An Interpreted Scheme Dialect with a Reflective Tower

Stephen Barnes, stbarnes@stanford.edu

$$x = 2$$

$$x = 2$$

```
x = 2
```

```
locals()['x'] = 2
```

```
x = 2
```

```
locals()['x'] = 2
```

locals()

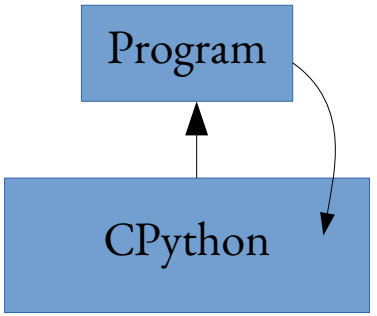
Update and return a dictionary representing the current local symbol table. Free variables are returned by `locals()` when it is called in function blocks, but not in class blocks.

Note: The contents of this dictionary should not be modified; changes may not affect the values of local and free variables used by the interpreter.

```
x = 2
```

```
locals()['x'] = 2
```

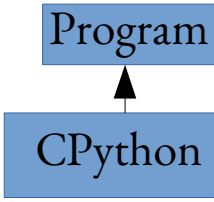
```
mem_file = open("/proc/self/mem", 'r', 0)
...
```



```
x = 2
```

```
locals()['x'] = 2
```

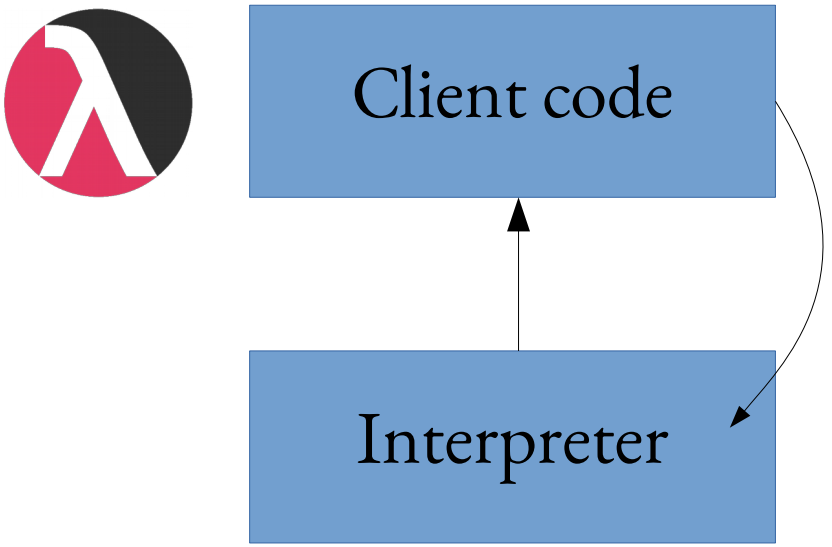
```
mem_file = open("/proc/self/mem", 'r', 0)
...
```

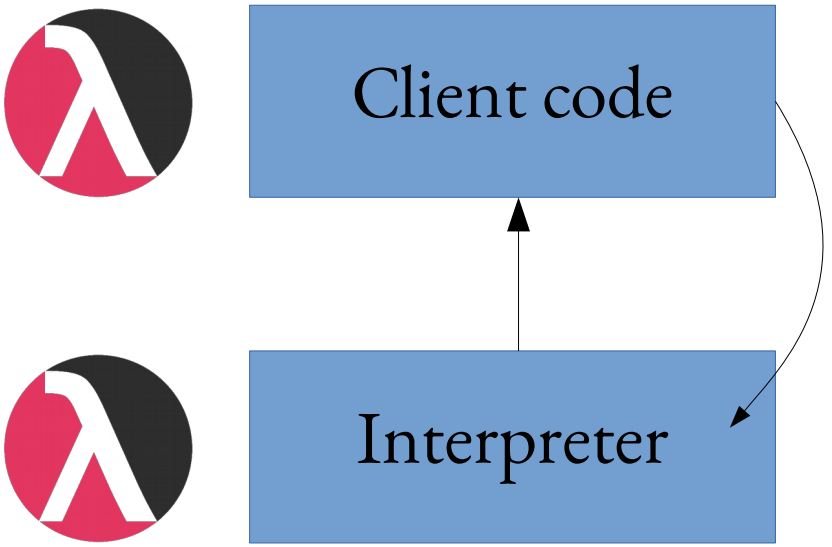


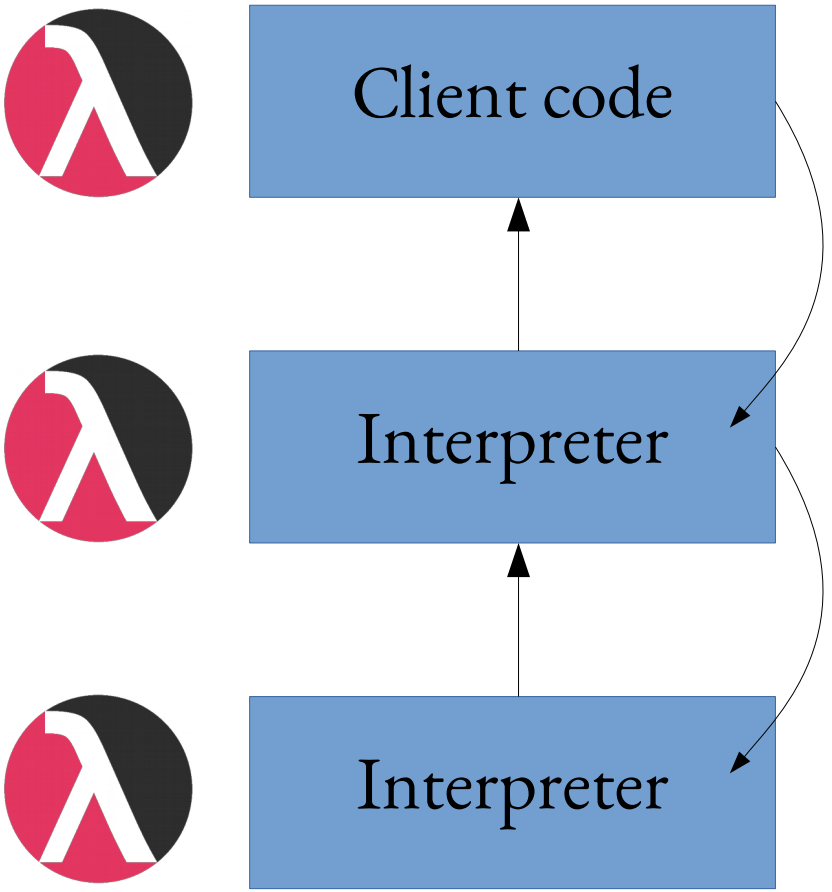
Reflective Tower

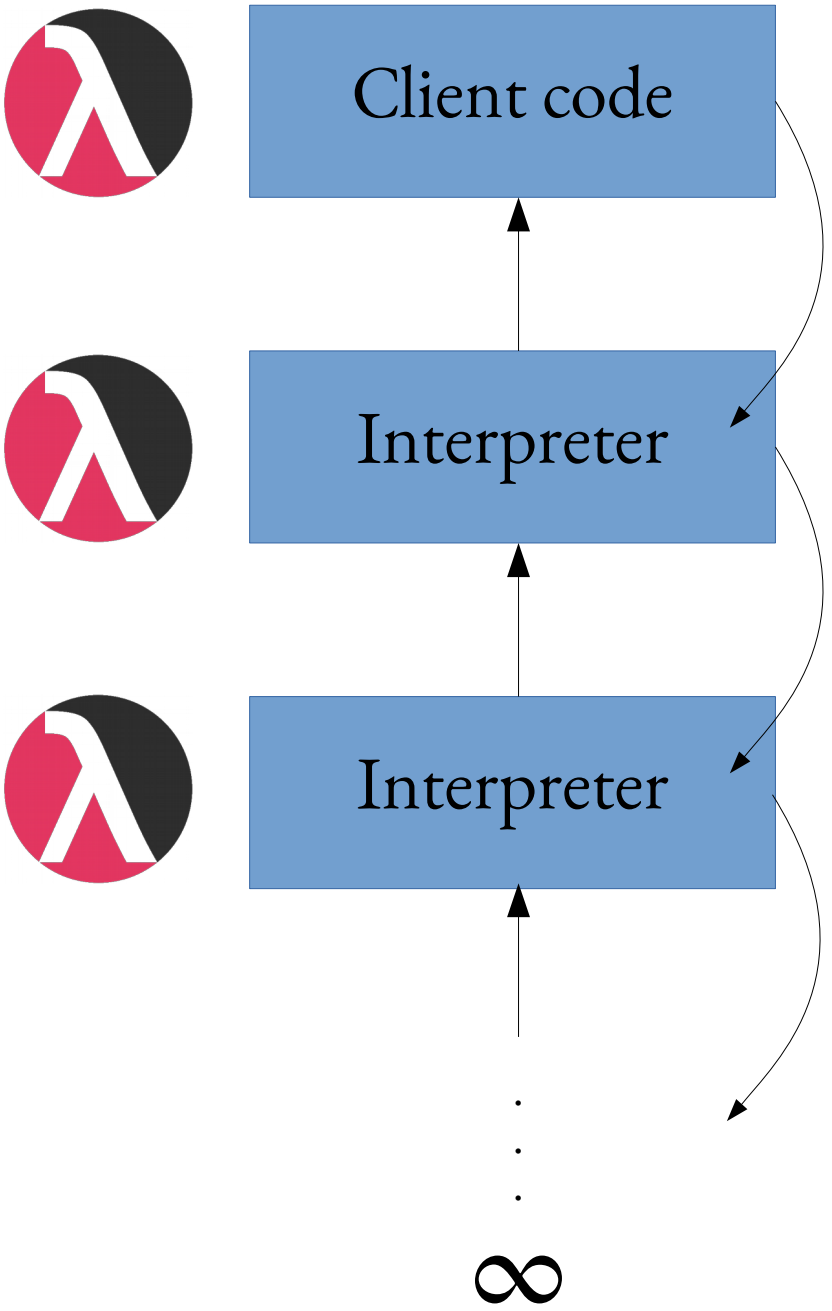
(this project)

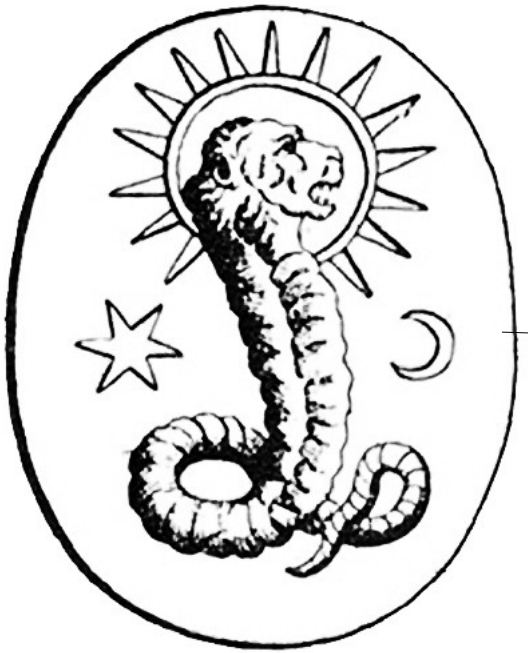
What is a Reflective Tower?


