CS Town Hall: Thurs (tomorrow), 3:30 pm, AND 329

Today: - end of term scheduling
- last assignment / letrec
- types

```
;; Try the above with a recursive function
(let ((fact (lambda (n)
              (if (equal? n 1)
                  1
                  (* n (fact (- n 1)))))))
  (fact 3))
```

closure points to global frame

unable to find fact in recursive calls.

global frame

broadly (details matter), but letrec moves the pointer

letrec version

P

P

fact

frame

params: n

code:

(if (eq? ----)

(fact 3)

apply closure

P n → 3

return

(* 3 (fact (- 3 1)))

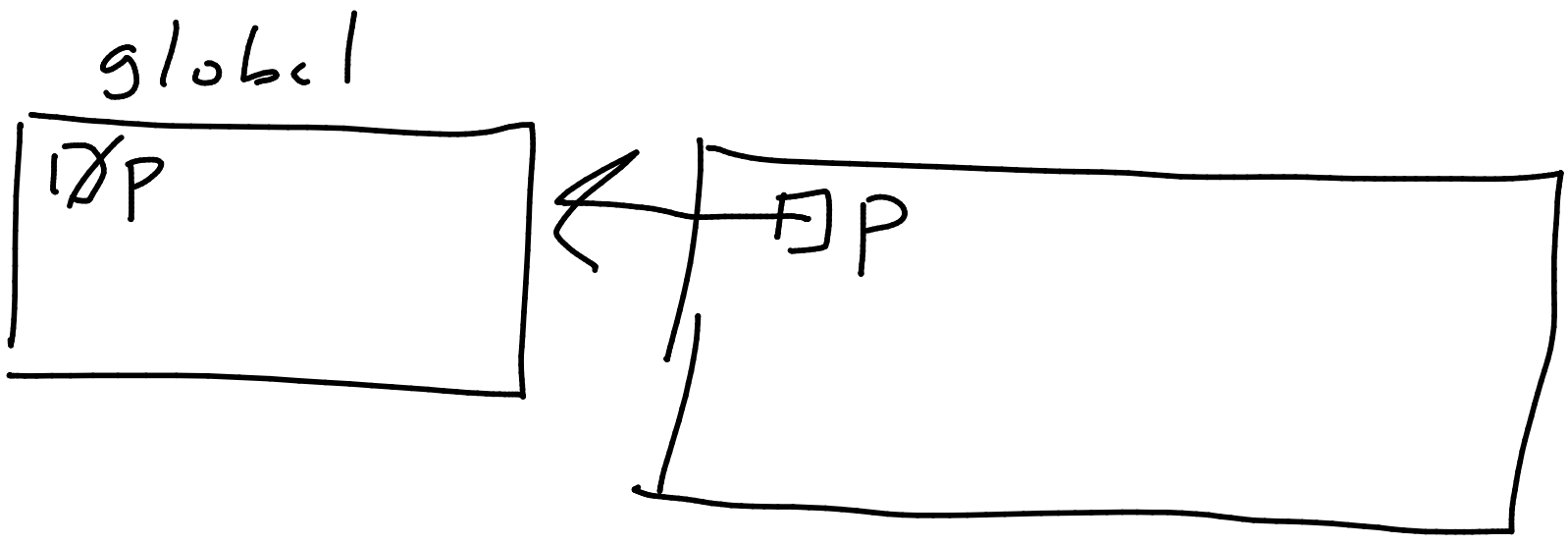
2

Can't find fact

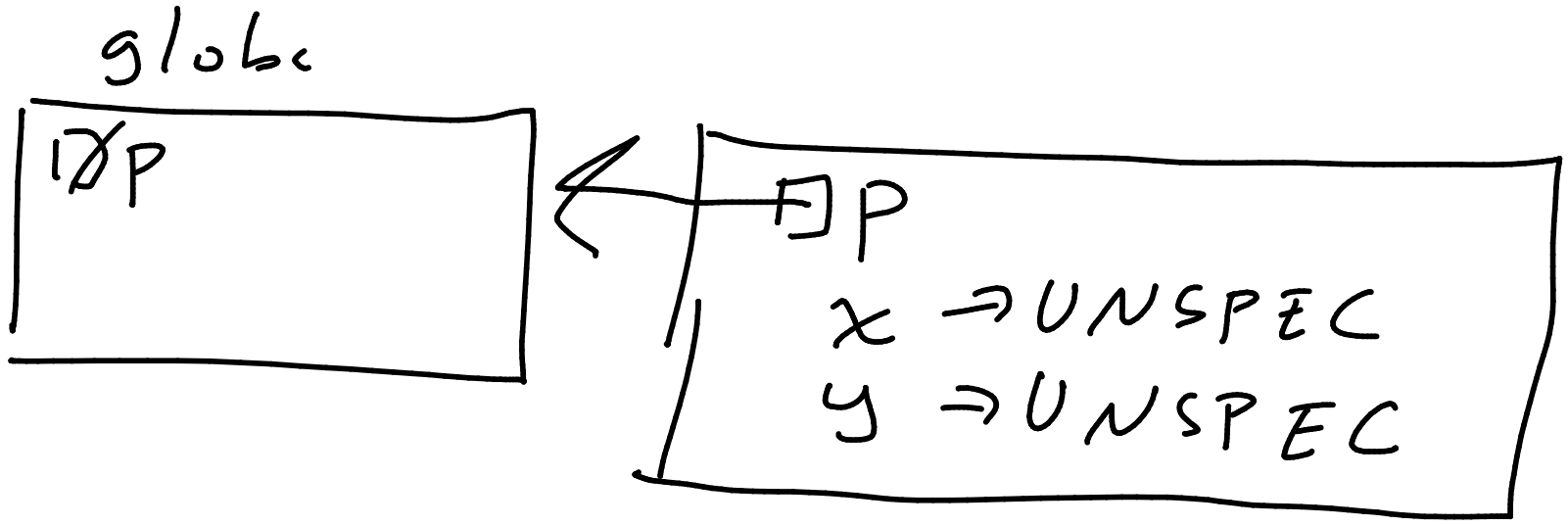
What does letrec actually do?

```
(let ((x 3) (y 5))
  (+ x y))
```

