Scope, Static, Linked Lists, Arrays

Discussion 02

Example Agenda

- 1:10 - 1:15 ~ announcements
- 1:15 - 1:30 ~ content review
- 1:30 - 1:40 ~ question 1
- 1:40 - 1:55 ~ question 2
- Question 3 if time

Announcements

- Weekly Survey 3 - due this Wednesday 9/11
- Lab 3 - due this Friday 9/13
- Proj 1A - due this Friday 9/13
- Project Party Wednesday 9/11
- Carefully read the OH guidelines if you attend

Content Review

GRoE: Golden Rule of Equals

```
“Given variables y and x:
y = x copies all the bits from x into y.”
```

Java is **pass-by-value**: when you call a function and give it some arguments, the function called receives an exact copy of those arguments, tied to its own local variables.

“Copies all the bits” means different things for **primitive vs. reference types.**

Primitive vs. Reference Types

- **Primitive Types** are represented by a certain number of bytes stored at the location of the variable in memory. There are only 8 in Java.

  Examples: `byte, short, int, long, float, double, boolean, char`

- **Reference Types** are represented by a memory address stored at the location of the variable which points to where the full object is (all objects are stored at addresses in memory). This memory address is often referred to as a *pointer*.

  Examples: Strings, Arrays, Linked Lists, Dogs, etc.

Back to the GRoE

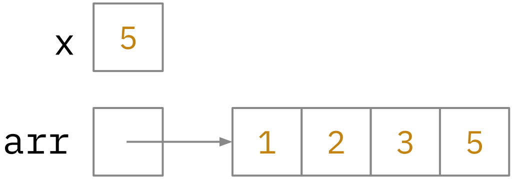
```
“Given variables y and x:
y = x copies all the bits from x into y.”
```

- The value of a primitive type gets copied directly upon variable assignment
  - Ex. `int x = 5;` means that variable x stores the value of 5

- The value of a reference type is a “shallow” copy upon variable assignment: the pointer (memory address) is copied, and the object itself in memory is not
  - Exception: `null` is a special pointer that we compare with `==`

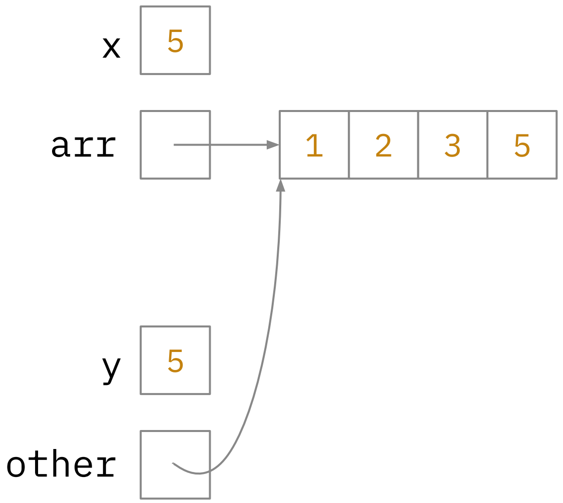
A Quick Example

```
int x = 5;
int[] arr = new int[]{1, 2, 3, 5};
```



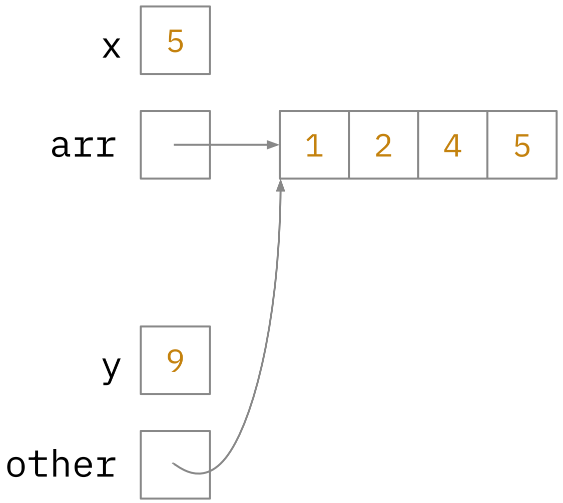
A Quick Example

```
int x = 5;
int[] arr = new int[]{1, 2, 3, 5};
doSomething(x, arr);
...

public void doSomething(int y, int[] other) {
    y = 9;
    other[2] = 4;
}
```



A Quick Example

```
int x = 5;
int[] arr = new int[]{1, 2, 3, 5};
doSomething(x, arr);
...

public void doSomething(int y, int[] other) {
    y = 9;
    other[2] = 4;
}
```



Static vs. Instance, Revisited

Static variables and functions belong to the whole class.
Example: Every 61B Student shares the same professor, and if the professor were to change it would change for everyone.

Instance variables and functions belong to each individual instance.
Example: Each 61B Student has their own ID number, and changing a student's ID number doesn't change anything for any other student.

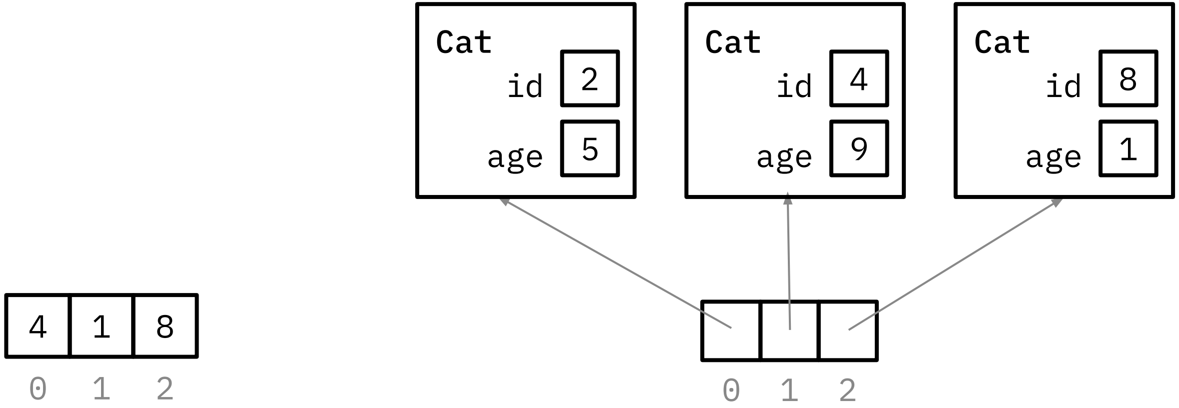
`this` vs. `static`

- `this`
  - Non-static methods can only be called using an instance of that object, so during evaluation of that function, you will always have access to this instance of the object, referred to as `this`
- `static` methods
  - do not require an instance of that object in order to be called, so during evaluation of that function, you cannot rely on access to this instance of the object
- `static` variables
  - shared by all instances of the class; each instance does not get its own copy but can access
- <u>Check for understanding:</u> can you reference `this` in static methods? Can you reference static variables in instance methods? Why or why not?

Arrays

Arrays are data structures that can only hold elements of the same (primitive or reference) type of value.

`arr[i]` holds a value in the ith position of the array (zero-indexed). We can also have n-dimensional arrays (ie. `int[][] a = new int[3][2];` you can index into these like a`[2][1]`)



Arrays have a set length when instantiated, so they cannot be extended / shortened with pointers like a Linked List. To resize, we need to copy over all elements to a new array (ie. `System.arraycopy`)

Linked Lists

Linked Lists are modular lists that are made up of nodes that each contain a value and a pointer to the next node. To access values in a Linked List, you must use dot notation.
Example: `intList.get(2)`

- Can be extended or shortened by changing the pointers of its nodes (unlike arrays)
- Can't be indexed directly into like an array: instead, the computer has to iterate through all of the nodes up to that point and follow their next pointers
- A sentinel is a special type of node that is often used as an empty placeholder for ease of adding / deleting nodes, especially from the front or back of the Linked List
    - In a circular doubly-linked implementation, the sentinel's `next` and `prev` pointers are the first and last nodes respectively

Worksheet

1 Static Electricity

```java
public class Pokemon {
    public String name;
    public int level;
    public static String trainer = "Ash";
    public static int partySize = 0;

    public Pokemon(String name, int level) {
        this.name = name;
        this.level = level;
        this.partySize += 1;
    }

    public static void main(String[] args) {
        Pokemon p = new Pokemon("Pikachu", 17);
        Pokemon j = new Pokemon("Jolteon", 99);
        System.out.println("Party size: " + Pokemon.partySize);
        p.printStats()
        int level = 18;
        Pokemon.change(p, level);
        p.printStats()
        Pokemon.trainer = "Ash";
        j.trainer = "Cynthia";
        p.printStats();
    }
```

```java
public static void change(Pokemon poke, int
level) {
        poke.level = level;
        level = 50;
        poke = new Pokemon("Luxray", 1);
        poke.trainer = "Team Rocket";
    }

    public void printStats() {
        System.out.print(name + " " + level
+ " " + trainer);
    }

}
```

Write what would be printed after the main method is executed.

