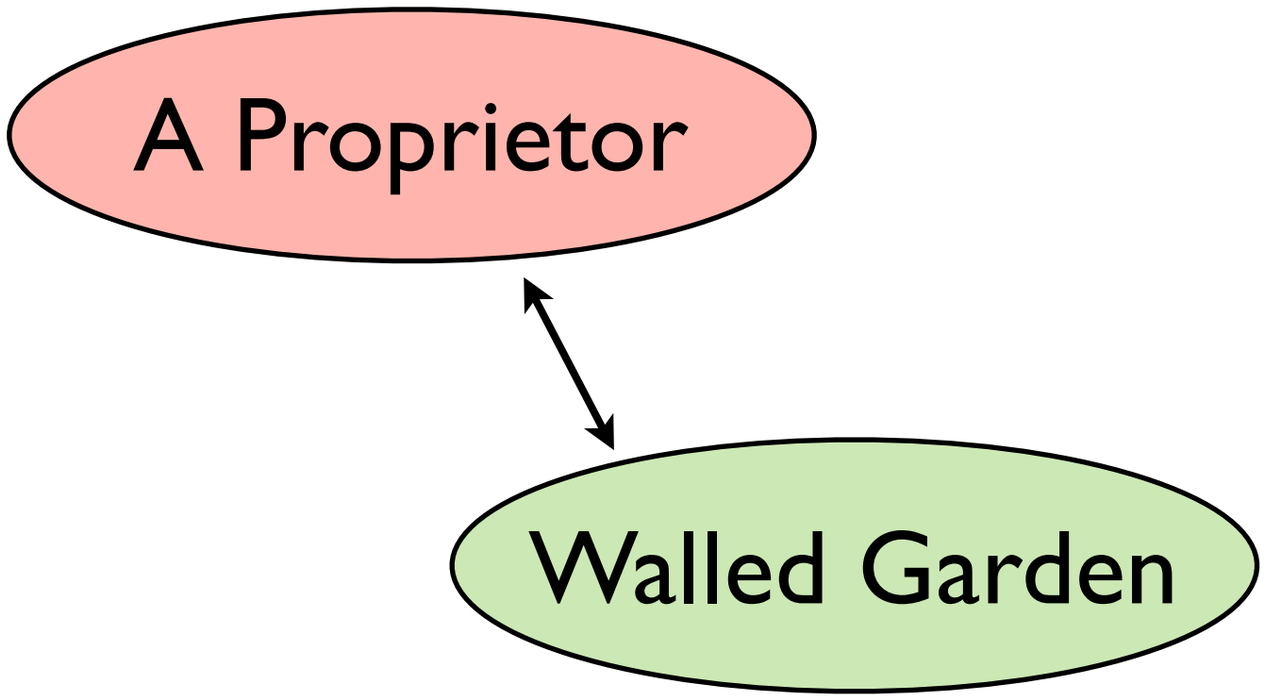
The Case for (Scheme)

