Managing Memory
6.037 – Structure and Interpretation of Computer Programs

Mike Phillips
Massachusetts Institute of Technology

Lecture 8B
Expanding our horizons

So far, we’ve examined just a bit of the overall Scheme story.
Expanding our horizons

So far, we’ve examined just a bit of the overall Scheme story.

primitive data structures (e.g., \texttt{cons cells})
Expanding our horizons

So far, we’ve examined just a bit of the overall Scheme story.

- complex data structures
- primitive data structures (e.g., cons cells)
Expanding our horizons

So far, we’ve examined just a bit of the overall Scheme story.

math

↑

lambda calculus
complex data structures
primitive data structures (e.g., cons cells)
RAM

electrical engineering
Implementing cons-cell memory

We’ve been using the cons-cell abstraction this whole class.

Computer memory doesn’t really work like that.
Conventional memory is an array of locations, each of which has an integer address, and stores a single value.

Addresses are sequential, so we often move around memory by adding and subtracting values from addresses.
Vectors

- We will model memory using **vectors**.
- Also a generally-useful data structure (similar to arrays in other languages).
- Vectors support **constant-time** access of an arbitrary element.
Vector Operations

- `(make-vector <size>) → <v>`
  - Returns a vector of the given size.

- `(vector-ref <v> <n>) → <elt>`
  - Returns the element at index `n` of `v` (0-indexed).

- `(vector-set! <v> <n> <val>) → undefined`
  - Sets the element at index `n` of `v`.

- `(vector-length <v>) → <size>`
Vector Operations

- `(make-vector <size>) → <v>`
  - Returns a vector of the given size.
- `(vector-ref <v> <n>) → <elt>`
  - Return the element at index n of v (0-indexed)
Vector Operations

- \( \text{(make-vector } \langle \text{size} \rangle) \rightarrow \langle \text{v} \rangle \)
  - Returns a vector of the given size.
- \( \text{(vector-ref } \langle \text{v} \rangle \langle n \rangle) \rightarrow \langle \text{elt} \rangle \)
  - Return the element at index \( n \) of \( v \) (0-indexed)
- \( \text{(vector-set! } \langle \text{v} \rangle \langle n \rangle \langle \text{val} \rangle) \rightarrow \text{undefined} \)
  - Sets the element at index \( n \) of \( v \).
Vector Operations

- \((\text{make-vector } <\text{size}> ) → <\text{v}>\)
  - Returns a vector of the given size.
- \((\text{vector-ref } <\text{v}> <\text{n}> ) → <\text{elt}>\)
  - Return the element at index \(n\) of \(v\) (0-indexed)
- \((\text{vector-set! } <\text{v}> <\text{n}> <\text{val}> ) → \text{undefined}\)
  - Sets the element at index \(n\) of \(v\).
- \((\text{vector-length } <\text{v}> ) → <\text{size}>\)
## Vectors and Lists

<table>
<thead>
<tr>
<th>Lists</th>
<th>Vectors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant-time append at the beginning</td>
<td>No append at all</td>
</tr>
<tr>
<td>Constant-time insert at any point (with mutation)</td>
<td>No insert at all</td>
</tr>
<tr>
<td>Accessing the ( n^{th} ) element takes ( O(n) )</td>
<td>Accessing the ( n^{th} ) element takes constant time</td>
</tr>
<tr>
<td>Structure can be shared between different lists</td>
<td>Every vector is entirely disjoint</td>
</tr>
<tr>
<td>Rich set of built-in procedures (\texttt{map}, etc.)</td>
<td>Few built-ins (but you can build more)</td>
</tr>
</tbody>
</table>
We will represent cons cells using two vectors, the-cars and the-cdrs.
Representing cons cells

- We will represent cons cells using two vectors, the-cars and the-cdrs.
- A cons cell is an index $i$ into the arrays
  - Its car is (vector-ref the-cars i)
  - Its cdr is (vector-ref the-cdrs i)
We will represent cons cells using two vectors, the-cars and the-cdrs.

A cons cell is an index $i$ into the arrays

- Its car is (vector-ref the-cars i)
- Its cdr is (vector-ref the-cdrs i)

To represent other data, we’ll use tagging.
We will represent cons cells using two vectors, the-cars and the-cdrs.

