Parsing
Scoping
If/let assignment

Parsing -

Remind how to handle

(define x '(a b c))
↑ ↑ ↑ ↑ ------

stack
(
define

x
x
(
a
b
c

a b c

normally, we would push onto stack

but ... ' is there, so pop that

wrap whats there (quote (a b c))

quote (a b c)

Reminder:

'(a b c) is just shorthand for

(quote (a b c)).

I saw "special casing" individual symbol vs lists

'a
'(a b c)

} rather than handle separately

... just redefine "push on the stack" as "before push on the stack, see if a single quote is there, if it is, Pop it, then wrap in (quote ...) and push"

For scope.scm, if it displayed a 2, is that static or dynamic scoping?

(A) (B)

What does Scheme actually do?

A (static scoping

Static scoping - a variable can be determined based on structure of program (typically by nested blocks of some sort)

Dynamic scoping - based on order of execution of code

Why was it originally fairly popular in early prog languages, and what were the problems?

- intuition
- simpler to build

Why was it a bad idea (often)?

- To debug fun2 you need to find x. To find it, need to know all possible paths through the code that call fun2 and what variables x they create as they get there.

- With static scoping, if you ask "which x"? There's only one answer, you can find it by looking at the code directly

- Dynamic scoping still lives w/ us
  - Bash does
  - Perl does both, depending on how you declare the variable

Next interpreter assignment - actually going to be interpreting Scheme code

Following parser - fork from the code you've written (going individual)

How do we implement <u>let</u> in Scheme? (statically scoped)

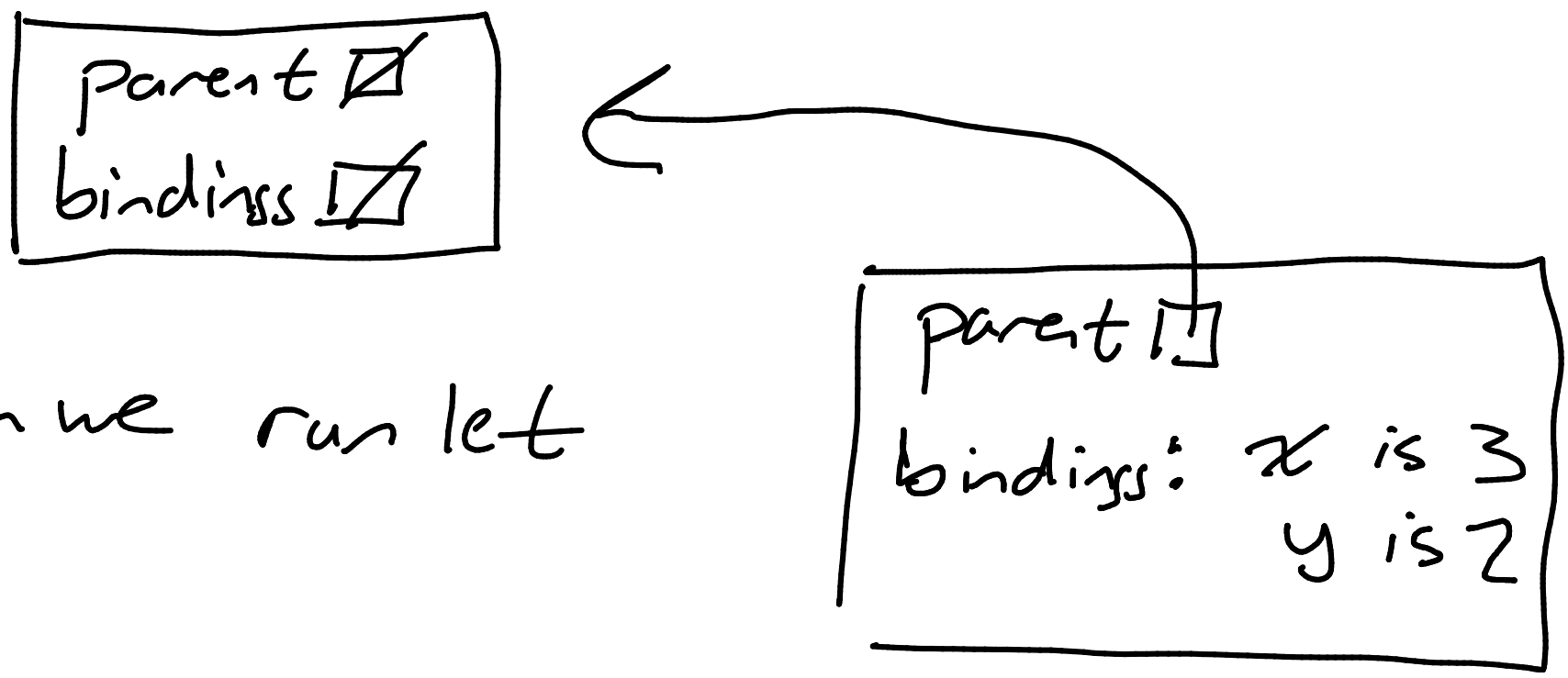
We store local variables in a <u>frame</u>.

