CS 600.226: Data Structures

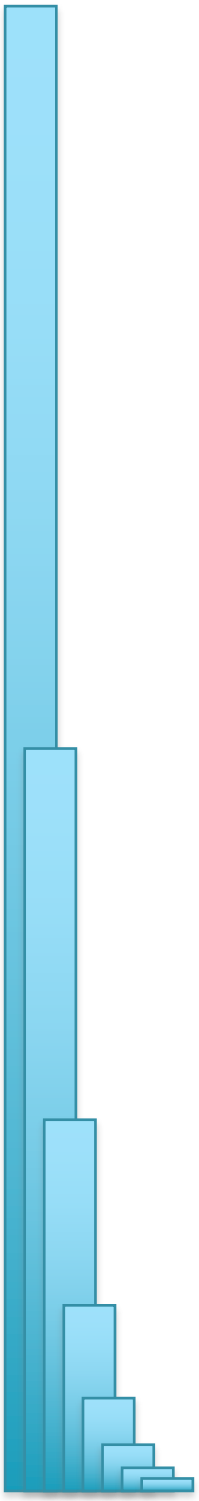
Michael Schatz

Sept 10 2018

Lecture 5: Iterators

Agenda

1. *Review HW1*
2. *References and Linked Lists*
3. *Nested Classes and Iterators*



Assignment 1: Due Friday Sept 14 @ 10pm

https://github.com/schatzlab/datastructures2018/blob/master/assignments/assignment01/assignment01.md

Assignment 1: Warming Up

- **Out on:** September 7, 2016
- **Due by:** September 14, 2016 before 10:00 pm
- **Collaboration:** None
- **Grading:**
  - Functionality 65%
  - ADT Solution 30%
  - Solution Design and READMDE 5%
  - Style 0%

Overview

The first assignment is mostly a warmup exercise to refresh your knowledge of Java and an ADT problem to start you thinking more abstractly about your data.

Assignment 1: Due Friday Sept 14 @ 10pm

https://github.com/schatzlab/datastructures2018/blob/master/assignments/assignment01/assignment01.md

Problem 2: Counter Varieties (30%)

Your second task is to write a number of "counters" that can be used interchangably (at least as far as Java is concerned). You are given the following interface (put it into a file `Counter.java` please):

```
/** The essence of any counter. */
public interface Counter {
  /** Current value of this counter. */
  int value();
  /** Increment this counter. */
  void up();
  /** Decrement this counter. */
  void down();
}
```

Develop the following:

- An interface `ResetableCounter` that supports the method `void reset()` in addition to those of `Counter`; this method should set the counter to its initial value
- An implementation of `ResetableCounter` called `BasicCounter` that starts at the value 0 and counts up and down by +1 and -1 respectively.
- An implementation of `ResetableCounter` called `EvenCounter` that starts at the value 0, counts up by adding 2, and counts down by substacting 2
- An implementation of `ResetableCounter` called `TenCounter` that starts at the value 1, counts up by multipling by 10, and counts down by dividing by 10. This should round *up* to the nearest integer if needed
- An implementation of `ResetableCounter` called `FlexibleCounter` that allows clients to specify a start value as well as an additive increment (used for counting up) when a counter is created. For example `new FlexibleCounter(-10, 42)` would yield a counter with the current value -10; after a call to `up()` its value would be 32.

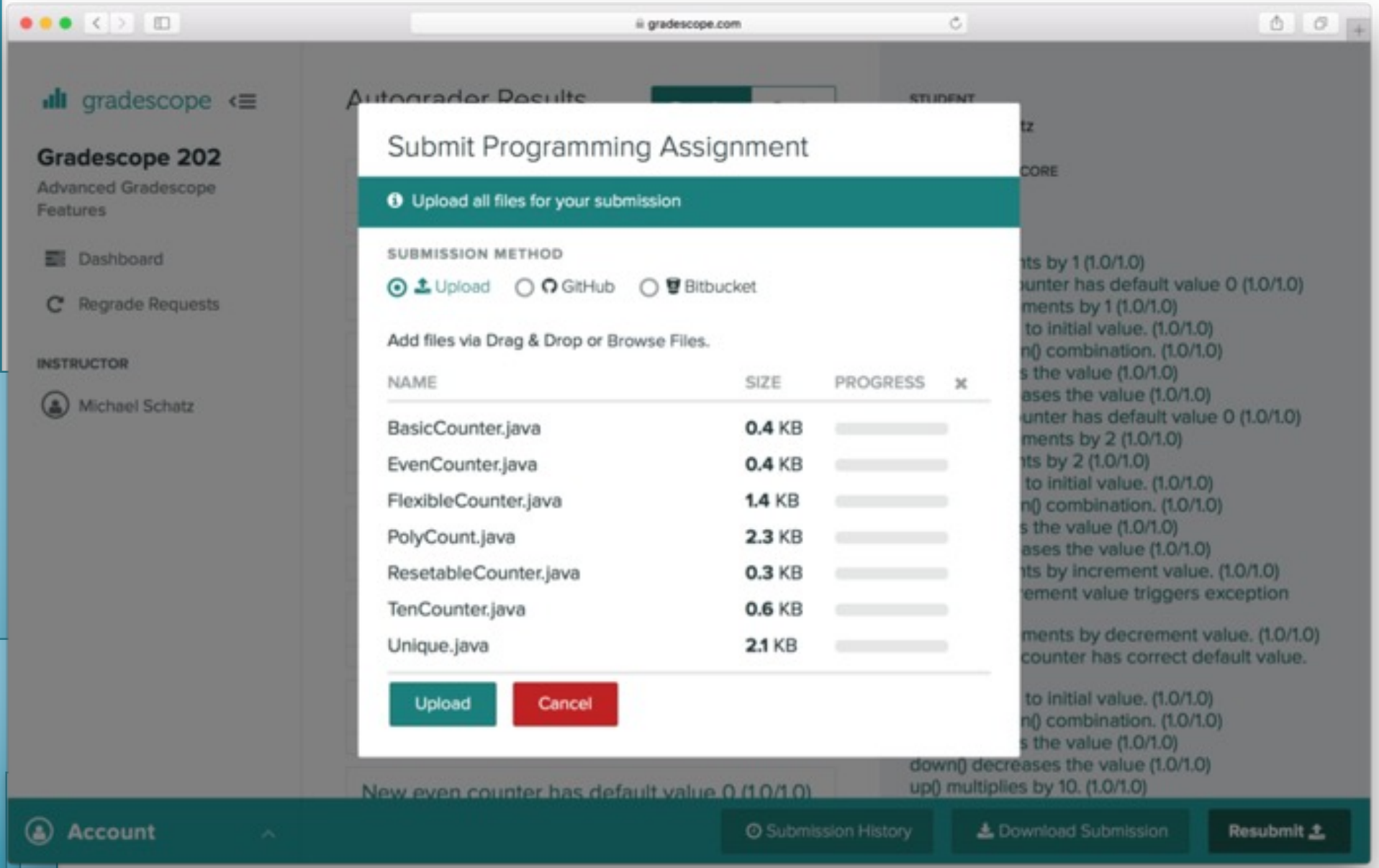
All of your implementations should be resetable, and each should contain a main method that tests whether the implementation works as expected using assert as we did in lecture (this is a simple approach to unit testing, we'll cover a better approach later).

Finally, make sure that your four counters work with the `PolyCount.java` test program we provide; it's probably a good idea to read and understand it. :-)

Hints

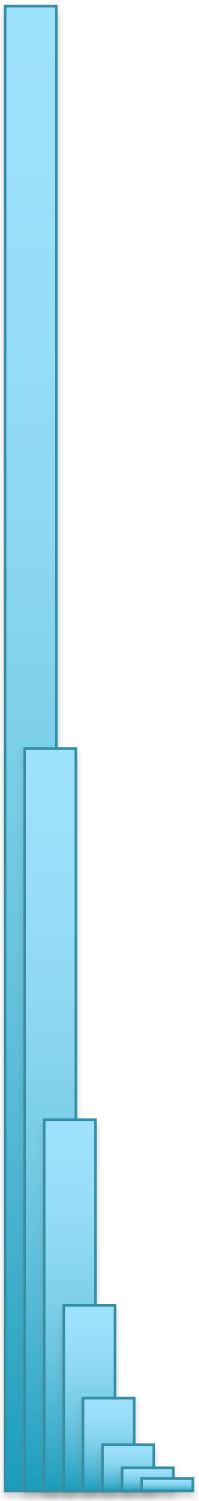
- Pay attention to your use of `public` and `private`! The essence of those counters is not just to hold a bunch of data, but to ensure that a certain approach to counting is followed; making everything public is a bad idea here.
- Remember that interfaces can extend one another in a way similar to classes (using the `extends` keyword). Classes implement interfaces however (using the `implements` keyword).

GradeScope.com
Entry Code: MDJYER

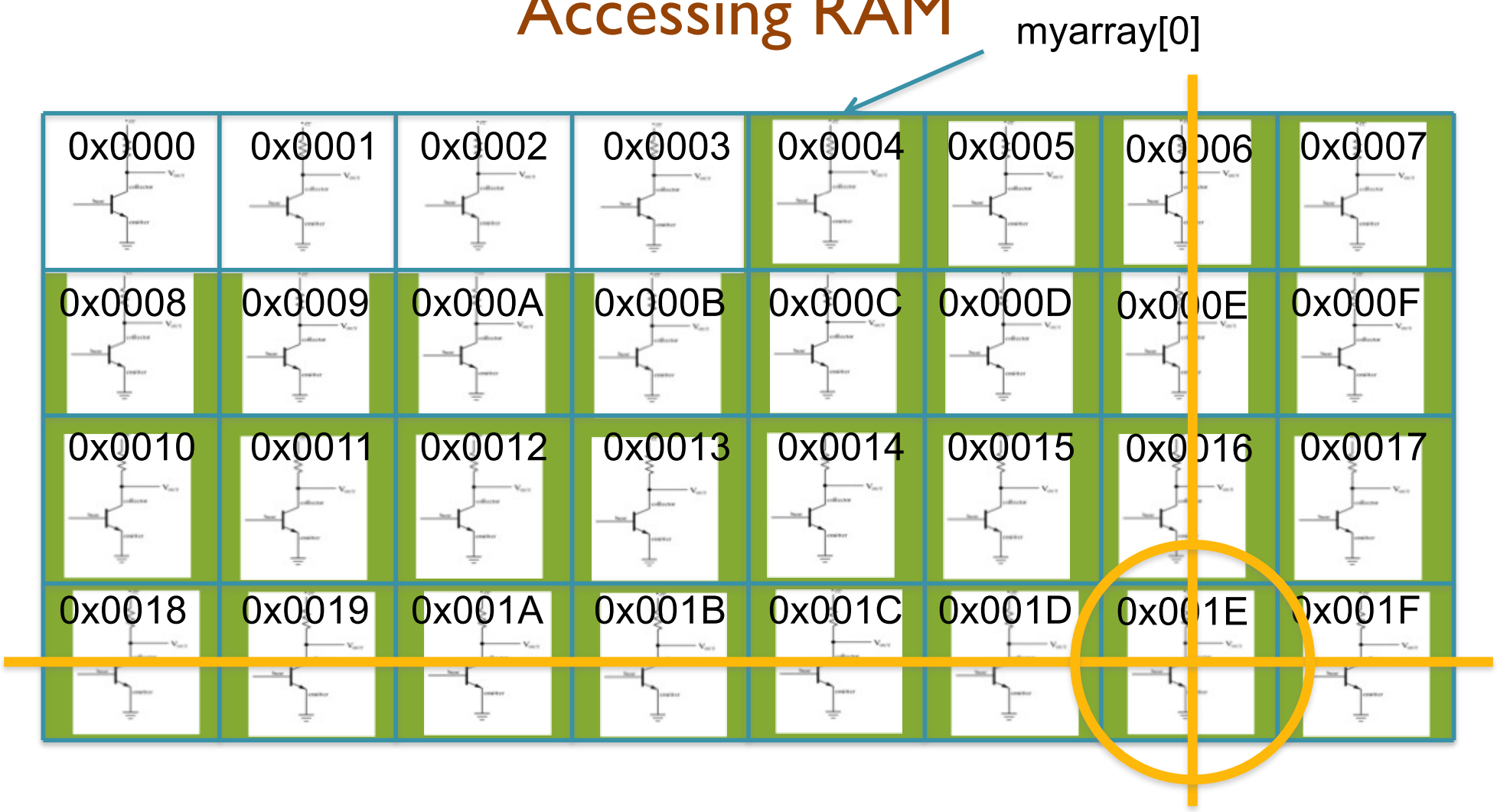


Agenda

1. *Review HW1*
2. *References and Linked Lists*
3. *Nested Classes and Iterators*



Accessing RAM

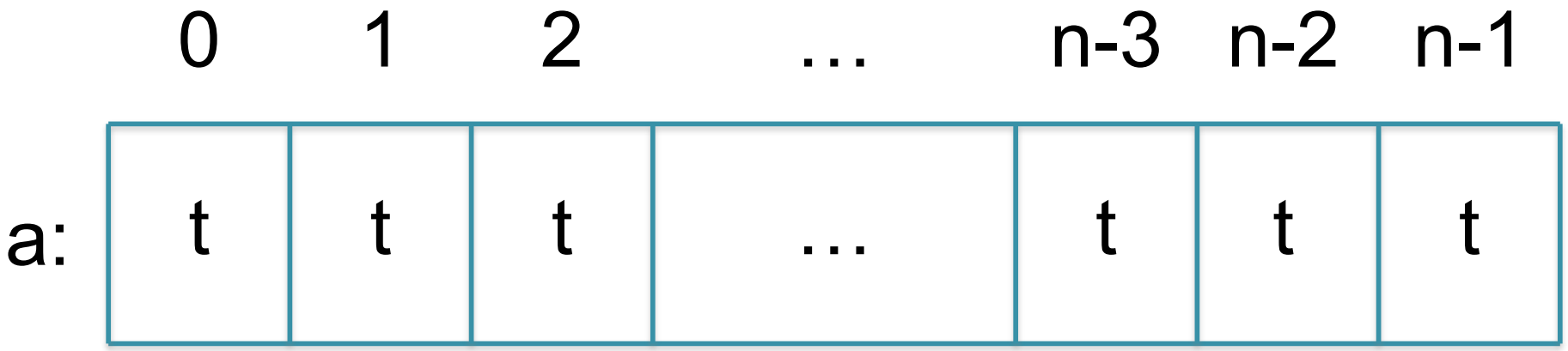


Byte [] myarray = new myarray[5000]; // myarray now starts at 0x0004

myarray[5] => offset for myarray + 5 * (sizeof type) => 0x04 + 0x05 * 1 => movl 0x09, %eax

myarray[26] => offset for myarray + 26 * (sizeof type) => 0x04 + 0x01A * 1 => movl 0x1E, %eax

ADT: Arrays

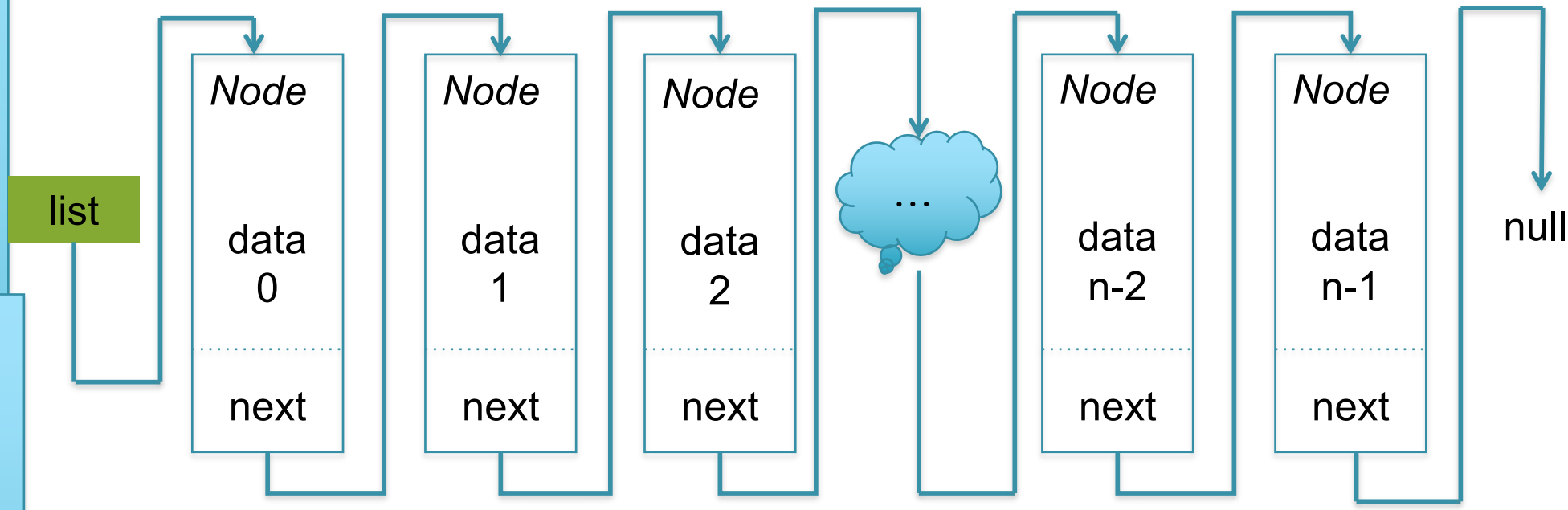


- Fixed length data structure
- Constant time get() and put() methods
- Definitely needs to be generic ☺

What are the limitations of arrays?