A cons cell is an index \( i \) into the arrays

- **Its car is** \((\text{vector-ref the-cars } i)\)
- **Its cdr is** \((\text{vector-ref the-cdrs } i)\)

To represent other data, we’ll use tagging.

- **\( n_i \) is a number with value** \( i \)
- **\( p_i \) is a pointer to a pair at index** \( i \)
- **\( e_0 \) is the special empty list**
An example

```
((1 2) 3 4)
```

Diagram showing a linked list representation of the list `((1 2) 3 4)`.
An example

((1 2) 3 4)

Diagram of a tree with nodes labeled 1, 2, 3, and 4.
An example

$((1\ 2)\ 3\ 4) \mapsto p1$

```
    1
   / \   /
  2   3   4
  / \  /  /
 1   2 3   4
  5  7 6  8
```

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>the-cars</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>…</td>
</tr>
<tr>
<td>the-cdrs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>…</td>
</tr>
</tbody>
</table>
An example

\[(1\ 2)\ 3\ 4) \mapsto p1\]

Index 0 1 2 3 4 5 6 7 8 . . .

<table>
<thead>
<tr>
<th>the-cars</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>the-cdrs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
An example

\[(1\ 2)\ 3\ 4) \mapsto p1\]

Index 0 1 2 3 4 5 6 7 8 ...  
the-cars p5 ... ... ... ... 
the-cdrs ... ... ... ... ...

Mike Phillips (MIT)  
Managing Memory  
Lecture 8B
An example

\[(1 \ 2) \ 3 \ 4) \mapsto p1\]

```
Index 0 1 2 3 4 5 6 7 8 ...
the-cars     p5           ...
the-cdres    p2           ...
```
An example

\[(1 \ 2) \ 3 \ 4) \ maps \ to \ p_1\]

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>the-cars</td>
<td></td>
<td>p5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>…</td>
</tr>
<tr>
<td>the-cdrs</td>
<td>p2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>…</td>
</tr>
</tbody>
</table>
An example

\[( (1 \ 2) \ 3 \ 4) \mapsto \text{p1} \]

```
<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>the-cars</td>
<td>p5</td>
<td>n3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>the-cdrs</td>
<td>p2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
```
An example

\[((1 \ 2) \ 3 \ 4) \mapsto p1\]

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>the-cars</td>
<td>p5</td>
<td>n3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>…</td>
</tr>
<tr>
<td>the-cdhrs</td>
<td>p2</td>
<td>p4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>…</td>
</tr>
</tbody>
</table>
An example

$$((1\ 2)\ 3\ 4) \mapsto p1$$

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>the-cars</td>
<td></td>
<td>p5</td>
<td>n3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>…</td>
</tr>
<tr>
<td>the-cdrs</td>
<td>p2</td>
<td>p4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>…</td>
</tr>
</tbody>
</table>
An example

$$((1\ 2)\ 3\ 4) \mapsto p_1$$

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>the-cars</td>
<td>p5</td>
<td>n3</td>
<td>n4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>…</td>
</tr>
<tr>
<td>the-cdrs</td>
<td>p2</td>
<td>p4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>…</td>
</tr>
</tbody>
</table>
An example

```
((1 2) 3 4) \mapsto p1
```

```
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | ...
|-------|---|---|---|---|---|---|---|---|---|-----
| the-cars | p5 | n3 | n4 |   |   |   |   |   |   | ...
| the-cdrs  | p2 | p4 | e0 |   |   |   |   |   |   | ...
```
An example

\(( (1 \ 2) \ 3 \ 4) \mapsto p1 \)
An example

$((1 \ 2) \ 3 \ 4) \mapsto p1$

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>the-cars</td>
<td>p5</td>
<td>n3</td>
<td>n4</td>
<td>n1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>the-cdrs</td>
<td>p2</td>
<td>p4</td>
<td>e0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
An example

(((1 2) 3 4) ↦→ p1)

Index 0 1 2 3 4 5 6 7 8 ...
the-cars p5 n3 n4 n1 ...
the-cdr s p2 p4 e0 p7 ...

