Parsing

On slides,

```
(+ 1 2)
```

means

```
(make-plus 1 2)
```

Parsing

On slides,

means

```
'x
```

Parsing

On slides,

```
(lambda (x) (+ 1 x))
```

means

```
(make-lambda 'x (make-plus 1 'x))
```

Parsing

On slides,

```
((lambda (g) (g 10))
 (lambda (x) (+ 1 x)))
```

means

```
(make-app
 (make-lambda 'g (make-app 'g 10))
 (make-lambda 'x (make-plus 1 'x)))
```

Cost of Substitution

```
(evaluate ((lambda (x)
             ((lambda (y)
                (+ 100 (+ 99 (+ 98 ... (+ y x)))))
              2))
           1))
```

⇒

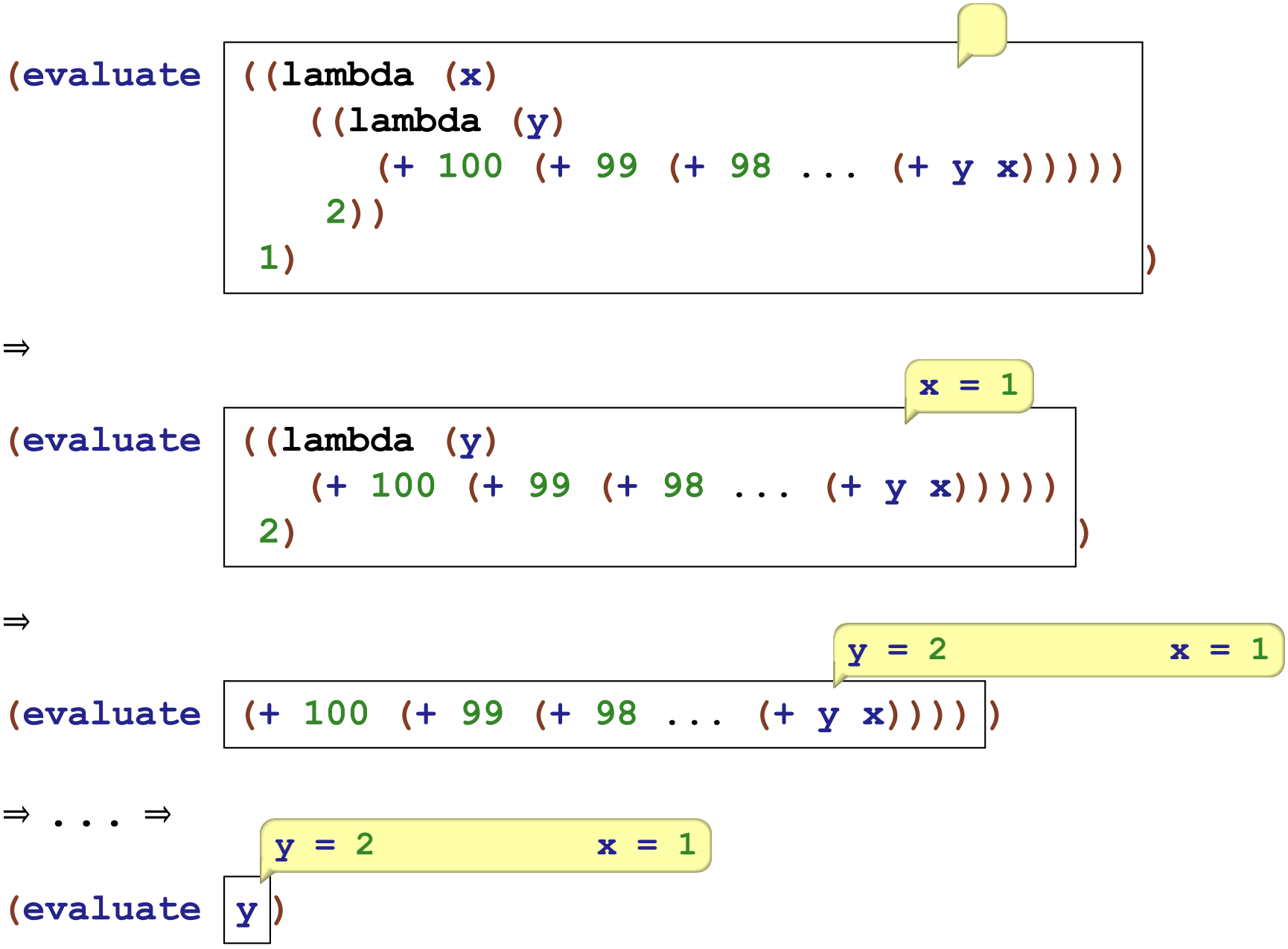
```
(evaluate ((lambda (y)
             (+ 100 (+ 99 (+ 98 ... (+ y 1)))))
           2))
```

