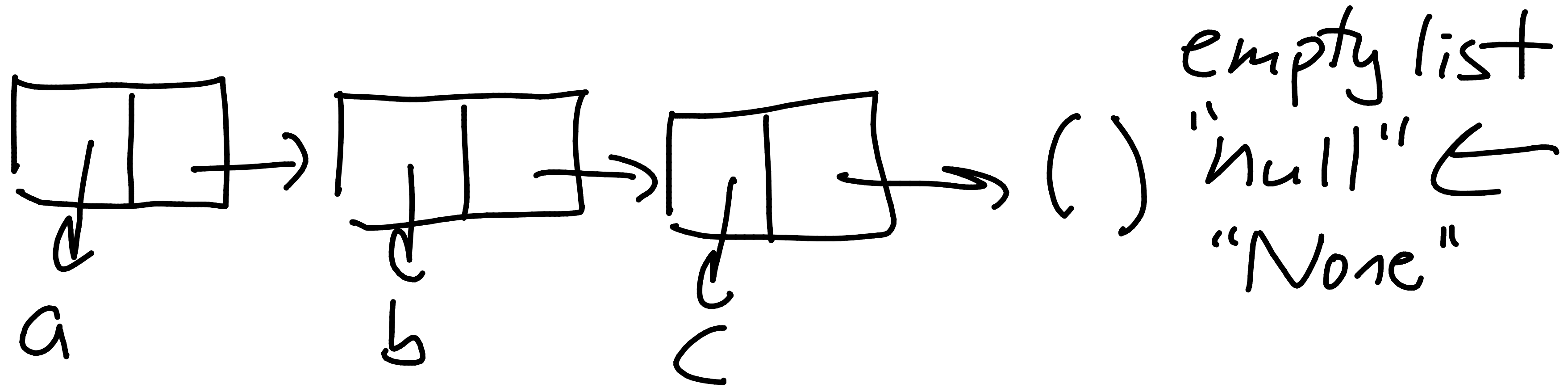
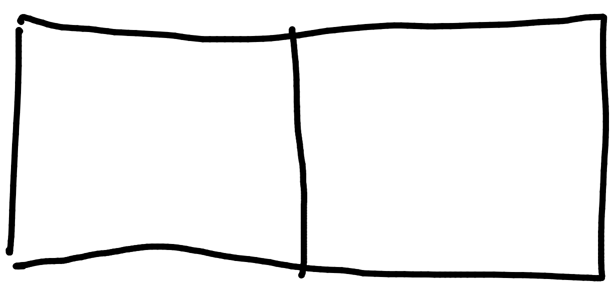
How do lists really work in Scheme?

