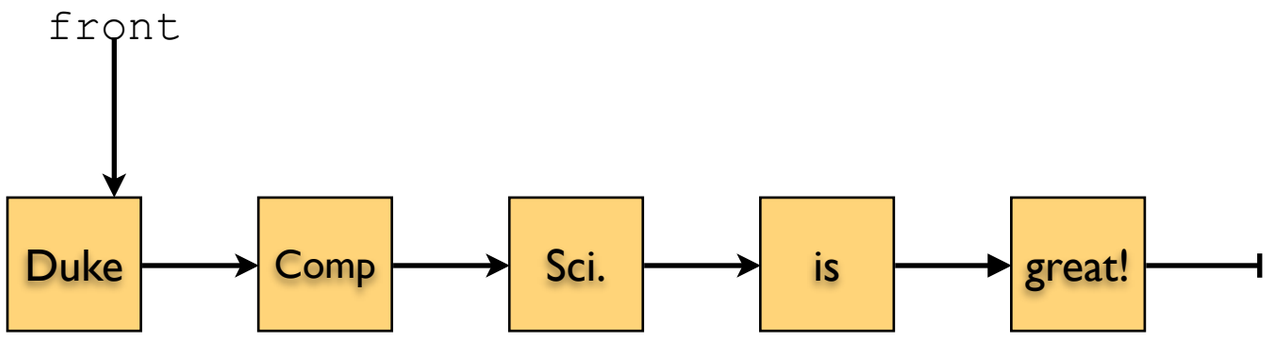
A Smorgasbord:

(and some review)

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9



(Thanks for the picture, Wikipedia.)

A review: .equals()

Asks: "Do these two objects have the same value?"

Ron Wallace may have just set the world pumpkin record.

```
public class Pumpkin {
    private int myMass;
    private String myGrowerName;

    public Pumpkin(int mass, String grower) {
        myMass = mass;
        myGrowerName = grower;
    }
}
```

http://www.seattlepi.com/news/article/RI-man-wins-15-500-with-1-ton-pumpkin-3906593.php

A review: .equals()

Asks: "Do these two objects have the same value?"

Ron Wallace may have just set the world pumpkin record.

He would like to prove that his pumpkin has no equal.

```
public class Pumpkin {
    private int myMass;
    private String myGrowerName;

    public Pumpkin(int mass, String grower) {
        myMass = mass;
        myGrowerName = grower;
    }

    public boolean equals(Object other) {

        }
    }
```

.equals takes an Object as its argument for historical reasons. We'd rather it didn't.

A review: .equals()

Asks: "Do these two objects have the same value?"

Ron Wallace may have just set the world pumpkin record.

He would like to prove that his pumpkin has no equal.

```
public class Pumpkin {
    private int myMass;
    private String myGrowerName;

    public Pumpkin(int mass, String grower) {
        myMass = mass;
        myGrowerName = grower;
    }

    public boolean equals(Object other) {

        }
    }
```

http://goo.gl/v9zat

.equals takes an Object as its argument for historical reasons. We'd rather it didn't.

A review: .equals()

Asks: "Do these two objects have the same value?"

Ron Wallace may have just set the world pumpkin record.

He would like to prove that his pumpkin has no equal.

```
public class Pumpkin {
    private int myMass;
    private String myGrowerName;

    public Pumpkin(int mass, String grower) {
        myMass = mass;
        myGrowerName = grower;
    }

    public boolean equals(Object other) {
        if (other == null) {
            return false;
        }
        if (other.getClass() != getClass()) {
            return false;
        }
        Pumpkin p = (Pumpkin)other;
        return myMass == p.myMass &&
           myGrowerName.equals(p.myGrowerName);
    }
}
```

Common case: check *all* of the instance variables.

.equals takes an Object as its argument for historical reasons. We'd rather it didn't.

A review: .compareTo()

Asks: "Which one of these objects is 'bigger' than the other?"