Java visualizer: https://tinyurl.com/48uk72kc

1 Static Electricity

```java
public class Pokemon {
    public String name;
    public int level;
    public static String trainer = "Ash";
    public static int partySize = 0;

    public Pokemon(String name, int level) {
        this.name = name;
        this.level = level;
        this.partySize += 1;
    }

    public static void main(String[] args) {
        Pokemon p = new Pokemon("Pikachu", 17);
        Pokemon j = new Pokemon("Jolteon", 99);
        System.out.println("Party size: " + Pokemon.partySize);
        p.printStats()
        int level = 18;
        Pokemon.change(p, level);
        p.printStats()
        Pokemon.trainer = "Ash";
        j.trainer = "Cynthia";
        p.printStats();
    }
public static void change(Pokemon poke, int level) {
        poke.level = level;
        level = 50;
        poke = new Pokemon("Luxray", 1);
        poke.trainer = "Team Rocket";
    }

    public void printStats() {
        System.out.print(name + " " + level + " " + trainer);
    }

}
```

```
Party size: 2
Pikachu 17 Ash
Pikachu 18 Team Rocket
Pikachu 18 Cynthia
```

Java visualizer: https://tinyurl.com/48uk72kc

1 Static Electricity

```java
1   public class Pokemon {
2       public String name;
3       public int level;
4       public static String trainer = "Ash";
5       public static int partySize = 0;
6
7       public Pokemon(String name, int level) {
8           this.name = name;
9           this.level = level;
10          this.partySize += 1;
11      }
12
13      public static void main(String[] args) {
14          Pokemon p = new Pokemon("Pikachu", 17);
15          Pokemon j = new Pokemon("Jolteon", 99);
16          System.out.println("Party size: " + Pokemon.partySize);
17          p.printStats()
18          int level = 18;
19          Pokemon.change(p, level);
20          p.printStats()
21          Pokemon.trainer = "Ash";
22          j.trainer = "Cynthia";
23          p.printStats();
24      }
```

```java
26  public static void change(Pokemon poke, int
    level) {
27          poke.level = level;
28          level = 50;
29          poke = new Pokemon("Luxray", 1);
30          poke.trainer = "Team Rocket";
31      }
32
33      public void printStats() {
34          System.out.print(name + " " + level
    + " " + trainer);
35      }
36
37  }
```

On line 28, is level:

- An instance variable of the Pokemon object?
- The local variable containing the parameter to the change method?
- The local variable in the main method?
- Something else?

1 Static Electricity

```
1   public class Pokemon {
2       public String name;
3       public int level;
4       public static String trainer = "Ash";
5       public static int partySize = 0;
6
7       public Pokemon(String name, int level) {
8           this.name = name;
9           this.level = level;
10          this.partySize += 1;
11      }
12
13      public static void main(String[] args) {
14          Pokemon p = new Pokemon("Pikachu", 17);
15          Pokemon j = new Pokemon("Jolteon", 99);
16          System.out.println("Party size: " + Pokemon.partySize);
17          p.printStats()
18          int level = 18;
19          Pokemon.change(p, level);
20          p.printStats()
21          Pokemon.trainer = "Ash";
22          j.trainer = "Cynthia";
23          p.printStats();
24      }
```

```
26  public static void change(Pokemon poke, int
    level) {
27          poke.level = level;
28          level = 50;
29          poke = new Pokemon("Luxray", 1);
30          poke.trainer = "Team Rocket";
31      }
32
33      public void printStats() {
34          System.out.print(name + " " + level
    + " " + trainer);
35      }
36
37  }
```

On line 28, is level:

- An instance variable of the Pokemon object
- The local variable containing the parameter to the change method
- The local variable in the main method
- Something else?

1 Static Electricity

```java
public class Pokemon {
    public String name;
    public int level;
    public static String trainer = "Ash";
    public static int partySize = 0;

    public Pokemon(String name, int level) {
        this.name = name;
        this.level = level;
        this.partySize += 1;
    }

    public static void main(String[] args) {
        Pokemon p = new Pokemon("Pikachu", 17);
        Pokemon j = new Pokemon("Jolteon", 99);
        System.out.println("Party size: " + Pokemon.partySize);
        p.printStats()
        int level = 18;
        Pokemon.change(p, level);
        p.printStats()
        Pokemon.trainer = "Ash";
        j.trainer = "Cynthia";
        p.printStats();
    }
public static void change(Pokemon poke, int level) {
        poke.level = level;
        level = 50;
        poke = new Pokemon("Luxray", 1);
        poke.trainer = "Team Rocket";
    }

    public void printStats() {
        System.out.print(name + " " + level + " " + trainer);
    }

}
```

If we were to call `Pokemon.printStats()` at the end of our main method, what would happen?

1 Static Electricity

```
public class Pokemon {
    public String name;
    public int level;
    public static String trainer = "Ash";
    public static int partySize = 0;

    public Pokemon(String name, int level) {
        this.name = name;
        this.level = level;
        this.partySize += 1;
    }

    public static void main(String[] args) {
        Pokemon p = new Pokemon("Pikachu", 17);
        Pokemon j = new Pokemon("Jolteon", 99);
        System.out.println("Party size: " + Pokemon.partySize);
        p.printStats()
        int level = 18;
        Pokemon.change(p, level);
        p.printStats()
        Pokemon.trainer = "Ash";
        j.trainer = "Cynthia";
        p.printStats();
    }
```

```
public static void change(Pokemon poke, int
level) {
        poke.level = level;
        level = 50;
        poke = new Pokemon("Luxray", 1);
        poke.trainer = "Team Rocket";
    }

    public void printStats() {
        System.out.print(name + " " + level
+ " " + trainer);
    }

}
```

Error!

- printStats() is an instance method
- Only static methods can be called using the name of the class (ie. Pokemon)
- static methods can only modify static variables, but instance methods can modify both

2 Rotate *Extra*

Implement `rotate` such that it returns a new array containing the elements in A have shifted k positions to the right, without modifying A. Note: the modulo of a negative number is still negative

```
public static int[] rotate(int[] A, int k) {
     int rightShift = _______________________________;
     if (_________________________) {
          _____________________________________________;
     }

     int[] newArr = ____________________________________;
     for (__________________________________________) {
          int newIndex = ________________________________;
          _____________________________________________;
     }
     return newArr;
}
```

2 Rotate *Extra*

Implement `rotate` such that it returns a new array containing the elements in A have shifted k positions to the right, without modifying A. Note: the modulo of a negative number is still negative

```
public static int[] rotate(int[] A, int k) {
     int rightShift = k % A.length;
     if (_________________________) {
          _____________________________________________;
     }

     int[] newArr = ____________________________________;
     for (__________________________________________) {
          int newIndex = ________________________________;
          _____________________________________________;
     }
     return newArr;
}
```

2 Rotate *Extra*

Implement `rotate` such that it returns a new array containing the elements in A have shifted k positions to the right, without modifying A. Note: the modulo of a negative number is still negative

```
public static int[] rotate(int[] A, int k) {
     int rightShift = k % A.length;
     if (rightShift < 0) {
          rightShift += A.length;;
     }

     int[] newArr = ____________________________________;
     for (__________________________________________) {
          int newIndex = ________________________________;
          _____________________________________________;
     }
     return newArr;
}
```

