Collections, Part One

Outline for Today

- ***Container Types***
  - Holding lots of pieces of data.
- ***The `Vector` type***
  - Storing sequences.
- ***Reference Parameters***
  - A key part of C++ programming.
- ***Recursion on `Vector`s***
  - Who won the tournament?

Container Types

- A ***collection class*** (also called an ***abstract data type*** or ***container class***) is a data type used to store and organize data in some form.
  - These are things like arrays, lists, maps, dictionaries, etc.
- Our next three lectures exploring collections and how to use them appropriately.
- Later, we'll analyze their efficiencies. For now, let's just focus on how to use them.

Vector

Vector

- A **`Vector`** is a collection class representing a list of things.
- It’s similar to Java's `ArrayList`, JavaScript’s arrays, and Python’s lists.
- To make a `Vector`, use this syntax:

`Vector<`*type*`>` *name*`;`

- All elements of a `Vector` have to have the same type. You specify that type by placing it in <angle brackets> after the word `Vector`.

`Vector` in Action

```
/*      Stanford C++ Version      */
Vector<int> v = { 1, 3, 7 };


v += 271;

cout << v[0] << endl;
cout << v[v.size() - 1] << endl;

Vector<int> first = v.subList(0, 2);
Vector<int> last  = v.subList(2);

v.remove(0);
```

```
"""   Python Version   """
v = [1, 3, 7]


v.append(271)

print(v[0])
print(v[-1])

first = v[0:2]
last  = v[2:]

del v[0]
```

```
/*          Java Version          */
List<> v = new ArrayList<Integer>();
v.add(1); v.add(3); v.add(7);

v.add(271);

System.out.println(v.get(0));
System.out.println(v.get(v.size()-1));

List<Integer> first = v.subList(0, 2);
List<Integer> last  = v.subList(2);

v.remove(0);
```

```
//    JavaScript Version
let v = [1, 3, 7];


v.push(271);

console.log(v[0]);
console.log(v[v.length - 1]);

let first = v.slice(0, 2);
let last  = v.slice(2);

v.splice(0, 0);
```

```
/*       Stanford C++ Version       */
Vector<int> v = { 1, 3, 7 };


v += 271;

cout << v[0] << endl;
cout << v[v.size() - 1] << endl;

Vector<int> first = v.subList(0, 2);
Vector<int> last  = v.subList(2);

v.remove(0);
```

```
"""    Python Version   """
v = [1, 3, 7]


v.append(271)

print(v[0])
print(v[-1])

first = v[0:2]
last  = v[2:]
```

Note the use of curly braces rather than square brackets here.

```
/*          Java Version          */
List<> v = new ArrayList<Integer>();
v.add(1); v.add(3); v.add(7);

v.add(271);

System.out.println(v.get(0));
System.out.println(v.get(v.size()-1));

List<Integer> first = v.subList(0, 2);
List<Integer> last  = v.subList(2);

v.remove(0);
```

```
let v = [1, 3, 7];


v.push(271);

console.log(v[0]);
console.log(v[v.length - 1]);

let first = v.slice(0, 2);
let last  = v.slice(2);

v.splice(0, 0);
```

```cpp
/*      Stanford C++ Version      */
Vector<int> v = { 1, 3, 7 };


v += 271;

cout << v[0] << endl;
cout << v[v.size() - 1] << endl;

Vector<int> first = v.subList(0, 2);
Vector<int> last  = v.subList(2);

v.remove(0);
```

```python
"""   Python Version   """
v = [1, 3, 7]


v.append(271)

print(v[0])
print(v[-1])

first = v[0:2]
last  = v[2:]
```

We append elements using the **+=** operator.

```java
/*          Java Version          */
List<> v = new ArrayList<Integer>();
v.add(1); v.add(3); v.add(7);

v.add(271);

System.out.println(v.get(0));
System.out.println(v.get(v.size()-1));

List<Integer> first = v.subList(0, 2);
List<Integer> last  = v.subList(2);

v.remove(0);
```

```javascript
//    JavaScript Version
let v = [1, 3, 7];


v.push(271);

console.log(v[0]);
console.log(v[v.length - 1]);

let first = v.slice(0, 2);
let last  = v.slice(2);

v.splice(0, 0);
```

```
/*       Stanford C++ Version       */
Vector<int> v = { 1, 3, 7 };


v += 271;

cout << v[0] << endl;
cout << v[v.size() - 1] << endl;

Vector<int> first = v.subList(0, 2);
Vector<int> last  = v.subList(2);

v.remove(0);
```

```
"""    Python Version   """
v = [1, 3, 7]


v.append(271)

print(v[0])
print(v[-1])

first = v[0:2]
last  = v[2:]
```

We select individual elements out of a **Vector** using square brackets. Everything is zero-indexed.

```
/*          Java Version          */
List<> v = new ArrayList<Integer>();
v.add(1); v.add(3); v.add(7);

v.add(271);

System.out.println(v.get(0));
System.out.println(v.get(v.size()-1));

List<Integer> first = v.subList(0, 2);
List<Integer> last  = v.subList(2);

v.remove(0);
```

```
v.push(271);

console.log(v[0]);
console.log(v[v.length - 1]);

let first = v.slice(0, 2);
let last  = v.slice(2);

v.splice(0, 0);
```

```
/*       Stanford C++ Version       */
Vector<int> v = { 1, 3, 7 };


v += 271;

cout << v[0] << endl;
cout << v[v.size() - 1] << endl;

Vector<int> first = v.subList(0, 2);
Vector<int> last  = v.subList(2);

v.remove(0);
```

```
"""    Python Version   """
v = [1, 3, 7]


v.append(271)

print(v[0])
print(v[-1])

first = v[0:2]
last  = v[2:]
```

C++ doesn't support negative array indices to mean "count from the back." We have to do some math to find the index of the last element.

We use the syntax **v.size()** to get the length of a **Vector**.

```
/*          Java Version          */
List<> v = new ArrayList<Integer>();
v.add(1); v.add(3); v.add(7);

v.add(271);

System.out.println(v.get(0));
System.out.println(v.get(v.size()-1));

List<Integer> first = v.subList(0, 2);
List<Integer> last  = v.subList(2);

v.remove(0);
```

```
let last  = v.slice(2);

v.splice(0, 0);
```

```
/*        Stanford C++ Version        */
Vector<int> v = { 1, 3, 7 };


v += 271;

cout << v[0] << endl;
cout << v[v.size() - 1] << endl;

Vector<int> first = v.subList(0, 2);
Vector<int> last  = v.subList(2);

v.remove(0);
```

```
"""   Python Version   """
v = [1, 3, 7]


v.append(271)

print(v[0])
print(v[-1])

first = v[0:2]
last  = v[2:]

del v[0]
```

```
/*          Java Version          */
List<> v = new ArrayList<Integer>();
v.add(1); v.add(3); v.add(7);

v.add(271);

System.out.println(v.get(0));
System.out.println(v.get(v.size()-1));

List<Integer> first = v.subList(0, 2);
List<Integer> last  = v.subList(2);

v.remove(0);
```

```
//     JavaScript Version
```

```
v.splice(0, 0);
```

The **subList** member function is used to get a subrange of the **subList**. Here, first will be the first two elements of the **Vector**, and last will be the list starting at position 2.

```
/*        Stanford C++ Version        */
Vector<int> v = { 1, 3, 7 };


v += 271;

cout << v[0] << endl;
cout << v[v.size() - 1] << endl;

Vector<int> first = v.subList(0, 2);
Vector<int> last  = v.subList(2);

v.remove(0);
```

```
"""    Python Version   """
v = [1, 3, 7]


v.append(271)

print(v[0])
print(v[-1])

first = v[0:2]
last  = v[2:]

del v[0]
```

```
/*          Java Version          */
List<> v = new ArrayList<Integer>();
v.add(1); v.add(3); v.add(7);

v.add(271);

System.out.println(v.get(0));
System.out.println(v.get(v.size()-1));

List<Integer> first = v.subList(0, 2);
List<Integer> last  = v.subList(2);

v.remove(0);
```

```
//     JavaScript Version
console.log(v[v.length - 1]);

let first = v.slice(0, 2);
let last  = v.slice(2);

v.splice(0, 0);
```

We can use the remove member function to remove the element at a given index.

```
/*      Stanford C++ Version      */
Vector<int> v = { 1, 3, 7 };


v += 271;

cout << v[0] << endl;
cout << v[v.size() - 1] << endl;

Vector<int> first = v.subList(0, 2);
Vector<int> last  = v.subList(2);

v.remove(0);
```

```
"""   Python Version   """
v = [1, 3, 7]


v.append(271)

print(v[0])
print(v[-1])

first = v[0:2]
last  = v[2:]

del v[0]
```

```
/*          Java Version          */
List<> v = new ArrayList<Integer>();
v.add(1); v.add(3); v.add(7);

v.add(271);

System.out.println(v.get(0));
System.out.println(v.get(v.size()-1));

List<Integer> first = v.subList(0, 2);
List<Integer> last  = v.subList(2);

v.remove(0);
```

```
//    JavaScript Version
let v = [1, 3, 7];


v.push(271);

console.log(v[0]);
console.log(v[v.length - 1]);

let first = v.slice(0, 2);
let last  = v.slice(2);

v.splice(0, 0);
```

```
/*      Stanford C++ Version       */
Vector<string> v = { "A", "B", "C" };

/* Counting for loop. */
for (int i = 0; i < v.size(); i++) {
    cout << v[i] << endl;
}

/* Range-based for loop. */
for (string elem: v) {
    cout << elem << endl;
}
```

```
"""    Python Version    """
v = ["A", "B", "C"]

# Counting for loop.
for i in range(len(v)):
    print(v[i])

# Range-based for loop.
for elem in v:
    print(elem)
```

```
/*          Java Version           */
List<> v = new ArrayList<String>();
v.add("A"); v.add("B"); v.add("C");

/* Counting for loop. */
for (int i = 0; i < v.size(); i++) {
    System.out.println(v[i]);
}

/* Range-based for loop. */
for (String elem: v) {
    System.out.println(elem);
}
```

```
//    JavaScript Version
let v = ["A", "B", "C"];

// Counting for loop.
for (let i in v) {
    console.log(v[i]);
}

// Range-based for loop.
for (let elem of v) {
    console.log(elem);
}
```

```
/*        Stanford C++ Version          */
Vector<string> v = { "A", "B", "C" };


/* Counting for loop. */
for (int i = 0; i < v.size(); i++) {
    cout << v[i] << endl;
}

/* Range-based for loop. */
for (string elem: v) {
    cout << elem << endl;
}
```

```
"""      Python Version      """
v = ["A", "B", "C"]


# Counting for loop.
for i in range(len(v)):
    print(v[i])


# Range-based for loop.
for elem in v:
    print(elem)
```

```
/*           Java Version               */
List<> v = new ArrayList<String>();
v.add("A"); v.add("B"); v.add("C");

/* Counting for loop. */
for (int i = 0; i < v.size(); i++) {
    System.out.println(v[i]);
}

/* Range-based for loop. */
for (String elem: v) {
    System.out.println(elem);
}
```

We can iterate over the elements of a **Vector** by counting upward from 0 (inclusive) to its size (exclusive) and accessing each element.

```
for (let elem of v) {
    console.log(elem);
}
```

```
/*       Stanford C++ Version          */
Vector<string> v = { "A", "B", "C" };


/* Counting for loop. */
for (int i = 0; i < v.size(); i++) {
    cout << v[i] << endl;
}

/* Range-based for loop. */
for (string elem: v) {
    cout << elem << endl;
}
```

```
"""      Python Version      """
v = ["A", "B", "C"]


# Counting for loop.
for i in range(len(v)):
    print(v[i])


# Range-based for loop.
for elem in v:
    print(elem)
```

```
/*           Java Version              */
List<> v = new ArrayList<String>();
v.add("A"); v.add("B"); v.add("C");

/* Counting for loop. */
for (int i = 0; i < v.size(); i++) {
    System.out.println(v[i]);
}

/* Range-based for loop. */
for (String elem: v) {
    System.out.println(elem);
}
```

We can also use this loop structure, which visits each element of the `Vector` in the order in which they appear.

```
}

// Range-based for loop.
for (let elem of v) {
    console.log(elem);
}
```

```
/*      Stanford C++ Version       */
Vector<string> v = { "A", "B", "C" };

/* Counting for loop. */
for (int i = 0; i < v.size(); i++) {
    cout << v[i] << endl;
}

/* Range-based for loop. */
for (string elem: v) {
    cout << elem << endl;
}
```

```
"""    Python Version    """
v = ["A", "B", "C"]

# Counting for loop.
for i in range(len(v)):
    print(v[i])

# Range-based for loop.
for elem in v:
    print(elem)
```

```
/*          Java Version           */
List<> v = new ArrayList<String>();
v.add("A"); v.add("B"); v.add("C");

/* Counting for loop. */
for (int i = 0; i < v.size(); i++) {
    System.out.println(v[i]);
}

/* Range-based for loop. */
for (String elem: v) {
    System.out.println(elem);
}
```

```
//    JavaScript Version
let v = ["A", "B", "C"];

// Counting for loop.
for (let i in v) {
    console.log(v[i]);
}

// Range-based for loop.
for (let elem of v) {
    console.log(elem);
}
```

To read more about the `Vector` and how to use it, check out the

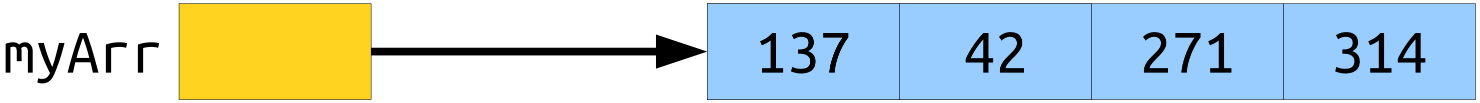
Stanford C++ Library Documentation

up on the course website.

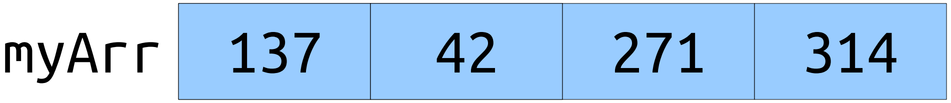
An Important Nuance

Objects in C++

- In most programming languages, object variables are *references*.
- The variable isn't the object; it just says where to look for that object.



- C++ is different. In C++, object variables *literally are* the objects.

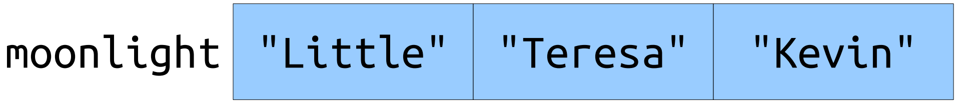


- While C++ does have a `new` keyword, we won't be using it until later in the quarter.

Pass-by-Value

- In C++, objects are passed into functions by *value*. The function gets its own local copy of the argument to work with.
- Don't just take my word for it – watch what happens!

```
int main() {
   Vector<string> moonlight = { "Little", "Teresa", "Kevin" };

   growUp(moonlight);

   /* … */
}
```



moonlight	"Little"	"Teresa"	"Kevin"

```
int main() {
   Vector<string> moonlight = { "Little", "Teresa", "Kevin" };

   growUp(moonlight);

   /* … */
}
```

moonlight	"Little"	"Teresa"	"Kevin"

```
int main() {
   Vector<string> moonlight = { "Little", "Teresa", "Kevin" };
   growUp(moonlight);
   /* … */
}
```

moonlight	"Little"	"Teresa"	"Kevin"

```
void growUp(Vector<string> cast) {
   cast += "Paula";
   cast[0] = "Chiron";
}
```

cast	"Little"	"Teresa"	"Kevin"

```
int main() {
   Vector<string> moonlight = { "Little", "Teresa", "Kevin" };

   growUp(moonlight);

   /* … */
}
```

moonlight	"Little"	"Teresa"	"Kevin"

```
void growUp(Vector<string> cast) {
   cast += "Paula";
   cast[0] = "Chiron";
}
```

cast	"Little"	"Teresa"	"Kevin"

```
int main() {
   Vector<string> moonlight = { "Little", "Teresa", "Kevin" };

   growUp(moonlight);

   /* … */
}
```

moonlight	"Little"	"Teresa"	"Kevin"

```
void growUp(Vector<string> cast) {
   cast += "Paula";
   cast[0] = "Chiron";
}
```

cast	"Little"	"Teresa"	"Kevin"	"Paula"

```
int main() {
   Vector<string> moonlight = { "Little", "Teresa", "Kevin" };

   growUp(moonlight);

   /* … */
}
```

moonlight	"Little"	"Teresa"	"Kevin"

```
void growUp(Vector<string> cast) {
   cast += "Paula";
   cast[0] = "Chiron";
}
```

cast	"Little"	"Teresa"	"Kevin"	"Paula"

```
int main() {
   Vector<string> moonlight = { "Little", "Teresa", "Kevin" };

   growUp(moonlight);

   /* … */
}
```

moonlight	"Little"	"Teresa"	"Kevin"

```
void growUp(Vector<string> cast) {
   cast += "Paula";
   cast[0] = "Chiron";
}
```

cast	"Chiron"	"Teresa"	"Kevin"	"Paula"

```
int main() {
   Vector<string> moonlight = { "Little", "Teresa", "Kevin" };

   growUp(moonlight);

   /* … */
}
```

moonlight	"Little"	"Teresa"	"Kevin"

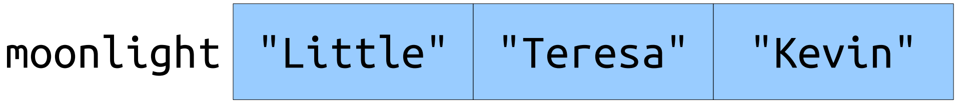
```
void growUp(Vector<string> cast) {
   cast += "Paula";
   cast[0] = "Chiron";
}
```

cast	"Chiron"	"Teresa"	"Kevin"	"Paula"

```
int main() {
   Vector<string> moonlight = { "Little", "Teresa", "Kevin" };

   growUp(moonlight);

   /* … */
}
```

moonlight	"Little"	"Teresa"	"Kevin"

```
int main() {
   Vector<string> moonlight = { "Little", "Teresa", "Kevin" };

   growUp(moonlight);

   /* … */
}
```

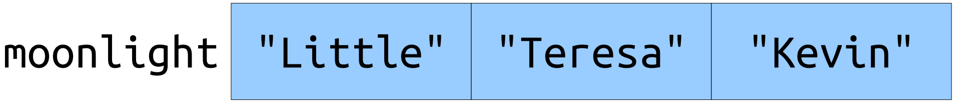


moonlight "Little" "Teresa" "Kevin"

Pass-by-Reference

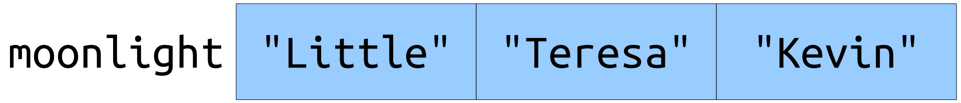
- In C++, there’s the option to pass parameters into function ***by reference***.
- This means that the actual argument itself gets sent into the function, not a copy of it.
- To declare a function that takes an argument by reference, put an ampersand (&) after the type of the argument.

```
int main() {
   Vector<string> moonlight = { "Little", "Teresa", "Kevin" };

   growUp(moonlight);

   /* … */
}
```



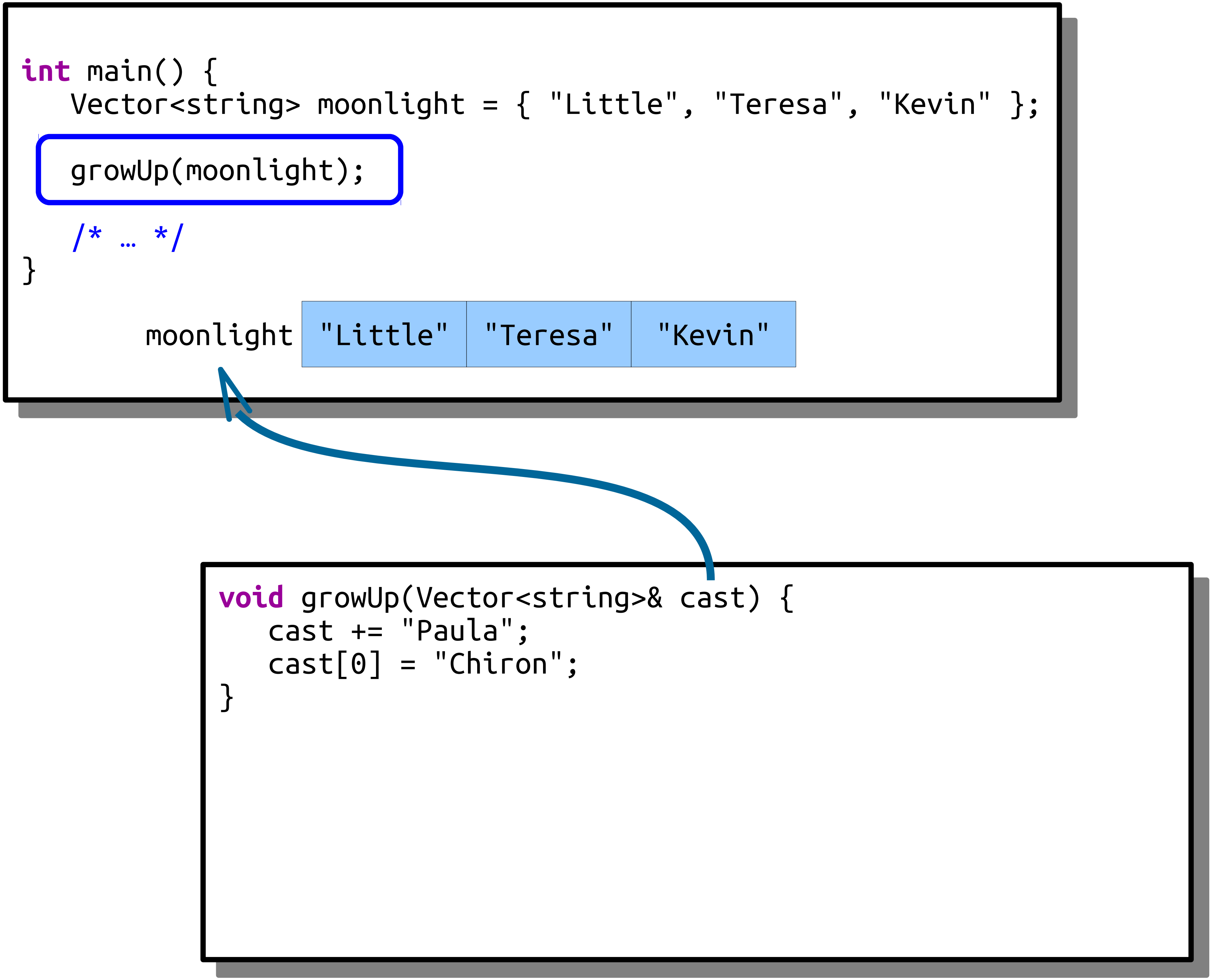
moonlight	"Little"	"Teresa"	"Kevin"

```
int main() {
   Vector<string> moonlight = { "Little", "Teresa", "Kevin" };

   growUp(moonlight);

   /* … */
}
```



moonlight

"Little"	"Teresa"	"Kevin"

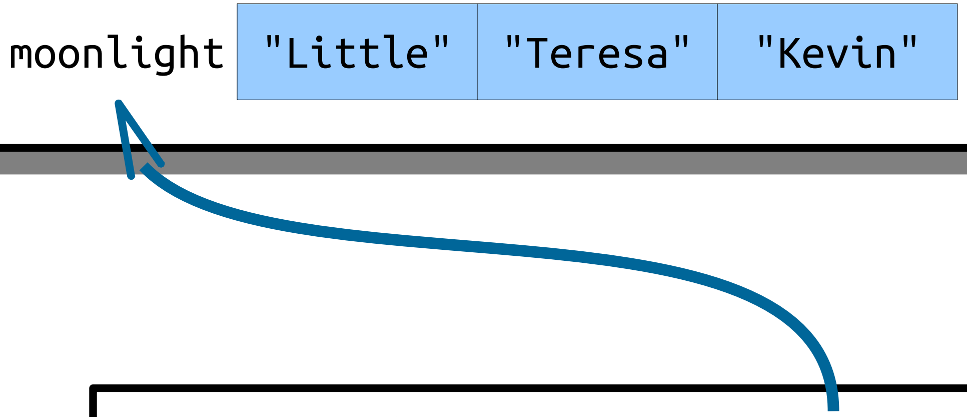


```
int main() {
   Vector<string> moonlight = { "Little", "Teresa", "Kevin" };
   growUp(moonlight);
   /* … */
}
```

moonlight	"Little"	"Teresa"	"Kevin"

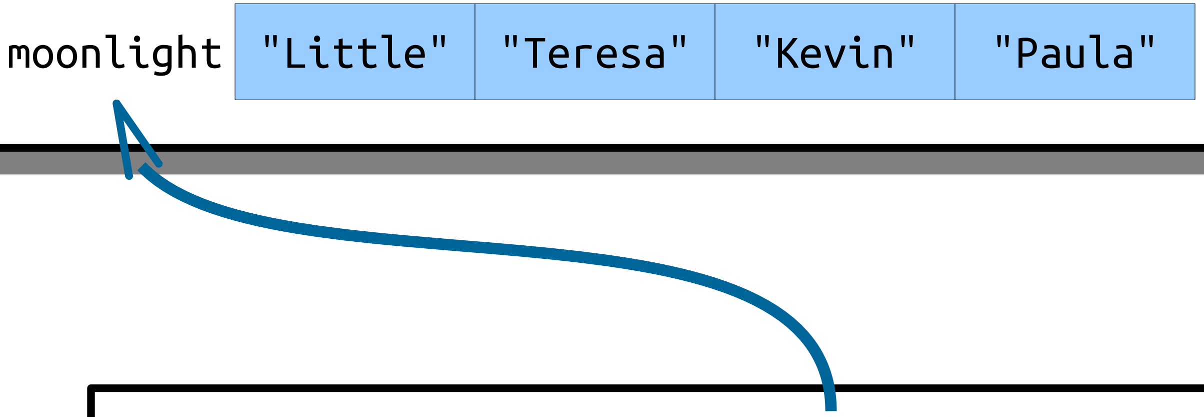
```
void growUp(Vector<string>& cast) {
   cast += "Paula";
   cast[0] = "Chiron";
}
```

```
int main() {
   Vector<string> moonlight = { "Little", "Teresa", "Kevin" };

   growUp(moonlight);

   /* … */
}
```



```
void growUp(Vector<string>& cast) {
   cast += "Paula";
   cast[0] = "Chiron";
}
```

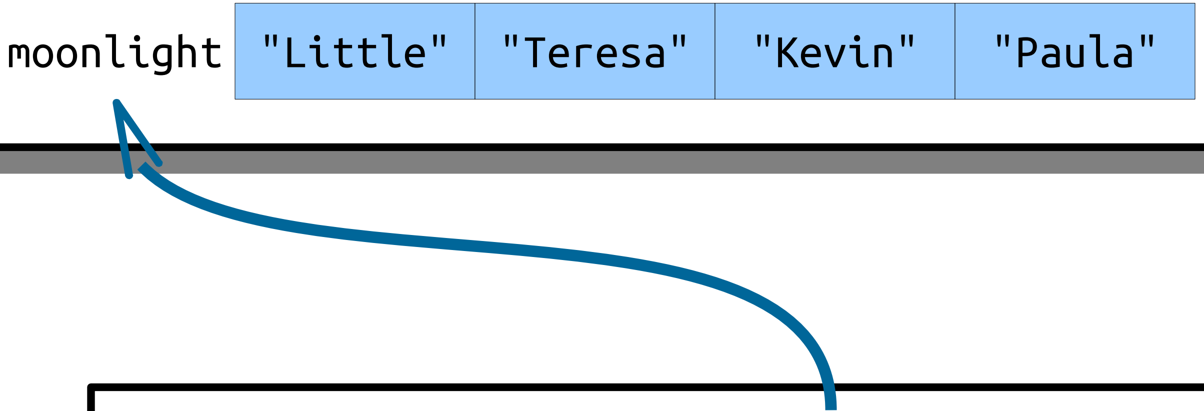
```
int main() {
   Vector<string> moonlight = { "Little", "Teresa", "Kevin" };

   growUp(moonlight);

   /* … */
}
```