Ron Wallace may have just set the world pumpkin record.

He has proven that his pumpkin has no equal. ✔

Now he wants to prove that his pumpkin is the best!

```
public class Pumpkin implements Comparable<Pumpkin>{
    private int myMass;
    private String myGrowerName;

    public Pumpkin(int mass, String grower) {
        myMass = mass;
        myGrowerName = grower;
    }
    // equals hidden for space's sake.
    public int compareTo(Pumpkin other) {
```

http://goo.gl/gKr0X

```
        }
    }
```

Note "implements Comparable<Pumpkin>". This is why we don't have to send .compareTo an Object; we wish .equals did this...

A review: .compareTo()

Asks: "Which one of these objects is 'bigger' than the other?"

Ron Wallace may have just set the world pumpkin record.

He has proven that his pumpkin has no equal. ✔

Now he wants to prove that his pumpkin is the best!

```
public class Pumpkin implements Comparable<Pumpkin>{
    private int myMass;
    private String myGrowerName;

    public Pumpkin(int mass, String grower) {
        myMass = mass;
        myGrowerName = grower;
    }
    // equals hidden for space's sake.
    public int compareTo(Pumpkin other) {
      if (myMass < other.myMass) {
          return -1;
      }
      if (myMass > other.myMass) {
          return 1;
      }
      return myGrowerName.compareTo(
          other.myGrowerName);
    }
}
```

Now we can use Arrays.sort on arrays of Pumpkins, Collections.sort on Lists of Pumpkins, and use Pumpkins in TreeSet and TreeMap.

Note "implements Comparable<Pumpkin>". This is why we don't have to send .compareTo an Object; we wish .equals did this...

A review: .compareTo()

Asks: "Which one of these objects is 'bigger' than the other?"

Ron Wallace may have just set the world pumpkin record.

He has proven that his pumpkin has no equal. ✔

He has proven that his pumpkin is the best! ✔

```
public class Pumpkin implements Comparable<Pumpkin>{
    private int myMass;
    private String myGrowerName;

    public Pumpkin(int mass, String grower) {
        myMass = mass;
        myGrowerName = grower;
    }
    // equals hidden for space's sake.
    public int compareTo(Pumpkin other) {
      if (myMass < other.myMass) {
          return -1;
      }
      if (myMass > other.myMass) {
          return 1;
      }
      return myGrowerName.compareTo(
          other.myGrowerName);
    }
}
```

Now we can use Arrays.sort on arrays of Pumpkins, Collections.sort on Lists of Pumpkins, and use Pumpkins in TreeSet and TreeMap.

Note "implements Comparable<Pumpkin>". This is why we don't have to send .compareTo an Object; we wish .equals did this...

A review: .hashCode()

Less obvious. Turns a your object into an integer.

Ron Wallace may have just set the world pumpkin record.

He has proven that his pumpkin has no equal. ✔

He has proven that his pumpkin is the best! ✔

Now he wants to hash his pumpkin.
(Nothing to do with breakfast foods)

```
public class Pumpkin implements Comparable<Pumpkin>{
    private int myMass;
    private String myGrowerName;

    public Pumpkin(int mass, String grower) {
        myMass = mass;
        myGrowerName = grower;
    }
    // equals & compareTo hidden for space's sake.
    public int hashCode() {
```

```
    }
}
```

A review: .hashCode()

Less obvious. Turns a your object into an integer.

Ron Wallace may have just set the world pumpkin record.

He has proven that his pumpkin has no equal. ✔

He has proven that his pumpkin is the best! ✔

Now he wants to hash his pumpkin.
(Nothing to do with breakfast foods)

```
public class Pumpkin implements Comparable<Pumpkin>{
    private int myMass;
    private String myGrowerName;

    public Pumpkin(int mass, String grower) {
        myMass = mass;
        myGrowerName = grower;
    }
    // equals & compareTo hidden for space's sake.
    public int hashCode() {
```

But wait!

```
    }
}
```

A review: .hashCode()

Less obvious. Turns a your object into an integer.

