Copying
Garbage Collection
Two-Space Copying Collectors

A two-space copying collector compacts memory as it collects, making allocation easier.

Allocator:

• Partitions memory into to-space and from-space
• Allocates only in to-space

Collector:

• Starts by swapping to-space and from-space
• Coloring gray ⇒ copy from from-space to to-space
• Choosing gray records ⇒ go through the new to-space, update pointers
Two-Space Collection

Left = from-space
Right = to-space
Two-Space Collection

Mark gray = copy and leave forward address
Two-Space Collection

Choose gray by walking through to-space

Diagram showing a two-space collection with arrows indicating movement through the spaces.
Two-Space Collection

Mark referenced as gray
Two-Space Collection

Mark black = move gray-choosing arrow
Two-Space Collection

Nothing to color gray; increment the arrow
Two-Space Collection

Color referenced record gray
Two-Space Collection

Increment the gray-choosing arrow
Two-Space Collection

Referenced is already copied, use forwarding address
Two-Space Collection

Choosing arrow reaches the end of to-space: done
Two-Space Collection

Right = from-space
Left = to-space
Two-Space Collection

- Cool diagrams, bro
- But what does that look like for an actual heap?
- Like, say, in `plai/gc2`?
- So let’s go through a more concrete example
- But the actual `plai/gc2` implementation is your job for HW8
The Setup

• Each object in memory starts with a tag
  ○ Just like in plai/gc2

• Tags tell us how to interpret the heap cells that follow
  ○ How many cells are part of the object?
  ○ Which cells hold pointers?
  ○ Which cells hold flat data?
  ○ Just like in plai/gc2
The Setup

• The kinds of objects we’ll be dealing with are simplified variants of the ones in plai/gc2

• Flat data will be integers only, to keep things simple

• Tags will be numbers, not symbols
  ○ Like real GCs, but unlike plai/gc2

• Tag 1: one integer
  ○ Simpler variant of 'flat

• Tag 2: one pointer
  ○ Simpler variant of 'pair

• Tag 3: one integer, then one pointer
  ○ Simpler variant of 'proc

• Tag 99: forwarding pointer (one pointer)
The Strategy

• Traverse the heap, starting at the roots, using breadth-first search
  ○ In contrast, mark-and-sweep uses depth-first

• Visiting a node = marking it gray
  ○ = copying from the from-space to the to-space
  ○ + leaving a forwarding pointer behind in the from-space
The Strategy

• Maintain a queue of the gray nodes in the to-space
  ◦ Marking a node gray → adding it to the queue
  ◦ Taking a node out of the queue → marking it black

• Use that queue to keep track of the BFS

• Invariant:
  ◦ objects in the queue have pointers to the from-space;
  ◦ objects outside the queue (black) have pointers to the to-space

• Represent the queue as two pointers into the to-space
  ◦ Increment the end pointer when enqueuing
  ◦ Increment the front pointer when dequeuing
  ◦ When the two pointers come together, we’re done
Two-Space Collection Example

• 26-byte memory (13 bytes per space), 2 roots
  ◦ Tag 1: one integer
  ◦ Tag 2: one pointer
  ◦ Tag 3: one integer, then one pointer

Root 1: 7   Root 2: 0

From:  1 75 2 0 3 2 10 3 2 2 3 1 4
Two-Space Collection Example

• 26-byte memory (13 bytes per space), 2 roots
  ○ Tag 1: one integer
  ○ Tag 2: one pointer
  ○ Tag 3: one integer, then one pointer

Root 1: 7  
Root 2: 0

From: 1 75 2 0 3 2 10 3 2 2 3 1 4
Addr: 00 01 02 03 04 05 06 07 08 09 10 11 12
Two-Space Collection Example

- 26-byte memory (13 bytes per space), 2 roots
  - Tag 1: one integer
  - Tag 2: one pointer
  - Tag 3: one integer, then one pointer

Root 1: 7  
Root 2: 0  
From: 1 75 2 0 3 2 10 3 2 2 3 1 4  
Addr: 00 01 02 03 04 05 06 07 08 09 10 11 12  
    ^     ^        ^        ^        ^        ^
Two-Space Collection Example

- 26-byte memory (13 bytes per space), 2 roots
  - Tag 1: one integer
  - Tag 2: one pointer
  - Tag 3: one integer, then one pointer

<table>
<thead>
<tr>
<th>From:</th>
<th>1</th>
<th>75</th>
<th>2</th>
<th>0</th>
<th>3</th>
<th>2</th>
<th>10</th>
<th>3</th>
<th>2</th>
<th>2</th>
<th>3</th>
<th>1</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addr:</td>
<td>00</td>
<td>01</td>
<td>02</td>
<td>03</td>
<td>04</td>
<td>05</td>
<td>06</td>
<td>07</td>
<td>08</td>
<td>09</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>^</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>To:</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Q:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Addr:</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
</tr>
</tbody>
</table>
Two-Space Collection Example

- 26-byte memory (13 bytes per space), 2 roots
  - Tag 1: one integer
  - Tag 2: one pointer
  - Tag 3: one integer, then one pointer
  - Tag 99: forwarding pointer (to to-space)

