Unit 05: Recursion

Anthony Estey
CSC 115: Fundamentals of Programming II
University of Victoria

Unit 05 Overview

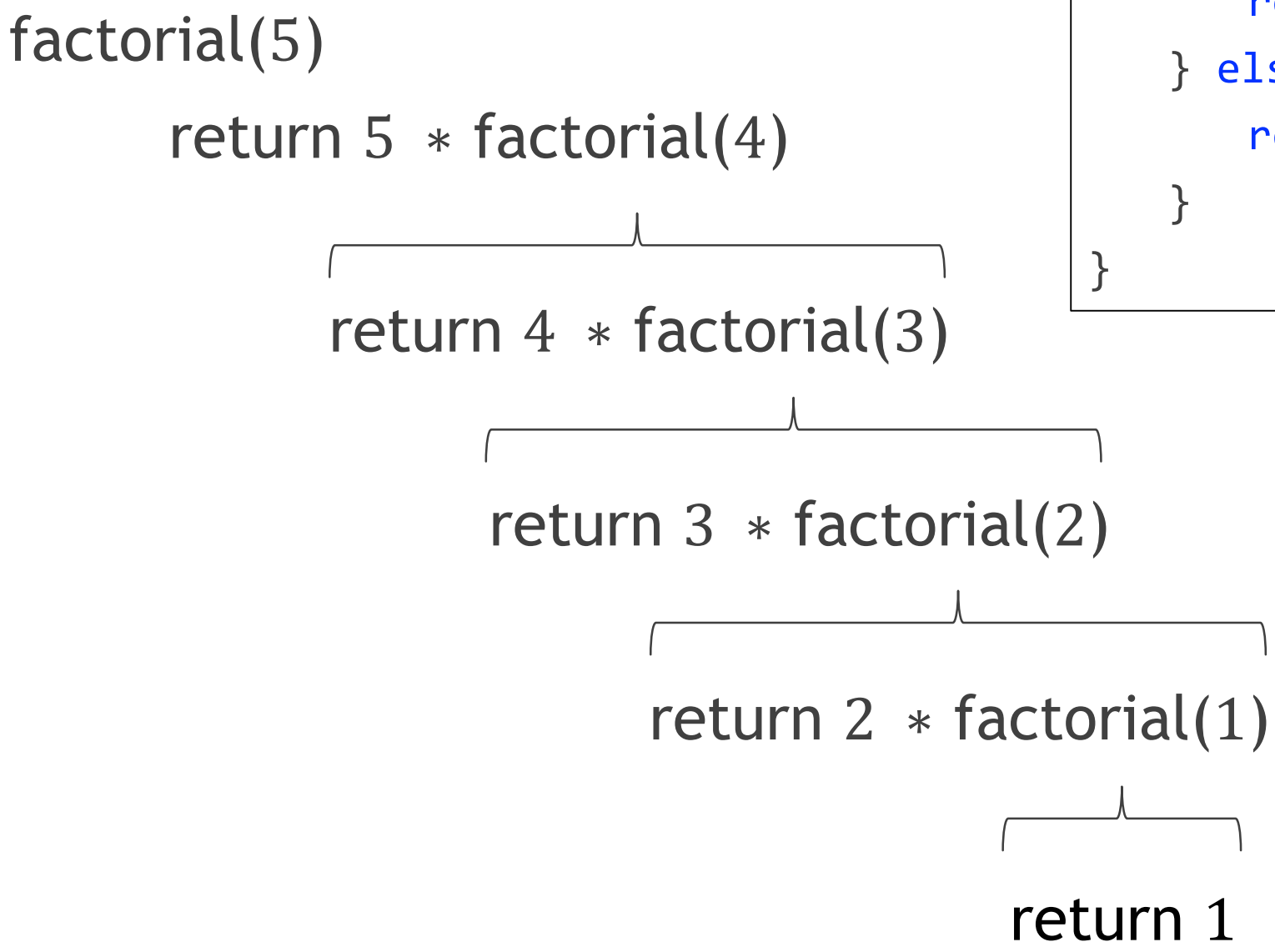
- Supplemental Reading:
  - Textbook: Chapter 3
- Learning Objectives: (You should be able to…)
  - describe how recursion can be used to solve problems
  - describe the three important properties of a recursive algorithm
  - translate a solution to a problem implemented with a for or while loop into a recursive implementation
  - solve problems using a recursive approach
  - read and write recursive Java code

What is recursion?

