

6.001 SICP Environment model

; Can you figure out why this code works?

```
(define make-counter
  (lambda (n)
    (lambda () (set! n (+ n 1))
              n )))

(define ca (make-counter 0))
(ca) ==> 1
(ca) ==> 2
(define cb (make-counter 0))
(cb) ==> 1
(ca) ==> 3 ; ca and cb are independent
```

1/35

The Environment Model is:

- A precise, completely mechanical description of:
 - name-rule looking up the value of a variable
 - define-rule creating a new definition of a var
 - set!-rule changing the value of a variable
 - lambda-rule creating a procedure
 - application applying a procedure
- Enables analyzing arbitrary scheme code:
 - Example: **make-counter**
- Basis for implementing a scheme interpreter
 - for now: draw EM state with boxes and pointers
 - later on: implement with code

2/35

A shift in viewpoint

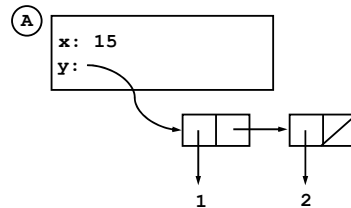
- As we introduce the environment model, we are going to shift our viewpoint on computation
- Variable:
 - OLD – name for value
 - NEW – place into which one can store things
- Procedure:
 - OLD – functional description
 - NEW – object with inherited context
- Expressions
 - Now **only** have meaning with respect to an environment

3/35

Frame: a table of bindings

- Binding:** a pairing of a name and a value

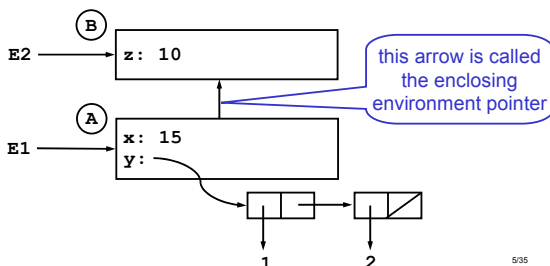
Example: **x** is bound to 15 in frame A
y is bound to (1 2) in frame A
 the value of the variable **x** in frame A is 15



4/35

Environment: a sequence of frames

- Environment E1 consists of frames A and B
- Environment E2 consists of frame B only
 - A frame may be shared by multiple environments



5/35

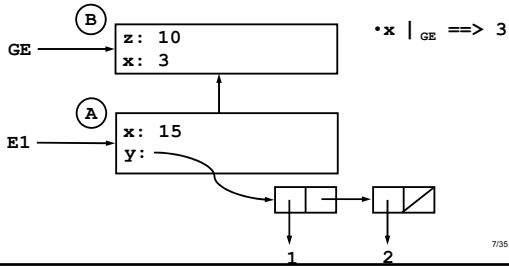
Evaluation in the environment model

- All evaluation occurs **in an environment**
 - The **current environment** changes when the interpreter applies a procedure
- The top environment is called the **global environment (GE)**
 - Only the GE has no enclosing environment
- To **evaluate** a combination
 - Evaluate the subexpressions in the current environment
 - Apply the value of the first to the values of the rest

6/35

Name-rule

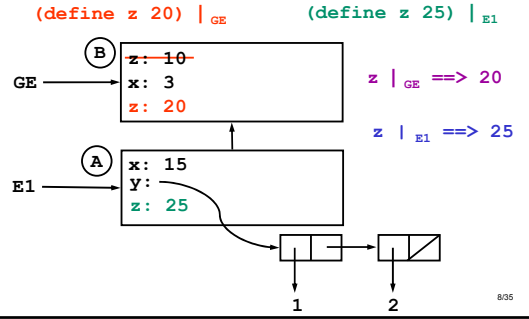
- A name X evaluated in environment E gives the value of X in the first frame of E where X is bound
- $z \mid_{GE} \implies 10$ $z \mid_{E1} \implies 10$ $x \mid_{E1} \implies 15$
- In E1, the binding of x in frame A **shadows** the binding of x in B