Mike Phillips (MIT)  
Managing Memory  
Lecture 8B  
9 / 1
An example

\[(1\ 2) \ 3 \ 4) \mapsto p1\]

Index 0 1 2 3 4 5 6 7 8 …
the-cars p5 n3 n4 n1 p2 p4 e0 p7 …
the-cdrs p2 n3 n4 n1 p4 p5 e0 …
An example

\((1 \ 2) \ 3 \ 4) \leftrightarrow p1\)

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
\downarrow & \downarrow & \downarrow & \downarrow \\
1 & 2 & 3 & 4 \\
\end{array}
\]

\[
\begin{array}{cccc}
5 & 7 & 3 & 4 \\
\downarrow & \downarrow & \downarrow & \downarrow \\
1 & 2 & 3 & 4 \\
\end{array}
\]

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>the-cars</td>
<td>p5</td>
<td>n3</td>
<td>n4</td>
<td>n1</td>
<td>n2</td>
<td>…</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>the-cdrs</td>
<td>p2</td>
<td>p4</td>
<td>e0</td>
<td>p7</td>
<td>…</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
An example

$$((1 \ 2) \ 3 \ 4) \mapsto p_1$$

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>the-cars</td>
<td>p5</td>
<td>n3</td>
<td>n4</td>
<td>n1</td>
<td>n2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>…</td>
</tr>
<tr>
<td>the-cdrs</td>
<td>p2</td>
<td>p4</td>
<td>e0</td>
<td>p7</td>
<td>e0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>…</td>
</tr>
</tbody>
</table>
An example

\(((1 \ 2) \ 3 \ 4) \mapsto p1\)

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>the-cars</td>
<td>p5</td>
<td>n3</td>
<td>n4</td>
<td>n1</td>
<td>n2</td>
<td>…</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>the-cdhrs</td>
<td>p2</td>
<td>p4</td>
<td>e0</td>
<td>p7</td>
<td>e0</td>
<td>…</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
(define (gc-car pair)  
  (vector-ref the-cars (pointer-value pair)))

(define (gc-cdr pair)  
  (vector-ref the-cdhrs (pointer-value pair)))

(define (gc-set-car! pair new-car)  
  (vector-set! the-cars  
    (pointer-value pair) new-car))

(define (gc-set-cdr! pair new-cdr)  
  (vector-set! the-cdhrs  
    (pointer-value pair) new-cdr))
(define (gc-cons car cdr)
  (let ((pair (gc-new-pair)))
    (gc-set-car! pair car)
    (gc-set-cdr! pair car)
    pair))
(define *free* 0)
(define (gc-new-pair)
  (let ((new-pair *free*))
    (set! *free* (+ *free* 1))
    (tag-pointer 'pair new-pair)))
(define *free* 0)
(define (gc-new-pair)
  (let ((new-pair *free*))
    (set! *free* (+ *free* 1))
    (tag-pointer 'pair new-pair)))

- What’s wrong?
(define (find-primes n)
  (define (helper ns)
    (cons (car ns) (find-primes
                   (filter (lambda (i) (not (divides? i n))) (cdr ns)))
          (helper (cdr (integers-less-than n)))))

  (2 3 4 5 6 7 8 9 10 11 12 ...)
  (3 5 7 9 11 13 15 17 19 21 ...)
  (5 7 11 13 17 19 23 25 27 ...)}
Every filter step generates intermediate lists
But those lists can never be accessed again!
We can re-use that storage space
The Big Idea

- We can **simulate** a machine with infinite memory by detecting and re-using memory that can never be used again.
- How do we do that?
Reachability

- There is a set of objects (the “root set”) the program can directly access (e.g. the global environment)
- Objects can point to other objects (e.g. cons cells, the environment pointer of a lambda)
- Any object that is transitively reachable by following pointers from the root set is **live** and must be preserved.
- Anything else is **garbage** and can be reused.
First try: Reference Counting

- We could keep track of how many pointers there are to each object.
- Every time we generate a new reference to an object, we increase the reference count.
  - `define`
  - `set!`
  - `apply` a compound procedure
  - ...
- Whenever we remove a reference to an object, decrease the count.
  - `set!` (The old value)
  - After applying a compound procedure.
  - ...

Mike Phillips (MIT)
Managing Memory
Lecture 8B
Reference Counting: Problems

- Naïve refcounting leaks circular objects!
  ```scheme
  (define x (list 'a))
  (set-cdr! x x)
  (set! x 0)
  ```

- Performance impact in many cases.
  - Every time you leave a frame, you need to walk its variables
Describe the “root set” explicitly.

- On real hardware, this is the “registers”
- In `m-eval` this is (roughly) the global environment plus the current environment.