Client code
Interpreter


Client code
Interpreter

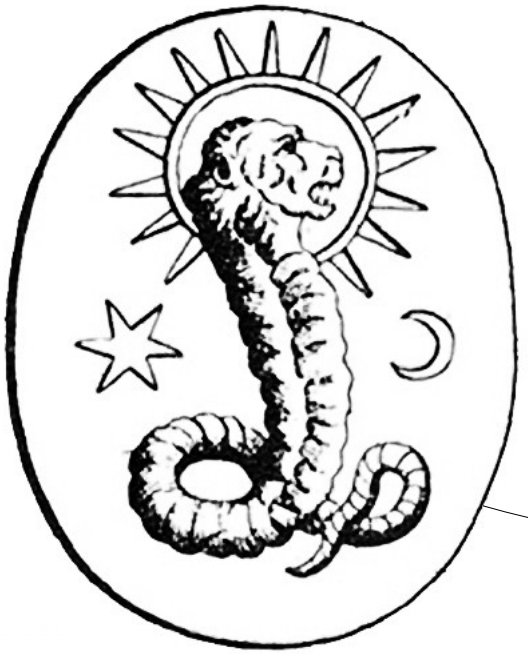

Client code
Interpreter
Interpreter


Client code
Interpreter
Interpreter
∞



Demiurge

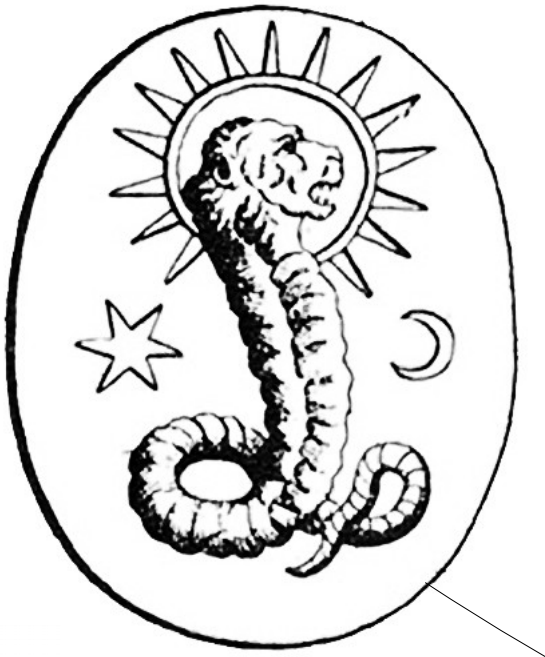
Client code



Demiurge

Client code

Interpreter



Demiurge

Client code

Interpreter

Interpreter

Implementation

Literals:

```
12 ↦ 12

“string” ↦ “string”

#t ↦ True
```

Literals:

```
12 ↦ 12

“string” ↦ “string”

#t ↦ True
```

S-expressions:

```
(fn a b) ↦ fn(a, b)

(+ 1 (+ 2 3)) ↦ 6
```

Literals:

```
12 ↦ 12
“string” ↦ “string”
#t ↦ True
```

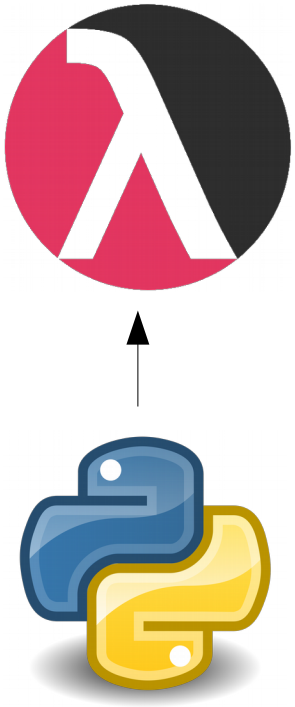
S-expressions:

```
(fn a b) ↦ fn(a, b)
(+ 1 (+ 2 3)) ↦ 6
```

Lambda-functions: `((lambda (a b) (+ a b)) 1 2) ↦ 3`

Assignment: `(set x 12) → x ↦ 12`

Branching: `(if #t 1 2) ↦ 1`



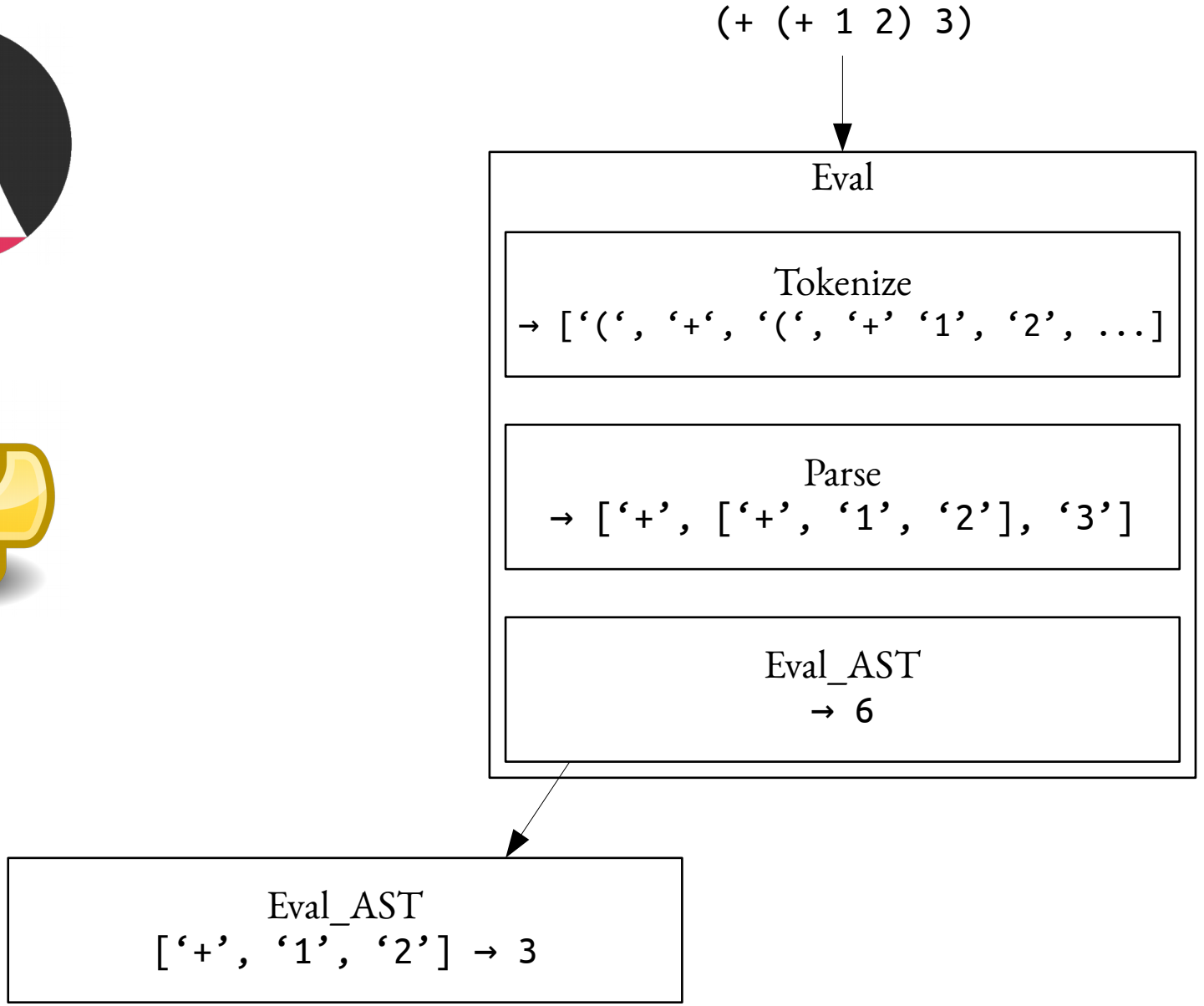
```
(+ (+ 1 2) 3)
```

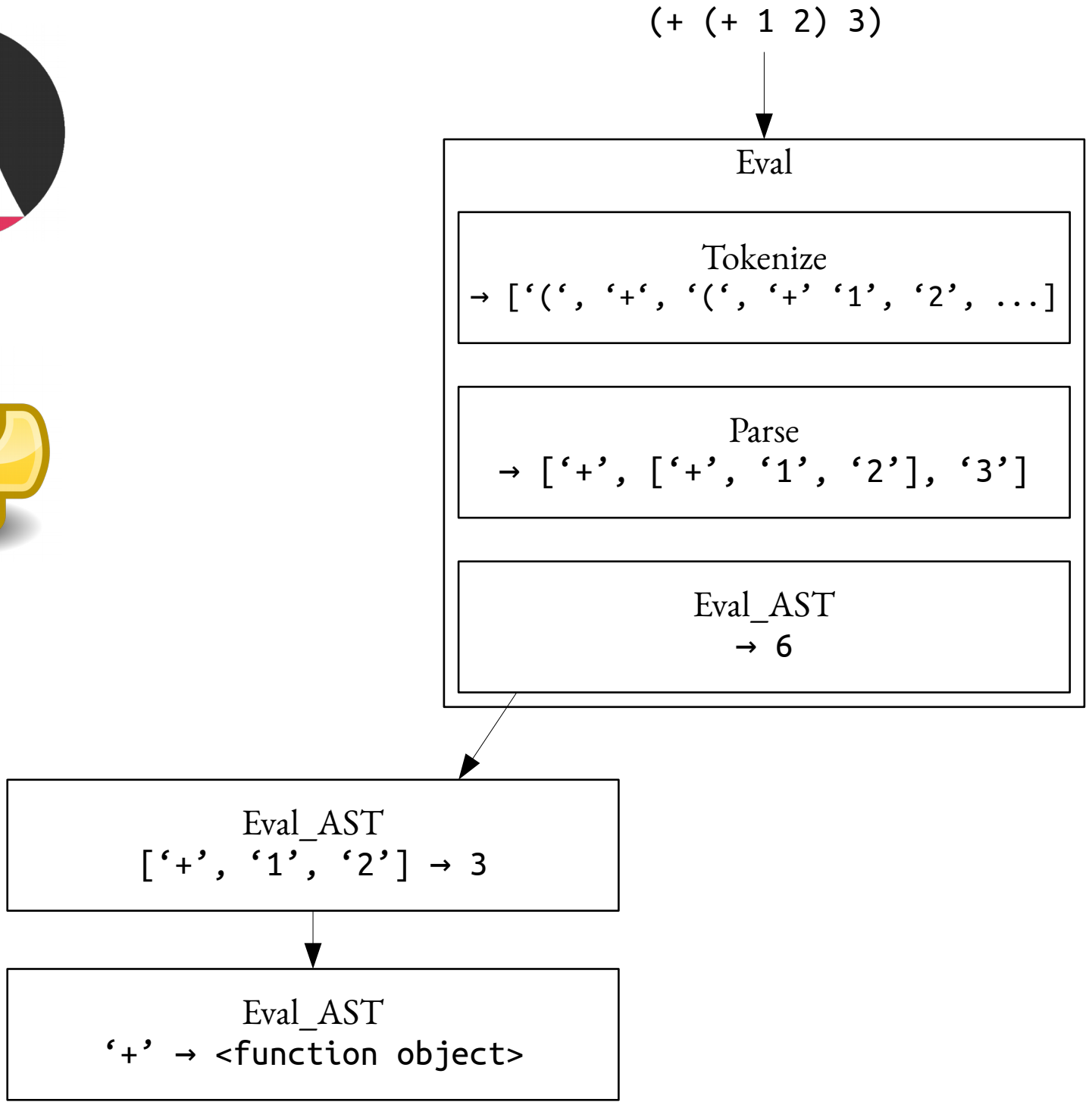
Eval

Tokenize
→ [‘(‘, ‘+‘, ‘(‘, ‘+’ ‘1’, ‘2’, ...]