- Fixed length data structure
  - wastes space and/or cant grow
- Adding/removing from the middle is slow
- Searching can be slow (if not sorted)

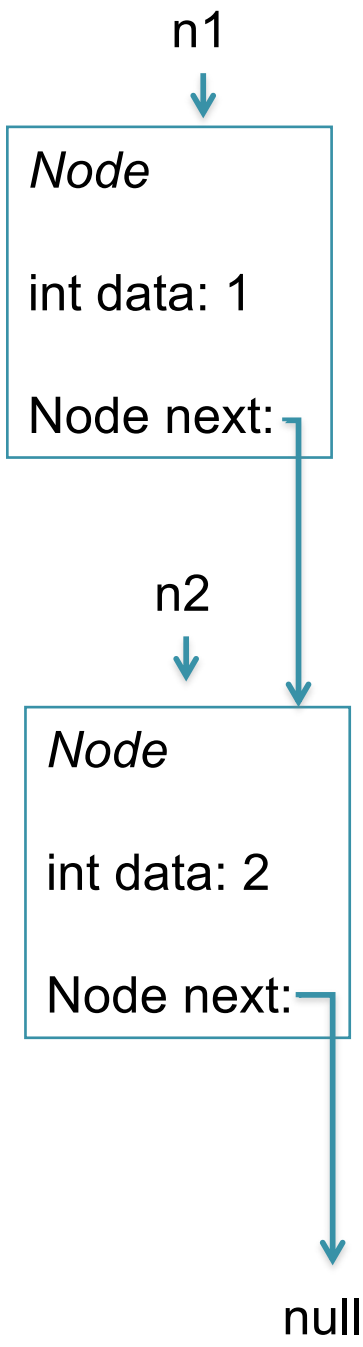
ADT: Linked List
(Singly Linked List)



- Variable length data structure
- Constant time to head of list
- Linear time seek, easy to add/remove to middle once found

Java Nodes

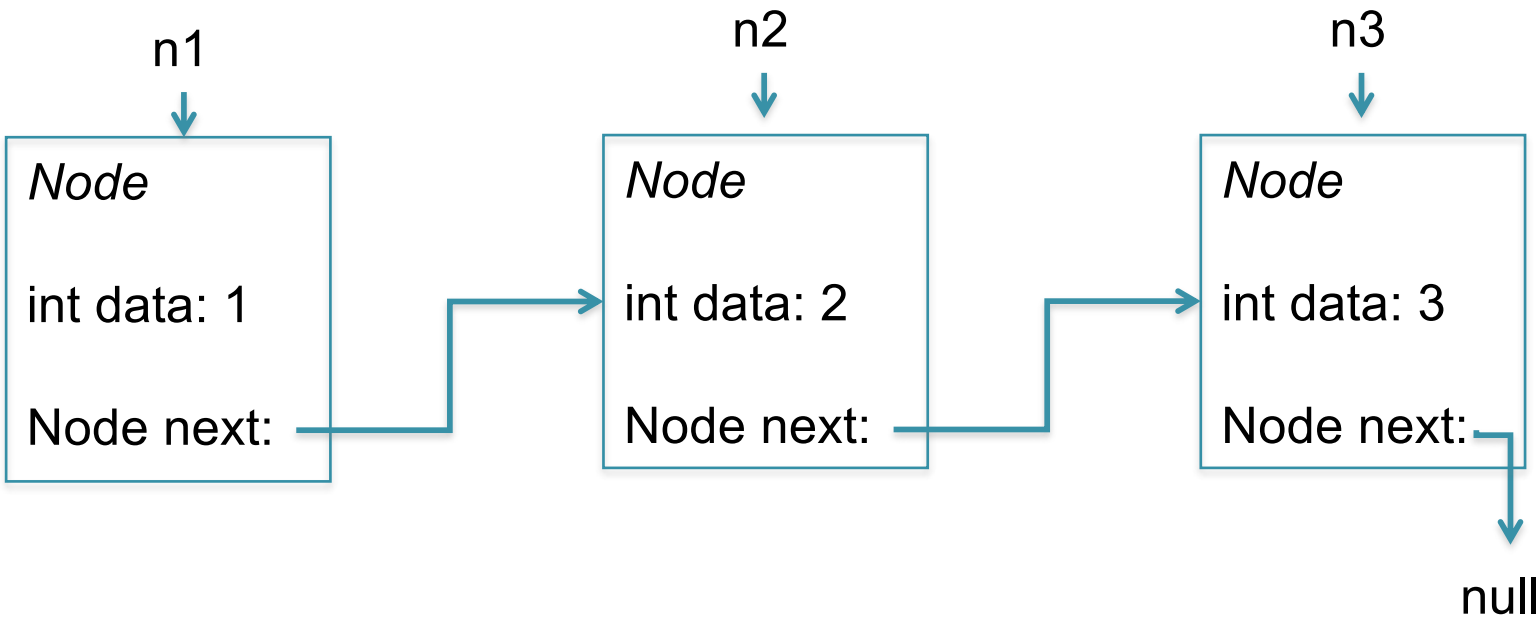
```
public class Node {
  private int data;
  private Node next;

  public Node(int d) {
    this.data = d;
    this.next = null; // will do this by default
  }

  public void setNext(Node n) {
    this.next = n;
  }

  public Node getNext() {
    return this.next;
  }

  public int getData() {
    return this.data;
  }

Node n1 = new Node(1);
Node n2 = new Node(2);
n1.setNext(n2);
```



Java Nodes

```
Node n1 = new Node(1);
Node n2 = new Node(2);
Node n3 = new Node(3);
n1.setNext(n2);
n2.setNext(n3);

System.out.println(n1.getData() + " -> " + n1.getNext().getData());
System.out.println(n2.getData() + " -> " + n2.getNext().getData());
System.out.println(n3.getData() + " -> " + n3.getNext().getData());
```

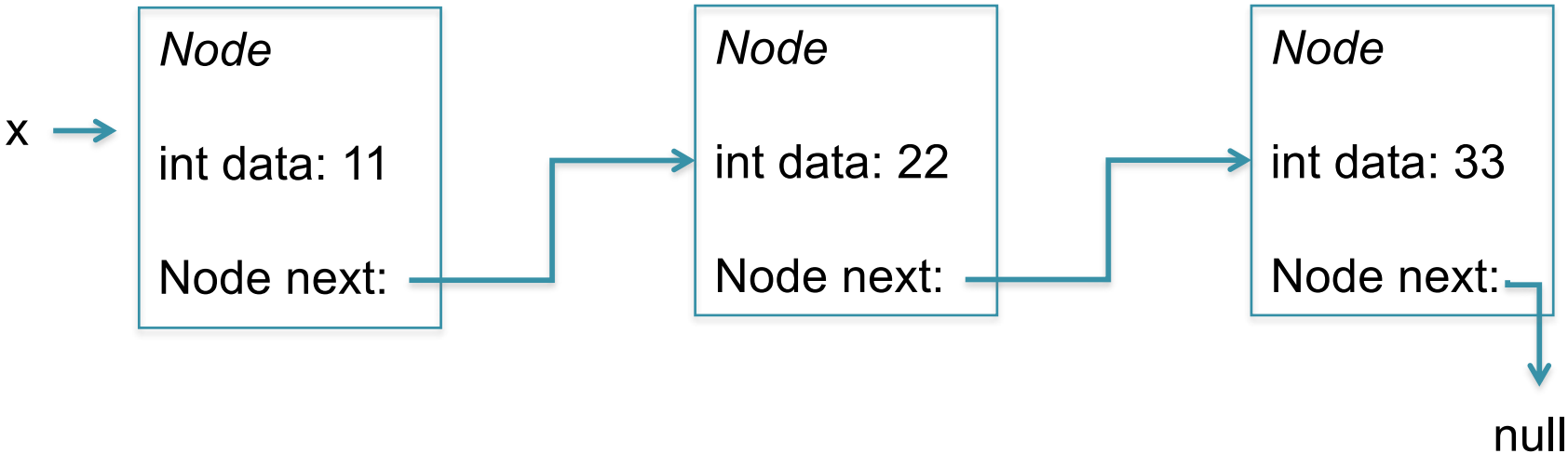
```
1 -> 2
2 -> 3
Exception in thread "main" java.lang.NullPointerException
        at Node.main(Node.java:32)
```



Java Nodes

```
Node x = new Node(11);
x.setNext(new Node(22));
x.getNext().setNext(new Node(33));

System.out.println(x.getData() + " -> " +
                   x.getNext().getData());
System.out.println(x.getNext().getData() + " -> " +
                   x.getNext().getNext().getData());
System.out.println(x.getNext().getNext().getData() + " -> " +
                   x.getNext().getNext().getNext().getData());
```

```
11 -> 22
22 -> 33
Exception in thread "main" java.lang.NullPointerException
        at Node.main(Node.java:32)
```



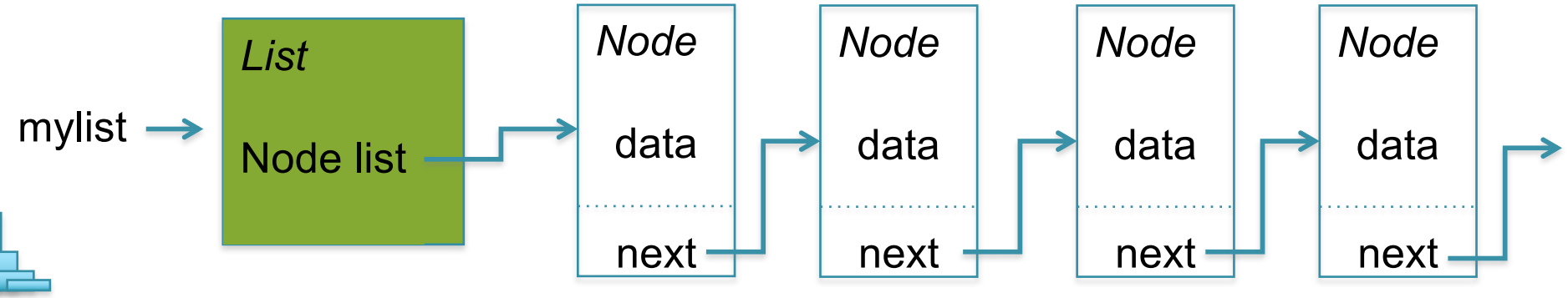
List Class

StringNode.java

```
public class StringNode {
  private String data;
  private StringNode next;

  public StringNode(String d) {
    this.data = d;
    this.next = null; // not necessary
  }

  public void setNext(StringNode n) {
    this.next = n;
  }

  public StringNode getNext() {
    return this.next;
  }

  public String getData() {
    return this.data;
  }
};
```

List.java

```
public class List {
  private StringNode list;

  public List() {
    this.list = null; // not necessary
  }

  public static void main(String[] args) {
    List mylist = new List();
  }
};
```



Linked List Construction


```
mylist.add("Mike");
```

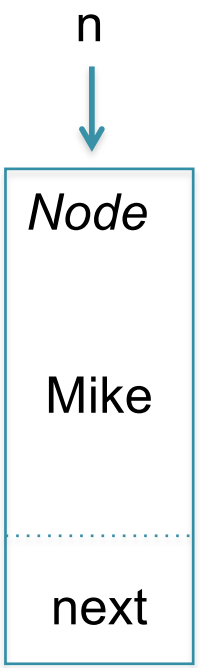
Linked List Construction


```
mylist.add("Mike");
```

List.java

```
public void add(String value){
    StringNode n = new StringNode(value);
    n.setNext(this.list);
    this.list = n;
 }
```

Linked List Construction

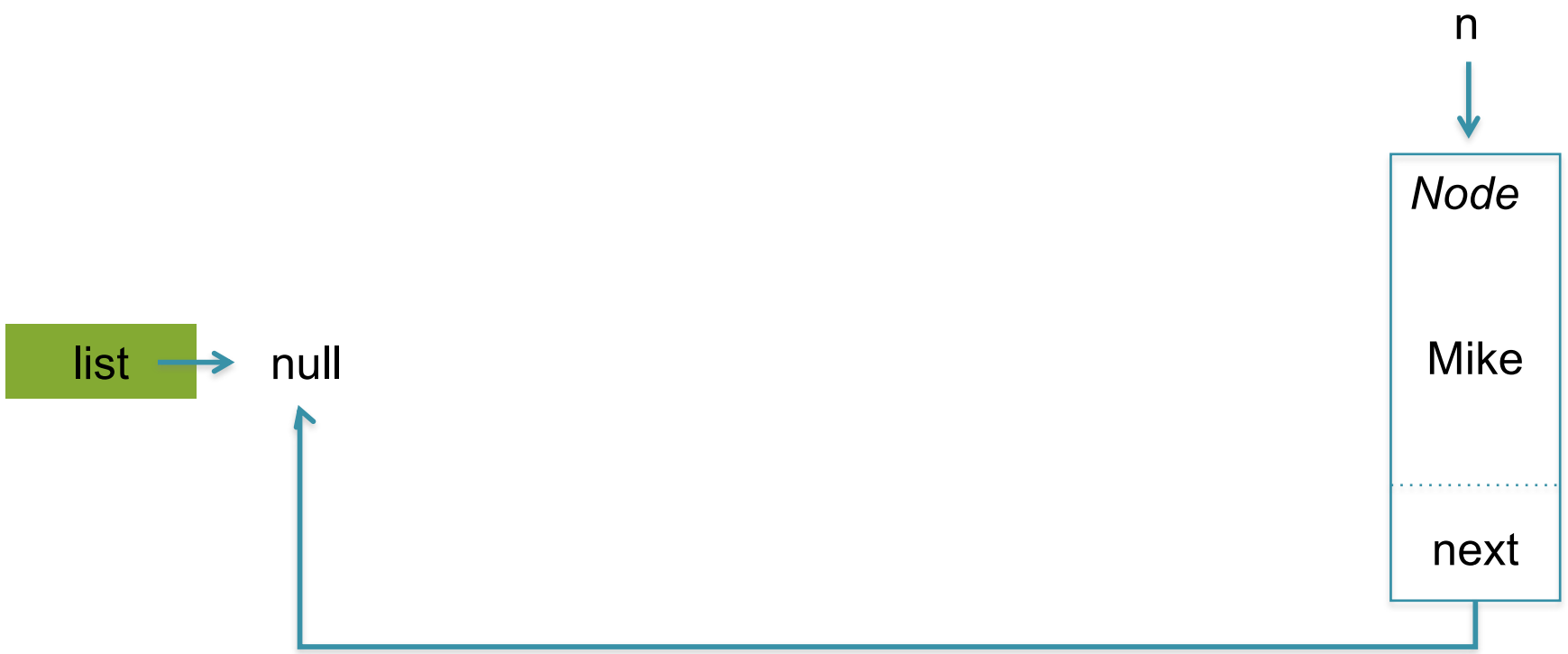



```
mylist.add("Mike");
```

List.java

```
public void add(String value){
   StringNode n = new StringNode(value);
   n.setNext(this.list);
   this.list = n;
 }
```

