



CS1 Recitation

Week 8



Variations on a Theme

- Several varieties of message-passing implementations.
 - Need to be comfortable with all of them
- All have the same basic theme
 - A procedure takes an operand symbol, and performs actions based on the operand
- Today: four versions of make-meter

make-meter Version 1

```
(define (make-meter x)
  (lambda (op . args)
    (cond ((eq? op 'get-meter) x)
          ((eq? op 'set-meter!) ;; implicit begin
           (set! x (car args))
           x)
          ((eq? op 'add-meter!) ;; implicit begin
           (set! x (+ x ((car args) 'get-meter)))
           x)
          (else (error "Invalid op: " op))))))
```

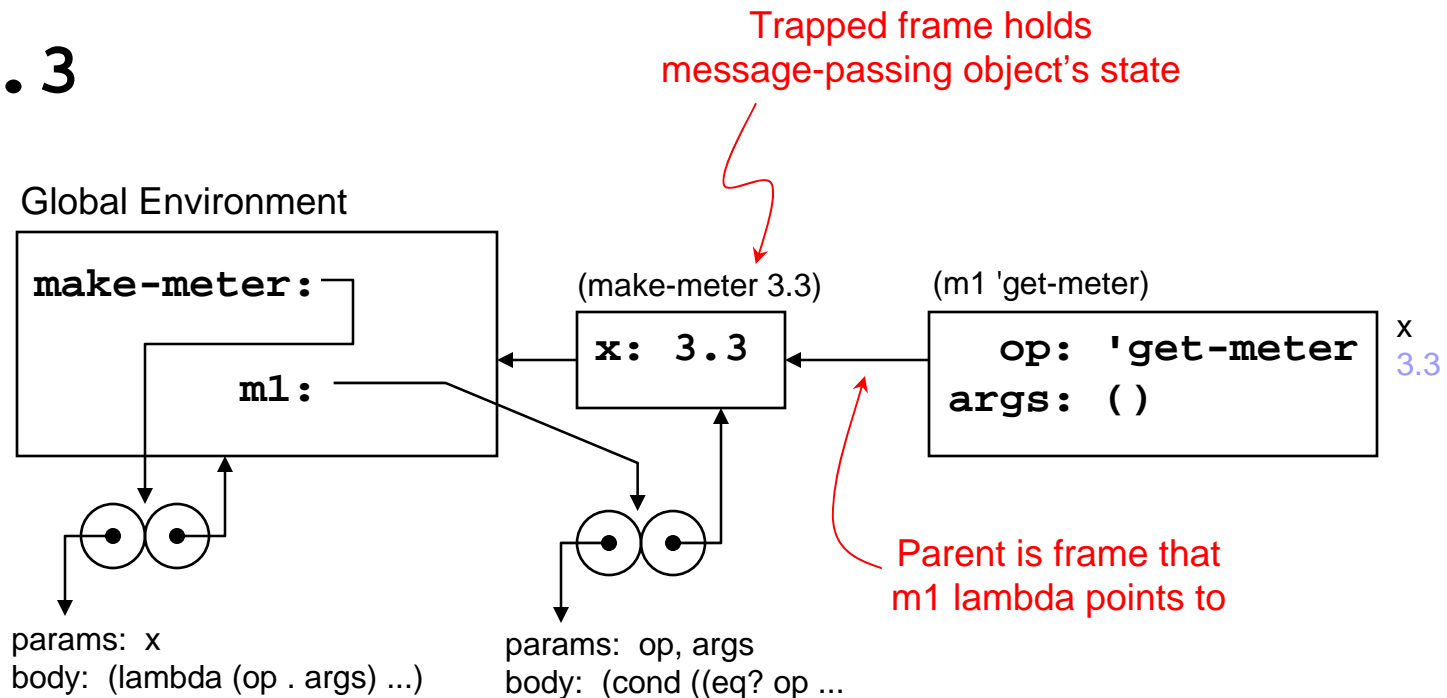
- This is the original technique
 - `make-meter` returns a procedure