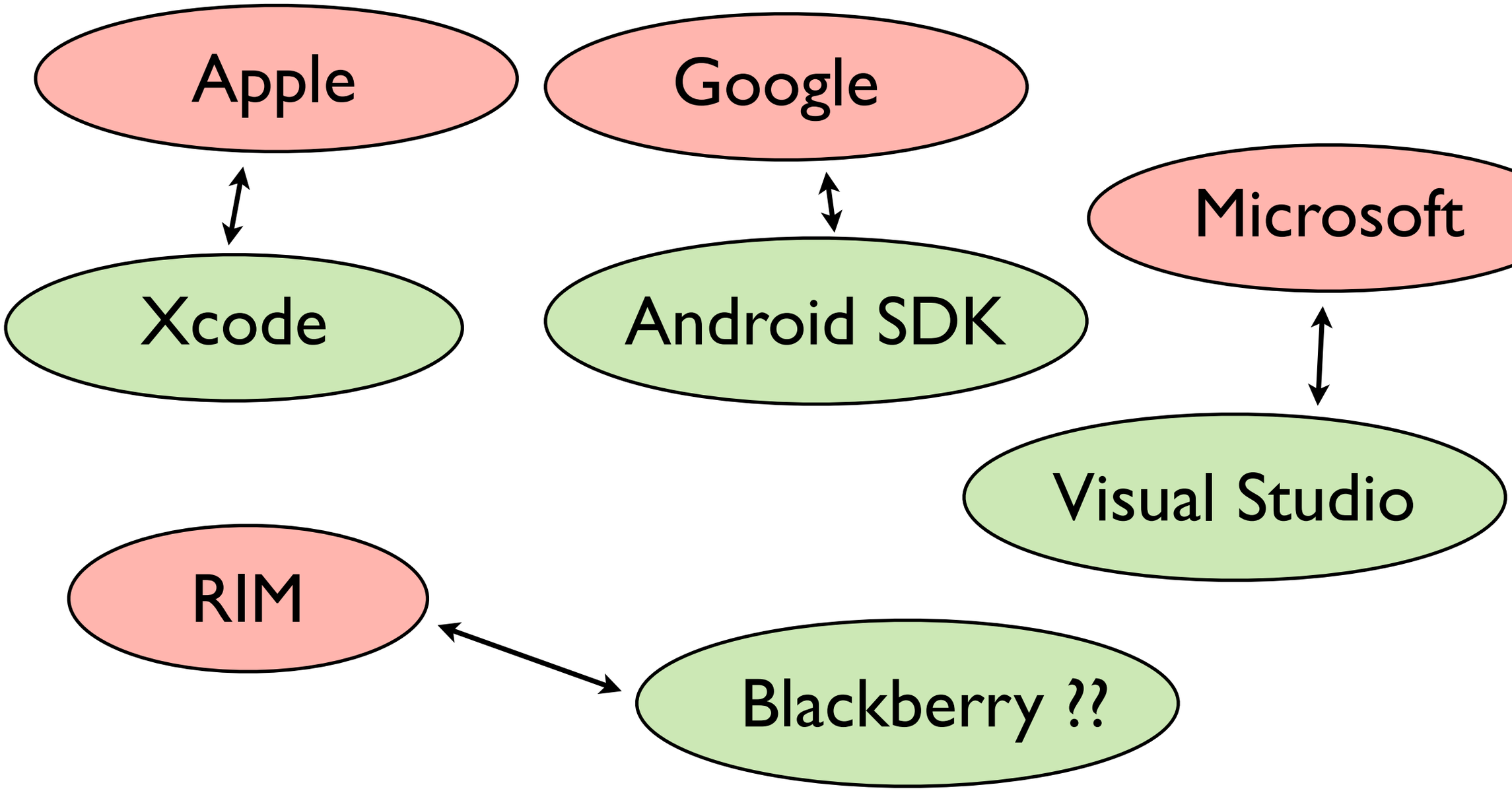
Jason Felice

@eraserhd
http://maitria.com/

A Mobile Platform



A Mobile Platform



Teams

Web

iOS

Android

Back-end

Windows

Platform Teams

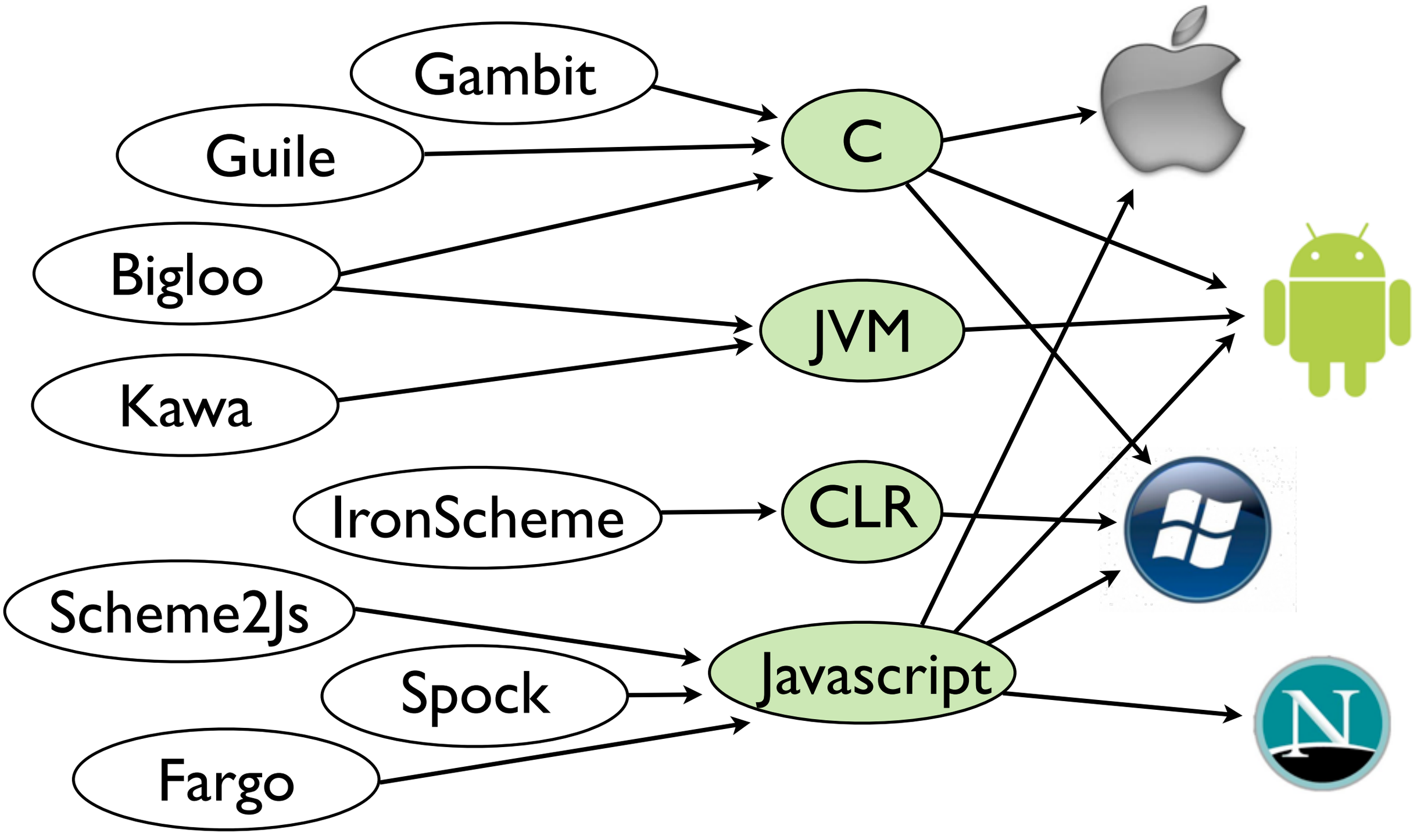
Steering

Hrmm...

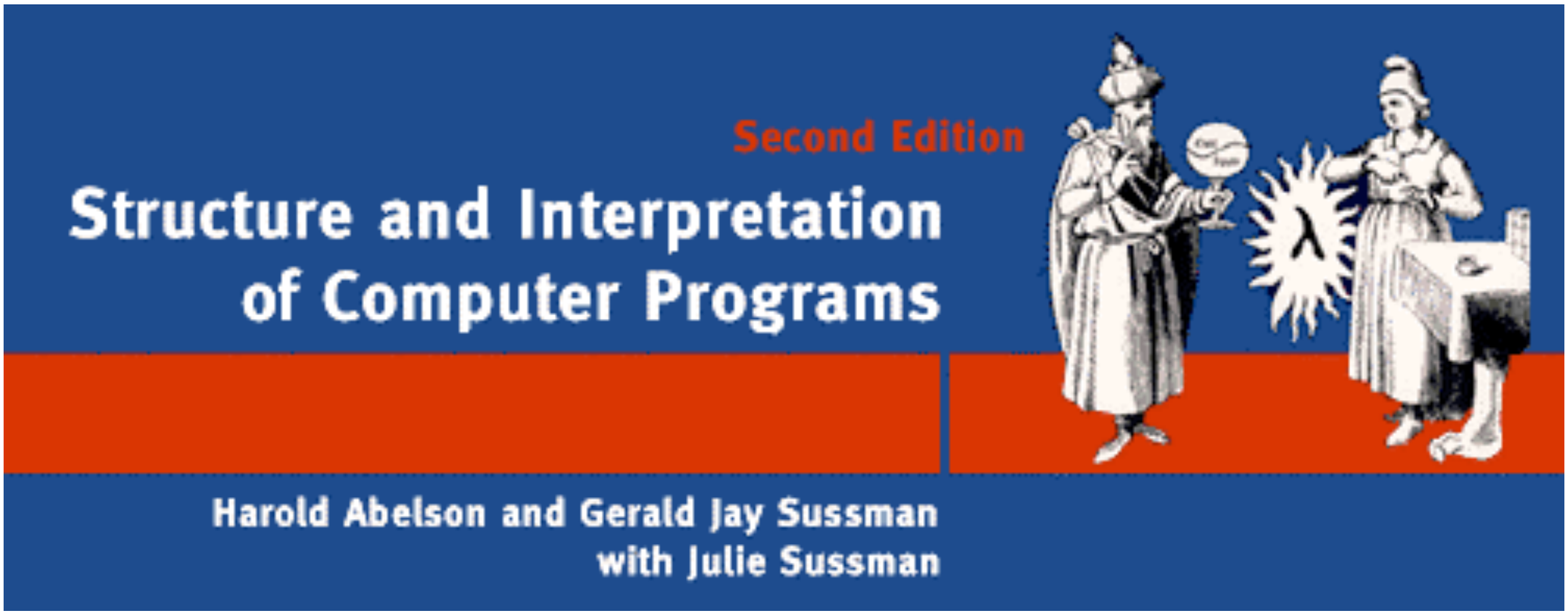
- One feature at a time.
- Implement on every platform.
- Simultaneously release them all.