2 Rotate *Extra*

Implement `rotate` such that it returns a new array containing the elements in A have shifted k positions to the right, without modifying A. Note: the modulo of a negative number is still negative

```
public static int[] rotate(int[] A, int k) {
     int rightShift = k % A.length;
     if (rightShift < 0) {
          rightShift += A.length;
     }

     int[] newArr = new int[A.length];
     for (__________________________________________) {
          int newIndex = ________________________________;
          _____________________________________________;
     }
     return newArr;
}
```

2 Rotate *Extra*

Implement `rotate` such that it returns a new array containing the elements in A have shifted k positions to the right, without modifying A. Note: the modulo of a negative number is still negative

```
public static int[] rotate(int[] A, int k) {
     int rightShift = k % A.length;
     if (rightShift < 0) {
          rightShift += A.length;
     }

     int[] newArr = new int[A.length];
     for (int i = 0; i < A.length; i++) {
          int newIndex = ________________________________;
          _____________________________________________;
     }
     return newArr;
}
```

2 Rotate *Extra*

Implement `rotate` such that it returns a new array containing the elements in A have shifted k positions to the right, without modifying A. Note: the modulo of a negative number is still negative

```
public static int[] rotate(int[] A, int k) {
     int rightShift = k % A.length;
     if (rightShift < 0) {
          rightShift += A.length;
     }

     int[] newArr = new int[A.length];
     for (int i = 0; i < A.length; i++) {
          int newIndex = (i + rightShift) % A.length;
          _____________________________________________;
     }
     return newArr;
}
```

2 Rotate *Extra*

Implement `rotate` such that it returns a new array containing the elements in A have shifted k positions to the right, without modifying A. Note: the modulo of a negative number is still negative

```
public static int[] rotate(int[] A, int k) {
     int rightShift = k % A.length;
     if (rightShift < 0) {
          rightShift += A.length;
     }

     int[] newArr = new int[A.length];
     for (int i = 0; i < A.length; i++) {
          int newIndex = (i + rightShift) % A.length;
          newArr[newIndex] = A[i];
     }
     return newArr;
}
```

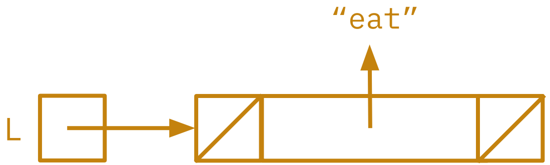
3 Cardinal Directions

Draw out the resulting diagram after executing all the lines.

```
DLLStringNode L = new DLLStringNode(null, "eat", null);
L = new DLLStringNode(null, "bananas", L);
L = new DLLStringNode(null, "never", L);
L = new DLLStringNode(null, "sometimes", L);
DLLStringNode M = L.next;
DLLStringNode R = new DLLStringNode(null, "shredded", null);
R = new DLLStringNode(null, "wheat", R);
R.next.next = R;
M.next.next.next = R.next;
L.next.next = L.next.next.next;
L = M.next;
M.next.next.prev = R;
L.prev = M;
L.next.prev = L;
R.prev = L.next.next;
```

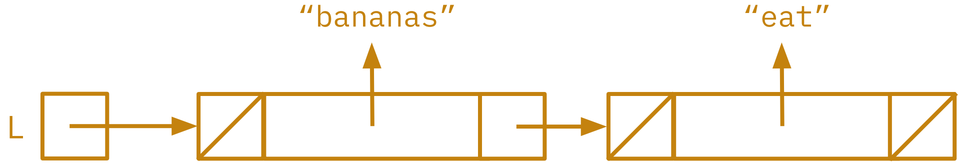
3 Cardinal Directions

```
DLLStringNode L = new DLLStringNode(null, "eat", null);
```



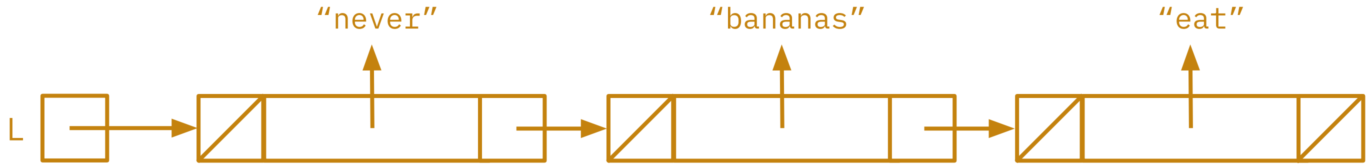
3 Cardinal Directions

```
L = new DLLStringNode(null, "bananas", L);
```



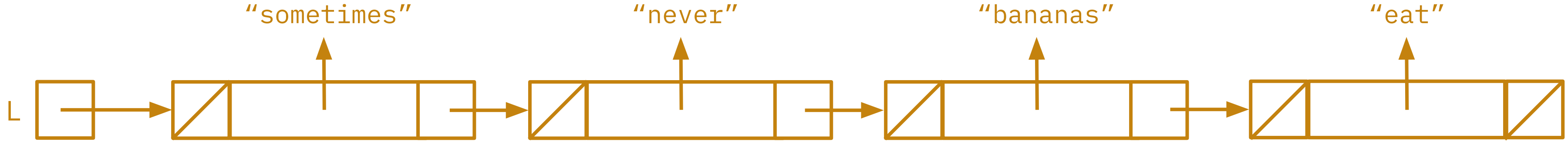
3 Cardinal Directions

```
L = new DLLStringNode(null, "never", L);
```



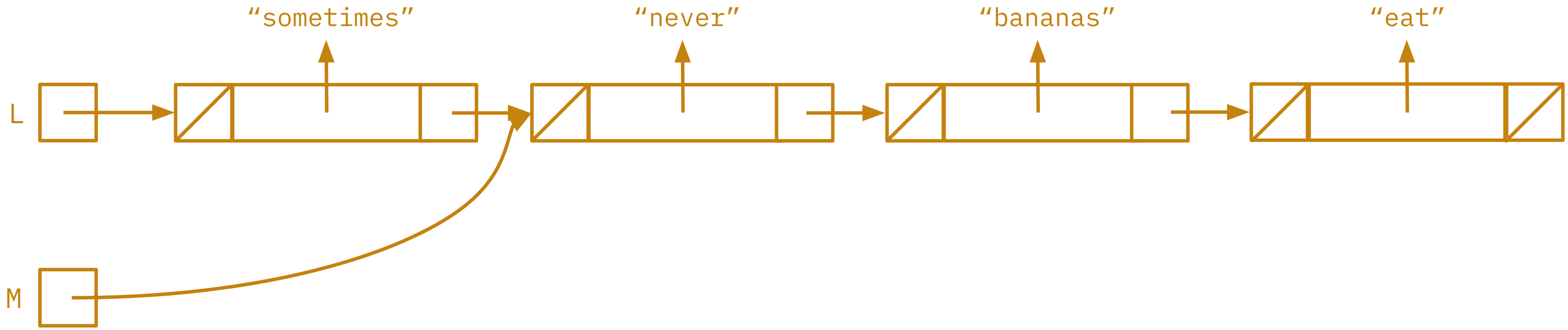
3 Cardinal Directions

```
L = new DLLStringNode(null, "sometimes", L);
```