Using `make-meter v1` (1)

```
(define m1 (make-meter 3.3))
```

```
(m1 'get-meter)
```

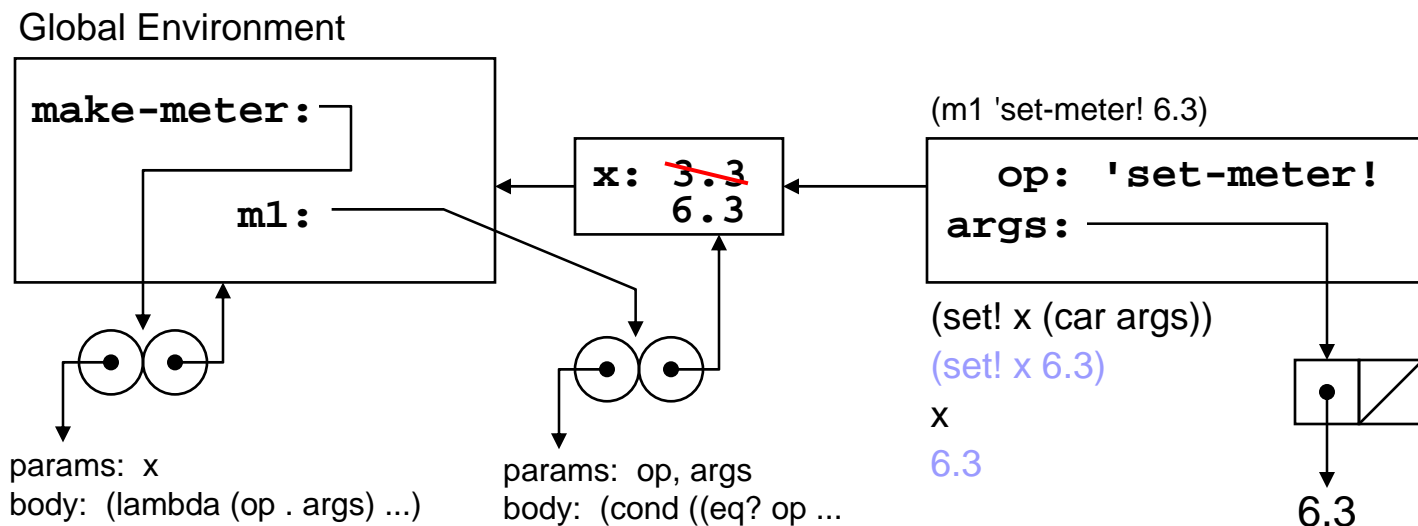
→ 3.3



Using make-meter v1 (2)

(m1 'set-meter! 6.3)