A frame is just

- a list of variables and their values
- a pointer to a parent frame, which represents the enclosing block of code.

(let ((x 3) (y 2))
  x) → 3

One frame that represents global variables



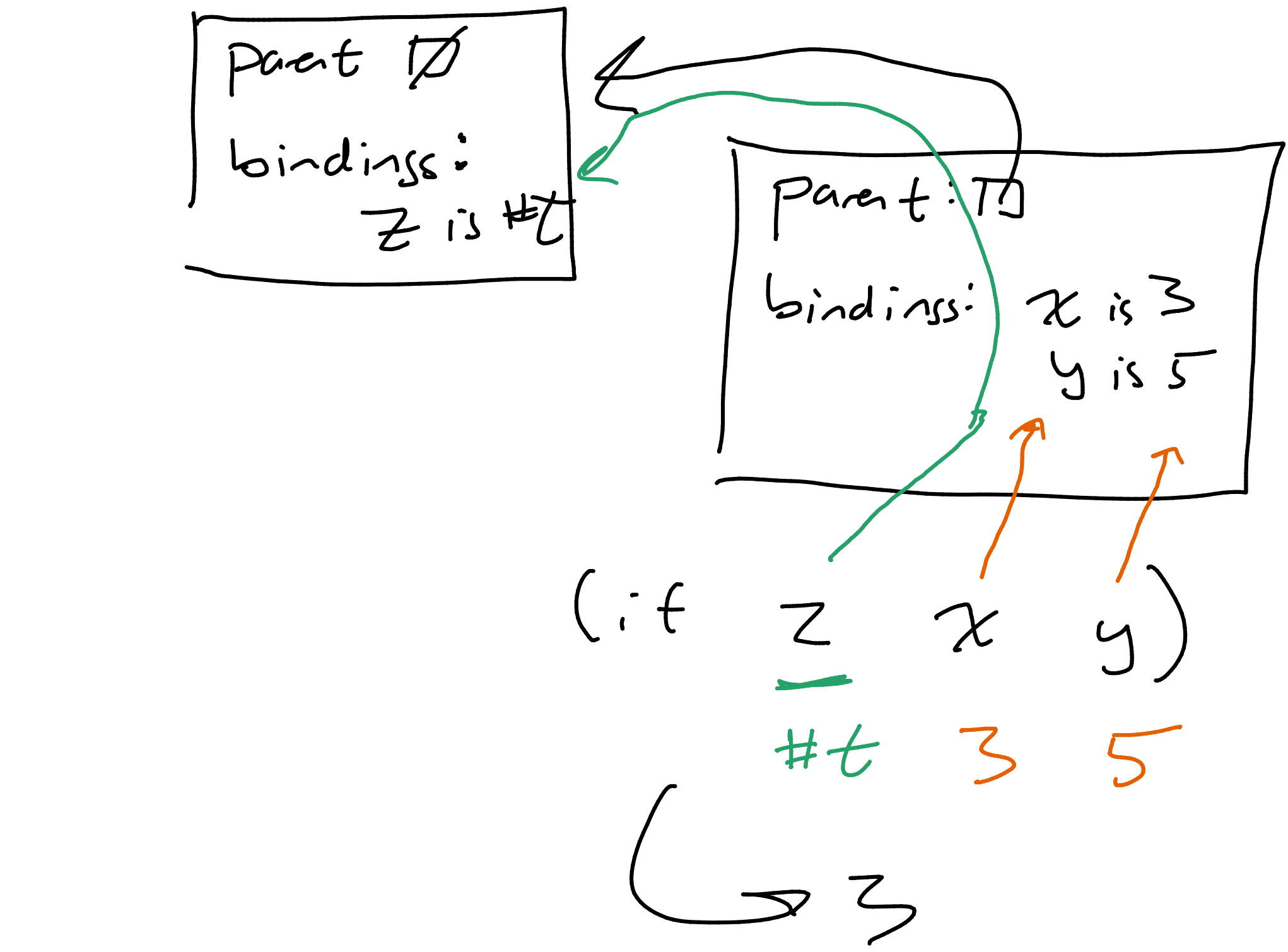
when we run let

evaluate the body of let in the context of that new frame

x → 3

} if the variable isn't there, try the parent (repeat as necessary)

```
(define z #t)
(let ((x 3) (y 5))
     (if z x y))
```



Python (scope.py)

has no way of specifying variable declactions. (no let, var, make, define)

In Python, in a function:

- a variable that is only ever read (not written) in that functions refers to a more global one if there is one
- a variable that is written to is a new local variable.