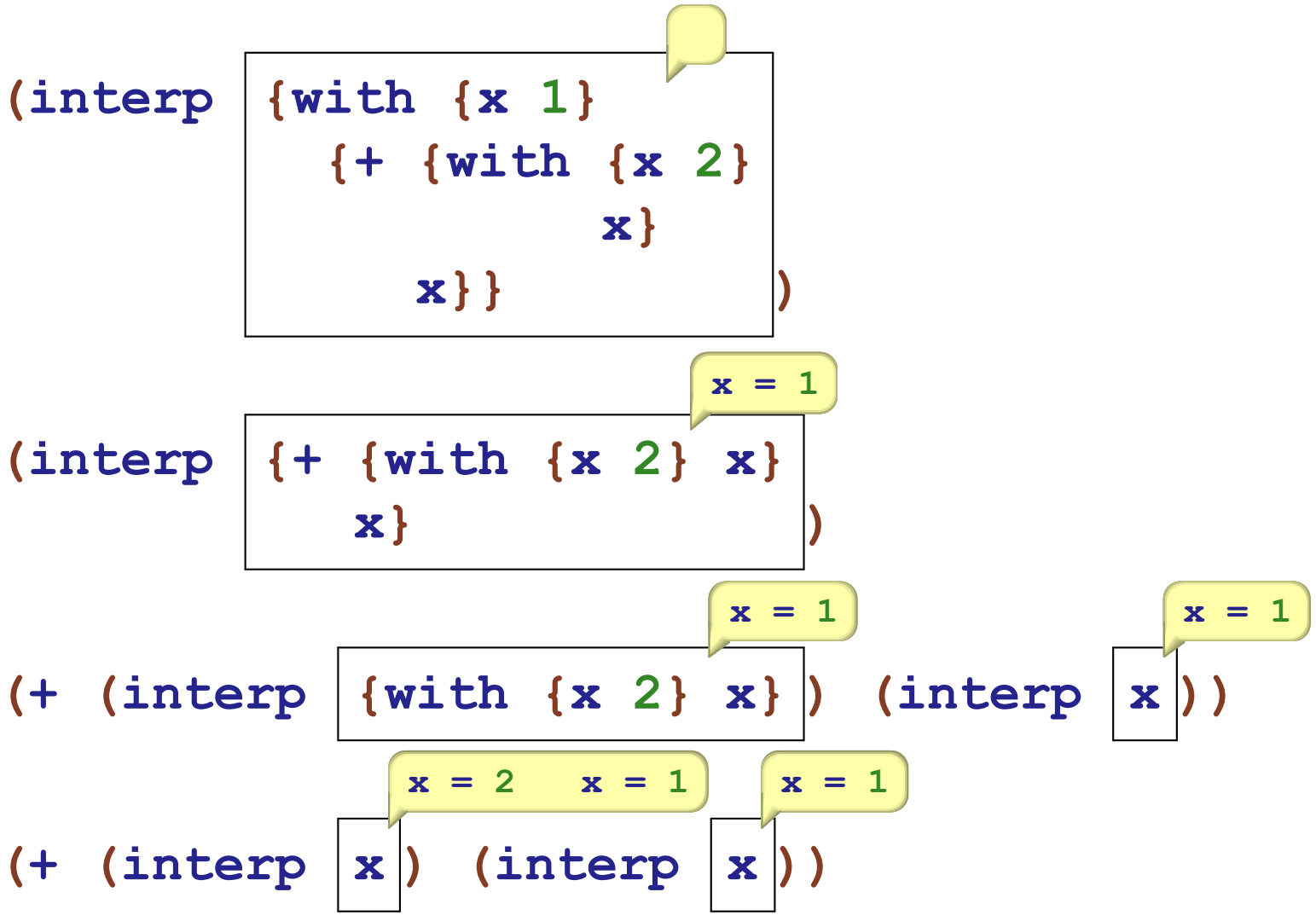
```
(interp {+ {with {x 2} x}
           x})
```