① Creates a new frame as usual



② Add the vars to the new frame, but assign them to UNSPECIFIED_TYPE

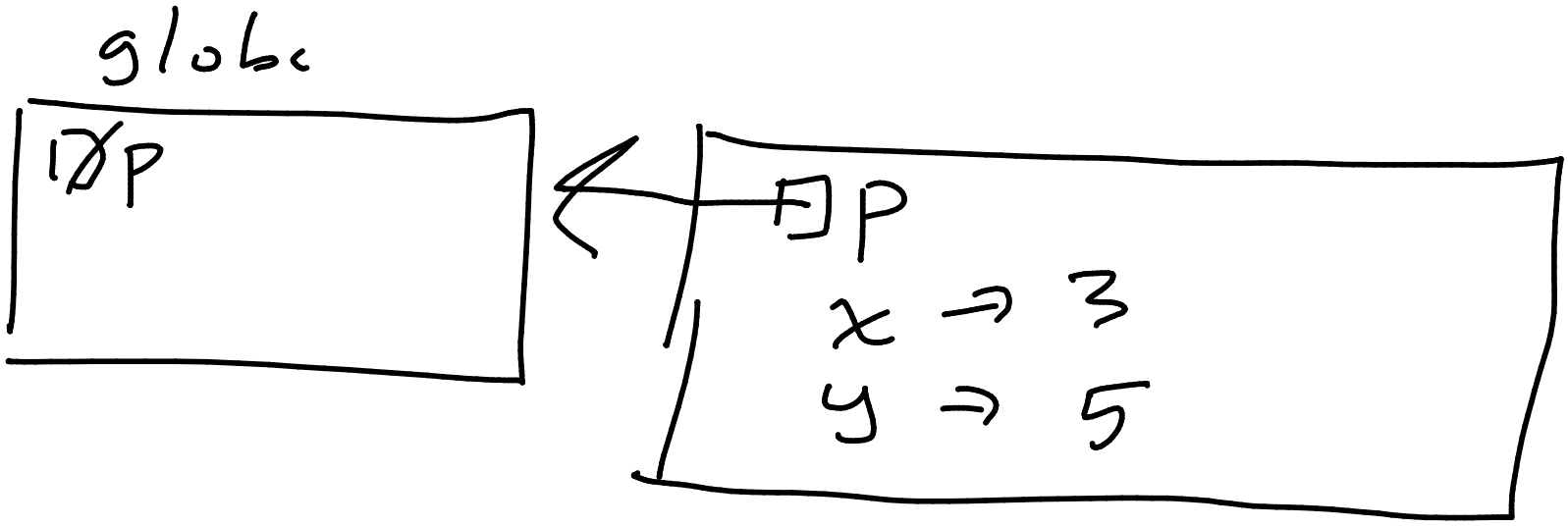


(3) Evaluate the right sides of each pair of bindings, and store somewhere temp. If anything uses an UNSPEC, throw an error. (In the new frame)

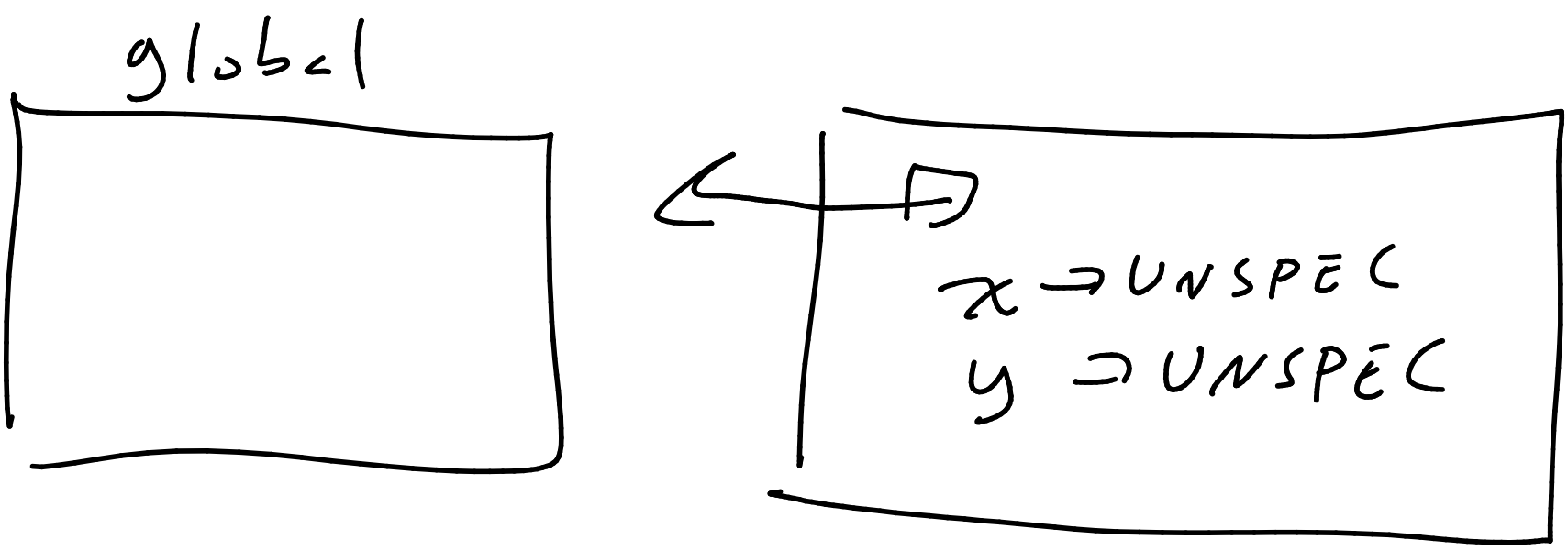
eval 3 ⟶ 3 remember

eval 5 ⟶ 5 remember.

No errors? Great! Now write these into new frame.