Work on ALL the Apps

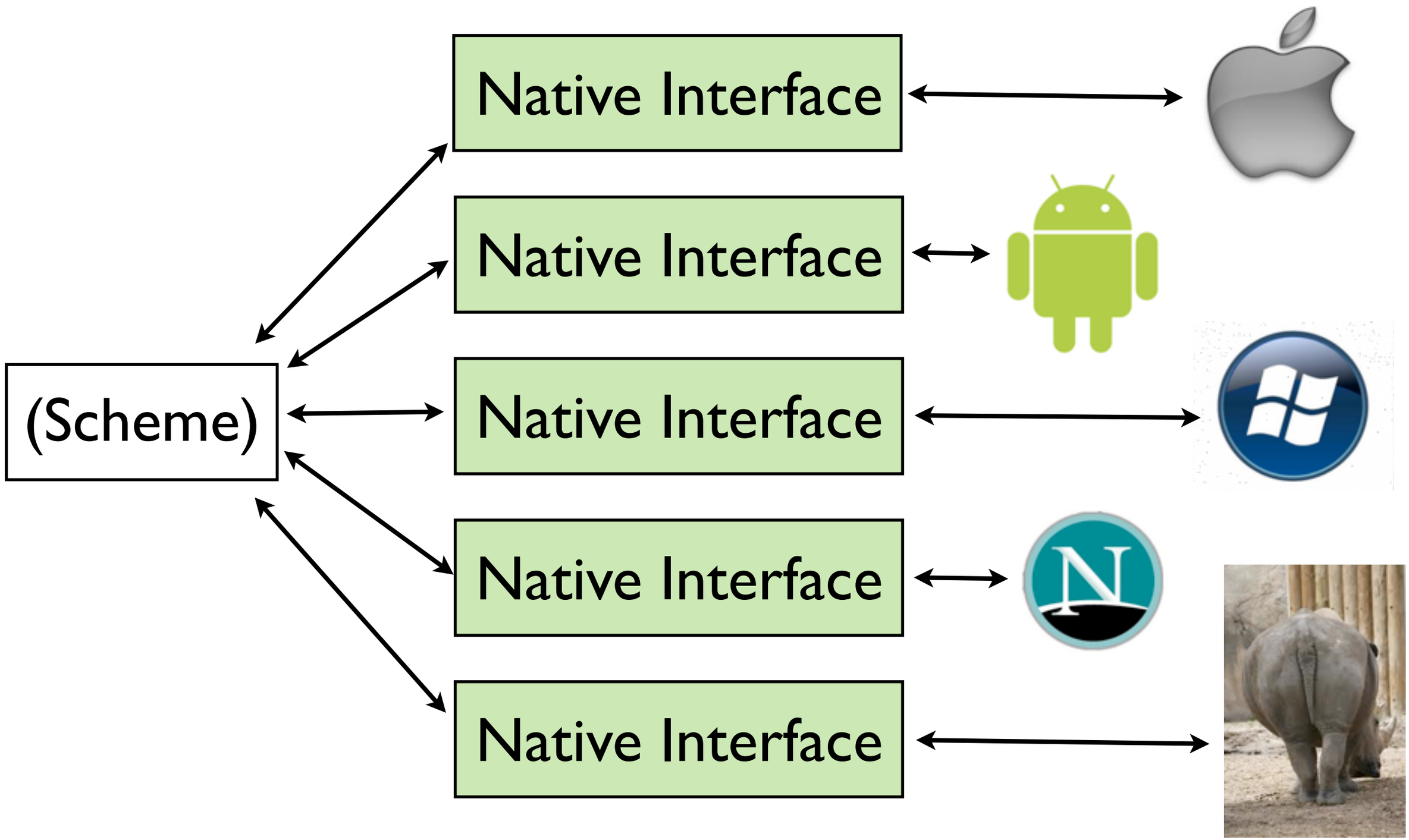
(Scheme)


Gambit
Guile
Bigloo
Kawa
IronScheme
Scheme2Js
Spock
Fargo
C
JVM
CLR
Javascript

SICP


Second Edition
Structure and Interpretation
of Computer Programs
Harold Abelson and Gerald Jay Sussman
with Julie Sussman

Structure


Native Interface
Native Interface
(Scheme)
Native Interface
Native Interface
Native Interface

Because we care about beautiful code

we hack with intention

Because we care about geek joy

we encourage geeks to code from the heart

Because we care about what's alive in the people we touch

we speak and listen with courageous curiosity

Because we care about what emerges when we collaborate

we show up with confident humility

maitria.com

Damn Parentheses!

```
(let loop ((i 0))
  (cond
    ((= i 10)
     #f)
    (else
     (display i)
     (loop (+ i 1)))))
```

Damn Parentheses!

```
if ((self = [super
     initWithNibName:[[self class]

                                 description]

       bundle:[NSBundle

                    bundleForClass:[self class]]

   ])) {

  ...

}
```

Hello, World!

```
(display "Hello, World!")
(newline)
```

Hello, World!

```
(define (say-hello)
  (display "Hello, World!")
  (newline))

(say-hello)
```

Hello, World!

```
(define (say-hello)
  (display "Hello, World!")
  (newline))

(say-hello)
```