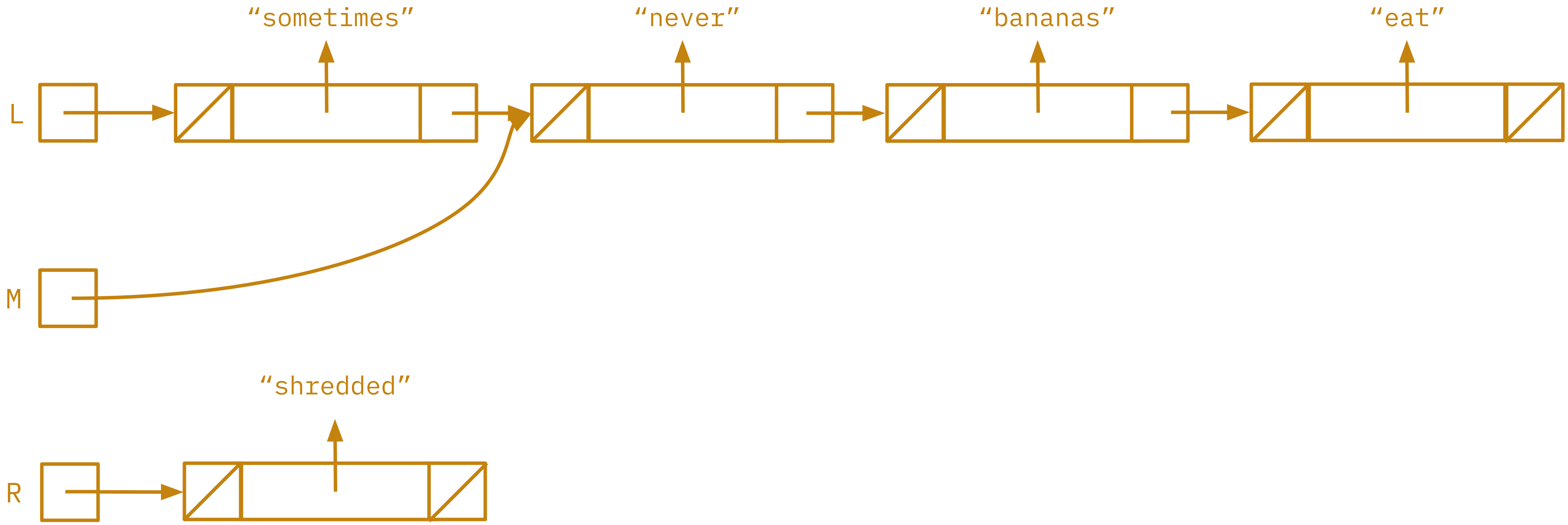
3 Cardinal Directions

```
DLLStringNode M = L.next;
```



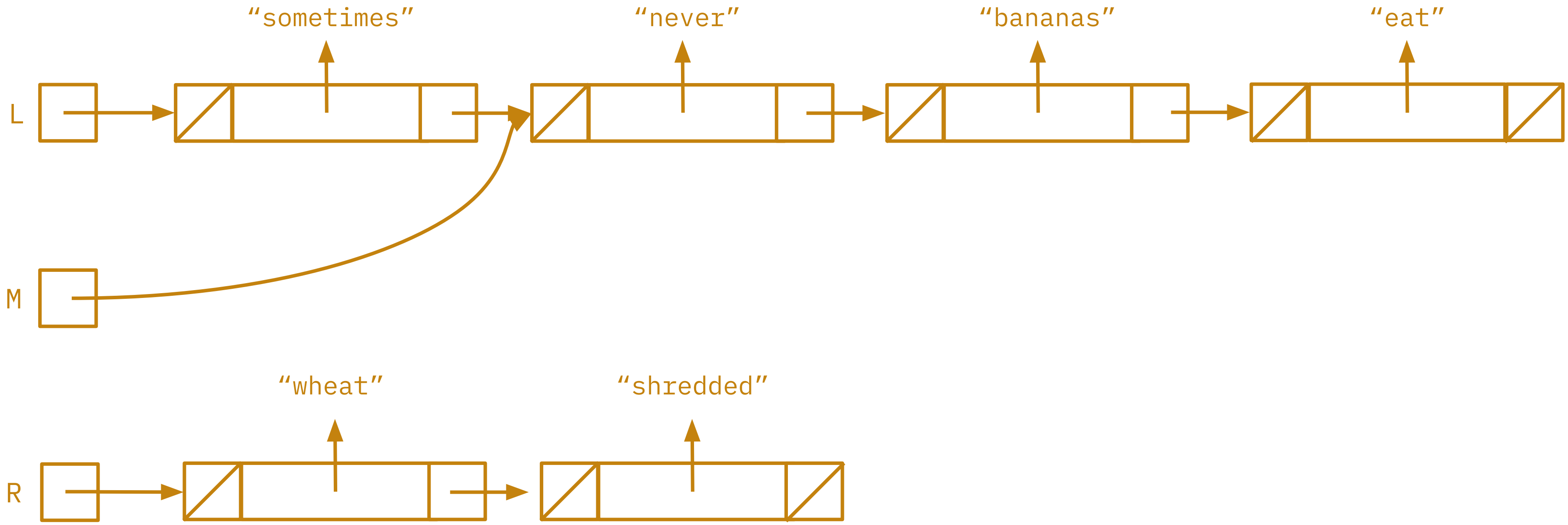
3 Cardinal Directions

```
DLLStringNode R = new DLLStringNode(null, "shredded", null);
```



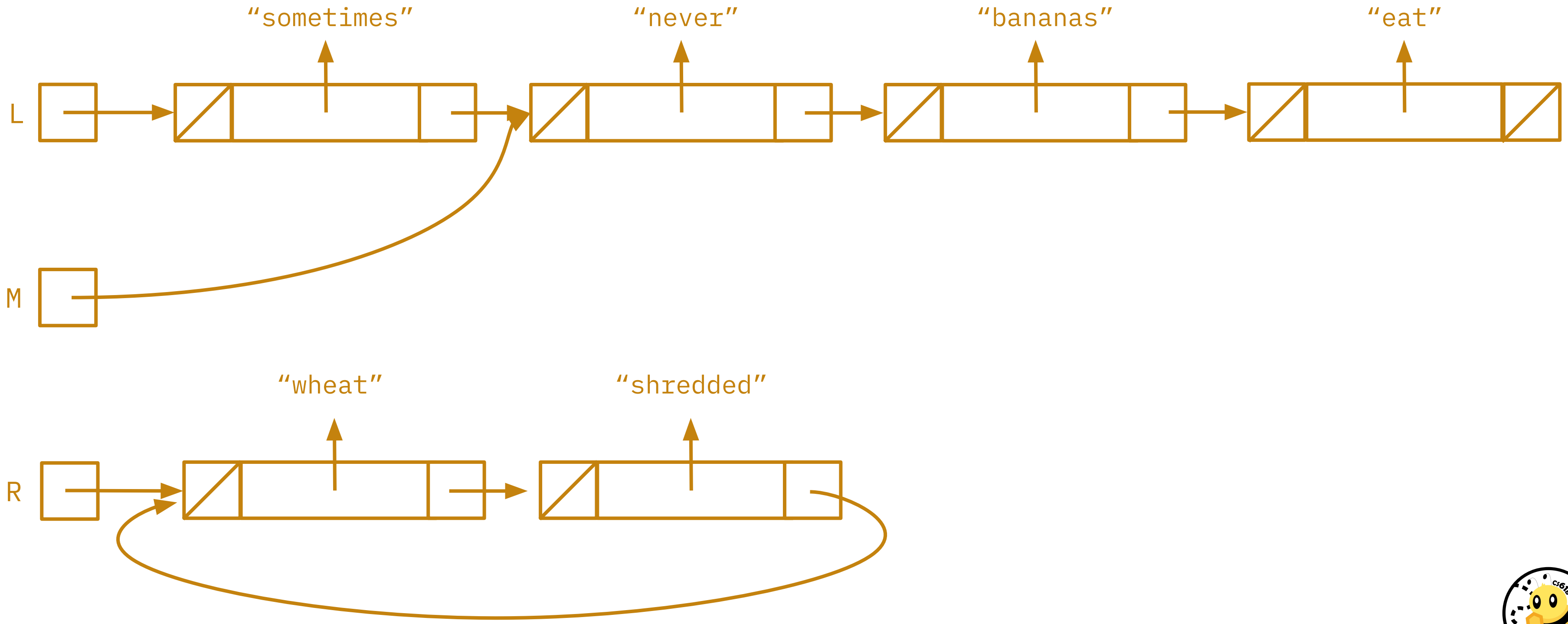
3 Cardinal Directions

```
R = new DLLStringNode(null, "wheat", R);
```