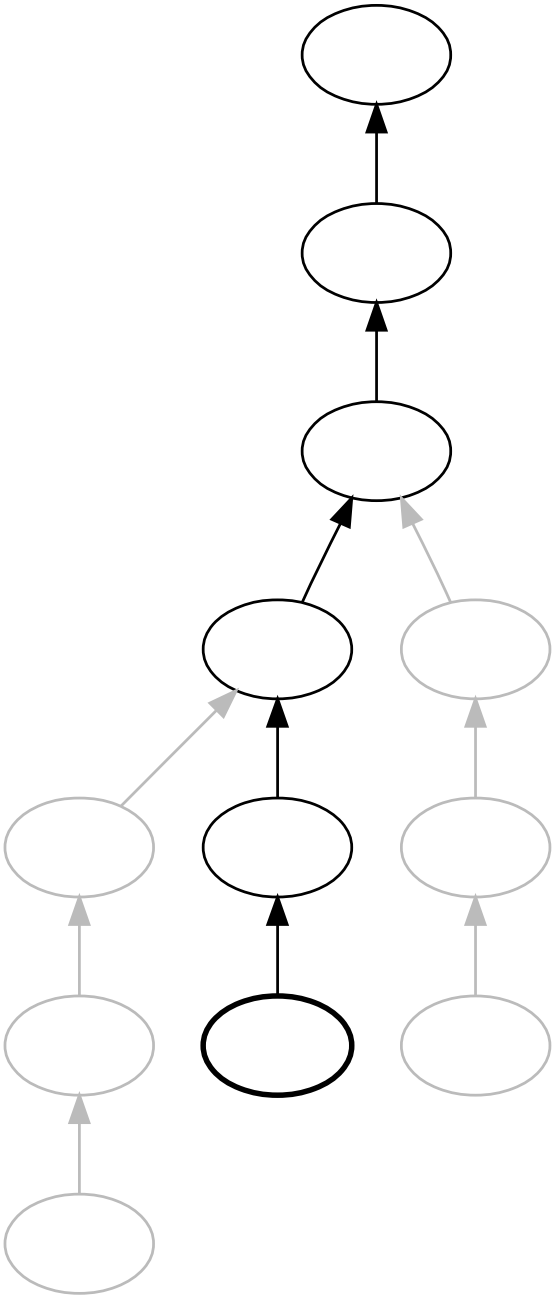
Parse
→ [‘+’, [‘+’, ‘1’, ‘2’], ‘3’]

Eval_AST
→ 6

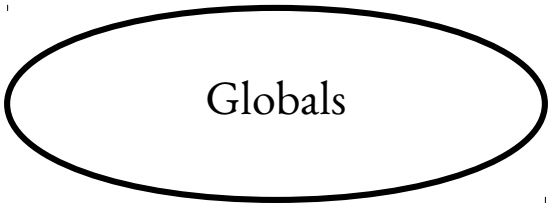

(+ (+ 1 2) 3)
Eval
Tokenize
→ [‘(‘, ‘+‘, ‘(‘, ‘+’ ‘1’, ‘2’, ...]
Parse
→ [‘+’, [‘+’, ‘1’, ‘2’], ‘3’]
Eval_AST
→ 6
Eval_AST
[‘+’, ‘1’, ‘2’] → 3


(+ (+ 1 2) 3)
Eval
Tokenize
→ ['(', '+', '(', '+' '1', '2', ...]
Parse
→ ['+', ['+', '1', '2'], '3']
Eval_AST
→ 6
Eval_AST
['+', '1', '2'] → 3
Eval_AST
'+' → <function object>