Hello, World!

```
(define (say-hello)
  (display "Hello, World!")
  (newline))

(say-hello)
```

factorial

5! =

5 * 4 * 3 * 2 * 1

n! =

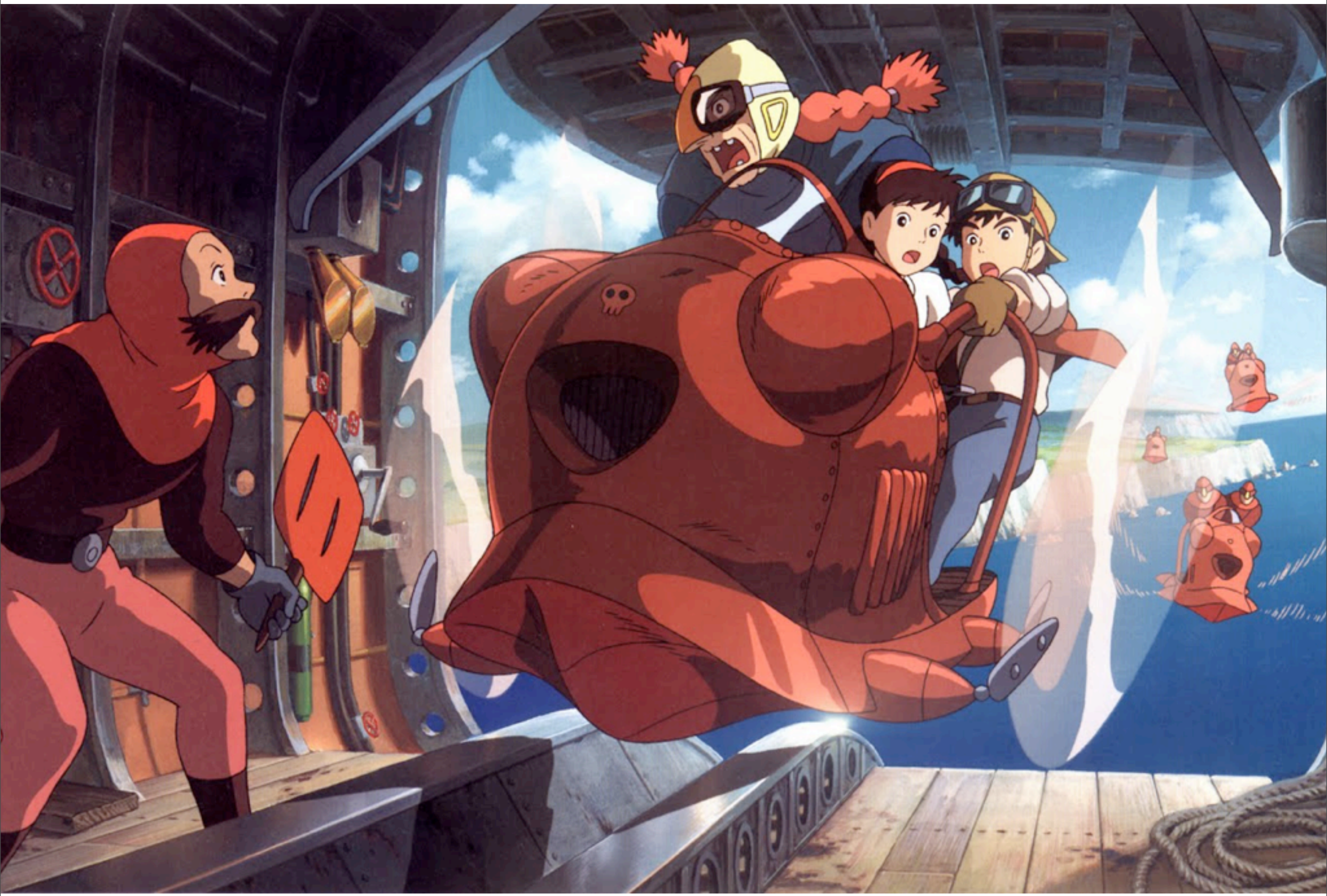
n * (n - 1) * (n - 2) * ... * 1

factorial

$$n! = \begin{cases} 1, & n=1 \\ n(n-1)!, & n>1 \end{cases}$$

$$1! = 1$$

$$5! = 5 \cdot 4!$$



factorial

```
(define (factorial n)
  (if (= n 1)
      1
      (* n (factorial (- n 1)))))
```

factorial

```
(define (factorial n)
  (if (= n 1)
      1
      (* n (factorial (- n 1)))))
```

factorial

```
(define (factorial n)
  (if (= n 1)
      1
      (* n (factorial (- n 1)))))
```

factorial

```
(define (factorial n)
  (if (= n 1)
      1
      (* n (factorial (- n 1)))))
```

factorial

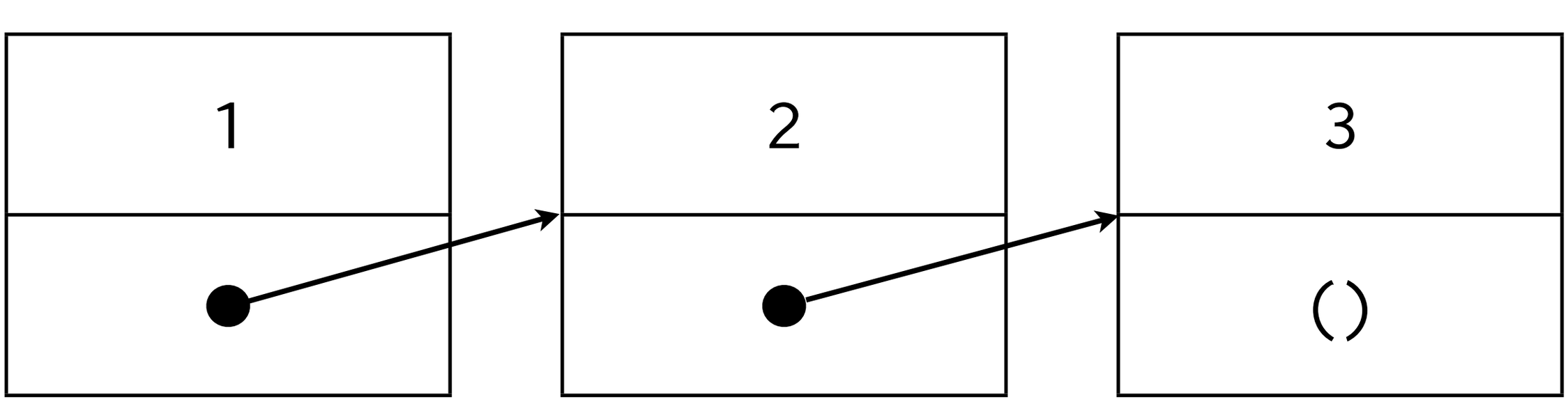
```
(define (factorial n)
  (if (= n 1)
      1
      (* n (factorial (- n 1)))))
```