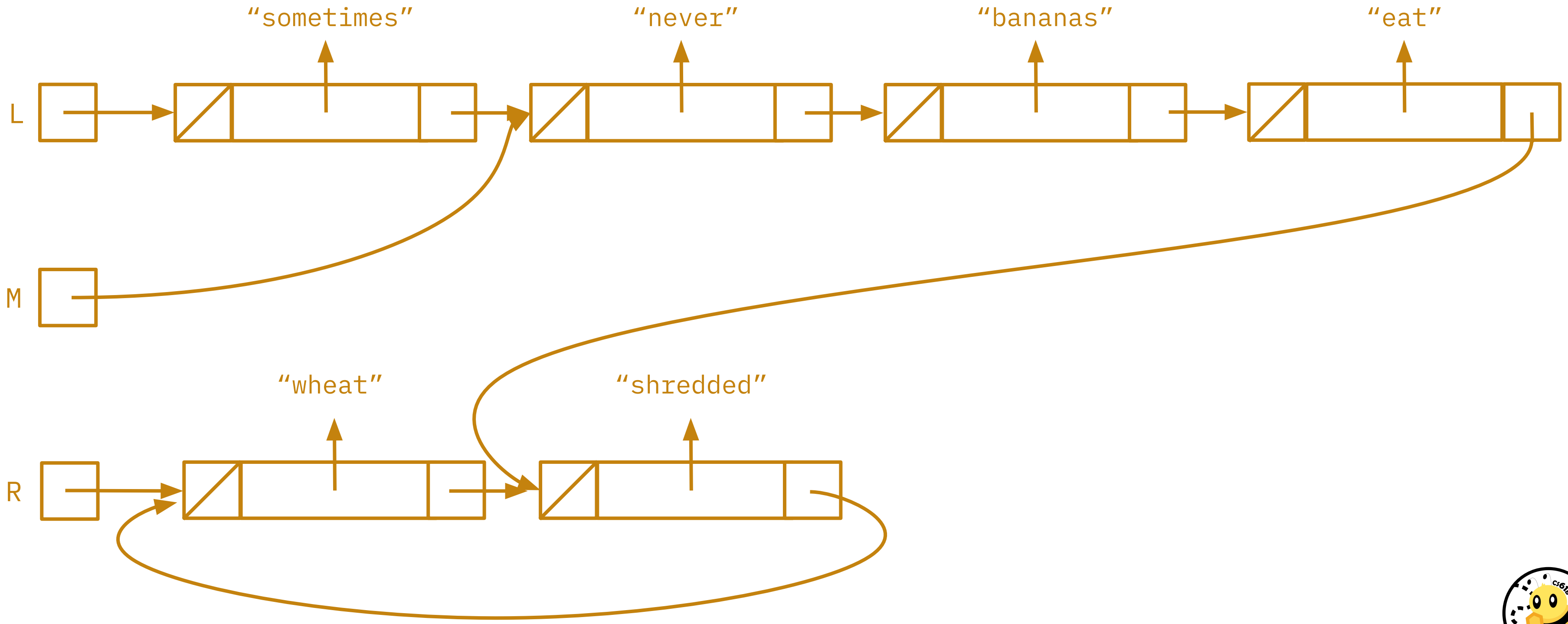
3 Cardinal Directions

```
R.next.next = R;
```



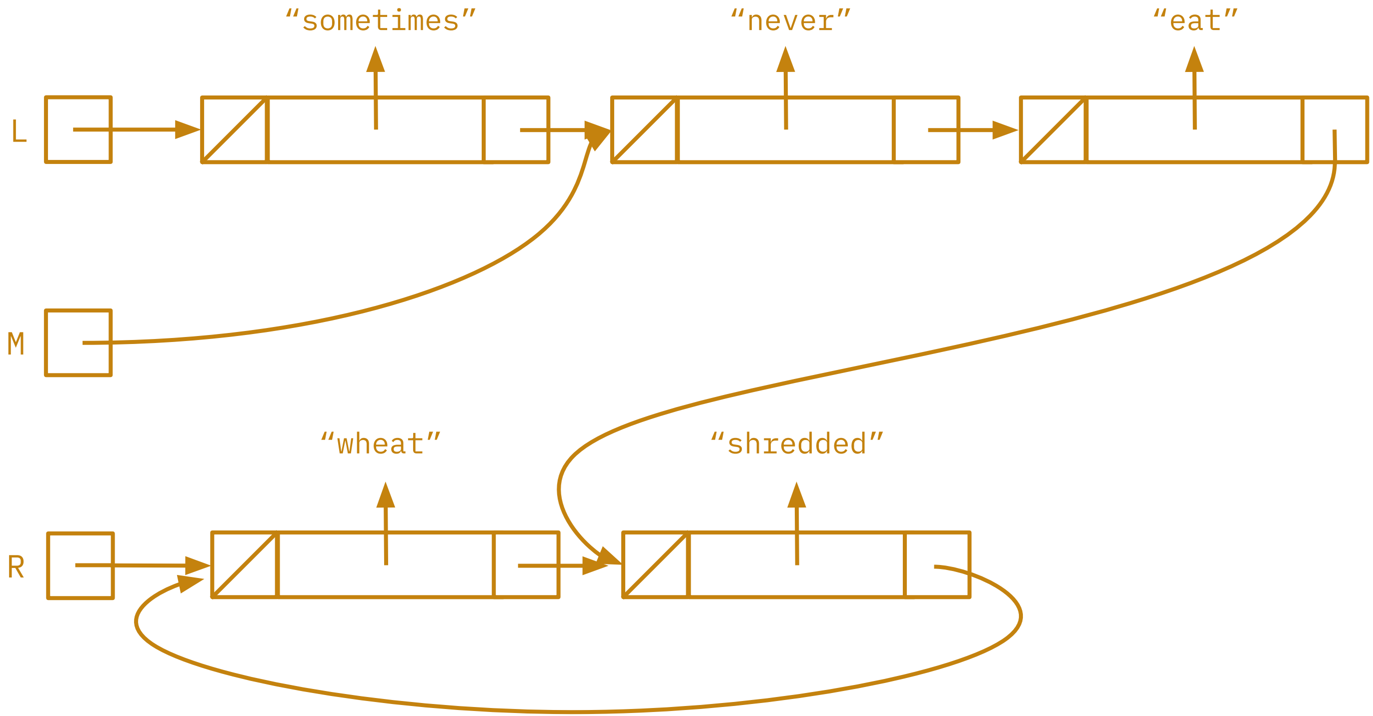
3 Cardinal Directions

```
M.next.next.next = R.next;
```