⇒

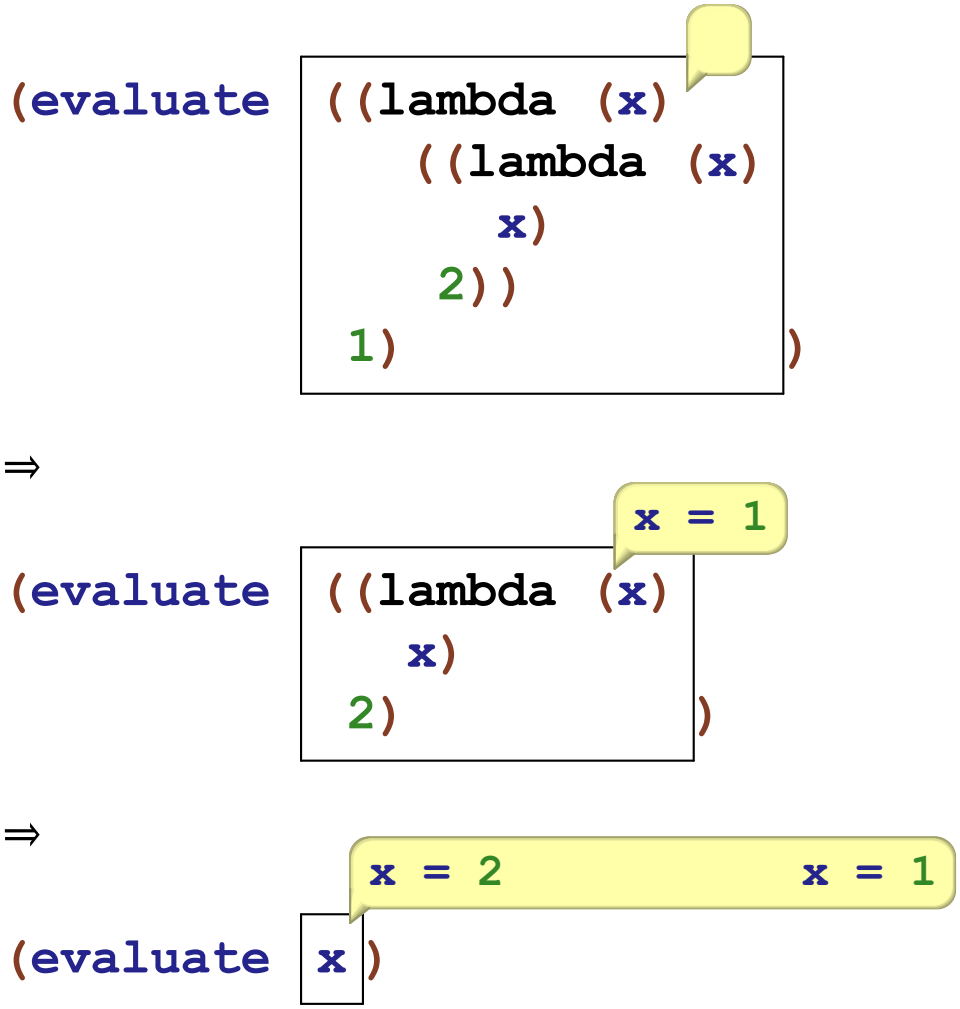
```
(evaluate (+ 100 (+ 99 (+ 98 ... (+ 2 1)))))
```

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

Deferring Substitution



Deferring Substitution with the Same Identifier



Always add to start, then always check from start

Environment

```
; An env is either
;   - empty
;   - (make-sub sym val env)
(define-struct sub (id val))
```