Only objects reachable from this set by some sequence of `car` and `cdr` can ever matter.

Any memory that is not accessible in this way is garbage, and can be reused.
mark-and-sweep is one of the simplest garbage collection algorithms, composed of two phases:

1. Starting from the root set, recursively mark every reachable object.
2. Sweep all of memory, collecting every unmarked object into the free list.
Allocation then takes place by removing new pairs from the free list.
Mark-and-sweep is one of the simplest garbage collection algorithms, composed of two phases:

1. Starting from the root set, recursively mark every reachable object.
mark-and-sweep

mark-and-sweep is one of the simplest garbage collection algorithms, composed of two phases:

1. Starting from the root set, recursively mark every reachable object.
2. Sweep all of memory, collecting every unmarked object into the free list.
mark-and-sweep

mark-and-sweep is one of the simplest garbage collection algorithms, composed of two phases:

1. Starting from the root set, recursively mark every reachable object.
2. Sweep all of memory, collecting every unmarked object into the free list.

Allocation then takes place by removing new pairs from the free list.
Index 0 1 2 3 4 5 6 7 8 ...  
the-cars  p5  n3  n4  n1  n2  ...  
the-cdhrs  p2  p4  e0  p7  e0  ...  
the-marks  ⇆
Index 0 1 2 3 4 5 6 7 8 ...
the-cars p5 n3 n4 n1 n2 ...
the-cdrsp2 p4 e0 p7 e0 ...
the-marks #t ↑
Index  0  1  2  3  4  5  6  7  8  ...  
the-cars | p5 | n3 | n4 | n1 | n2 | ...  
the-cdrs | p2 | p4 | e0 | p7 | e0 | ...  
the-marks | #t | #t | #t | #t | ...  

Mike Phillips (MIT)  Managing Memory  Lecture 8B 21 / 1
Index 0 1 2 3 4 5 6 7 8 ...
the-cars    p5  n3  n4  n1  n2  ...
the-cdrs    p2  p4  e0  p7  e0  ...
the-marks   #t  #t  #t  #t  #t  ...

root

1
2
3
4
5
7

Mike Phillips (MIT)
Managing Memory
Lecture 8B
Index 0 1 2 3 4 5 6 7 8 ...

the-cars      p5  n3 | n4  n1 | n2 | ...

the-cdr      p2  p4 | e0  p7 | e0 | ...

the-marks    #t  #t | #t  #t | #t | ...
Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | ...  
---|---|---|---|---|---|---|---|---|---|---  
the-cars | | p5 | n3 | | n4 | n1 | | n2 | | ...  
the-cdtrs | | p2 | p4 | | e0 | p7 | | e0 | | ...  
the-marks | #t | #t | | #t | #t | #t | | #t | | ...
### Index 0 1 2 3 4 5 6 7 8 ...

<table>
<thead>
<tr>
<th>the-cars</th>
<th>p5</th>
<th>n3</th>
<th>n4</th>
<th>n1</th>
<th>n2</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>the-cdhrs</td>
<td>p2</td>
<td>p4</td>
<td>e0</td>
<td>p7</td>
<td>e0</td>
<td>...</td>
</tr>
<tr>
<td>the-marks</td>
<td>#t</td>
<td>#t</td>
<td>#t</td>
<td>#t</td>
<td>#t</td>
<td>...</td>
</tr>
</tbody>
</table>

↑
free-list →

| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | ...
|-------|---|---|---|---|---|---|---|---|---|-----|
| the-cars | p5 | n3 | n4 | n1 | n2 | | | | | ...
| the-cdrs | p2 | p4 | e0 | p7 | e0 | | | | | ...
| the-marks | #t | #t | #t | #t | #t | | | | | ↑ |
The diagram illustrates a memory management structure, possibly for garbage collection. The structure is labeled as `root` and consists of nodes labeled as `1`, `2`, `3`, `4`, `5`, and `7`. There is also a `free-list` labeled as `8`.

A table is also shown with the following columns:

- **Index**: 0, 1, 2, 3, 4, 5, 6, 7, 8, ...
- **the-cars**: p5, n3, n4, n1, n2, e0, ...
- **the-cdrs**: p2, p4, e0, p7, e0, e0, ...
- **the-marks**: #t, #t, #t, #t, #t, #t, ...