Linked List Construction

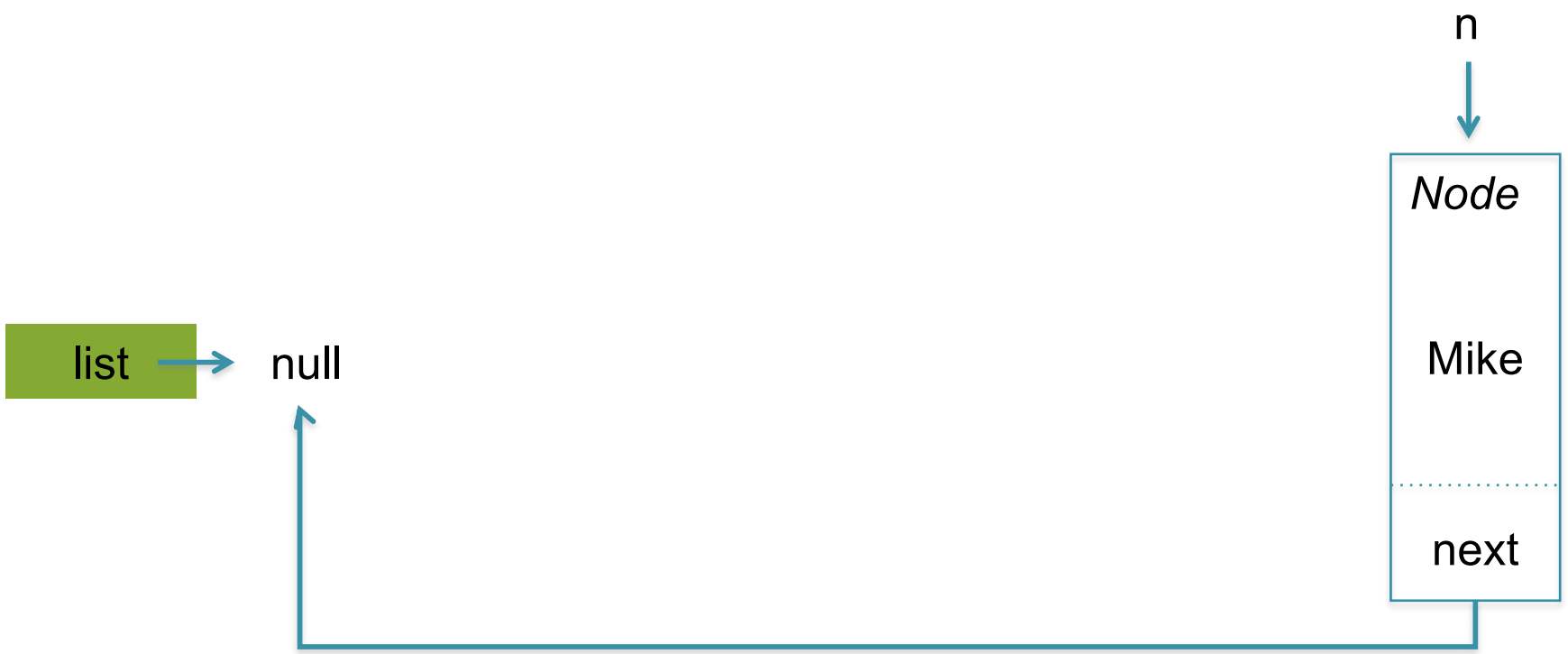



```
mylist.add("Mike");
```

List.java

```
public void add(String value){
   StringNode n = new StringNode(value);
   n.setNext(this.list);
   this.list = n;
 }
```

Linked List Construction




```
mylist.add("Mike");
```

List.java

```
public void add(String value){
   StringNode n = new StringNode(value);
   n.setNext(this.list);
   this.list = n;
 }
```

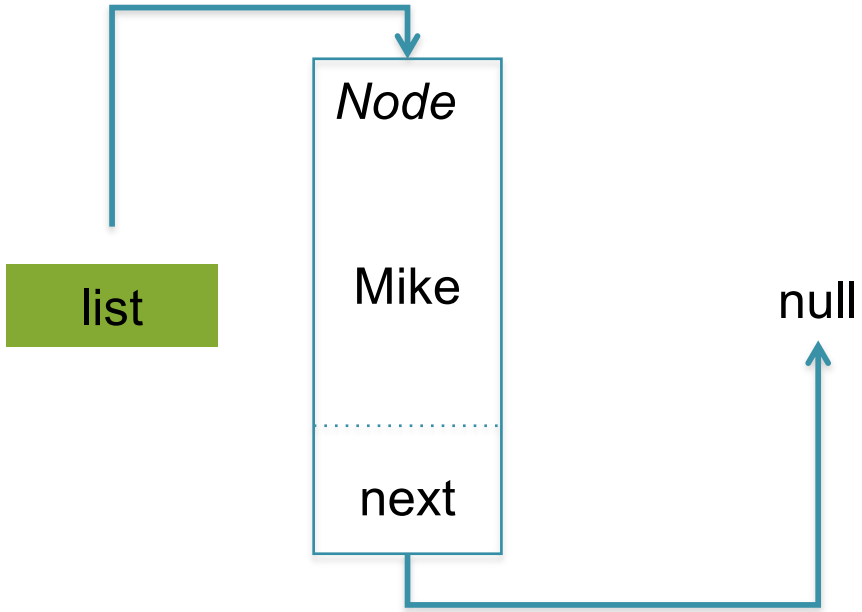
Linked List Construction


```
mylist.add("Mike");
```

List.java

```
public void add(String value){
   StringNode n = new StringNode(value);
   n.setNext(this.list);
   this.list = n;
 }
```

Linked List Construction

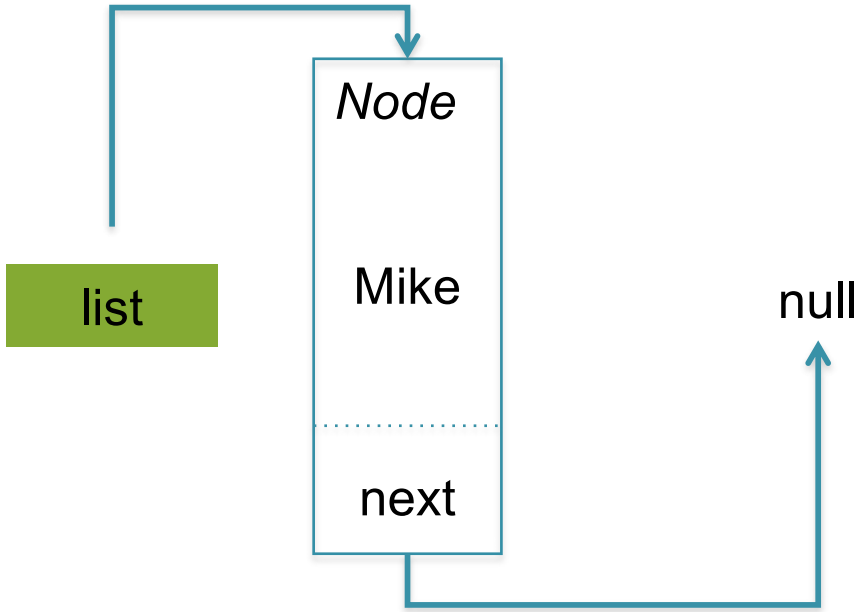



```
mylist.add("Mike");
```

List.java

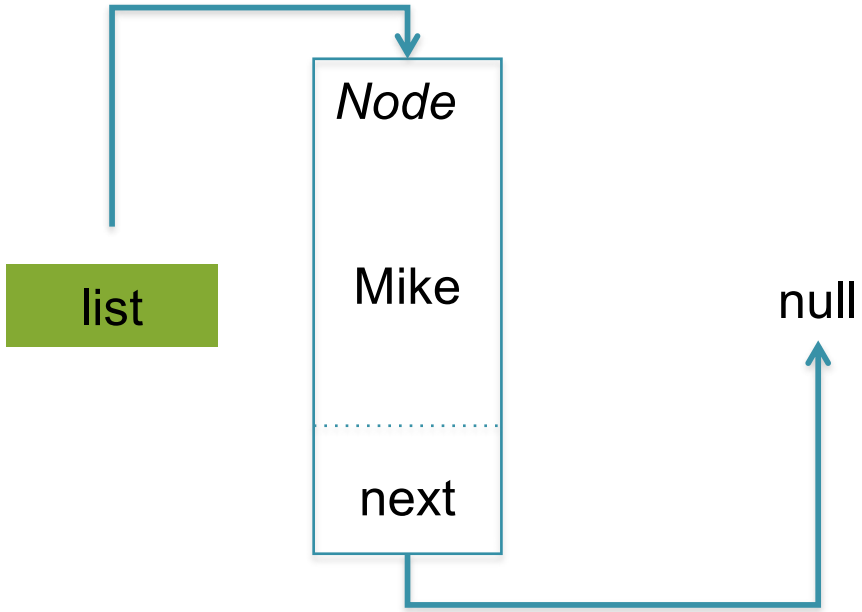
```
public void add(String value){
   StringNode n = new StringNode(value);
   n.setNext(this.list);
   this.list = n;
 }
```

Linked List Construction



First node successfully added (prepend)
Questions?

Linked List Construction

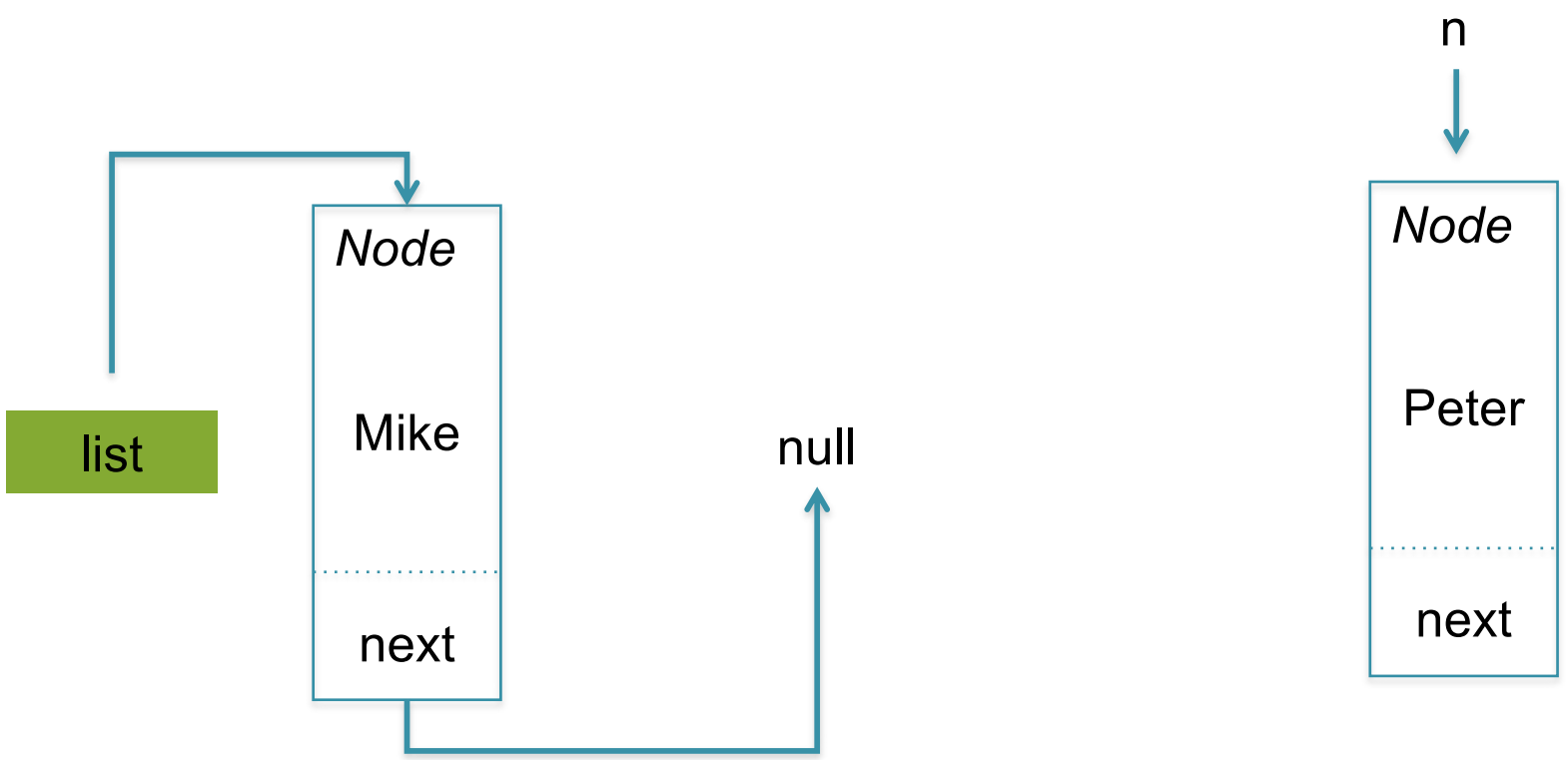



```
mylist.add("Peter");
```

List.java

```
public void add(String value){
   StringNode n = new StringNode(value);
   n.setNext(this.list);
   this.list = n;
 }
```

