CS339: Abstractions and Paradigms for Programming

Environment Model

Manas Thakur
CSE, IIT Bombay

Autumn 2024

Problems brought by assignments

```
(define count 1)
(define (f v)
  (set! count (+ 1 count))
  (+ v count))
```

```
> (f 2)
> 4
> (f 2)
> 5
```

- Procedures are no longer mathematical functions!

```
(define (make-simplified-withdraw balance)
  (lambda (amount)
    (set! balance (- balance amount))
    balance))
```

```
((make-simplified-withdraw 25) 20)
==> ((lambda (amount) (set! balance (- 25 amount) 25) 20)
==> (set! balance (- 25 20)) 25
==> Set balance to 5 and return 25
```

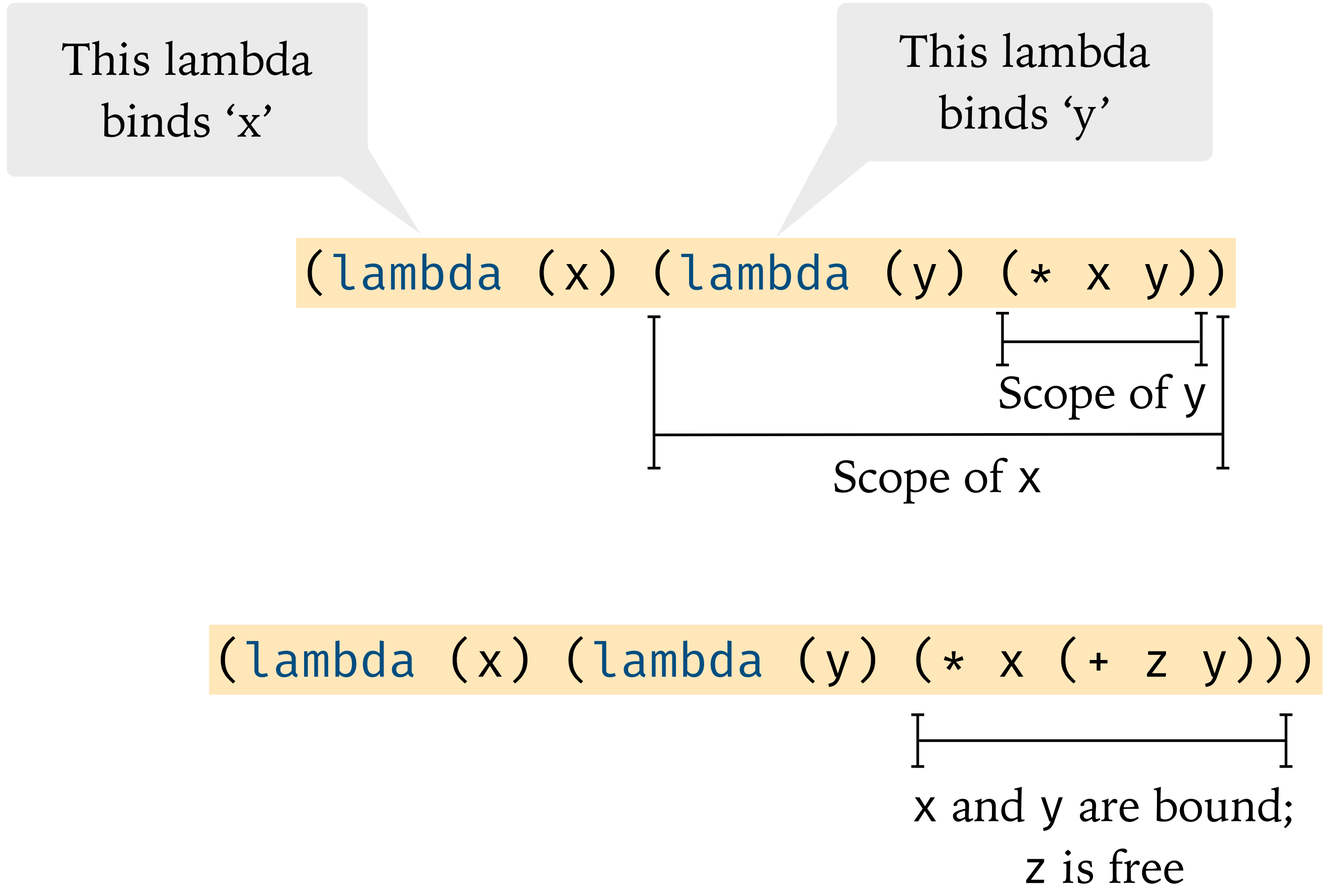
- Substitution model of procedure application no longer works!
  - Reason: Variables are no longer simply names for expressions!