The table indicates various pointers and values associated with different index positions.
Mike Phillips (MIT)  
Managing Memory  
Lecture 8B  

```
<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>the-cars</td>
<td>p5</td>
<td>n3</td>
<td>n4</td>
<td>n1</td>
<td>n2</td>
<td>e0</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>the-cdrs</td>
<td>p2</td>
<td>p4</td>
<td>e0</td>
<td>p7</td>
<td>e0</td>
<td>e0</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>the-marks</td>
<td>#t</td>
<td>#t</td>
<td>#t</td>
<td>#t</td>
<td>#t</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

The diagram shows a data structure with nodes labeled 'root', 'free-list', and various nodes numbered 1 to 8. The nodes are connected with arrows indicating the flow or relationship between them. The diagram also includes a table with columns labeled 'Index', 'the-cars', 'the-cdrs', and 'the-marks', containing values such as 'p5', 'n3', 'n4', 'n1', 'p2', 'p4', 'e0', 'p7', '#t', and 'e0'.
root → 1 → 2 → 3 → 4

free-list →

| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | ...
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>the-cars</td>
<td>p5</td>
<td>n3</td>
<td>n4</td>
<td>n1</td>
<td>n2</td>
<td>e0</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>the-cdrs</td>
<td>p2</td>
<td>p4</td>
<td>e0</td>
<td>p7</td>
<td>e0</td>
<td>e0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>the-marks</td>
<td>#t</td>
<td>#t</td>
<td>#t</td>
<td>#t</td>
<td>#t</td>
<td>#t</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Index 0 1 2 3 4 5 6 7 8 ...

the-cars

| p5 | n3 | n4 | n1 | e0 | n2 | e0 | ...

the-cdrs

| p2 | p4 | e0 | p7 | p8 | e0 | e0 | ...

the-marks

| #t | #t | #t | #t | #t | #t | #t |
The diagram illustrates a memory management structure with a free-list and two linked lists, `the-cars` and `the-cdrs`. The free-list contains nodes that can be reused, and the linked lists store pointers to memory blocks.

### Index Table

<table>
<thead>
<tr>
<th>Index</th>
<th>the-cars</th>
<th>the-cdrs</th>
<th>the-marks</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>p5</td>
<td>p2</td>
<td>#t</td>
</tr>
<tr>
<td>1</td>
<td>n3</td>
<td>p4</td>
<td>#t</td>
</tr>
<tr>
<td>2</td>
<td>n4</td>
<td>e0</td>
<td>#t</td>
</tr>
<tr>
<td>3</td>
<td>n1</td>
<td>p7</td>
<td>#t</td>
</tr>
<tr>
<td>4</td>
<td>e0</td>
<td>p8</td>
<td>#t</td>
</tr>
<tr>
<td>5</td>
<td>n2</td>
<td>e0</td>
<td>#t</td>
</tr>
<tr>
<td>6</td>
<td>e0</td>
<td>e0</td>
<td>#t</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
root

free-list

Index 0 1 2 3 4 5 6 7 8 ...
the-cars p5 n3 e0 n4 n1 e0 n2 e0 ...
the-cdrs p2 p4 p6 e0 p7 p8 e0 e0 ...
the-marks #t #t #t #t #t #t #t #t #t
root

free-list

Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | ...
---|---|---|---|---|---|---|---|---|---|---
the-cars | p5 | n3 | e0 | n4 | n1 | e0 | n2 | e0 | ... |
the-cdrs | p2 | p4 | p6 | e0 | p7 | p8 | e0 | e0 | ... |
the-marks | #t | #t | #t | #t | #t | #t | #t | #t | ... |
root

free-list

Index 0 1 2 3 4 5 6 7 8 ...
the-cars
the-cdrs
the-marks

↑
Index 0 1 2 3 4 5 6 7 8 ...
the-cars e0 p5 n3 e0 n4 n1 e0 n2 e0 ...
the-cdr p3 p2 p4 p6 e0 p7 p8 e0 e0 ...
the-marks #t #t #t #t #t #t #t
↑
(define (mark p)
    (if (and (gc-pair? p)
               (not (vector-ref the-marks (pointer-value p))))
        (begin
         (vector-set! the-marks (pointer-value p) #t)
         (mark (gc-car p))
         (mark (gc-cdr p))))
)
(define (sweep i)
  (if (not (vector-ref the-marks i))
    (begin
      (vector-set! the-cars i *gc-nil*)
      (vector-set! the-cdrs i *free-list*)
      (set! *free-list* (tag-pointer 'pair i)))
    (if (> i 0)
      (sweep (- i 1)))))
(define (mark-and-sweep root)
  (clear-all-marks)
  (mark root)
  (set! *free-list* *gc-nil*)
  (sweep (- *memory-size* 1)))
(define (mark-and-sweep-new-pair)
  (if (eq? *free-list* *gc-nil*)
      (begin (mark-and-sweep root)
              (if (eq? *free-list* *gc-nil*)
                  (error "Out of memory")))
      (let ((pair *free-list*))
          (set! *free-list* (gc-cdr *free-list*)
               pair)))
mark-and-sweep: problems

How do we keep track of state during mark?
mark-and-sweep: problems

- How do we keep track of state during mark?
- sweep needs to examine all of memory.