→ 6.3



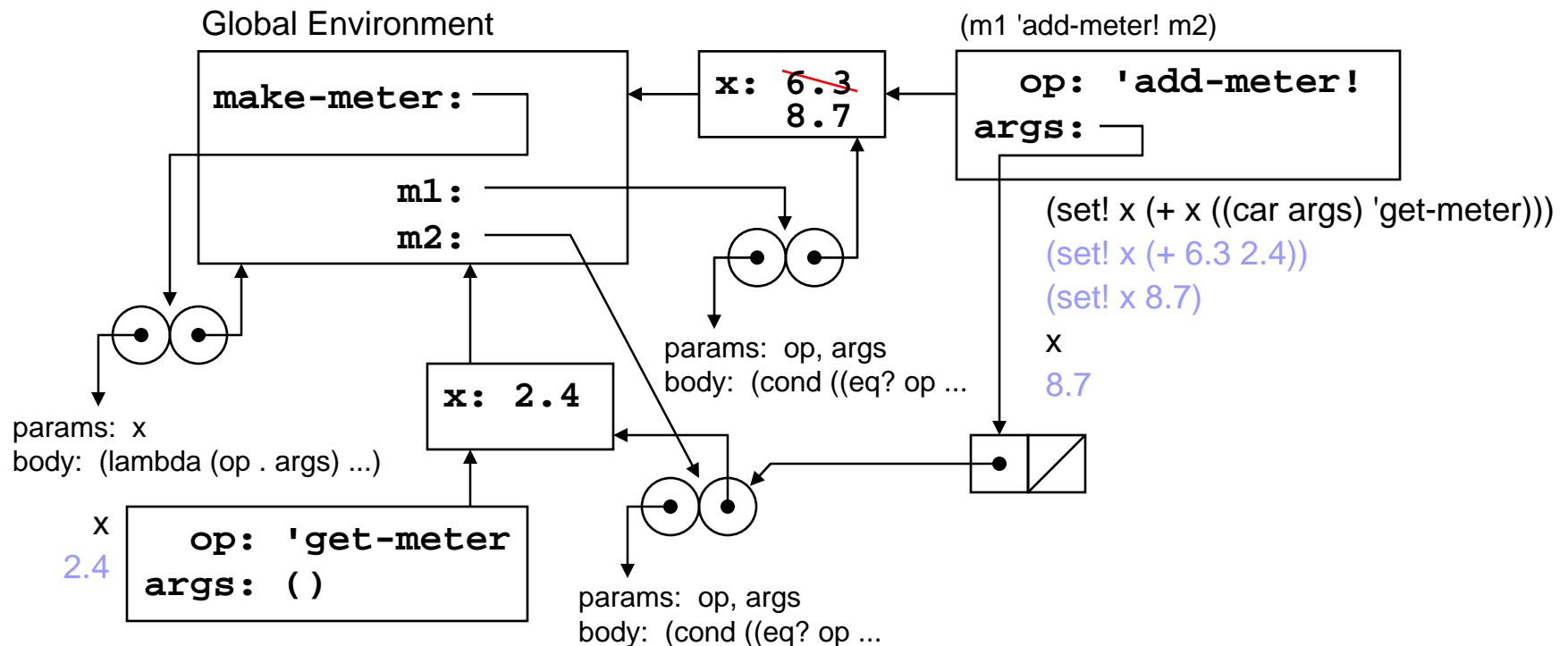
Strongly encouraged to
draw out argument lists!

Using make-meter v1 (3)

```
(define m2 (make-meter 2.4))
```

```
(m1 'add-meter! m2)
```

→ 8.7



Using `make-meter v1` (4)

Note the operations that take arguments!

```
(m1 'get-meter)
```

```
(m1 'set-meter 6.3)
```

```
(m1 'add-meter m2)
```

- Same calling pattern as ones that don't take args
 - Not all message-passing implementations follow this pattern

make-meter Version 2

```
(define (make-meter x)
  (lambda (op)
    (cond ((eq? op 'get-meter) x)
          ((eq? op 'set-meter!)
           (lambda (new-x) ;; op needs a numeric arg
             (set! x new-x)
             x))
          ((eq? op 'add-meter!)
           (lambda (m2) ;; op needs another meter
             (set! x (+ x (m2 'get-meter))))
           x))
          (else (error "Invalid op: " op))))))
```

- Operations that take args return a procedure
 - Better argument checking than before...

Using `make-meter` v2 (1)

```
(define m1 (make-meter 3.3))
```

```
(m1 'get-meter)
```

```
→ 3.3
```

```
((m1 'set-meter!) 6.3)
```

```
→ 6.3
```

- Code for `'set-meter!` is:

```
((eq? op 'set-meter!)
```

```
(lambda (new-x)
```

```
(set! x new-x)
```

```
x))
```

- Another lambda is returned, which takes one argument
 - Lambda then applied to 6.3, to perform `'set-meter!` operation
 - Lambda can access message-passing object's frame

Using `make-meter` v2 (2)

```
(define m2 (make-meter 2.4))
```

```
(m2 'get-meter)
```

```
→ 2.4
```

```
((m1 'add-meter!) m2)
```

```
→ 8.7
```

- Operations that take arguments need extra parens
- This mechanism is more like how C++, Java, other OOP languages work internally
 - Each object-method knows how to parse its own args