- A divide-and-conquer approach to solving problems, where the solution depends on solutions to smaller instances of the *same* problem
- A recursive solution to a problem by using methods that call themselves within their own code
- Recursion is one of the central ideas of computer science; we will see a lot of recursion solutions as we explore different algorithms and data structures throughout this course

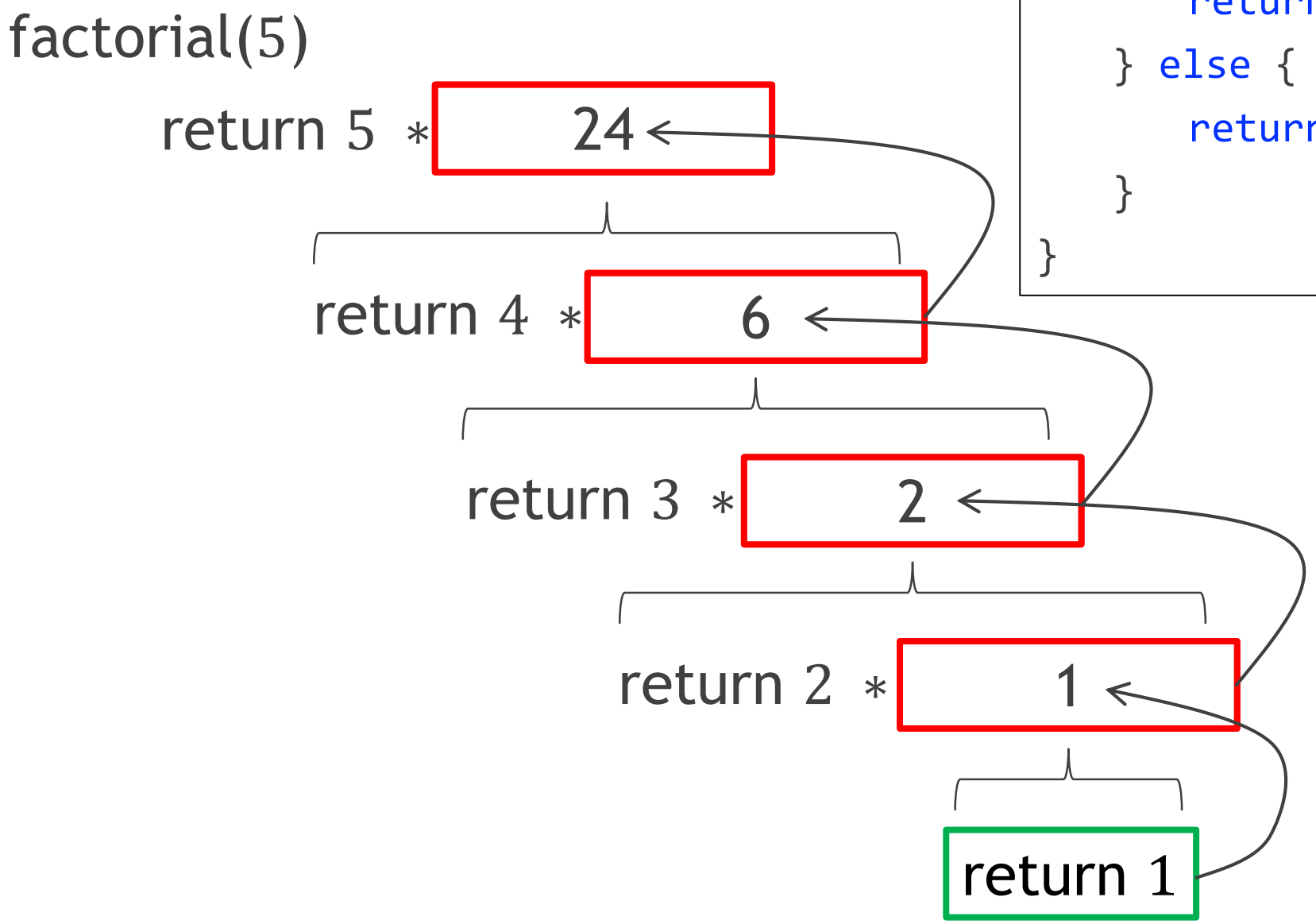
Example

```
public static int factorial(int n) {
    if (n <= 1) {
        return 1;
    } else {
        return n * factorial(n-1);
    }
}
```