7/35

Define-rule

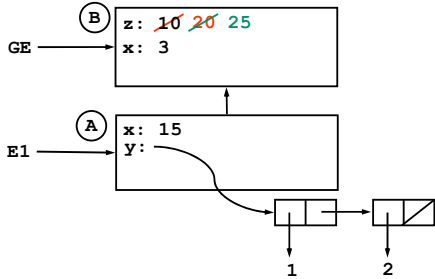
- A define special form evaluated in environment E creates or replaces a binding in the first frame of E



8/35

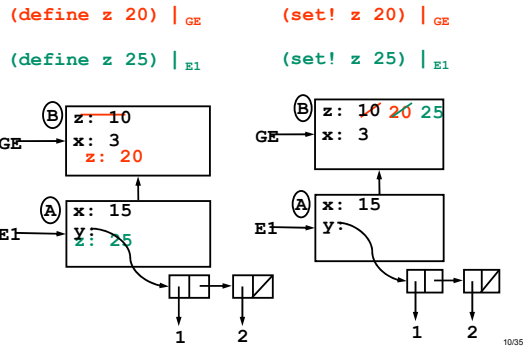
Set!-rule

- A set! of variable X evaluated in environment E changes the binding of X in the first frame of E where X is bound



9/35

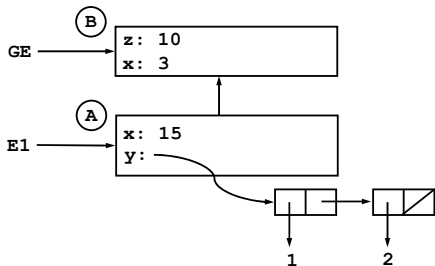
Compare define versus set!



10/35

Your turn: evaluate the following in order

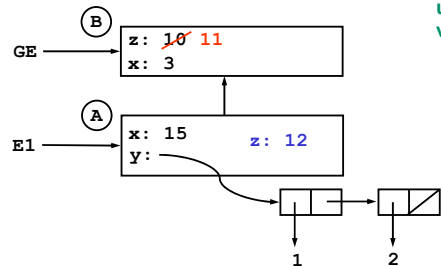
- $(+ z 1) \mid_{E1} \implies$
- $(set! z (+ z 1)) \mid_{E1}$ (modify EM)
 - $(define z (+ z 1)) \mid_{E1}$ (modify EM)
 - $(set! y (+ z 1)) \mid_{GE}$ (modify EM)



11/35

Your turn: evaluate the following in order

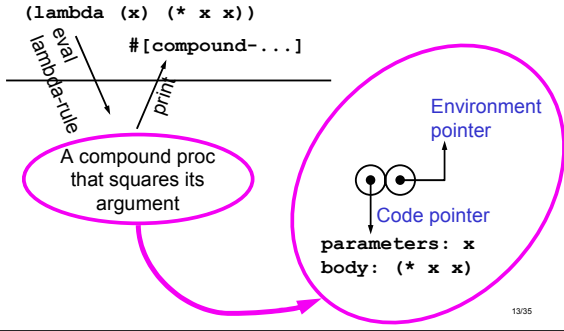
- $(+ z 1) \mid_{E1} \implies$ 11
- $(set! z (+ z 1)) \mid_{E1}$ (modify EM)
 - $(define z (+ z 1)) \mid_{E1}$ (modify EM)
 - $(set! y (+ z 1)) \mid_{GE}$ (modify EM)



Error:
unbound
variable: y

12/35

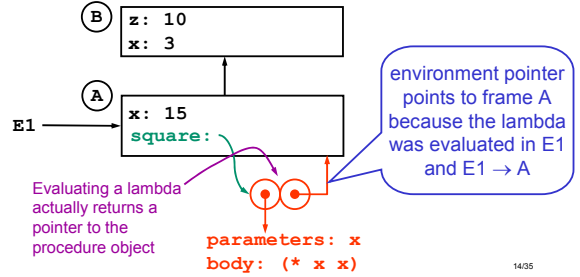
Double bubble: how to draw a procedure



13/35

Lambda-rule

- A lambda special form evaluated in environment E creates a procedure whose environment pointer is E
- (define square (lambda (x) (* x x))) |_{E1}