Linked List Construction

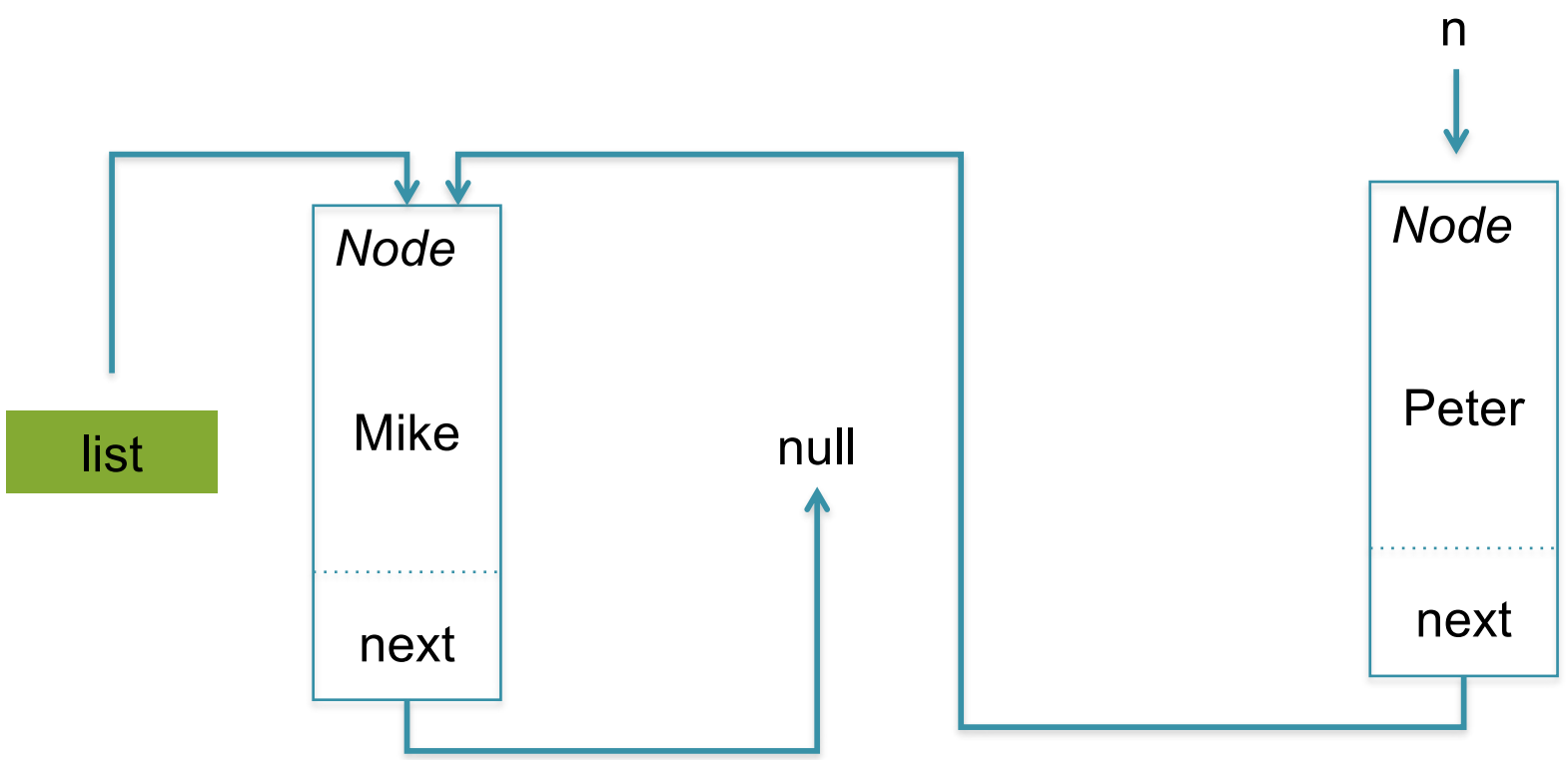



```
mylist.add("Peter");
```

List.java

```
public void add(String value){
   StringNode n = new StringNode(value);
   n.setNext(this.list);
   this.list = n;
 }
```

Linked List Construction

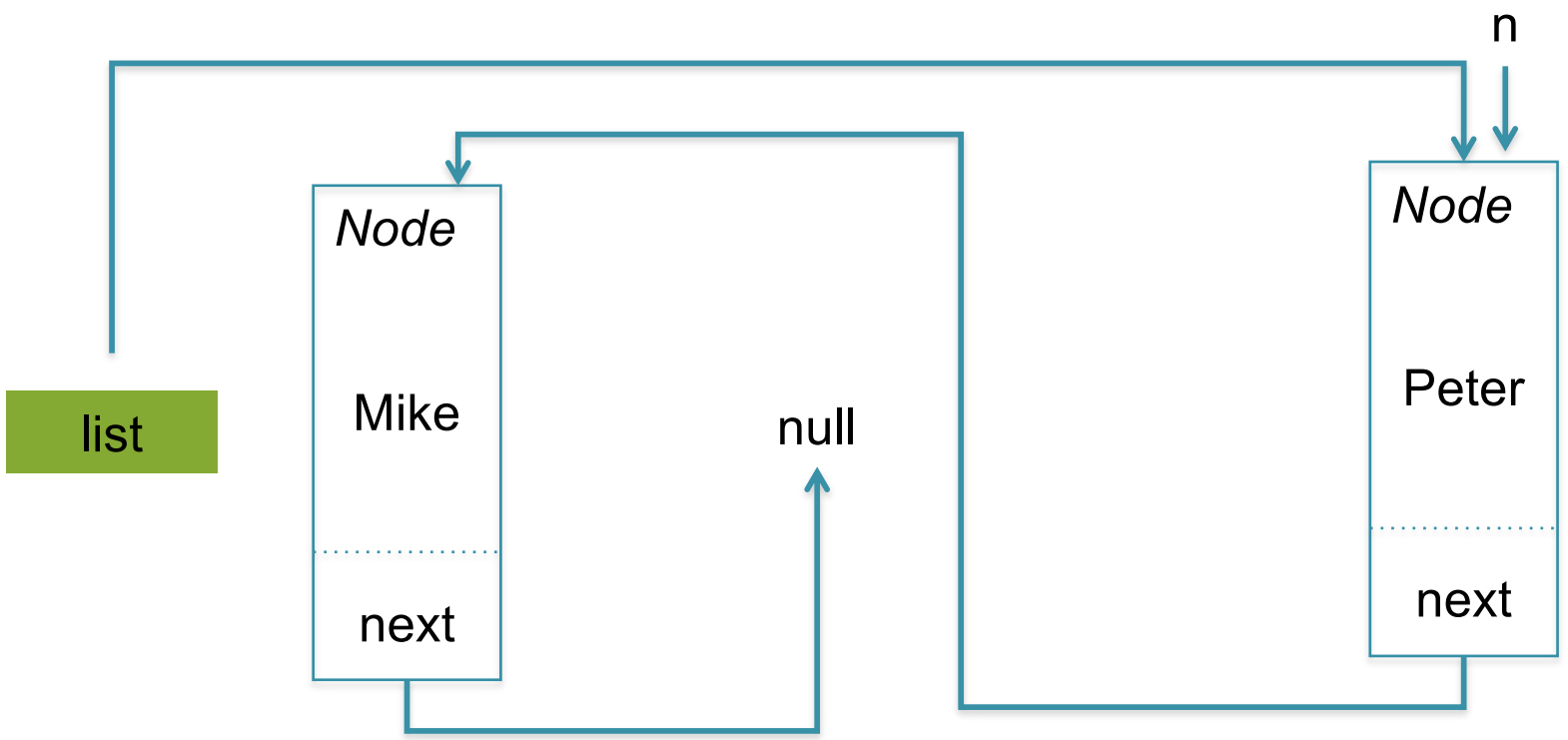



```
mylist.add("Peter");
```

List.java

```
public void add(String value){
   StringNode n = new StringNode(value);
   n.setNext(this.list);
   this.list = n;
 }
```

Linked List Construction

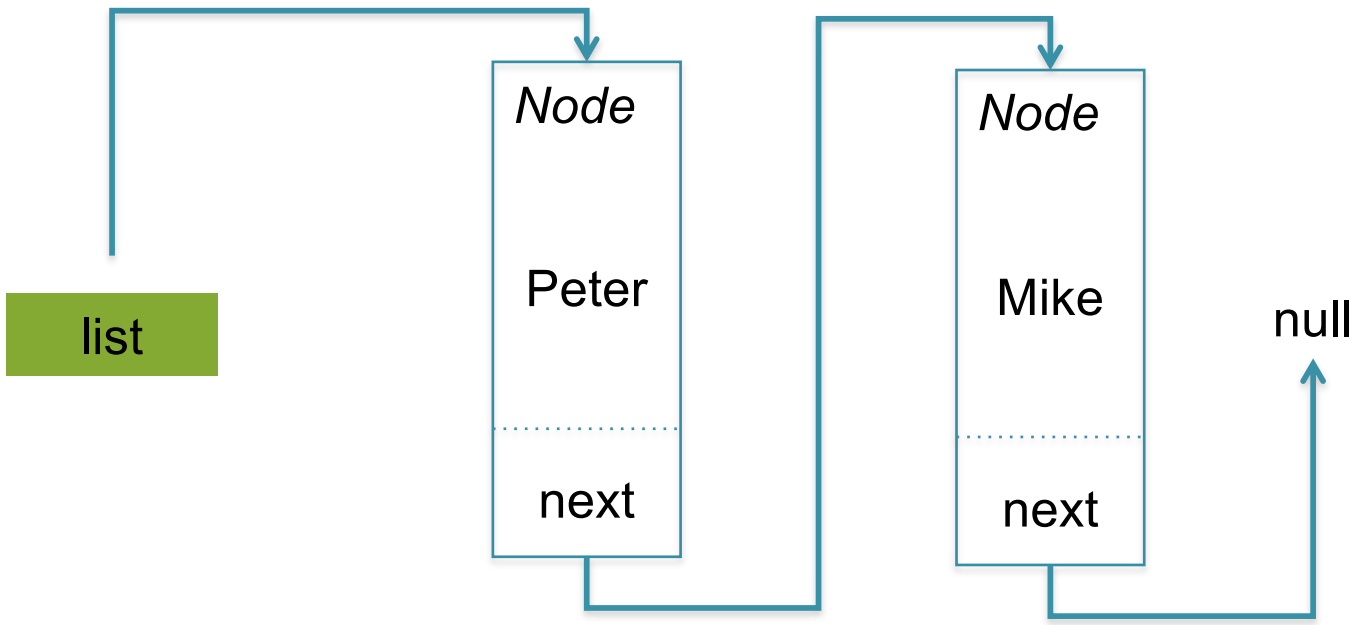



```
mylist.add("Peter");
```

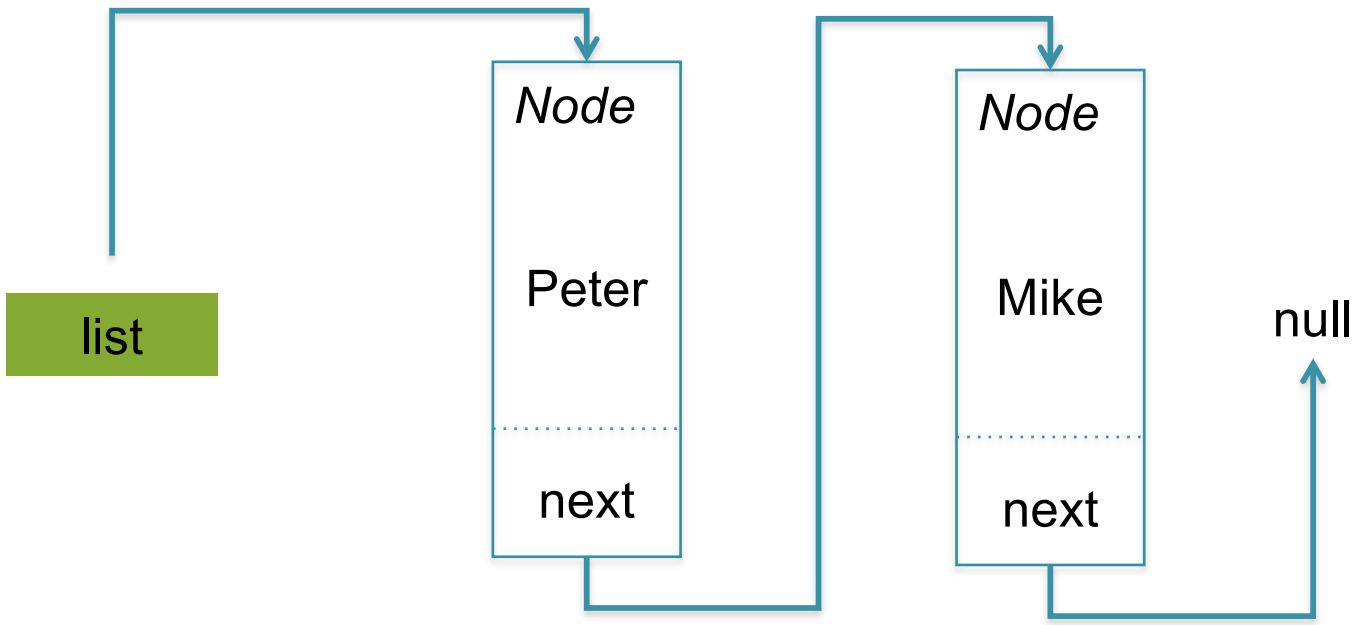
List.java

```
public void add(String value){
   StringNode n = new StringNode(value);
   n.setNext(this.list);
   this.list = n;
 }
```

Linked List Construction



Linked List Construction

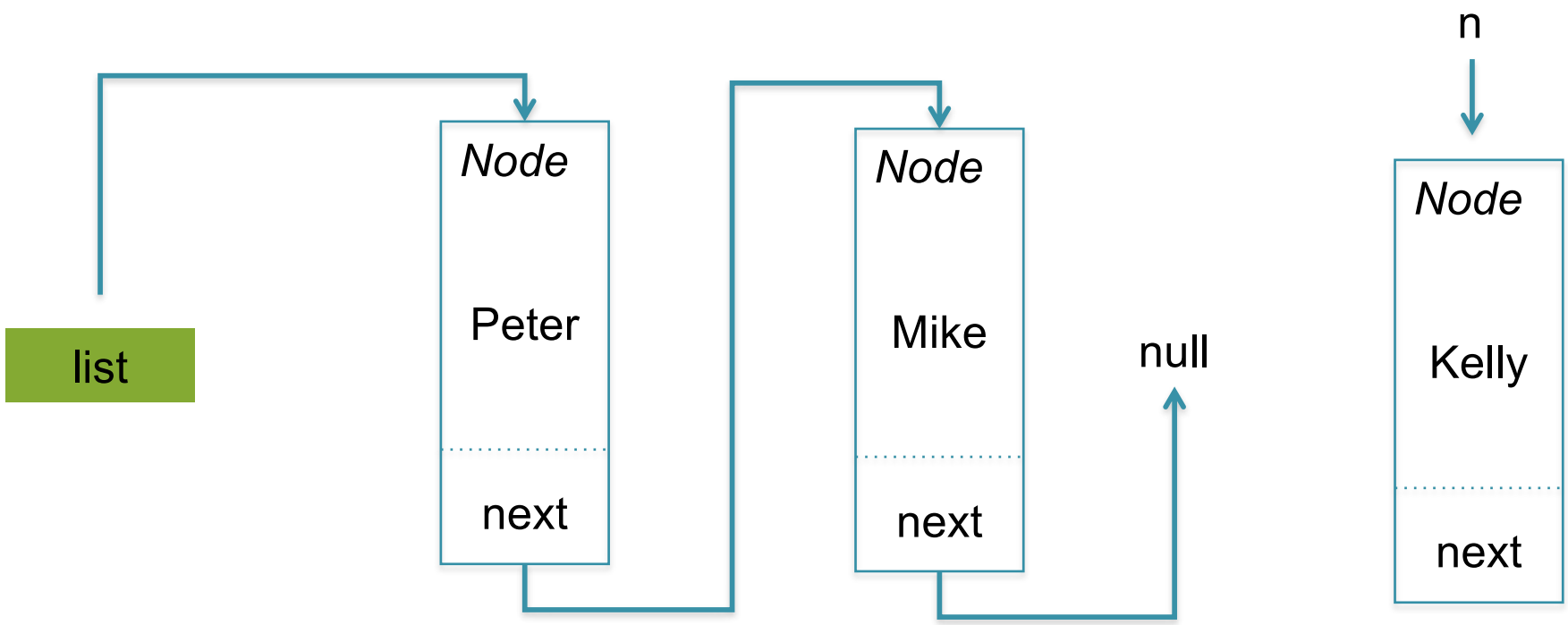



```
mylist.add("Kelly");
```

List.java

```
public void add(String value){
   StringNode n = new StringNode(value);
   n.setNext(this.list);
   this.list = n;
 }
```