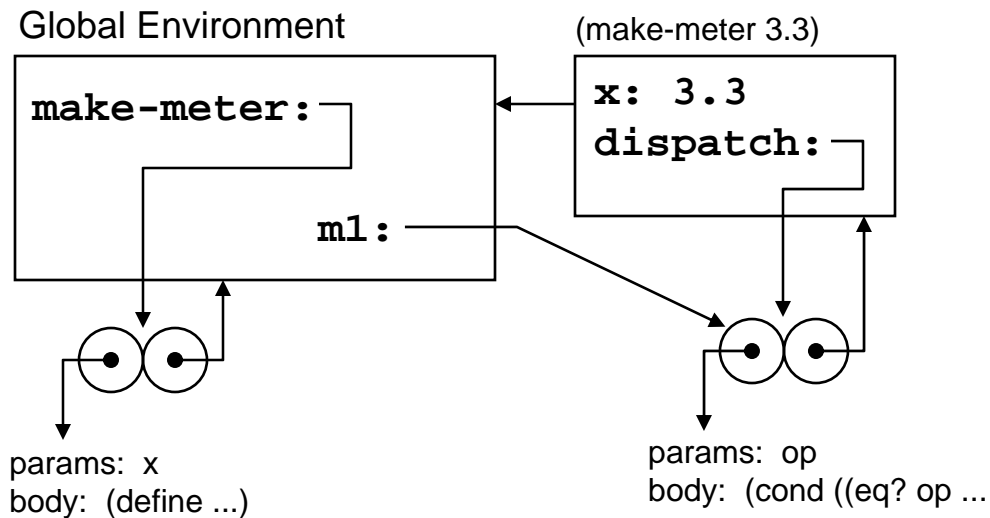
make-meter Version 3

```
(define (make-meter x)
  (define (dispatch op)      ;; internal dispatch proc.
    (cond ((eq? op 'get-meter) x)
          ((eq? op 'set-meter!)
           (lambda (new-x)
             (set! x new-x)
             x))
          ((eq? op 'add-meter!)
           (lambda (m2)
             (set! x (+ x (m2 'get-meter))))
           x))
          (else (error "Invalid op: " op))))
  dispatch)
```

- Now the dispatch procedure is named.

make-meter Version 3 (2)

```
(define (make-meter x)
  (define (dispatch op) ... )
  dispatch)
(define m1 (make-meter 3.3))
```



Note that this lambda now has a binding to refer to itself!

Using `make-meter` v3

```
(define m1 (make-meter 3.3))
```

```
(m1 'get-meter)
```

```
→ 3.3
```

```
((m1 'set-meter!) 6.3)
```

```
→ 6.3
```

```
(define m2 (make-meter 2.4))
```

```
(m2 'get-meter)
```

```
→ 2.4
```

```
((m1 'add-meter!) m2)
```

```
→ 8.7
```

- This is used *exactly* like the second version

But wait... there's more!

- A named dispatch procedure inside the object allows for easy calls to “self” or “this object”
 - A very common feature in OOP languages!
- Can implement complex operations in terms of simple operations
 - ...abstraction!

make-meter Version 4

```
(define (make-meter x)
  (define (dispatch op)
    (cond ((eq? op 'get-meter) x)
          ((eq? op 'set-meter!)
           (lambda (new-x)
             (set! x new-x)
             x))
          ((eq? op 'add-meter!)
           (lambda (m2)      ;; use get-meter/set-meter!
             ((dispatch 'set-meter!)
              (+ (dispatch 'get-meter) (m2 'get-meter))))))
    (else (error "Invalid op: " op))))
  dispatch)
```

- Can invoke operations on “self” now.

Abstractions

- Again, usage of this version of meter is same as versions 2, 3
- But, meter now has layers of abstraction inside.
 - Implemented `add-meter!` in terms of `get-meter` and `set-meter!`

```
((eq? op 'add-meter!)  
  (lambda (m2)      ;; use get-meter/set-meter!  
    ((dispatch 'set-meter!)  
      (+ (dispatch 'get-meter) (m2 'get-meter))))))
```
- Can even change the way that our object stores data
 - Only need to change `get-meter` and `set-meter!`
 - `add-meter!` will be completely unaffected