14/35

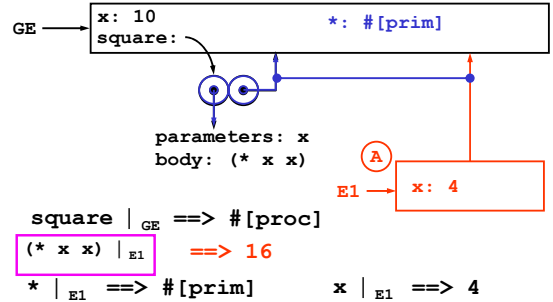
To apply a compound procedure P to arguments:

1. Create a new frame A
2. Make A into an environment E: A's enclosing environment pointer goes to the same frame as the environment pointer of P
3. In A, bind the parameters of P to the argument values
4. Evaluate the body of P with E as the current environment



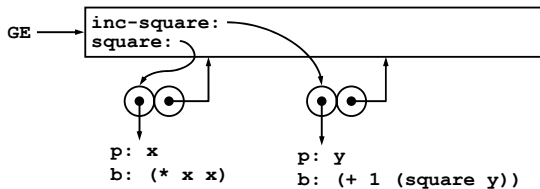
15/35

(square 4) |_{GE}



16/35

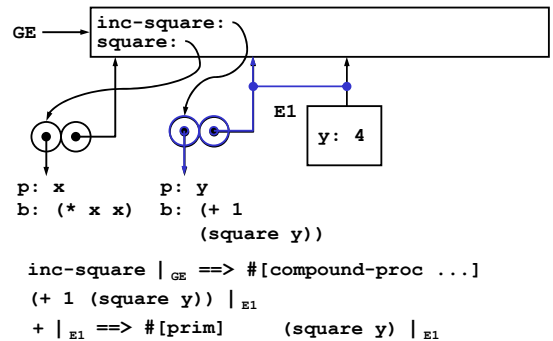
Example: inc-square



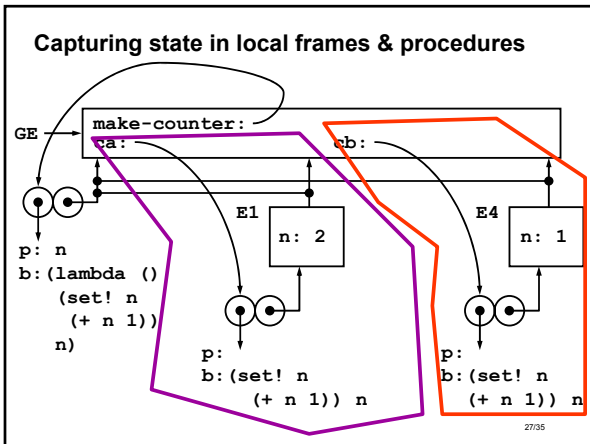
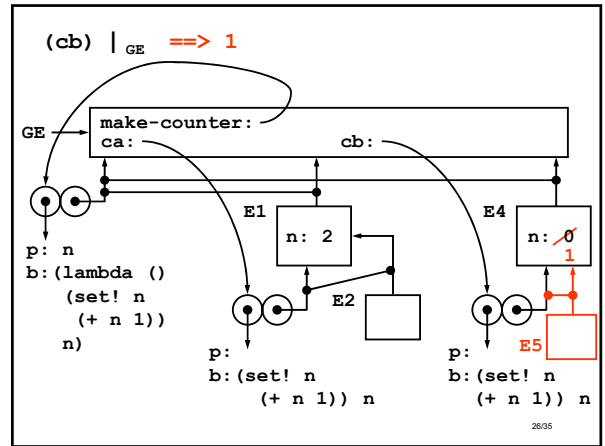
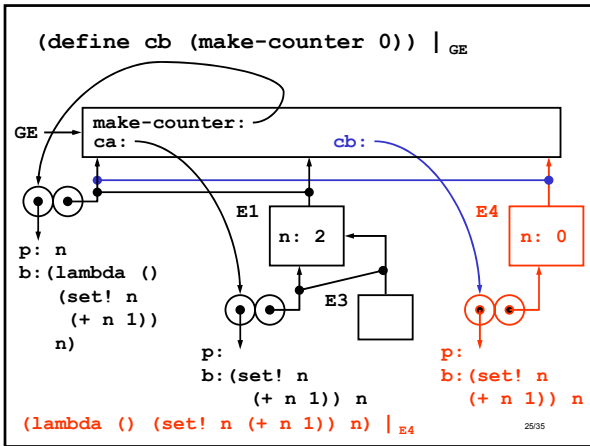
(define square (lambda (x) (* x x))) |_{GE}
(define inc-square (lambda (y) (+ 1 (square y)))) |_{GE}

