Environment model

- Models of computation
  - Substitution model
    - A way to figure out what happens during evaluation

```
(define l '(a b c))
(car l) ==> a
(define m '(1 2 3))
(car l) ==> a
```

    - Not really what happens in the computer

```
(car l) ==> a
(set-car! l 'z)
(car l) ==> z
```

  - The Environment Model

Why does this code work?

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

(define ca (make-counter 0))
(ca) ==> 1
(ca) ==> 2   ; not functional programming!
(define cb (make-counter 0))
(cb) ==> 1
(ca) ==> 3   ; ca and cb are independent
```


The Environment Model

- The Environment Model (EM) 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
- EM enables analyzing more complex Scheme code
  - Example: **make-counter**
- EM will be a basis for implementing a Scheme interpreter
  - for now, we'll just draw EM state with boxes and pointers
  - later on, we'll implement EM with code

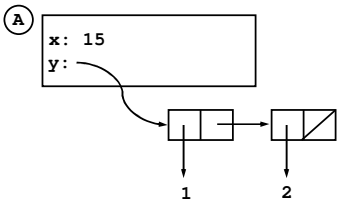
A shift in viewpoint

- 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

Frame: a table of bindings

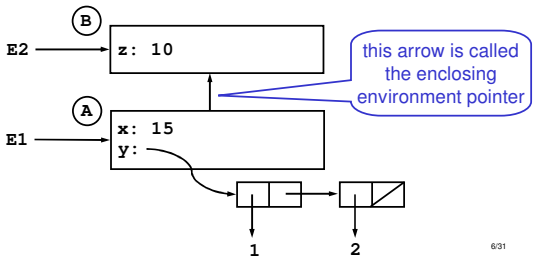
- Binding: a pairing of a name and a value

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



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



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

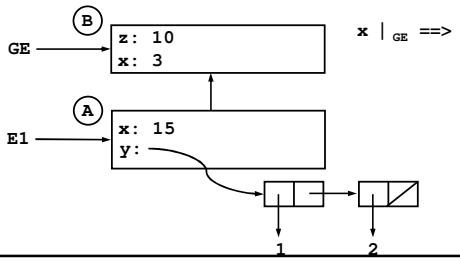
7/31

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 | GE ==>` `z | E1 ==>` `x | E1 ==>`

- In E1, the binding of x in frame A shadows the binding of x in B

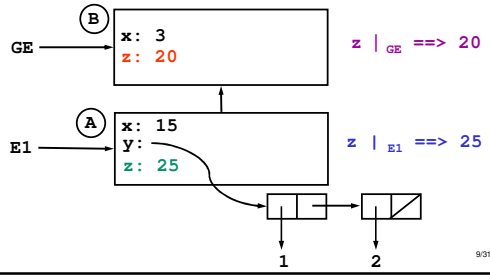


8/31

Define-rule

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

`(define z 20) | GE` `(define z 25) | E1`

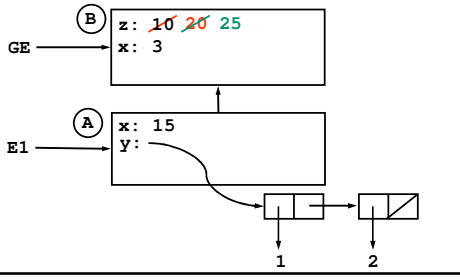


9/31

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

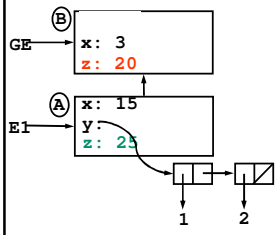
`(set! z 20) | GE` `(set! z 25) | E1`



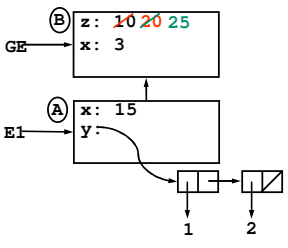
10/31

Define versus Set!

Using define



Using set!

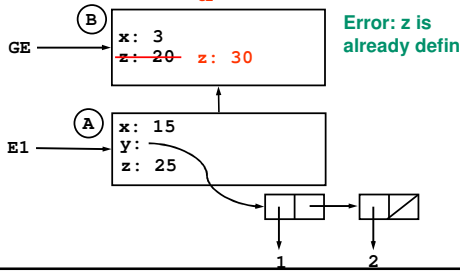


11/31

Define-rule for existing variables (DrScheme only)

- If X is already bound in the first frame of E, and E is:
  - the global environment: define replaces the binding
  - any other environment: it is an error

`(define z 30) | GE` `(define z 35) | E1`



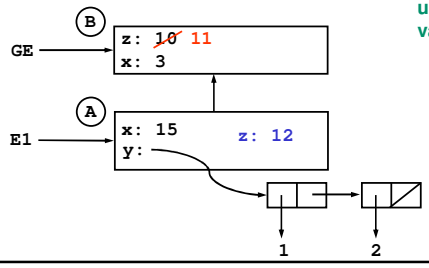
Error: z is already defined

12/31

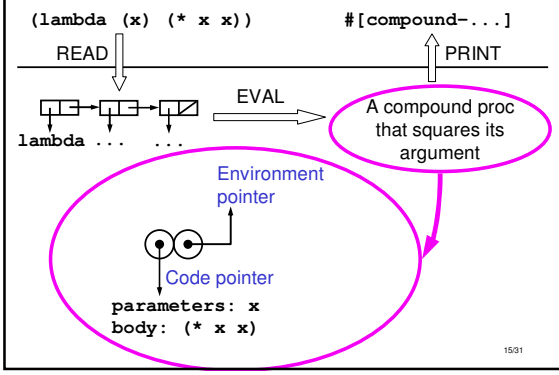
Your turn: evaluate the following in order