Substitution model revisited

- **Evaluating a combination:**
  - Evaluate the subexpressions of the combination.
  - Apply the value of the operator subexpression to the values of the operand subexpressions.

- **Applying a procedure:**
  - To apply a compound procedure to arguments, evaluate the body of the procedure with each formal parameter replaced by the corresponding actual argument.

Bindings and scope revisited



The only way to create a binding is via a lambda!

Lambda — The Ultimate Binder

```
(let ((var1 expr1)
      (var2 expr2))
  <body>)
```

```
((lambda (var1 var2)
    <body>)
    expr1 expr2)
```

```
(define (foo x)
  <body>)
```

```
(define foo
  (lambda (x)
    <body>))
```

```
(define (foo x)
  (define (var expr))
    <body>)
```

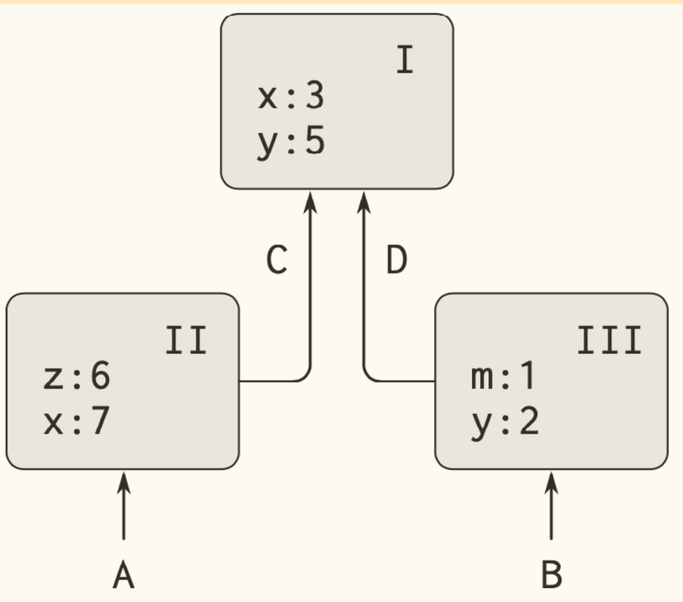
```
(define (foo x)
  (let ((var expr))
    <body>))
```

The Environment Model

Frames and environments

I, II, III: Frames

A, B, C, D: Environments



Environments as chains of frames

Environment ‘A’:
- x: 7, y: 5, z: 6
- m: unbound

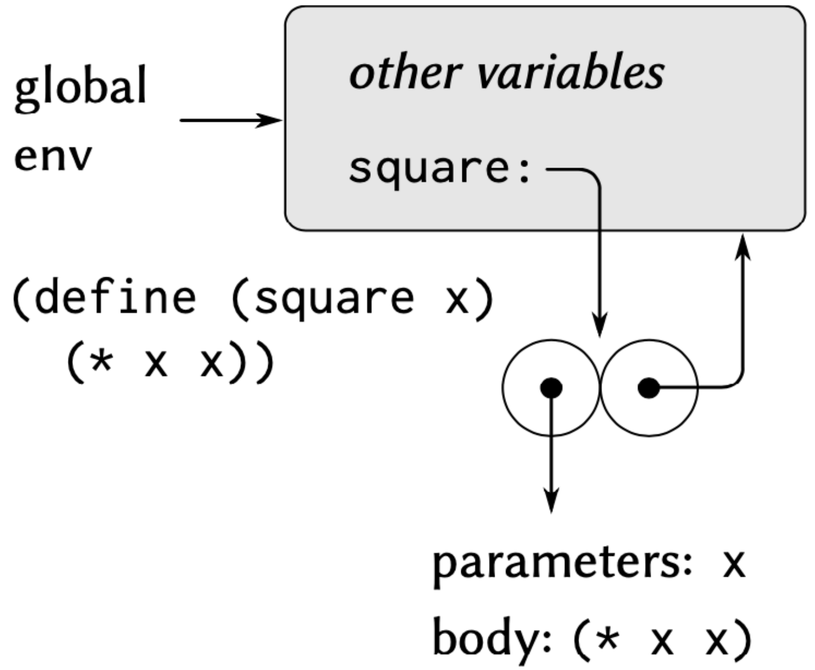
Environment ‘B’:
- x: 3, y: 2, m: 1
- z: unbound

