

# Modeling with Mutable Data

- Mutable list structure
- Queues
- Tables

# Mutable List Structure

- We introduced the notion of assignment with `set` !
- Assignment greatly increased the expressive power of programming
- What about compound objects?

# Mutable List Structure

- We often need to modify compound objects
- We defined compound data in terms of constructors and selectors
- We need one more type of operation for modifying compound data: *mutators*
- For instance, a data structure for bank accounts might have an operation

```
(set-balance! < account > < new-value >)
```

# Mutable List Structure

- Fundamentally, what do we need to think about in order to make mutators in Scheme?

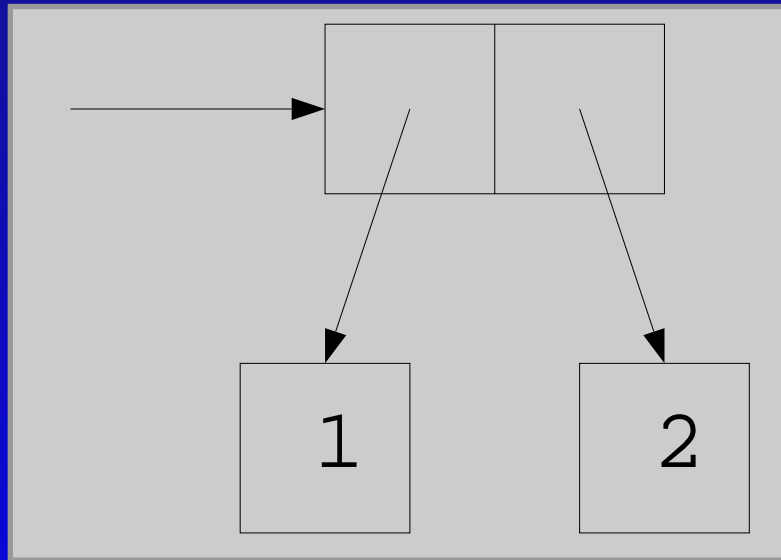
# Mutable List Structure

- Fundamentally, what do we need to think about in order to make mutators in Scheme?
- We need to be able to modify pairs
- In particular, we need to be able to modify the car and cdr of a pair

# Mutable List Structure

- The basic pair operations — `cons`, `car`, `cdr` — can be used to build constructors and selectors for compound data
- They are incapable of modifying an existing list structure

# Mutable List Structure



# Mutable List Structure

- What will happen if we evaluate

```
1] => (define x (cons 1 2))
```

```
1] => (set! (car x) 3)
```

# Mutable List Structure

- Scheme provides two mutators for pairs

`set-car!`

`set-cdr!`

# Mutable List Structure

- Special forms or procedures?

`set-car!`

`set-cdr!`

# Mutable List Structure

- What will happen if we evaluate

```
1] => (define x (cons 1 2))
```

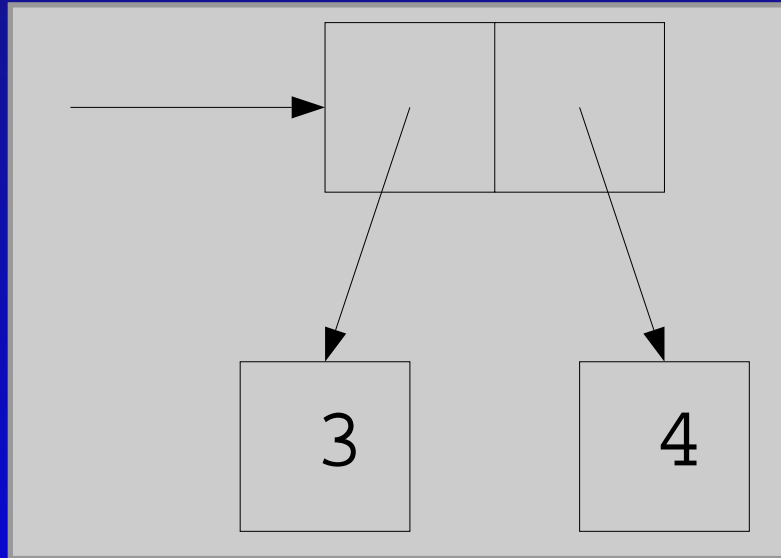
```
1] => (set-car! x 3)
```

```
1] => x
```

```
1] => (set-cdr! x 4)
```

```
1] => x
```