x = 1

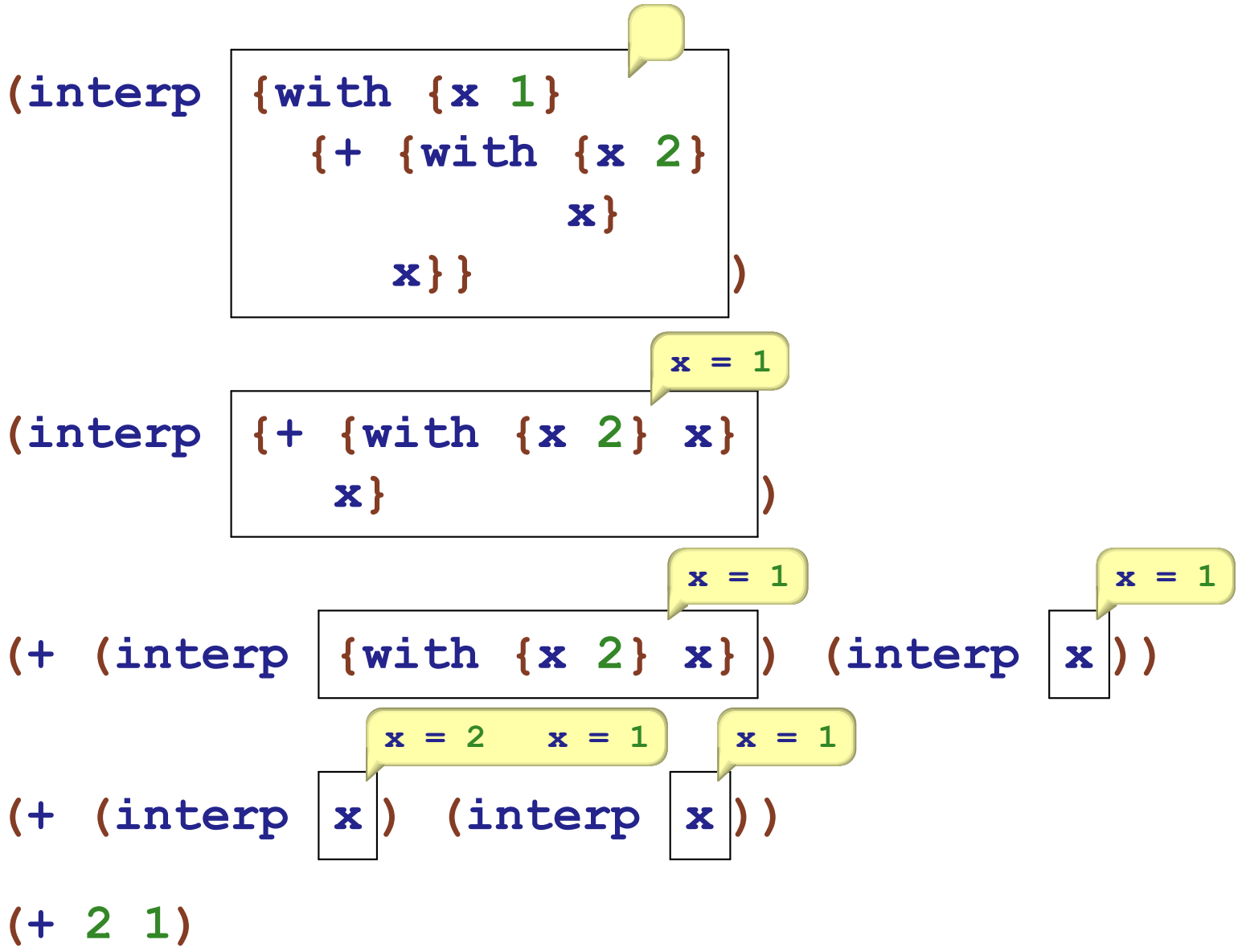
Deferring Substitution with the Same Identifier



Deferring Substitution with the Same Identifier



Deferring Substitution with the Same Identifier



Representing Deferred Substitution

Change

```
; interp : WAE -> num
```

to

```
; interp : WAE DefrdSub -> num
```

Representing Deferred Substitution

Change

```
; interp : WAE -> num
```

to

```
; interp : WAE DefrdSub -> num
```

```
(define-type DefrdSub
  [mtSub]
  [aSub (name symbol?)
        (value number?)
        (rest DefrdSub?)])
```

Interp with DefrdSub

```
(interp {with {x 1}
          {with {y 2}
            {+ 100 {+ 99 {+ 98 ... {+ y x}}}}}}
        (mtSub))
```