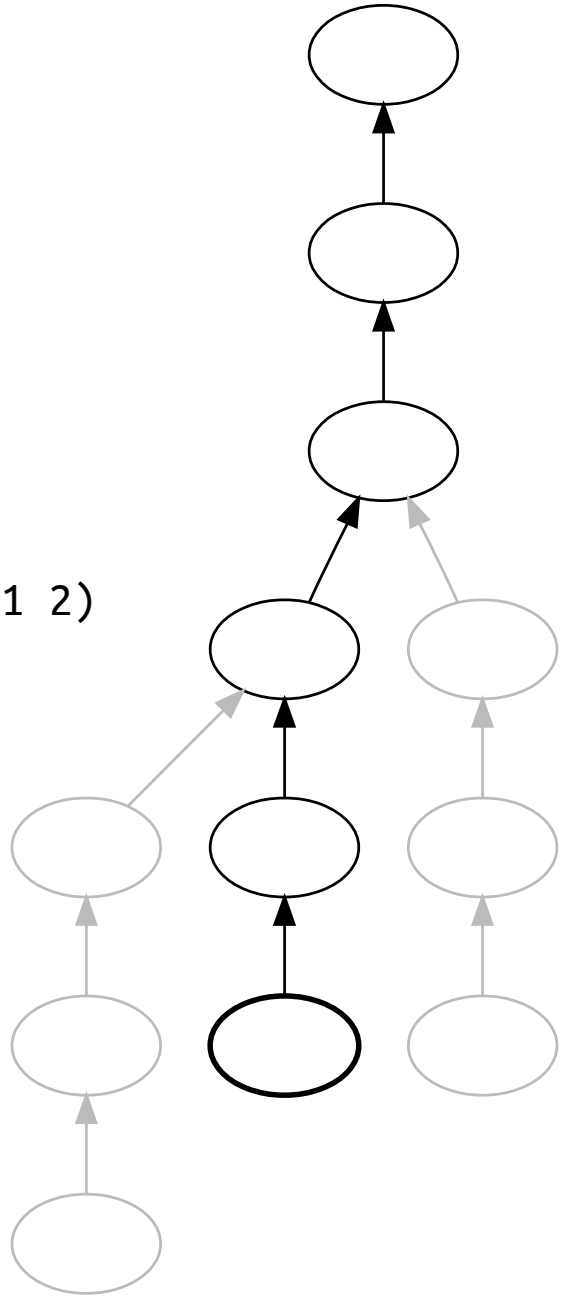
Contexts



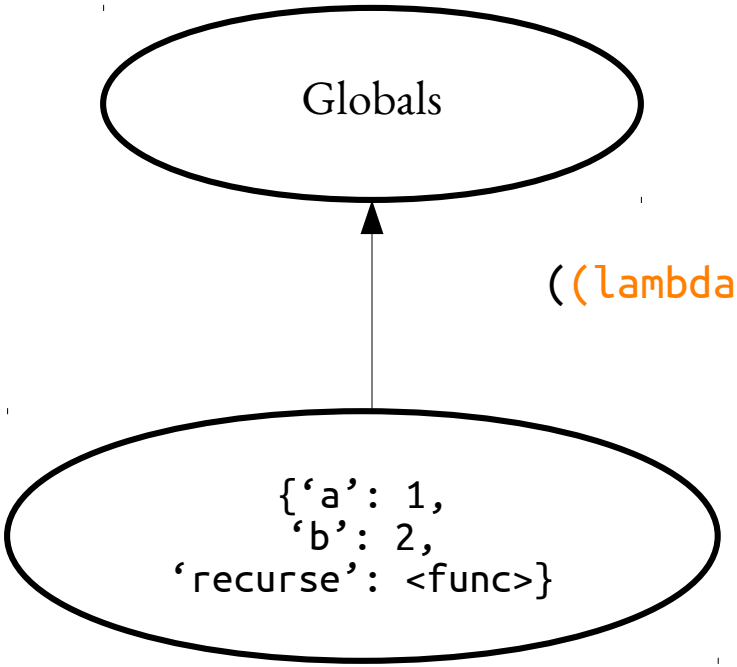
Contexts



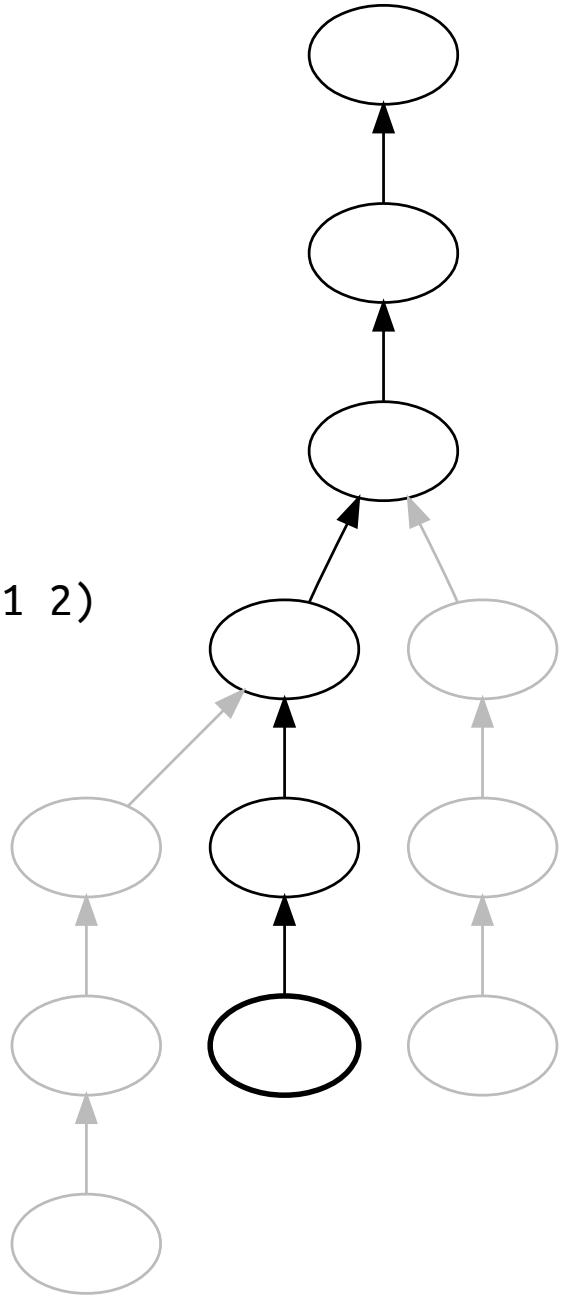

```
((lambda (a b) (+ a b)) 1 2)
```

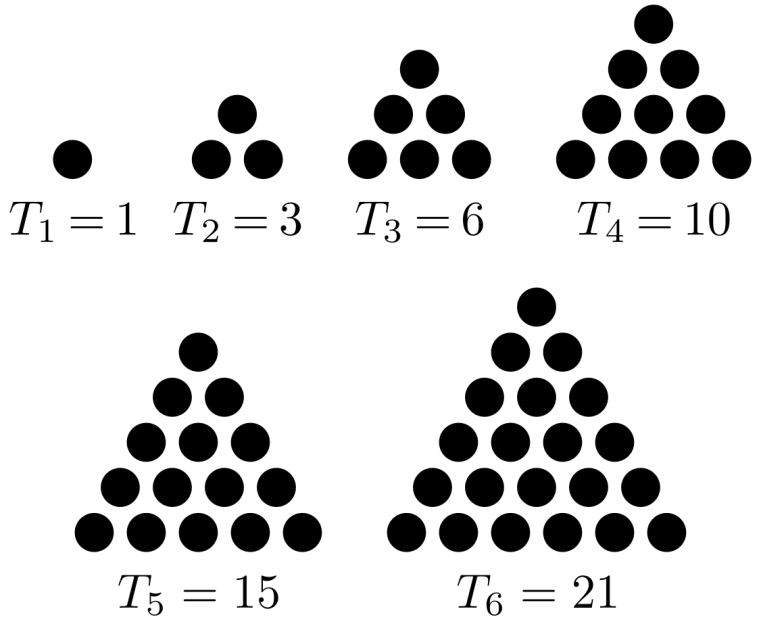


Contexts




```
((lambda (a b) (+ a b)) 1 2)
```

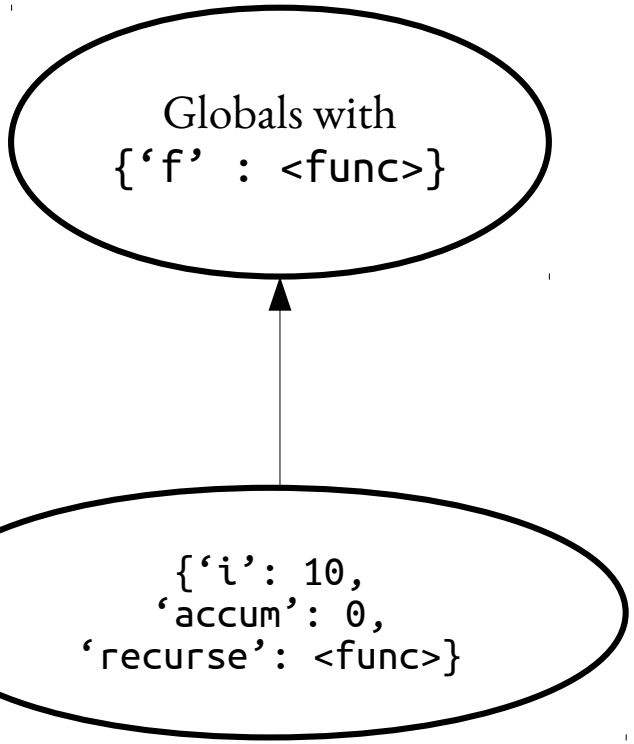


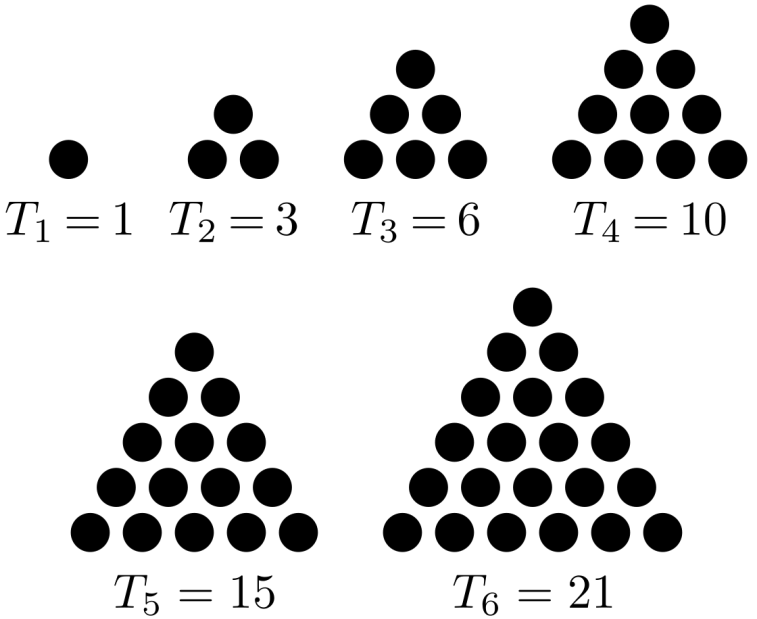


$$T_n = n + T_{n-1} \qquad T_o = o$$

```
(set f (lambda (i accum)
    (if (= 0 i)
        accum
        (f (- x 1)
           (+ accum i)))))

(f 10 0) ↦ 1 + 2 + … + 10
```