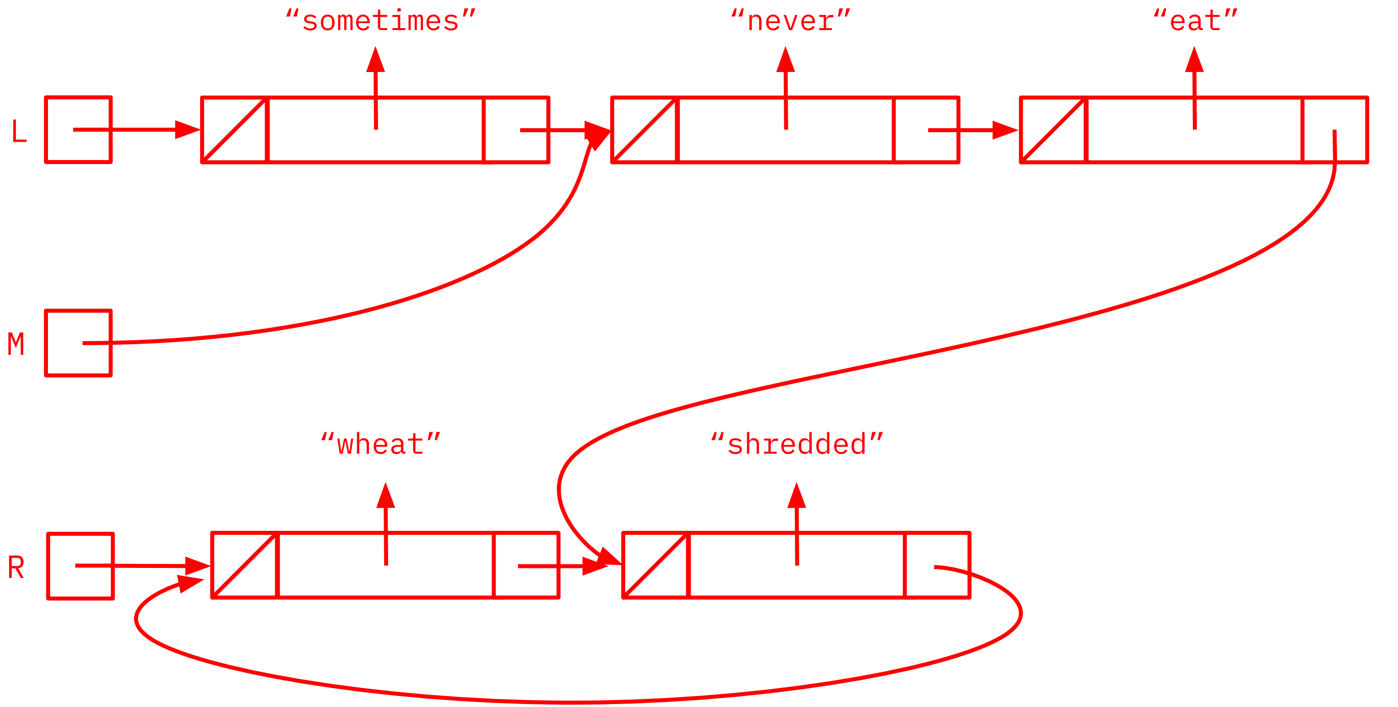


3 Cardinal Directions

```
L.next.next = L.next.next.next;
```

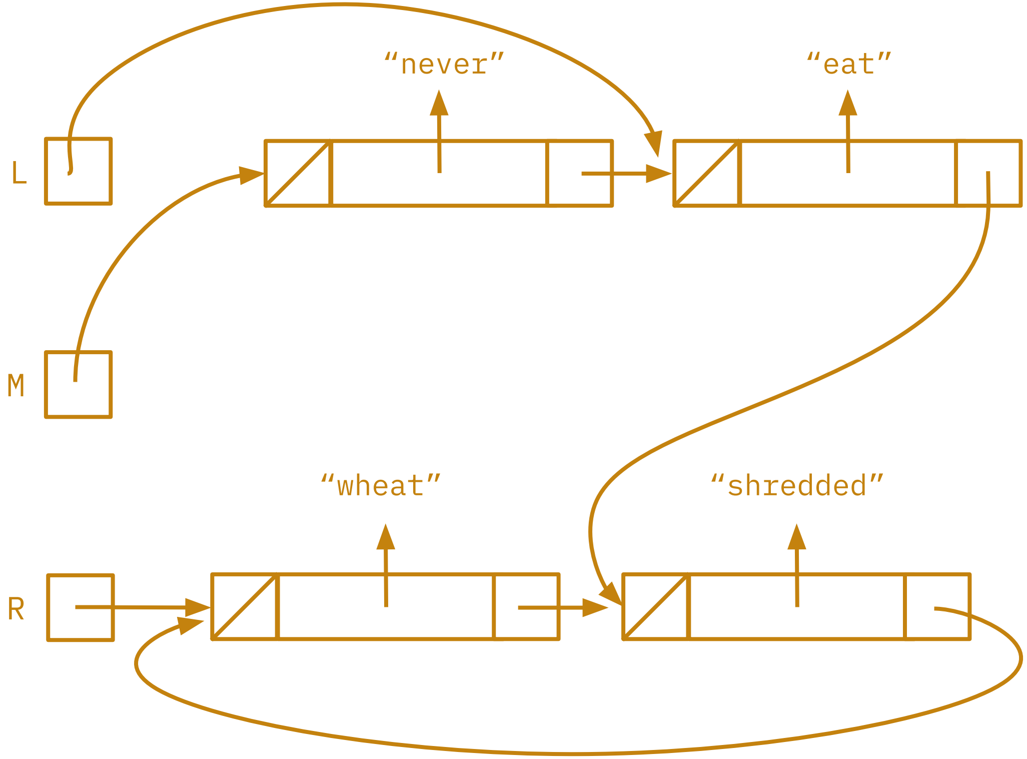


3 Cardinal Directions



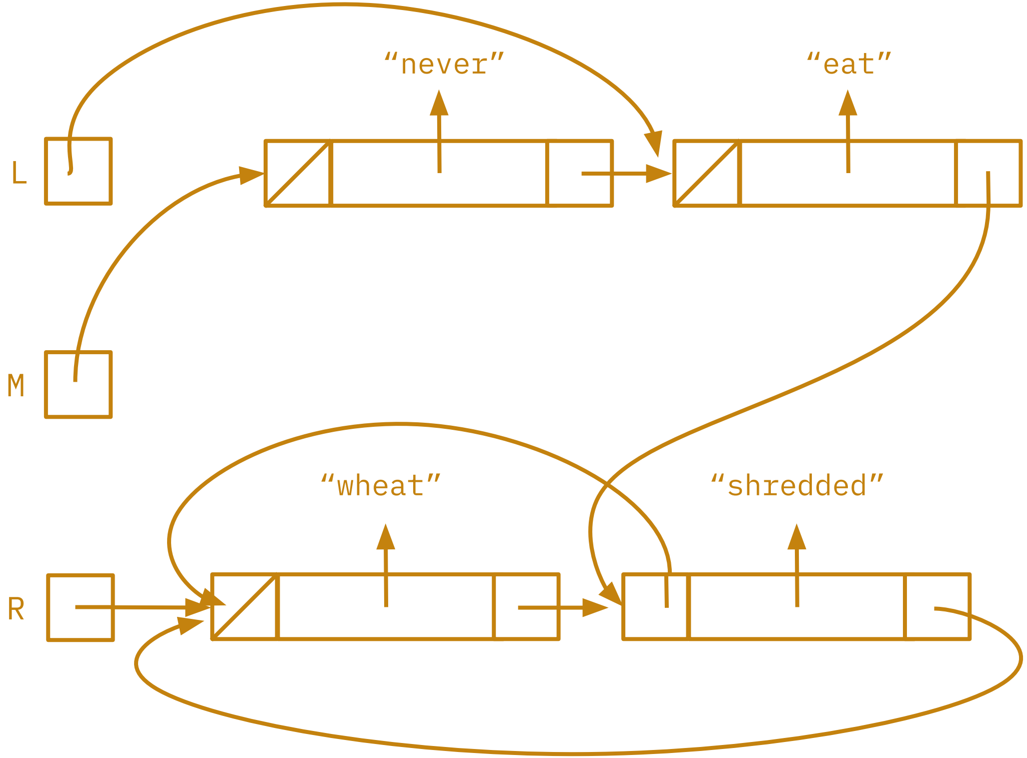
3 Cardinal Directions

```
L = M.next;
```



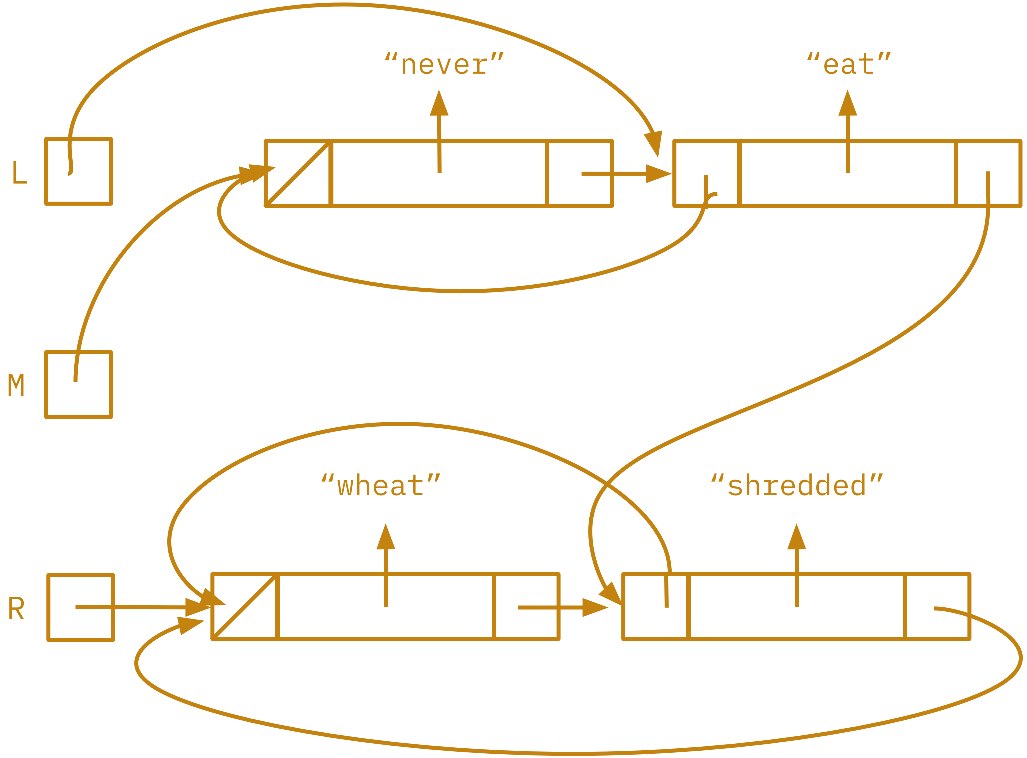
3 Cardinal Directions

```
M.next.next.prev = R;
```