Linked List Construction

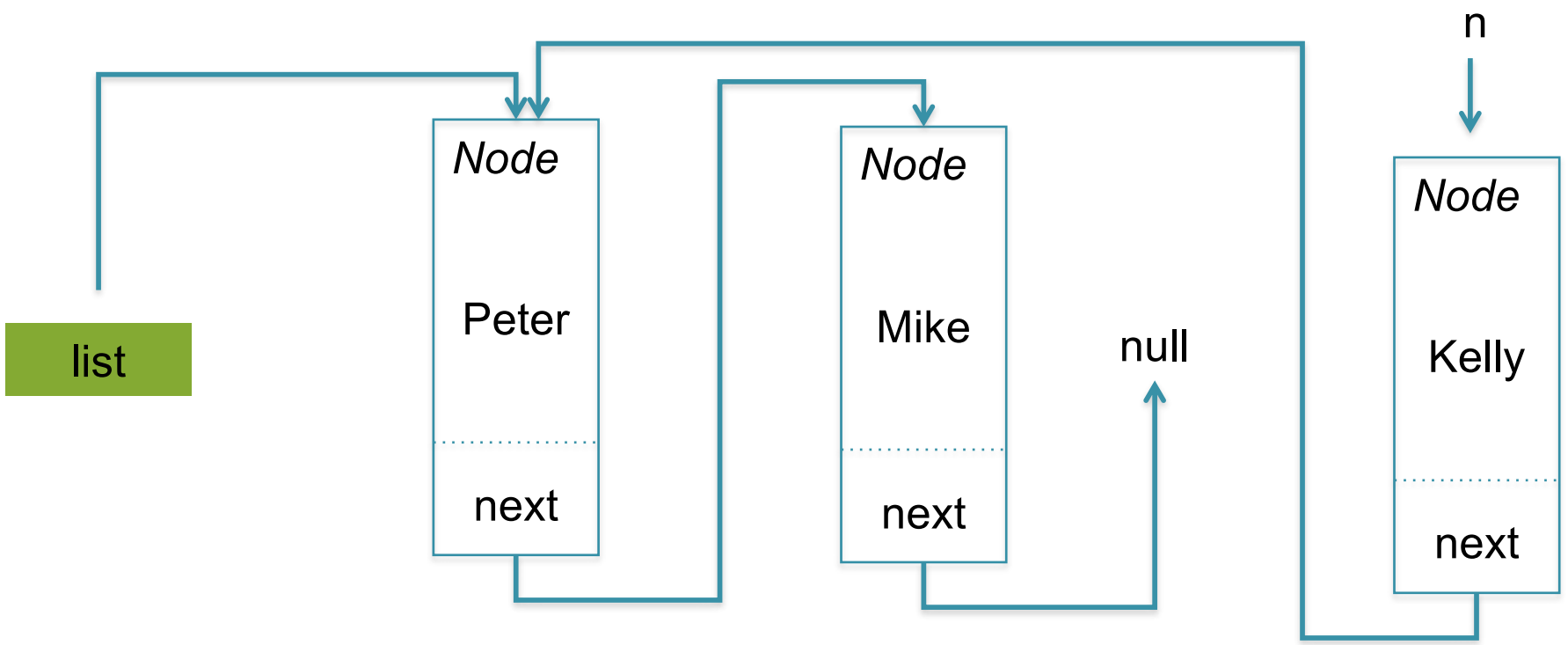



```
mylist.add("Kelly");
```

List.java

```
public void add(String value){
   StringNode n = new StringNode(value);
   n.setNext(this.list);
   this.list = n;
 }
```

Linked List Construction

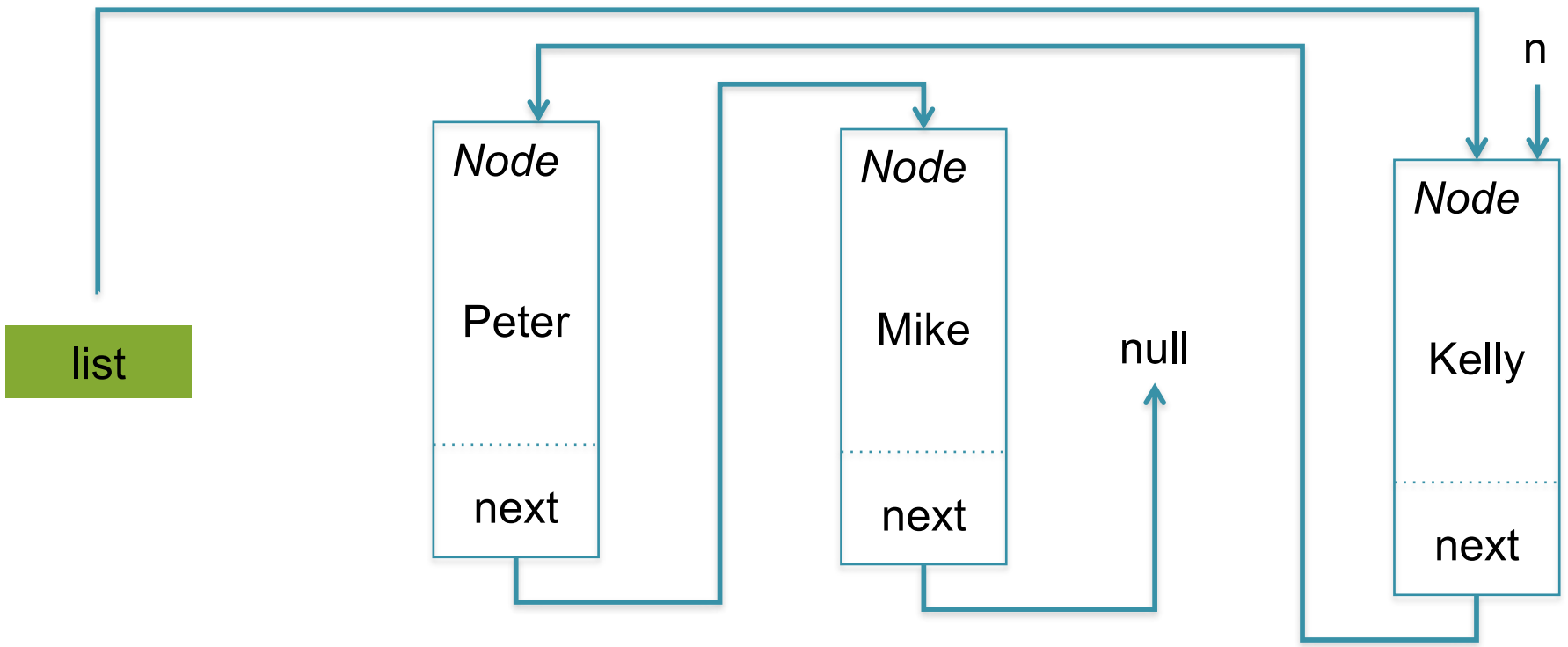



```
mylist.add("Kelly");
```

List.java

```
public void add(String value){
   StringNode n = new StringNode(value);
   n.setNext(this.list);
   this.list = n;
 }
```

Linked List Construction

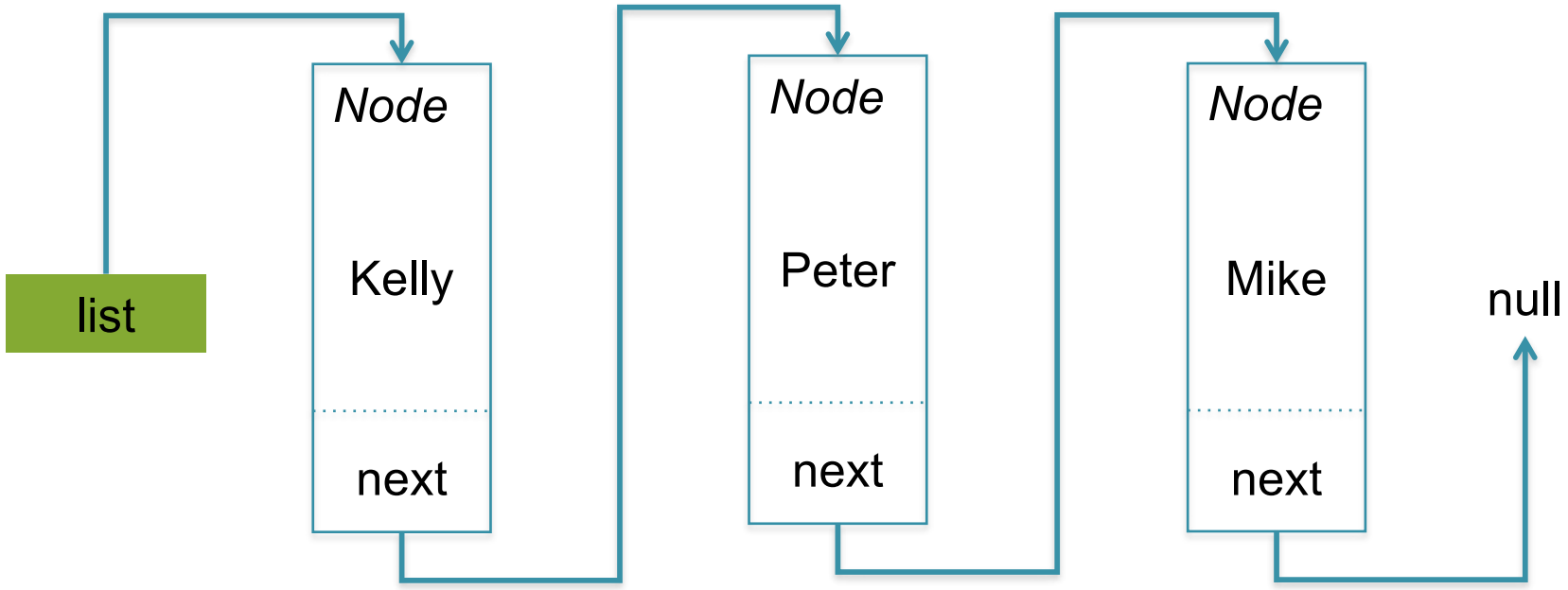



```
mylist.add("Kelly");
```

List.java

```
public void add(String value){
   StringNode n = new StringNode(value);
   n.setNext(this.list);
   this.list = n;
 }
```

Linked List Construction



Why do we insert at the beginning of the list (prepend)?
How long would it take to insert at the tail?
How could you make that faster?

Linked List Construction

list

null

Wh *end)?*

Searching

StringNode.java

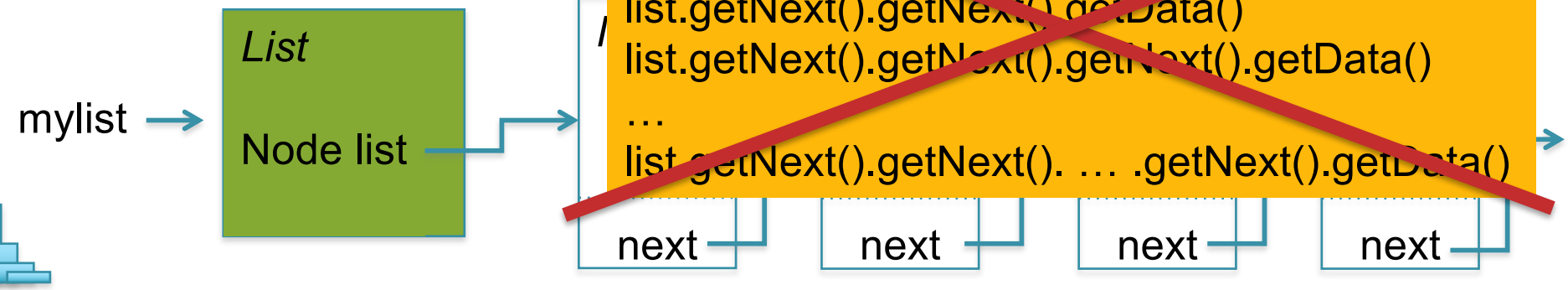
```
public class StringNode {
  private String data;
  private StringNode next;

  public StringNode(String d) {
    this.data = d;
    this.next = null; // not necessary
  }

  public void setNext(StringNode n) {
    this.next = n;
  }

  public StringNode getNext() {
    return this.next;
  }

  public String getData() {
    return this.data;
  }
};
```

List.java

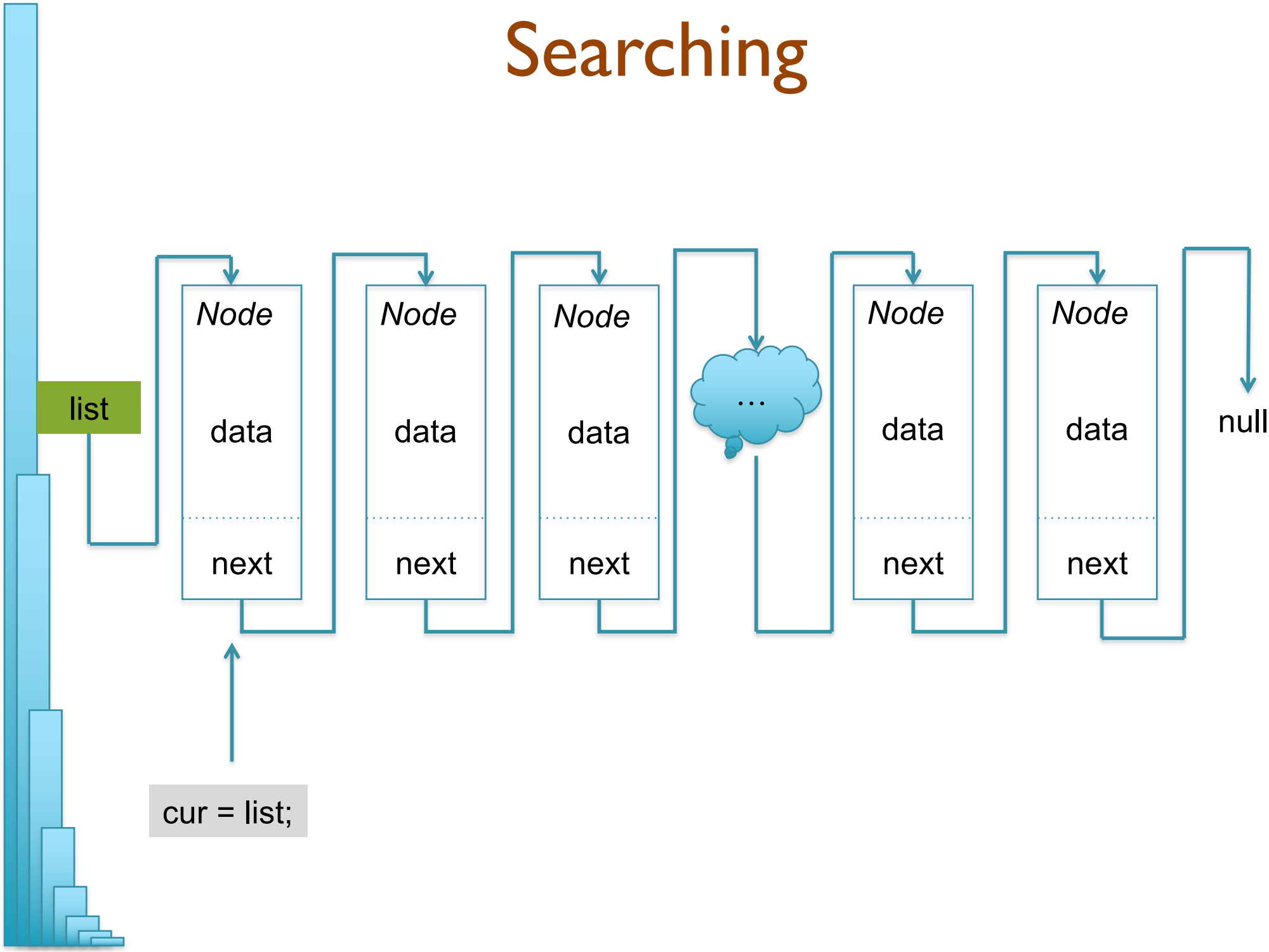
```
public class List {
  private StringNode list;

  public List() {
    this.list = null; // not necessary
  }

  public static void main(String[] args) {
    List mylist = new List();
  }
};
```

If you have a linked list of items (photos in Instagram), how would you search?