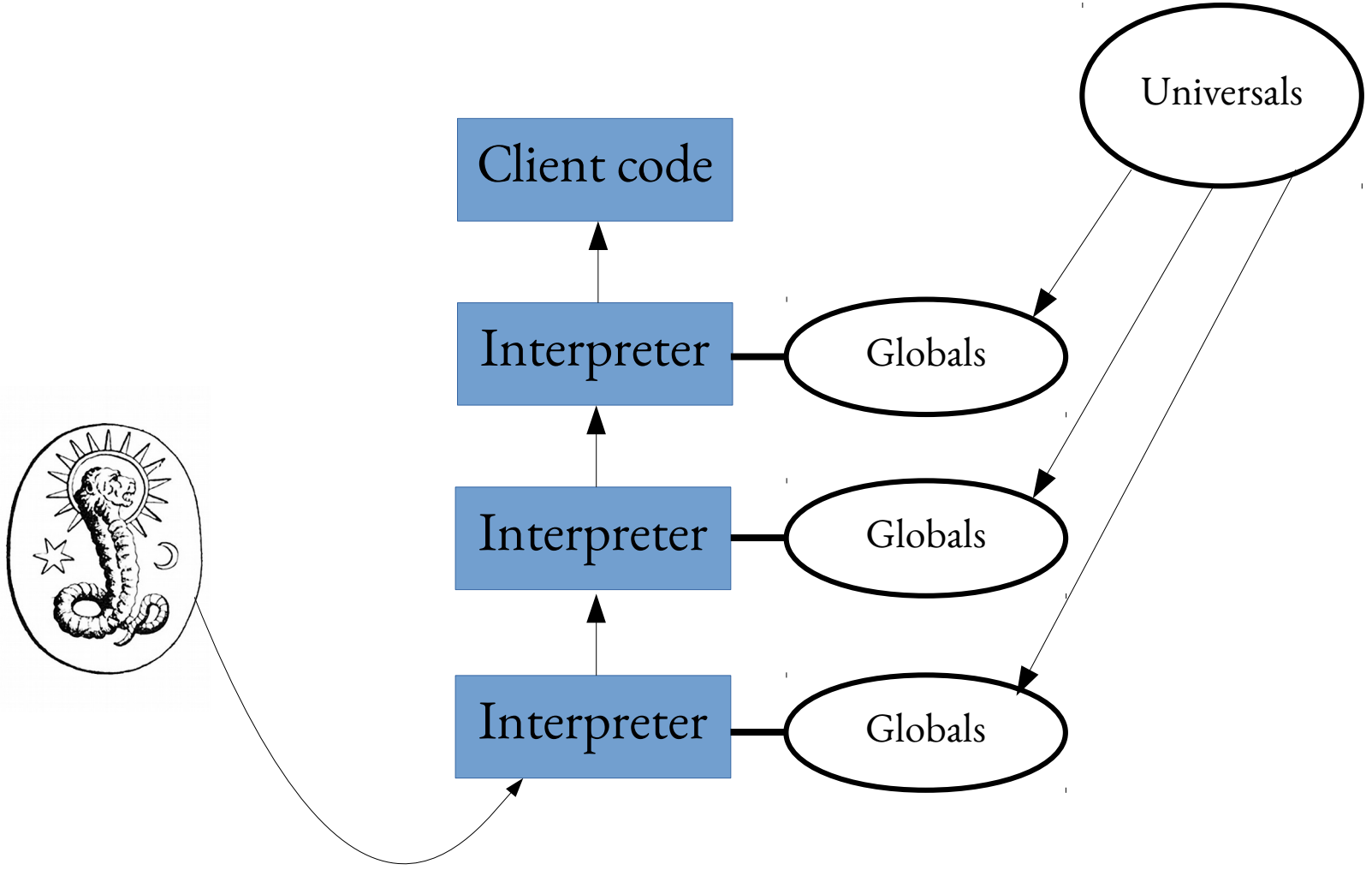


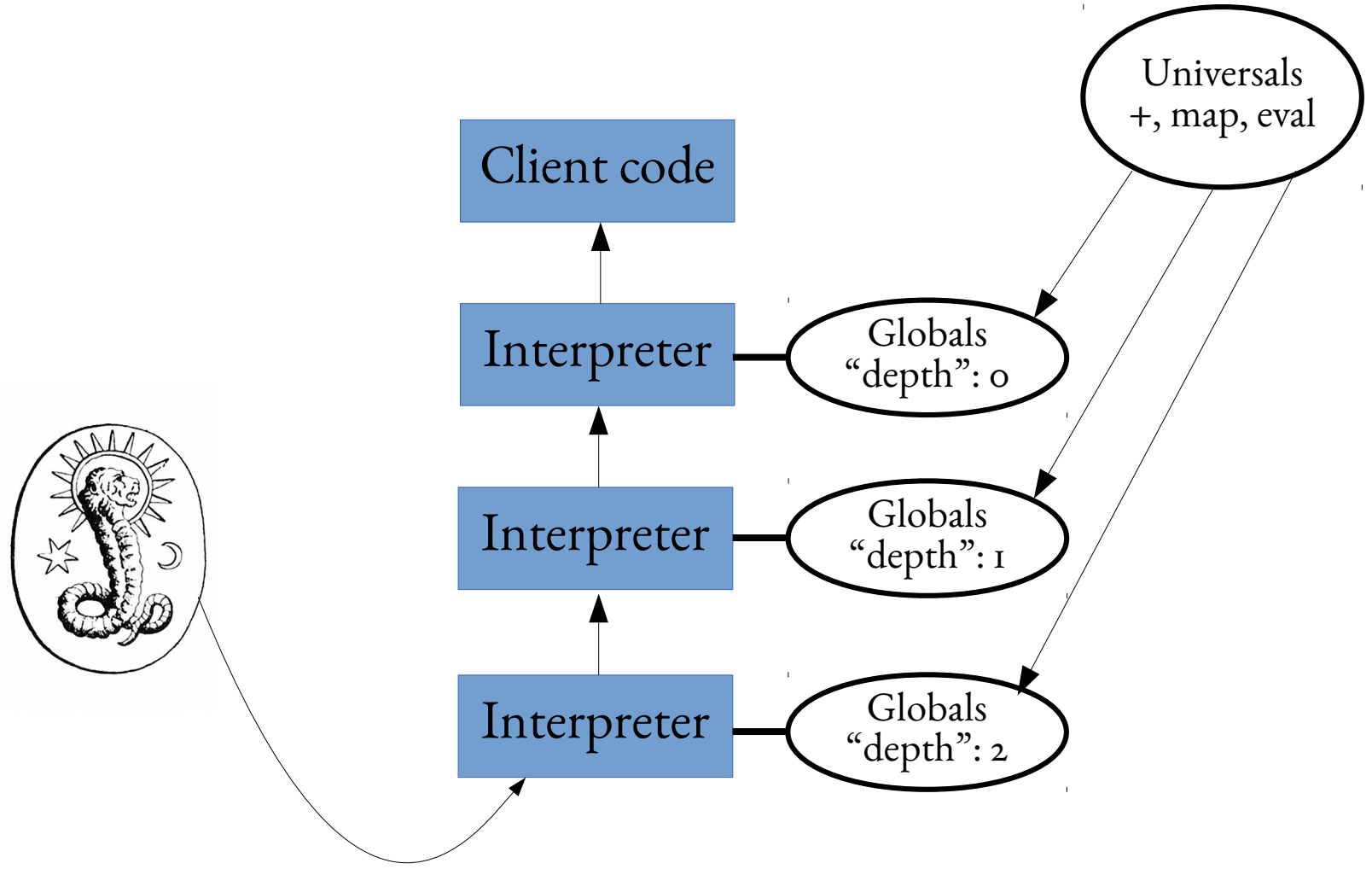
$$T_n = n + T_{n-1} \qquad T_o = o$$

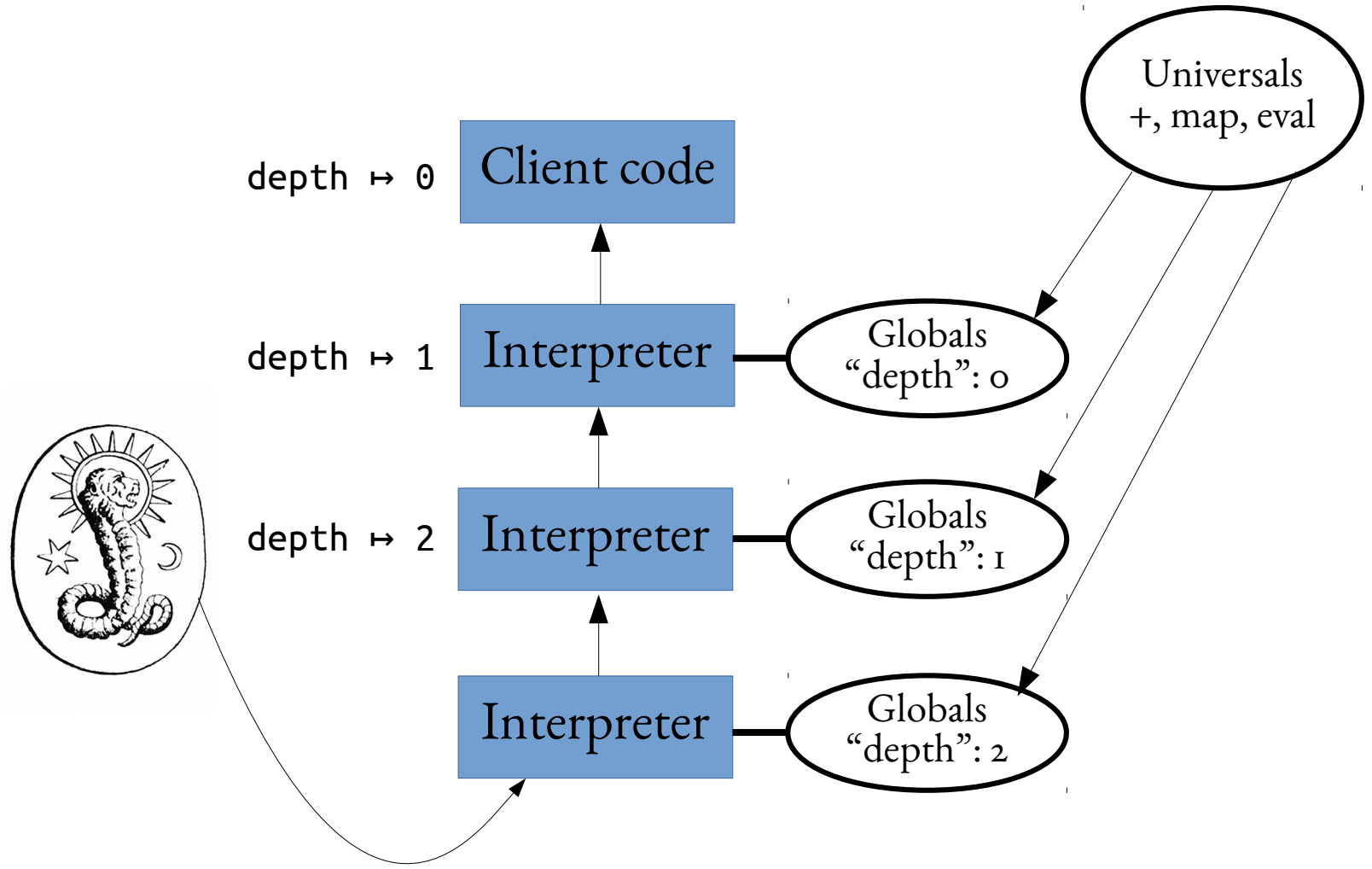
```
(set f (lambda (i accum)
    (if (= 0 i)
        accum
        (f (- x 1)
           (+ accum i)))))
```