Ron Wallace may have just set the world pumpkin record.

He has proven that his pumpkin has no equal. ✔

He has proven that his pumpkin is the best! ✔

Now he wants to hash his pumpkin.
(Nothing to do with breakfast foods)

```
public class Pumpkin implements Comparable<Pumpkin>{
    private int myMass;
    private String myGrowerName;

    public Pumpkin(int mass, String grower) {
        myMass = mass;
        myGrowerName = grower;
    }
    // equals & compareTo hidden for space's sake.
    public int hashCode() {

    }
}
```

hashCode rules:
1. Depends only on the instance variables.
2. If a.equals(b), then a.hashCode() == b.hashCode().
3. If !a.equals(b), then a.hashCode *might* == b.hashCode().

Very important.

4. All built-in Object types have a .hashCode. It's a handy building block for your own hashCodes.
5. If !a.equals(b), a.hashCode() should try not to == b.hashCode()

HashSet & HashMap will be faster if it isn't. More details later!

A review: .hashCode()

Less obvious. Turns a your object into an integer.

Ron Wallace may have just set the world pumpkin record.

He has proven that his pumpkin has no equal. ✔

He has proven that his pumpkin is the best! ✔

Now he wants to hash his pumpkin.
(Nothing to do with breakfast foods)

```
public class Pumpkin implements Comparable<Pumpkin>{
    private int myMass;
    private String myGrowerName;

    public Pumpkin(int mass, String grower) {
        myMass = mass;
        myGrowerName = grower;
    }
    // equals & compareTo hidden for space's sake.
    public int hashCode() {
```

http://goo.gl/xQbJJ

```
    }
}
```

A review: .hashCode()

Less obvious. Turns a your object into an integer.

Ron Wallace may have just set the world pumpkin record.

He has proven that his pumpkin has no equal. ✔

He has proven that his pumpkin is the best! ✔

Now he wants to hash his pumpkin.
(Nothing to do with breakfast foods)

```
public class Pumpkin implements Comparable<Pumpkin>{
    private int myMass;
    private String myGrowerName;

    public Pumpkin(int mass, String grower) {
        myMass = mass;
        myGrowerName = grower;
    }
    // equals & compareTo hidden for space's sake.
    public int hashCode() {
        return myGrowerName.hashCode() +
                new Integer(myMass).hashCode();
    }
}
```

Rely on rule #4

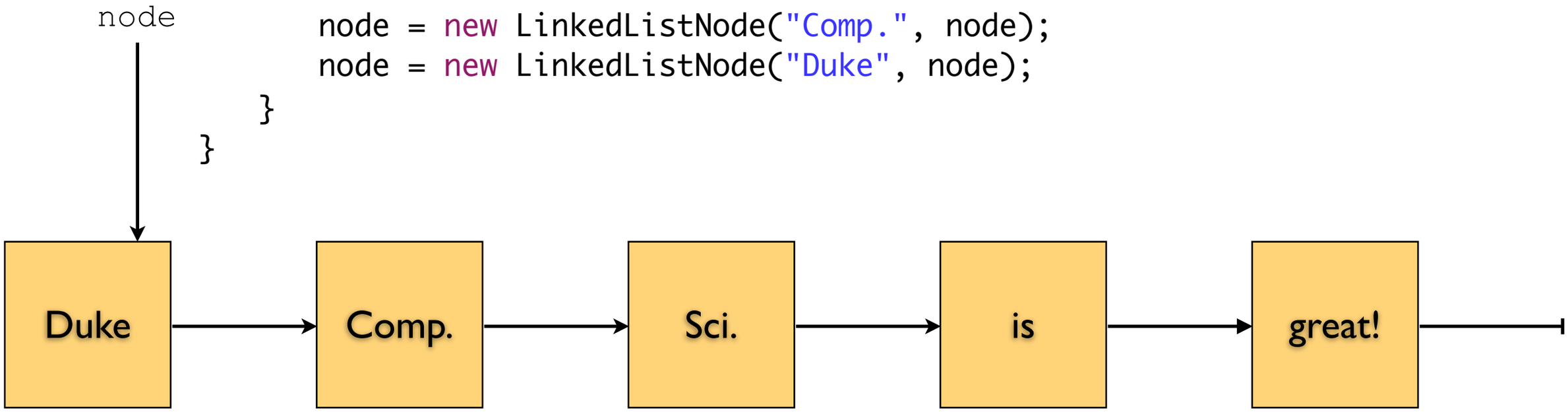
hashCode rules:

1. Depends only on the instance variables.
2. If a.equals(b), then a.hashCode() == b.hashCode().
3. If !a.equals(b), then a.hashCode *might* == b.hashCode().
4. All built-in Object types have a .hashCode. It's a handy building block for your own hashCodes.
5. If !a.equals(b), a.hashCode() should try not to == b.hashCode()

In sum:

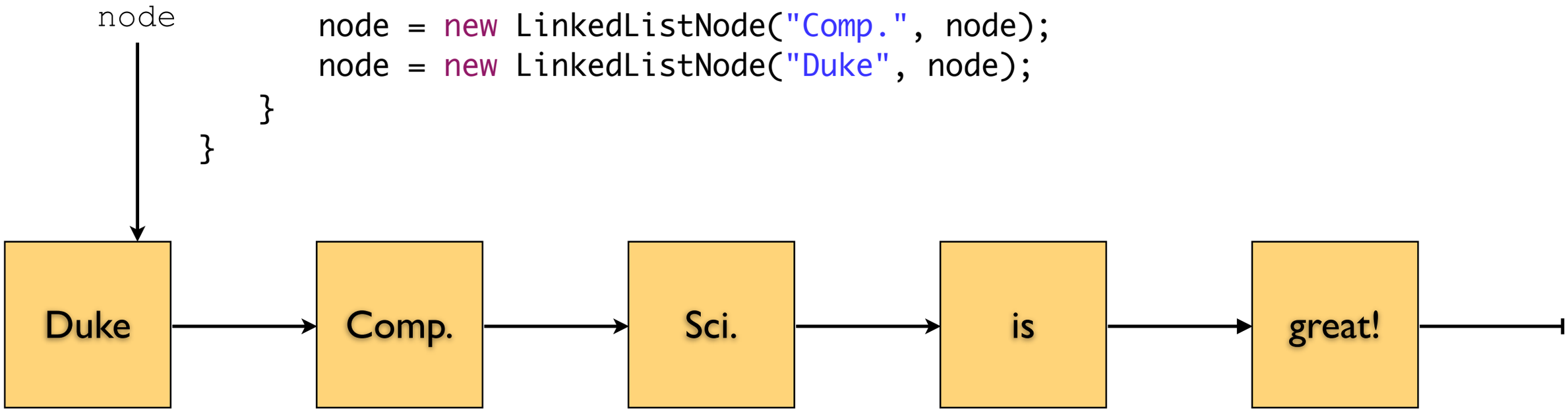
```
public class Pumpkin implements Comparable<Pumpkin>{
    private int myMass;
    private String myGrowerName;
    private int myHashCode;

    public Pumpkin(int mass, String grower) {
        myMass = mass;
        myGrowerName = grower;
        computeHashCode();
    }

    public boolean equals(Object other) {
        if (other == null) {
            return false;
        }
        if (other.getClass() != getClass()) {
            return false;
        }
        Pumpkin p = (Pumpkin)other;
        return myMass == p.myMass &&
               myGrowerName.equals(p.myGrowerName);
    }

    public int compareTo(Pumpkin other) {
        if (myMass < other.myMass) {
            return -1;
        }
        if (myMass > other.myMass) {
            return 1;
        }
        return myGrowerName.compareTo(
                other.myGrowerName);
    }

    private void computeHashCode() {
        myHashCode = myGrowerName.hashCode() +
                     new Integer(myMass).hashCode();
    }

    public int hashCode() {
        return myHashCode;
    }
}
```

```java
public class LinkedListDemo {
    public static void main(String[] args) {
        LinkedListNode node = new LinkedListNode("great!", null);
        node = new LinkedListNode("is", node);
        node = new LinkedListNode("Sci.", node);
        node = new LinkedListNode("Comp.", node);
        node = new LinkedListNode("Duke", node);
    }
}
```




```java
public class LinkedListNode {
    private String myString;
    private LinkedListNode myNext;
    public LinkedListNode(String s,
                          LinkedListNode n) {
        myString = s;
        myNext = n;
    }
    public String getString() {
        return myString;
    }
    public LinkedListNode getNext() {
        return myNext;
    }
}
```