17/35

Example cont'd: (inc-square 4) |_{GE}



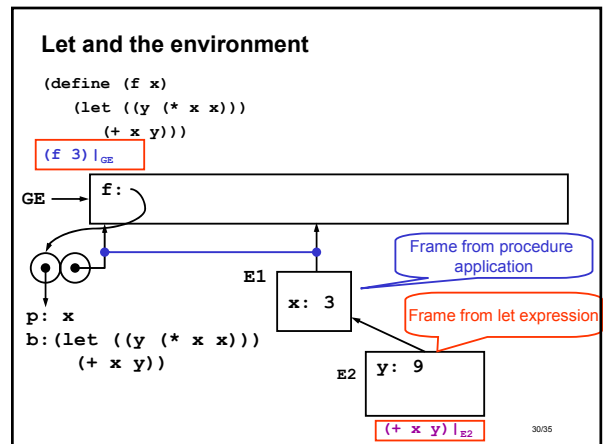
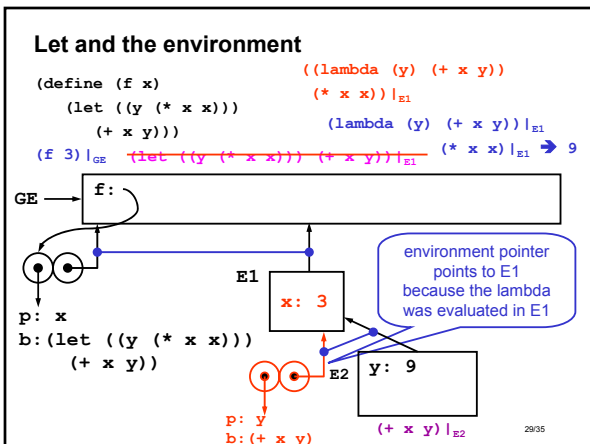
18/35



Lessons from the make-counter example

- Environment diagrams get complicated very quickly
 - Rules are meant for the computer to follow, not to help humans
- A lambda inside a procedure body captures the frame that was active when the lambda was evaluated
 - this effect can be used to store **local state**

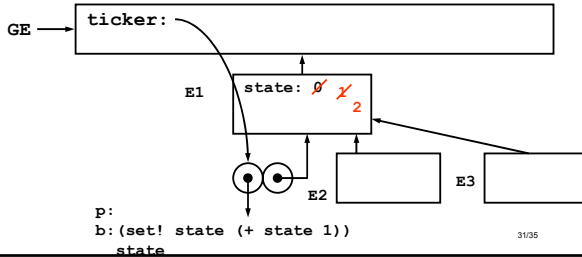
28/35



Capturing state with let

```
(define ticker
  (let ((state 0))
    (lambda ()
      (set! state (+ state 1))
      state)))
```

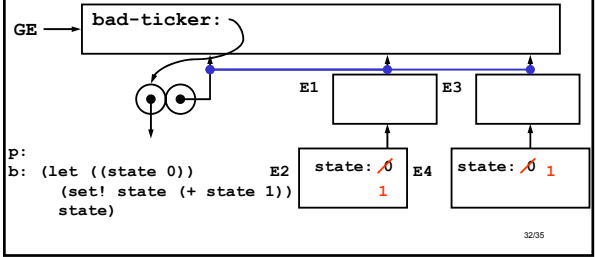
(ticker) → 1
(ticker) → 2



Capturing state with let

```
(define bad-ticker
  (lambda ()
    (let ((state 0))
      (set! state (+ state 1))
      state)))
```

(bad-ticker) → 1
(bad-ticker) → 1



Summary of environments

- Evaluation of all expressions and subexpressions take place with respect to an environment.
- Environments serve as dictionaries to define all terms
- Specific rules for creating and manipulating environments
- Shift in environment takes place with procedure application