Cons Diagrams and Boxes

CIS352 — Spring 2021

Kris Micinski

Derived Types

- **S-expressions** (**symbolic** expression)
  - Untyped lists that generalize neatly to trees:

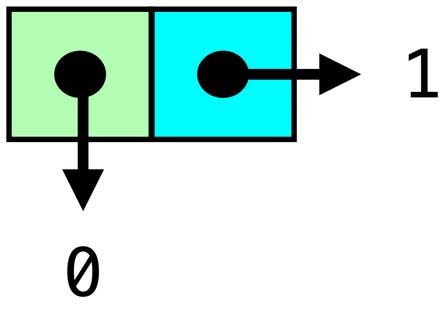
```
(this (is an) s expression)
```

- Computer represents these as **linked** structures
  - Cons cells of head & tail `(cons 1 2)`

Derived Types

- Racket also has **structural** types
  - Defined via **struct**; aids robustness
  - We will usually prefer agility of “tagged” S-expressions
- Also an elaborate object-orientation system (we won’t cover)

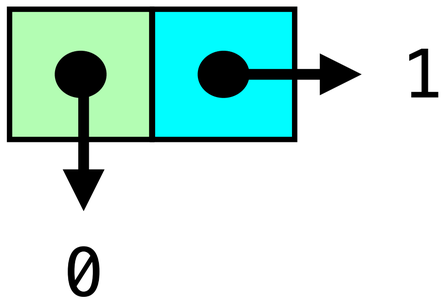
```
(cons 0 1)
```



The function **cons** builds a cons cell

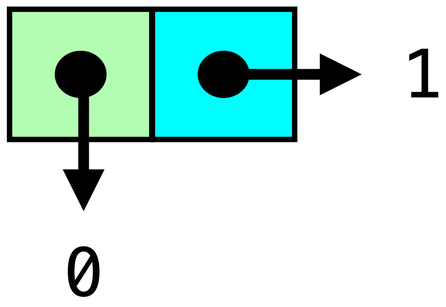
The function **car** gets the left element

`(car (cons 0 1))` is `0`



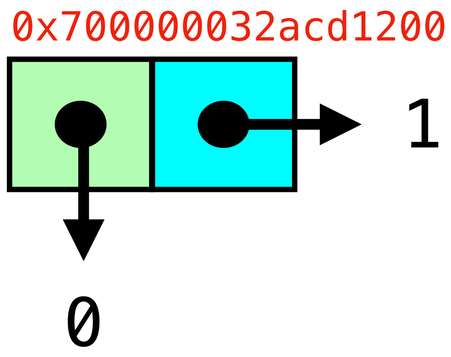
The function **cdr** gets the left element

`(cdr(cons 0 1))` is `1`



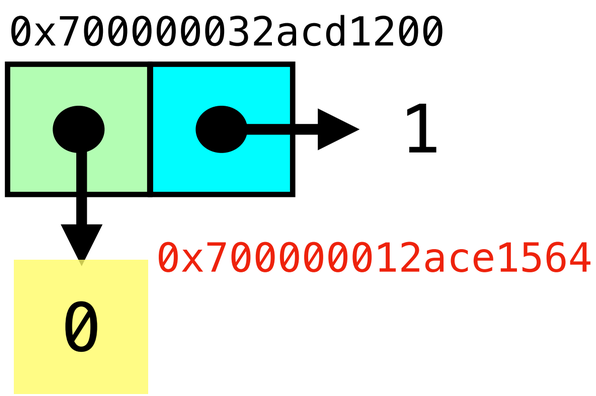
At runtime, each cons cell sits at an **address** in memory

`(cdr(cons 0 1))` is `1`



In fact, numbers are **also** stored in memory locations.

They are thus said to be a “boxed” type



Actually, every Racket variable stores a value in some “box” (i.e., memory location)

```
(define x 23)
(displayln x)
(set! x 24)
(displayln x)
```

0x700000033dea2280

x **23**

Actually, every Racket variable stores a value in some “box” (i.e., memory location)

```
(define x 23)
(displayln x)
(set! x 24)
(displayln x)
```

0x700000033dea2280

x **23**

Console output…
> 23

Actually, every Racket variable stores a value in some "box" (i.e., memory location)

```
(define x 23)
(displayln x)
(set! x 24)
(displayln x)
```

0x700000033dea2280

x **24**

x's value **changes** to 24

```
(define x (vector 1 2 3))
(vector-set! x 1 0)
x
;; '#(1 0 3)
```

Vectors (similar to arrays) are mutable, and give O(1) indexing and updating

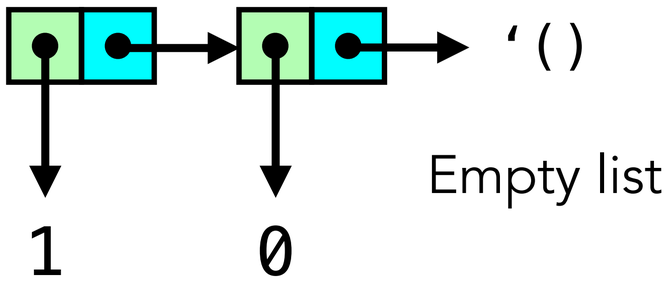
Unless we say otherwise, you should avoid using set!, any use will be at your own risk

Similarly, avoid vector-set!, hash-set!, …

Using `set!` will, in CIS352, lead to hard-to-debug code that will make it much harder for instructors to understand your code

Pairs enable us to build **linked lists** of data

```
(cons 1 (cons 0 '()))
```



This is how Racket represents lists in memory

Note that in Racket, the following are equivalent

```
(cons 2 (cons 1 (cons 0 '())))
'(2 1 0)
```