Environments ‘C’, ‘D’:
- x: 3, y: 5
- z, m: unbound

Evaluating procedure definitions

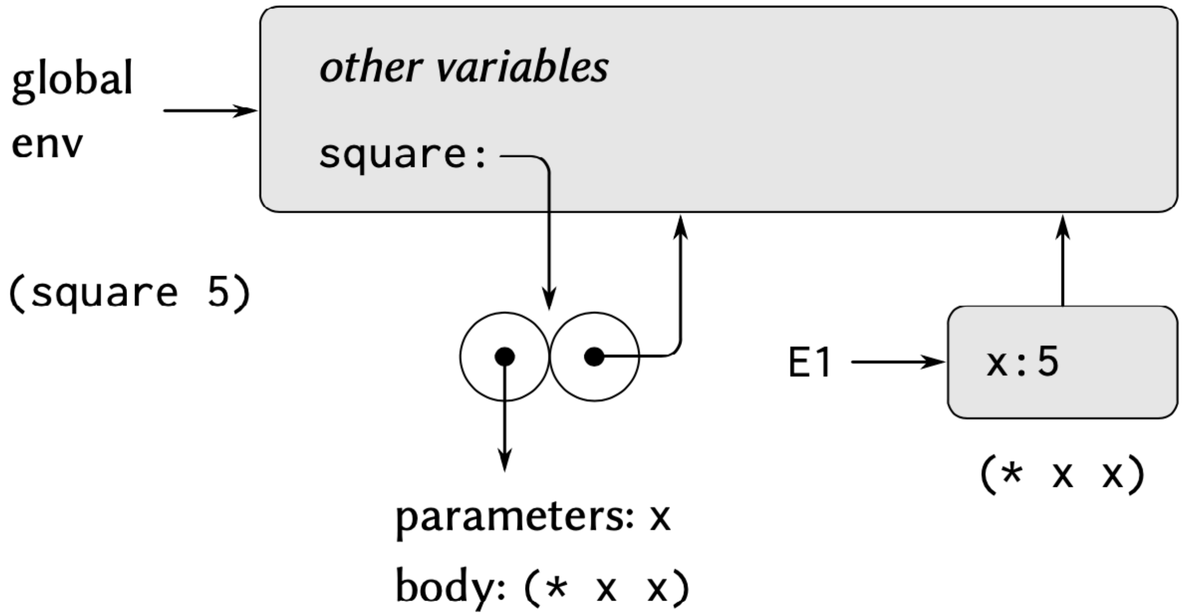
```
(define (square x)
  (* x x))
```

```
(define square
  (lambda (x) (* x x)))
```