Interp with DefrdSub

```
(interp {with {x 1}
          {with {y 2}
            {+ 100 {+ 99 {+ 98 ... {+ y x}}}}}}
        (mtSub))
```

```
⇒ (interp {with {y 2}
            {+ 100 {+ 99 {+ 98 ... {+ y x}}}}}
          (aSub 'x 1 (mtSub)))
```

Interp with DefrdSub

```
(interp {with {x 1}
          {with {y 2}
            {+ 100 {+ 99 {+ 98 ... {+ y x}}}}}}
        (mtSub))
```

```
⇒ (interp {with {y 2}
            {+ 100 {+ 99 {+ 98 ... {+ y x}}}}}
          (aSub 'x 1 (mtSub)))
```

```
⇒ (interp {+ 100 {+ 99 {+ 98 ... {+ y x}}}}
          (aSub 'y 2 (aSub 'x 1 (mtSub))))
```

Interp with DefrdSub

```
(interp {with {x 1}
          {with {y 2}
            {+ 100 {+ 99 {+ 98 ... {+ y x}}}}}}
        (mtSub))
```

```
⇒ (interp {with {y 2}
            {+ 100 {+ 99 {+ 98 ... {+ y x}}}}}
          (aSub 'x 1 (mtSub)))
```

```
⇒ (interp {+ 100 {+ 99 {+ 98 ... {+ y x}}}}
          (aSub 'y 2 (aSub 'x 1 (mtSub))))
```

```
⇒ ...
```

```
⇒ (interp y (aSub 'y 2 (aSub 'x 1 (mtSub))))
```

WAE Interpreter with Deferred Substitutions

```
; interp : WAE DefrdSub -> num
(define (interp a-wae ds)
  (type-case WAE a-wae
    [num (n) n]
    [add (l r) (+ (interp l ds) (interp r ds))]
    [sub (l r) (- (interp l ds) (interp r ds))]
    [with (bound-id named-expr body-expr)
      ...]
    [id (name) ...]))
```

WAE Interpreter with Deferred Substitutions

```
; interp : WAE DefrdSub -> num
(define (interp a-wae ds)
  (type-case WAE a-wae
    [num (n) n]
    [add (l r) (+ (interp l ds) (interp r ds))]
    [sub (l r) (- (interp l ds) (interp r ds))]
    [with (bound-id named-expr body-expr)
      ...]
    [id (name) (lookup name ds)]))
```

WAE Interpreter with Deferred Substitutions

```
; lookup : symbol DefrdSub -> num
(define (lookup name ds)
  (type-case DefrdSub ds
    [mtSub () (error 'lookup "free variable")]
    [aSub (sub-name num rest-ds)
          (if (symbol=? sub-name name)
              num
              (lookup name rest-ds))]))
```

WAE Interpreter with Deferred Substitutions

```
; interp : WAE DefrdSub -> num
(define (interp a-wae ds)
  (type-case WAE a-wae
    [num (n) n]
    [add (l r) (+ (interp l ds) (interp r ds))]
    [sub (l r) (- (interp l ds) (interp r ds))]
    [with (bound-id named-expr body-expr)
      ...]
    [id (name) (lookup name ds)]))
```

WAE Interpreter with Deferred Substitutions

```
; interp : WAE DefrdSub -> num
(define (interp a-wae ds)
  (type-case WAE a-wae
    [num (n) n]
    [add (l r) (+ (interp l ds) (interp r ds))]
    [sub (l r) (- (interp l ds) (interp r ds))]
    [with (bound-id named-expr body-expr)
      ... (interp named-expr ds) ...]
    [id (name) (lookup name ds)]))
```

WAE Interpreter with Deferred Substitutions

```
; interp : WAE DefrdSub -> num
(define (interp a-wae ds)
  (type-case WAE a-wae
    [num (n) n]
    [add (l r) (+ (interp l ds) (interp r ds))]
    [sub (l r) (- (interp l ds) (interp r ds))]
    [with (bound-id named-expr body-expr)
      ...
      (aSub bound-id (interp named-expr ds) ds)
      ...]
    [id (name) (lookup name ds)]))
```

WAE Interpreter with Deferred Substitutions

```
; interp : WAE DefrdSub -> num
(define (interp a-wae ds)
  (type-case WAE a-wae
    [num (n) n]
    [add (l r) (+ (interp l ds) (interp r ds))]
    [sub (l r) (- (interp l ds) (interp r ds))]
    [with (bound-id named-expr body-expr)
      (interp
       body-expr
       (aSub bound-id (interp named-expr ds) ds))]
    [id (name) (lookup name ds)]))
```

Function Calls

```
{deffun {f x} {+ 1 x}}

(interp {with {y 2}
          {f 10}})
```

Function Calls

```
{deffun {f x} {+ 1 x}}

(interp {with {y 2}
          {f 10}})
```

⇒

```
(interp {f 10})
```

y = 2

Function Calls

```
{deffun {f x} {+ 1 x}}

(interp {with {y 2}
          {f 10}})
```

⇒

```
(interp {f 10})
```

y = 2

⇒

```
(interp {+ 1 x})
```

...