But the following is called an **improper list**

```
(cons 2 (cons 1 0))
'(2 1 . 0)
```

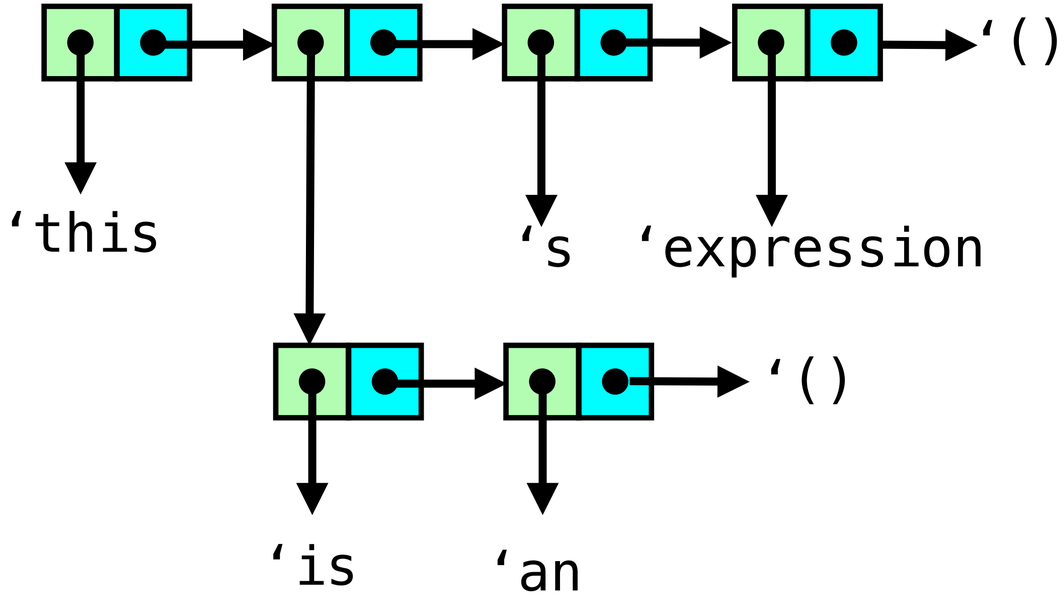
Dot indicates a cons cell of a left and right element

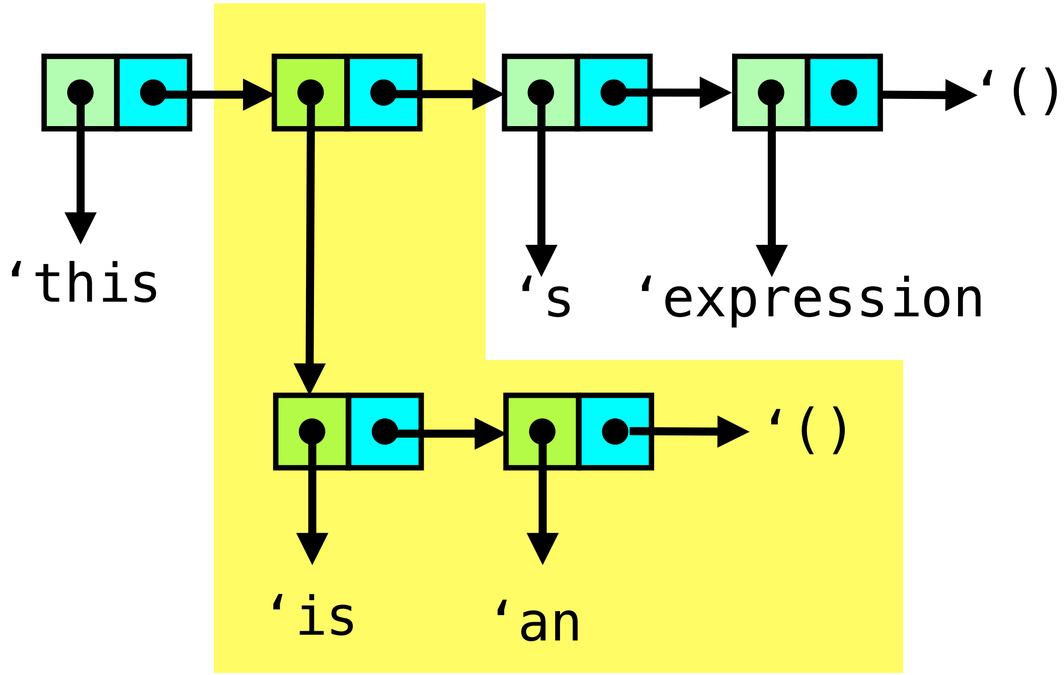
Also can build **compound** expressions

```
'(this (is an) s expression)
```

Also can build **compound** expressions

`'(this (is an) s expression)`

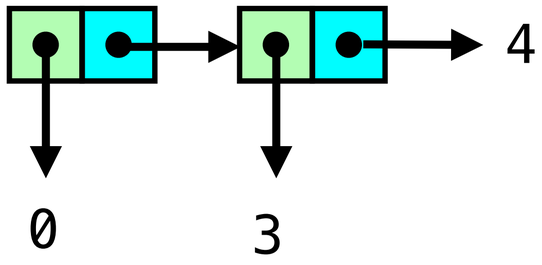



'()
'this
's
'expression
'()
'is
'an

Draw the cons diagram for…

- (cons 0 (cons 3 4))
- Is this a list? If not, what is it?
- (cons 0 (cons 3 (cons 4 ‘()))))
- Is this a list? If not, what is it?

(cons 0 (cons 3 4))



This is *not* a list (an improper list)

(cons 0 (cons 3 (cons 4 ‘())))