Lists

(1 2 3)

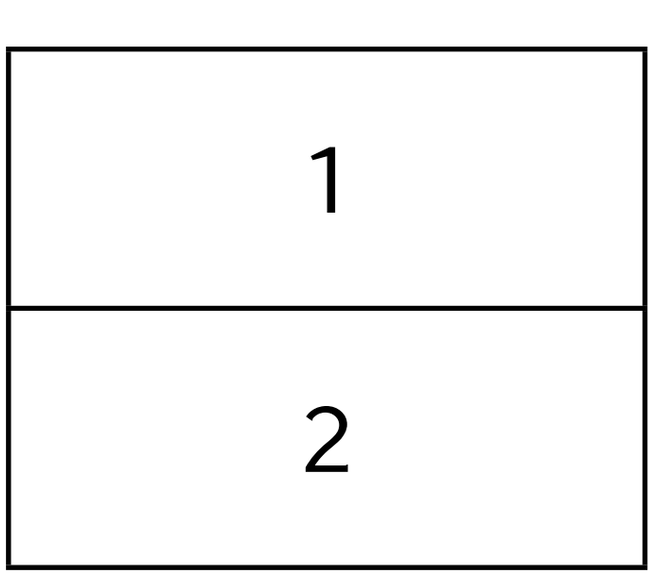
Lists

(1 2 3)



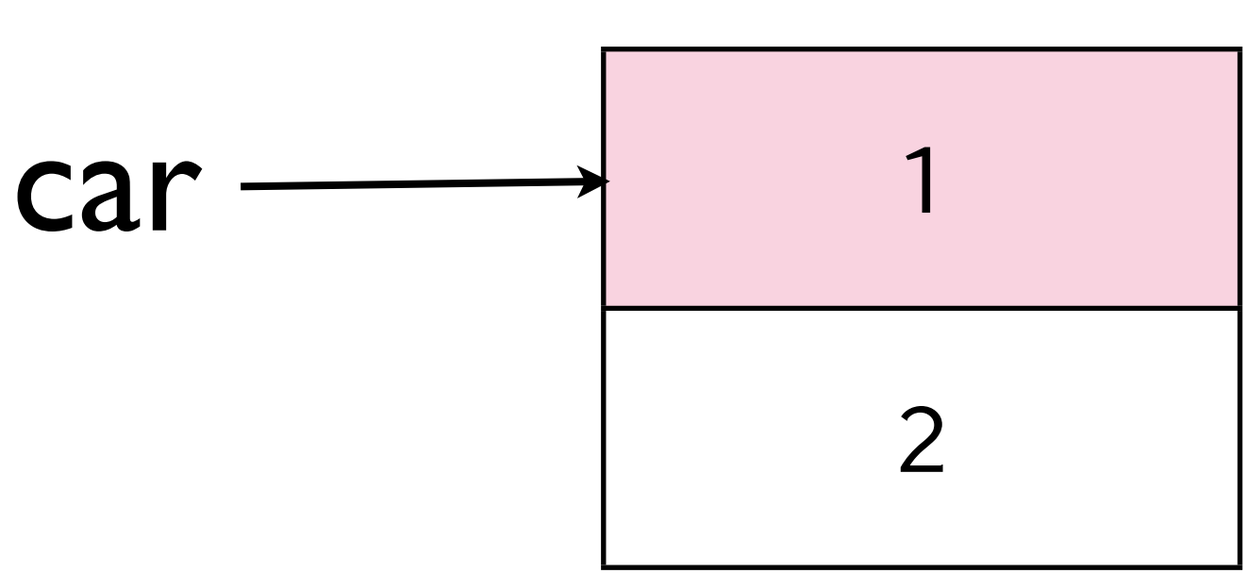
Pairs

(1 . 2)