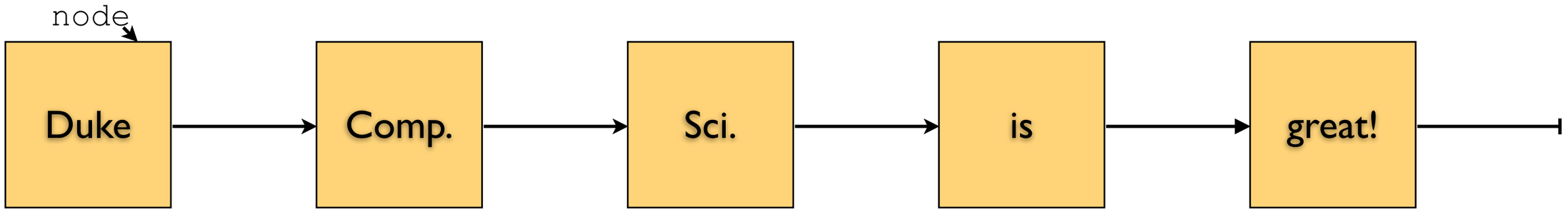
```
public class LinkedListDemo {
    public static void main(String[] args) {
        LinkedListNode node = new LinkedListNode("great!", null);
        node = new LinkedListNode("is", node);
        node = new LinkedListNode("Sci.", node);
        node = new LinkedListNode("Comp.", node);
        node = new LinkedListNode("Duke", node);
    }
}
```



Recursive data!

...suggests recursive algorithms.

```
public class LinkedListNode {
    private String myString;
    private LinkedListNode myNext;
    public LinkedListNode(String s,
                          LinkedListNode n) {
        myString = s;
        myNext = n;
    }
    public String getString() {
        return myString;
    }
    public LinkedListNode getNext() {
        return myNext;
    }
}
```




```
void printList(LinkedListNode node) {
```