(a b c) list

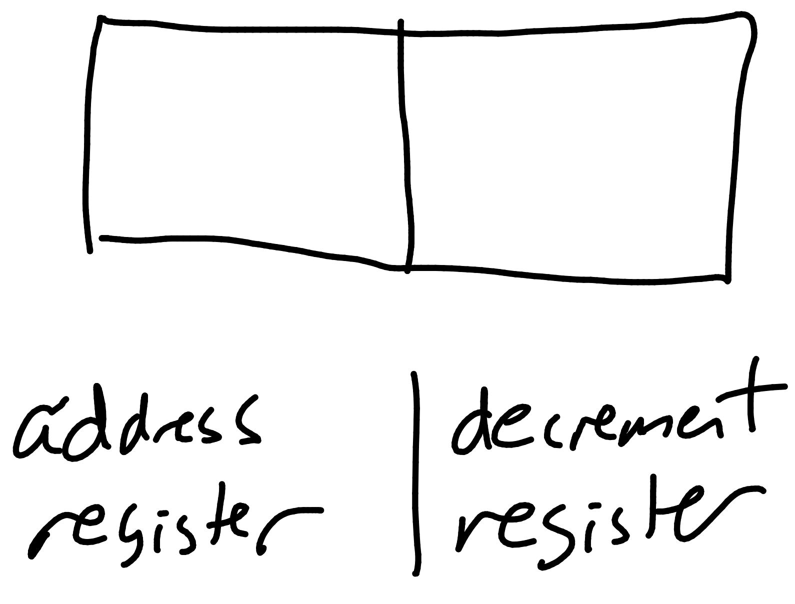
stored as a linked list



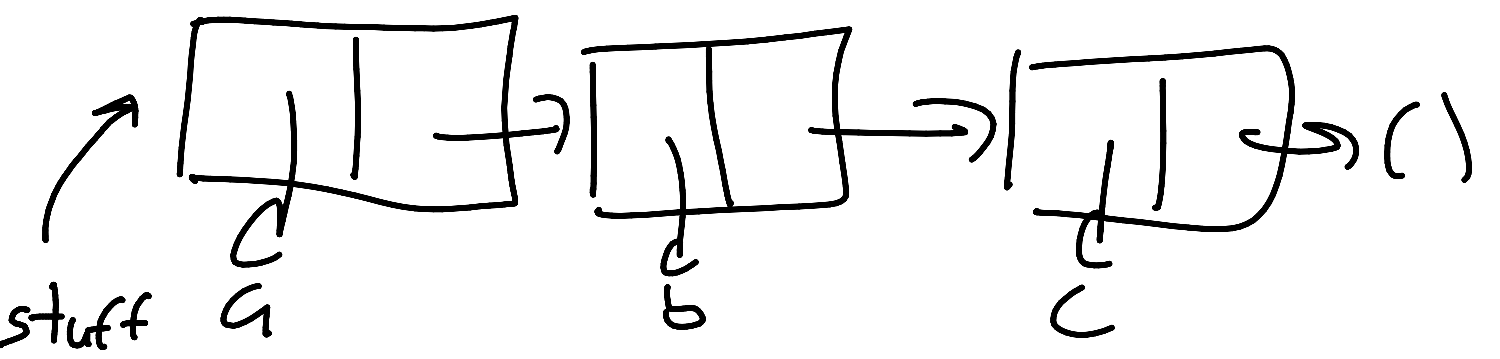


"pair"
"cons cell"
"cell"

history
1957 John McCarthy MIT
-created Lisp (effectively Scheme)



made use of registers
- address register
- decrement register



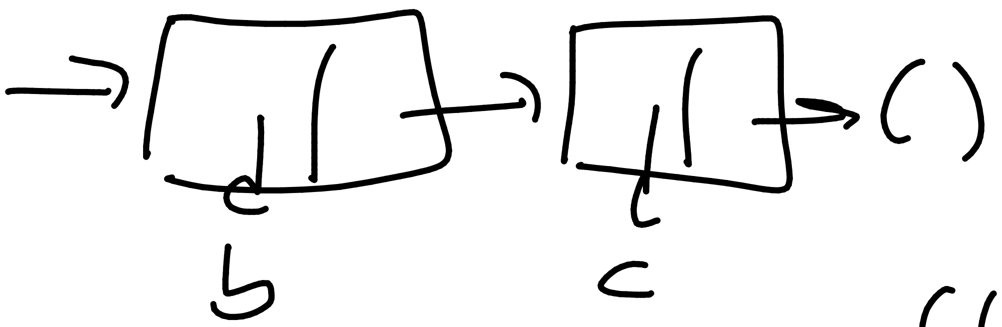
(a b c)

fist item in stuff

- "contents of address register"

(car stuff)

- rest of list



- "contents of decrement register" (b c)

(cdr stuff)

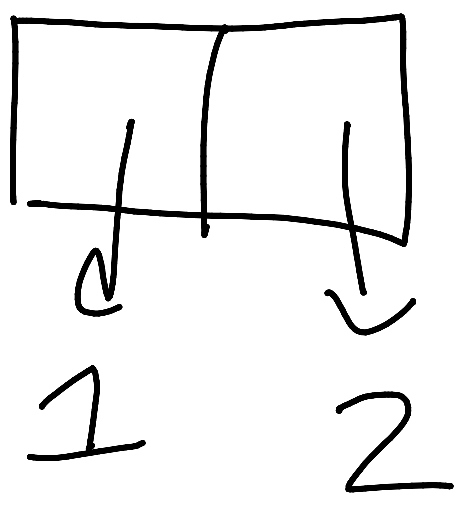
List don't have to be flat, or even properly made.