Example

```
public static int factorial(int n) {
    if (n <= 1) {
        return 1;
    } else {
        return n * factorial(n-1);
    }
}
```



Example

- This function computes $n!$
  - Pronounced “n-factorial”

```
public static int factorial(int n) {
    if (n <= 1) {
        return 1;
    } else {
        return n * factorial(n-1);
    }
}
```

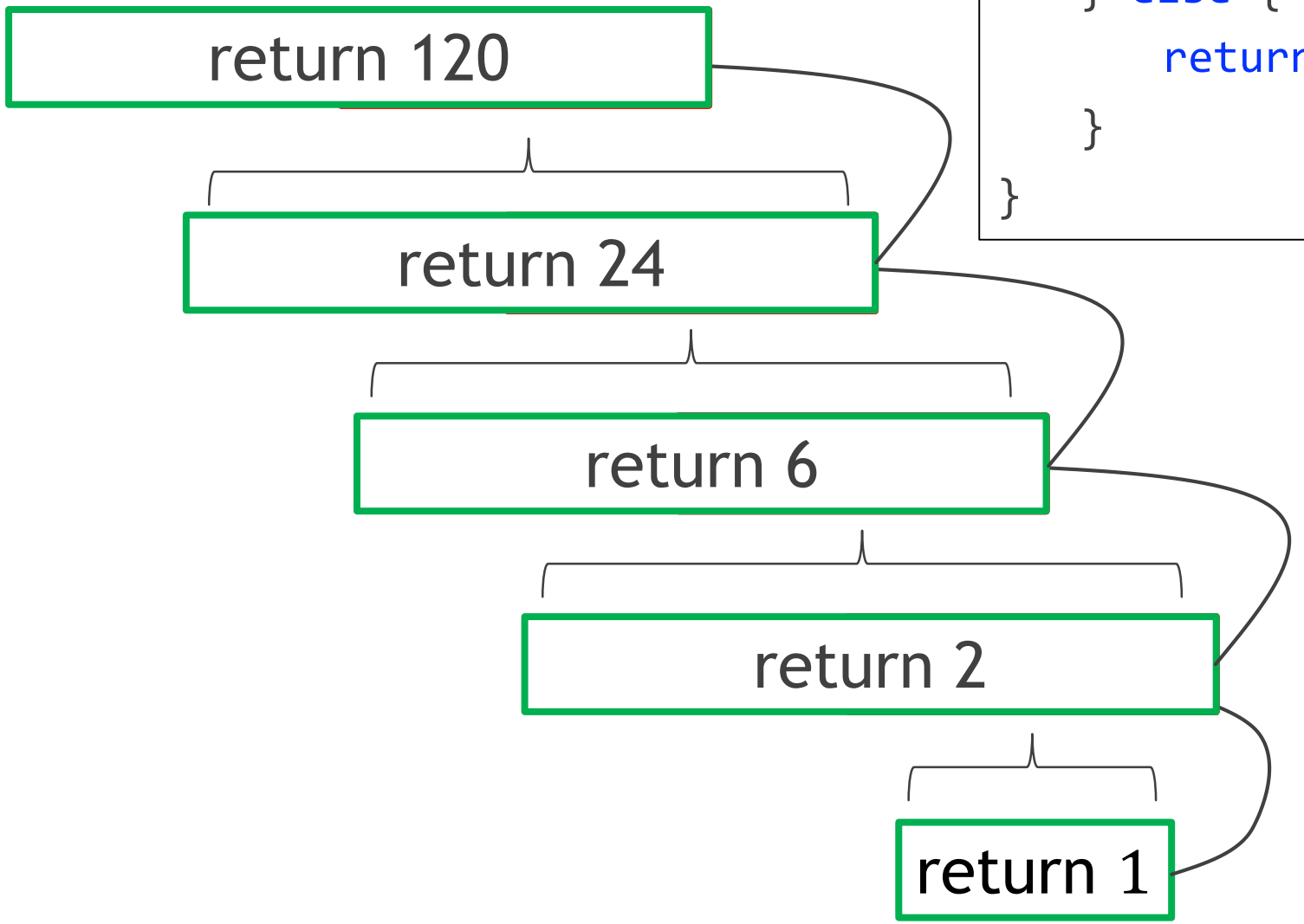
- What is $n!$?
  - $n * (n-1) * (n-2) * \cdots * 3 * 2 * 1$
  - So $4! = 4 * 3 * 2 * 1 = 24$
  - The special case is that $0! = 1$