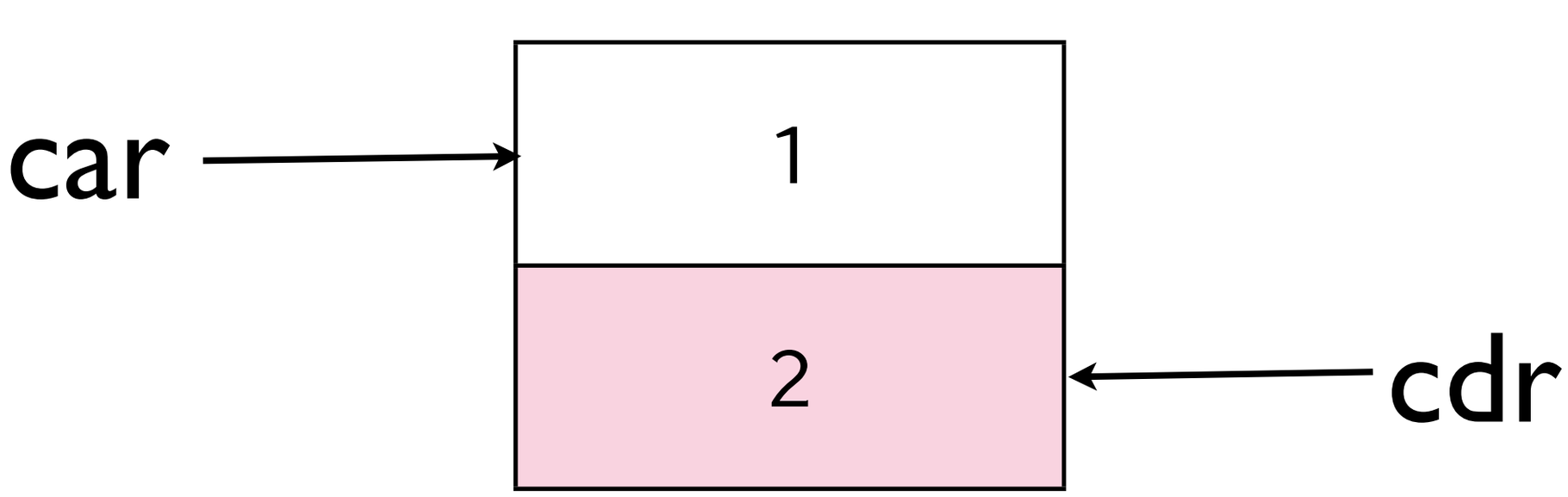
Pairs

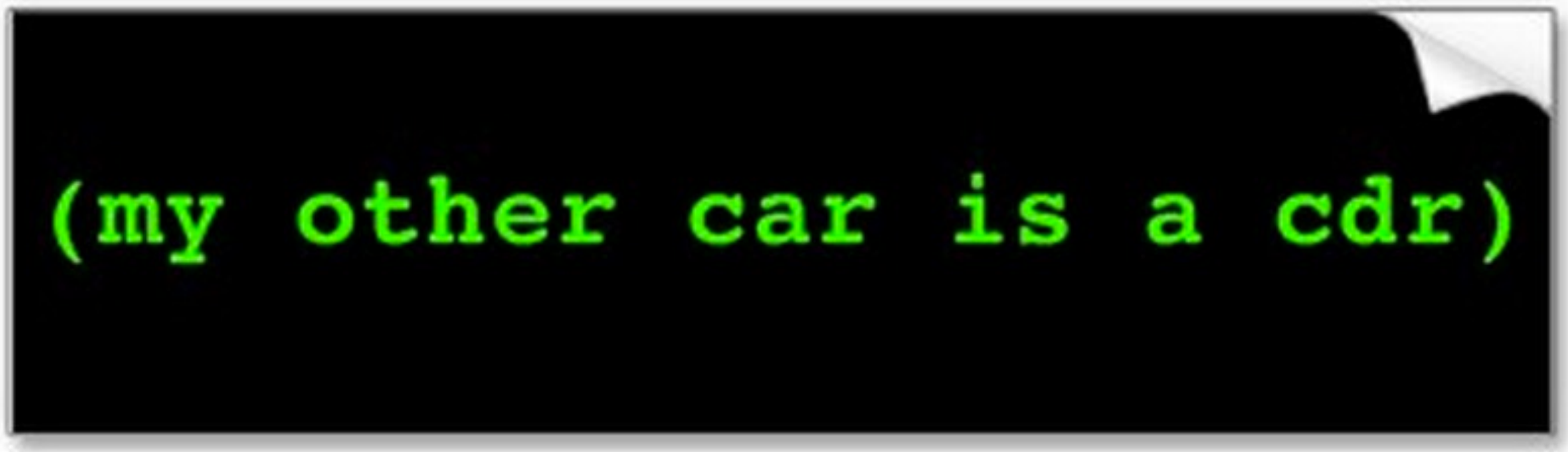
(1 . 2)



Pairs

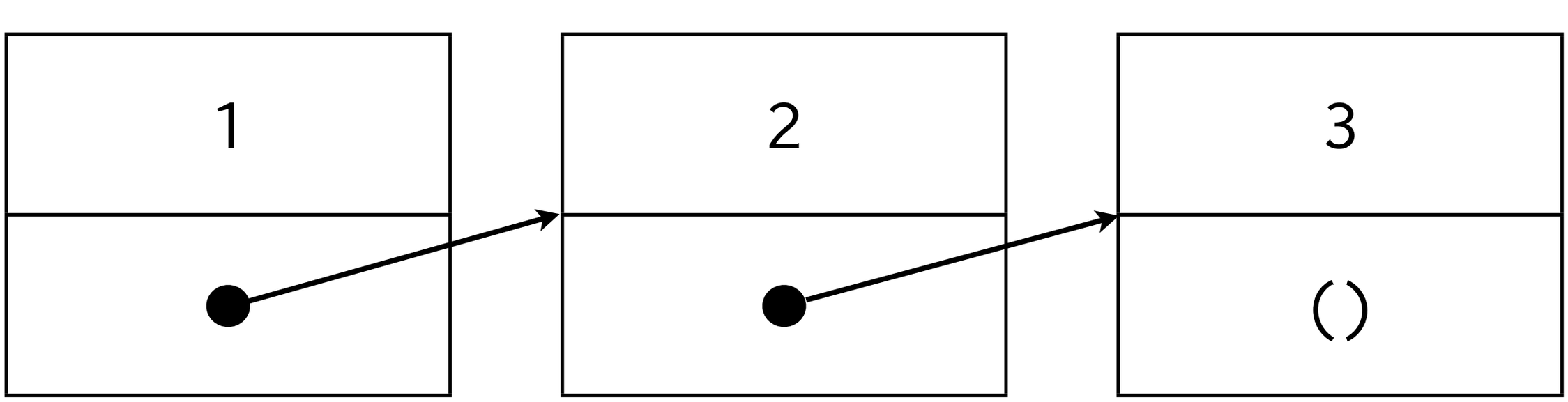
(1 . 2)




(my other car is a cdr)

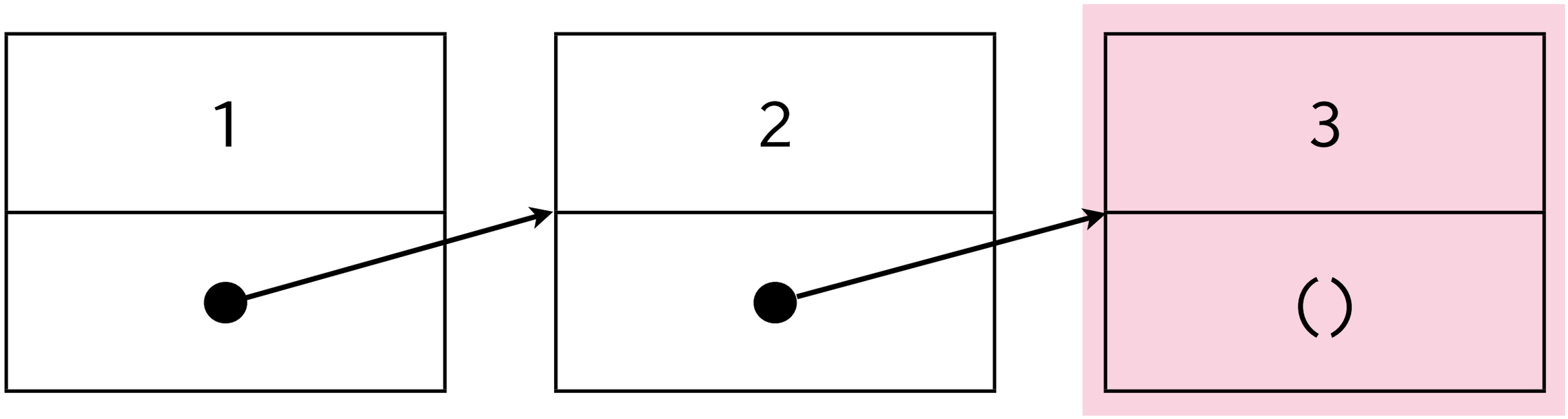
Lists

(1 . (2 . (3 . ())))