(letrec ((x 3) (y (+ x 1)))
  (+ x y))



eval 3 ⇒ 3

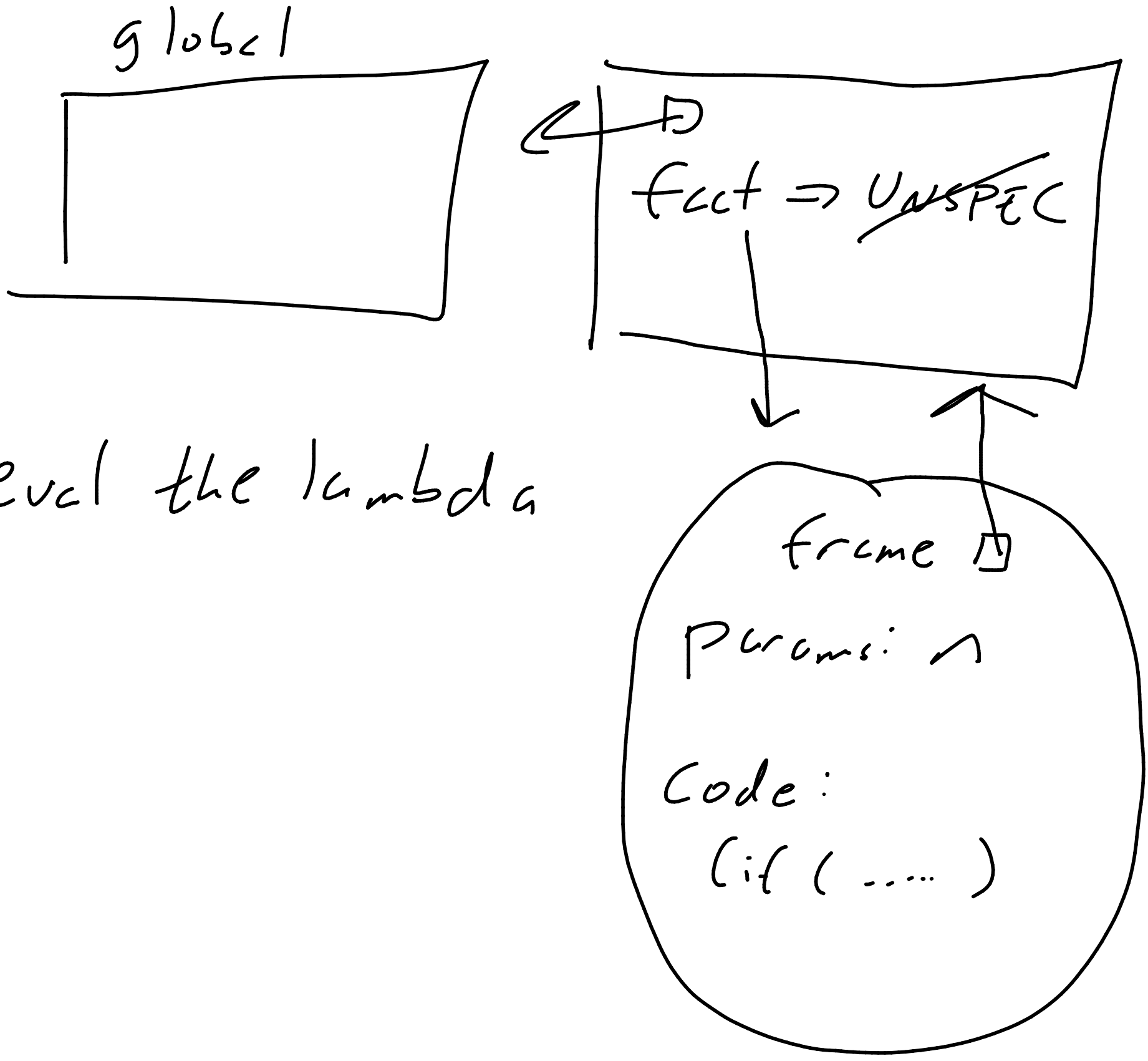
eval (+ x 1) oh no, x is UNSPEC

⇒ error.

```
;; Try the above with a recursive function
(let ((fact (lambda (n)
              (if (equal? n 1)
                  1
                  (* n (fact (- n 1)))))))

  (fact 3))
```

letrec



test61.scm (E-tests)
due to Donald Knuth

—

Way, way back he wrote a letter
to the editor of a PL journal
and said (in different words)
- all these ALGOL interpreters stink.
They don't do eval right.
Here's a test I wrote to
distinguish the good ones
from the bad ones.

One specific example Guile gets
wrong(!)

(letrec ((x 3) (y x))
x)

not in frame