- At first glance, maybe there isn’t anything special about this method
  - a **recursive call** is when a method calls itself

Example

```
public static int factorial(int n) {
    if (n <= 1) {
        return 1;
    } else {
        return n * factorial(n-1);
    }
}
```

factorial(5)



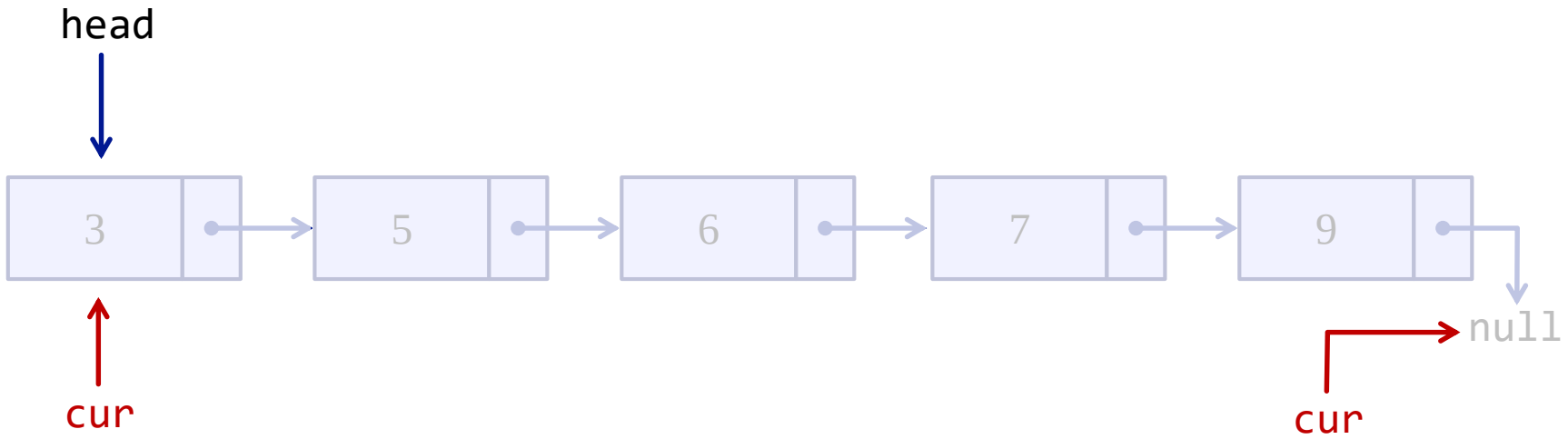
Rules of Recursion

```
public static int factorial(int n) {
    if (n <= 1) {
        return 1;
    } else {
        return n * factorial(n-1);
    }
}
```

- Three important properties of a recursive algorithm:

1. The recursive algorithm must call itself (called a *recursive call*)
2. The recursive algorithm must have a base case
3. The recursive calls must converge to the base case