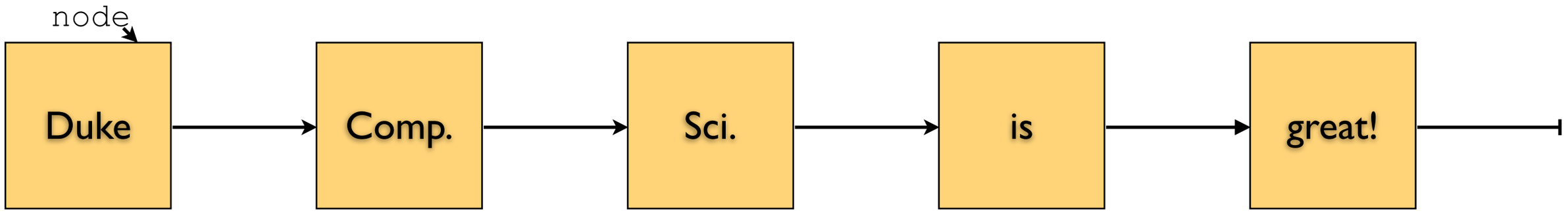
Base Case

Deal with this node

Deal with the rest of the list

```
}
```

```
public class LinkedListNode {
    private String myString;
    private LinkedListNode myNext;
    public LinkedListNode(String s,
                          LinkedListNode n) {
        myString = s;
        myNext = n;
    }
    public String getString() {
        return myString;
    }
    public LinkedListNode getNext() {
        return myNext;
    }
}
```



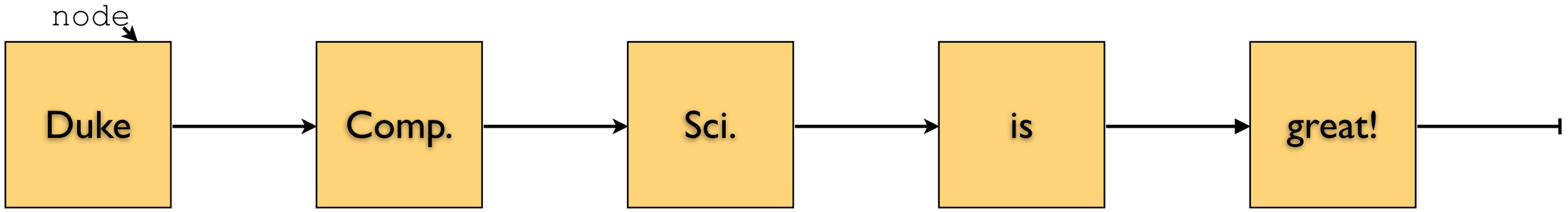

```
void printList(LinkedListNode node) {
   if (node == null) {
      return;
   }
```

Deal with this node

Deal with the rest of the list

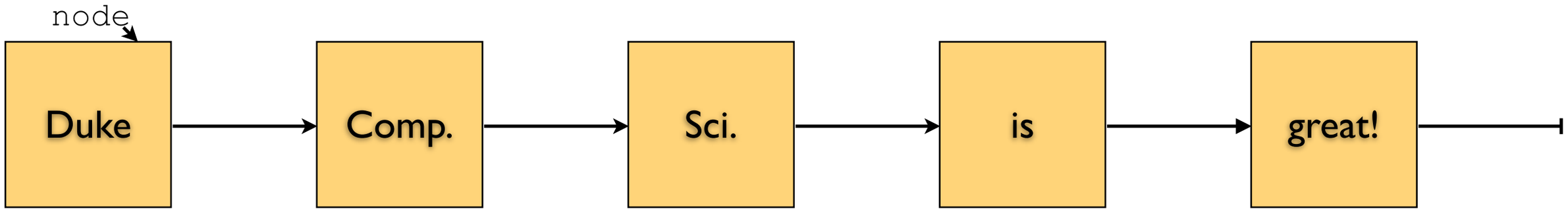
```
}
```

```
public class LinkedListNode {
    private String myString;
    private LinkedListNode myNext;
    public LinkedListNode(String s,
                          LinkedListNode n) {
        myString = s;
        myNext = n;
    }
    public String getString() {
        return myString;
    }
    public LinkedListNode getNext() {
        return myNext;
    }
}
```