Evaluating procedure applications

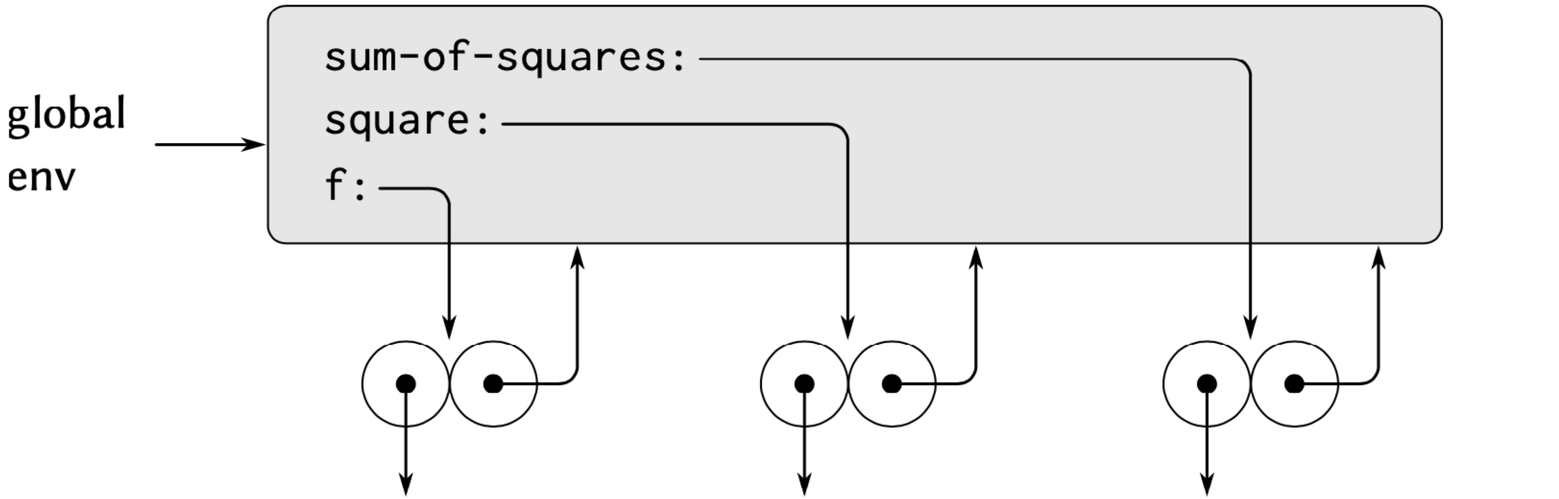
```
(define (square x)
  (* x x))

> (square 5)
```



The Environment Model: Summary

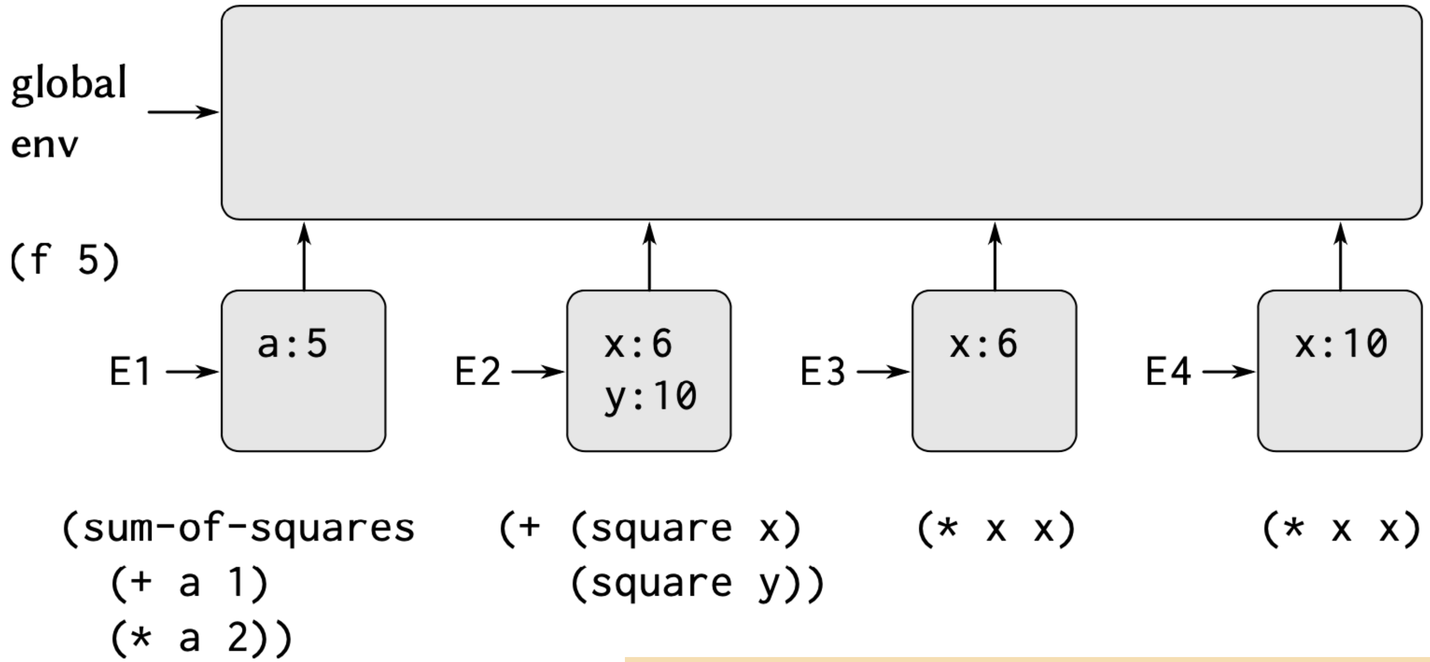
- **Evaluating a combination:** Same as the substitution model
  - Evaluate the subexpressions of the combination.
  - Apply the value of the operator subexpression to the values of the operand subexpressions.
- **Evaluating a procedure definition:**
  - Create a procedure object by evaluating a lambda-expression relative to a given environment.
- **Evaluating a procedure application:**
  - Create a new environment containing a frame that binds the parameters to the arguments, and then evaluate the body of the procedure **in the** new environment.

