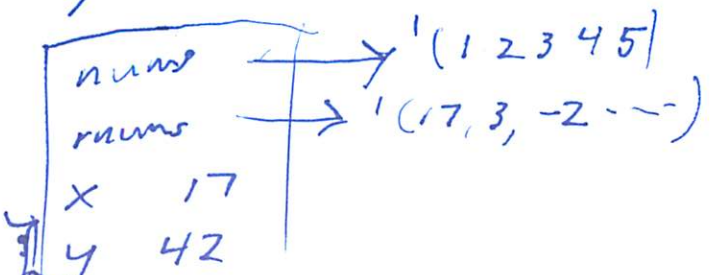
Top-level env (global)



evaluation always relative to env

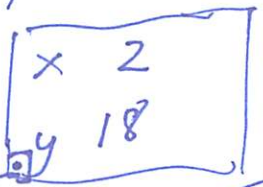
(+ 1 x) – top-level

⇒ 18

(+ 1 rabbit) – not defined

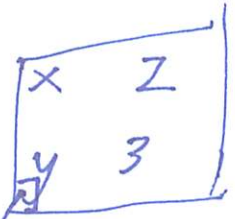
(let ([x 2]
[y (+ 1 x)])
(+ x y))

env is new
+ parent is top-level



⇒ 20

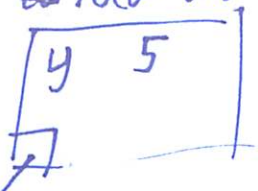
(let ([x 2]
[y (+ 1 x)])
(+ x y))



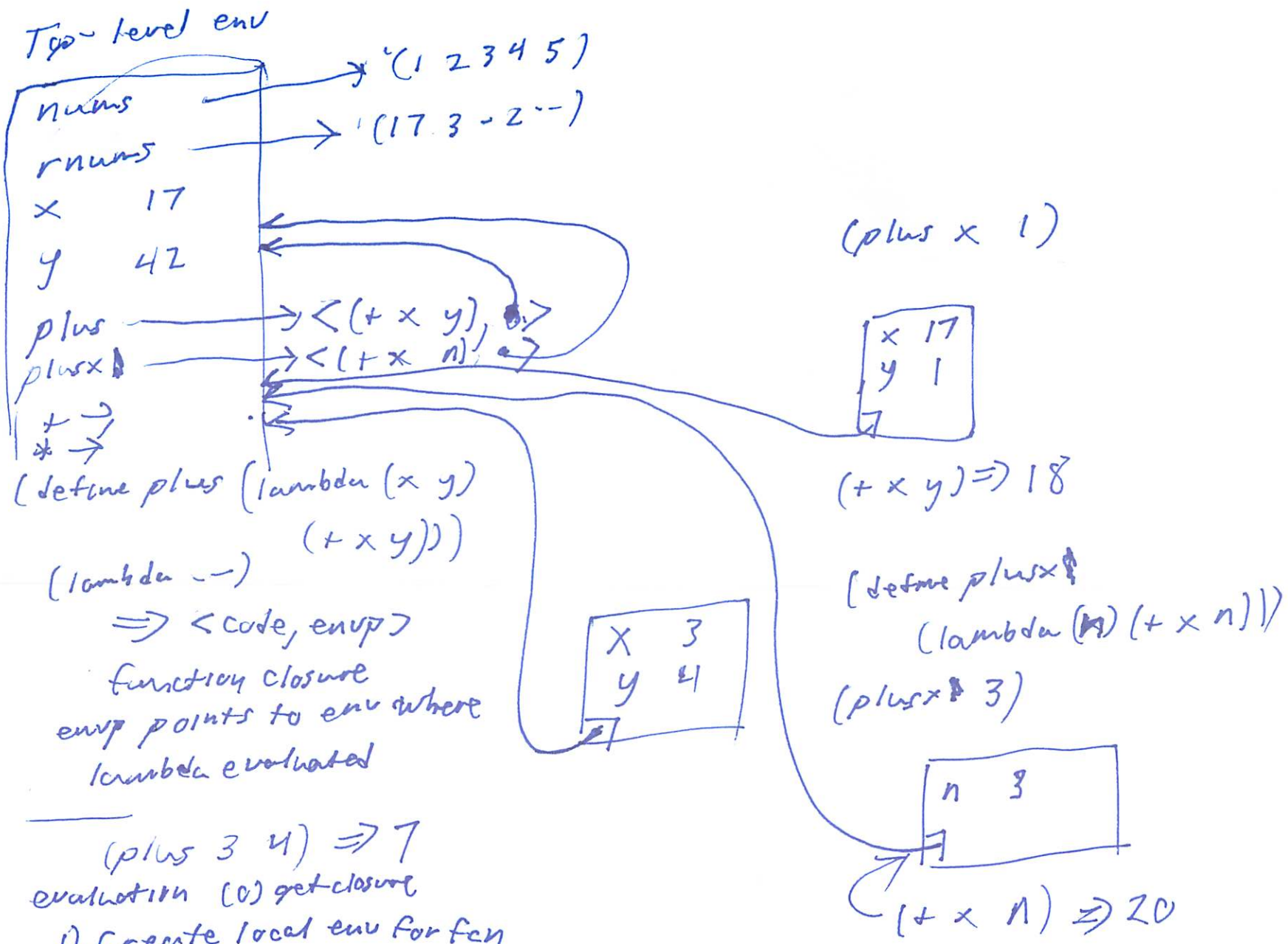
⇒ 5

(let ([y 5])
(+ x y)

local env



⇒ 22


Top-level env
'(1 2 3 4 5)
'(17 3 -2 --)
nums
rnums
x 17
y 42
plus
plusx
+
*
(define plus (lambda (x y)
(+ x y)))
(lambda --)
⇒ <code, envp>
function closure
envp points to env where
lambda evaluated
(plus x 1)
x 17
y 1
(+ x y) ⇒ 18
(define plusx
(lambda (n) (+ x n)))
(plusx 3)
n 3
(+ x n) ⇒ 20
x 3
y 4
(plus 3 4) ⇒ 7
evaluation (c) get closure
1) Create local env for fcn
env parent is the envp
from closure (maybe not current-env)
2) Bind local names to values
3) eval func. code from closure
in relative to new env.