mark-and-sweep: problems

- How do we keep track of state during mark?
- sweep needs to examine all of memory.
- Heap fragmentation becomes a big problem.
An alternate plan: Stop-and-copy

To solve these problems, many real systems use some form of a copying garbage collector.
To solve these problems, many real systems use some form of a copying garbage collector.

Our stop-and-copy collector maintains two regions of memory, the working memory and the free memory.
An alternate plan: Stop-and-copy

To solve these problems, many real systems use some form of a copying garbage collector.

Our stop-and-copy collector maintains two regions of memory, the working memory and the free memory.

When we run out of memory, we copy live objects into the free memory, and switch the roles of the halves.
We allocate pairs as we did initially with a *free* pointer.
Stop-and-Copy

- We allocate pairs as we did initially with a *free* pointer.
- When we run out of memory, we switch the free and working memories, and we relocate root into the new free memory.
Stop-and-Copy

- We allocate pairs as we did initially with a *free* pointer.
- When we run out of memory, we switch the free and working memories, and we **relocate** root into the new free memory.
- We use a new pointer, scan, initially pointing at the start of the new free memory.
- As long as scan < *free*, we relocate the car and cdr of scan, and increment scan.
To relocate a pointer:

- If the value it points to has already been copied, update it to point at the new location.
- Otherwise, allocate a new pair, copy the pair it points to there, and update it.
To relocate a pointer:
- If the value it points to has already been copied, update it to point at the new location.
- Otherwise, allocate a new pair, copy the pair it points to there, and update it.
  - Replace the car of the old pair with a tag known as a broken heart.
  - Replace the cdr of the old pair with the pair’s new address.
root: p1

| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | ...
|-------|---|---|---|---|---|---|---|---|---|-----|
| the-cars | p4 | n3 | n1 | n2 | p6 |   |   |   |   | ...
| the-cdr | p7 | e0 | p6 | e0 | p3 |   |   |   |   | ...

Mike Phillips (MIT)  
Managing Memory  
Lecture 8B  
30 / 1
root: p1

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>the-cars</td>
<td>p4</td>
<td>n3</td>
<td>n1</td>
<td>n2</td>
<td>p6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>the-cdrs</td>
<td>p7</td>
<td>e0</td>
<td>p6</td>
<td>e0</td>
<td>p3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*free*

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>new-cars</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>new-cdrs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*scan*
root: p1

| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | ...
|-------|---|---|---|---|---|---|---|---|---|------
| the-cars | p4 | n3 | n1 | n2 | p6 |   |   |   |   | ...
| the-cdhrs | p7 | e0 | p6 | e0 | p3 |   |   |   |   | ...

*free* ↓

| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | ...
|-------|---|---|---|---|---|---|---|---|---|------
| new-cars |   |   |   |   |   |   |   |   |   |   ...
| new-cdhrs |   |   |   |   |   |   |   |   |   |   ...

*scan* ↑
root: p0

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>the-cars</td>
<td>p4</td>
<td>n3</td>
<td>n1</td>
<td>n2</td>
<td>p6</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>the-cdrs</td>
<td>p7</td>
<td>e0</td>
<td>p6</td>
<td>e0</td>
<td>p3</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*free*

↓

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>new-cars</td>
<td>p4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>new-cdrs</td>
<td>p7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*scan*
root: p0

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>the-cars</td>
<td></td>
<td>n3</td>
<td>n1</td>
<td>n2</td>
<td>p6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>the-cdr</td>
<td>p0</td>
<td>e0</td>
<td>p6</td>
<td>e0</td>
<td>p3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*free*

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>new-cars</td>
<td>p4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>new-cdr</td>
<td>p7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
root: p0