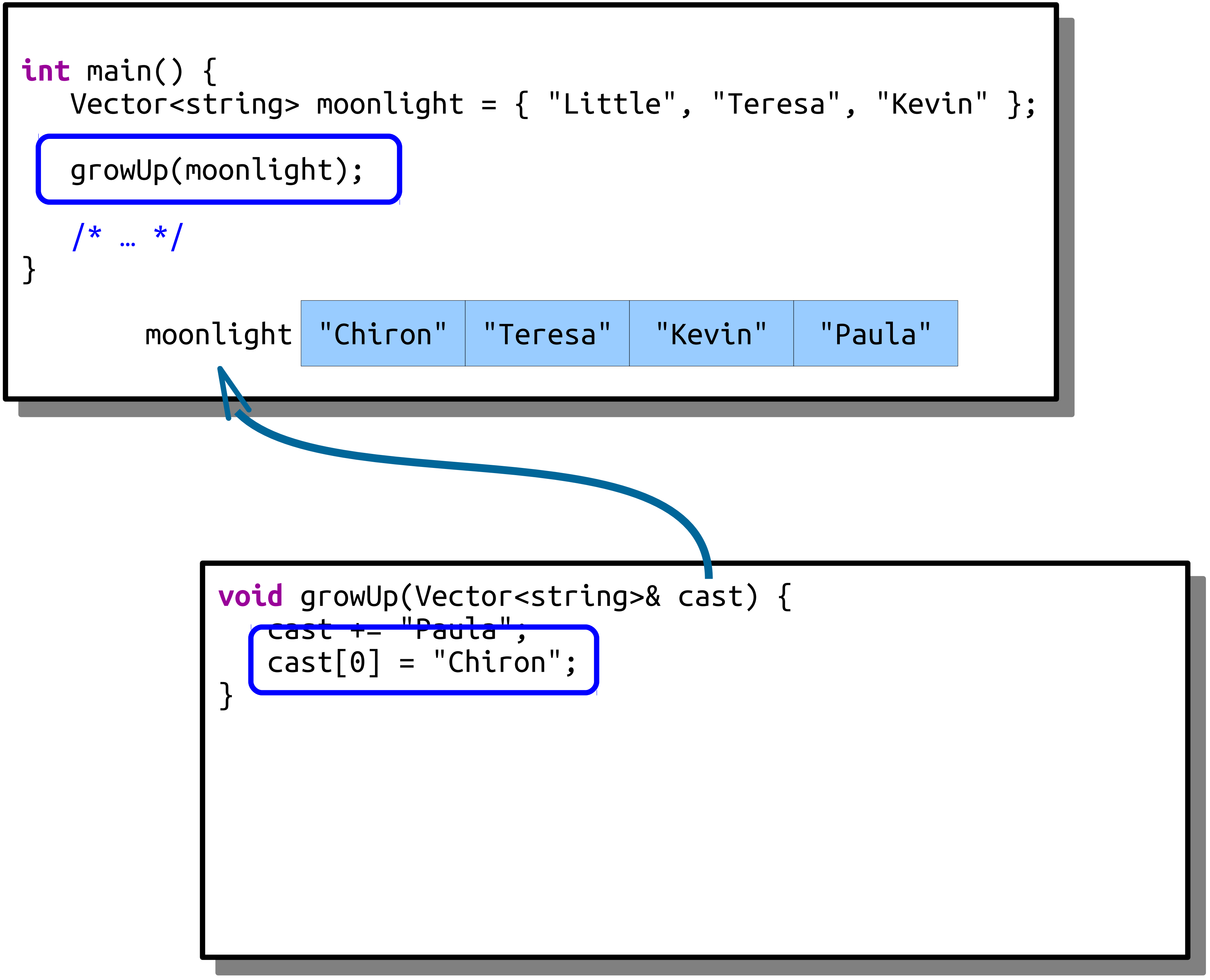
moonlight	"Little"	"Teresa"	"Kevin"	"Paula"

```
void growUp(Vector<string>& cast) {
   cast += "Paula";
   cast[0] = "Chiron";
}
```

```
int main() {
   Vector<string> moonlight = { "Little", "Teresa", "Kevin" };

   growUp(moonlight);

   /* … */
}
```



```
void growUp(Vector<string>& cast) {
   cast += "Paula";
   cast[0] = "Chiron";
}
```

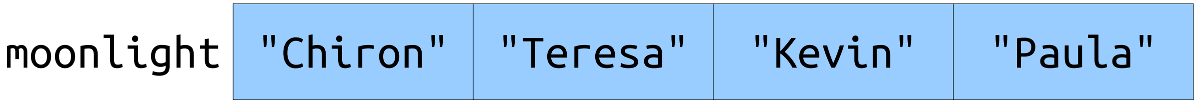


```
int main() {
   Vector<string> moonlight = { "Little", "Teresa", "Kevin" };

   growUp(moonlight);

   /* … */
}
```

moonlight	"Chiron"	"Teresa"	"Kevin"	"Paula"

```
void growUp(Vector<string>& cast) {
   cast += "Paula";
   cast[0] = "Chiron";
}
```

```
int main() {
   Vector<string> moonlight = { "Little", "Teresa", "Kevin" };

   growUp(moonlight);

   /* … */
}
```



moonlight	"Chiron"	"Teresa"	"Kevin"	"Paula"

Time-Out for Announcements!

Sections

- Discussion sections start this week!
- Forgot to sign up? The signup link will reopen on Tuesday at 5PM, and you can choose any open section time.
- If your section time doesn’t work for you, you can switch into any section with available space starting Tuesday at 5PM. Visit cs198.stanford.edu to do this.
- Still doesn’t work for you? Ping Katherine!

```
return;
```

Recursion on `Vectors`

Finding the Largest Number

Finding the Largest Number

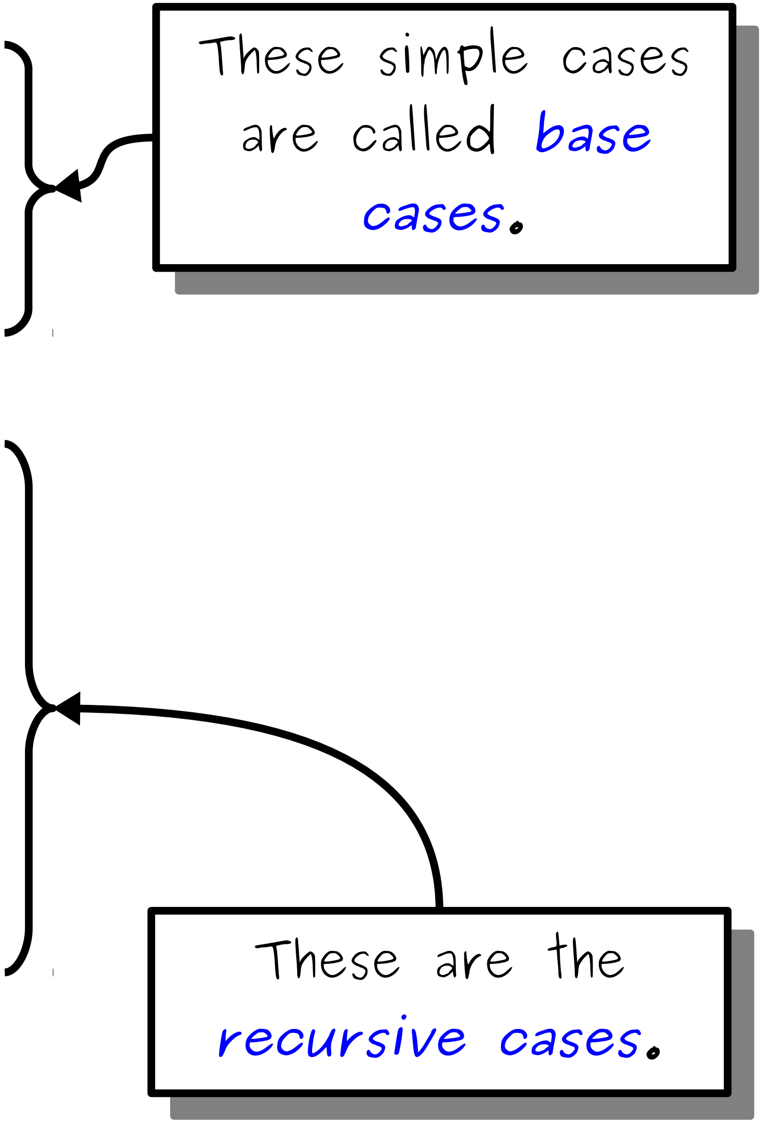
- Our goal is to write a function

  ```
  int maxOf(Vector<int> numbers);
  ```

  that takes as input a `Vector<int>`, then returns the largest number in the `Vector`.
- We're going to assume the `Vector` has at least one element in it; otherwise, it's not possible to return the largest value!
- Let's see how to do this.

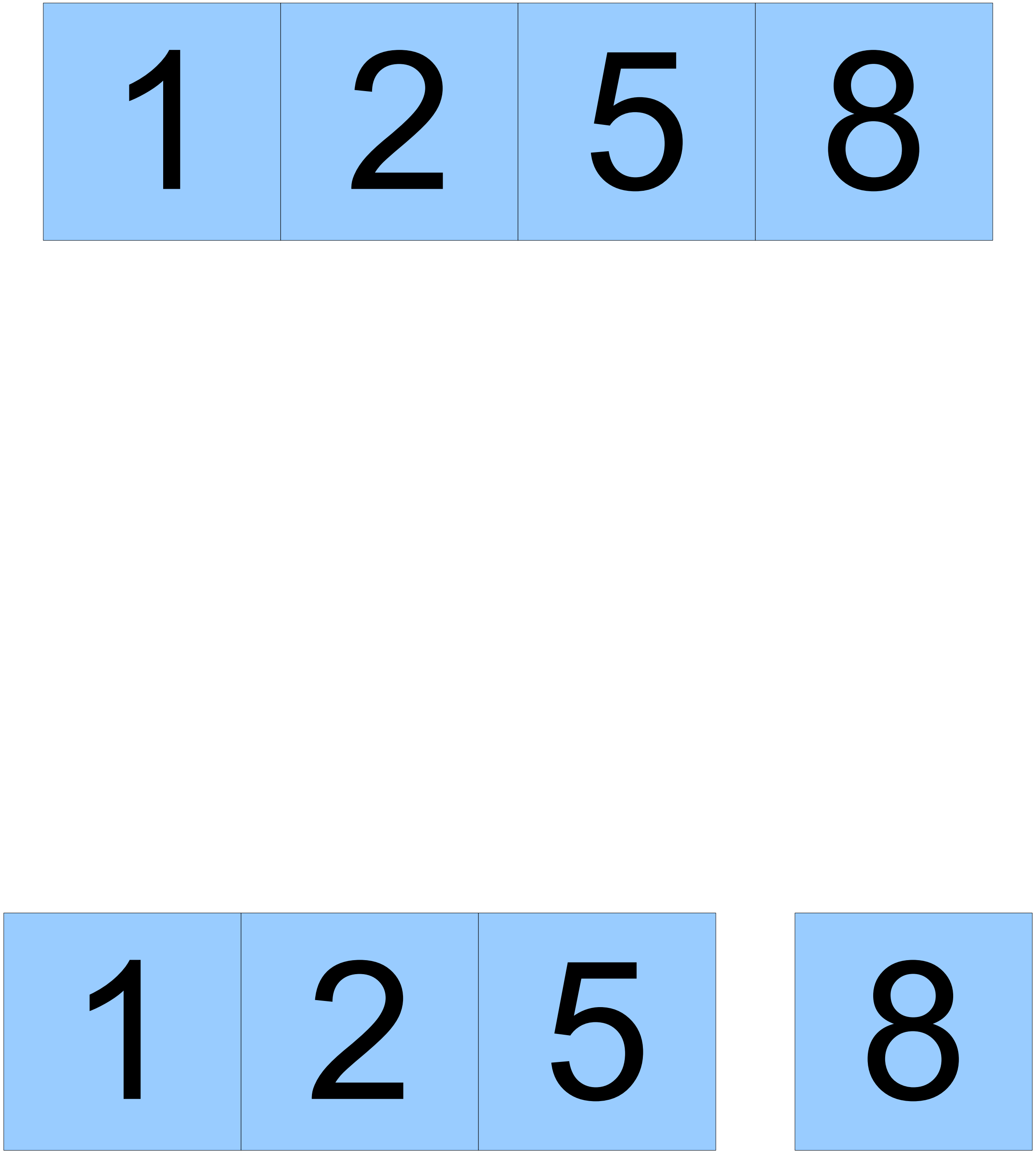
Thinking Recursively

```
if (The problem is very simple) {
    Directly solve the problem.
    Return the solution.
} else {
    Split the problem into one or more
    smaller problems with the same
    structure as the original.
    Solve each of those smaller problems.
    Combine the results to get the overall
    solution.
    Return the overall solution.
}
```

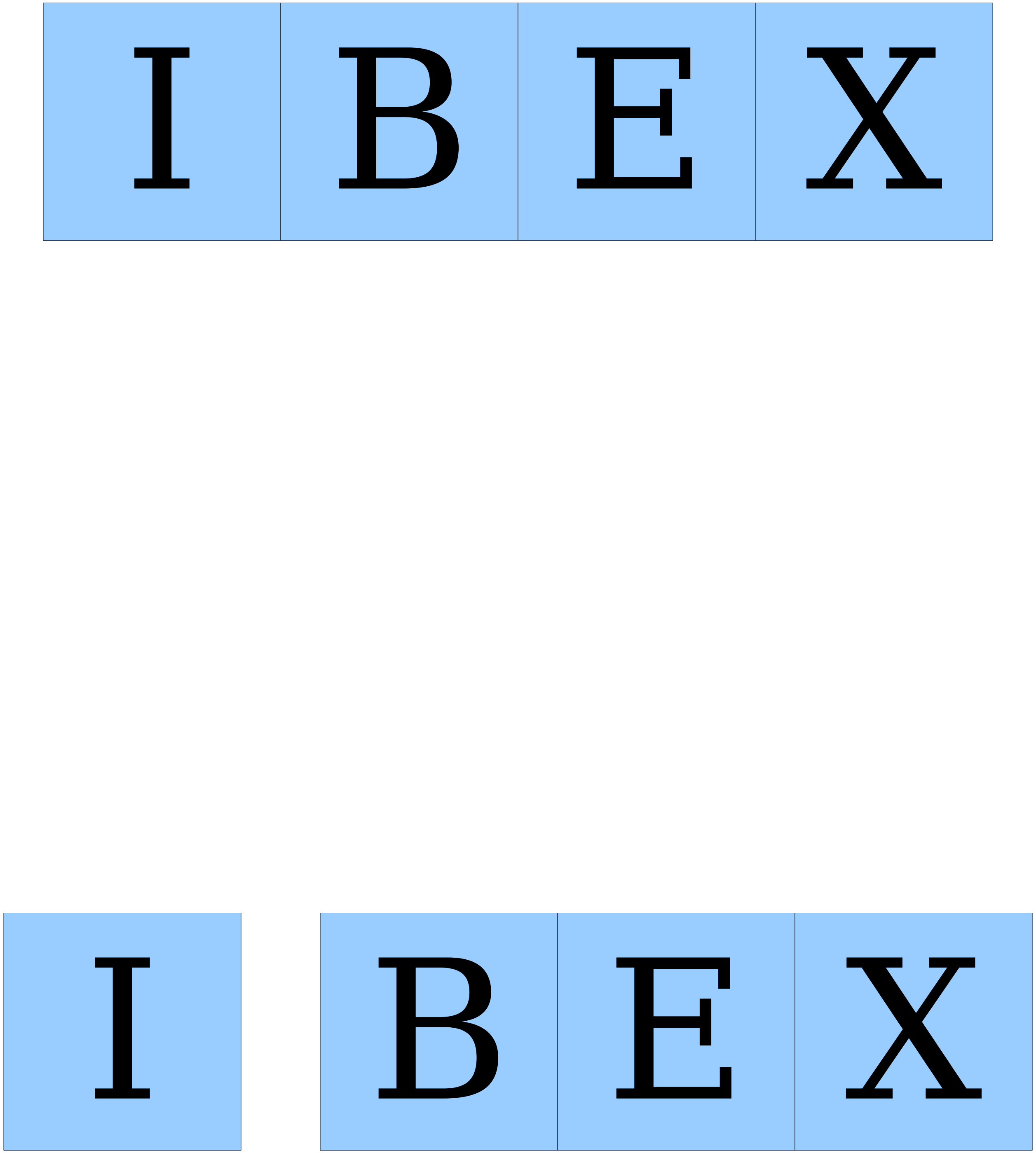


These simple cases are called **base cases.**

These are the **recursive cases.**

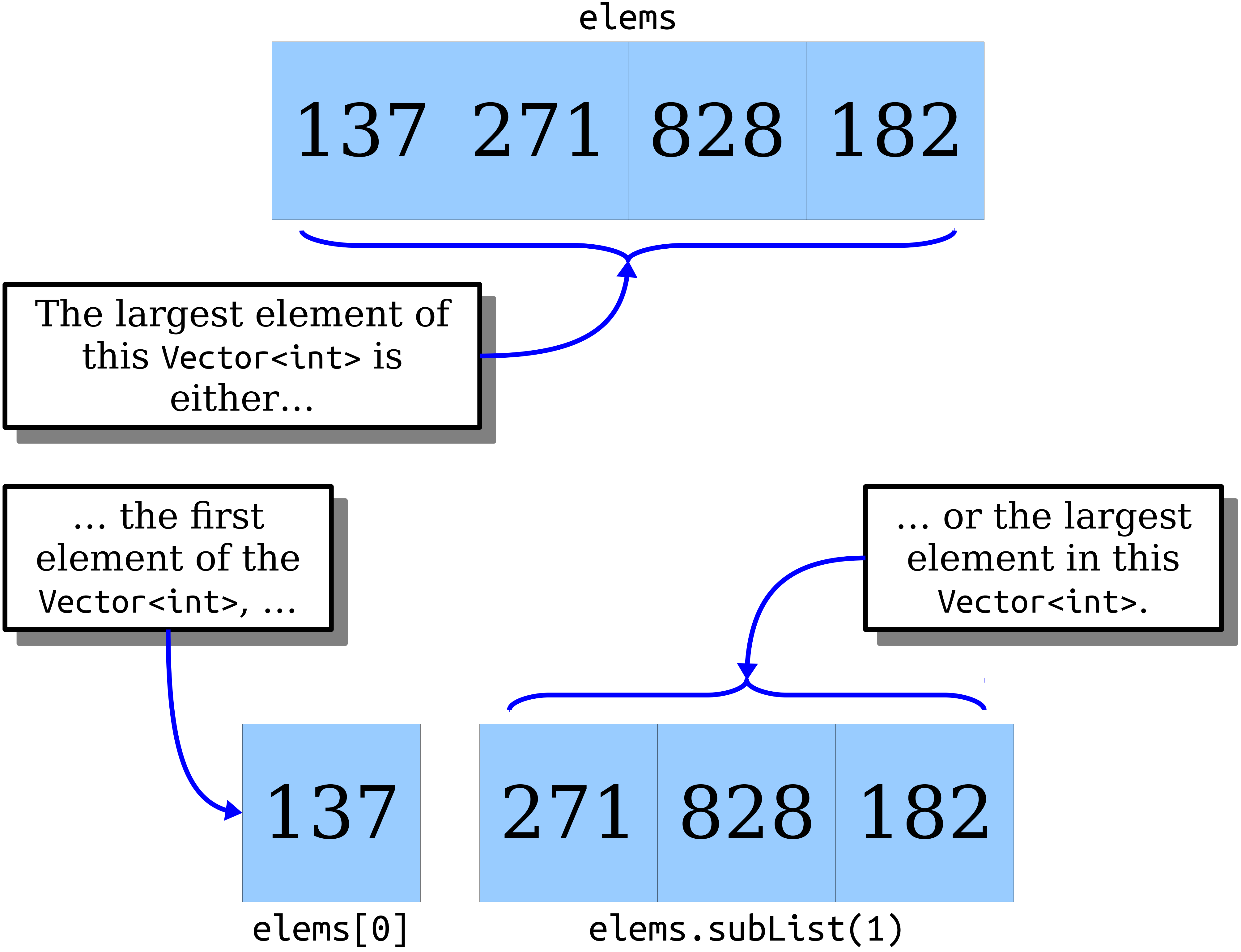


1 2 5 8
1 2 5
8



I B E X

I B E X



elems

137	271	828	182

The largest element of this Vector<int> is either…

… the first element of the Vector<int>, …

… or the largest element in this Vector<int>.

137

elems[0]

271	828	182

elems.subList(1)

Tracing the Recursion

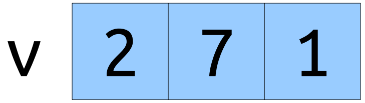
```
int main() {
   Vector<int> v = { 2, 7, 1 };
   cout << maxOf(v) << endl;
   return 0;
}
```

Tracing the Recursion

```
int main() {
    Vector<int> v = { 2, 7, 1 };
    cout << maxOf(v) << endl;
    return 0;
}
```

Tracing the Recursion

```
int main() {
    Vector<int> v = { 2, 7, 1 };
    cout << maxOf(v) << endl;
    return 0;
}
```



Tracing the Recursion

```
int main() {
   Vector<int> v = { 2, 7, 1 };
   cout << maxOf(v) << endl;
   return 0;
}
```

v	2	7	1

Tracing the Recursion

```
int main() {
   Vector<int> v = { 2, 7, 1 };
   cout << maxOf(v) << endl;
   return 0;
}
```

v	2	7	1

Tracing the Recursion

```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int first = elems[0];
    Vector<int> rest = elems.subList(1);
    return max(first, maxOf(rest));
  }
}
```

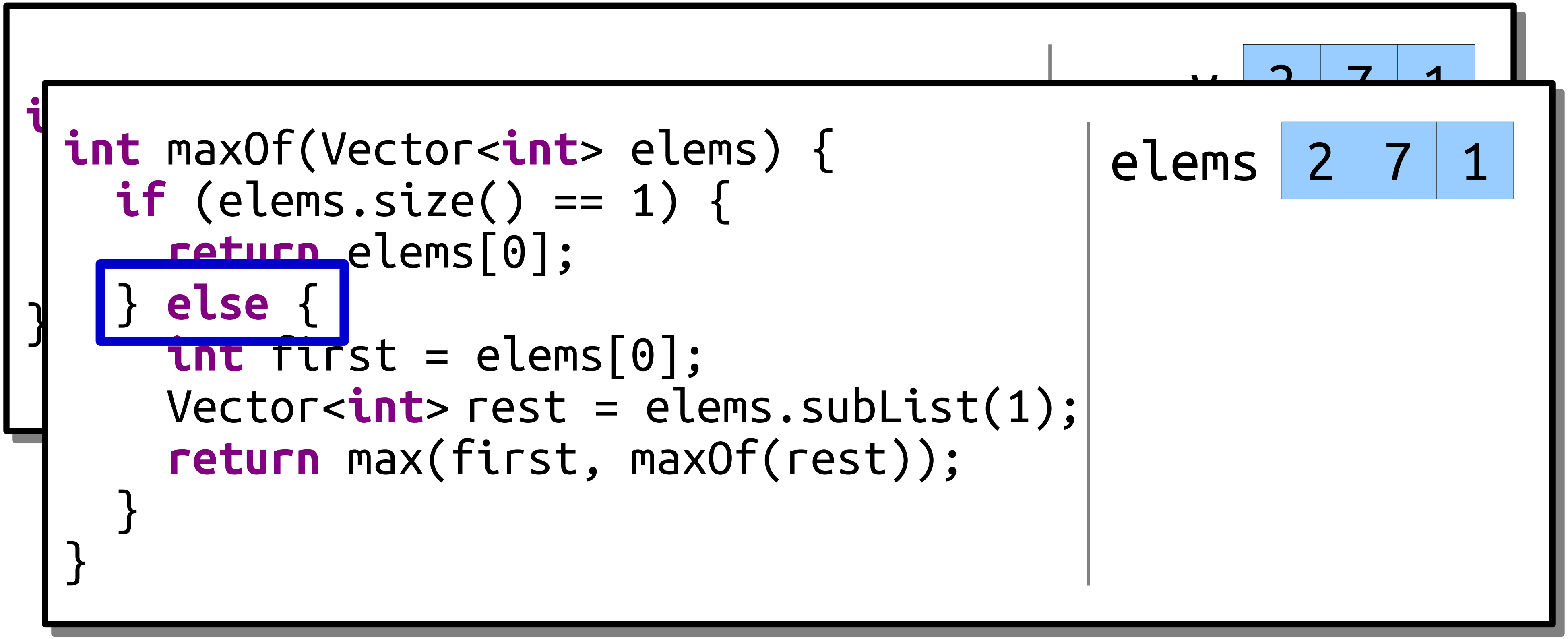
elems	2	7	1

Tracing the Recursion

```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int first = elems[0];
    Vector<int> rest = elems.subList(1);
    return max(first, maxOf(rest));
  }
}
```