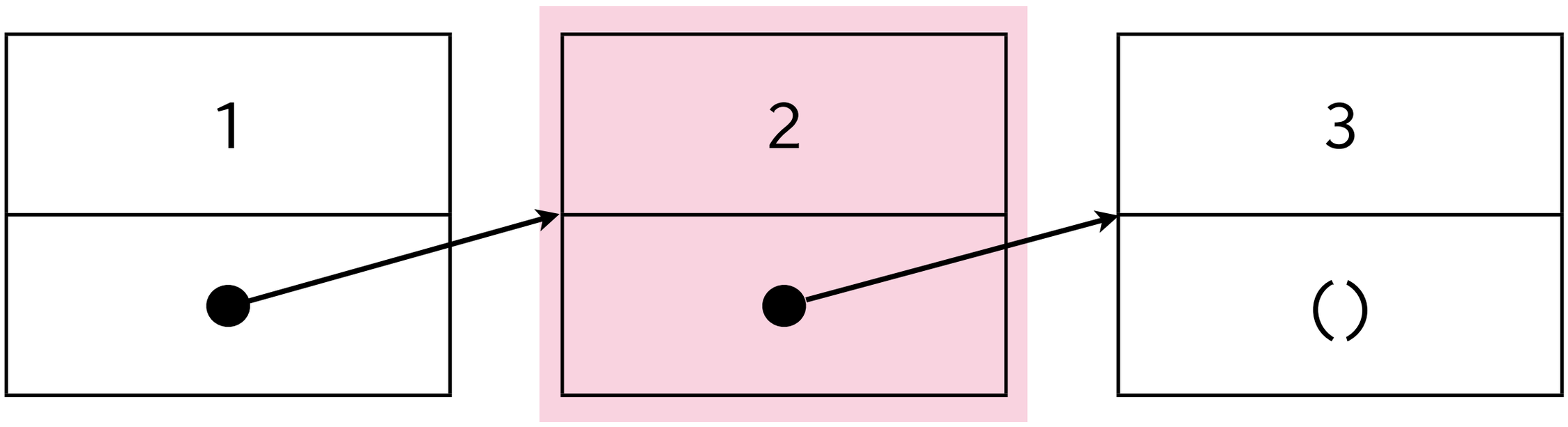
Lists

(1 . (2 . (3 . ())))



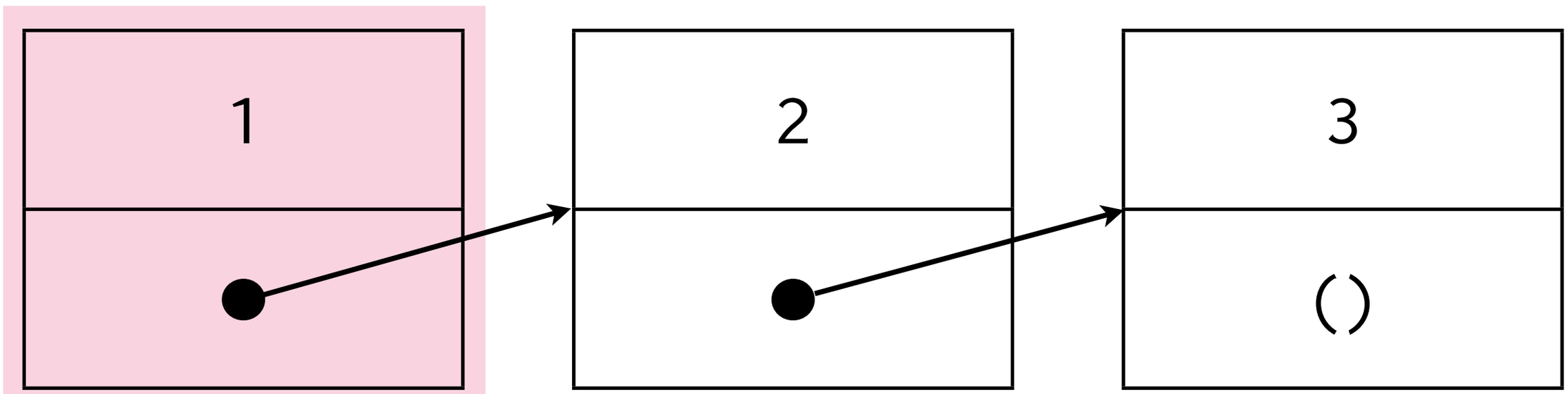
Lists

(1 . (2 . (3 . ())))



Lists

(1 . (2 . (3 . ())))