```
(+ z 1) | E1                    ==> 11
(set! z (+ z 1)) | E1           (modify EM)
(define z (+ z 1)) | E1         (modify EM)
(set! y (+ z 1)) | GE           (modify EM)
```

Error: unbound variable: y



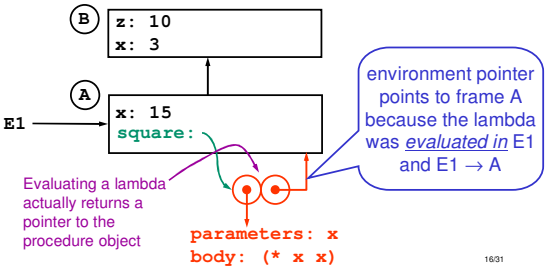
Double bubble: how to draw a procedure



Lambda-rule

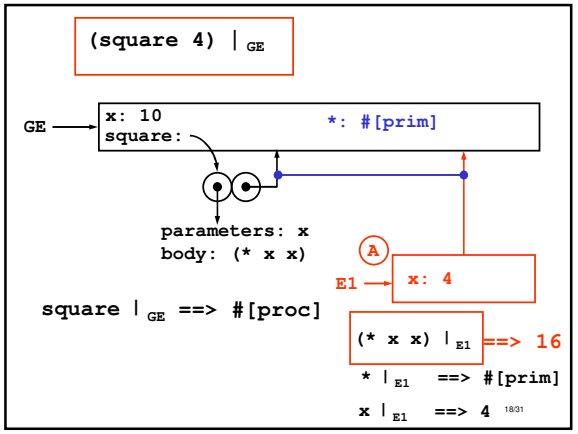
- A lambda special form evaluated in environment E creates a procedure whose environment pointer is E

```
(define square (lambda (x) (* x x))) | E1
```

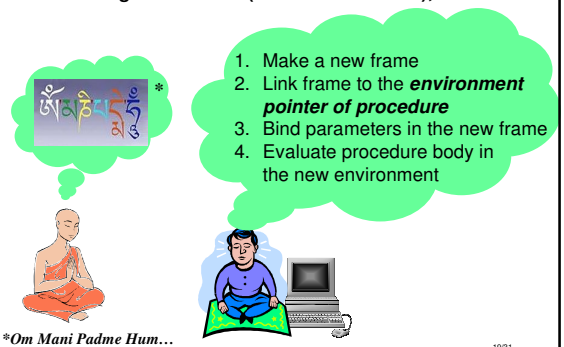


To apply a compound procedure P to arguments:

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

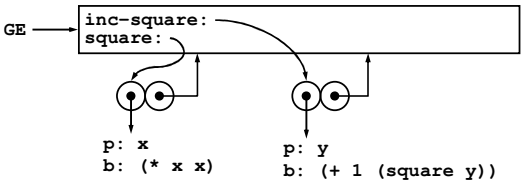


Achieving Inner Peace (and A Good Grade), Part II



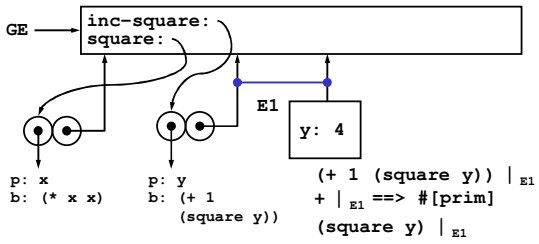
Example: inc-square