Cons - "stick onto front of a list" (naive)

(cons 1 '(2 3)) → (1 2 3)

What cons <u>really does</u> is make a new cons cell

(cons 1 2) →



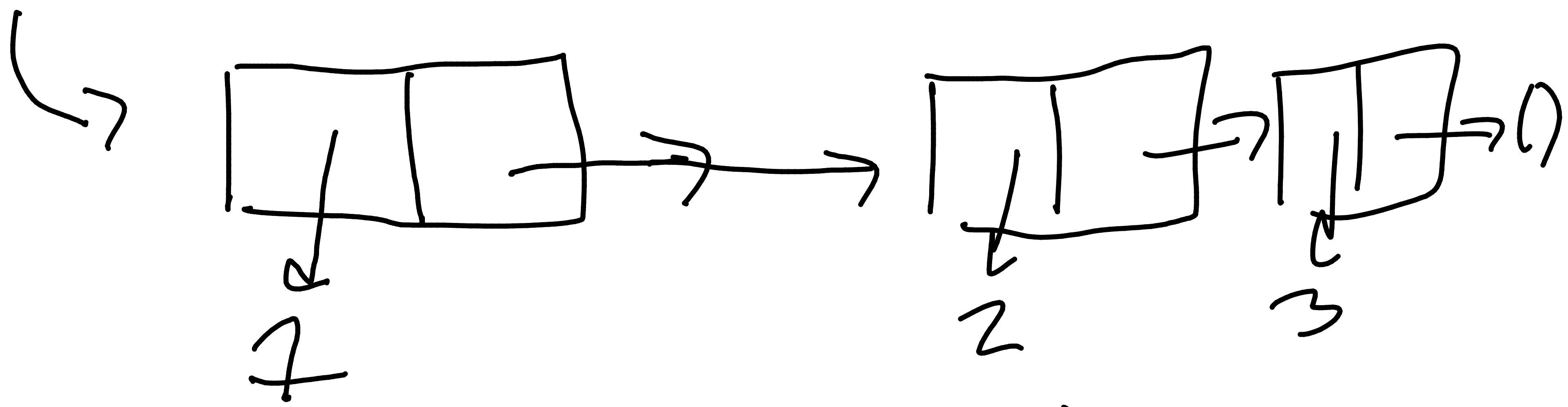
$\underset{\text{car}}{\uparrow}$ $\underset{\text{cdr}}{\uparrow}$

"imprope list"

$\underbrace{(1 \;.\; 2)}$

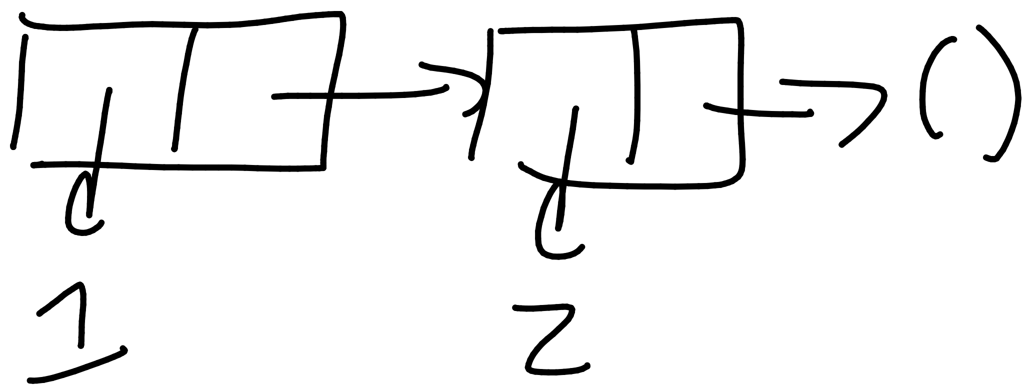
a single pair with car on left and cdr on right