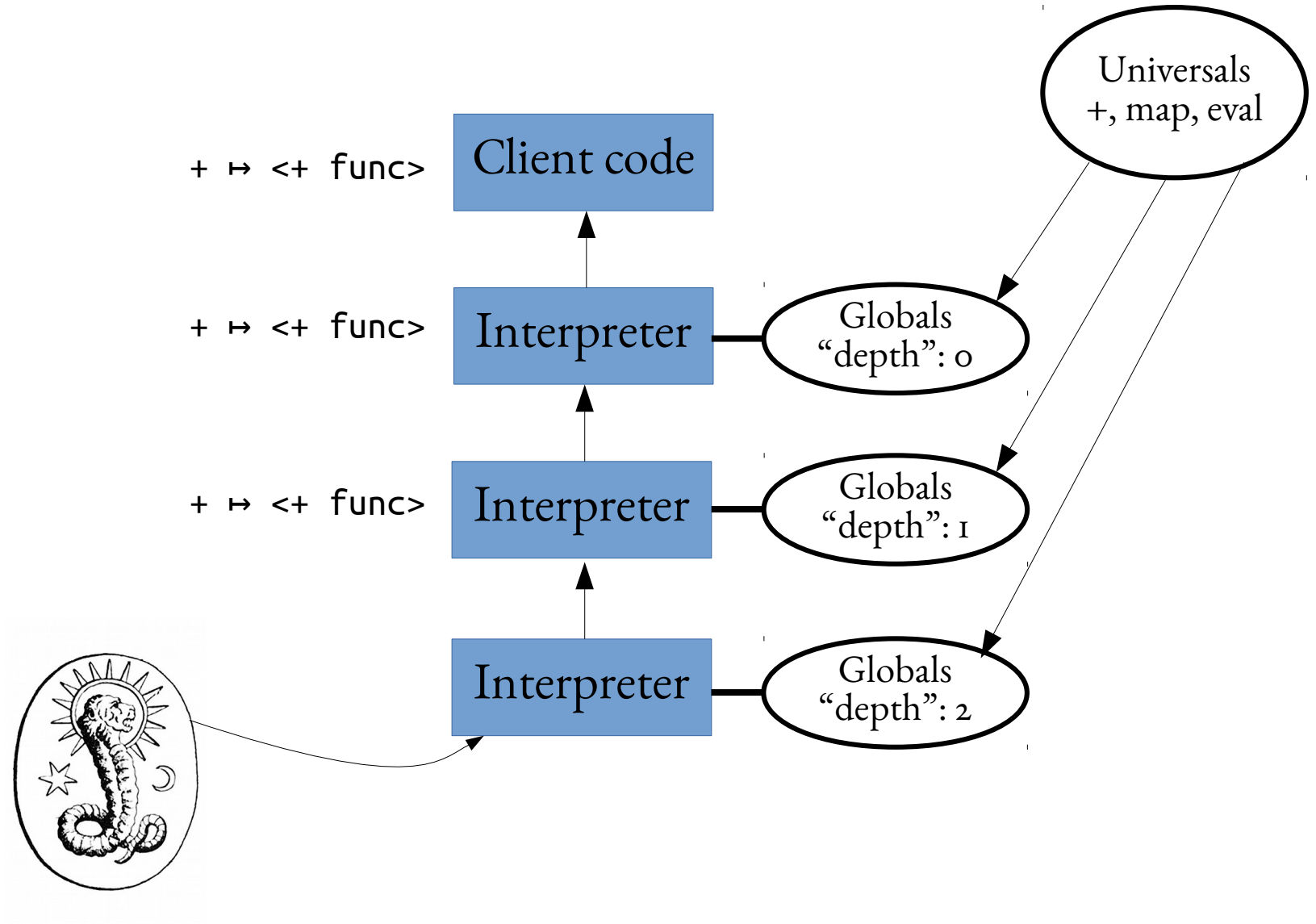
```
(f 10 0) ↦ 1 + 2 + … + 10
```

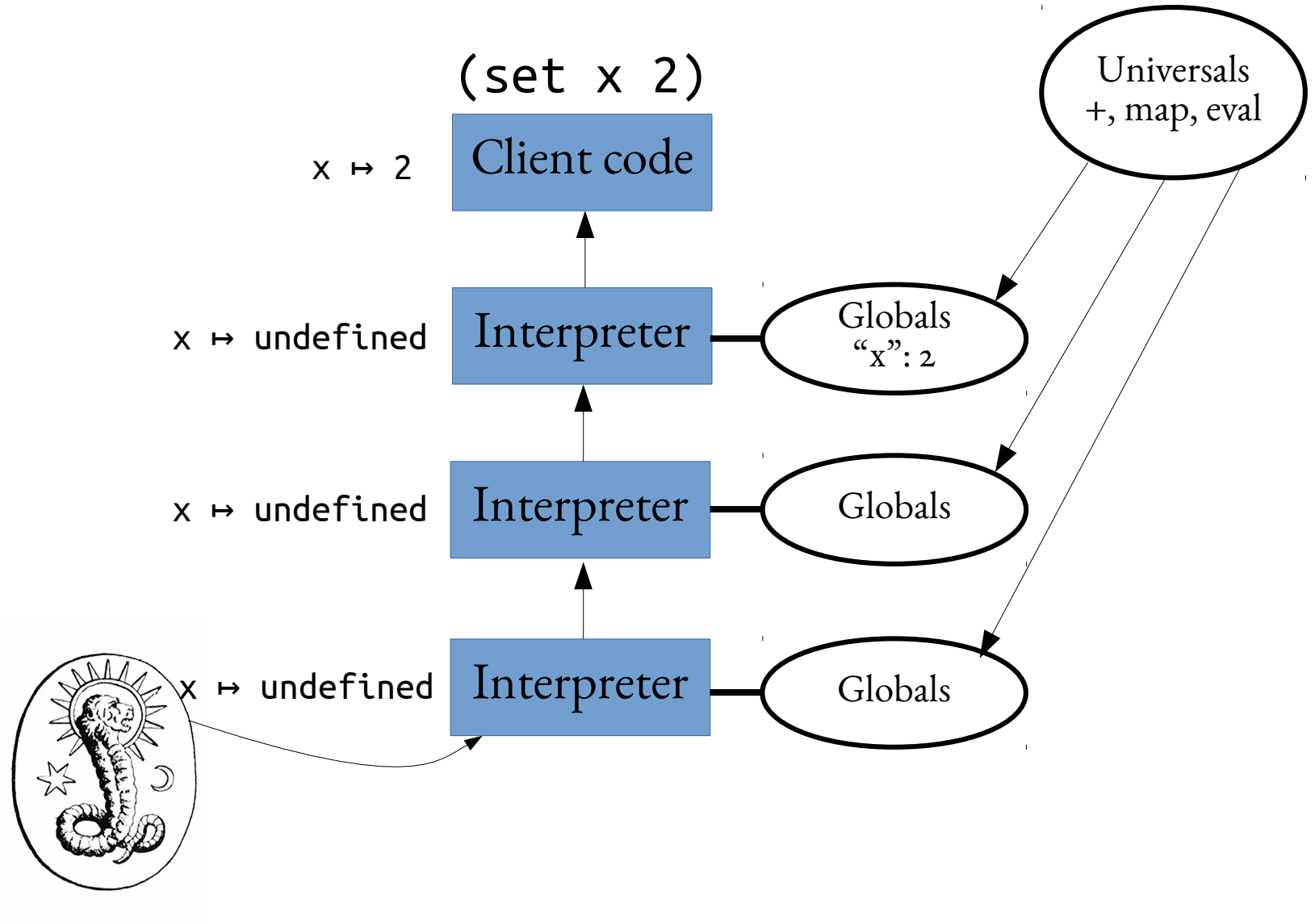
Contexts can change after capture

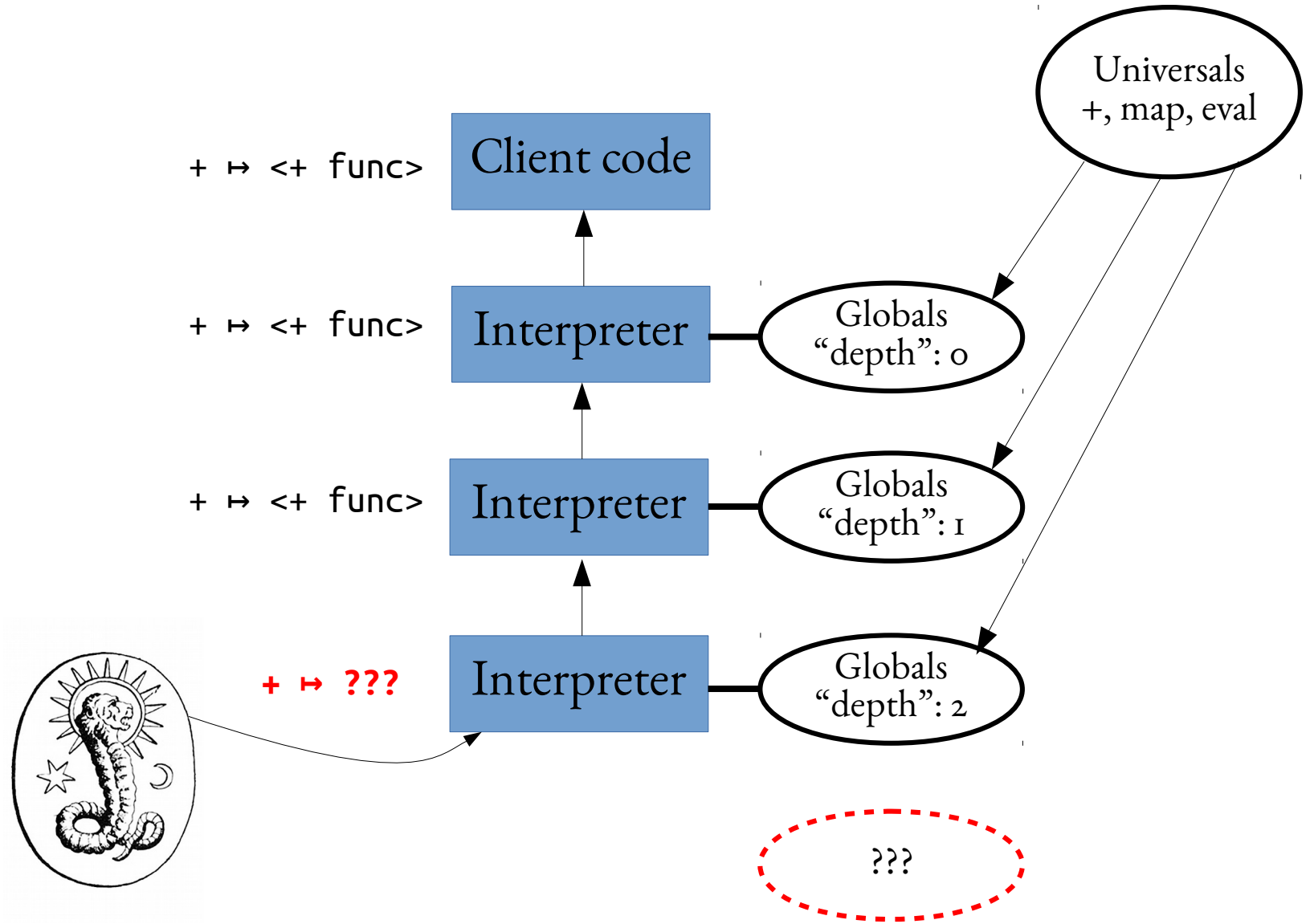

Universals
Client code
Interpreter
Globals
Interpreter
Globals
Interpreter
Globals

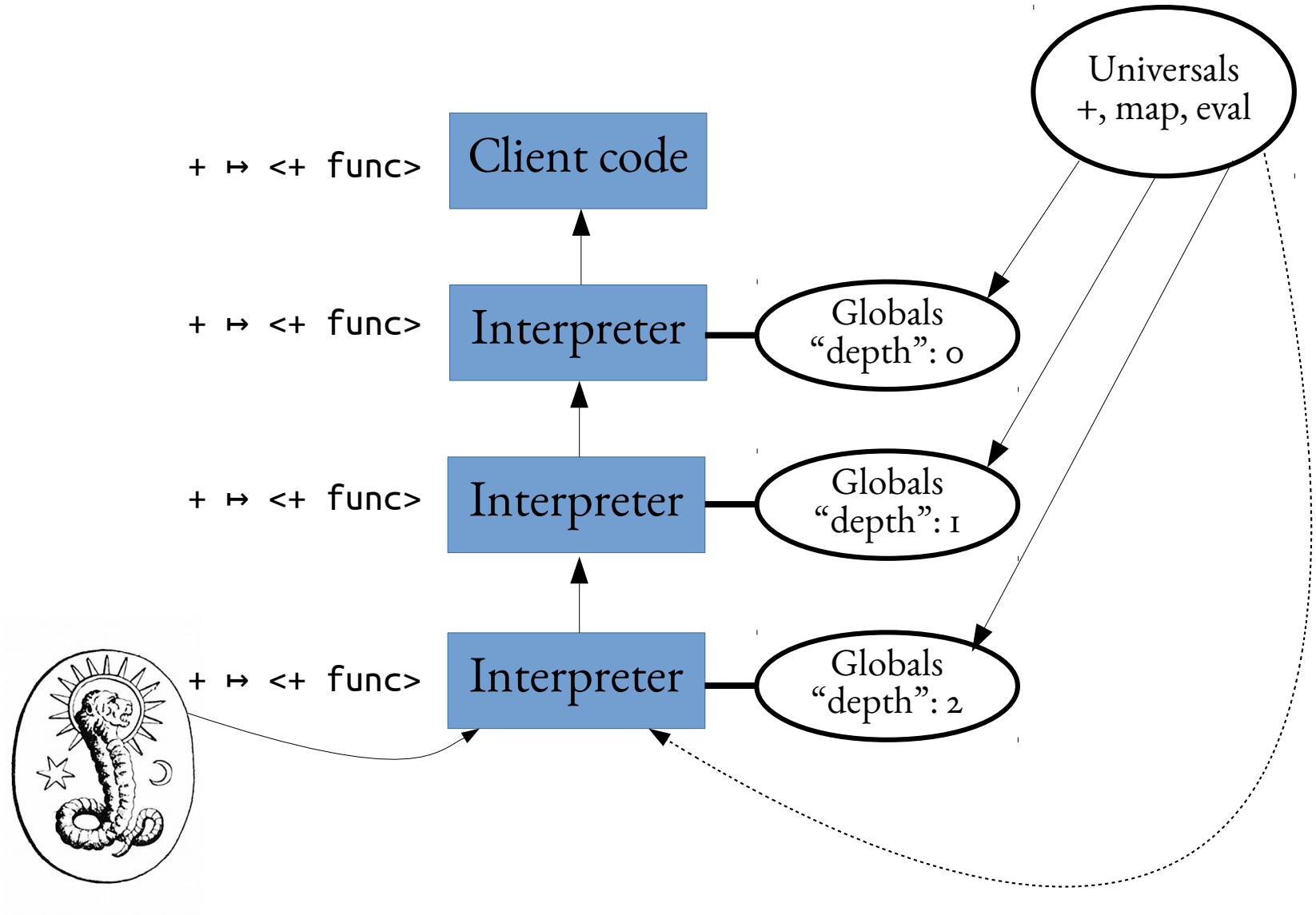

Universals
+, map, eval
Client code
Interpreter
Globals
“depth”: 0
Interpreter
Globals
“depth”: 1
Interpreter
Globals
“depth”: 2

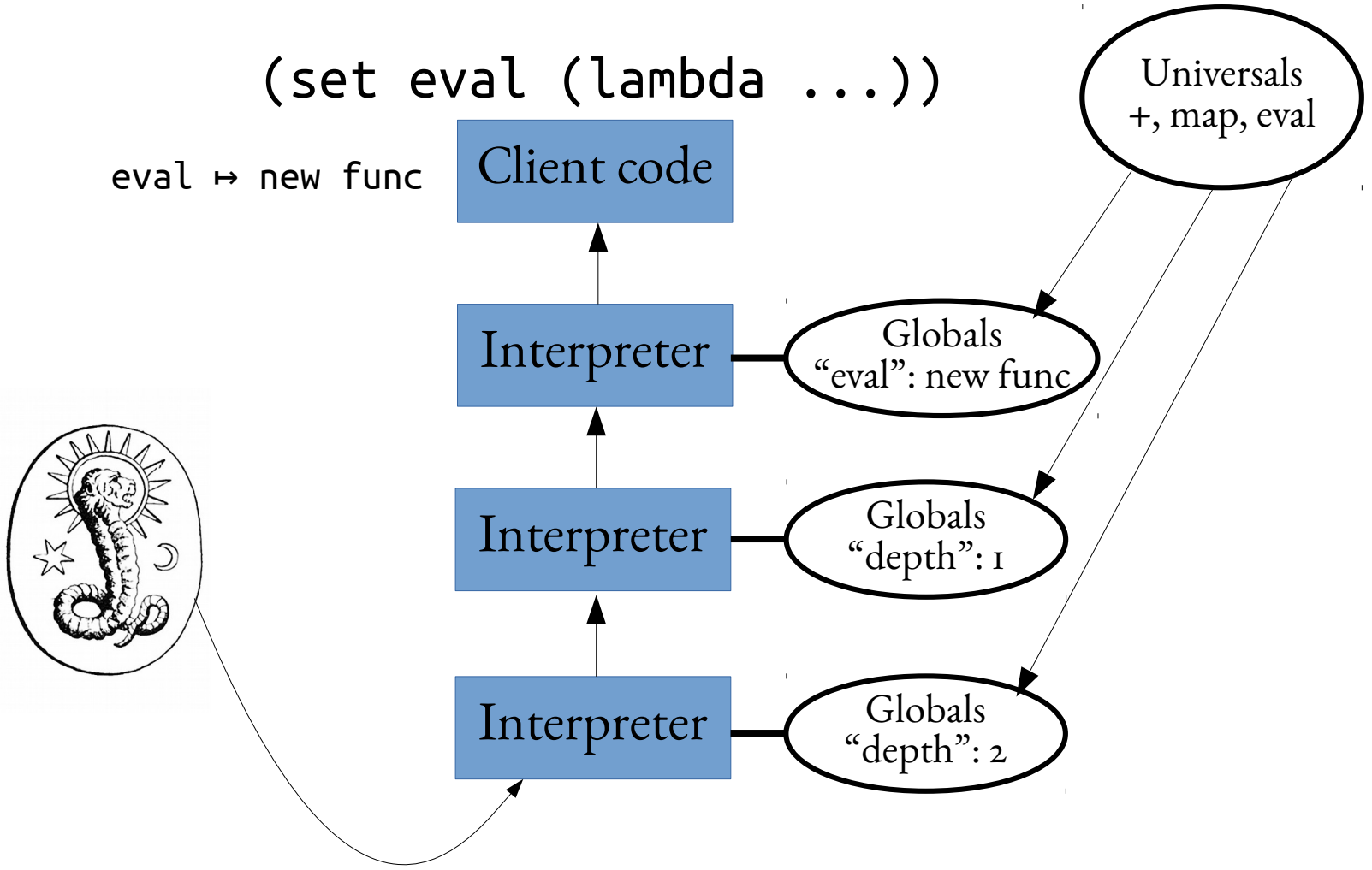

Universals
+, map, eval
depth ↦ 0
Client code
depth ↦ 1
Interpreter
Globals
“depth”: 0
depth ↦ 2
Interpreter
Globals
“depth”: 1
Interpreter
Globals
“depth”: 2


+ ↦ <+ func>
Client code
Universals
+, map, eval
+ ↦ <+ func>
Interpreter
Globals
“depth”: 0
+ ↦ <+ func>
Interpreter
Globals
“depth”: 1
Interpreter
Globals
“depth”: 2


(set x 2)
x ↦ 2
Client code
Universals
+, map, eval
x ↦ undefined
Interpreter
Globals
"x": 2
x ↦ undefined
Interpreter
Globals
x ↦ undefined
Interpreter
Globals


+ ↦ <+ func>
Client code
Universals
+, map, eval
+ ↦ <+ func>
Interpreter
Globals
"depth": 0
+ ↦ <+ func>
Interpreter
Globals
"depth": 1
+ ↦ ???
Interpreter
Globals
"depth": 2
???