```
(define square (lambda (x) (* x x))) | GE
(define inc-square
          (lambda (y) (+ 1 (square y))) | GE
```



20/31

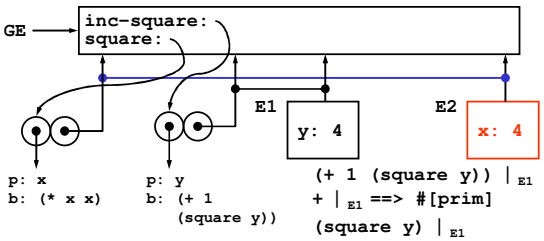
Example cont'd: `(inc-square 4) | GE`



`inc-square | GE ==> #[proc]`

21/31

Example cont'd: `(square y) | E1`



`square | E1 ==> #[proc]`

`y | E1 ==> 4`

`(* x x) | E2 ==> 16`

`(+ 1 16) ==> 17`

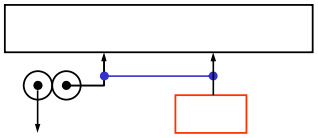
`* | E2 ==> #[prim]`

`x | E2 ==> 4`

22/31

Lessons from the `inc-square` example

- EM doesn't show the complete state of the interpreter
  - missing the stack of pending operations
- The GE contains all standard bindings (`*`, `cons`, etc)
  - usually omitted from EM drawings
- Useful to link the environment pointer of each new frame to the procedure that created it



23/31

Example: make-counter

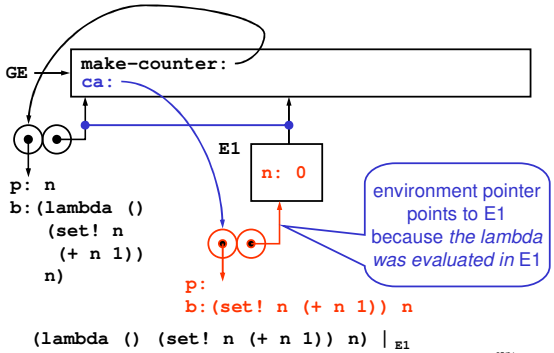
- Counter: counts upwards from a starting number

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

(define ca (make-counter 0))
(ca) ==> 1
(ca) ==> 2 ; not functional programming
(define cb (make-counter 0))
(cb) ==> 1
(ca) ==> 3
(cb) ==> 2  ; ca and cb are independent
```