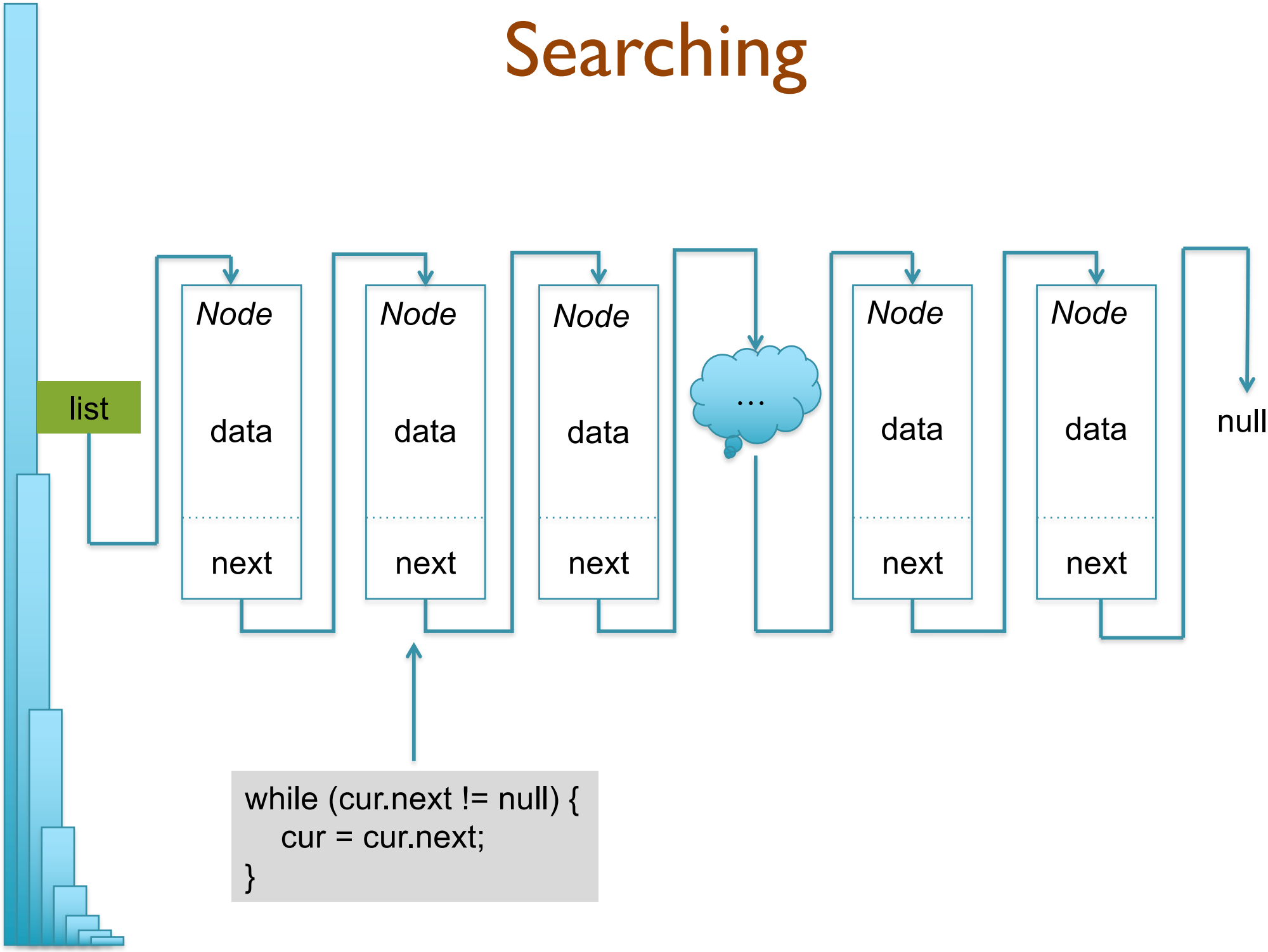
~~list.getData()~~
~~list.getNext().getData()~~
~~list.getNext().getNext().getData()~~
~~list.getNext().getNext().getNext().getData()~~
~~…~~
~~list.getNext().getNext(). … .getNext().getData()~~



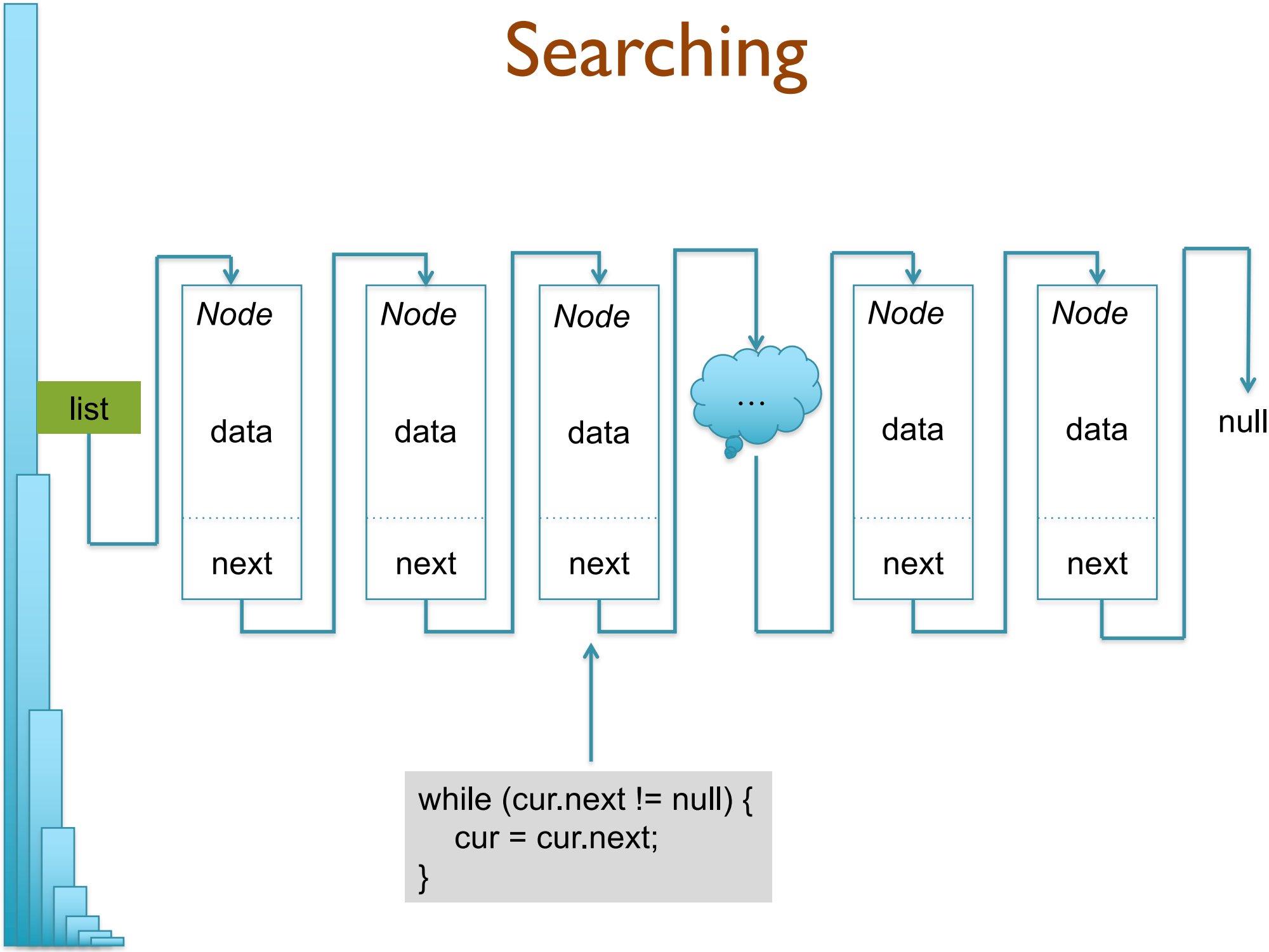
Searching


list
Node
data
next
Node
data
next
Node
data
next
...
Node
data
next
Node
data
next
null
cur = list;

Searching

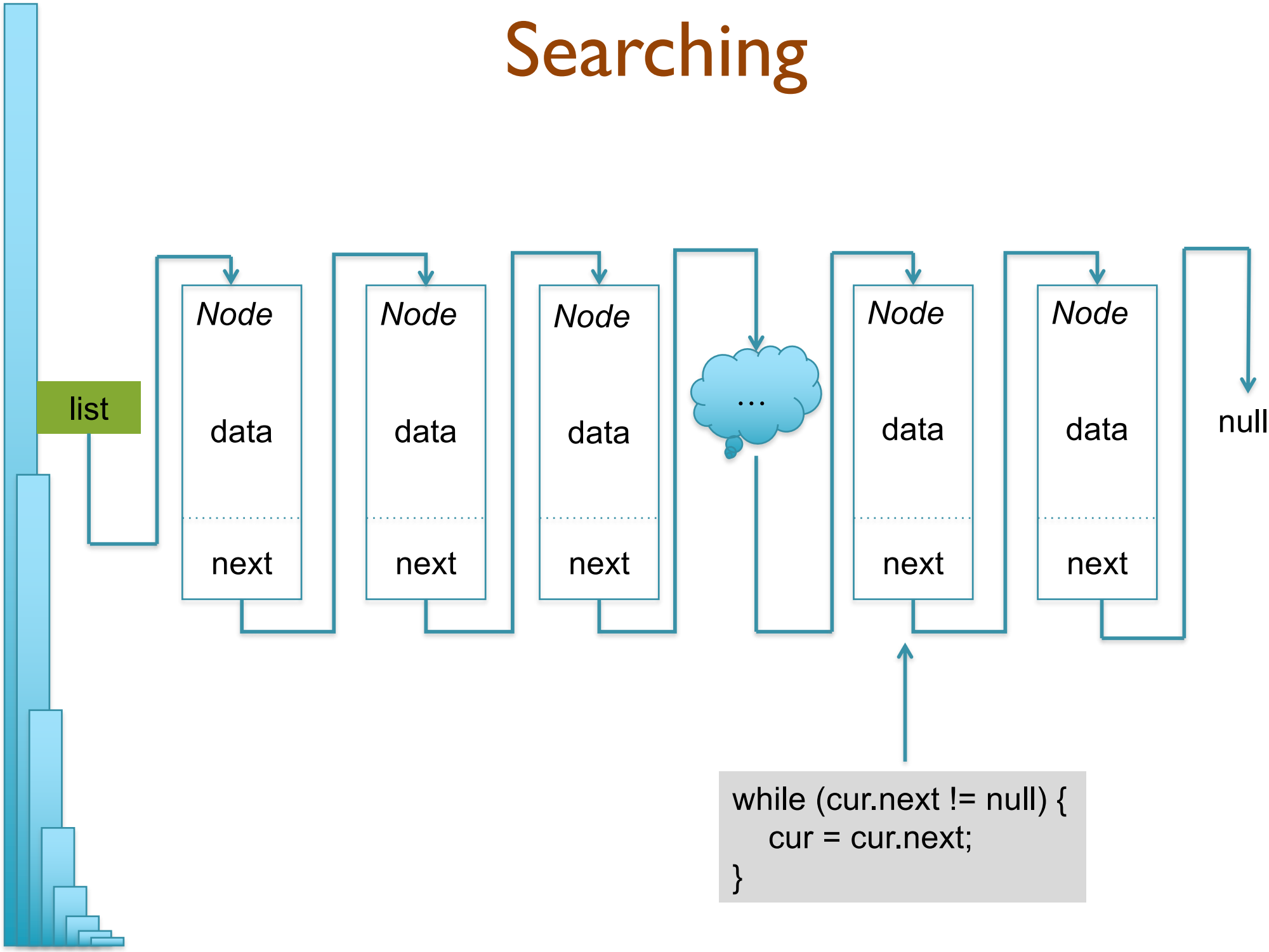

list
Node
data
next
Node
data
next
Node
data
next
...
Node
data
next
Node
data
next
null
while (cur.next != null) {
cur = cur.next;
}

Searching

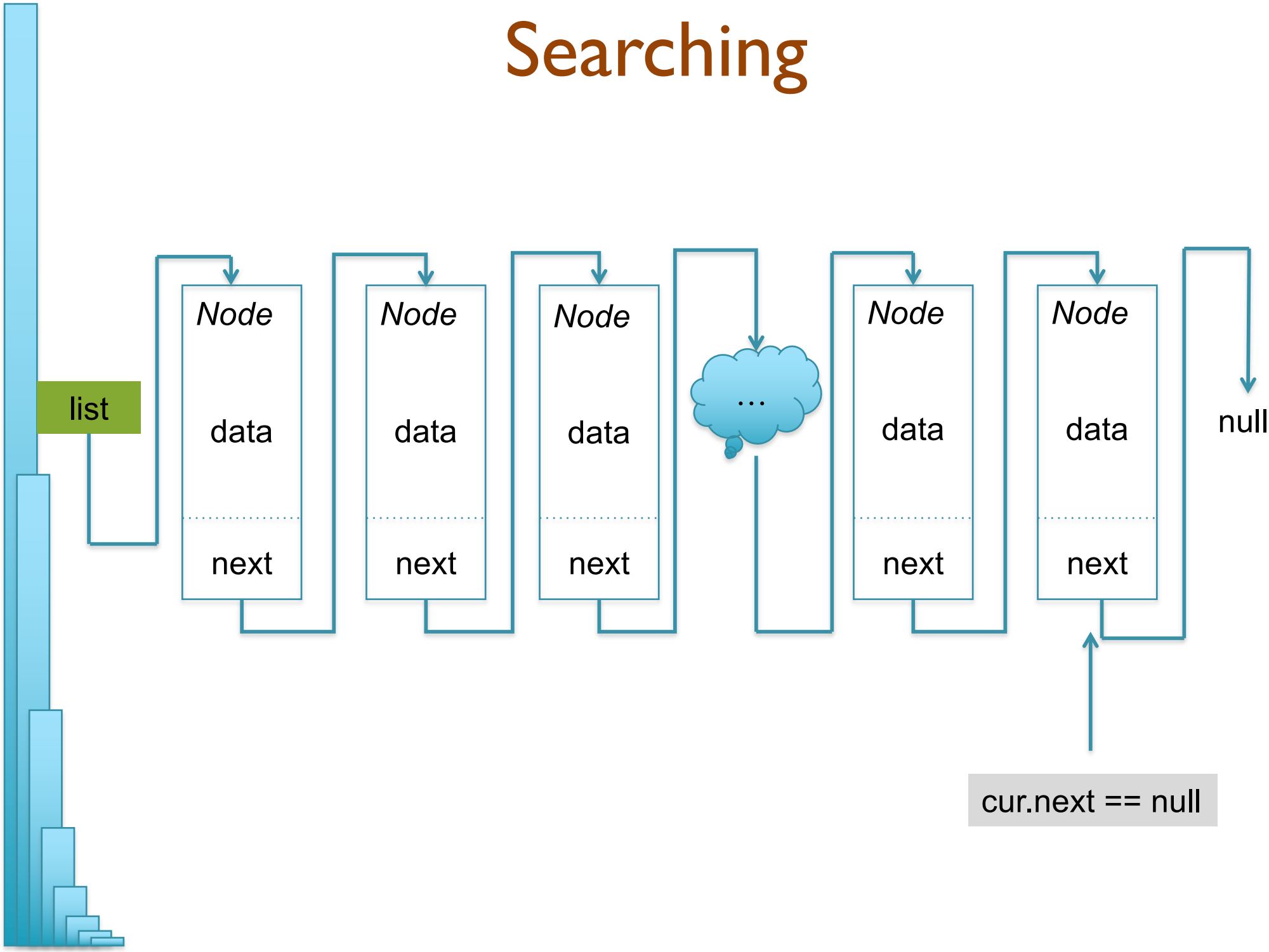



```
while (cur.next != null) {
    cur = cur.next;
}
```

Searching


list
Node
data
next
Node
data
next
Node
data
next
...
Node
data
next
Node
data
next
null
while (cur.next != null) {
cur = cur.next;
}

Searching


list
Node
data
next
Node
data
next
Node
data
next
...
Node
data
next
Node
data
next
null
cur.next == null