Extended example 1: Definition



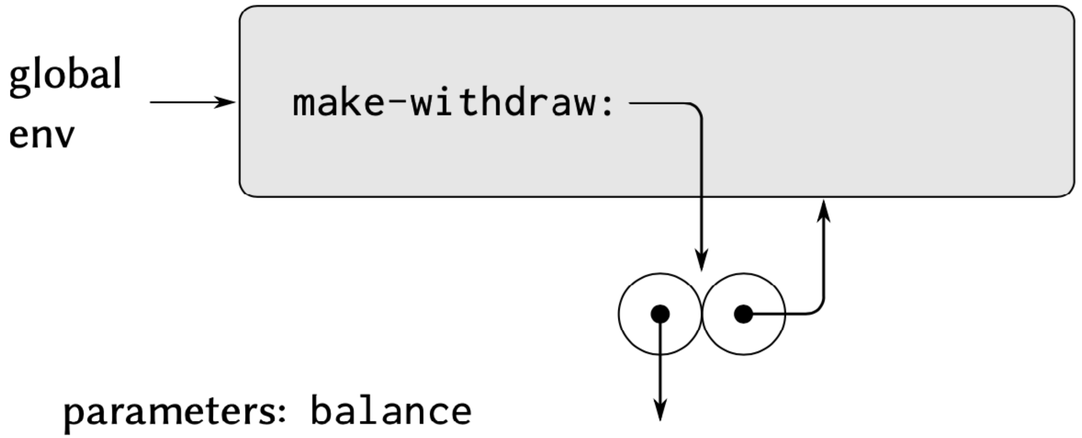

```
(define (square x)
  (* x x))
(define (sum-of-squares x y)
  (+ (square x) (square y)))
(define (f a)
  (sum-of-squares (+ a 1) (* a 2)))
```

Extended example 1: Application



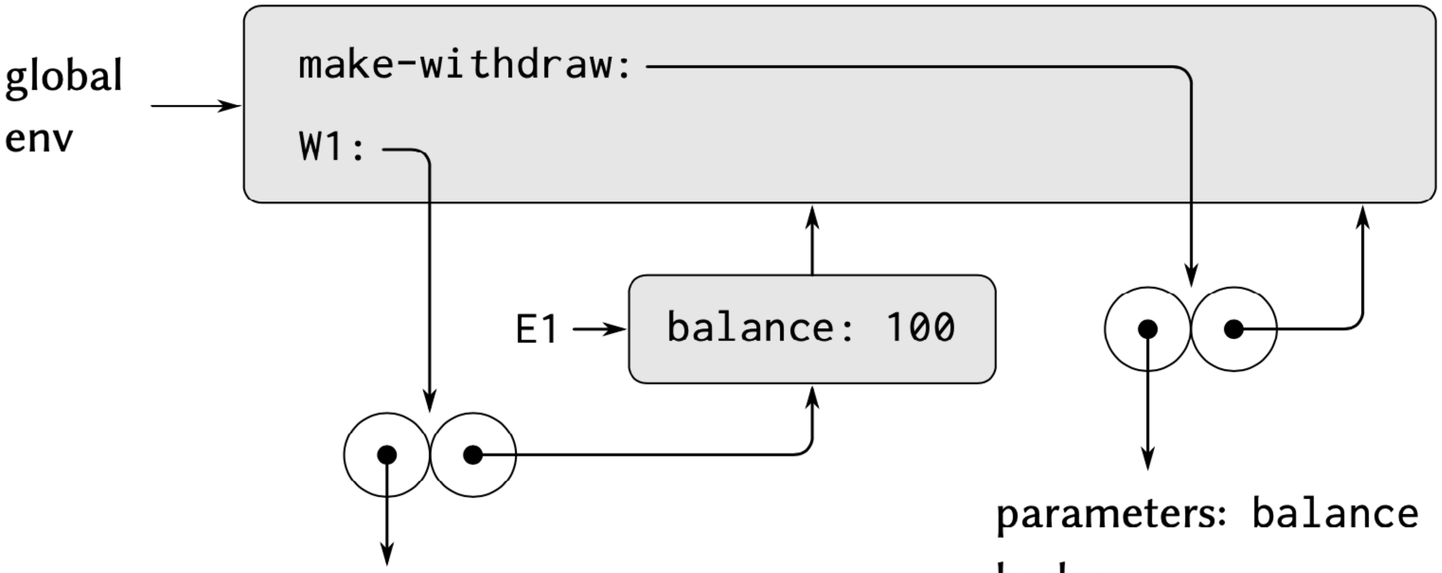

```
(define (square x)
  (* x x))
(define (sum-of-squares x y)
  (+ (square x) (square y)))
(define (f a)
  (sum-of-squares (+ a 1) (* a 2)))

> (f 5)
```