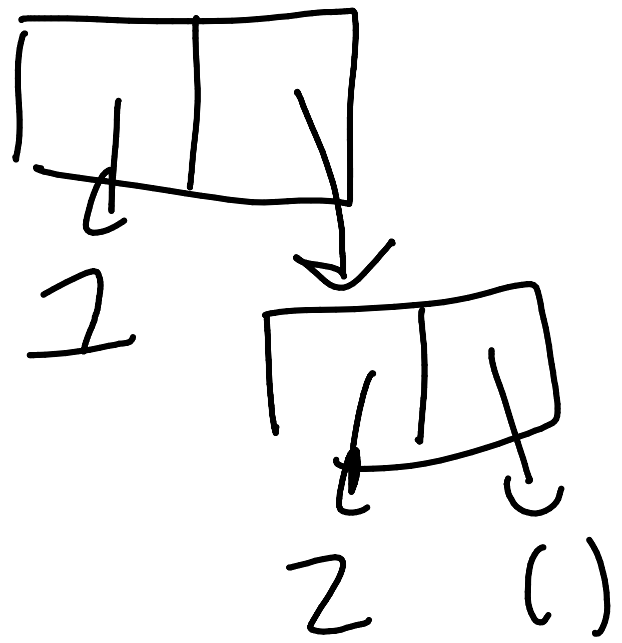
(cons 1 '(2 3))



(1 2 3)

(1 2)





(1 . (2 . ())) ↙

everything above is the same

But (1 2) is <u>not</u> (1 . 2)

(Single quote)

By default, Scheme <u>evaluates</u> everything you enter.

$(+\ 1\ 2) \xrightarrow{\text{evals}} 3$

$(\text{cons} \;\; 1 \;\; 2) \xrightarrow{\text{evals}} (1 \,.\, 2)$

$2 \xrightarrow{\text{evals}} 2$ any constat evals to itself

$(1 \;\; 2 \;\; 3) \xrightarrow{\text{evals}}$ error that 1 is not a function

'(1 2 3)

don't eval it

'(1 2 (+ 3 5))

$\Rightarrow$ (1 2 (+ 3 5))

Cons - makes a cons cell

- common applicaction - make a new list with addl item on front

not a list

$(\text{cons} \;\; \underbrace{\underline{1}}_{} \;\; \underbrace{\text{'}(2 \;\; 3)}_{}) \to (1 \;\; 2 \;\; 3)$

2 params

$$\overset{\swarrow \text{ list}}{}$$

$$\text{(append '(1 } \underbrace{\text{2) '(3}}_{\text{2 params}} \text{ 4))}$$

$\rightarrow$ (1 2 3 4)

$$\text{(list } \underbrace{\text{1 2 3 4 5}}_{\text{many params}} \text{)}$$

$\rightarrow$ (1 2 3 4 5)

(cons '(1) '(2 3))

$\hookrightarrow$ ((1) 2 3)

(list '(1) 2 '(3) '(4 5))

((1) 2 (3) (4 5))

Creating functions

In Scheme, there is a built-in function that creates functions

→ <u>lambda</u>

1930s Alonzo Church designed a prog lang based on nothing but functions

```
(lambda (arguments)
   body)
```

Create a function to add 2 numbers

```
(lambda (x y)
   (+ x y))
```

```
(define my-add
   (lambda (x y)
      (+ x y)))
```

```
(my-add 3 5)
```