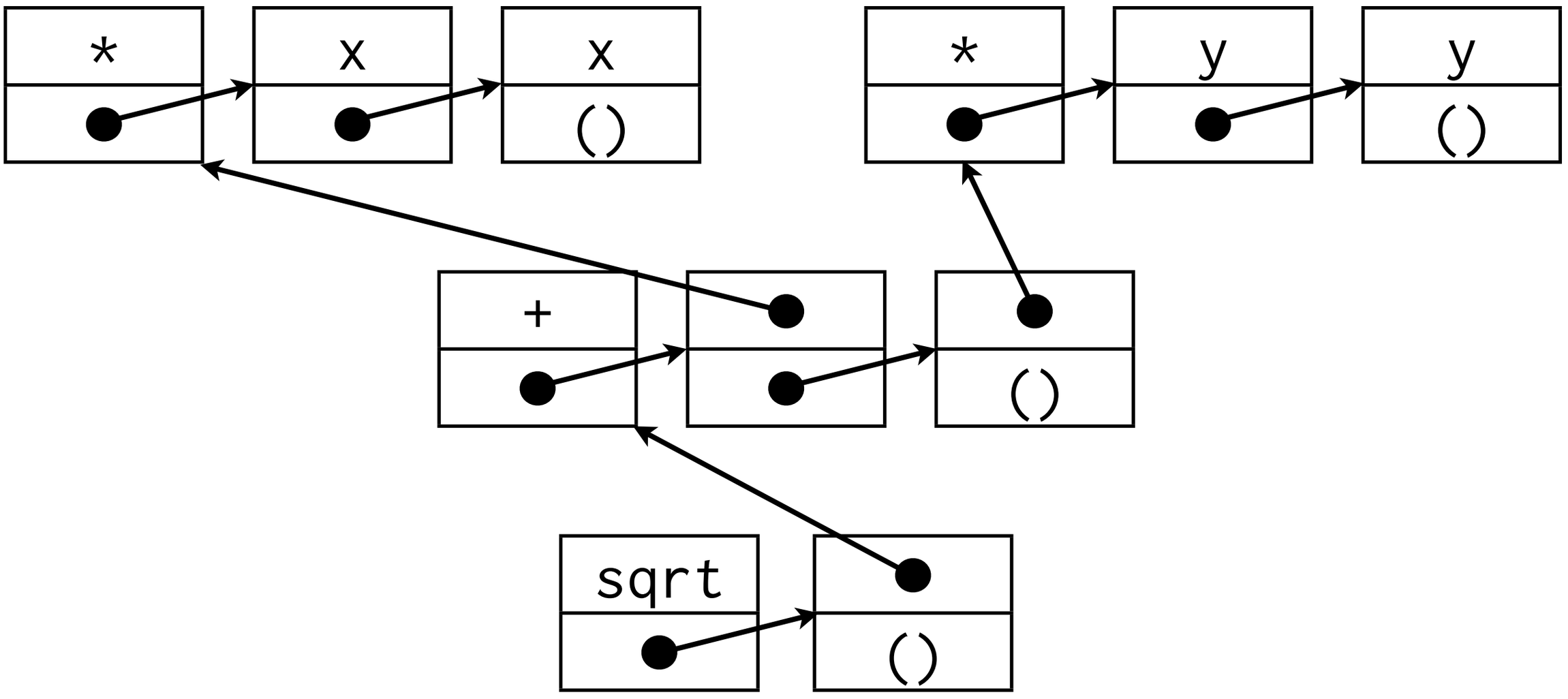


Code is Data

```
(sqrt (+ (* x x) (* y y)))
```

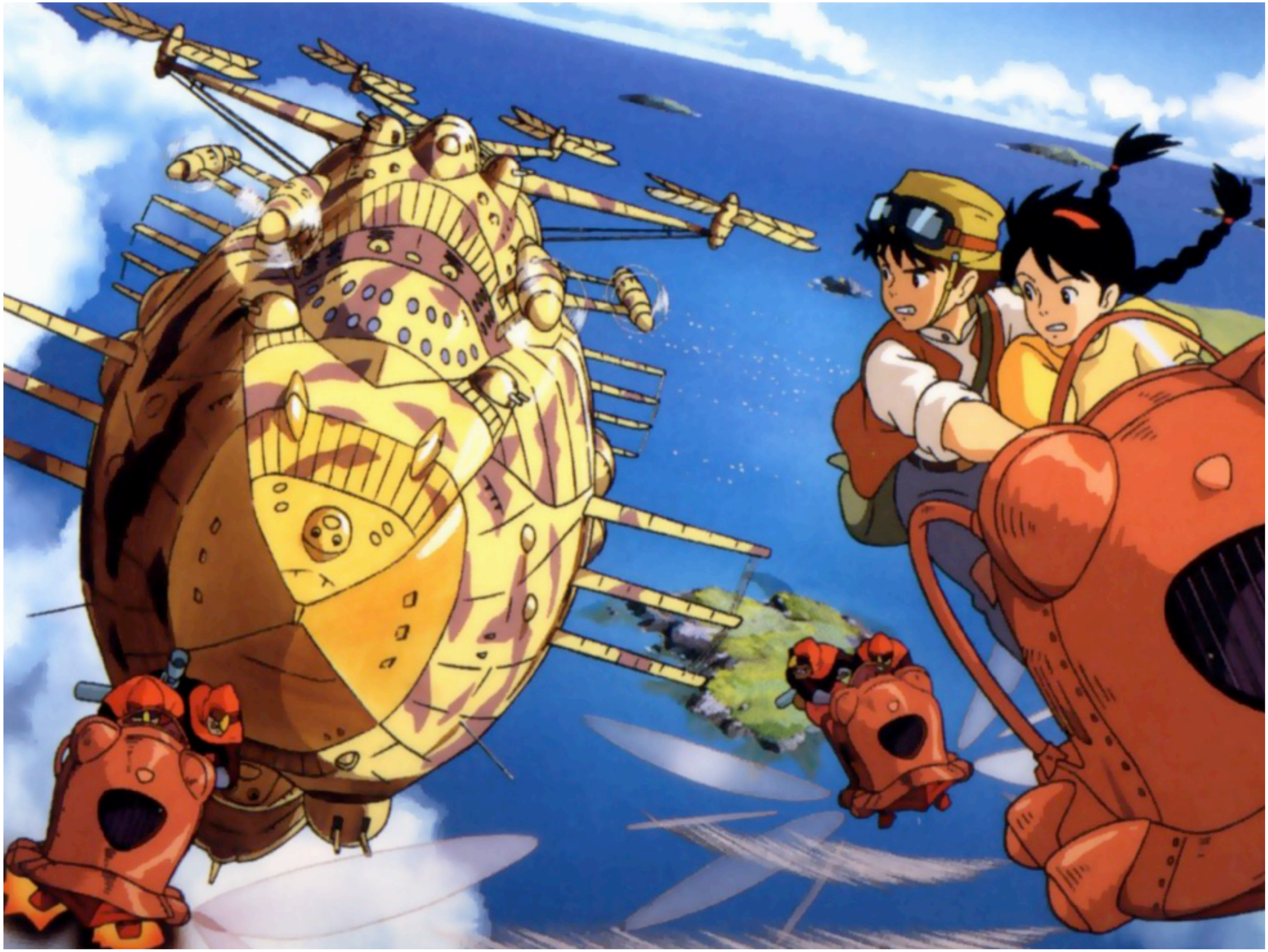
Code is Data

```
(sqrt (+ (* x x) (* y y)))
```



Data is Code

```
(define x 7)
(define y 4)
(define x2 (list '* 'x 'x))
(define y2 (list '* 'y 'y))
(define x2+y2 (list '+ x2 y2))
(define hypot (list 'sqrt x2+y2))
(eval hypot) => 8.062257748298550
```




```
(on (collision (ice-block? b) (ice-hole? h))
    (in-succession
      (remove-object b)
      (simultaneously
        (play-sound "splunk")
        (animate h "block-melts-into-ice-hole"))
      (remove-object h)))
```

```
(animation “helper-slides-south”
  (play-frames (1 .. 5) for: 0.3)
  (repeat
    (play-frames (6 7) for: 0.2)
  (play-frames (8 .. 10) for: 0.2))
```