 = empty

y = 1 = (make-sub 'y 1 empty)

x = 2 y = 1 =

(make-sub 'x 2 (make-sub 'y 1 empty))

Evaluation with an Environment

```
(evaluate ((lambda (x)
             ((lambda (y)
                (+ 100 (+ 99 (+ 98 ... (+ y x)))))
              2))
           1)
          empty)
```

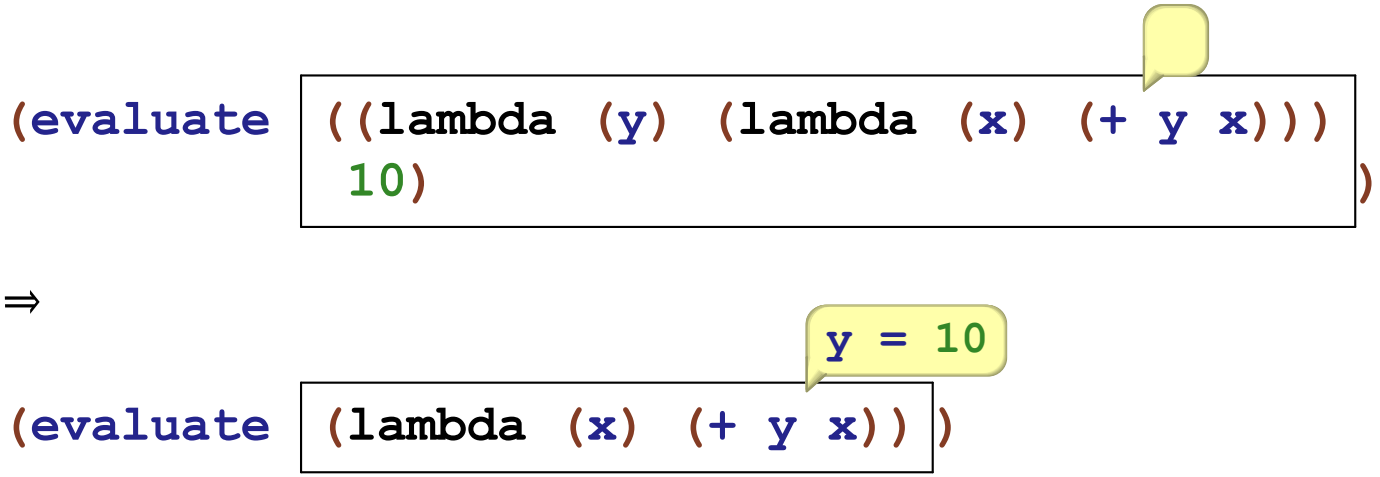
```
⇒ (evaluate ((lambda (y)
               (+ 100 (+ 99 (+ 98 ... (+ y x)))))
             2)
            (make-sub 'x 1 empty))
```

```
⇒ (evaluate (+ 100 (+ 99 (+ 98 ... (+ y x))))
            (make-sub 'y 2 (make-sub 'x 1 empty)))
```

```
⇒ ...
```

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

Functions and Environments



Function Calls with Environments

```
(evaluate (((lambda (y) (lambda (x) (+ y x))) 10)
          ((lambda (y) y) 7)))
```

Argument expression:

```
(evaluate ((lambda (y) y) 7))
```

⇒

y = 7

```
(evaluate y) ⇒ 7
```

Function expression:

```
(evaluate ((lambda (y) (lambda (x) (+ y x))) 10))
```

⇒

y = 10

```
(evaluate (lambda (x) (+ y x))) ⇒ ?
```

Functions as Values

A function value needs to keep its environment

```
; A function is
;   (make-function sym expr env)
(define-struct function (arg-name body env))
```

```
(test (evaluate ((lambda (y) (lambda (x) (+ y x))) 10)
                empty)
      (make-function 'x (+ y x)
                     (make-sub 'y 10 empty)))
```

Continuing Evaluation

Function: `(lambda (x) (+ y x))` — `y = 10`

Argument: `7`

To apply, interpret the function body with the given argument:

```
(evaluate (+ y x)
          (make-sub 'x 7
                    (make-sub 'y 10 empty)))
```