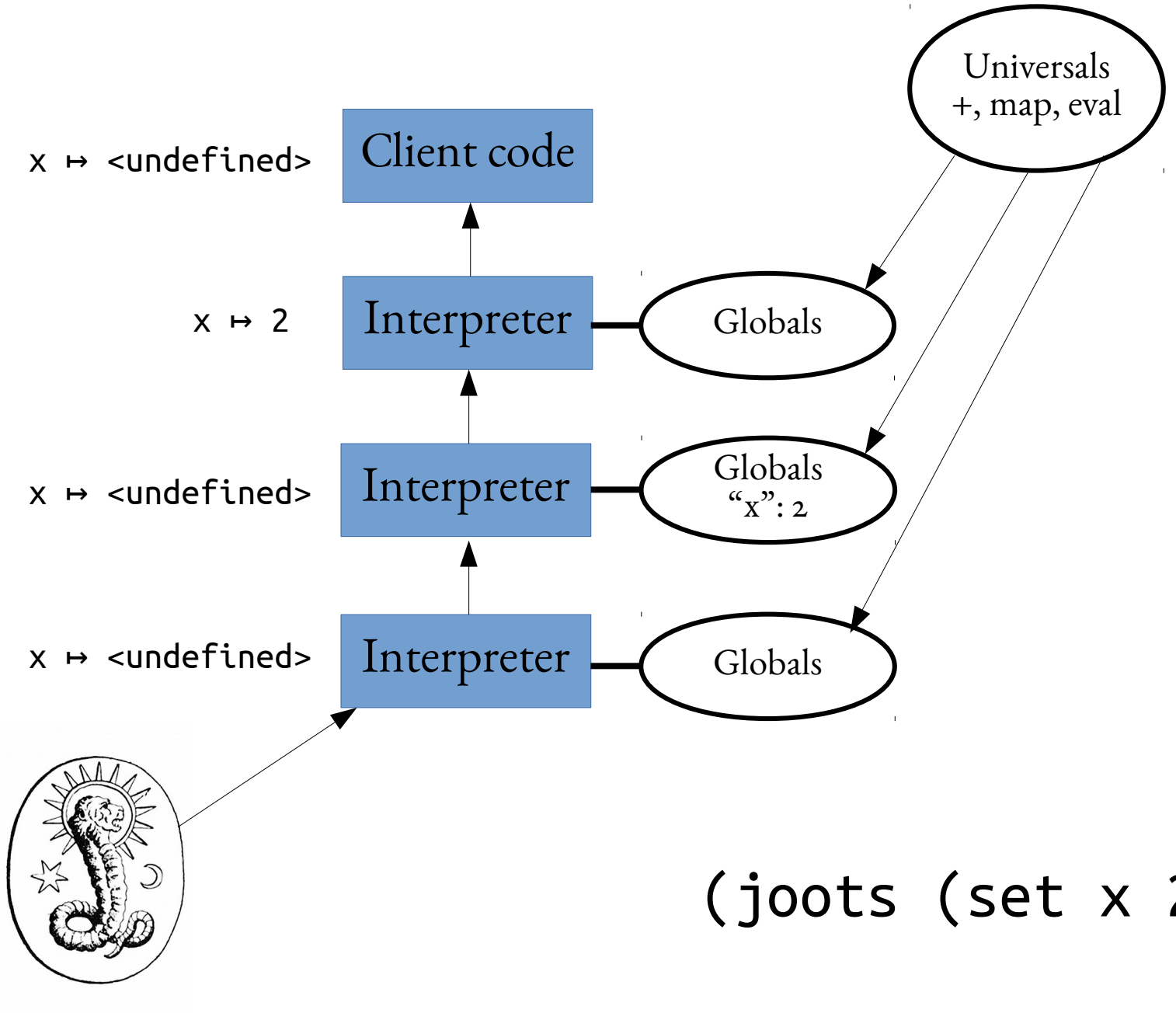

+ ↦ <+ func>
Client code
Universals
+, map, eval
+ ↦ <+ func>
Interpreter
Globals
“depth”: 0
+ ↦ <+ func>
Interpreter
Globals
“depth”: 1
+ ↦ <+ func>
Interpreter
Globals
“depth”: 2


(set eval (lambda ...))
eval ↦ new func
Client code
Interpreter
Interpreter
Interpreter
Universals
+, map, eval
Globals
“eval”: new func
Globals
“depth”: 1
Globals
“depth”: 2

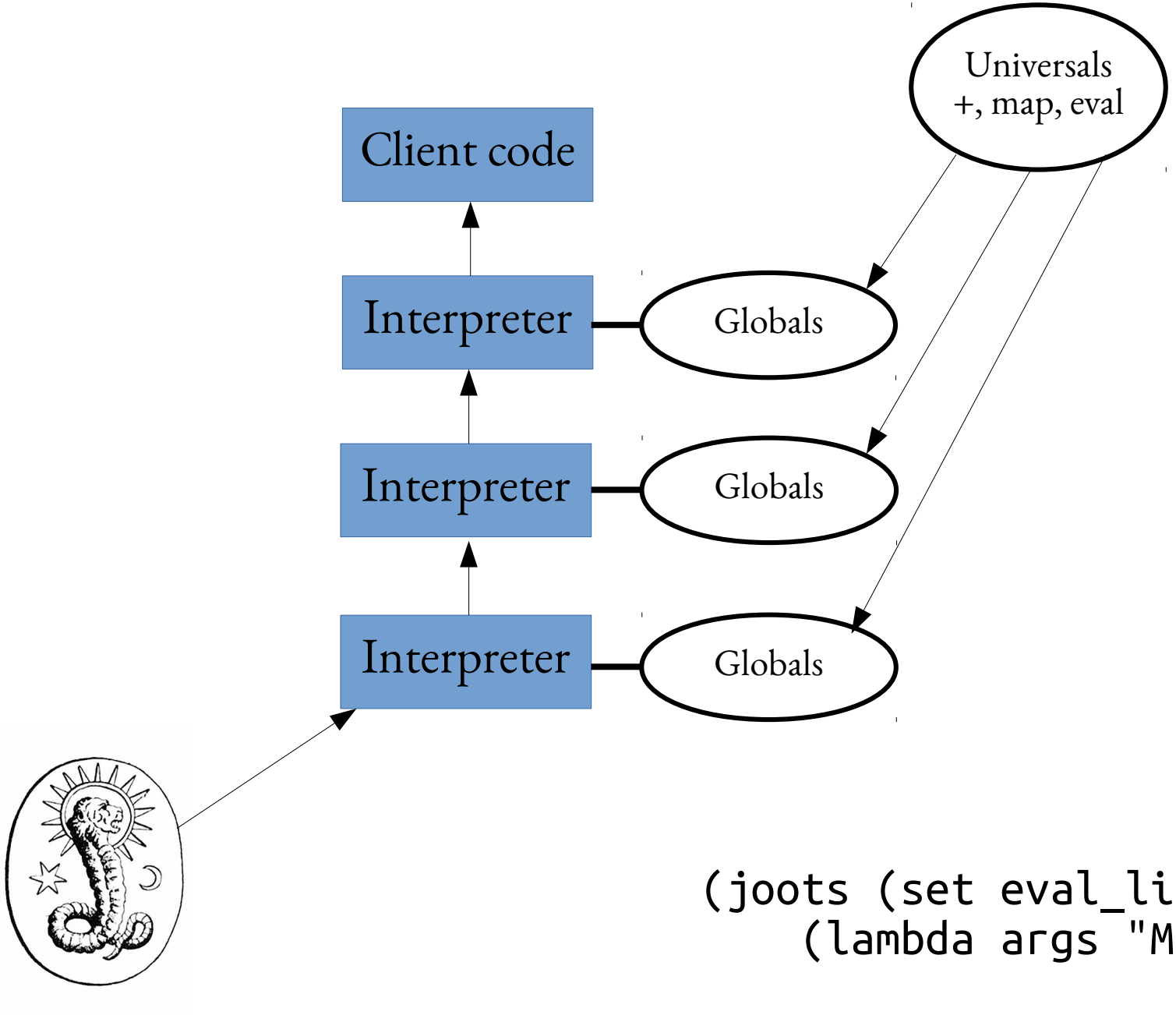
`joots`

"Jump out of the system"

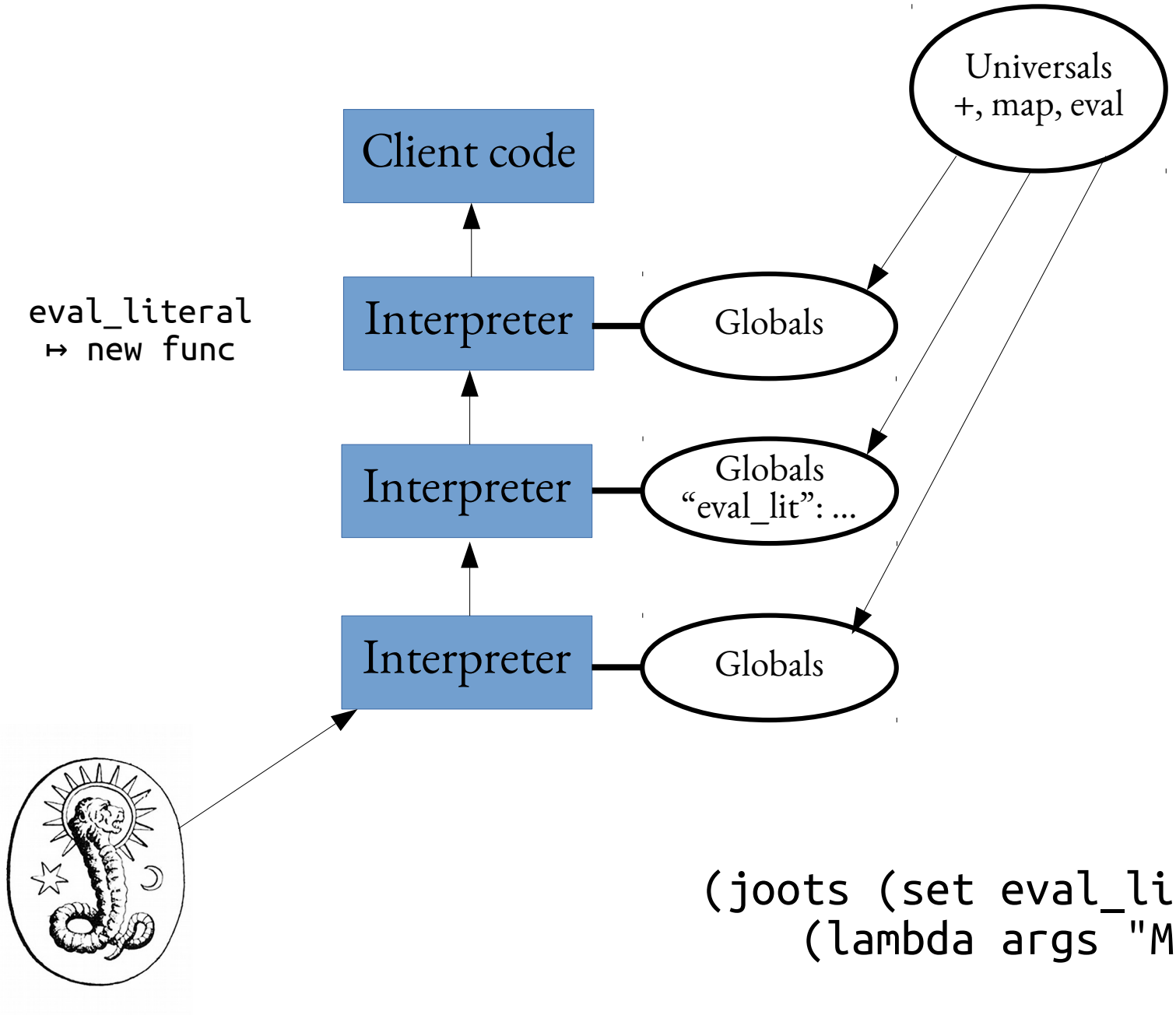
Evaluate code as if executed by the layer interpreting this