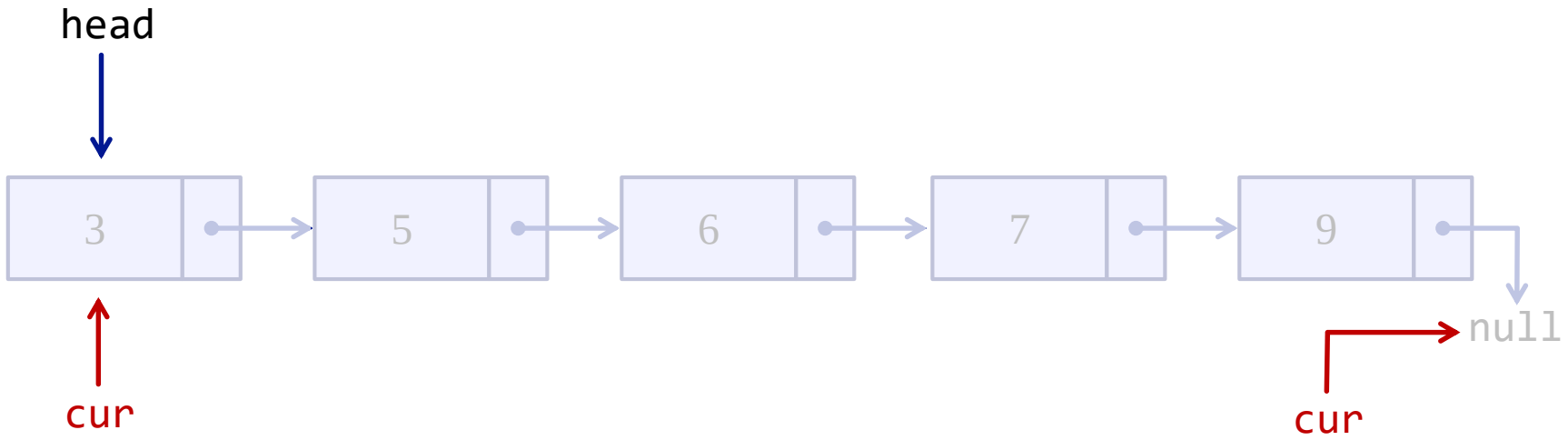
Printing a list: loops




```
Node cur = head;
while (cur != null) {
      System.out.print(cur.data));
      cur = cur.next;
}
```

Output: 3 5 6 7 9

Printing a list: recursion

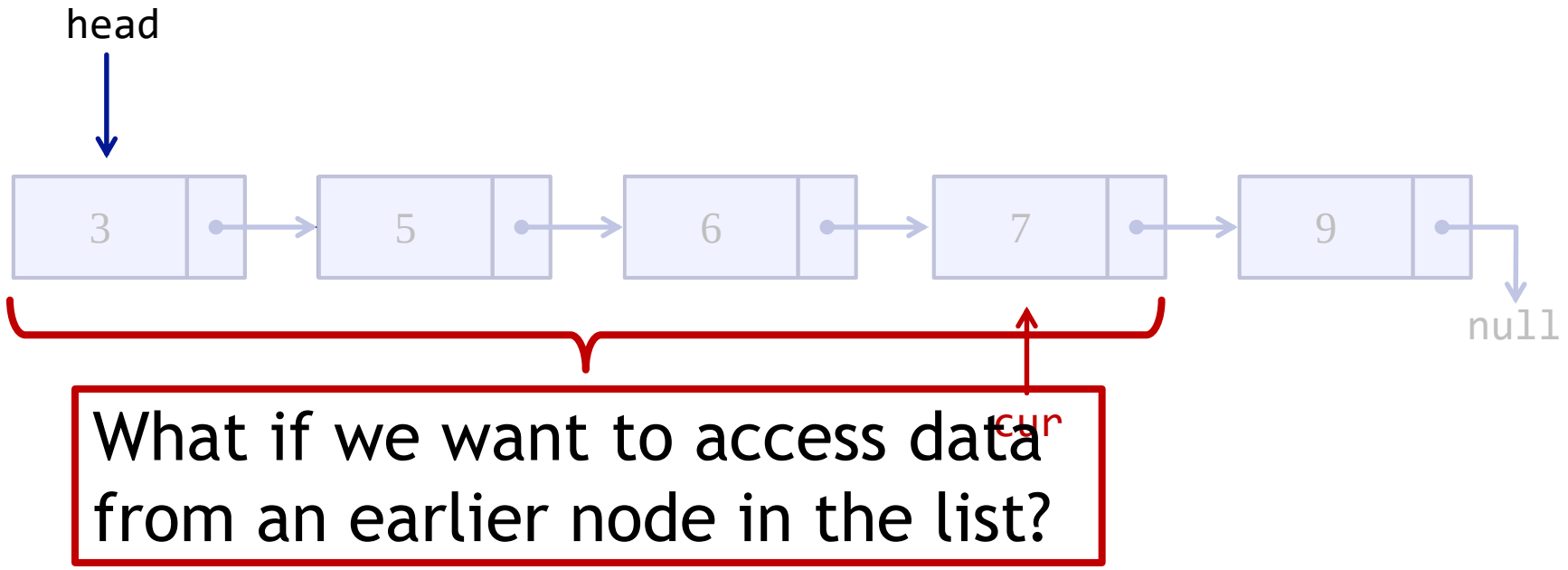



```
public void printList(Node cur) {
   if(cur == null) {
      return;
   }
   System.out.print(cur.data));
   printList(cur.next)
}
```

```
printList(head);
```