Function Calls

```
{deffun {f x} {+ 1 x}}

(interp {with {y 2}
          {f 10}})
```

⇒

```
(interp {f 10})
```

y = 2

⇒

```
(interp {+ 1 x})
```

x = 10

Interpreting function body starts with only one substitution

Function Calls

What goes wrong if you extend the old substitution?

```
{deffun {f x} {+ y x}}

(interp {with {y 2}
          {f 10}})
```

Function Calls

What goes wrong if you extend the old substitution?

```
{deffun {f x} {+ y x}}

(interp {with {y 2}
          {f 10}})
```

⇒

```
(interp {f 10})
```

y = 2

Function Calls

What goes wrong if you extend the old substitution?

```
{deffun {f x} {+ y x}}

(interp {with {y 2}
          {f 10}})
```

⇒

```
(interp {f 10})
```

y = 2

⇒

```
(interp {+ y x})
```

x = 10 y = 2

⇒ 12 wrong!

Function Calls

What goes wrong if you extend the old substitution?

```
{deffun {f x} {+ y x}}

(interp {with {y 2}
          {f 10}})

⇒

(interp {f 10})        ; y = 2

⇒

(interp {+ y x})       ; x = 10

⇒ "free var: y"
```


Interpreting function body starts with only one

F1WAE Interpreter with Deferred Substitutions

```
; interp : F1WAE list-of-FunDef DefrdSub -> num
(define (interp a-f1wae fundefs ds)
  (type-case F1WAE a-f1wae
    ...
    [app (name arg-expr)
         ...]))
```

F1WAE Interpreter with Deferred Substitutions

```
; interp : F1WAE list-of-FunDef DefrdSub -> num
(define (interp a-f1wae fundefs ds)
  (type-case F1WAE a-f1wae
    ...
    [app (name arg-expr)
         (local [(define a-fundef
                   (lookup-fundef name fundefs))]
           (interp (fundef-body a-fundef)
                   fundefs
                   ...
                   (interp arg-expr fundefs ds)
                   ...))]))
```

F1WAE Interpreter with Deferred Substitutions

```
; interp : F1WAE list-of-FunDef DefrdSub -> num
(define (interp a-f1wae fundefs ds)
  (type-case F1WAE a-f1wae
    ...
    [app (name arg-expr)
         (local [(define a-fundef
                   (lookup-fundef name fundefs))]
           (interp (fundef-body a-fundef)
                   fundefs
                   (aSub (fundef-arg-name a-fundef)
                         (interp arg-expr fundefs ds)
                         (mtSub))))]))
```

Timing tests

```
(define (mk-sums n)
  (cond
    [(zero? n) 1]
    [else
     (let ([varn (string->symbol (format "x~a" n))])
       `{+ ,varn ,(mk-sums (- n 1))})]))

(define (mk-withs n body)
  (cond
    [(zero? n) body]
    [else
     (let ([varn (string->symbol (format "x~a" n))])
       `{with {,varn 1}
              ,(mk-withs (- n 1) body)})]))
```

Timing tests, 2

```
(define (mk-exp n) (mk-withs n (mk-sums n)))

(test (mk-exp 2)
      `{with {x2 1}
             {with {x1 1}
                   {+ x2 {+ x1 1}}}})

(define (run n)
  (let ([expr (parse (mk-exp n))])
    (time (interp-expr expr '()))))
```

Timing tests, 3

With the substitution-based interpreter, expect the difference between adjacent timings to be growing linearly. With the environment-based one, you will also see linear growth, but if you make the environment use a more efficient data structure, that'll go away

(you may need to make the numbers bigger or smaller to see what is going on here)

```
(collect-garbage) (collect-garbage)
(collect-garbage) (collect-garbage)
(run 100) (run 110) (run 120)
(run 130) (run 140) (run 160)
```