elems	2	7	1

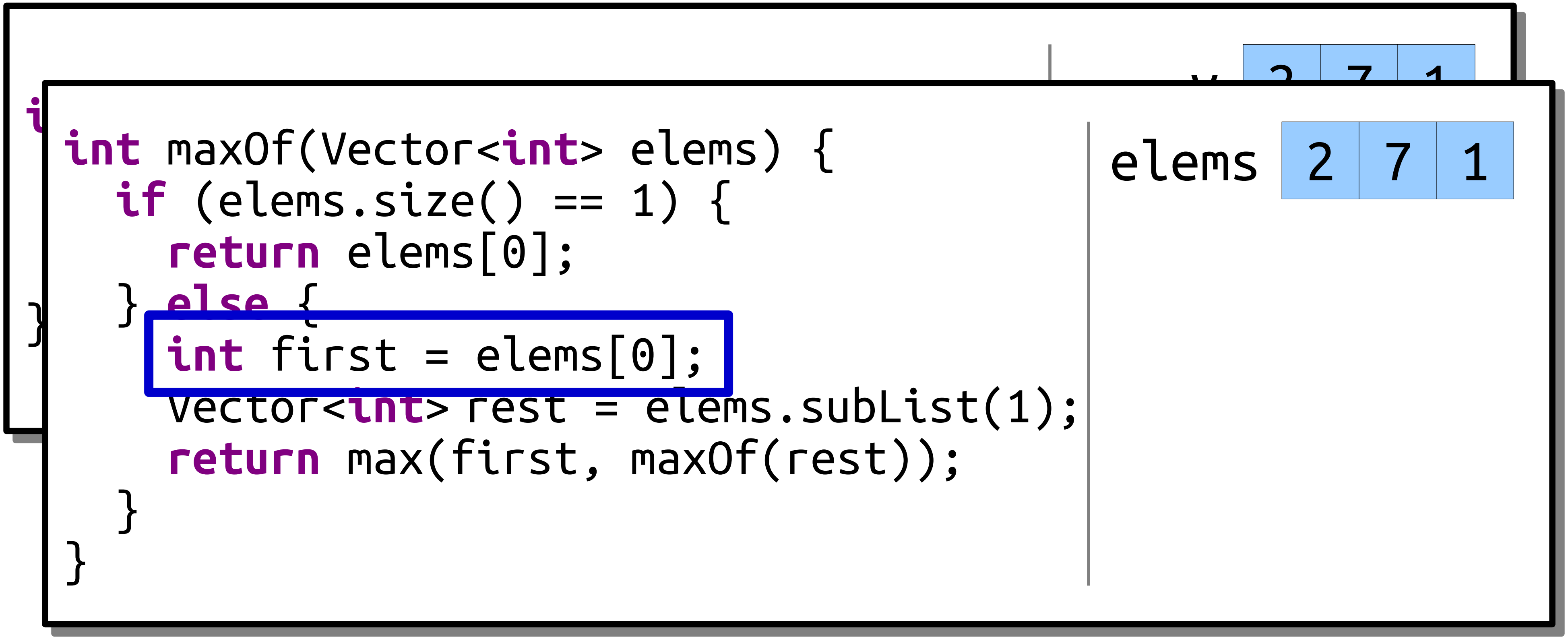
Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int first = elems[0];
    Vector<int> rest = elems.subList(1);
    return max(first, maxOf(rest));
  }
}
```

elems	2	7	1

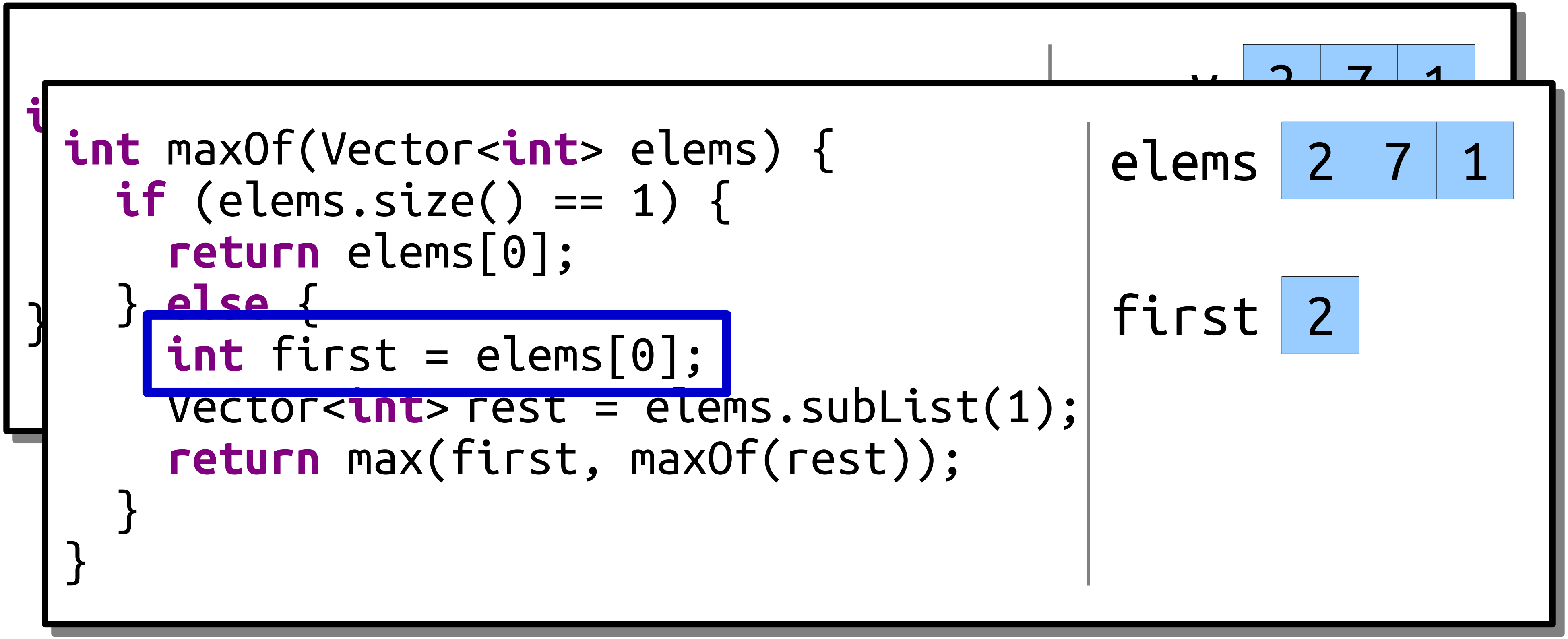
Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int first = elems[0];
    Vector<int> rest = elems.subList(1);
    return max(first, maxOf(rest));
  }
}
```

elems	2	7	1

Tracing the Recursion



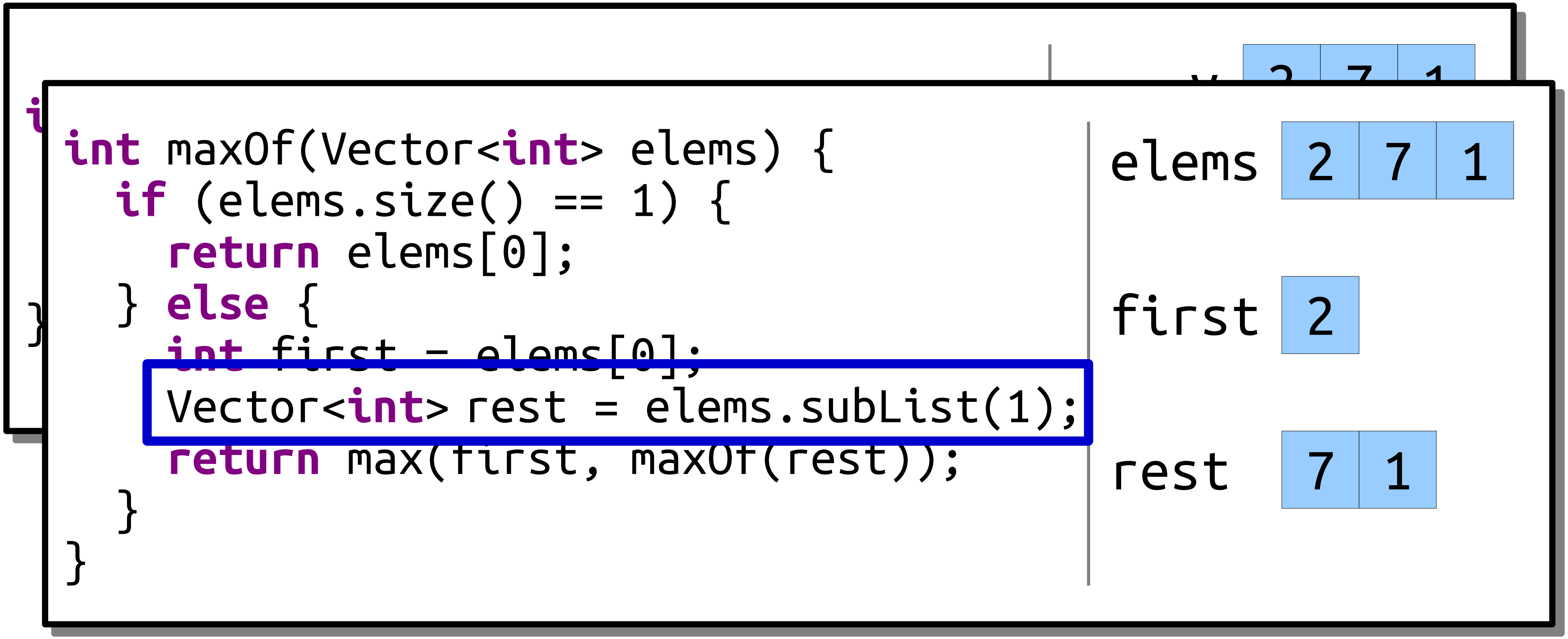
```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int first = elems[0];
    Vector<int> rest = elems.subList(1);
    return max(first, maxOf(rest));
  }
}
```

Tracing the Recursion

```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int first = elems[0];
    Vector<int> rest = elems.subList(1);
    return max(first, maxOf(rest));
  }
}
```

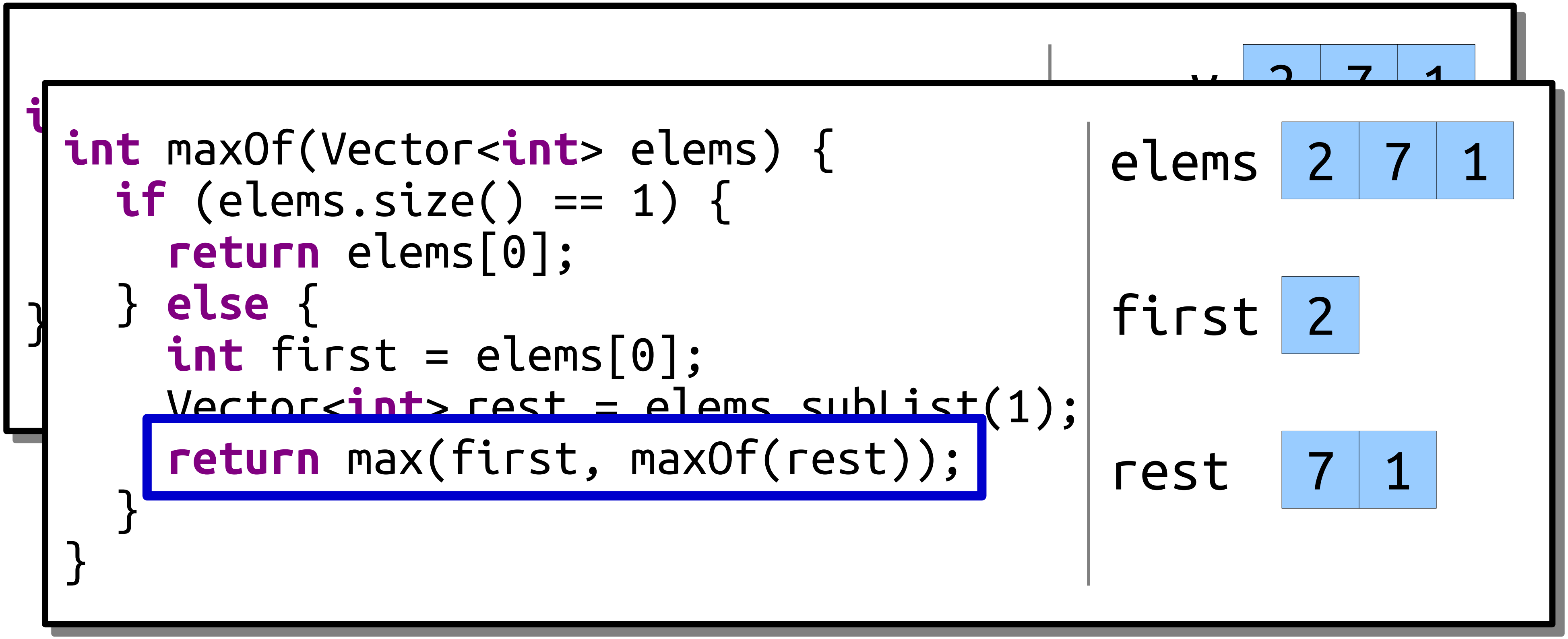
elems	2	7	1
first	2		

Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int first = elems[0];
    Vector<int> rest = elems.subList(1);
    return max(first, maxOf(rest));
  }
}
```

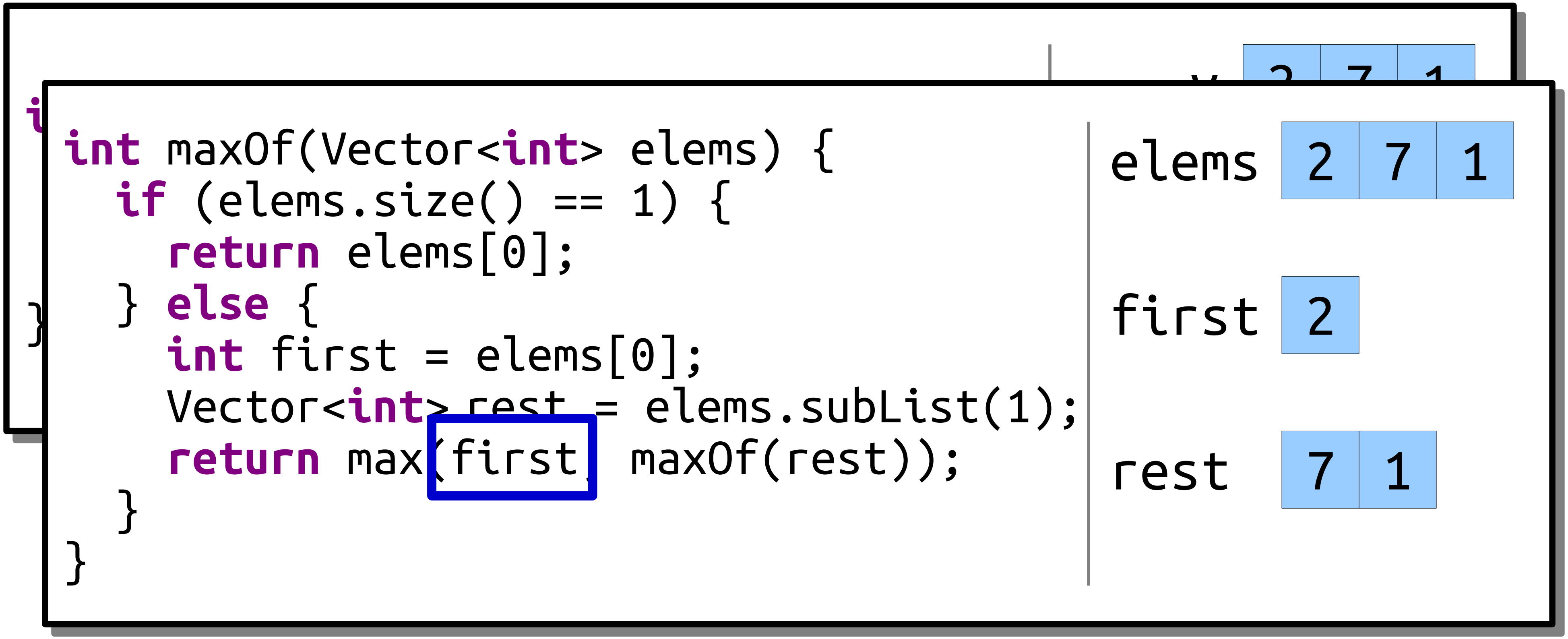
Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int first = elems[0];
    Vector<int> rest = elems.subList(1);
    return max(first, maxOf(rest));
  }
}
```

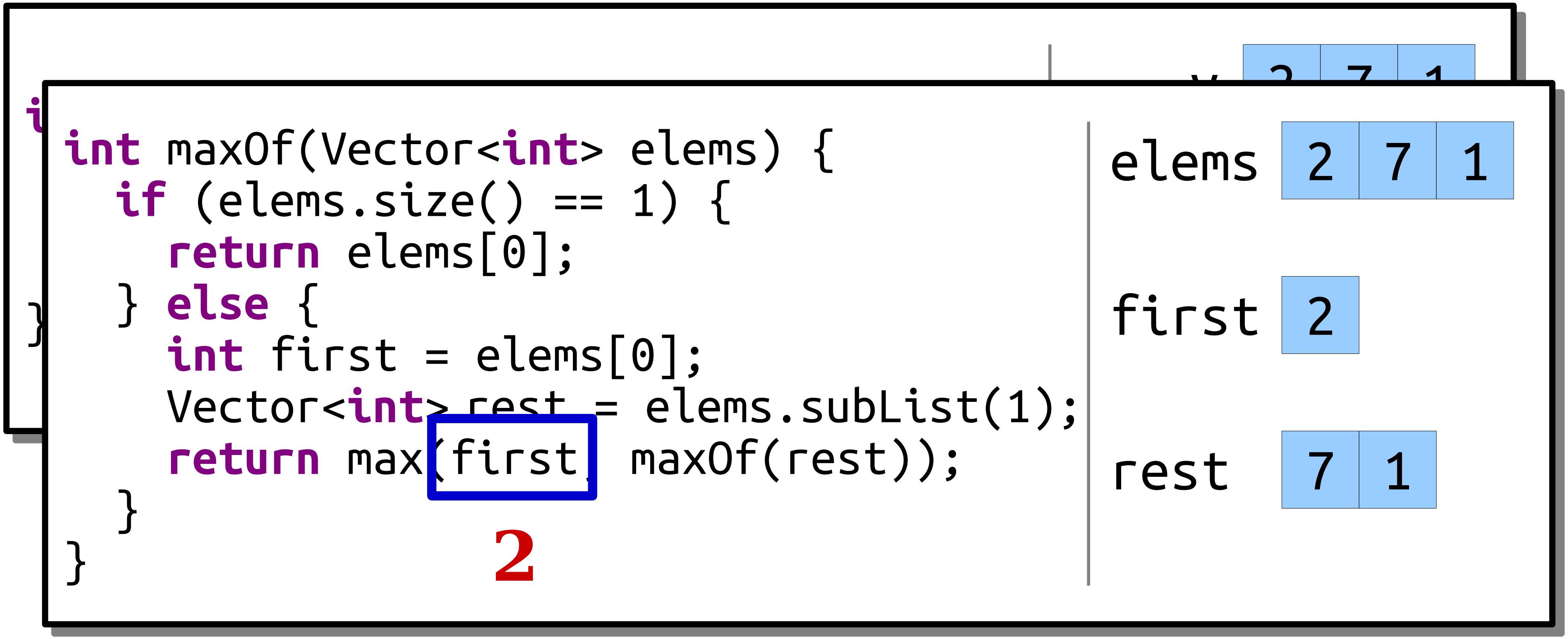
elems	2	7	1
first	2		
rest	7	1	

Tracing the Recursion



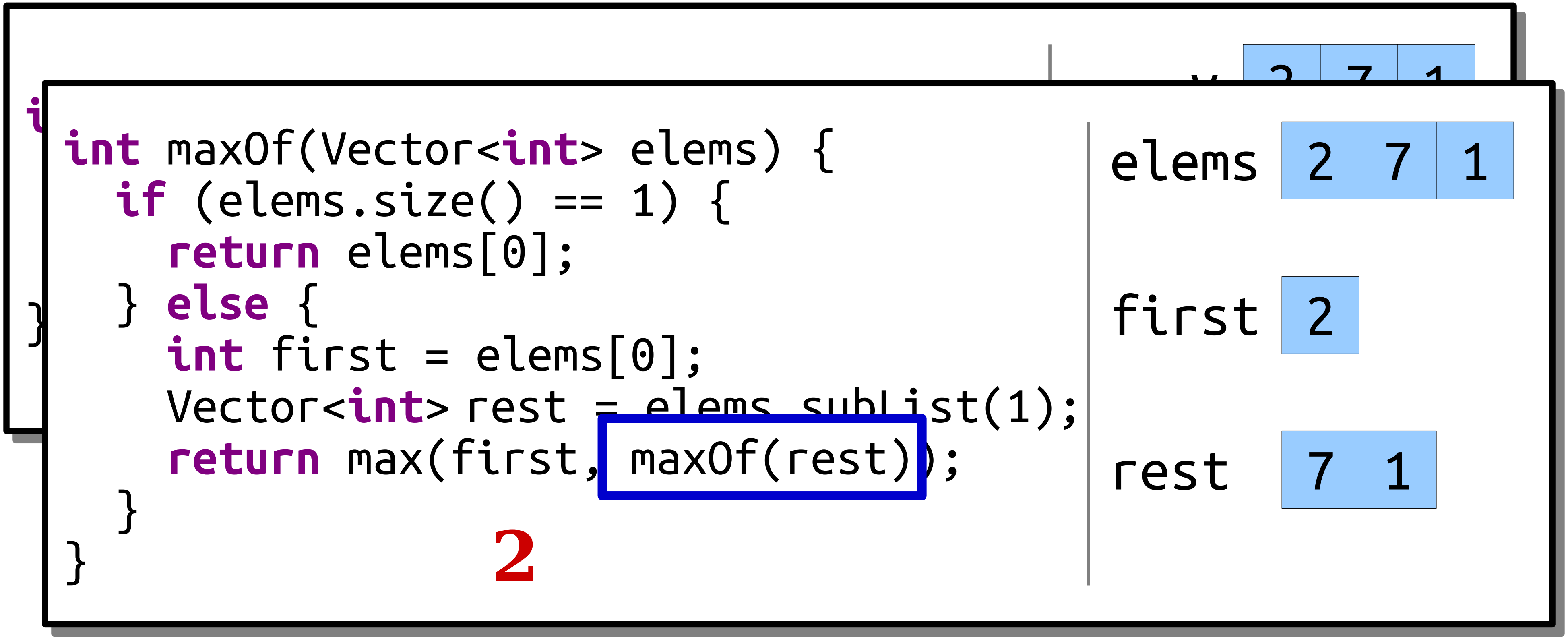
```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int first = elems[0];
    Vector<int> rest = elems.subList(1);
    return max(first, maxOf(rest));
  }
}
```

Tracing the Recursion



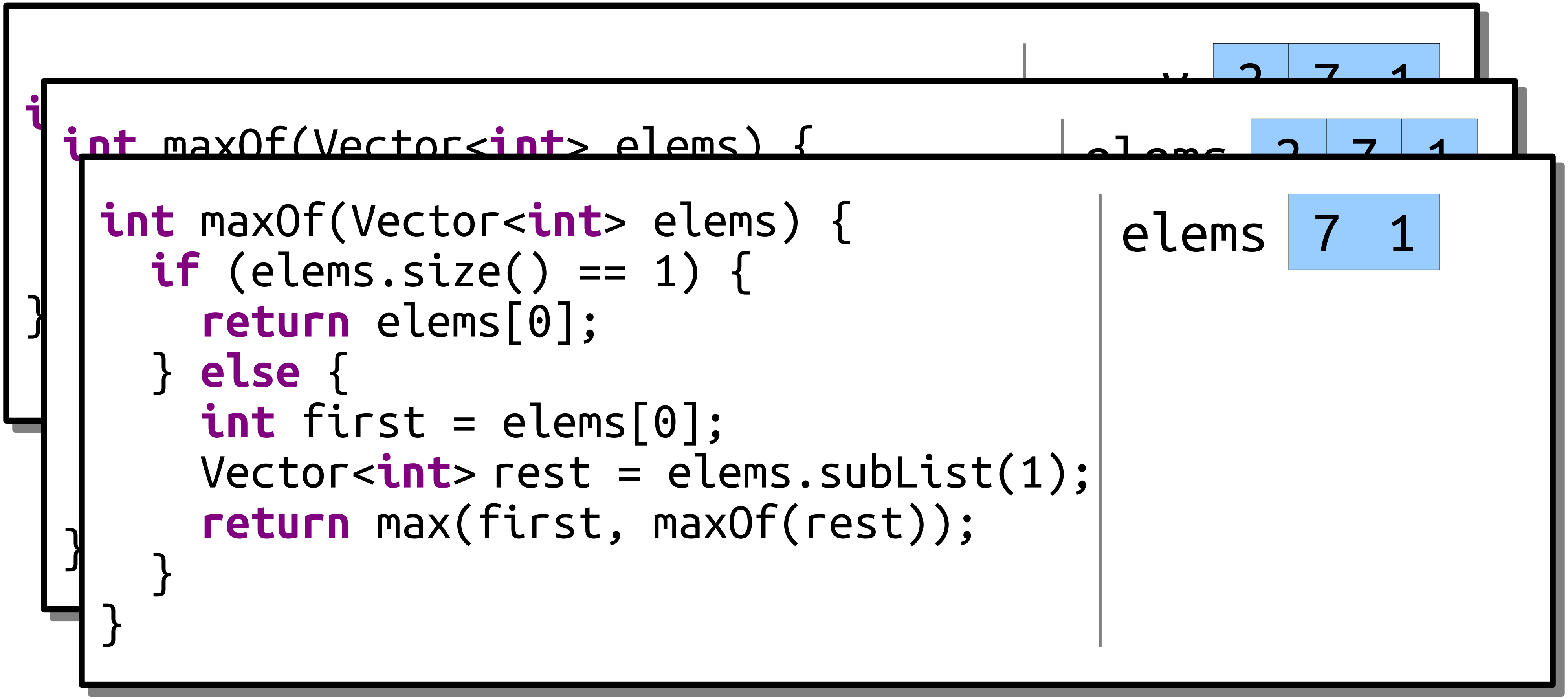
```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int first = elems[0];
    Vector<int> rest = elems.subList(1);
    return max(first, maxOf(rest));
  }
}
```

Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int first = elems[0];
    Vector<int> rest = elems.subList(1);
    return max(first, maxOf(rest));
  }
}
```

Tracing the Recursion

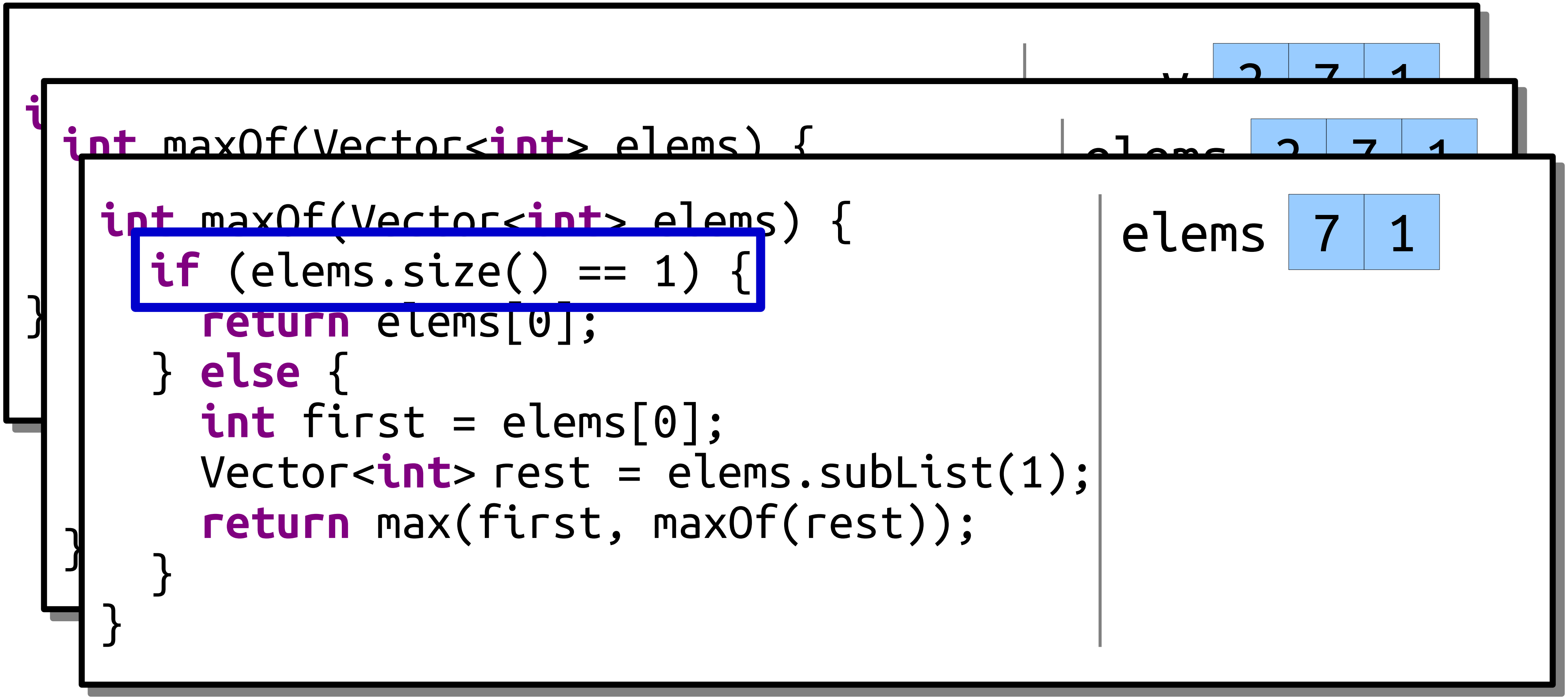