EE2 Def: Local states in frames



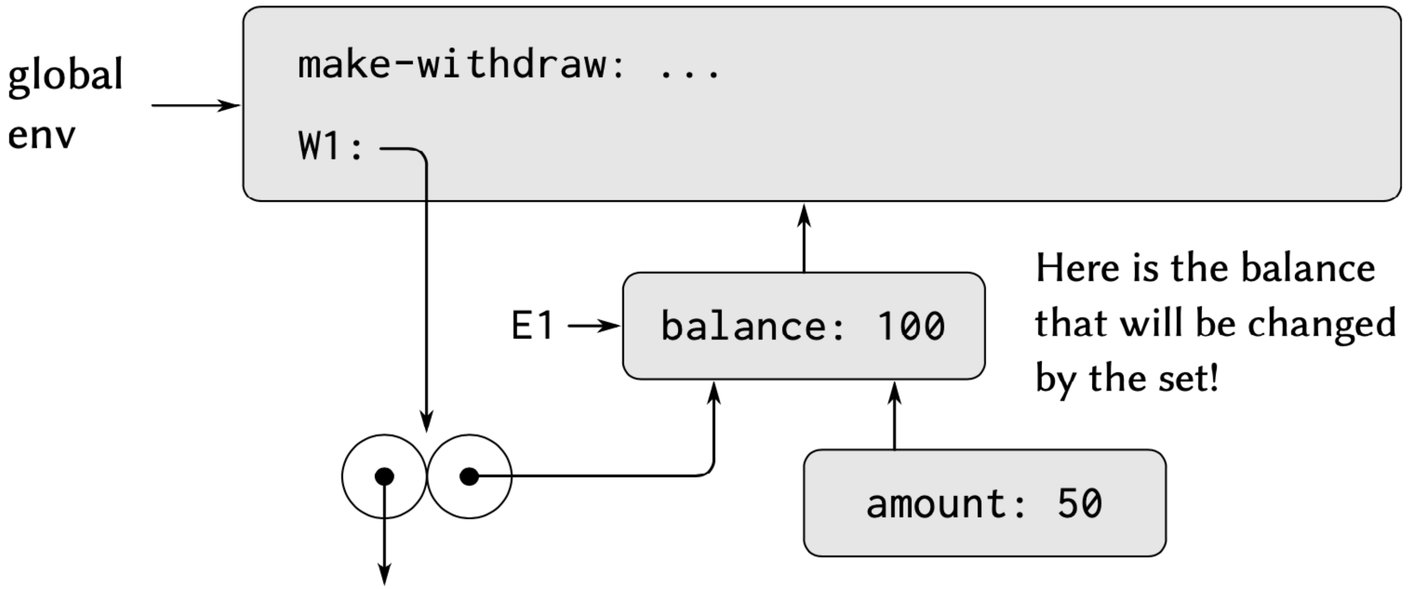

```
(define (make-withdraw balance)
  (lambda (amount)
    (if (>= balance amount)
      (begin (set! balance (- balance amount))
             balance)
      “insufficient funds”)))
```