Root 1: 13     Root 2: 0

From:   1  75  2  0  3  2  10  99  13  2  3  1  4
Addr:   00  01  02  03  04  05  06  07  08  09  10  11  12
        ^    ^    ^    ^    ^    ^
To:     3  2  2  0  0  0  0  0  0  0  0  0  0

Q:      ^    ^
Addr:   13  14  15  16  17  18  19  20  21  22  23  24  25
# Two-Space Collection Example

- 26-byte memory (13 bytes per space), 2 roots
  - Tag 1: one integer
  - Tag 2: one pointer
  - Tag 3: one integer, then one pointer
  - Tag 99: forwarding pointer (to to-space)

<table>
<thead>
<tr>
<th>From:</th>
<th>99 16 2 0 3 2 10 99 13 2 3 1 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addr:</td>
<td>00 01 02 03 04 05 06 07 08 09 10 11 12</td>
</tr>
<tr>
<td></td>
<td>^   ^   ^   ^   ^   ^   ^   ^   ^   ^   ^   ^</td>
</tr>
<tr>
<td>To:</td>
<td>3 2 2 1 75 0 0 0 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Q:</td>
<td>^   ^   ^   ^   ^   ^   ^   ^   ^   ^   ^   ^</td>
</tr>
<tr>
<td>Addr:</td>
<td>13 14 15 16 17 18 19 20 21 22 23 24 25</td>
</tr>
</tbody>
</table>
## Two-Space Collection Example

- 26-byte memory (13 bytes per space), 2 roots
  - Tag 1: one integer
  - Tag 2: one pointer
  - Tag 3: one integer, then one pointer
  - Tag 99: forwarding pointer (to to-space)

<table>
<thead>
<tr>
<th>Root 1: 13</th>
<th>Root 2: 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>From:</td>
<td>99 16 99 18 3 2 10 99 13 2 3 1 4</td>
</tr>
<tr>
<td>Addr:</td>
<td>00 01 02 03 04 05 06 07 08 09 10 11 12</td>
</tr>
<tr>
<td></td>
<td>^ ^ ^ ^ ^</td>
</tr>
<tr>
<td>To:</td>
<td>3 2 18 1 75 2 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Q:</td>
<td>^ ^</td>
</tr>
<tr>
<td>Addr:</td>
<td>13 14 15 16 17 18 19 20 21 22 23 24 25</td>
</tr>
</tbody>
</table>
Two-Space Collection Example

- 26-byte memory (13 bytes per space), 2 roots
  - Tag 1: one integer
  - Tag 2: one pointer
  - Tag 3: one integer, then one pointer
  - Tag 99: forwarding pointer (to to-space)

<table>
<thead>
<tr>
<th>Root 1: 13</th>
<th>Root 2: 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>From:</td>
<td>99 16 99 18 3 2 10 99 13 2 3 1 4</td>
</tr>
<tr>
<td>Addr:</td>
<td>00 01 02 03 04 05 06 07 08 09 10 11 12</td>
</tr>
<tr>
<td></td>
<td>^ ^ ^ ^ ^ ^ ^ ^</td>
</tr>
<tr>
<td>To:</td>
<td>3 2 18 1 75 2 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Q:</td>
<td>^ ^</td>
</tr>
<tr>
<td>Addr:</td>
<td>13 14 15 16 17 18 19 20 21 22 23 24 25</td>
</tr>
</tbody>
</table>
# Two-Space Collection Example

- 26-byte memory (13 bytes per space), 2 roots
  - Tag 1: one integer
  - Tag 2: one pointer
  - Tag 3: one integer, then one pointer
  - Tag 99: forwarding pointer (to to-space)

<table>
<thead>
<tr>
<th>Root 1: 13</th>
<th>Root 2: 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>From:</td>
<td>99 16 99 18 3 2 10 99 13 2 3 1 4</td>
</tr>
<tr>
<td>Addr:</td>
<td>00 01 02 03 04 05 06 07 08 09 10 11 12</td>
</tr>
<tr>
<td></td>
<td>^     ^     ^        ^        ^</td>
</tr>
<tr>
<td>To:</td>
<td>3 2 18 1 75 2 16 0 0 0 0 0 0 0</td>
</tr>
<tr>
<td>Q:</td>
<td>^</td>
</tr>
<tr>
<td>Addr:</td>
<td>13 14 15 16 17 18 19 20 21 22 23 24 25</td>
</tr>
</tbody>
</table>
Two-Space Pros and Cons

- Doesn’t suffer from fragmentation
- Time cost proportional to live data (not garbage!)
- Allocation is simple, just bump a pointer
- Collection doesn’t require much state (handful of pointers, no stack)

- Only half the heap is in use at any time
  - Not a big deal when combined with generational collection
- Still "stop the world"
Further reading

• GC first appeared circa 1958 (original LISP)
• Went mainstream with Java in the 90s
• Tremendous amount of work: new techniques, improvements, etc.
• Still an active research area to this day

Good reference: Uniprocessor Garbage Collection Techniques, by Wilson