```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int first = elems[0];
    Vector<int> rest = elems.subList(1);
    return max(first, maxOf(rest));
  }
}
```

elems

7	1

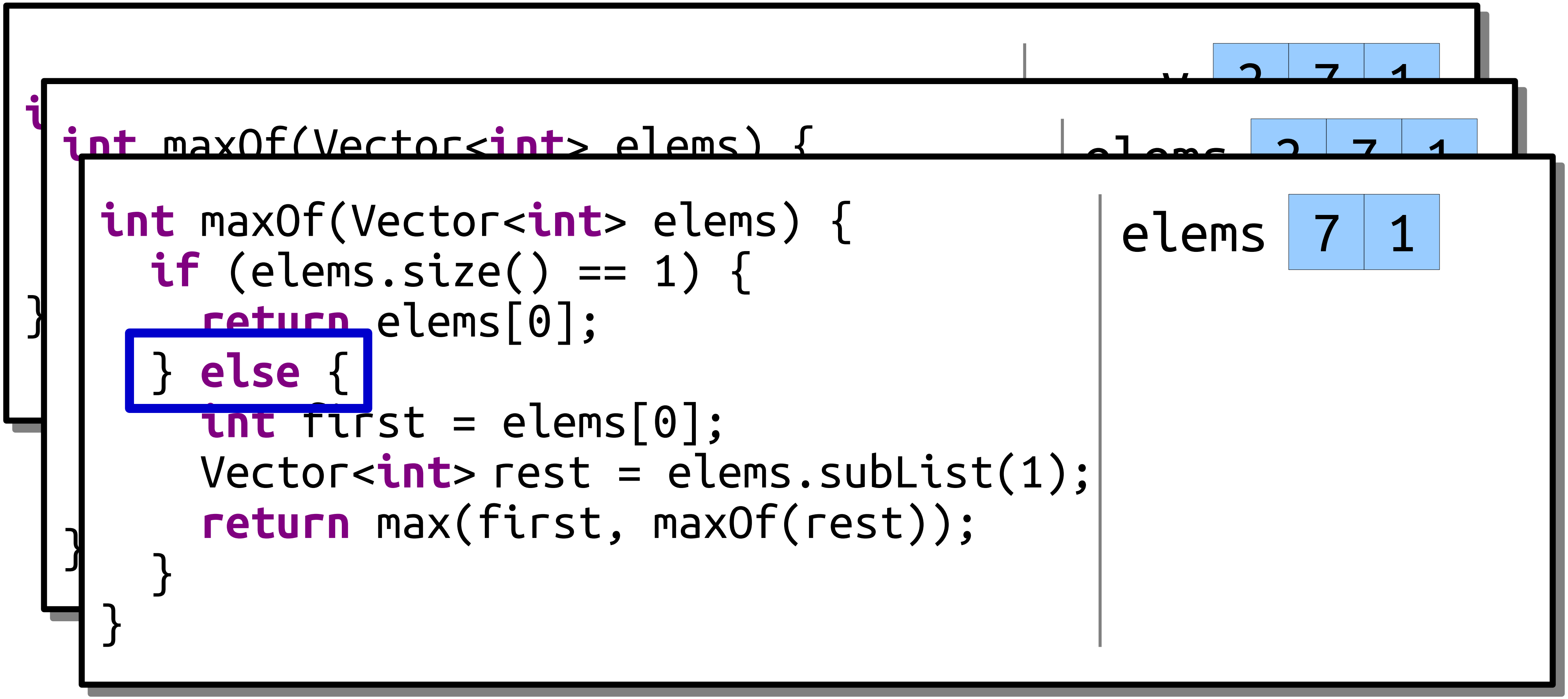
Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int first = elems[0];
    Vector<int> rest = elems.subList(1);
    return max(first, maxOf(rest));
  }
}
```

elems	7	1

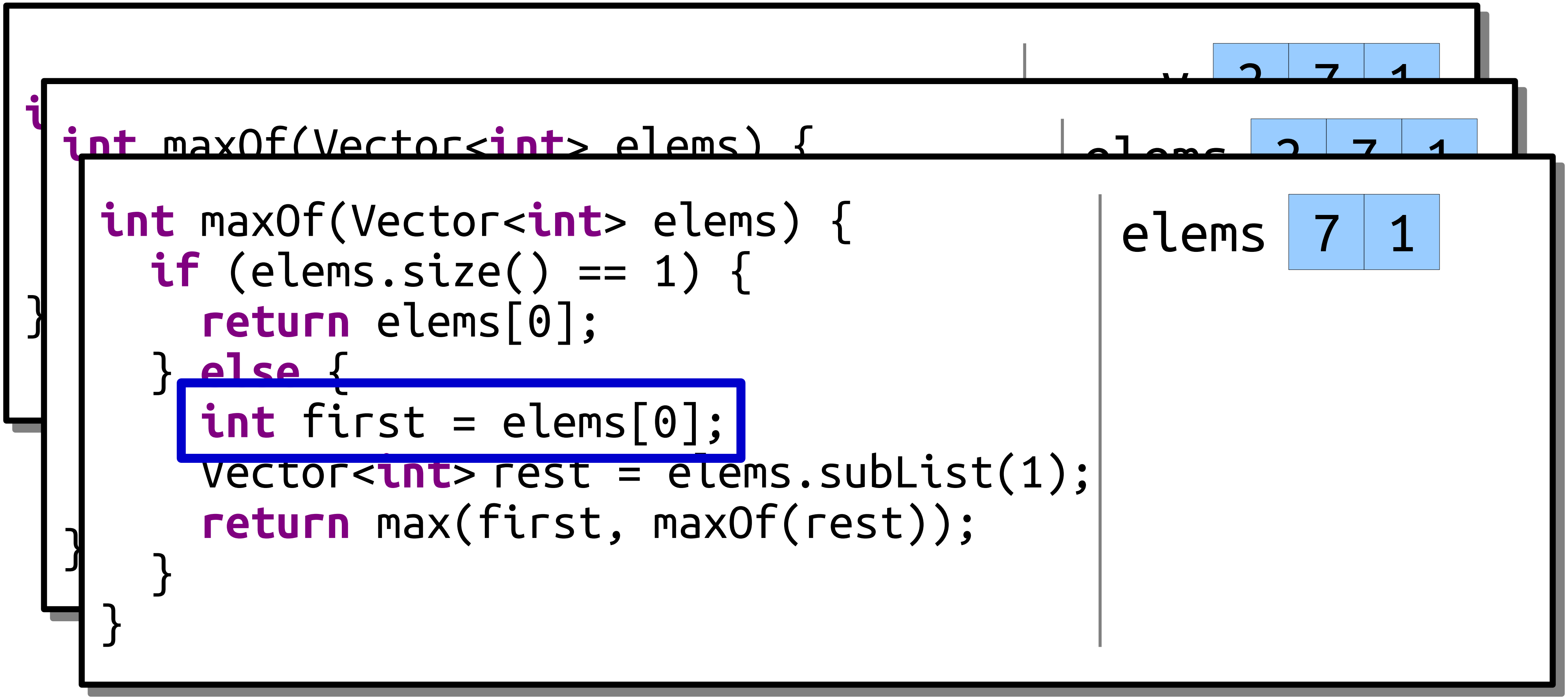
Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int first = elems[0];
    Vector<int> rest = elems.subList(1);
    return max(first, maxOf(rest));
  }
}
```

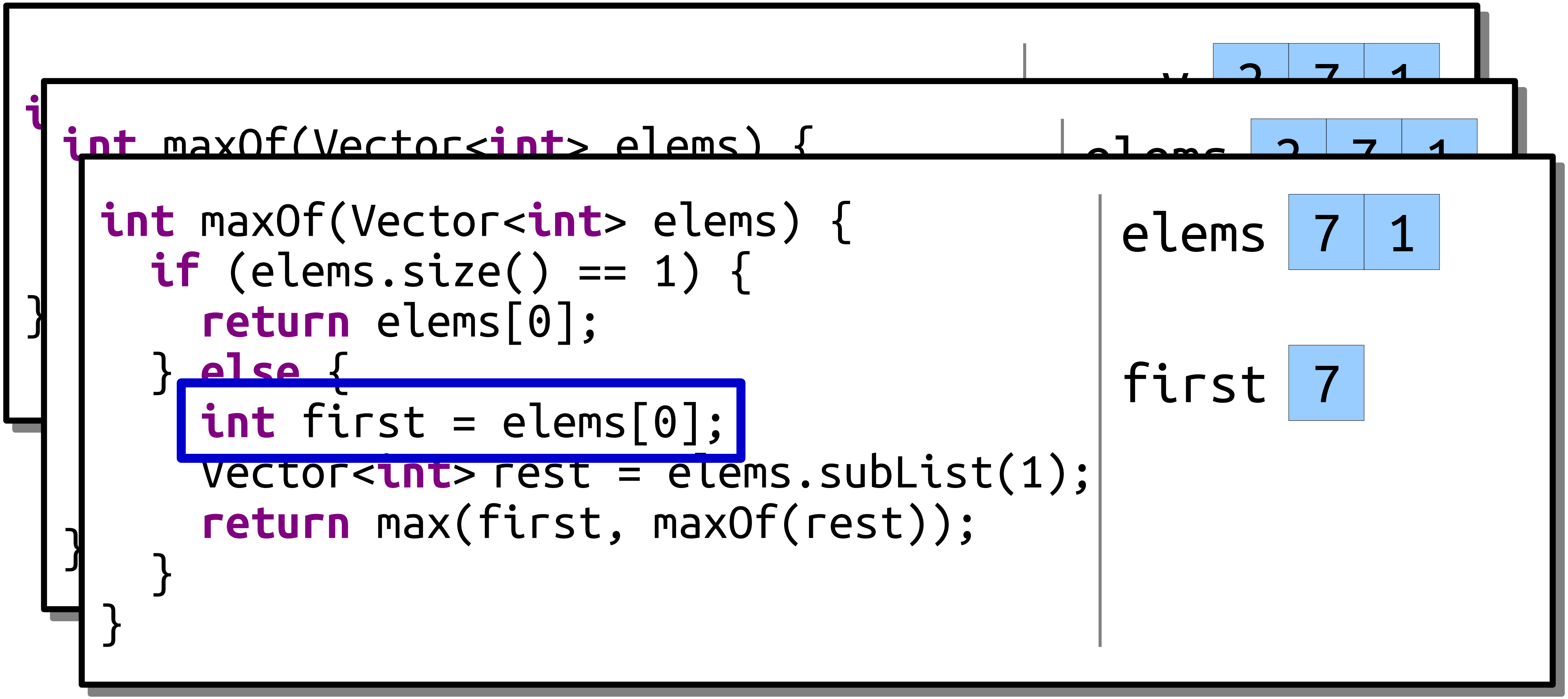
elems	7	1

Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int first = elems[0];
    Vector<int> rest = elems.subList(1);
    return max(first, maxOf(rest));
  }
}
```

Tracing the Recursion

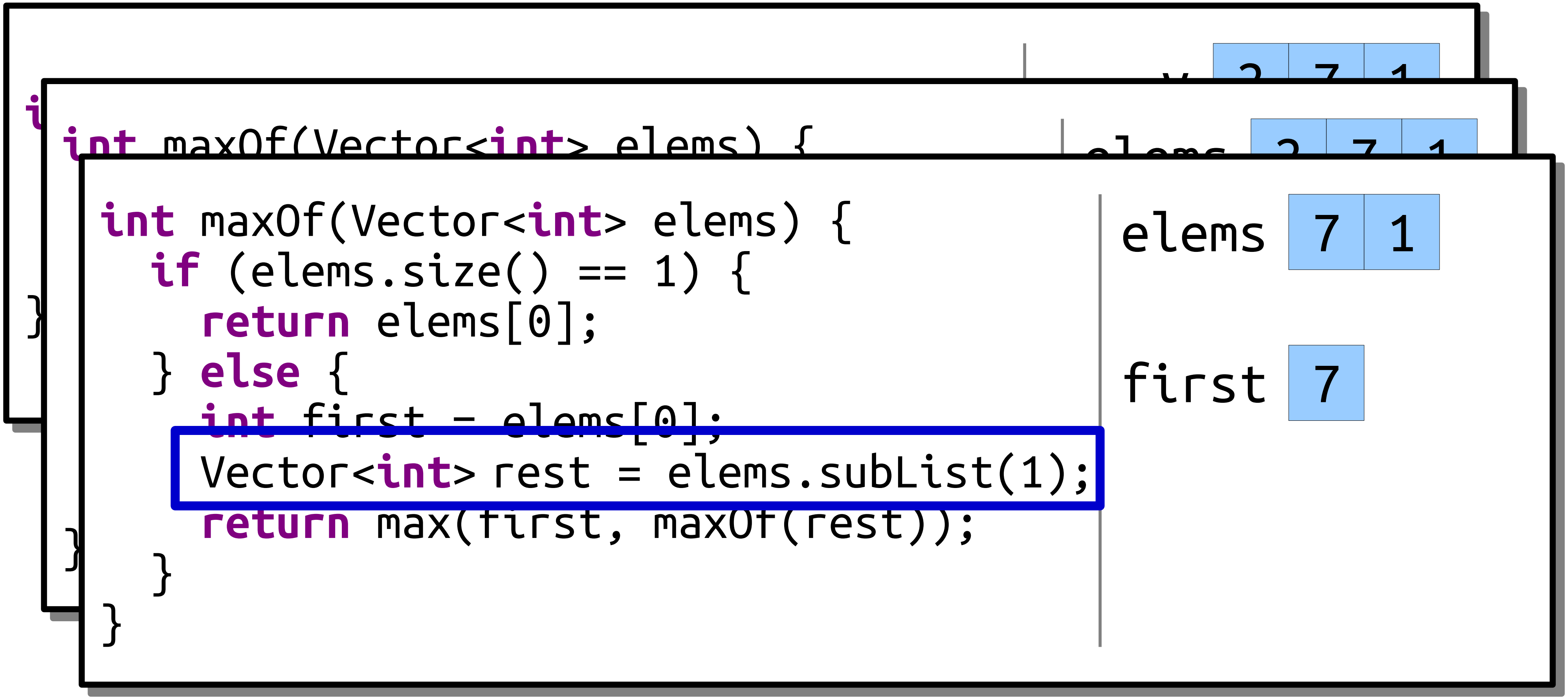


```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int first = elems[0];
    Vector<int> rest = elems.subList(1);
    return max(first, maxOf(rest));
  }
}
```

elems 7 1

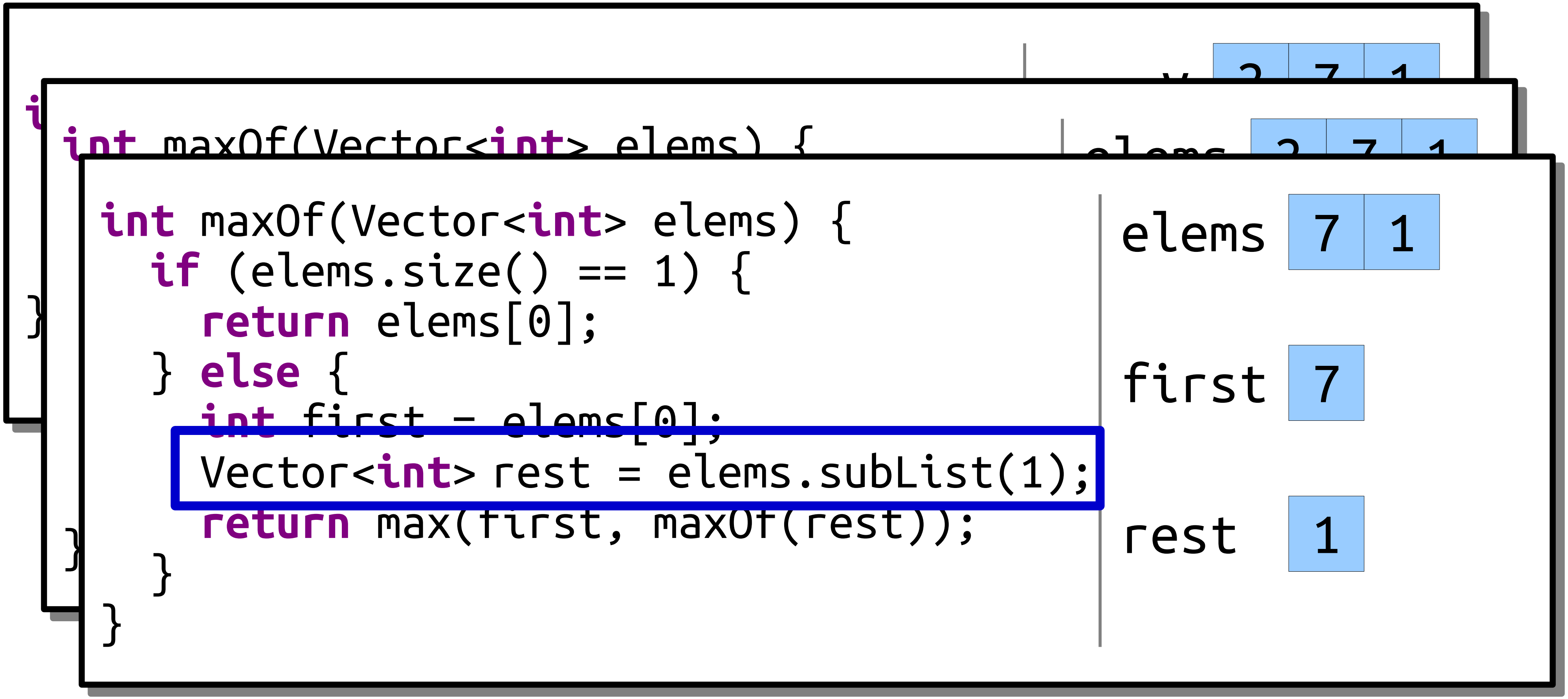
first 7

Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int first = elems[0];
    Vector<int> rest = elems.subList(1);
    return max(first, maxOf(rest));
  }
}
```

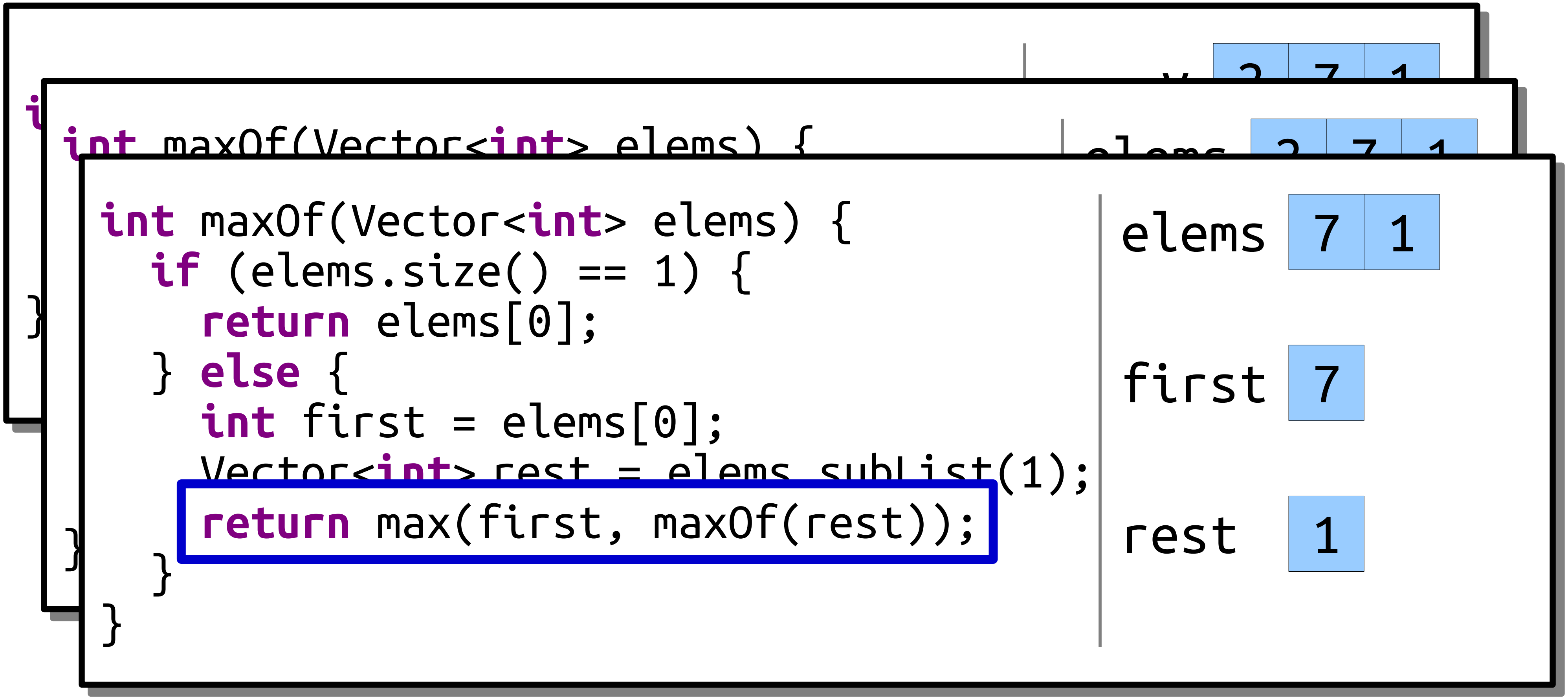
Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int first = elems[0];
    Vector<int> rest = elems.subList(1);
    return max(first, maxOf(rest));
  }
}
```

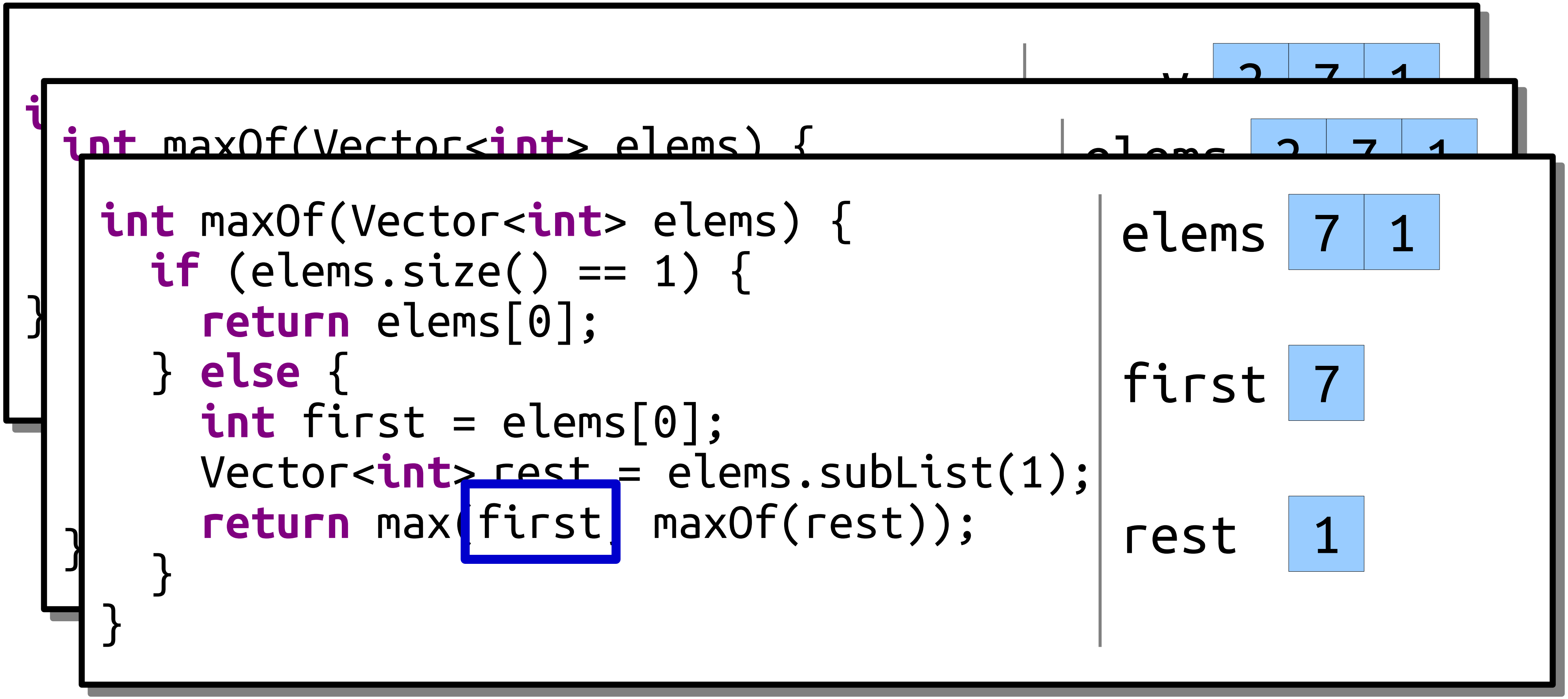
elems	7	1
first	7	
rest	1	

Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int first = elems[0];
    Vector<int> rest = elems.sublist(1);
    return max(first, maxOf(rest));
  }
}
```

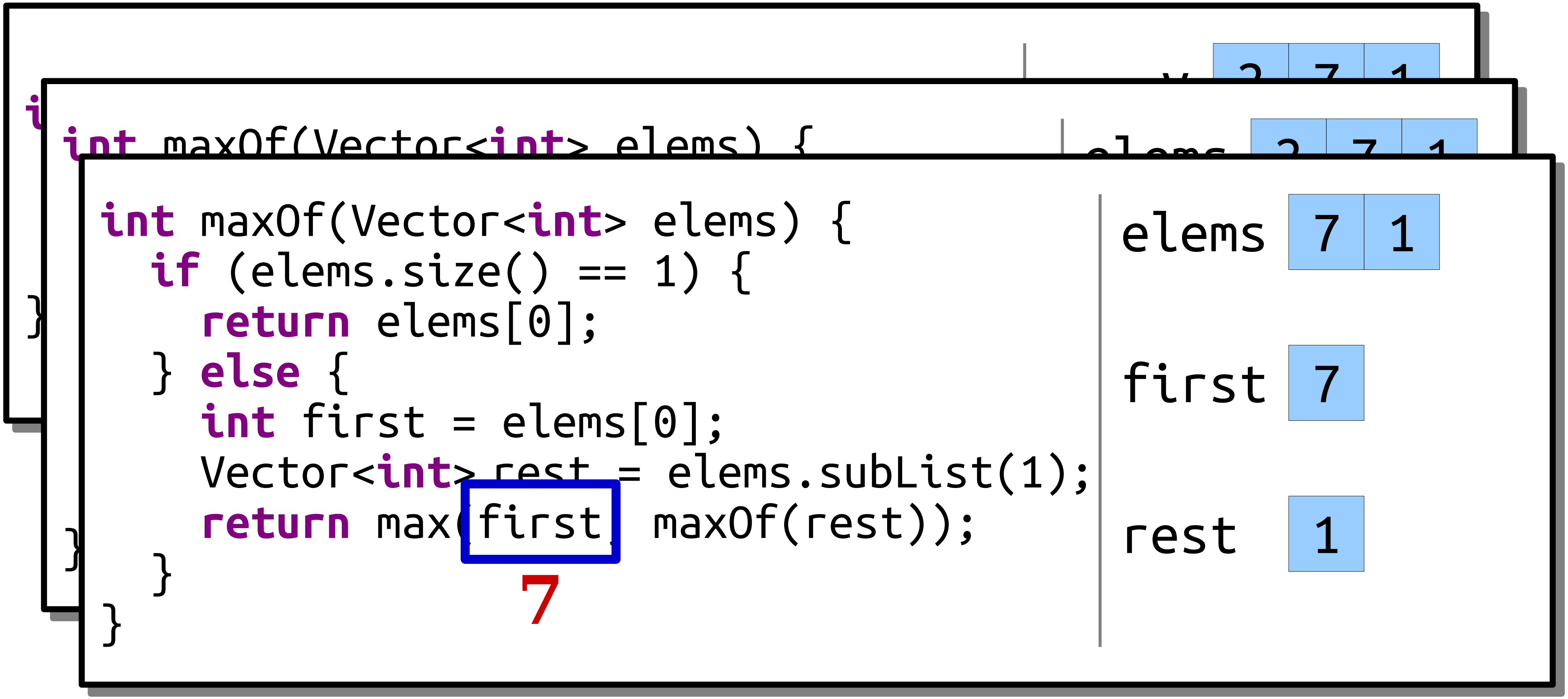
Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int first = elems[0];
    Vector<int> rest = elems.subList(1);
    return max(first, maxOf(rest));
  }
}
```