# Mutable List Structure



## Example

- The following procedure returns a copy of a given list:

```
(define (copy-list x)
  (if (null? x)
      '()
      (cons (car x)
            (copy-list (cdr x)))))
```

## Example

- Consider the evaluation of the following expressions:

```
(define x '((a b) c))
```

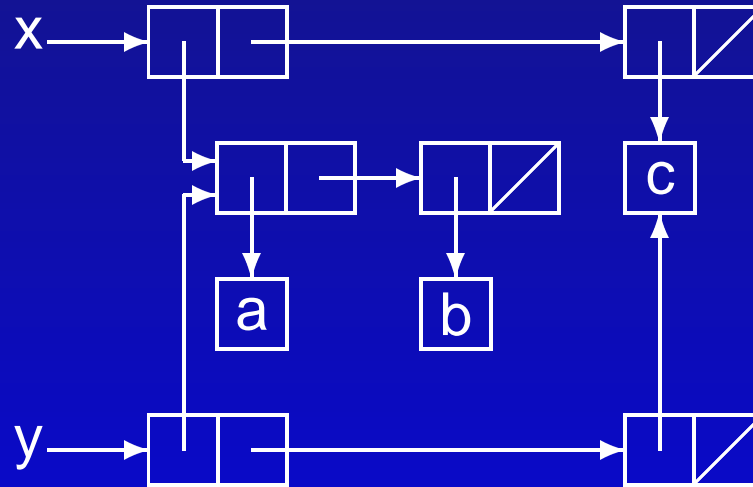
```
(define y (copy-list x))
```

- What will the interpreter print as the values of `x` and `y`?
- Draw a box and pointer diagram to explain your answer.

## Example

```
(define x '((a b) c))  
(define y (copy-list x))  
1]=> x  
;Value: ((a b) c)  
1]=> y  
;Value: ((a b) c)
```

# Example



# Example

- Next we will evaluate the expression  
`(set-car! (car x) 'wow)`
- What will the interpreter print as the values of `x` and `y`?
- Draw a box and pointer diagram to explain your answer.

## Example

```
(set-car! (car x) 'wow)
```

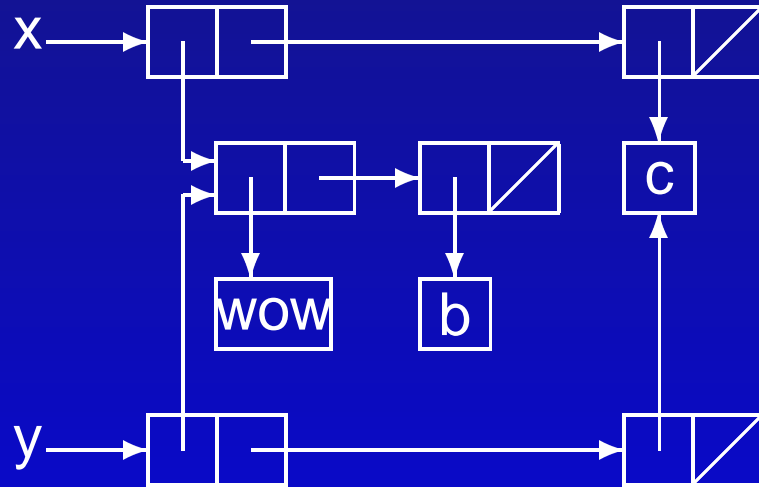
```
==> x
```

```
;Value: ((wow b) c)
```

```
==> y
```

```
;Value: ((wow b) c)
```

# Example



# Example

- Consider the following procedure for counting pairs

```
(define (count-pairs x)
  (if (atom? x)
      0
      (+ (count-pairs (car x))
         (count-pairs (cdr x))
         1)))
```

# Example

- Draw box and pointer diagrams representing list structures made up of exactly three pairs for which this procedure would
  - return 3
  - return 4
  - return 7
  - never returns at all

# Example

- Given

```
(define x '(1 2 3))
```

- Give sequences of mutative expressions to create the four box and pointer models given above