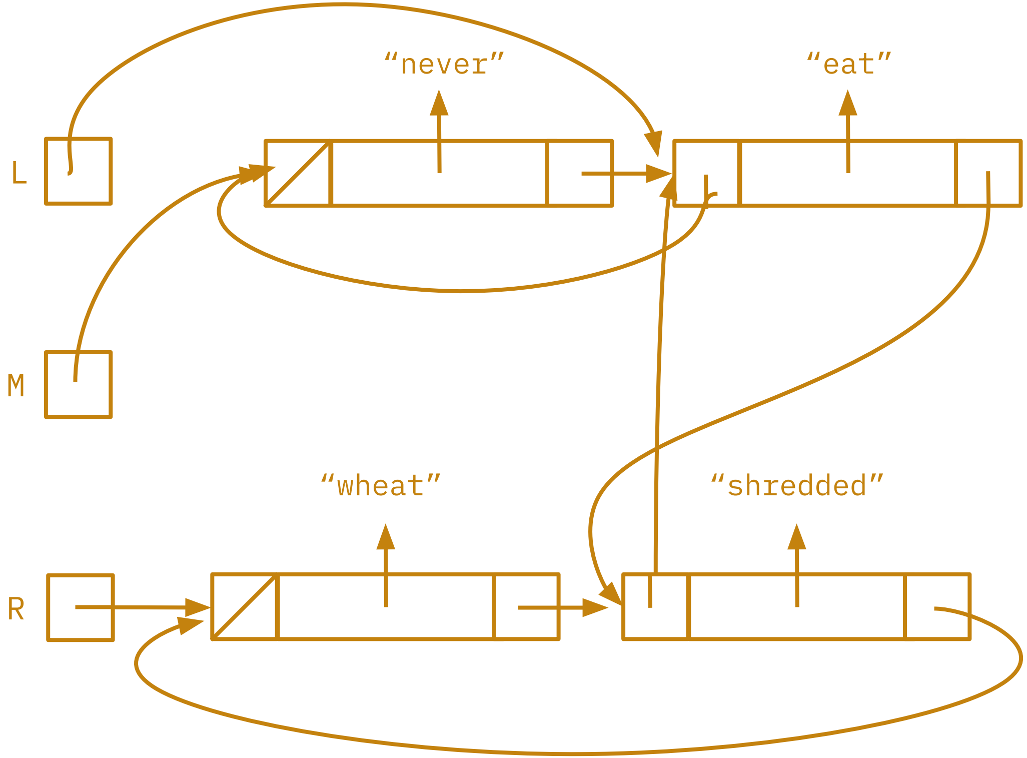
3 Cardinal Directions

```
L.prev = M;
```



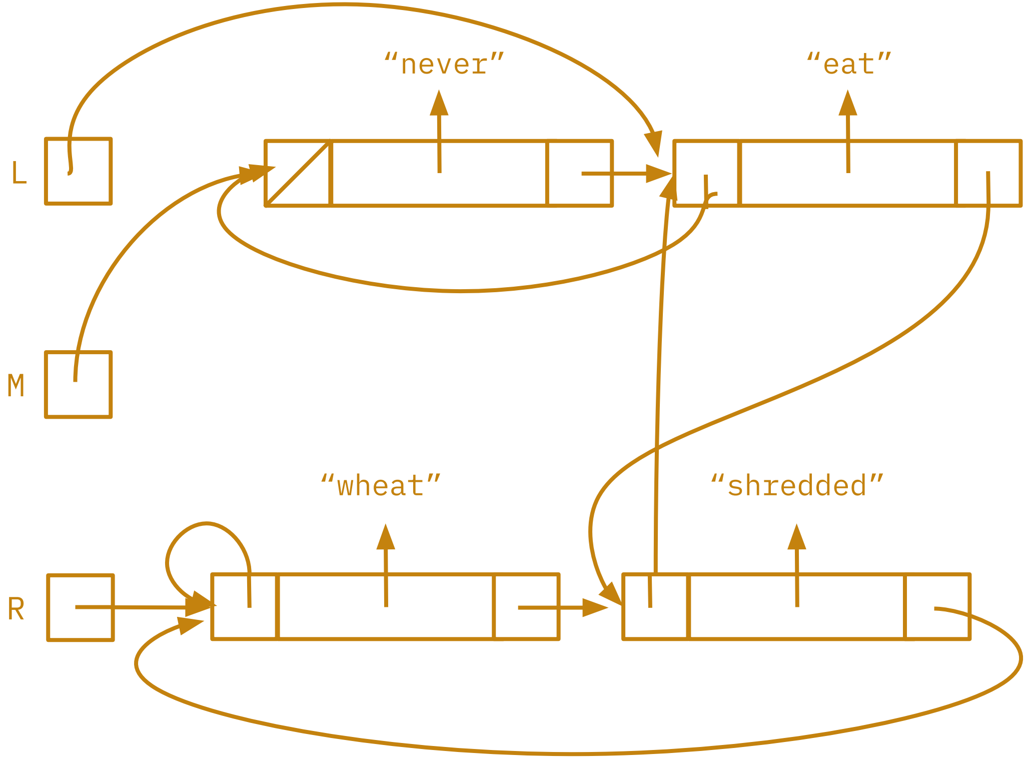
3 Cardinal Directions

```
L.next.prev = L;
```