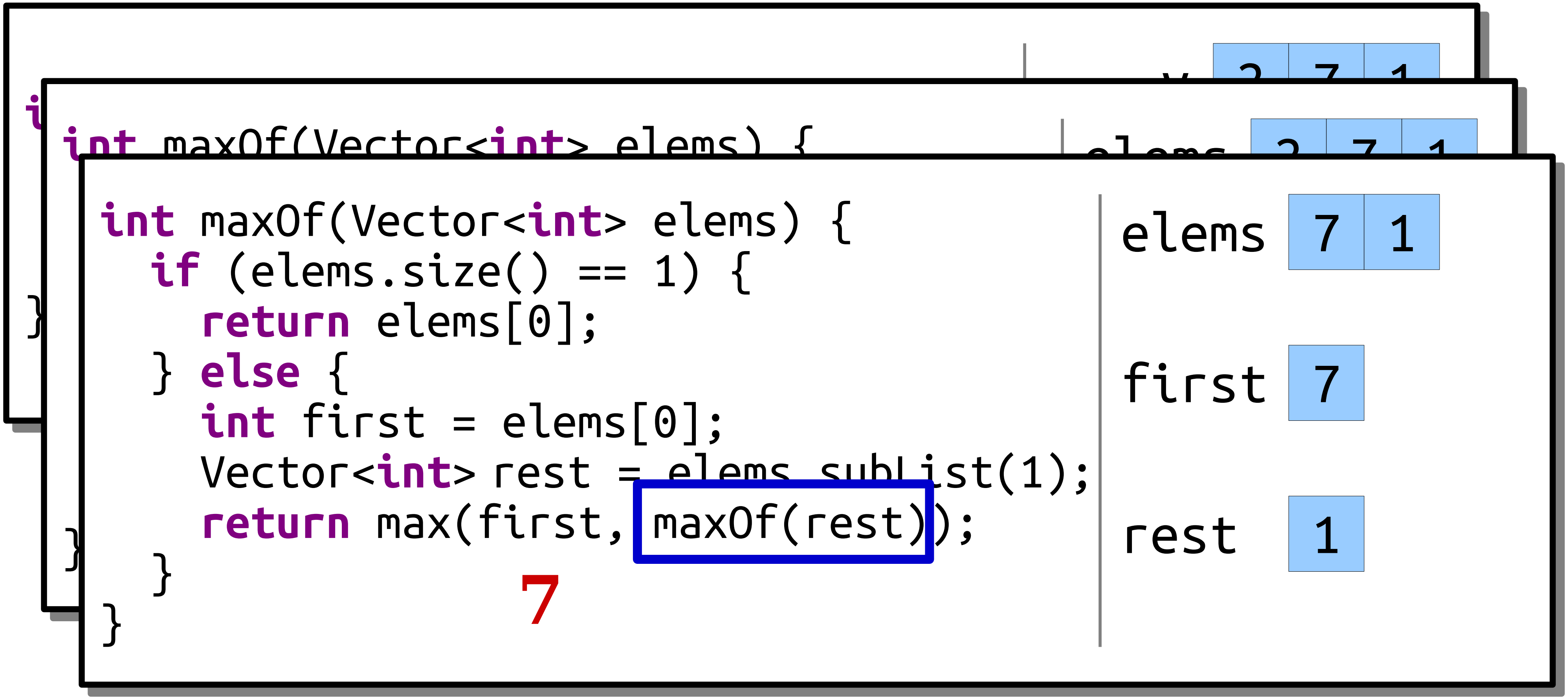
elems	7	1
first	7	
rest	1	

Tracing the Recursion



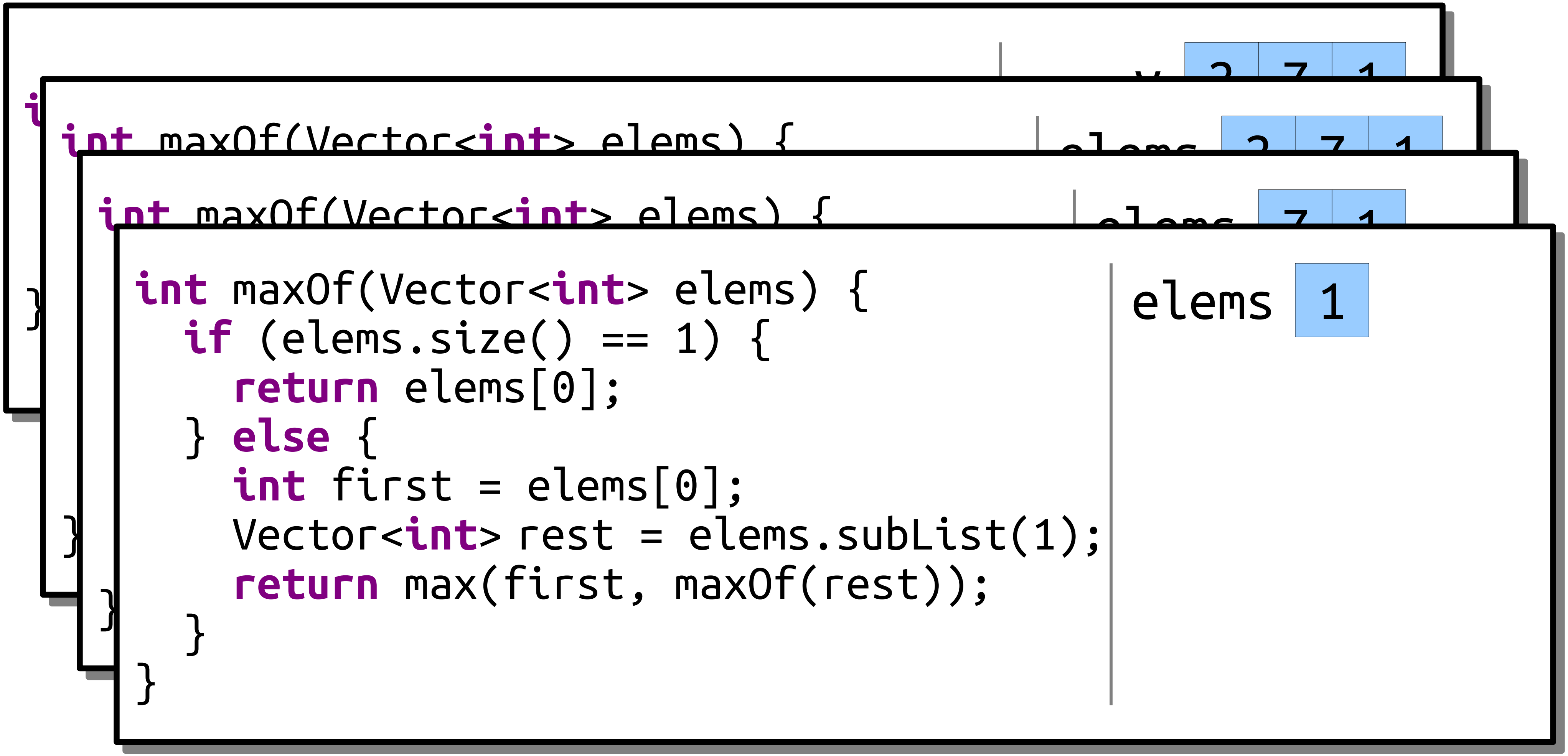
```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int first = elems[0];
    Vector<int> rest = elems.subList(1);
    return max(first, maxOf(rest));
  }
}
```

Tracing the Recursion



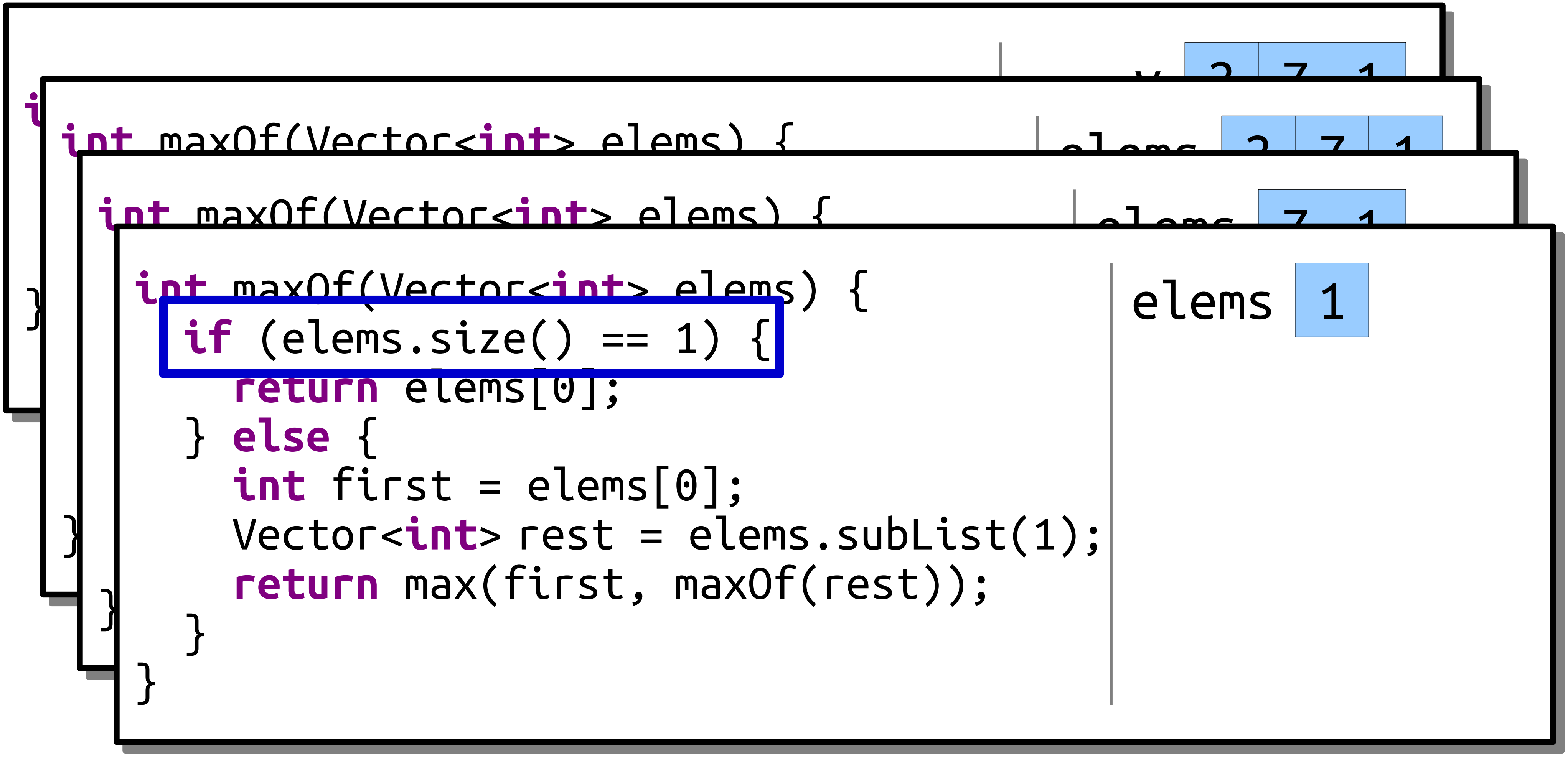
```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int first = elems[0];
    Vector<int> rest = elems.sublist(1);
    return max(first, maxOf(rest));
  }
}
```

Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int first = elems[0];
    Vector<int> rest = elems.subList(1);
    return max(first, maxOf(rest));
  }
}
```

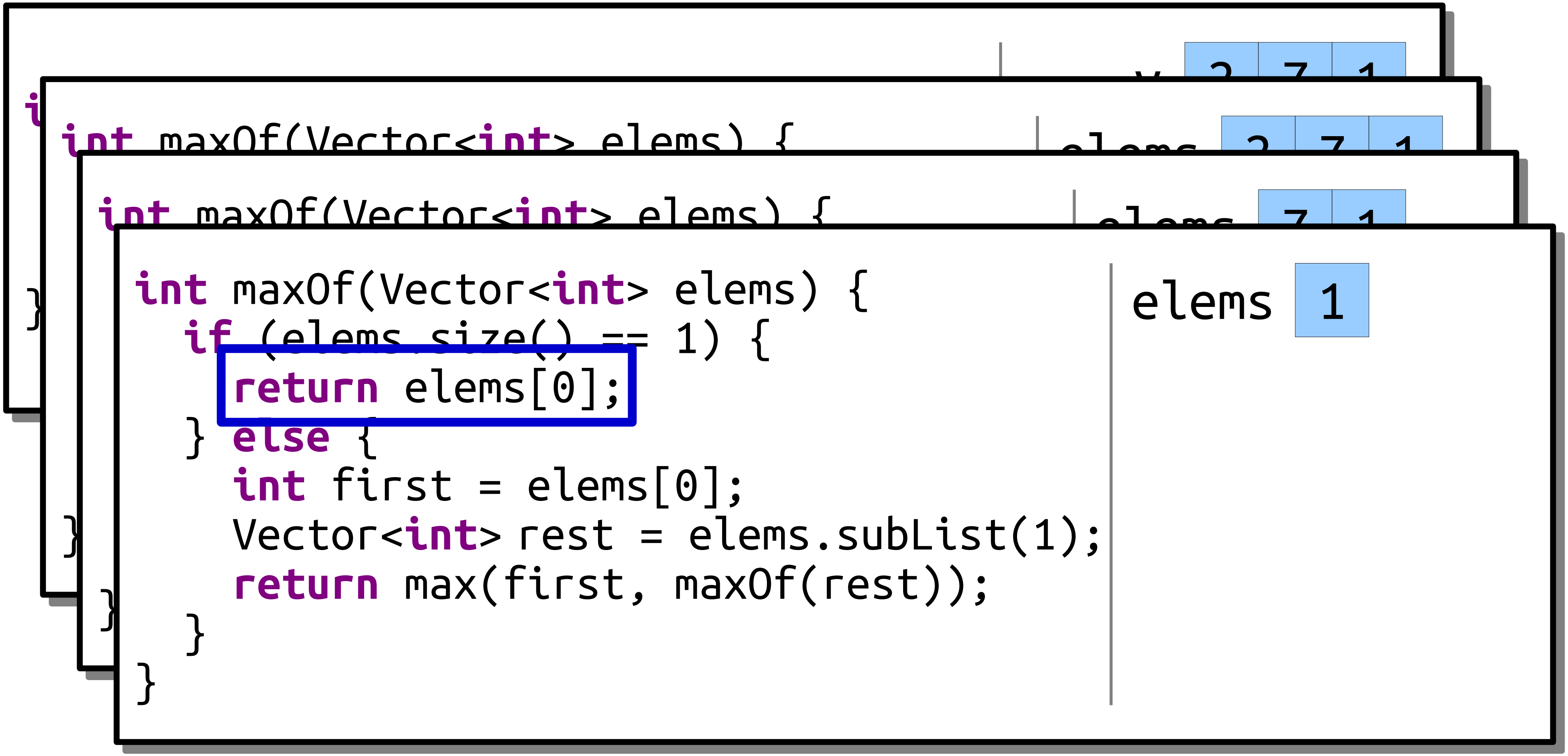
Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int first = elems[0];
    Vector<int> rest = elems.subList(1);
    return max(first, maxOf(rest));
  }
}
```

elems 1

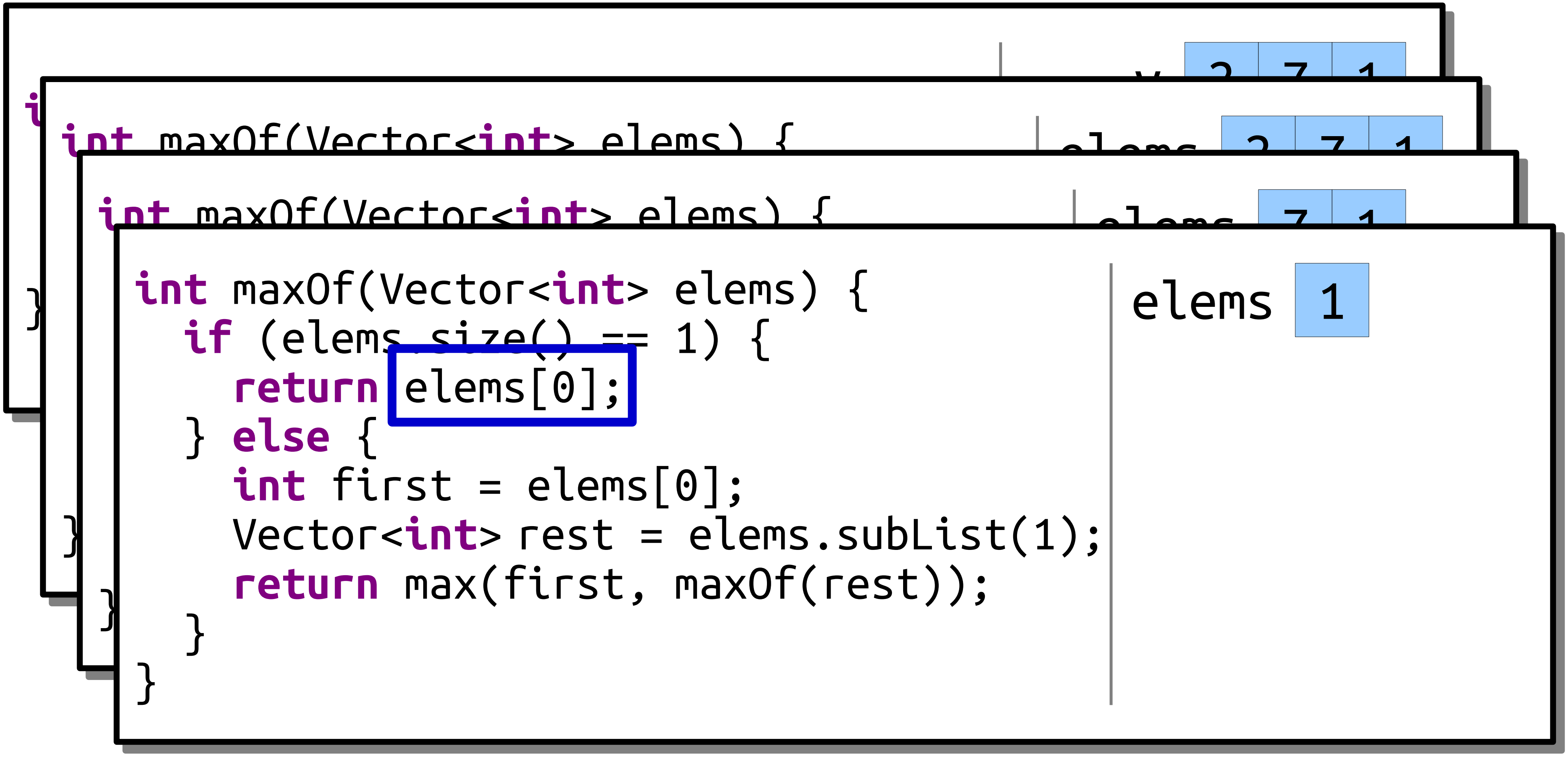
Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int first = elems[0];
    Vector<int> rest = elems.subList(1);
    return max(first, maxOf(rest));
  }
}
```

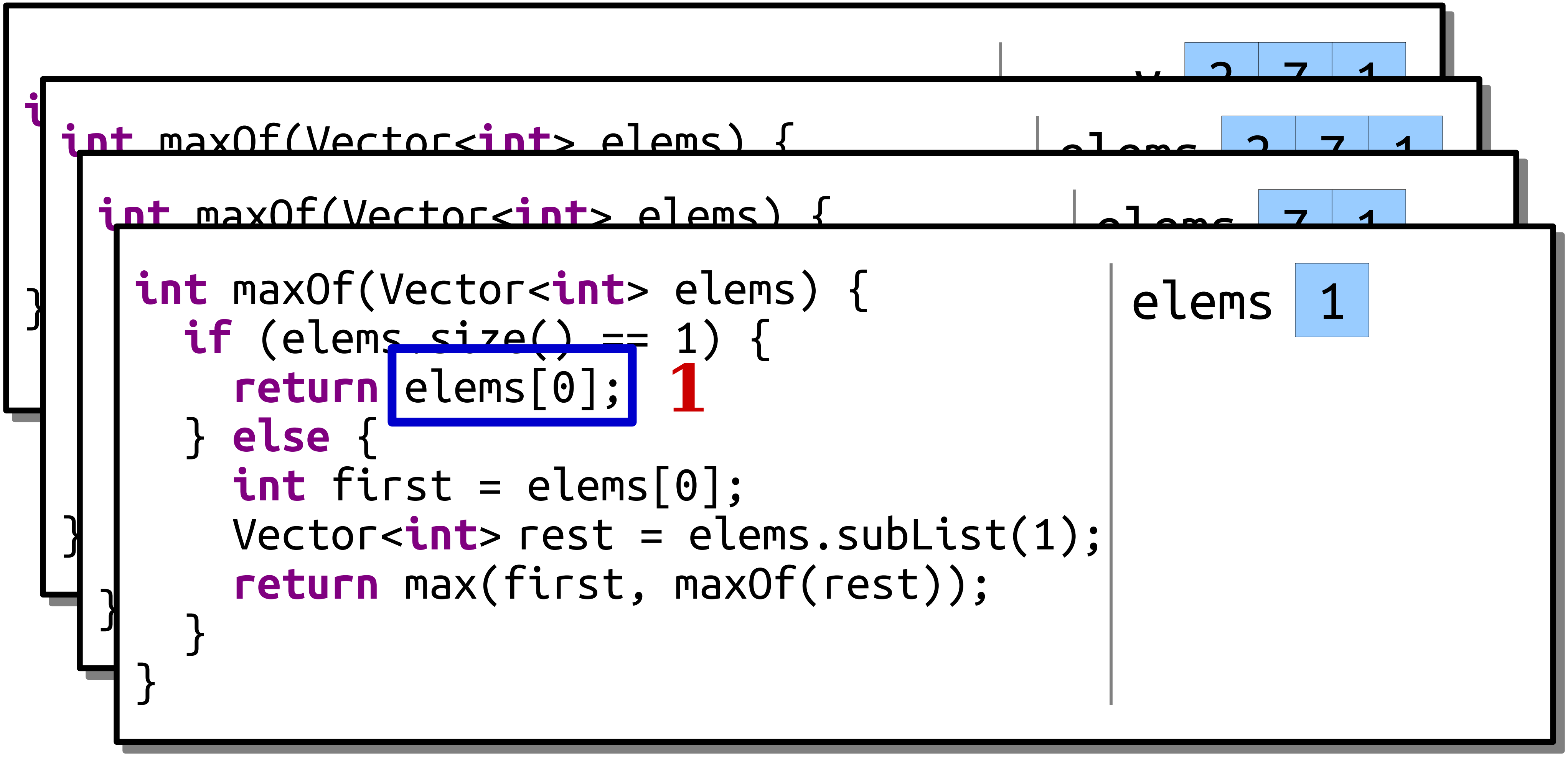
elems | 1

Tracing the Recursion



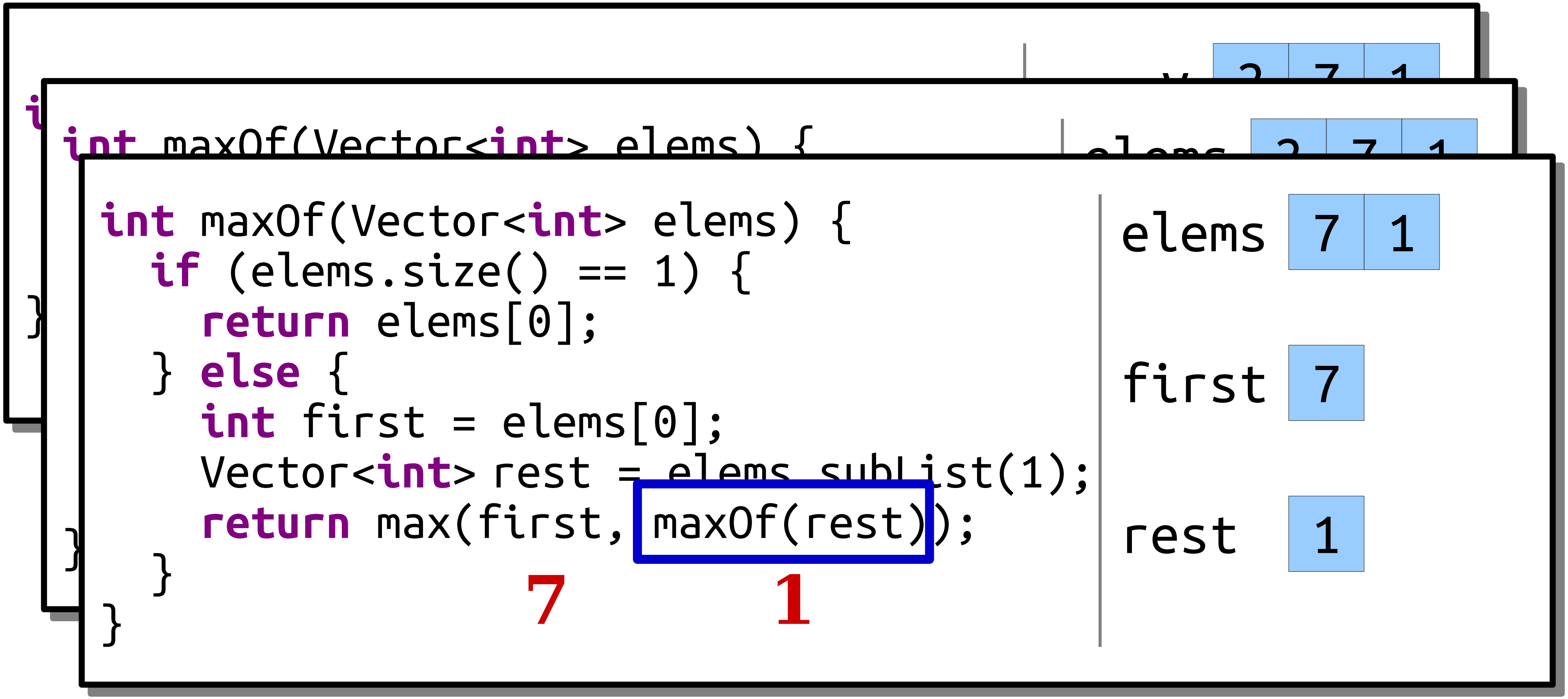
```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int first = elems[0];
    Vector<int> rest = elems.subList(1);
    return max(first, maxOf(rest));
  }
}
```

Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int first = elems[0];
    Vector<int> rest = elems.subList(1);
    return max(first, maxOf(rest));
  }
}
```