Linked List Searching

List.java

```
public StringNode find(String value) {
  StringNode cur = list;
  while (cur != null) {
    System.out.println("checking: " + value + " at " + cur.getData());
    if (cur.getData().equals(value)) {
      return cur;
    }

    cur = cur.getNext();
  }

  return cur; // must be null
}
```

```
List mylist = new List();
mylist.add("Mike");
mylist.add("Peter");
mylist.add("Kelly");
```

```
StringNode f1 = mylist.find("Mike");
System.out.println("f1 finished");
System.out.println(f1.getData());
```

Linked List Searching

List.java

```
public StringNode find(String value) {
  StringNode cur = list;
  while (cur != null) {
    System.out.println("checking: " + value + " at " + cur.getData());
    if (cur.getData().equals(value)) {
      return cur;
    }

    cur = cur.getNext();
  }

  return cur; // must be null
}
```

```
List mylist = new List();
mylist.add("Mike");
mylist.add("Peter");
mylist.add("Kelly");
```

```
StringNode f1 = mylist.find("Mike");
System.out.println("f1 finished");
System.out.println(f1.getData());
```

```
checking: Mike at Kelly
checking: Mike at Peter
checking: Mike at Mike
f1 finished
Mike
```

Linked List Searching

List.java

```
public StringNode find(String value) {
  StringNode cur = list;
  while (cur != null) {
    System.out.println("checking: " + value + " at " + cur.getData());
    if (cur.getData().equals(value)) {
      return cur;
    }

    cur = cur.getNext();
  }

  return cur; // must be null
}
```

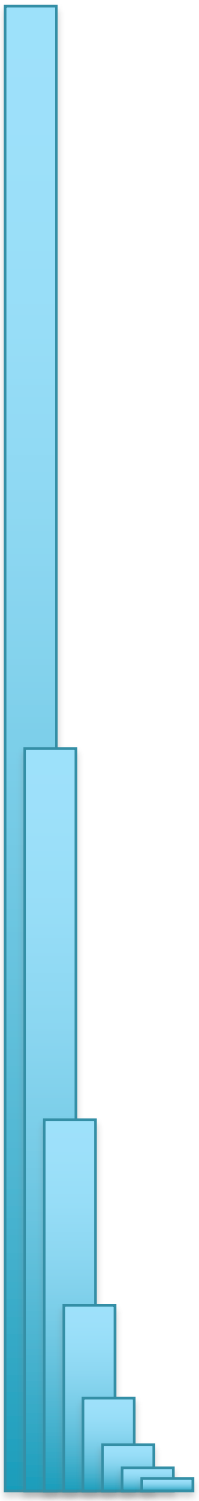
```
List mylist = new List();
mylist.add("Mike");
mylist.add("Peter");
mylist.add("Kelly");
```

```
StringNode f2 = mylist.find("James");
System.out.println("f2 finished");
System.out.println(f2.getData());
```

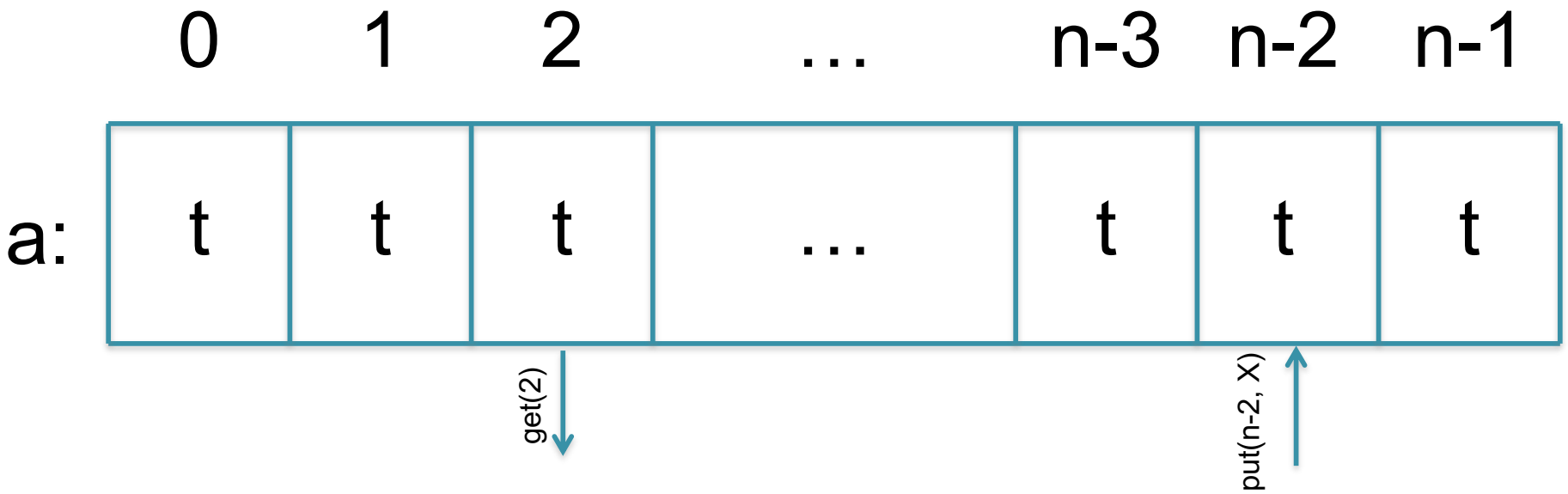
```
checking: James at Kelly
checking: James at Peter
checking: James at Mike
f2 finished
Exception in thread "main"
java.lang.NullPointerException
    at List.main(List.java:43)
```

Agenda

1. *Review HW1*
2. *References and Linked Lists*
3. *Nested Classes and Iterators*



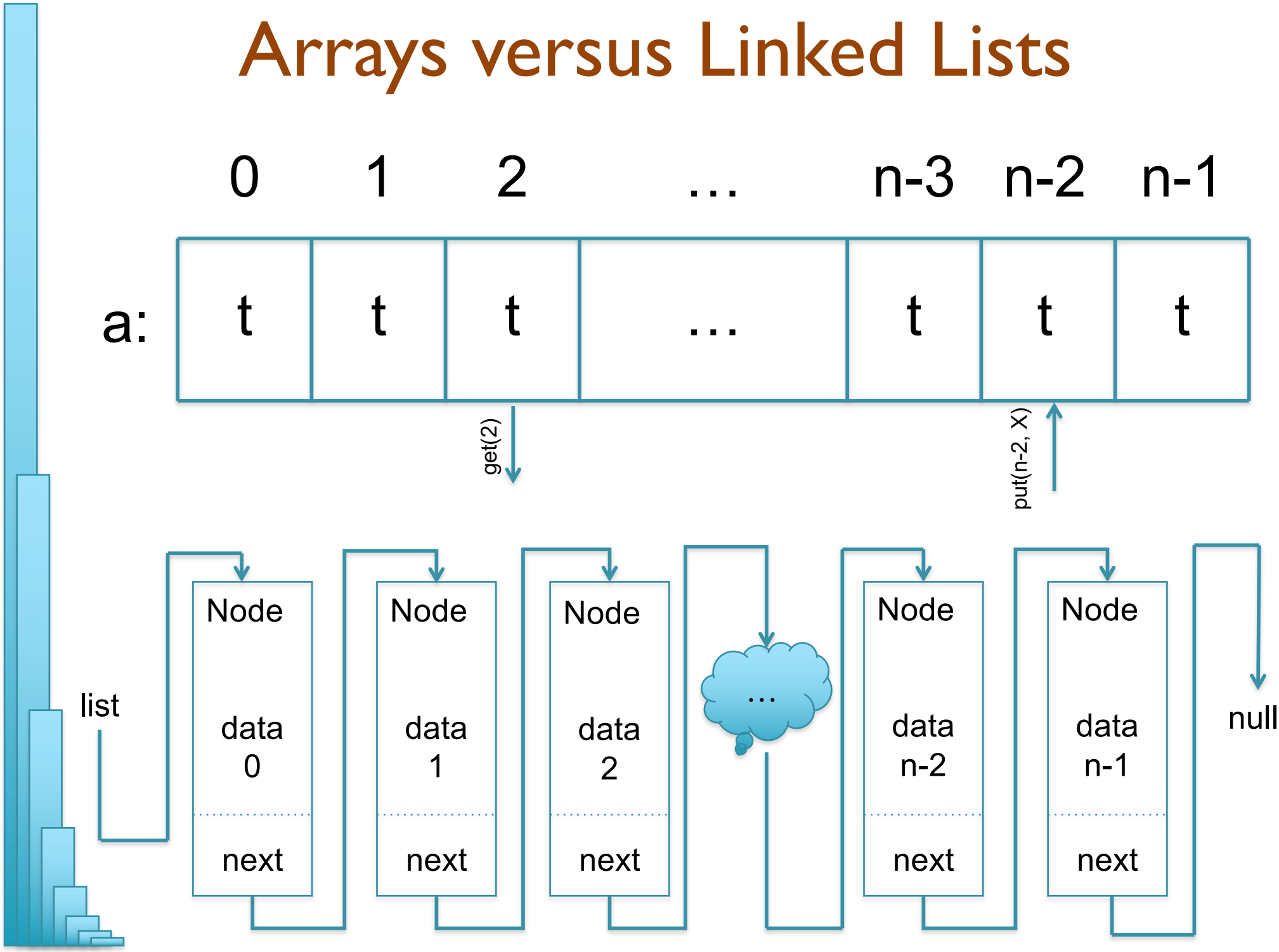
Arrays versus Linked Lists



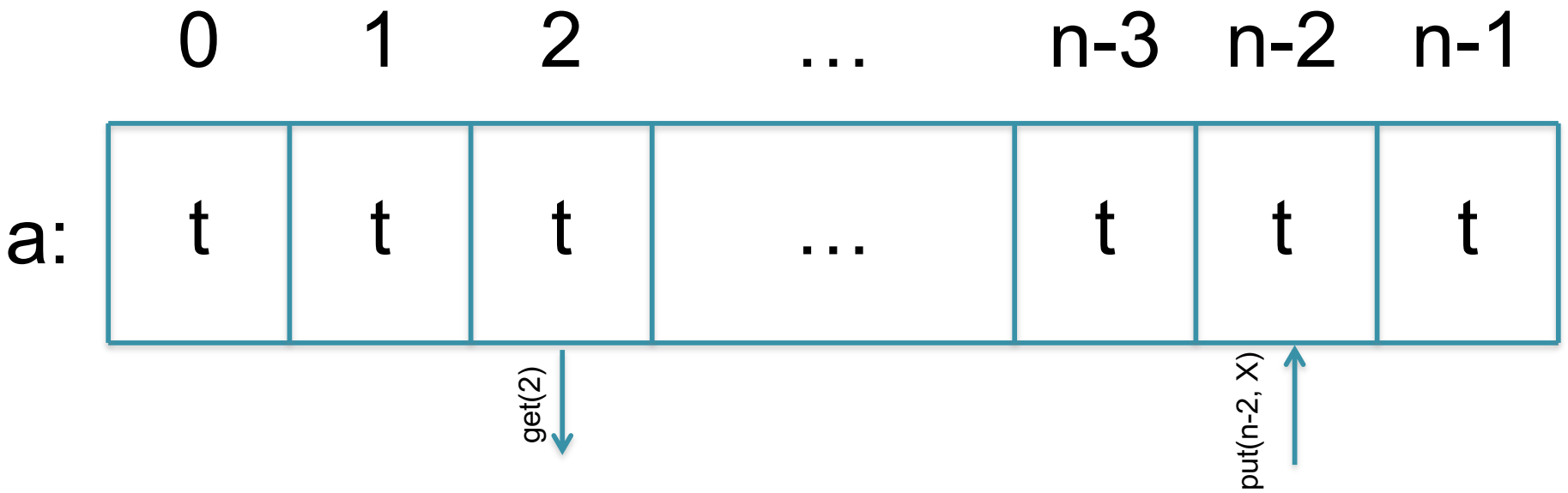
Array.java

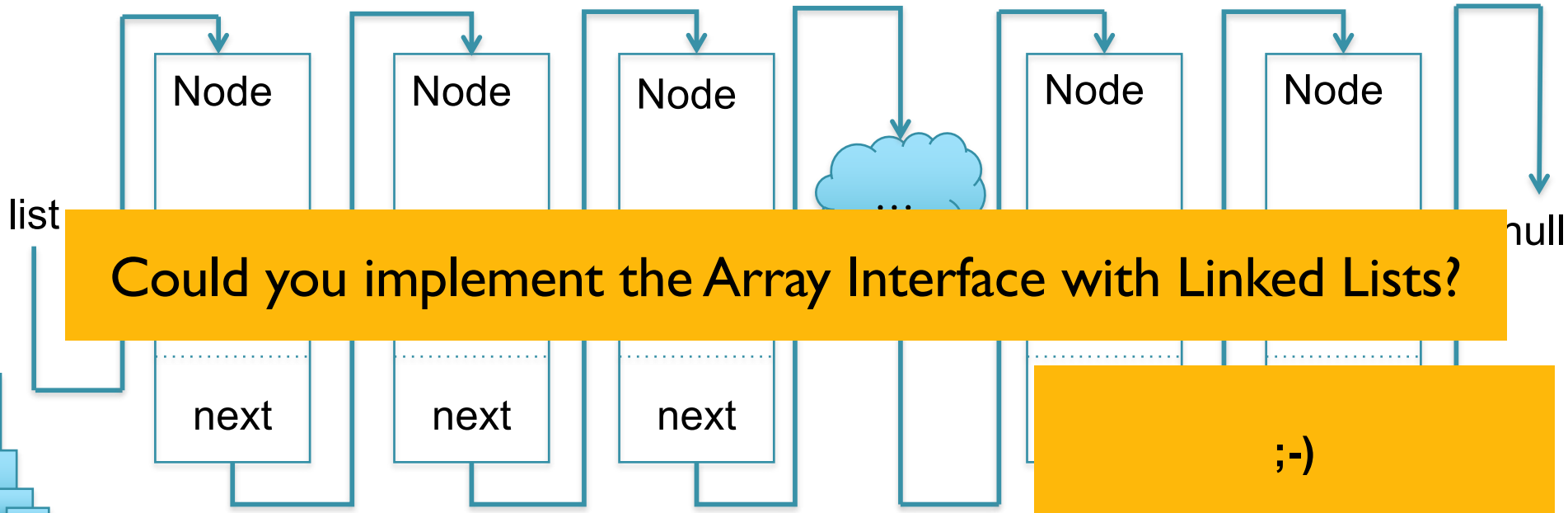
```
public interface Array<T> {
    void put(int i, T t) throws IndexException;
    T get(int i) throws IndexException;
    int length();
}
```