EE2 App1



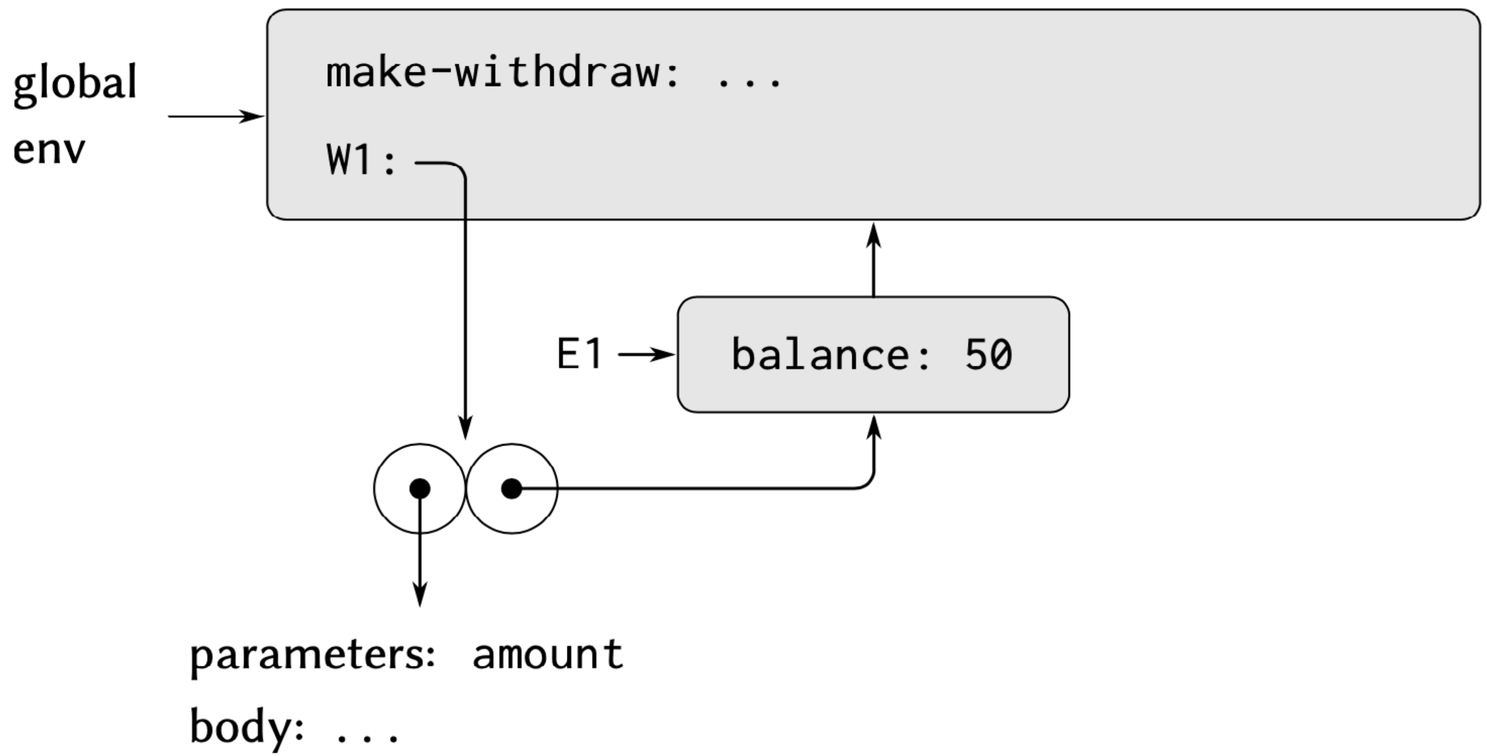

```
(define W1 (make-withdraw 100))
```

EE2 App1 (Cont.)