Tracing the Recursion



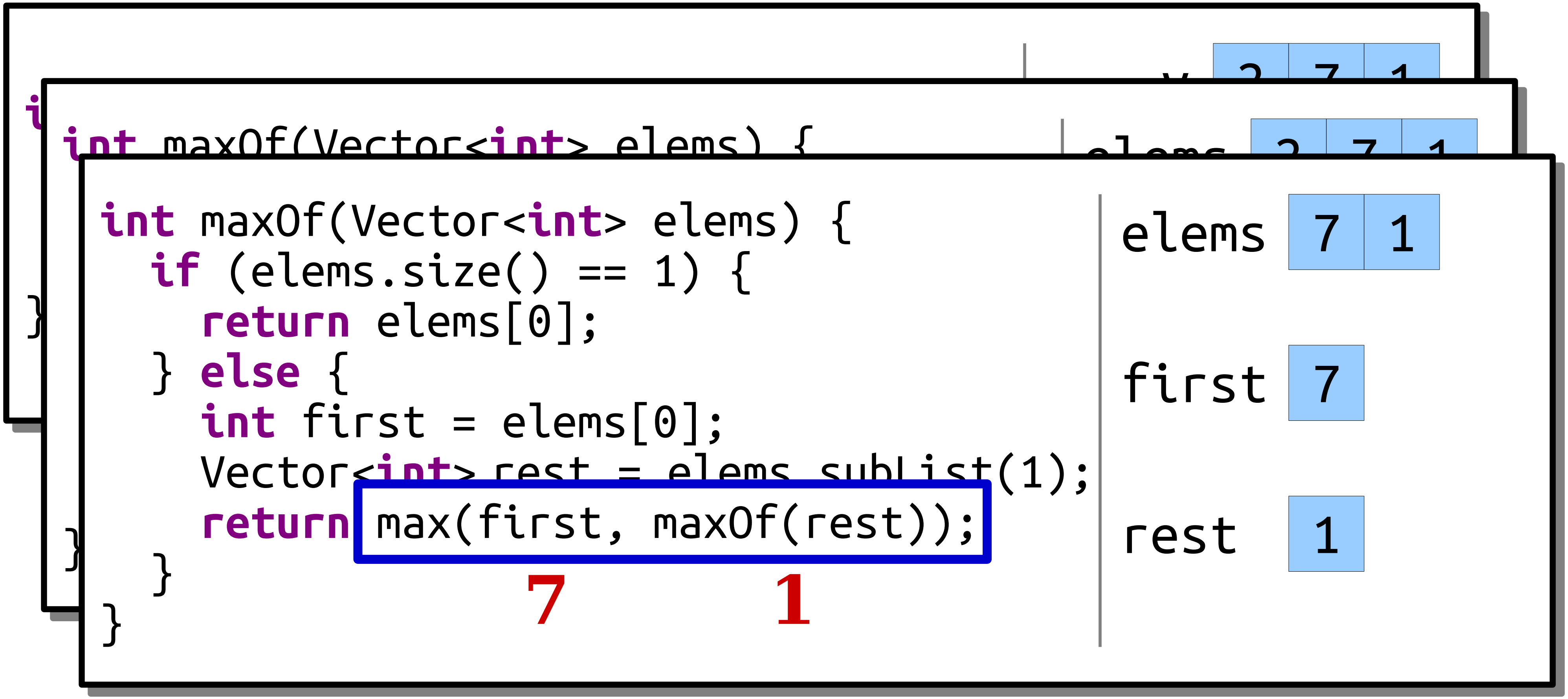
```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int first = elems[0];
    Vector<int> rest = elems.sublist(1);
    return max(first, maxOf(rest));
  }
}
```

elems 7 1

first 7

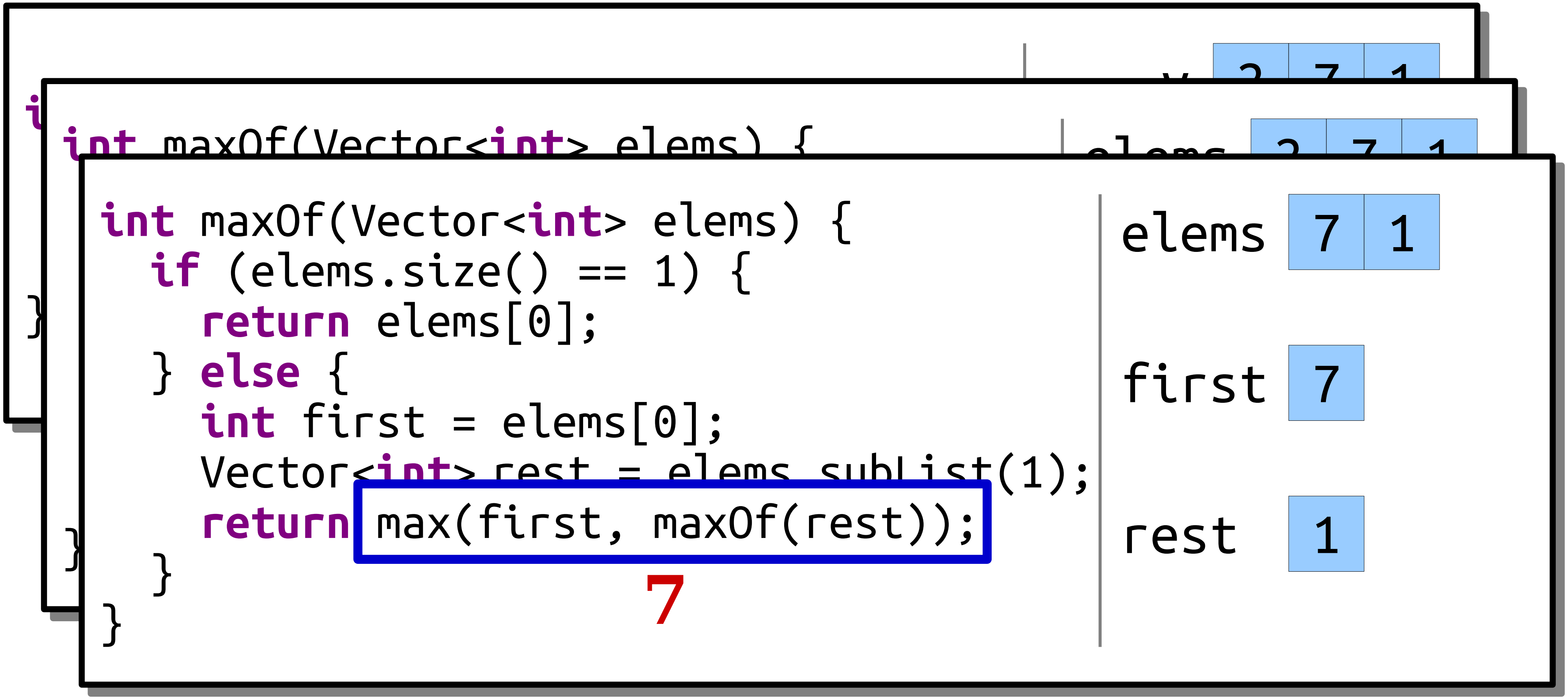
rest 1

Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int first = elems[0];
    Vector<int> rest = elems.sublist(1);
    return max(first, maxOf(rest));
  }
}
```

Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int first = elems[0];
    Vector<int> rest = elems.sublist(1);
    return max(first, maxOf(rest));
  }
}
```

7

elems	7	1
first	7	
rest	1	

Tracing the Recursion

v 2 7 1

```
int maxOf(Vector<int> elems) {
```

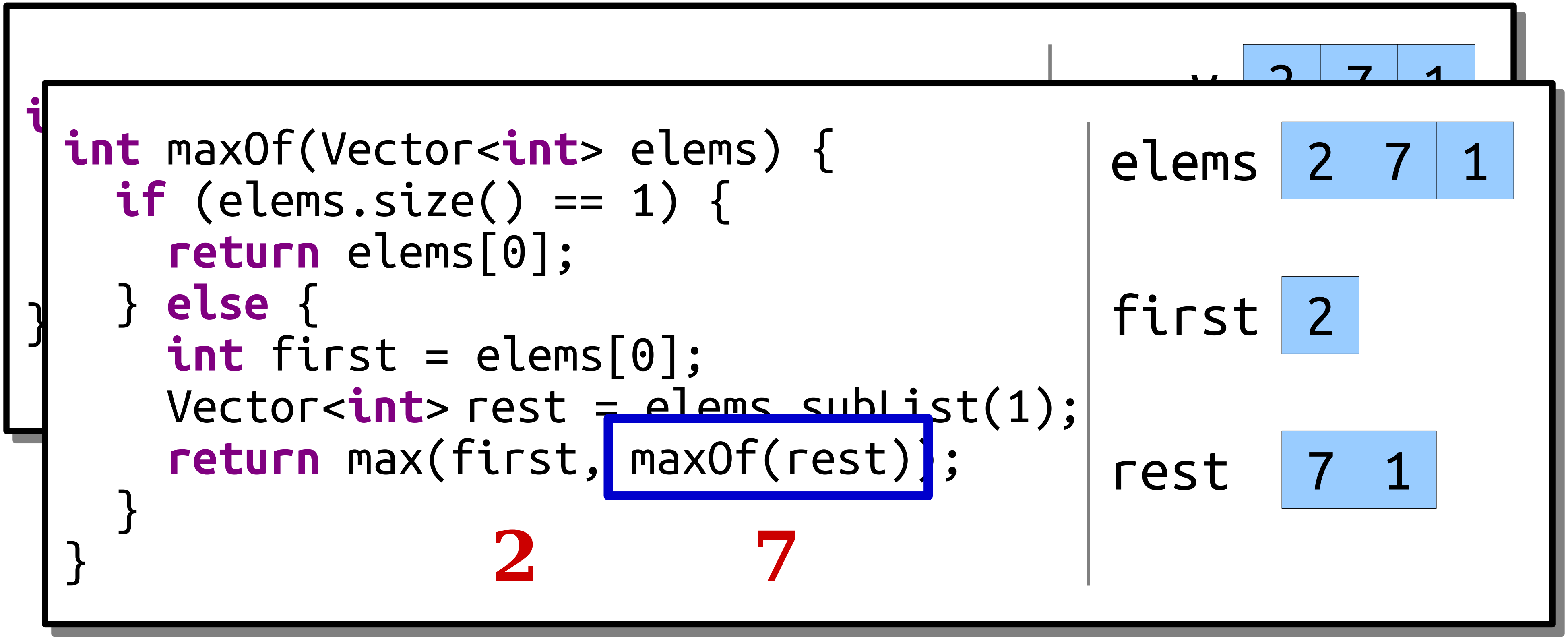
elems 2 7 1

```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int first = elems[0];
    Vector<int> rest = elems.subList(1);
    return max(first, maxOf(rest));
  }
}
```

7

elems	7	1
first	7	
rest	1	

Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int first = elems[0];
    Vector<int> rest = elems.sublist(1);
    return max(first, maxOf(rest));
  }
}
```

elems	2	7	1
first	2		
rest	7	1	

Tracing the Recursion

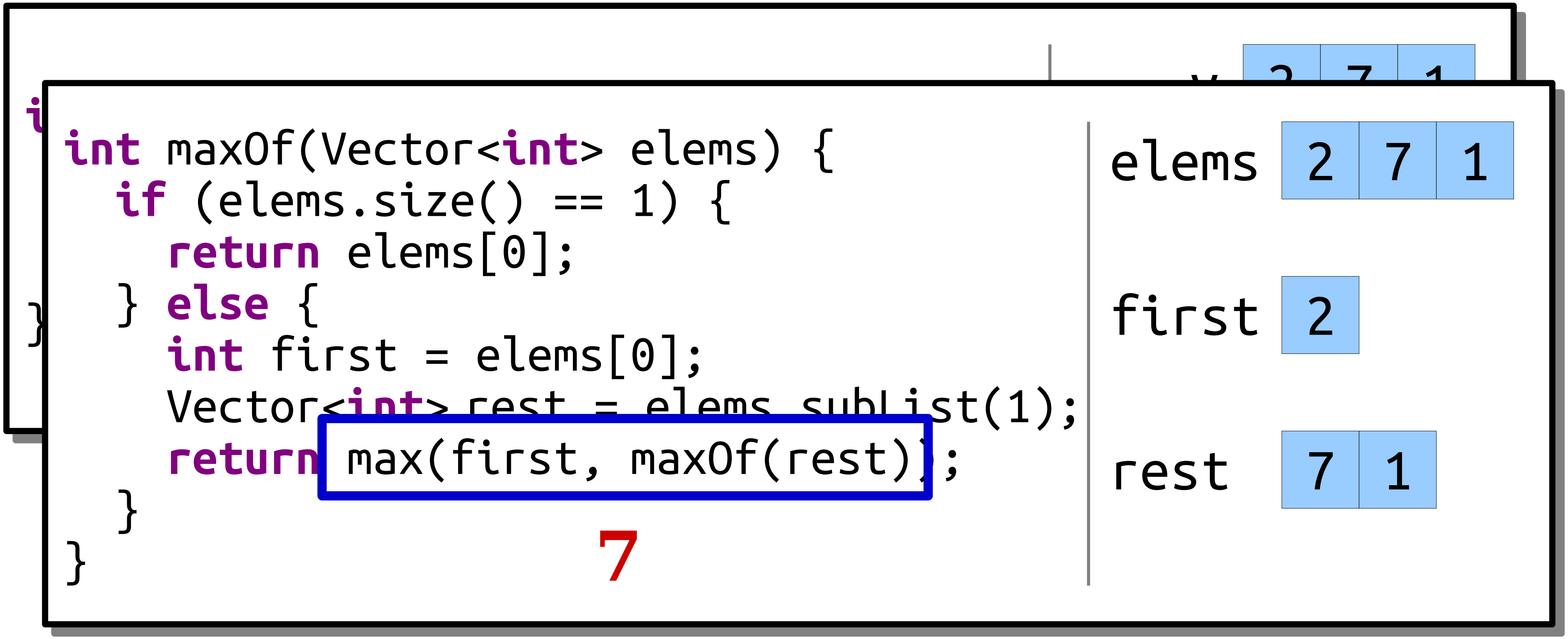
```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int first = elems[0];
    Vector<int> rest = elems.subList(1);
    return max(first, maxOf(rest));
  }
}
                 2          7
```

elems 2 7 1

first 2

rest 7 1

Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int first = elems[0];
    Vector<int> rest = elems.subList(1);
    return max(first, maxOf(rest));
  }
}
```

7

elems	2	7	1
first	2		
rest	7	1	

Tracing the Recursion

```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int first = elems[0];
    Vector<int> rest = elems.subList(1);
    return max(first, maxOf(rest));
  }
}
```

7

elems 2 7 1

first 2

rest 7 1

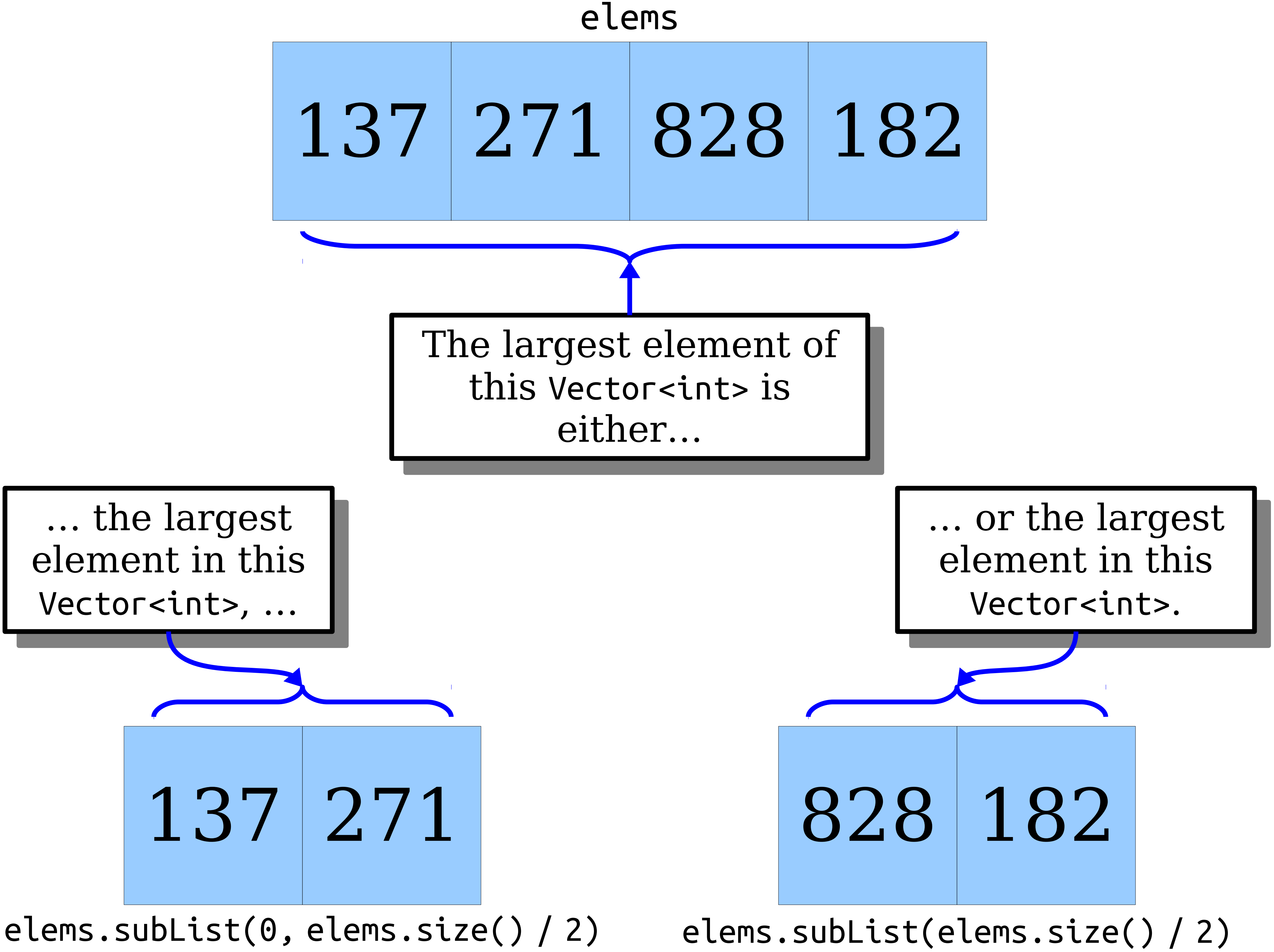
Tracing the Recursion

```
int main() {
   Vector<int> v = { 2, 7, 1 };
   cout << maxOf(v) << endl;
   return 0;
}
```

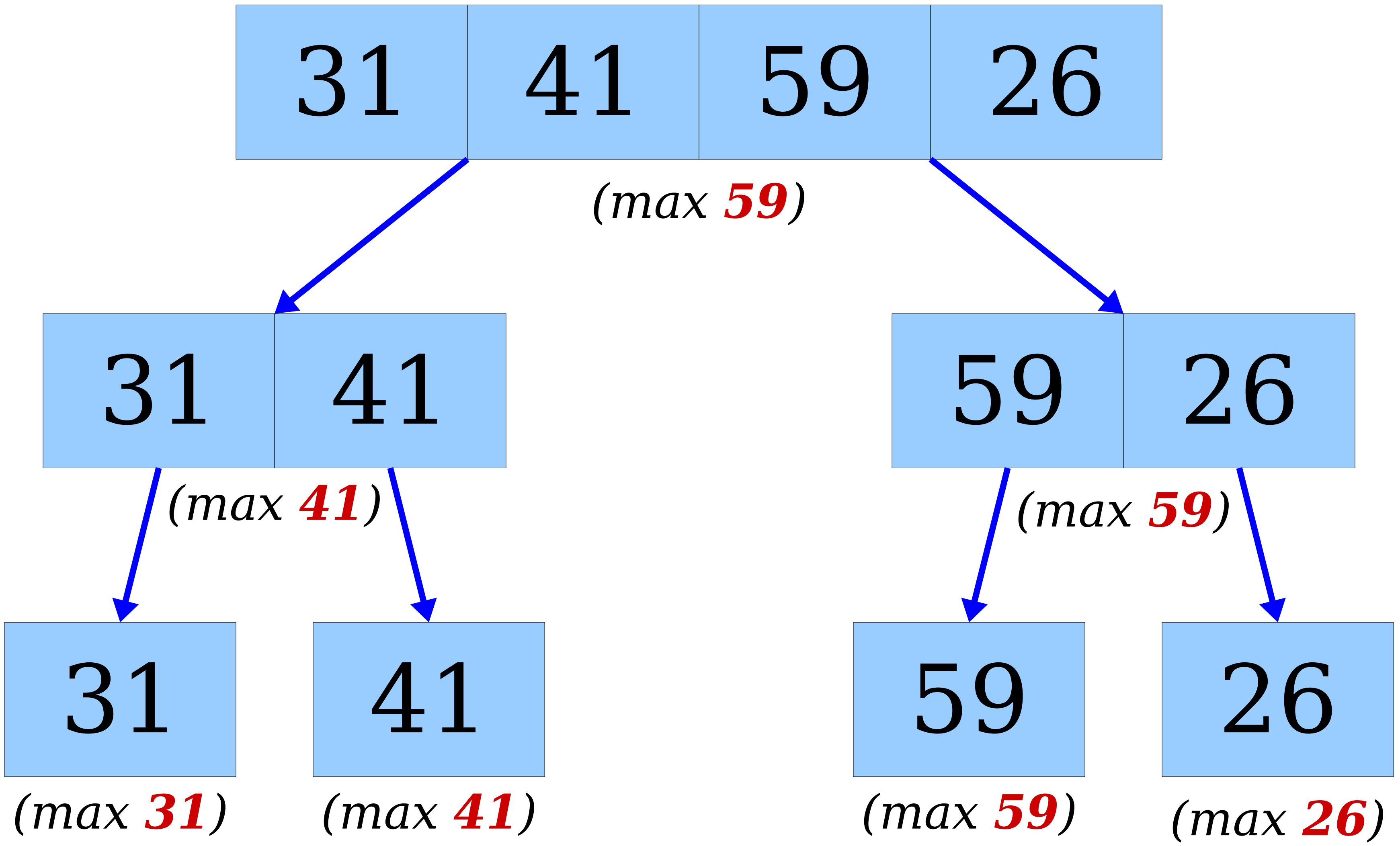
7

v	2	7	1

A Different Approach



`maxOf` as a Tournament



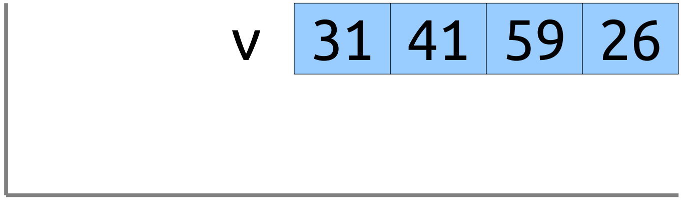
Tracing the Recursion

```
int main() {
   Vector<int> v = { 31, 41, 59, 26 };
   cout << maxOf(v) << endl;
   return 0;
}
```

Tracing the Recursion

```
int main() {
    Vector<int> v = { 31, 41, 59, 26 };
    cout << maxOf(v) << endl;
    return 0;
}
```

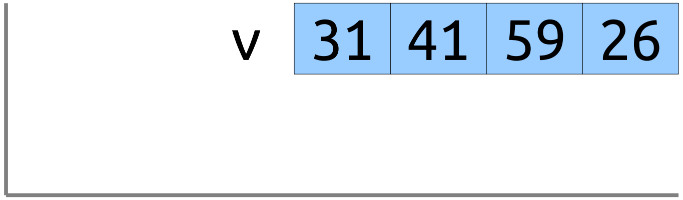
Tracing the Recursion



v	31	41	59	26

```
int main() {
    Vector<int> v = { 31, 41, 59, 26 };
    cout << maxOf(v) << endl;
    return 0;
}
```

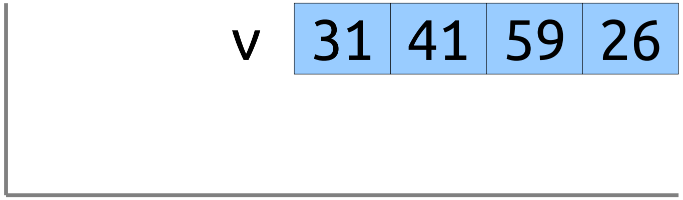
Tracing the Recursion



v	31	41	59	26

```
int main() {
    Vector<int> v = { 31, 41, 59, 26 };
    cout << maxOf(v) << endl;
    return 0;
}
```

Tracing the Recursion



v	31	41	59	26

```
int main() {
    Vector<int> v = { 31, 41, 59, 26 };
    cout << maxOf(v) << endl;
    return 0;
}
```

Tracing the Recursion

v 31 41 59 26

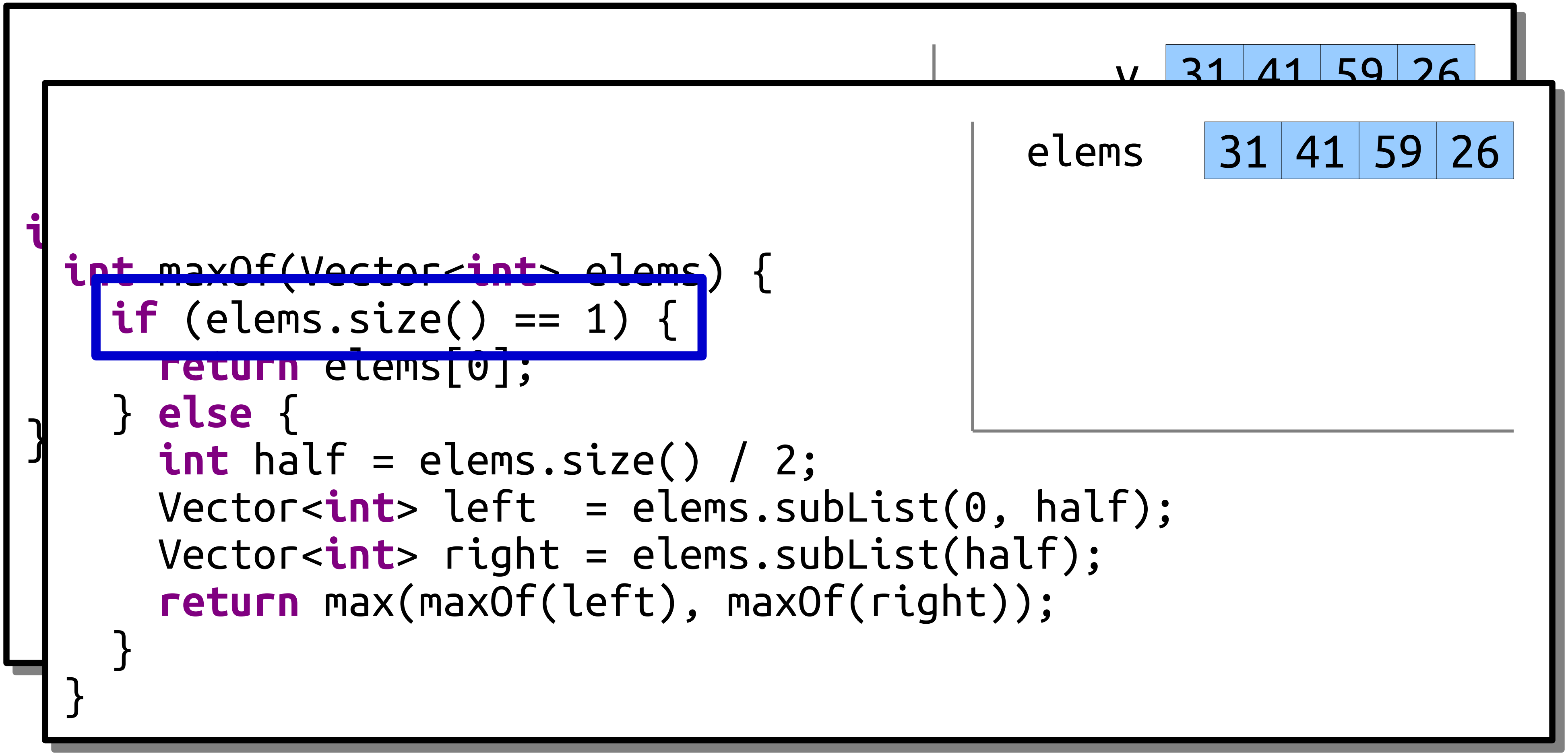
i

}

elems	31	41	59	26

```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

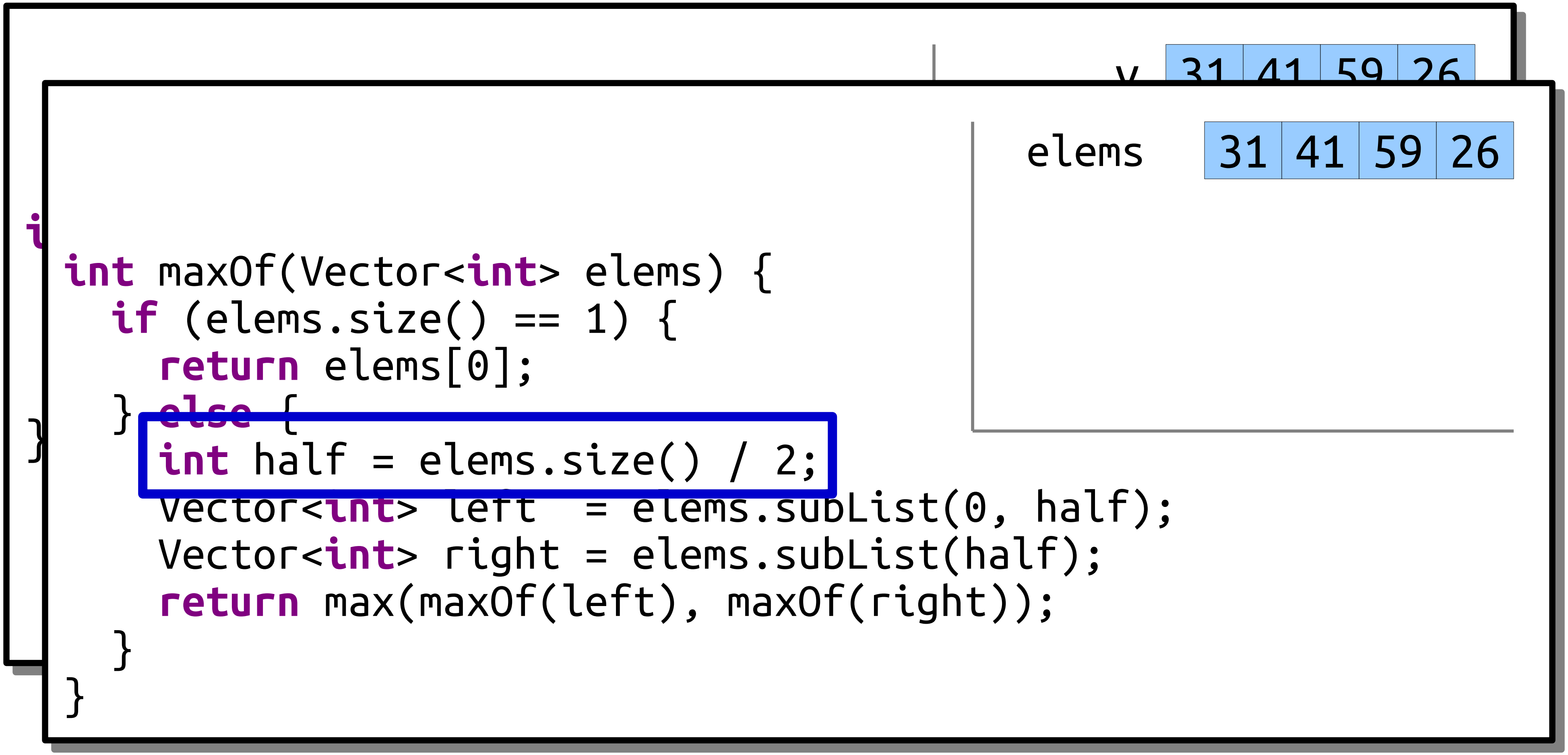
Tracing the Recursion

v 31 41 59 26

elems 31 41 59 26