Output: 3 5 6 7 9

Context-preserving accumulator




```
Node cur = head;
while (cur != null) {
    System.out.print(cur.data));
    cur = cur.next;
}
```

Output: 3 5 6 7

Context-preserving accumulator

- So far, we have used accumulators to accumulate a result as we iterate through all of the elements in a collection
  - For example: sum up all of the values in a list
    - initialize sum to 0
    - as we visit each element in the list, add its data value to sum
    - return sum at the end

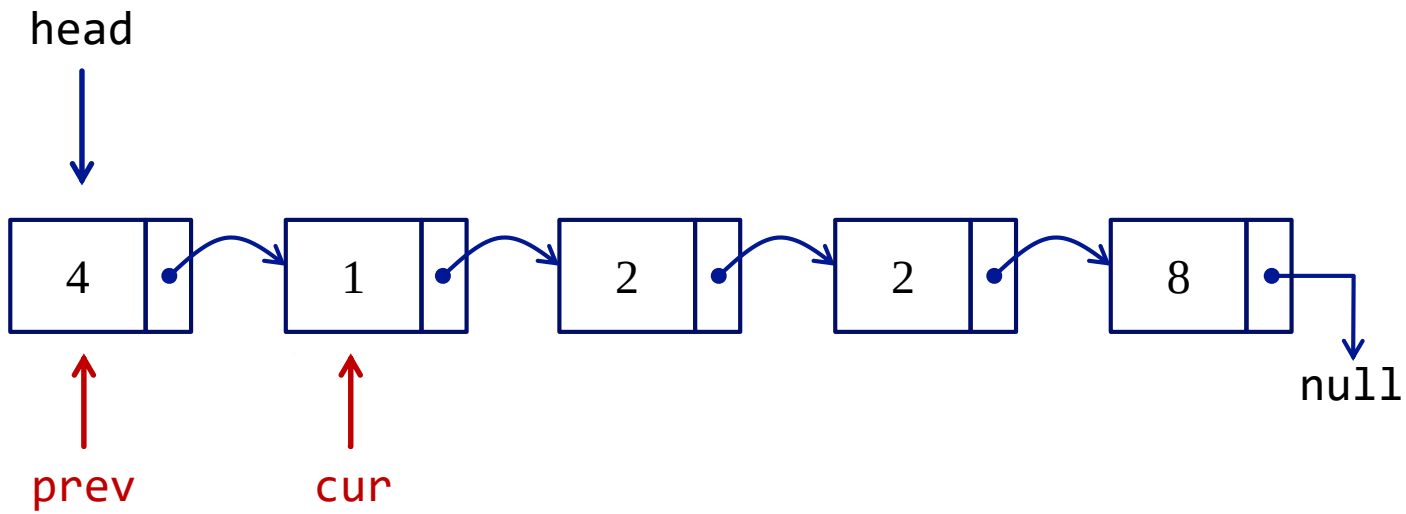
- We can also use accumulators to hold information for us that we might need later, but may not have direct access to
  - For example: comparing all elements to the first element, or to the element that comes before them

Context-preserving accumulator example

```
public boolean twoInARow() {

  if (this.size() <= 1) {
    return false;
  }

  Node prev = head;
  Node cur = head.next;
  while (cur != null) {
    if (prev.value==cur.value) {
      return true;
    }
    prev = cur;
    cur = cur.next;
  }
  return false;
}
```