Universals
+, map, eval
x ↦ <undefined>
Client code
x ↦ 2
Interpreter
Globals
x ↦ <undefined>
Interpreter
Globals
"x": 2
x ↦ <undefined>
Interpreter
Globals


```
(joots (set x 2))
```



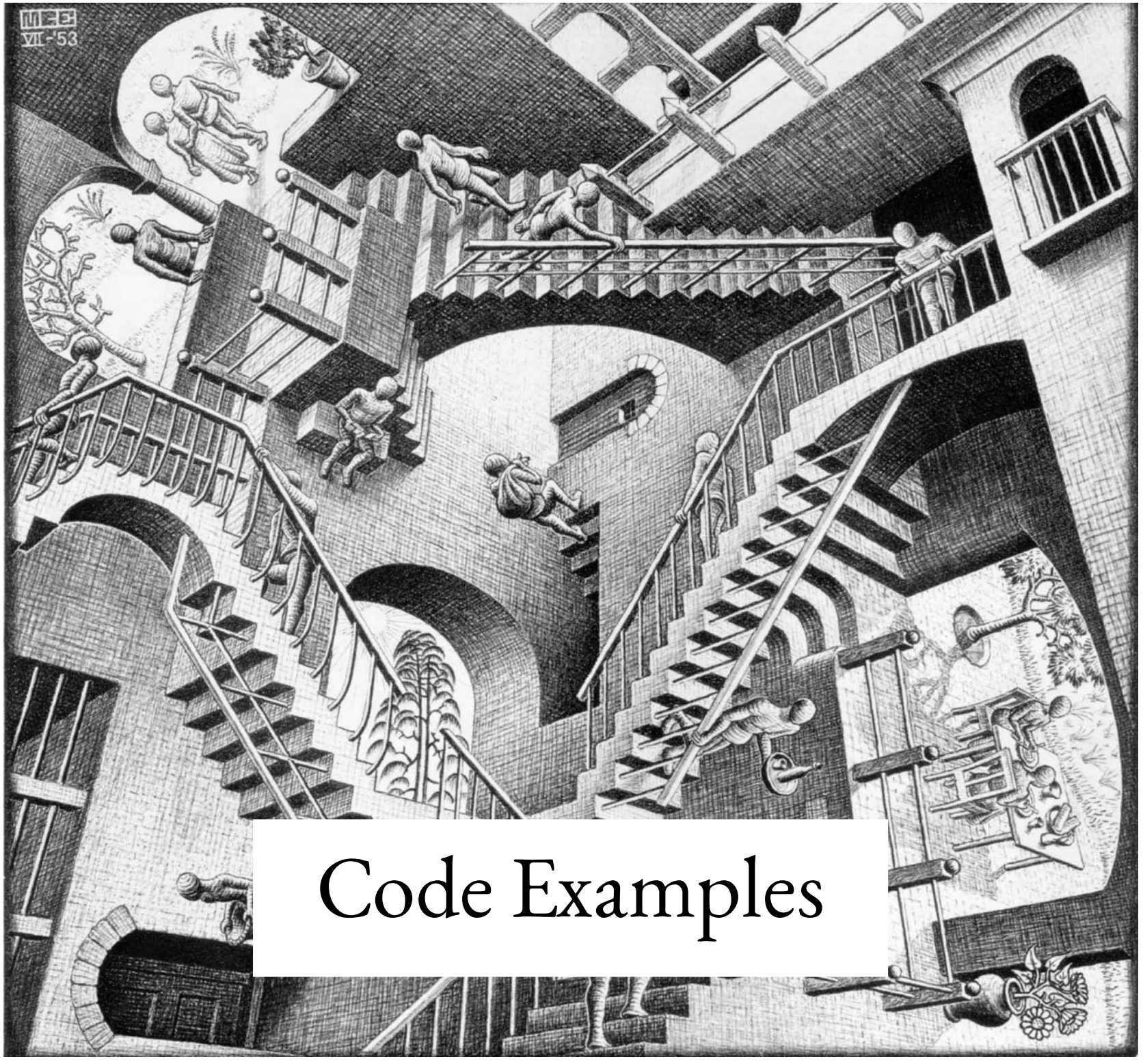

```
(joots (set eval_literal
    (lambda args "Malkovich")))
```




```
eval_literal
 ↦ new func
```

```
(joots (set eval_literal
    (lambda args "Malkovich")))

(+ 1 1) ↦ “MalkovichMalkovich”
```



Code Examples

2:00

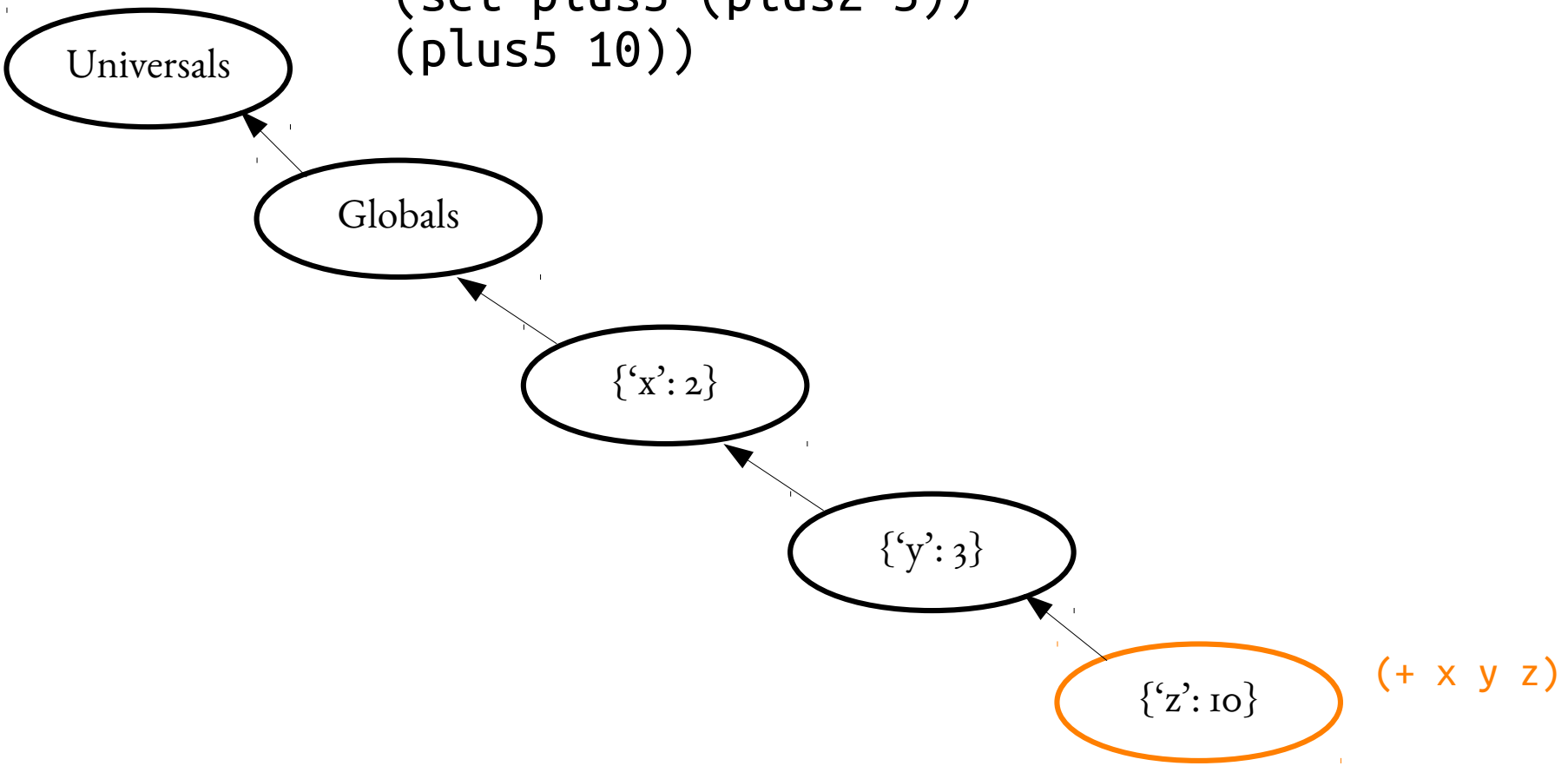
```
(set_index (get_current_context) "x" 10)
```

```
x ↦ 10
```

Analogous to:

```
locals()['x'] = 10
```

```
(begin
   (set curriedplus
      (lambda (x)
         (lambda (y)
            (lambda (z)
               (+ x y z)))))
   (set plus2 (curriedplus 2))
   (set plus5 (plus2 3))
   (plus5 10))
```



```
(set i 0)
(set inc (lambda ()
   (begin
      (set_global i (+ i 1))
      (print i))))
(inc) ; prints 1
(inc) ; prints 2
```

```
(set_universal descender
    (floatinglambda ()
        (begin
            (print "DESCENDER IS AT DEPTH" DEPTH)
            (joots (descender)))))
(descender)
```

```
DESCENDER IS AT DEPTH 60
DESCENDER IS AT DEPTH 61
DESCENDER IS AT DEPTH 62
DESCENDER IS AT DEPTH 63
DESCENDER IS AT DEPTH 64
DESCENDER IS AT DEPTH 65
DESCENDER IS AT DEPTH 66
DESCENDER IS AT DEPTH 67
DESCENDER IS AT DEPTH 68
DESCENDER IS AT DEPTH 69
EXCEPTION: maximum recursion depth exceeded
```

```
(set_universal new_tokenize
    (lambda (s interpreter) (begin
        ; first remove comments
        (set s
            (remove_comments s INTERPRETER))
        ; then parse into tokens
        (set tokens
            (regex_findall (+ "\{|\}|[^\'{}\s]+|\'[^\']*\'") s))
        ; then replace the [] with (), to plug into parse_tokens properly
        (set tokens
            (map
                (lambda (x) (if
                    (= x "{")
                        "("
                        (if (= x "}")
                            ")"
                            x)))
                tokens))
        (` tokens #f))))

(joots (set tokenize new_tokenize))

;FINISH-BEFORE

{print {+ 1 {* 2 3}}}
```

Q & A