```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

Tracing the Recursion



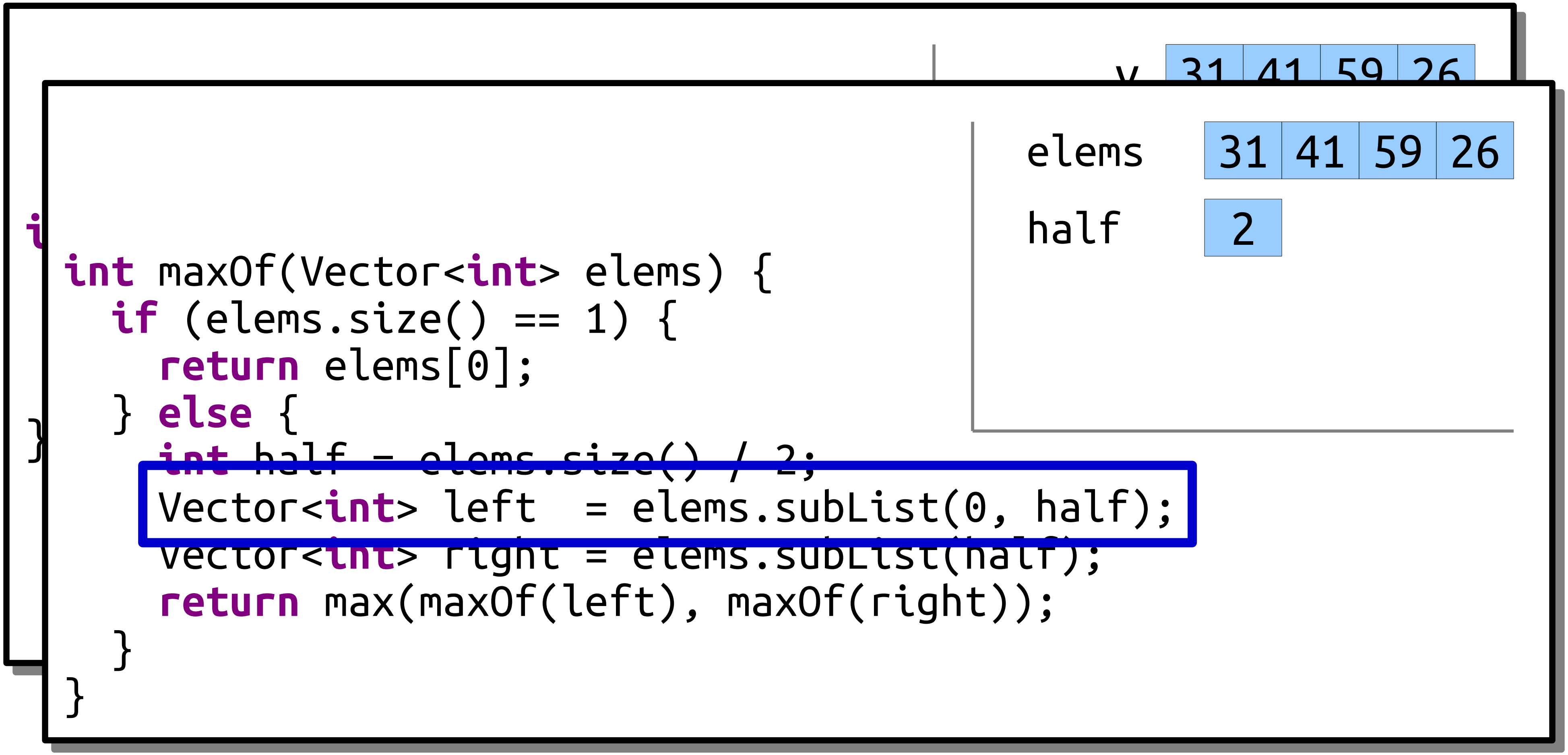
```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

Tracing the Recursion

```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

elems	31	41	59	26
half	2			

Tracing the Recursion

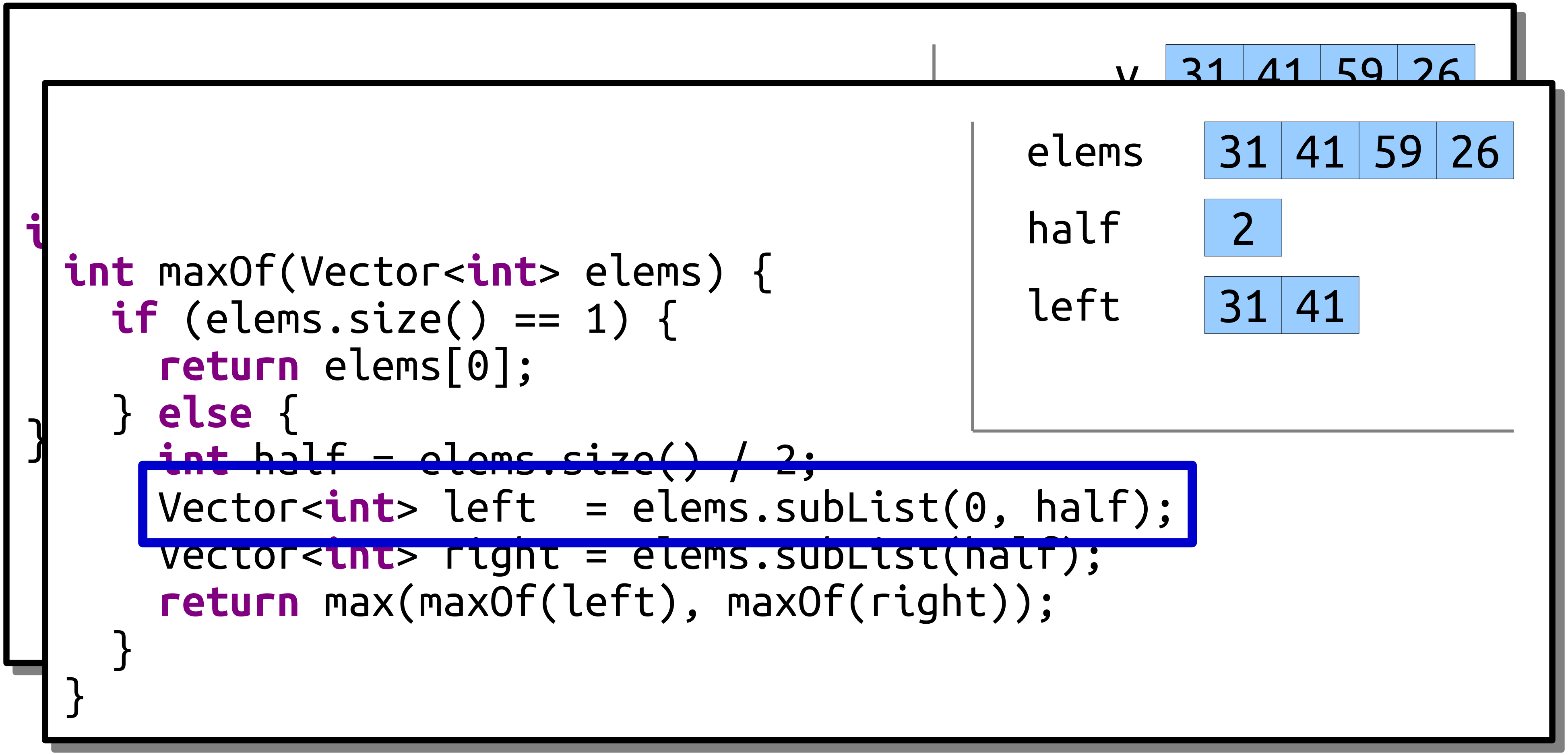


v	31	41	59	26

```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

elems	31	41	59	26
half	2			

Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

elems	31	41	59	26
half	2			
left	31	41		

Tracing the Recursion

v	31	41	59	26

```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

elems	31	41	59	26
half	2			
left	31	41		

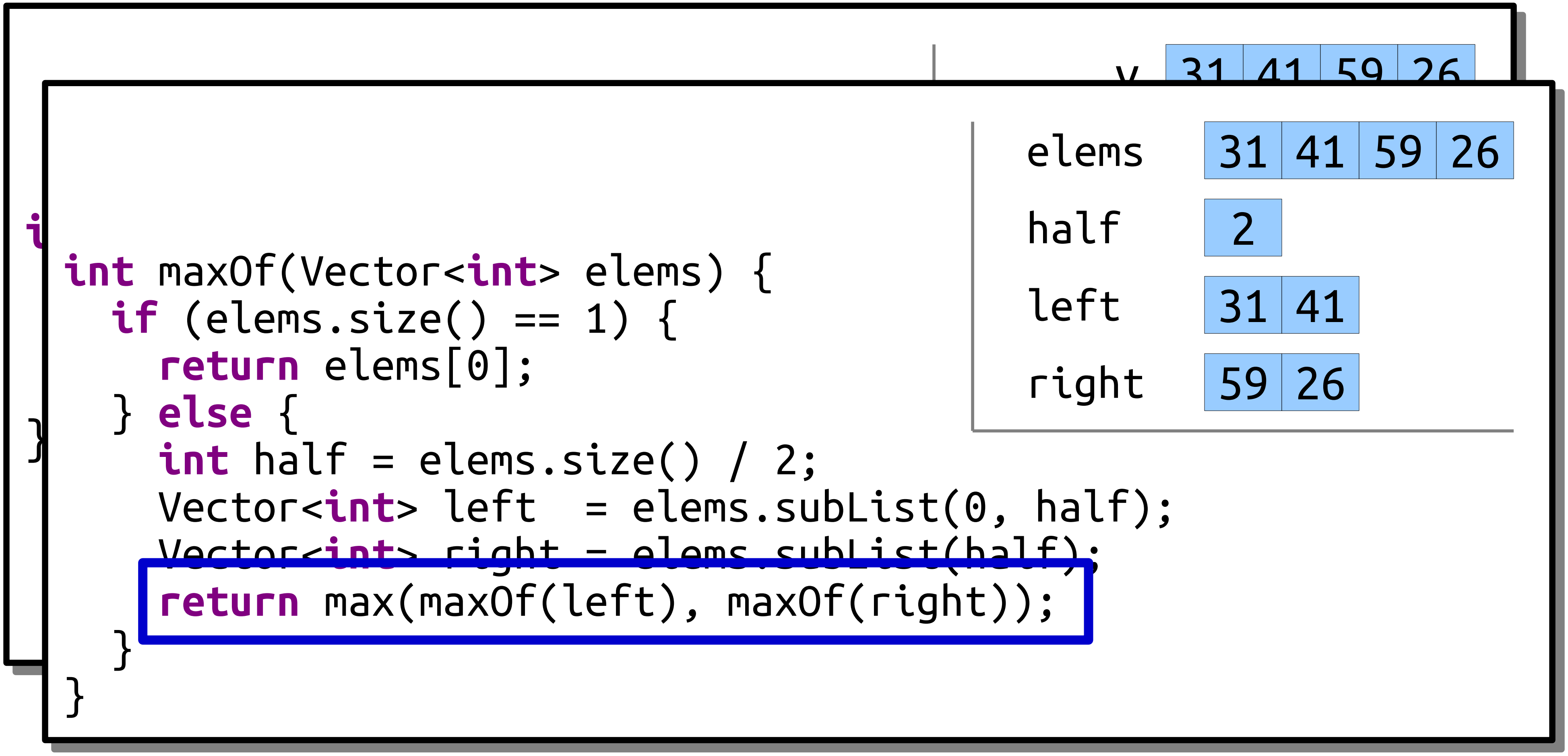
Tracing the Recursion

v 31 41 59 26

```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

elems	31	41	59	26
half	2			
left	31	41		
right	59	26		

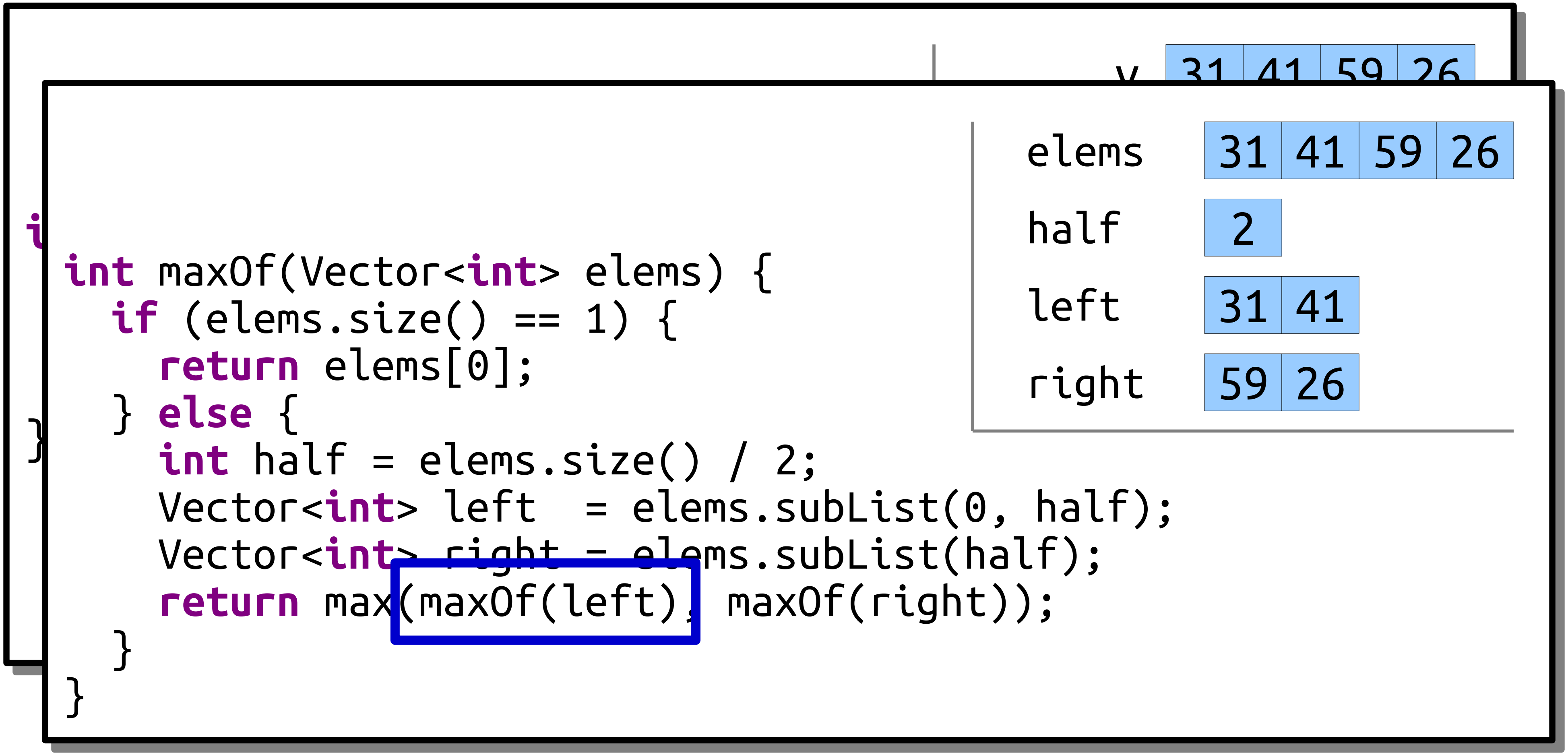
Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

elems	31	41	59	26
half	2			
left	31	41		
right	59	26		

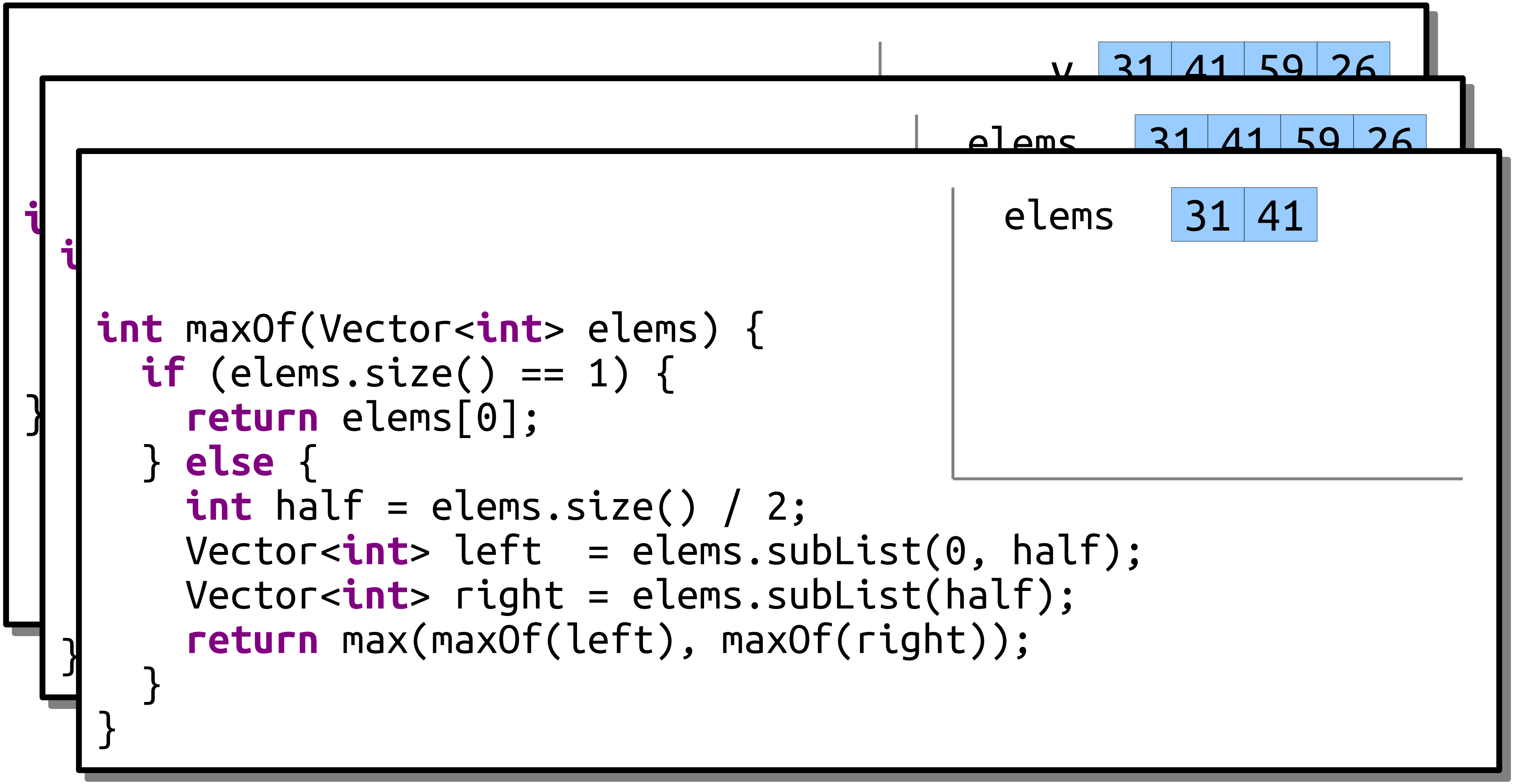
Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

elems	31	41	59	26
half	2			
left	31	41		
right	59	26		

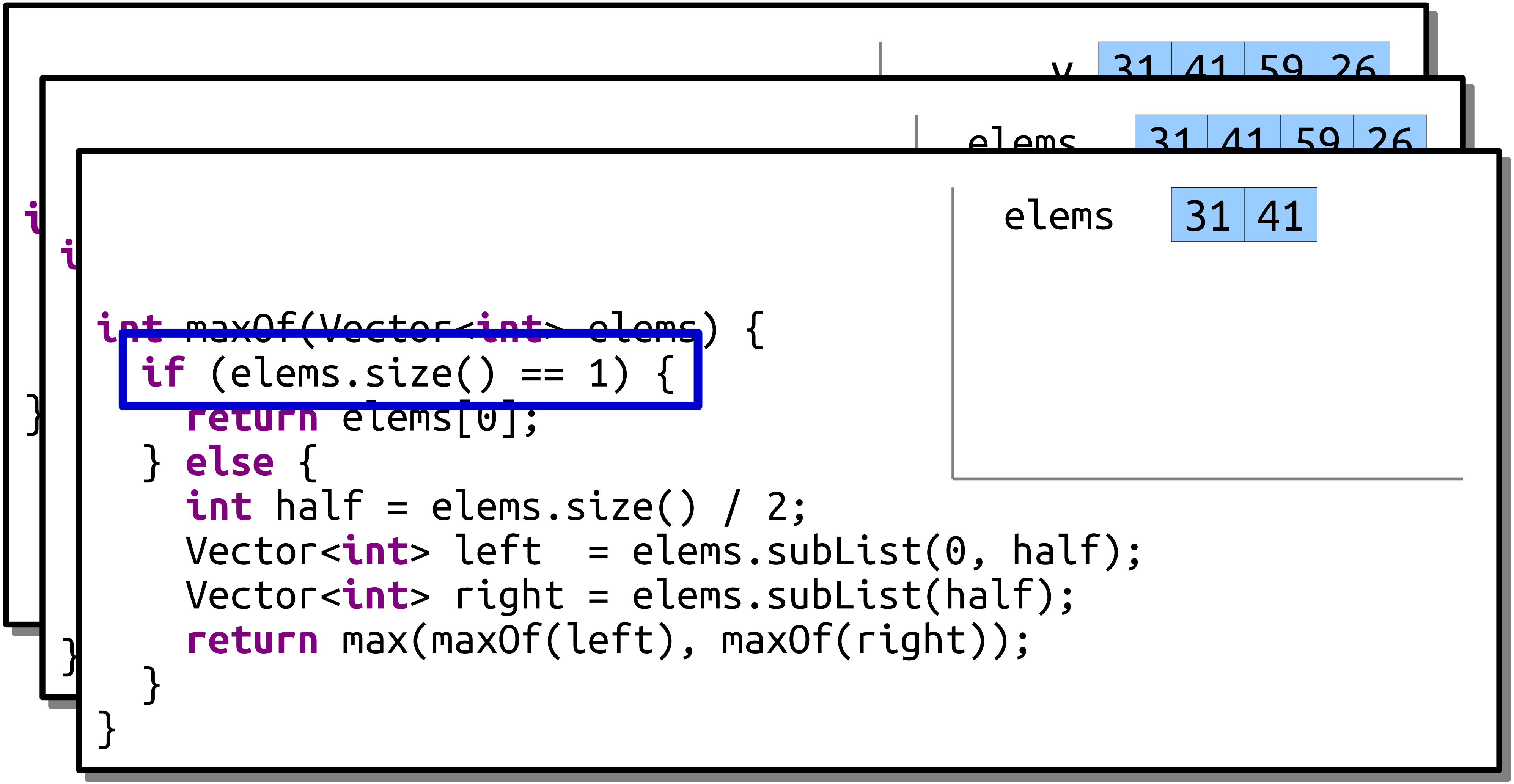
Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

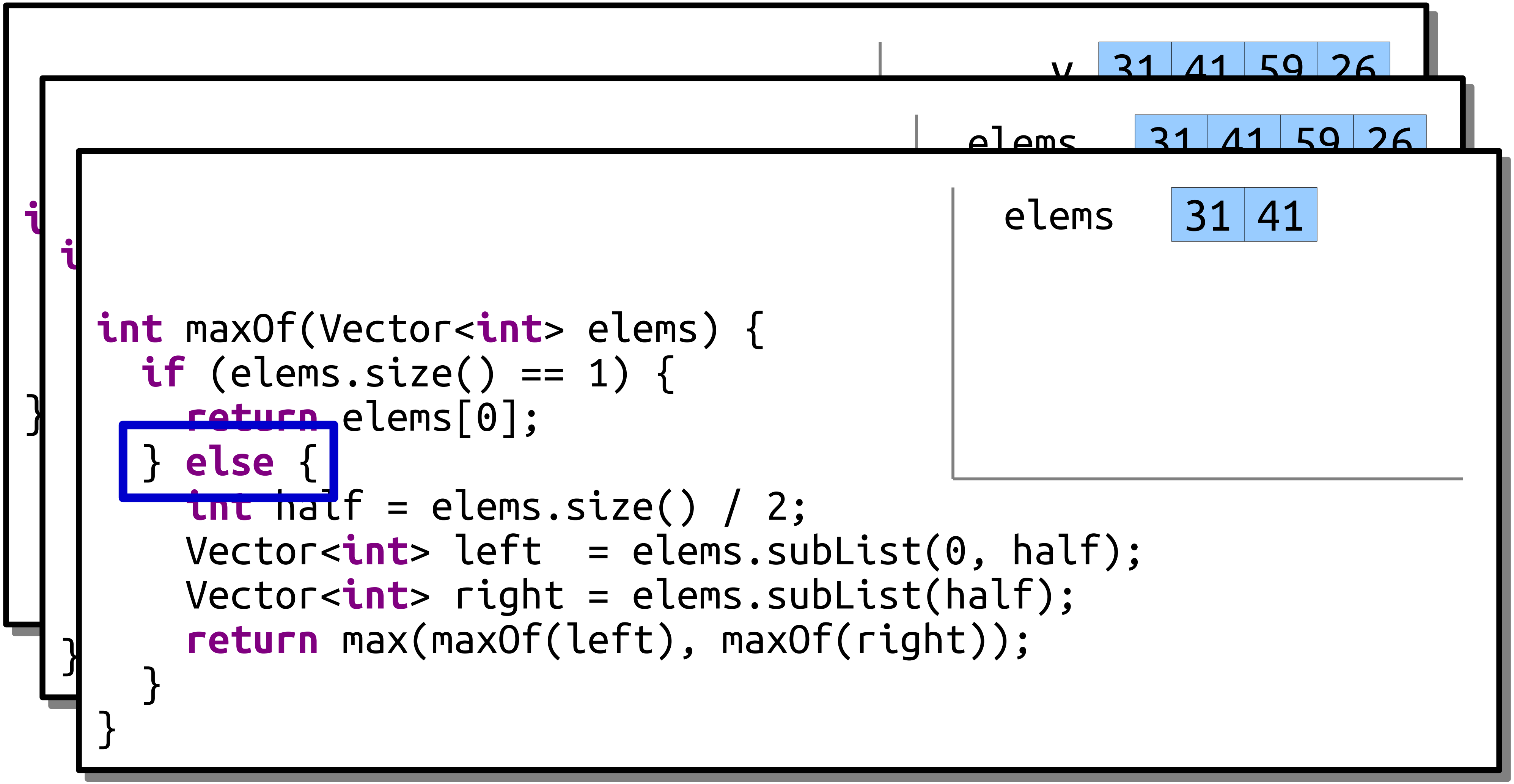
elems	31	41

Tracing the Recursion



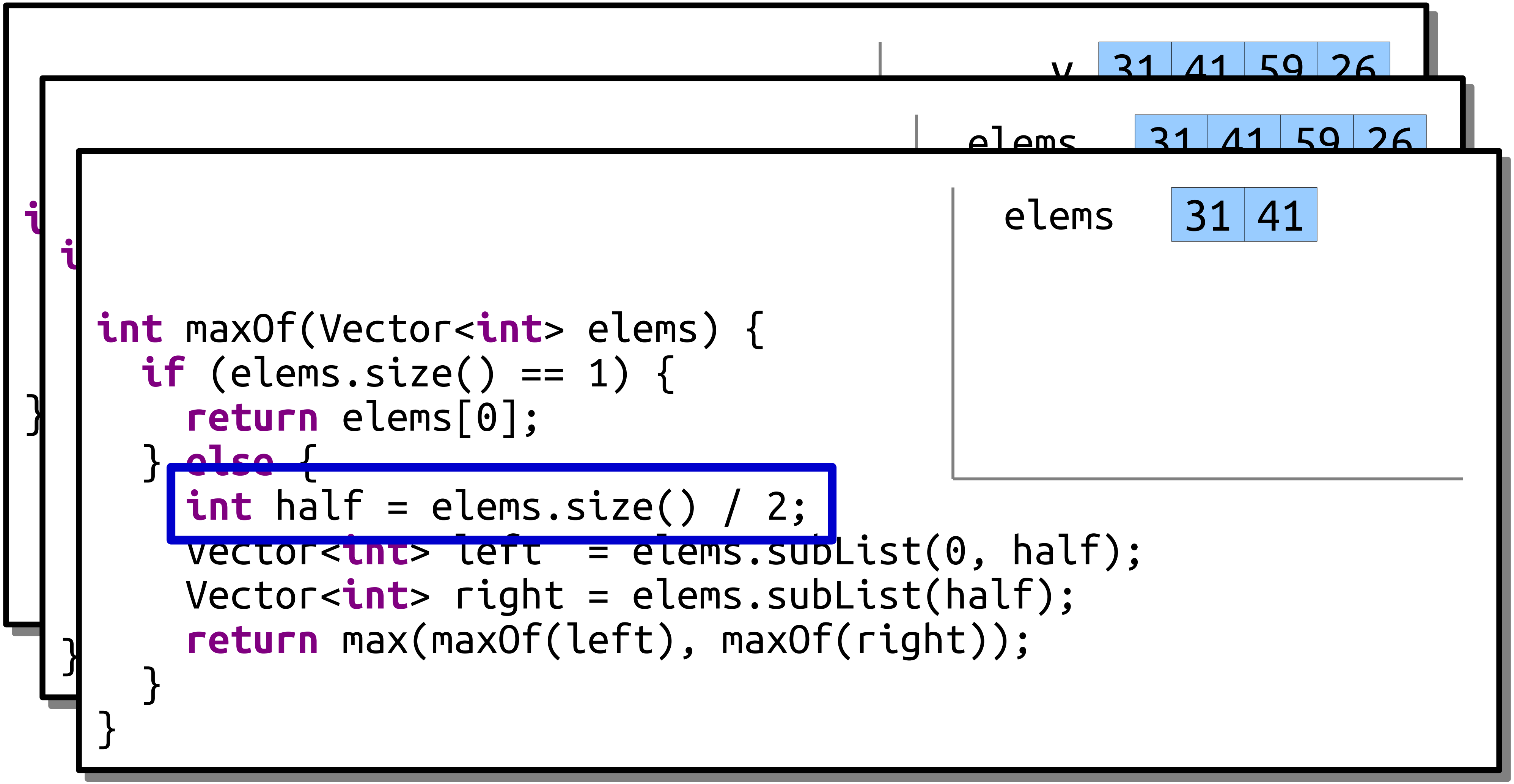
```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