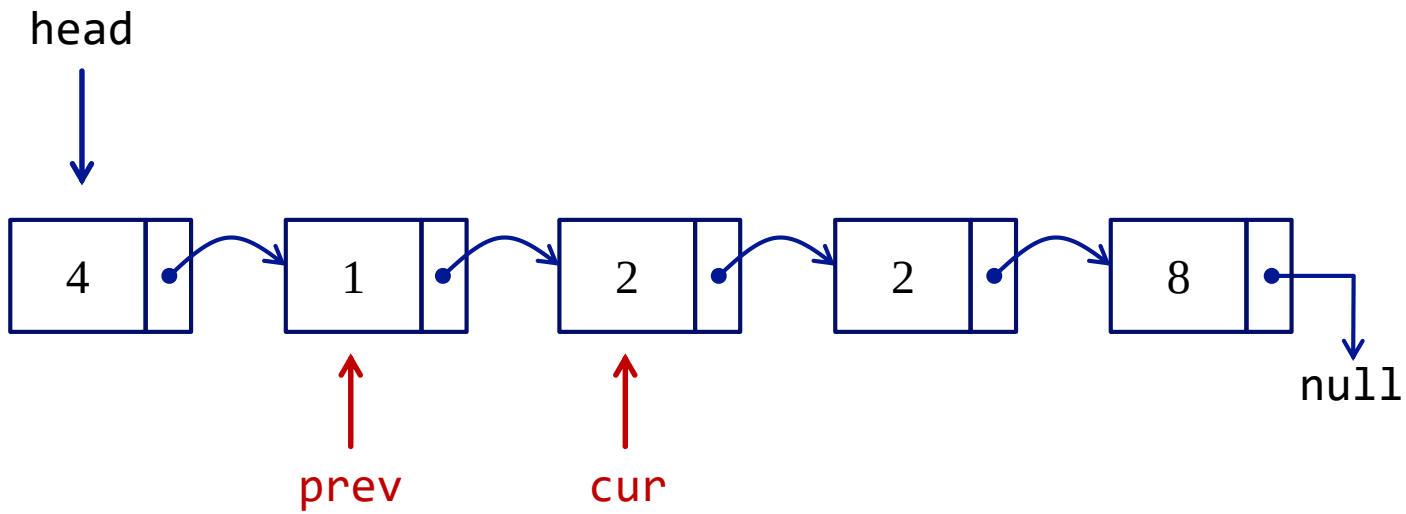


Context-preserving accumulator example

```
public boolean twoInARow() {

  if (this.size() <= 1) {
    return false;
  }

  Node prev = head;
  Node cur = head.next;
  while (cur != null) {
    if (prev.value==cur.value) {
      return true;
    }
    prev = cur;
    cur = cur.next;
  }
  return false;
}
```