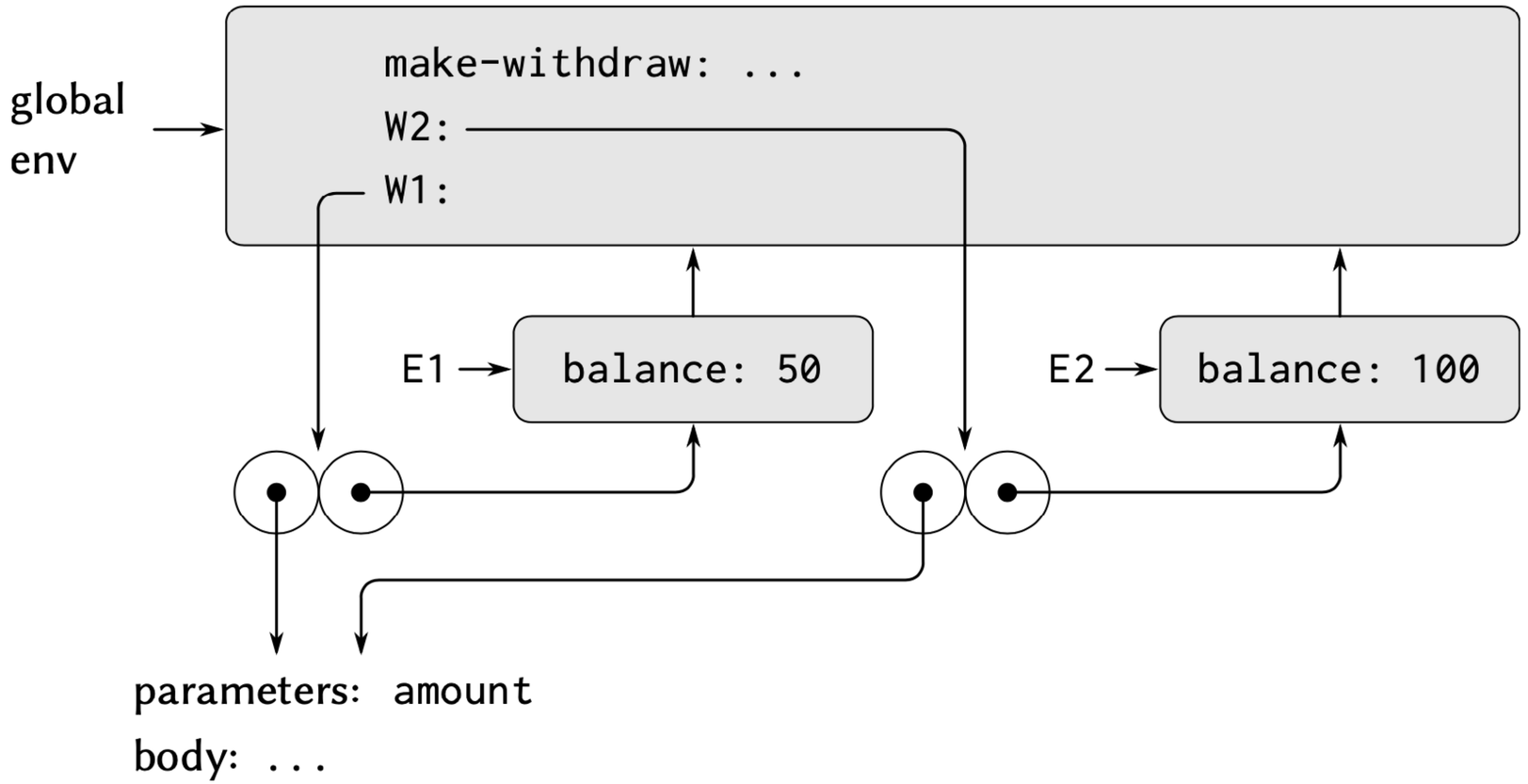

```
> (W1 50)
```

EE2 App1 (Cont.)




```
After (W1 50)
```

EE2 App2




```
(define W2 (make-withdraw 100))
```

Internal Definitions

```
> (sqrt 2)
```

```
(define (sqrt x)
  (define (good-enough? guess)
    (< (abs (- (square guess) x)) 0.001))
  (define (improve guess)
    (average guess (/ x guess)))
  (define (sqrt-iter guess)
    (if (good-enough? guess)
        guess
        (sqrt-iter (improve guess))))
  (sqrt-iter 1.0))
```