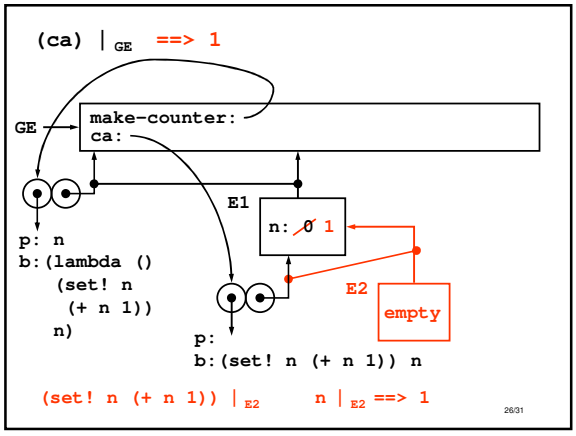
24/31

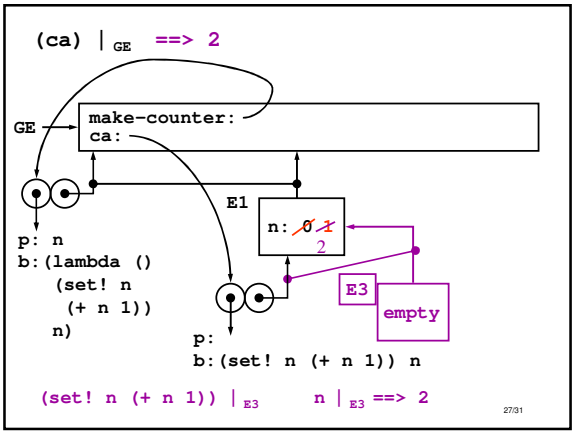
`(define ca (make-counter 0)) | GE`

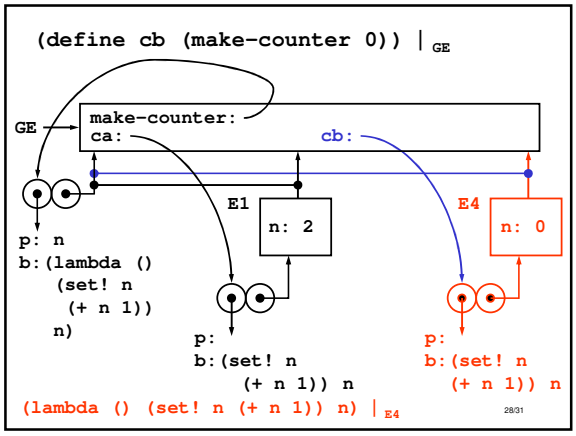


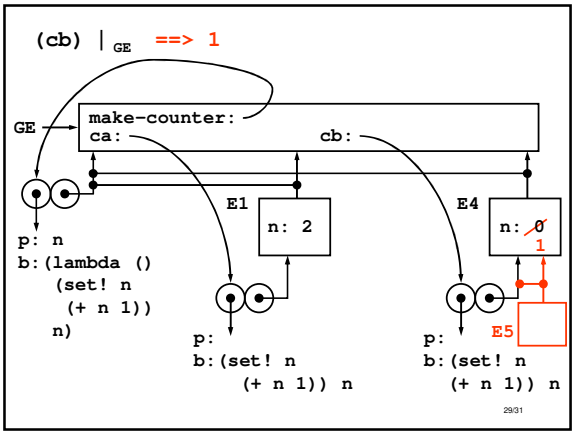
`(lambda () (set! n (+ n 1)) n) | E1`

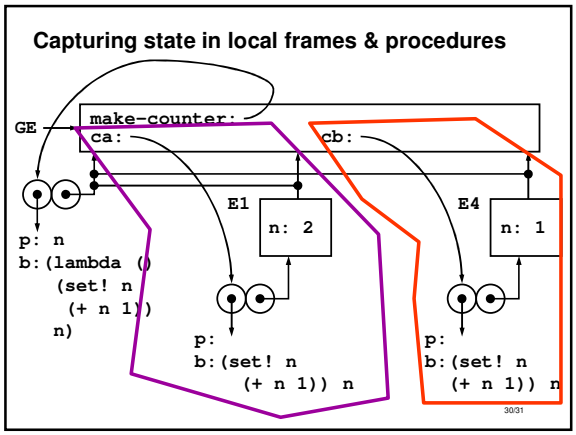
25/31





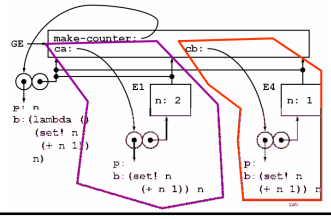




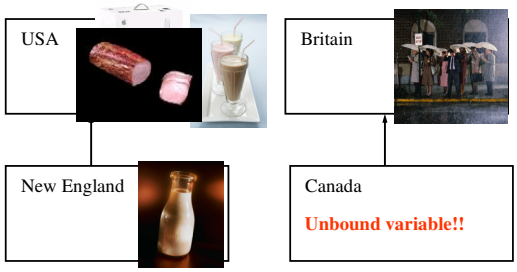


Lessons from the `make-counter` example

- 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



Environments are important in other languages



Macintosh | USA
Milkshake | USA
Canadian bacon | New England

Macintosh | Britain
Milkshake | New England
Canadian bacon | Canada

32/31