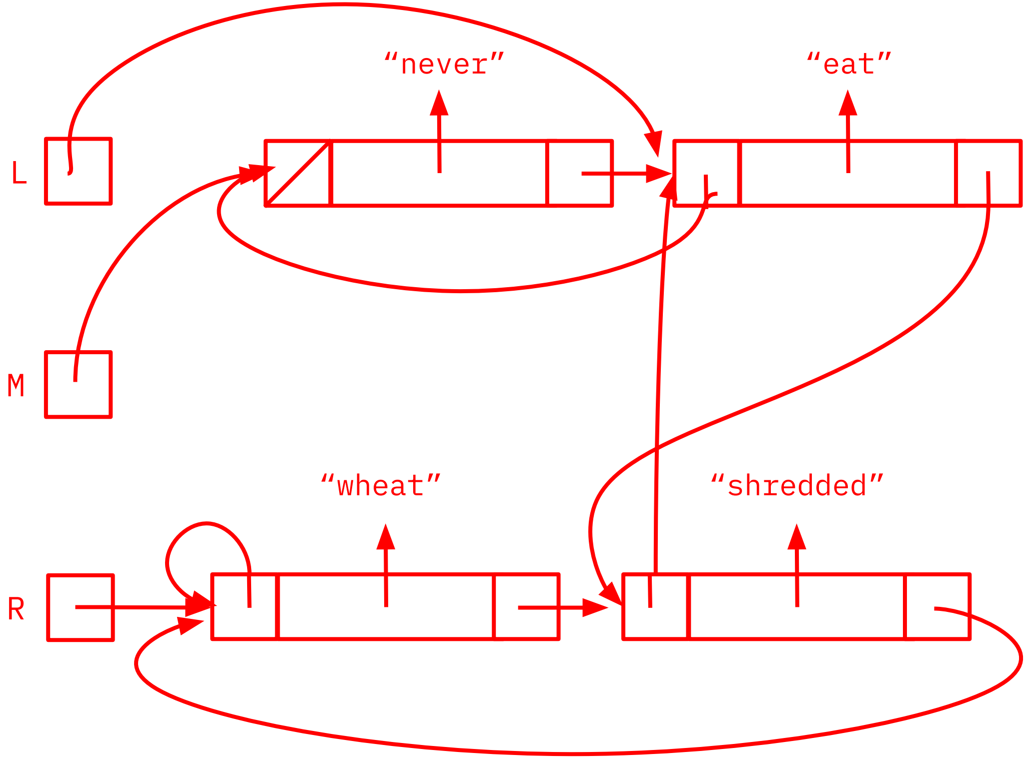


3 Cardinal Directions

```
R.prev = L.next.next;
```



3 Cardinal Directions



4 Gridify

```
public class SLList {
   Node sentinel;

   public SLList() {
      this.sentinel = new Node();
   }

   private static class Node {
      int item;
      Node next;
   }

   public int[][] gridify(int rows, int cols) {
      int[][] grid = ____________________________;
      ___________________________________________;
      return grid;
    }

```

```
private void gridifyHelper(int[][] grid, Node curr,
int numFilled) {
      if (_________________________) {
         return;
      }

      int row = ______________________;
      int col = ______________________;

      _________________ = _______________;
      ___________________________________;

    }


```

4 Gridify

```
public class SLList {
   Node sentinel;

   public SLList() {
      this.sentinel = new Node();
   }

   private static class Node {
      int item;
      Node next;
   }

   public int[][] gridify(int rows, int cols) {
      int[][] grid = new int[rows][cols];
      ___________________________________________;
      return grid;
    }

   private void gridifyHelper(int[][] grid, Node curr,
   int numFilled) {
         if (_________________________) {
            return;
         }

         int row = ______________________;
         int col = ______________________;

         _________________ = _______________;
         __________________________________;

       }

```

4 Gridify

```
public class SLList {
   Node sentinel;

   public SLList() {
      this.sentinel = new Node();
   }

   private static class Node {
      int item;
      Node next;
   }

   public int[][] gridify(int rows, int cols) {
      int[][] grid = new int[rows][cols];
      gridifyHelper(grid, sentinel.next, 0);
      return grid;
    }

```

```
private void gridifyHelper(int[][] grid, Node curr,
int numFilled) {
      if (_________________________) {
         return;
      }

      int row = ______________________;
      int col = ______________________;

      grid[row][col] = _______________;
      ___________________________________;

    }

```

4 Gridify

```
public class SLList {
   Node sentinel;

   public SLList() {
      this.sentinel = new Node();
   }

   private static class Node {
      int item;
      Node next;
   }

   public int[][] gridify(int rows, int cols) {
      int[][] grid = new int[rows][cols];
      gridifyHelper(grid, sentinel.next, 0);
      return grid;
    }

```

```
private void gridifyHelper(int[][] grid, Node curr,
int numFilled) {
      if (curr == sentinel || numFilled >= grid.length
* grid[0].length) {
         return;
      }

      int row = ______________________;
      int col = ______________________;

      grid[row][col] = _______________;
      ___________________________________;

    }

```

4 Gridify

```
public class SLList {
   Node sentinel;

   public SLList() {
      this.sentinel = new Node();
   }

   private static class Node {
      int item;
      Node next;
   }

   public int[][] gridify(int rows, int cols) {
      int[][] grid = new int[rows][cols];
      gridifyHelper(grid, sentinel.next, 0);
      return grid;
    }

```

```
private void gridifyHelper(int[][] grid, Node curr,
int numFilled) {
      if (curr == sentinel || numFilled >= grid.length
* grid[0].length) {
         return;
      }

      int row = numFilled / grid[0].length;
      int col = ______________________;

      grid[row][col] = _______________;
      ___________________________________;

    }


```

4 Gridify

```
public class SLList {
   Node sentinel;

   public SLList() {
      this.sentinel = new Node();
   }

   private static class Node {
      int item;
      Node next;
   }

   public int[][] gridify(int rows, int cols) {
      int[][] grid = new int[rows][cols];
      gridifyHelper(grid, sentinel.next, 0);
      return grid;
    }

```

```
private void gridifyHelper(int[][] grid, Node curr, int numFilled) {
      if (curr == sentinel || numFilled >= grid.length * grid[0].length) {
         return;
      }

      int row = numFilled / grid[0].length;
      int col = numFilled % grid[0].length;

      grid[row][col] = _______________;
      ___________________________________;

    }


```

4 Gridify

```
public class SLList {
   Node sentinel;

   public SLList() {
      this.sentinel = new Node();
   }

   private static class Node {
      int item;
      Node next;
   }

   public int[][] gridify(int rows, int cols) {
      int[][] grid = new int[rows][cols];
      gridifyHelper(grid, sentinel.next, 0);
      return grid;
    }

```

```
private void gridifyHelper(int[][] grid, Node curr,
int numFilled) {
      if (curr == sentinel || numFilled >= grid.length
* grid[0].length) {
         return;
      }

      int row = numFilled / grid[0].length;
      int col = numFilled % grid[0].length;

      grid[row][col] = curr.item;
      ___________________________________;

    }


```

4 Gridify

```
public class SLList {
   Node sentinel;

   public SLList() {
      this.sentinel = new Node();
   }

   private static class Node {
      int item;
      Node next;
   }

   public int[][] gridify(int rows, int cols) {
      int[][] grid = new int[rows][cols];
      gridifyHelper(grid, sentinel.next, 0);
      return grid;
    }

   private void gridifyHelper(int[][] grid, Node curr, int numFilled) {
         if (curr == sentinel || numFilled >= grid.length * grid[0].length) {
            return;
         }

         int row = numFilled / grid[0].length;
         int col = numFilled % grid[0].length;

         grid[row][col] = curr.item;
         gridifyHelper(grid, curr.next, numFilled + 1);

       }


```

4 Gridify

```java
public class SLList {
   Node sentinel;

   public SLList() {
      this.sentinel = new Node();
   }

   private static class Node {
      int item;
      Node next;
   }

   public int[][] gridify(int rows, int cols) {
      int[][] grid = new int[rows][cols];
      gridifyHelper(grid, sentinel.next, 0);
      return grid;
    }

   private void gridifyHelper(int[][] grid, Node curr, int numFilled) {
         if (curr == sentinel || numFilled >= grid.length * grid[0].length) {
            return;
         }

         int row = numFilled / grid[0].length;
         int col = numFilled % grid[0].length;

         grid[row][col] = curr.item;
         gridifyHelper(grid, curr.next, numFilled + 1);

       }


```

Why helper method here? Why can't we just have the signature for `gridify` also have a pointer to the curr node, such that the user of the function passes in the sentinel each time?

4 Gridify

```
public class SLList {
   Node sentinel;

   public SLList() {
      this.sentinel = new Node();
   }

   private static class Node {
      int item;
      Node next;
   }

   public int[][] gridify(int rows, int cols) {
      int[][] grid = new int[rows][cols];
      gridifyHelper(grid, sentinel.next, 0);
      return grid;
    }

```

```
private void gridifyHelper(int[][] grid, Node curr,
int numFilled) {
      if (curr == sentinel || numFilled >= grid.length
* grid[0].length) {
         return;
      }

      int row = numFilled / grid[0].length;
      int col = numFilled % grid[0].length;

      grid[row][col] = curr.item;
      gridifyHelper(grid, curr.next, numFilled + 1);

    }


```

Why helper method here? Why can't we just have the signature for `gridify` also have a pointer to the curr node, such that the user of the function passes in the sentinel each time?

We need a helper to keep track of which node and index we're on.
If we make the change: it breaks the abstraction barrier - requires our user to understand sentinels.
If they pass in random values - incorrect answer.