```
<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>the-cars</td>
<td></td>
<td>n3</td>
<td>n1</td>
<td>n2</td>
<td>p6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>the-cdrs</td>
<td>p0</td>
<td>e0</td>
<td>p6</td>
<td>e0</td>
<td>p3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*free*

```

```
<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>new-cars</td>
<td>p4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>new-cdrs</td>
<td>p7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*scan*

```
root: p0

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>the-cars</td>
<td>p0</td>
<td>e0</td>
<td>p6</td>
<td>e0</td>
<td>p3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>the-cdrs</td>
<td>n3</td>
<td>n1</td>
<td>n2</td>
<td>p6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*free*

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>new-cars</td>
<td>p1</td>
<td>n1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>new-cdrs</td>
<td>p7</td>
<td>p6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*scan*
root: p0

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>the-cars</td>
<td></td>
<td></td>
<td>n3</td>
<td></td>
<td>n2</td>
<td>p6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>the-cdr</td>
<td>p0</td>
<td></td>
<td>e0</td>
<td>p1</td>
<td>e0</td>
<td>p3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*free*

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>new-cars</td>
<td>p1</td>
<td>n1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>new-cdr</td>
<td>p7</td>
<td>p6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*scan*
### root: p0

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>the-cars</td>
<td></td>
<td>n3</td>
<td></td>
<td>n2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>the-cdrs</td>
<td>p0</td>
<td>e0</td>
<td>p1</td>
<td>e0</td>
<td>p2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

*free*

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>new-cars</td>
<td>p1</td>
<td>n1</td>
<td>p6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>new-cdrs</td>
<td>p2</td>
<td>p6</td>
<td>p3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

*scan*
root: p0

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>the-cars</td>
<td></td>
<td></td>
<td>n3</td>
<td></td>
<td>n2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>the-cdhrs</td>
<td>p0</td>
<td>e0</td>
<td>p1</td>
<td>e0</td>
<td>p2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*free*

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>new-cars</td>
<td>p1</td>
<td>n1</td>
<td>p6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>new-cdhrs</td>
<td>p2</td>
<td>p6</td>
<td>p3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*scan*
root: p0

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>the-cars</td>
<td></td>
<td>n3</td>
<td>p0</td>
<td></td>
<td>p1</td>
<td>p3</td>
<td>p2</td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>the-cdrs</td>
<td>p0</td>
<td>e0</td>
<td>p1</td>
<td>p3</td>
<td>p2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

*free*

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>new-cars</td>
<td>p1</td>
<td>n1</td>
<td>p6</td>
<td>n2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>new-cdrs</td>
<td>p2</td>
<td>p3</td>
<td>p3</td>
<td>e0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

*scan*
root: p0

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>the-cars</td>
<td></td>
<td>n3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>the-cdrs</td>
<td>p0</td>
<td>e0</td>
<td>p1</td>
<td>p3</td>
<td>p2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

*free*

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>new-cars</td>
<td>p1</td>
<td>n1</td>
<td>p6</td>
<td>n2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>new-cdrs</td>
<td>p2</td>
<td>p3</td>
<td>p3</td>
<td>e0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

*scan*
root: p0

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>the-cars</td>
<td>*</td>
<td>n3</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>the-cdrs</td>
<td>p0</td>
<td>e0</td>
<td>p1</td>
<td>p3</td>
<td>p2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

*free*

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>new-cars</td>
<td>p1</td>
<td>n1</td>
<td>p3</td>
<td>n2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>new-cdrs</td>
<td>p2</td>
<td>p3</td>
<td>p3</td>
<td>e0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

*scan*
root: p0

```
<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
</table>
| the-cars |   |   |   |   |   |   |   |   |   |   ...
| the-cdtrs | p0 | p4 | p1 | p3 | p2 |   |   |   |   |   ...
```

*free*