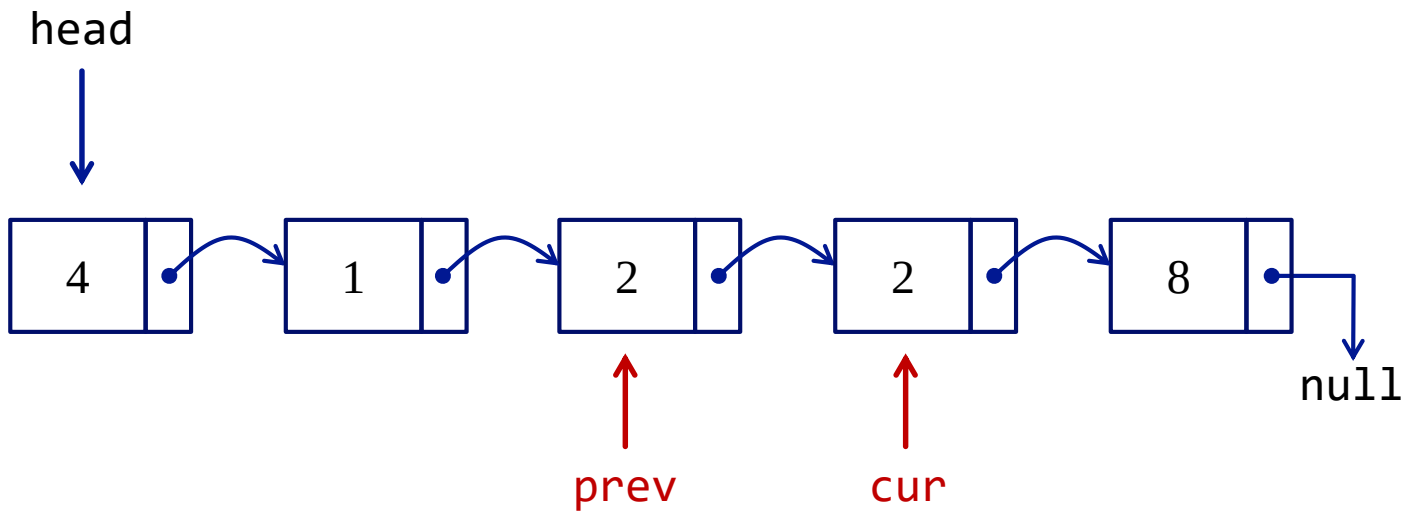


Context-preserving accumulator example

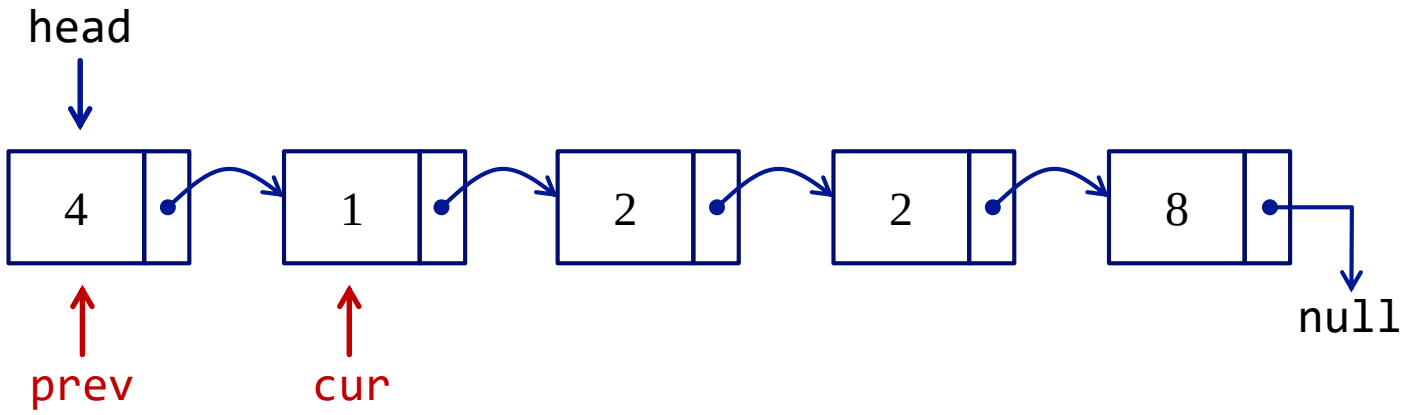
```
public boolean twoInARow() {

  if (this.size() <= 1) {
    return false;
  }

  Node prev = head;
  Node cur = head.next;
  while (cur != null) {
    if (prev.value==cur.value) {
      return true;
    }
    prev = cur;
    cur = cur.next;
  }
  return false;
}
```



Context-preserving accumulator example




```
public boolean twoInARow() {

  if (this.size() <= 1) {
    return false;
  }

  return twoInARowRecursive(head, head.next);
}

public boolean twoInARowRecursive(Node prev, Node cur) {
  if (cur == null) {
    return false;
  }
  if (prev.value==cur.value) {
    return true;
  }
  return twoInARowRecursive(cur, cur.next);
}
```

Context-preserving accumulator example

```
public boolean twoInARow() {

  if (this.size() <= 1) {
    return false;
  }

  return twoInARowRecursive(head, head.next);
}

public boolean twoInARowRecursive(Node prev, Node cur) {
  if (cur == null) {
    return false;
  }
  if (prev.value==cur.value) {
    return true;
  }
  return twoInARowRecursive(cur, cur.next);
}
```