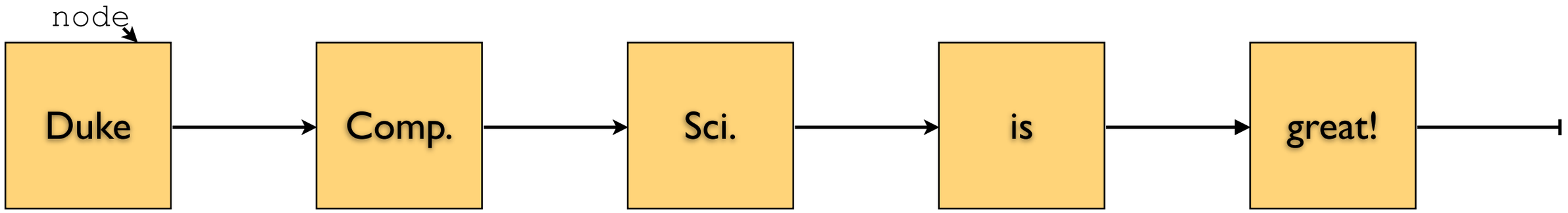

```
void printList(LinkedListNode node) {
   if (node == null) {
      return;
   }

   System.out.println(node.getString());
   Deal with the rest of the list
}
```

```
public class LinkedListNode {
    private String myString;
    private LinkedListNode myNext;
    public LinkedListNode(String s,
                          LinkedListNode n) {
        myString = s;
        myNext = n;
    }
    public String getString() {
        return myString;
    }
    public LinkedListNode getNext() {
        return myNext;
    }
}
```




```
void printList(LinkedListNode node) {
   if (node == null) {
      return;
   }

   System.out.println(node.getString());
   printList(node.getNext());
}
```

```
public class LinkedListNode {
    private String myString;
    private LinkedListNode myNext;
    public LinkedListNode(String s,
                          LinkedListNode n) {
        myString = s;
        myNext = n;
    }
    public String getString() {
        return myString;
    }
    public LinkedListNode getNext() {
        return myNext;
    }
}
```

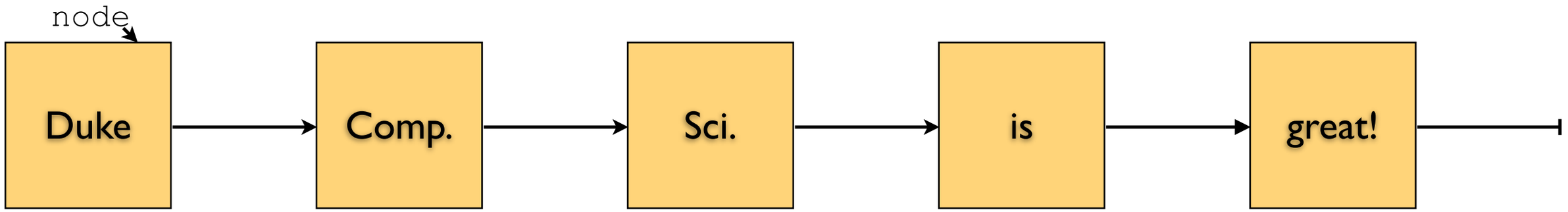



```
void printList(LinkedListNode node) {
   if (node == null) {
      return;
   }

   System.out.println(node.getString());
   printList(node.getNext());
}
```

Snarf LinkDemo

http://goo.gl/6q1el

```
public class LinkedListNode {
    private String myString;
    private LinkedListNode myNext;
    public LinkedListNode(String s,
                          LinkedListNode n) {
        myString = s;
        myNext = n;
    }
    public String getString() {
        return myString;
    }
    public LinkedListNode getNext() {
        return myNext;
    }
}
```




```
void printList(LinkedListNode node) {
   if (node == null) {
      return;
   }

   System.out.println(node.getString());
   printList(node.getNext());
}
```

Snarf LinkDemo

http://goo.gl/6q1el

http://goo.gl/GNDhP

```
public class LinkedListNode {
    private String myString;
    private LinkedListNode myNext;
    public LinkedListNode(String s,
                          LinkedListNode n) {
        myString = s;
        myNext = n;
    }
    public String getString() {
        return myString;
    }
    public LinkedListNode getNext() {
        return myNext;
    }
}
```

What values can we put in this square?

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Snarf Sudoku

Implement recursiveHelper (in Sudoku)

http://goo.gl/A70fL