```
<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
</table>
| new-cars | p1 | n1 | p3 | n2 | n3 |   |   |   |   |   ...
| new-cdtrs | p2 | p3 | p4 | e0 | e0 |   |   |   |   |   ...
```

*scan*
root: p0

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>the-cars</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>the-cdrs</td>
<td>p0</td>
<td>p4</td>
<td>p1</td>
<td>p3</td>
<td>p2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*free*

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>new-cars</td>
<td>p1</td>
<td>n1</td>
<td>p3</td>
<td>n2</td>
<td>n3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>new-cdrs</td>
<td>p2</td>
<td>p3</td>
<td>p4</td>
<td>e0</td>
<td>e0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*scan*
**root:**  p0

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>the-cars</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><img src="image1" alt="Red Ball" /></td>
<td><img src="image2" alt="Red Ball" /></td>
<td><img src="image3" alt="Red Ball" /></td>
<td><img src="image4" alt="Red Ball" /></td>
<td><img src="image5" alt="Red Ball" /></td>
<td><img src="image6" alt="Red Ball" /></td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>the-cdrs</strong></td>
<td>p0</td>
<td>p4</td>
<td>p1</td>
<td>p3</td>
<td>p2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*free*

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>new-cars</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p1</td>
<td>n1</td>
<td>p3</td>
<td>n2</td>
<td>n3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>new-cdrs</strong></td>
<td>p2</td>
<td>p3</td>
<td>p4</td>
<td>e0</td>
<td>e0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*scan*
root: p0

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>the-cars</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>the-cdrs</td>
<td>p0</td>
<td>p4</td>
<td>p1</td>
<td>p3</td>
<td>p2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*free*

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>new-cars</td>
<td>p1</td>
<td>n1</td>
<td>p3</td>
<td>n2</td>
<td>n3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>new-cdrs</td>
<td>p2</td>
<td>p3</td>
<td>p4</td>
<td>e0</td>
<td>e0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*scan*
root: p0

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>new-cars</td>
<td>p</td>
<td>n</td>
<td>p</td>
<td>n</td>
<td>p</td>
<td>n</td>
<td>p</td>
<td>n</td>
<td>p</td>
<td>...</td>
</tr>
<tr>
<td>new-cdrys</td>
<td>p0</td>
<td>p4</td>
<td>p1</td>
<td>p3</td>
<td>p2</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*free*

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>the-cars</td>
<td>p</td>
<td>n</td>
<td>p</td>
<td>n</td>
<td>p</td>
<td>n</td>
<td>p</td>
<td>n</td>
<td>p</td>
<td>...</td>
</tr>
<tr>
<td>the-cdrys</td>
<td>p</td>
<td>p</td>
<td>p</td>
<td>e</td>
<td>e</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*scan*
(define (stop-and-copy)
    (define (loop scan)
        (if (< scan *free*)
            (begin
                (vector-set! new-cars scan
                                (relocate
                                    (vector-ref new-cars scan)))
                (vector-set! new-cdrs scan
                                (relocate
                                    (vector-ref new-cdrs scan)))
                (loop (+ scan 1)))))
    (set! *free* 0)
    (set! *root* (relocate *root*))
    (loop 0)
    (swap-spaces))
(define (relocate ptr)
  (if (gc-pair? ptr)
      (if (broken-heart? (gc-car pair))
          (gc-cdr pair)
          (let ((new-pair *free*))
            (set! *free* (+ 1 *free*))
            (vector-set! new-cars new-pair
                        (gc-car ptr))
            (vector-set! new-cdhrs new-pair
                        (gc-cdr ptr))
            (gc-set-car! ptr *broken-heart*)
            (gc-set-cdr! ptr
                        (tag-pointer 'pair new-pair))
            (tag-pointer 'pair new-pair)))
      ptr))
Properties of stop-and-copy

- Since it moves things around, the garbage collector must know about every pointer into the heap.
- Compacts used memory into a single chunk
  - This means allocation is extremely efficient.
- You only get to use half of your memory.
  - But with mark-and-sweep you potentially needed that for the stack.
- Most modern GCs use something that looks more like stop-and-copy than mark-and-sweep.
Think about the kinds of garbage a program creates.

- `find-primes` generated a lot of garbage, but it was very short-lived.

- In the adventure game, players, brains and items are created and destroyed, but tend to last a while first.

- This turns out to be true in general: A large amount of garbage is destroyed very quickly, whereas garbage that sticks around for a while is likely to stick around more.
Generational GC

- Big Idea: Have two (or more!) memory pools.
- Allocate everything into a small one, and scan it every time you do a GC.
- If an object survives a few garbage collections, move it into a larger pool, which is only fully scanned rarely.
- Nearly every real modern GC works roughly this way.