Arrays versus Linked Lists



Arrays versus Linked Lists

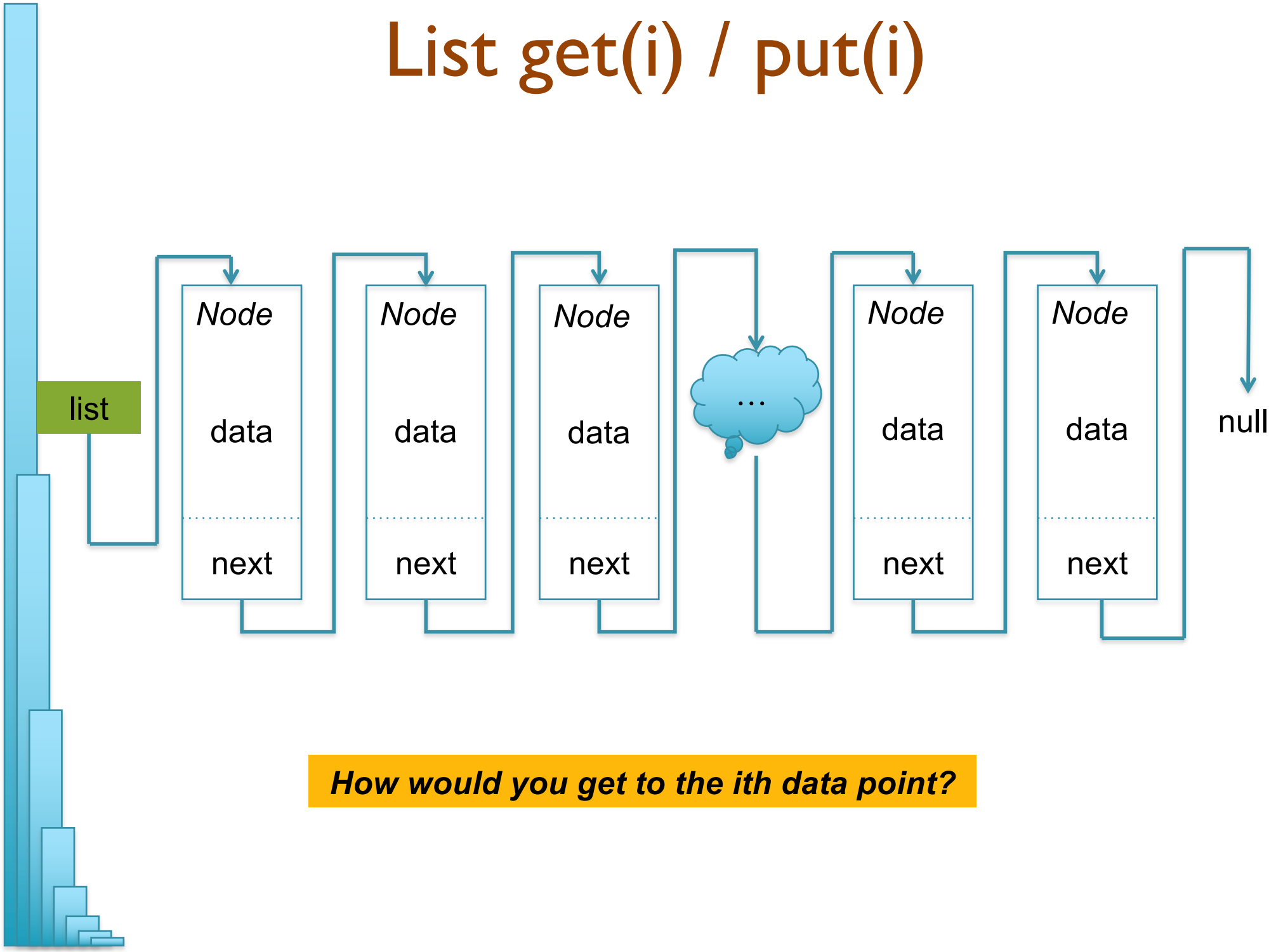




Could you implement the Array Interface with Linked Lists?

;-)

List get(i) / put(i)



How would you get to the ith data point?

List get(i) / put(i)

list

Node

data

next

Node

data

next

null

Nested Classes

The Java programming language allows you to define a class within another class – a nested class.

```
class OuterClass {
    ...
    class NestedClass {
        ...
    }
}
```

- ***It is a way of logically grouping classes that are only used in one place:*** If a class is useful to only one other class, then it is logical to embed it in that class and keep the two together. Nesting such "helper classes" makes their package more streamlined.

- ***It increases encapsulation:*** Consider two top-level classes, A and B, where B needs access to members of A that would otherwise be declared private. By hiding class B within class A, A's members can be declared private and B can access them. In addition, B itself can be hidden from the outside world.

- ***It can lead to more readable and maintainable code:*** Nesting small classes within top-level classes places the code closer to where it is used.

https://docs.oracle.com/javase/tutorial/java/javaOO/nested.html

Nested and Inner Classes

Nested classes are divided into two categories: static and non-static.

- Nested classes that are declared static are called ***static nested classes***.
- Non-static nested classes are called ***inner classes.***

```
class OuterClass {
    ...
    static class StaticNestedClass {
        ...
    }
    class InnerClass {
        ...
    }
}
```

A static nested class is associated with its outer class. And like static class methods, a static nested class cannot refer directly to instance variables or methods defined in its enclosing class: it can use them only through an object reference.

An inner class is associated with an instance of its enclosing class and has direct access to that object's methods and fields. Also, because an inner class is associated with an instance, it cannot define any static members itself.

ListArray (1)

ListArray.java

```
/**
    Array implementation using a linked list.
    @param <T> Element type.
*/
public class ListArray<T> implements Array<T> {
    // A nested Node<T> class to build our linked list out of. We use a
    // static nested class (instead of an inner class) here since we don't need
    // access to the ListArray object we're part of.
    private static class Node<T> {
        T data;
        Node<T> next;
    }

    // The not-so-obvious representation of our abstract Array: A linked
    // list with "length" nodes instead of an array of "length" slots.
    private Node<T> list;
    private int length;
```

Why length?

Static Nested Class

Private static class embedded inside a parent class

One less file to checkstyle ☺

No leakage of implementation details, also doesn't need getters & setters

ListArray (2)

```
/**
     Constructs a new ListArray.

     @param n Length of array, must be n &gt; 0.
     @param t Default value to store in each slot.
     @throws LengthException if n &le; 0.
 */
public ListArray(int n, T t) throws LengthException {
     if (n <= 0) {
         throw new LengthException();
     }

     // Initialize all positions as we promise in the specification.
     // Unlike in SimpleArray we cannot avoid the initialization even
     // if t == null since the nodes still have to be created. On the
     // upside we don't need a cast anywhere.
     for (int i = 0; i < n; i++) {
         this.prepend(t);
     }

     // Remember the length!
     this.length = n;
 }

 // Insert a node at the beginning of the linked list.
 private void prepend(T t) {
     Node<T> n = new Node<T>();
     n.data = t;
     n.next = this.list;
     this.list = n;
 }
```

Constructor
Make sure to follow the interface by initializing N empty nodes!

prepend()
Not in interface, but needed for implementation

ListArray (3)

```
    // Find the node for a given index. Assumes valid index.
    private Node<T> find(int index) {
        Node<T> n = this.list;
        int i = 0;
        while (n != null && i < index) {
            n = n.next;
            i = i + 1;
        }
        return n;
    }

    @Override
    public T get(int i) throws IndexException {
        if (i < 0 || i >= this.length) {
            throw new IndexException();
        }
        Node<T> n = this.find(i);
        return n.data;
    }

    @Override
    public void put(int i, T t) throws IndexException {
        if (i < 0 || i >= this.length) {
            throw new IndexException();
        }
        Node<T> n = this.find(i);
        n.data = t;
    }

    @Override
    public int length() {
        return this.length;

    }
}
```

find()
Get the i^{th} value in list
Be careful to not walk off the end of the list

get()
From the Array Interface
Return what you find()

put()
From the Array Interface
Update what you find()

length()
Why is this needed?

Printing the list

```
public static void main(String[] args) {
  ListArray<String> mylist = new ListArray(5, "Mike");

  System.out.println("Created array of length: " + mylist.length());
  for (int i = 0; i < mylist.length(); i++) {
      String val = mylist.get(i);
      System.out.println("mylist[" + i + "]: " + val);
  }

  mylist.put(3, "Peter");

  System.out.println("Printing Array after setting mylist[3]=Peter");

  for (int i = 0; i < mylist.length(); i++) {
      String val = mylist.get(i);
      System.out.println("mylist[" + i + "]: " + val);
  }
}
```

Printing the list

```
public static void main(String[] args) {
  ListArray<String> mylist = new ListArray(5, "Mike");

  System.out.println("Created array of length: " + mylist.length());
  for (int i = 0; i < mylist.length(); i++) {
      String val = mylist.get(i);
      System.out.println("mylist[" + i + "]: " + val);
  }

  mylist.put(3, "Peter");

  System.out.println("Printing Array after setting mylist[3]=Peter");

  for (int i = 0; i < mylist.leng
      String val = mylist.get(i);
      System.out.println("mylist[
  }
}
```

```
Created array of length: 5
mylist[0]: Mike
mylist[1]: Mike
mylist[2]: Mike
mylist[3]: Mike
mylist[4]: Mike
Printing Array after setting mylist[3]=Peter
mylist[0]: Mike
mylist[1]: Mike
mylist[2]: Mike
mylist[3]: Peter
mylist[4]: Mike
```

Java Iterators

http://docs.oracle.com/javase/6/docs/api/java/util/Iterator.html

java.util

Interface Iterator<E>

All Known Subinterfaces:
ListIterator<E>, XMLEventReader

All Known Implementing Classes:
BeanContextSupport.BCSIterator, EventReaderDelegate, Scanner

```
public interface Iterator<E>
```

An iterator over a collection. Iterator takes the place of Enumeration in the Java collections framework. Iterators differ from enumerations in two ways:

- Iterators allow the caller to remove elements from the underlying collection during the iteration with well-defined semantics.
- Method names have been improved.

This interface is a member of the Java Collections Framework.

Since:
1.2

See Also:
Collection, ListIterator, Enumeration

Method Summary	
boolean	**hasNext()** Returns `true` if the iteration has more elements.
E	**next()** Returns the next element in the iteration.
void	**remove()** Removes from the underlying collection the last element returned by the iterator (optional operation).

Java Iterators

http://docs.oracle.com/javase/6/docs/api/java/util/Iterator.html

java.util

Interface Iterator<E>

All Known Subinterfaces:

ListIterator<E>, XMLEventReader

All Known Implementing Classes:

BeanContextSupport.BCSIterator, EventReaderDelegate, Scanner

```
for (MyType obj : list) {
    System.out.print(obj);
}
```

ffer from

fined

Since:

1.2

See Also:

Collection, ListIterator, Enumeration

Method Summary	
boolean	**hasNext()** Returns `true` if the iteration has more elements.
E	**next()** Returns the next element in the iteration.
void	**remove()** Removes from the underlying collection the last element returned by the iterator (optional operation).

ListArray Iterator

Array.java

```
public interface Array<T> extends Iterable<T> {
```

ListArray.java

```
import java.util.Iterator;

    // An iterator to traverse the array from front to back.
    private class ArrayIterator implements Iterator<T> {
        // Current position in the linked list.
        Node<T> current;

        ArrayIterator() {
            this.current = ListArray.this.list;
        }

        public T next() {
            T t = this.current.data;
            this.current = this.current.next;
            return t;
        }

        public boolean hasNext() {
            return this.current != null;
        }

        public void remove() {
            throw new UnsupportedOperationException();
        }
    }

    public Iterator<T> iterator() {
        return new ArrayIterator();
    }
```

ListArray Iterator

Array.java

```
public interface Array<T> extends Iterable<T> {
```

ListArray.java

```
import java.util.Iterator;

    // An iterator to traverse the array from front to back.
    private class ArrayIterator implements Iterator<T> {
        // Current position in the linked list.
        Node<T> current;

        ArrayIterator() {
            this.current = ListArray.this.list;
        }

        public T next() {
            T t = this.current.data;
            this.current = this.current.next;
            return t;
        }

        public boolean hasNext() {
            return this.current != null;
        }

        public void remove() {
            throw new UnsupportedOperationException();
        }
    }

    public Iterator<T> iterator() {
        return new ArrayIterator();
    }
```

Extend Iterable<T>

Inner class

Maintains a reference to something in the parent class

Simplifies use by client: a regular nested class would need a reference to list for every method

ListArray Iterator

Array.java

```
public interface Array<T> extends Iterable<T> {
```

ListArray.java

```
import java.util.Iterator;

    // An iterator to traverse the array from front to back.
    private class ArrayIterator implements Iterator<T> {
        // Current position in the linked list.
        Node<T> current;

        ArrayIterator() {
            this.current = ListArray.this.list;
        }

        public T next() {
            T t = this.current.data;
            this.current = this.current.next;
            return t;
        }

        public boolean hasNext() {
            return this.current != null;
        }

        public void remove() {
            throw new UnsupportedOperationException();
        }
    }

    public Iterator<T> iterator() {
        return new ArrayIterator();
    }
```

Extend Iterable<T>

Implement Iterator<T>

Why current?

ListArray.this.what???

What happens if you call next() without checking hasNext()?

Required by interface

Lets the constructor access "this" list

Printing the list

```
public static void main(String[] args) {
  ListArray<String> mylist = new ListArray(5, "Mike");

  System.out.println("Created array of length: " + mylist.length());
  for (int i = 0; i < mylist.length(); i++) {
      String val = mylist.get(i);
      System.out.println("mylist[" + i + "]: " + val);
  }

  mylist.put(3, "Peter");

  System.out.println("Printing Array after setting mylist[3]=Peter");

  for (int i = 0; i < mylist.length(); i++) {
      String val = mylist.get(i);
      System.out.println("mylist[" + i + "]: " + val);
  }

   System.out.println("Printing Array with Iterator");
   Iterator<String> it = mylist.iterator();
   while (it.hasNext()) {
     String val = it.next();
     System.out.println(val);
   }
}
```

Notice the iterator must be generated from an existing List object (non-static method)

Printing the list

```
public static void main(String[] args) {
  ListArray<String> mylist = new ListArray(5, "Mike");

  System.out.println("Created array of length: " + mylist.length());
  for (int i = 0; i < mylist.length(); i++) {
      String val = mylist.get(i);
      System.out.println("mylist[" + i + "]: " + val);
  }

  mylist.put(3, "Peter");

  System.out.println("Printing Array after setting mylist[3]=Peter");

  for (int i = 0; i < mylist.length(); i++) {
      String val = mylist.get(i);
      System.out.println("mylist[" + i + "]: " + val);
  }

   System.out.println("Printing Array with Iterator");
   Iterator<String> it = mylist.iterator();
   while (it.hasNext()) {
     String val = it.next();
     System.out.println(val);
   }
}
```

Why don't we print the index?

How will this affect performance compared to get()ing each element?

```
…
Printing Array with Iterator
Mike
Mike
Mike
Peter
Mike
```

Printing the list

```
        ...

        System.out.println("Printing Array with Iterator");
        Iterator<String> it = mylist.iterator();
        while (it.hasNext()) {
           String val = it.next();
           System.out.println(val);
        }

        System.out.println("Printing Array with Iterator (using true)");
        it = mylist.iterator();
        while (true) {
          String val = it.next();
          System.out.println(val);
        }
    }
```

```
…
Printing Array with Iterator (using true)
Mike
Mike
Mike
Peter
Mike
Exception in thread "main" java.lang.NullPointerException
        at ListArray$ArrayIterator.next(ListArray.java:39)
        at ListArray.main(ListArray.java:169)
```

Printing the list

```
        ...

        System.out.println("Printing Array after setting mylist[3]=Peter");

        for (int i = 0; i < mylist.length(); i++) {
            String val = mylist.get(i);
            System.out.println("mylist[" + i + "]: " + val);
        }

        System.out.println("Printing Array with Iterator");
        Iterator<String> it = mylist.iterator();
        while (it.hasNext()) {
           String val = it.next();
           System.out.println(val);
        }

        System.out.println("Printing Array with foreach");
        for (String val : mylist) {
            System.out.println(val);
        }
    }
```

Is this better than explicitly using the iterator?

```
…
Printing Array with foreach
Mike
Mike
Mike
Peter
Mike
```

SimpleArray

```
public class SimpleArray<T> implements Array<T> {
    // The obvious representation of our abstract Array.
    private T[] data;

    // An iterator to traverse the array from front to back.
    // (So we can use Array with for-each loops just like we
    // could basic Java arrays.) Using an inner class (not a
    // nested class) allows us to access the outer object's
    // "data" member without any stunts.
    private class ArrayIterator implements Iterator<T> {
        // Current position in the basic Java array.
        int current;

        @Override
        public T next() {
            T t = SimpleArray.this.data[this.current];
            this.current += 1;
            return t;
        }

        @Override
        public boolean hasNext() {
            return this.current < SimpleArray.this.data.length;
        }

        @Override
        public void remove() {
            throw new UnsupportedOperationException();
        }
    }

    @Override
    public Iterator<T> iterator() {
        return new ArrayIterator();
    }
```

Use an array to store the data

What differences do you see compared to ListArray.ArrayIterator?

Next Steps

1. Work on HW1

2. Check on Piazza for tips & corrections!