Tracing the Recursion



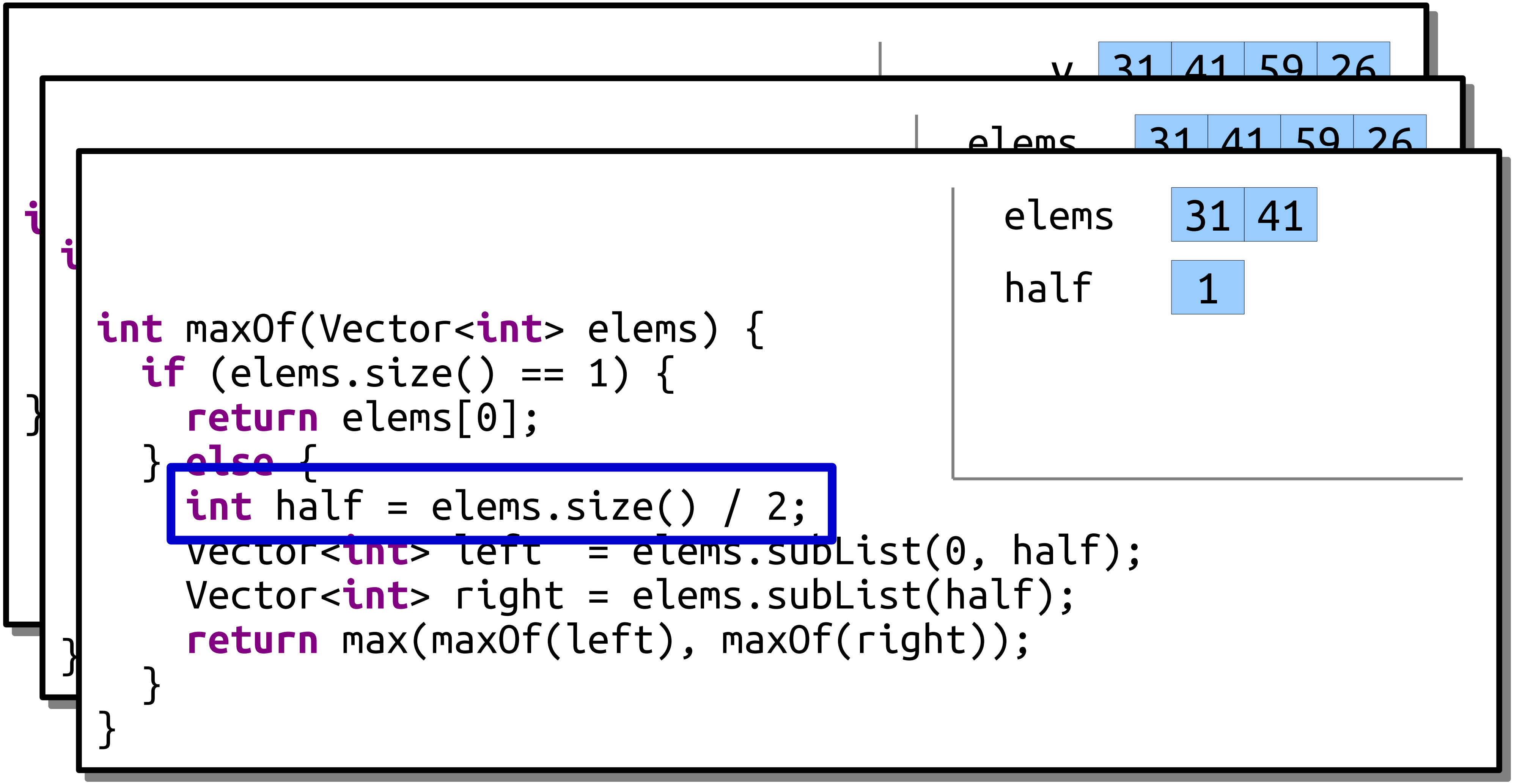
```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

Tracing the Recursion



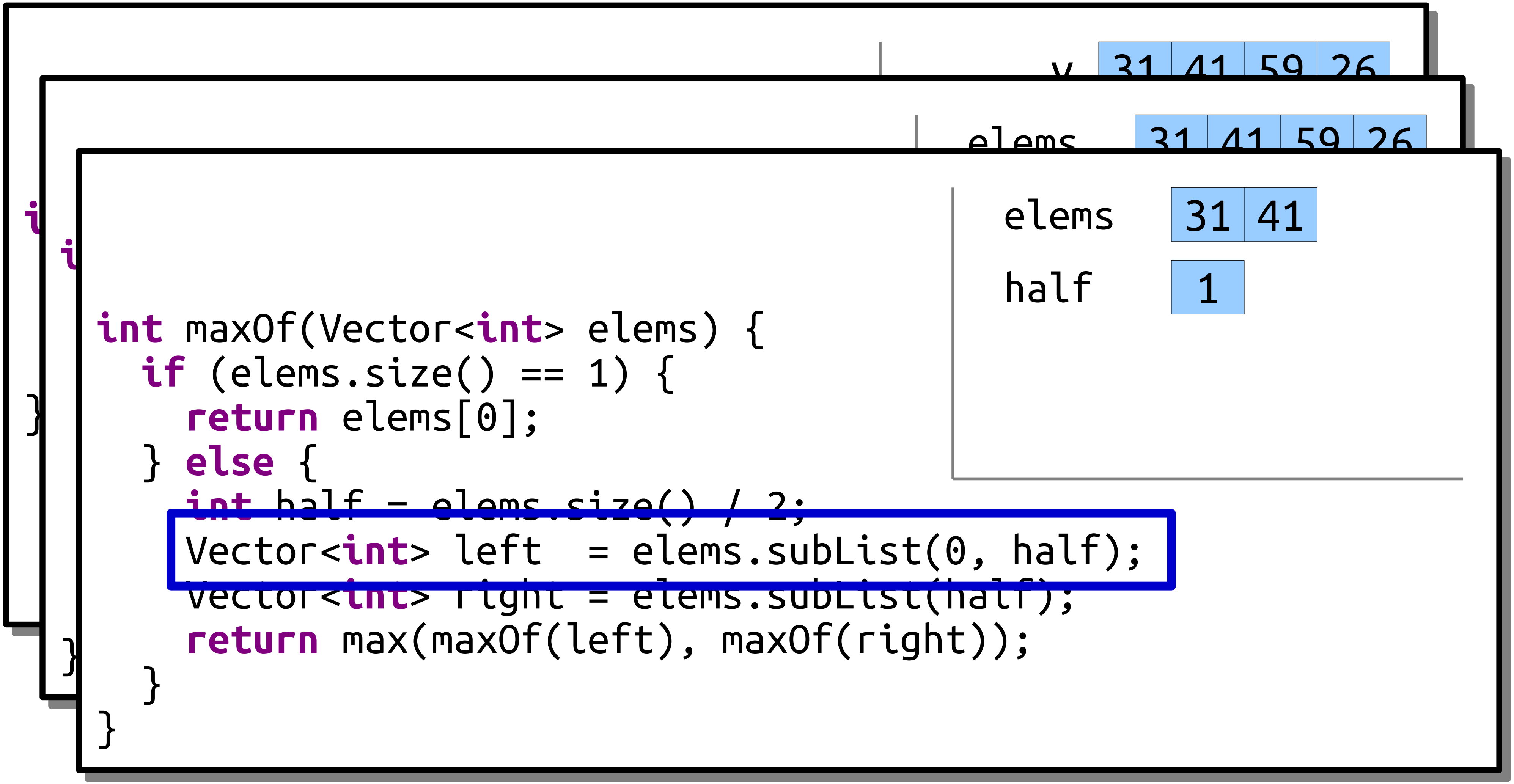
```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

Tracing the Recursion



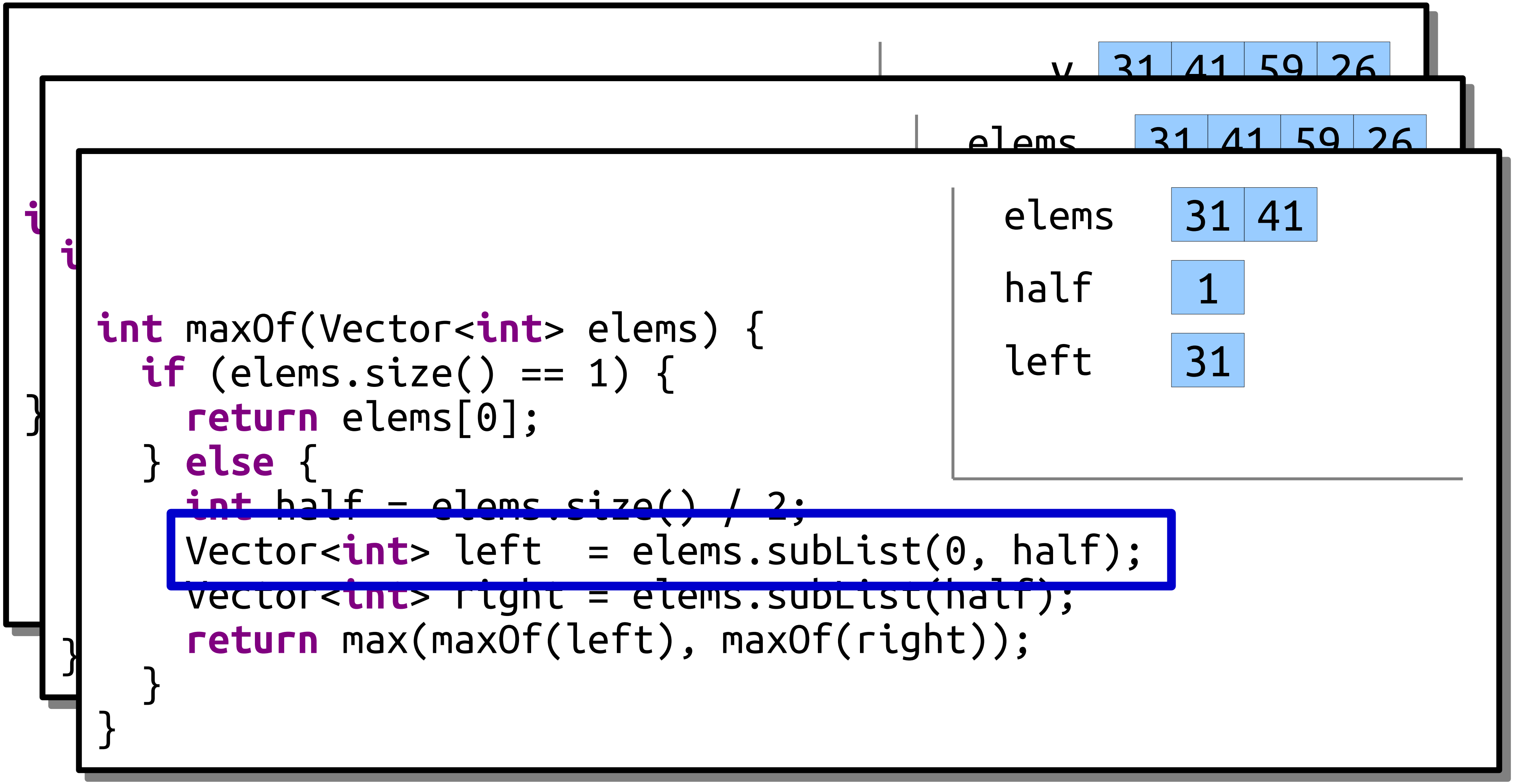
```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

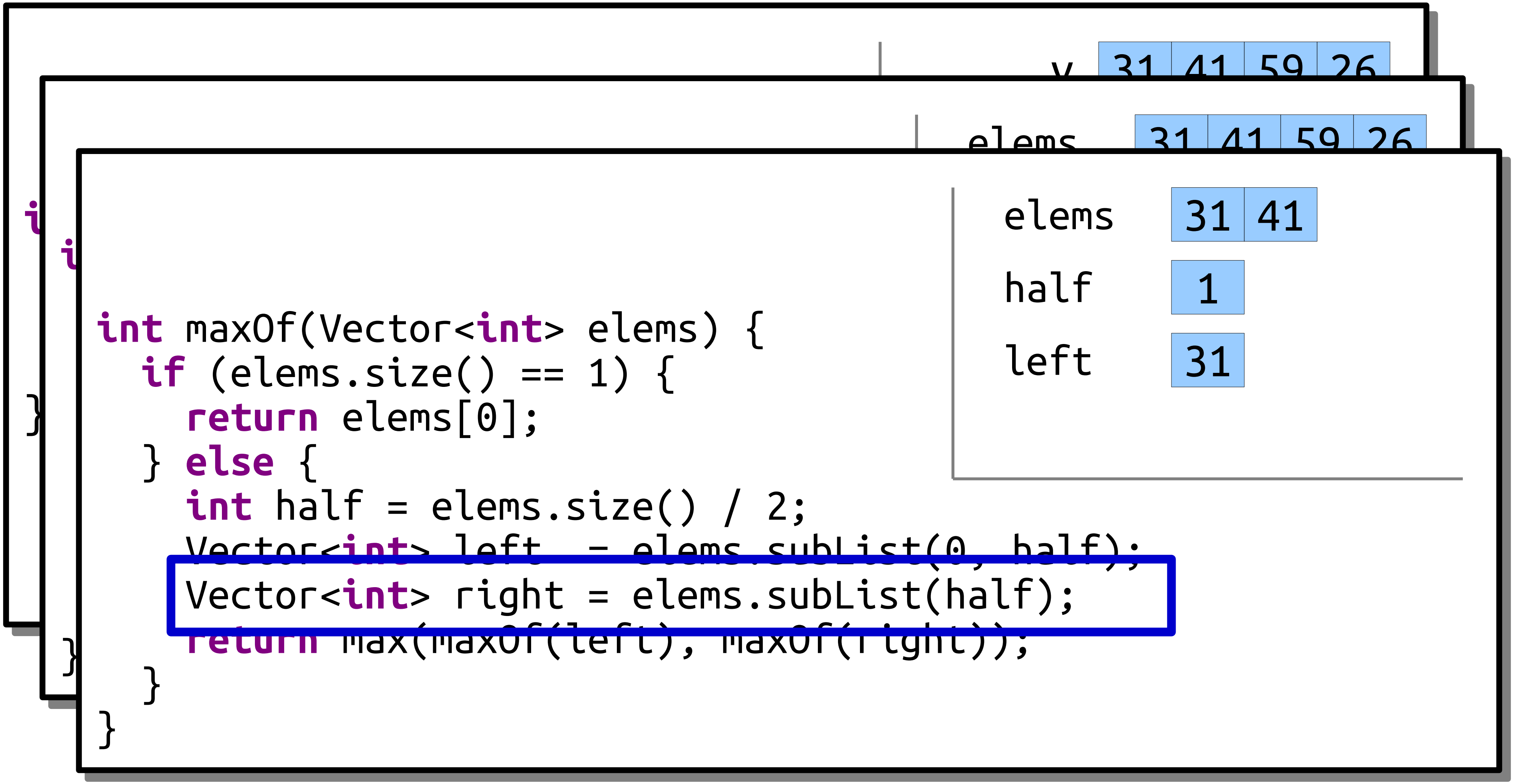
Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

elems	31 41
half	1
left	31

Tracing the Recursion

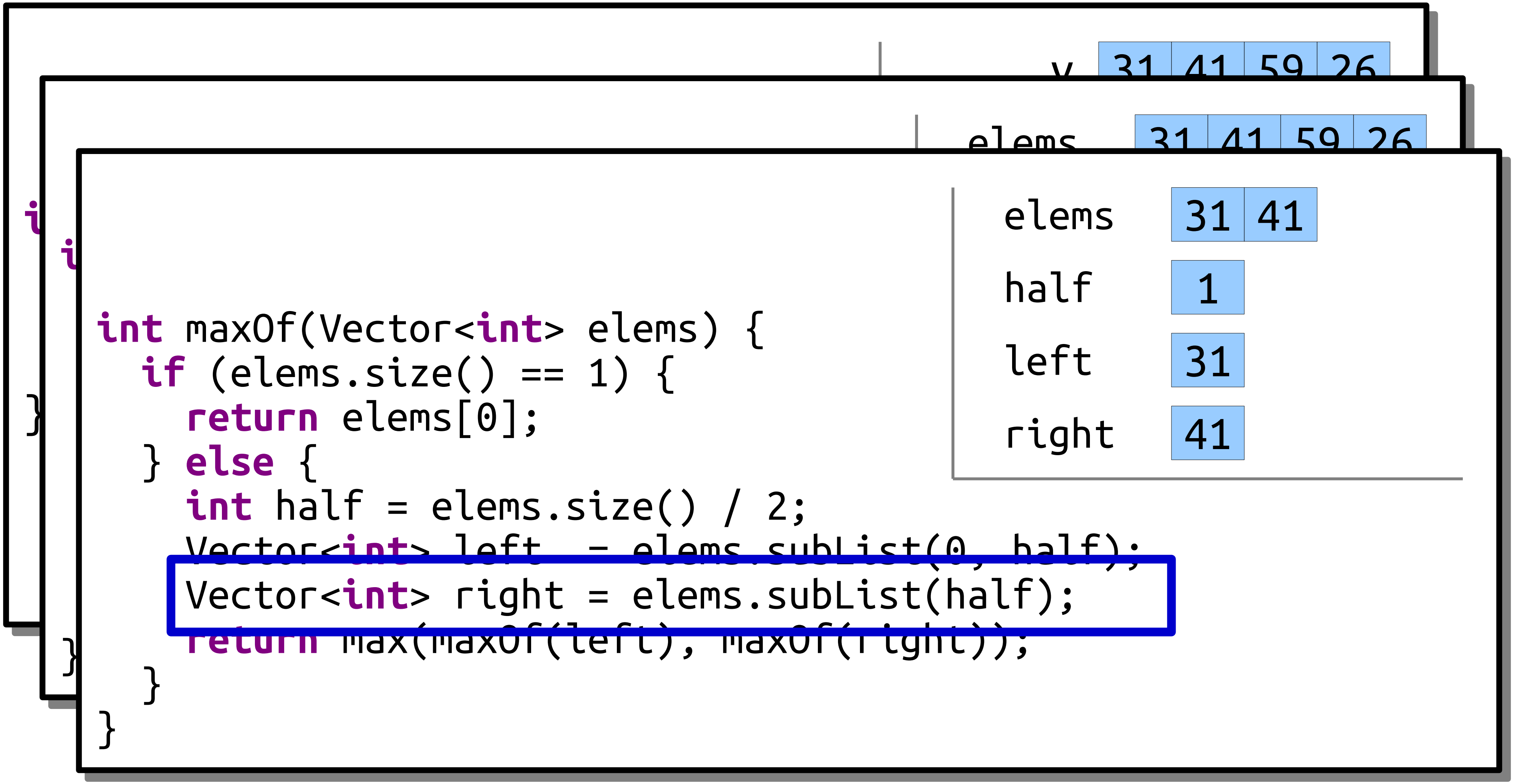


v	31	41	59	26
elems	31	41	59	26

```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

elems	31	41
half	1	
left	31	

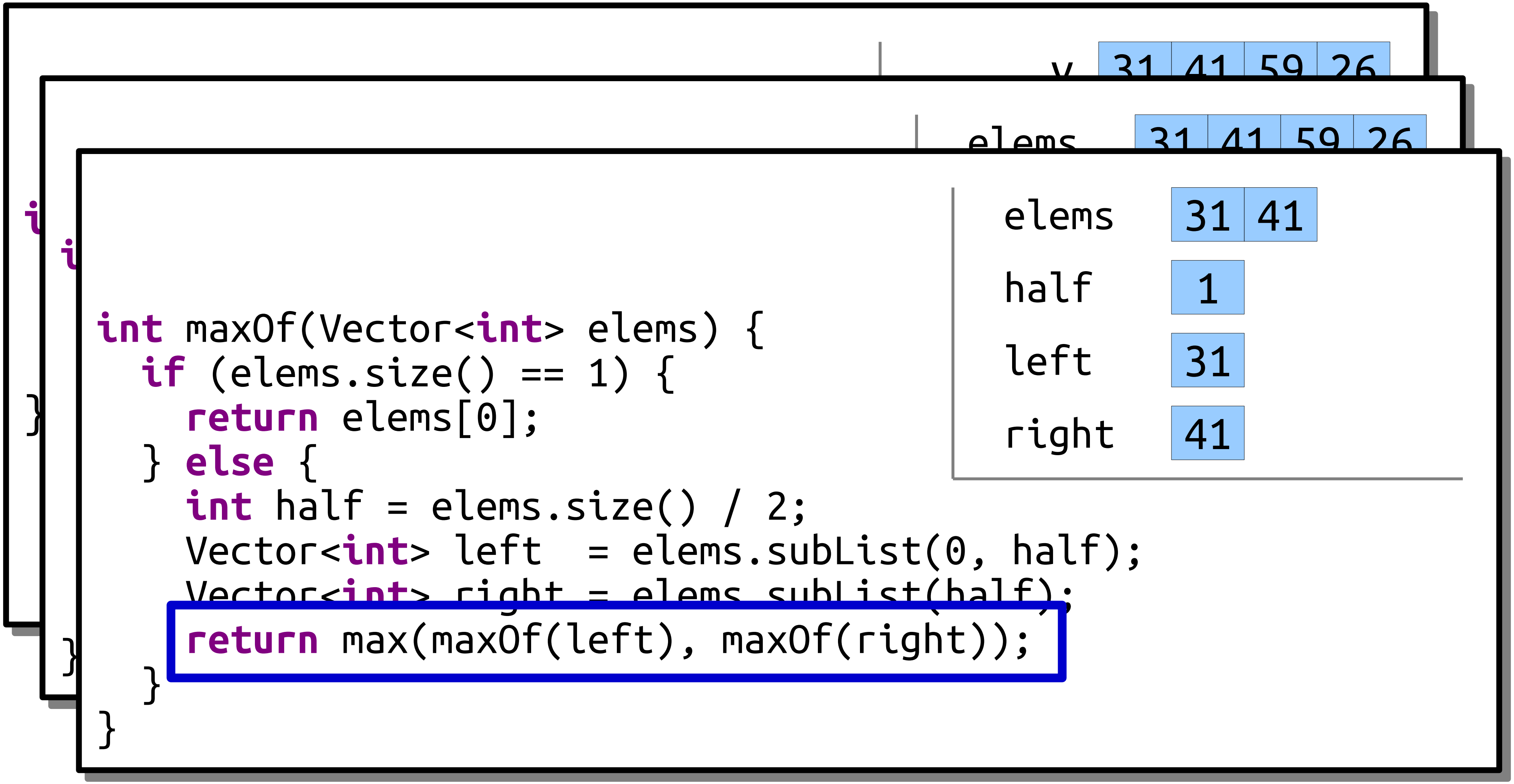
Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

elems	31 41
half	1
left	31
right	41

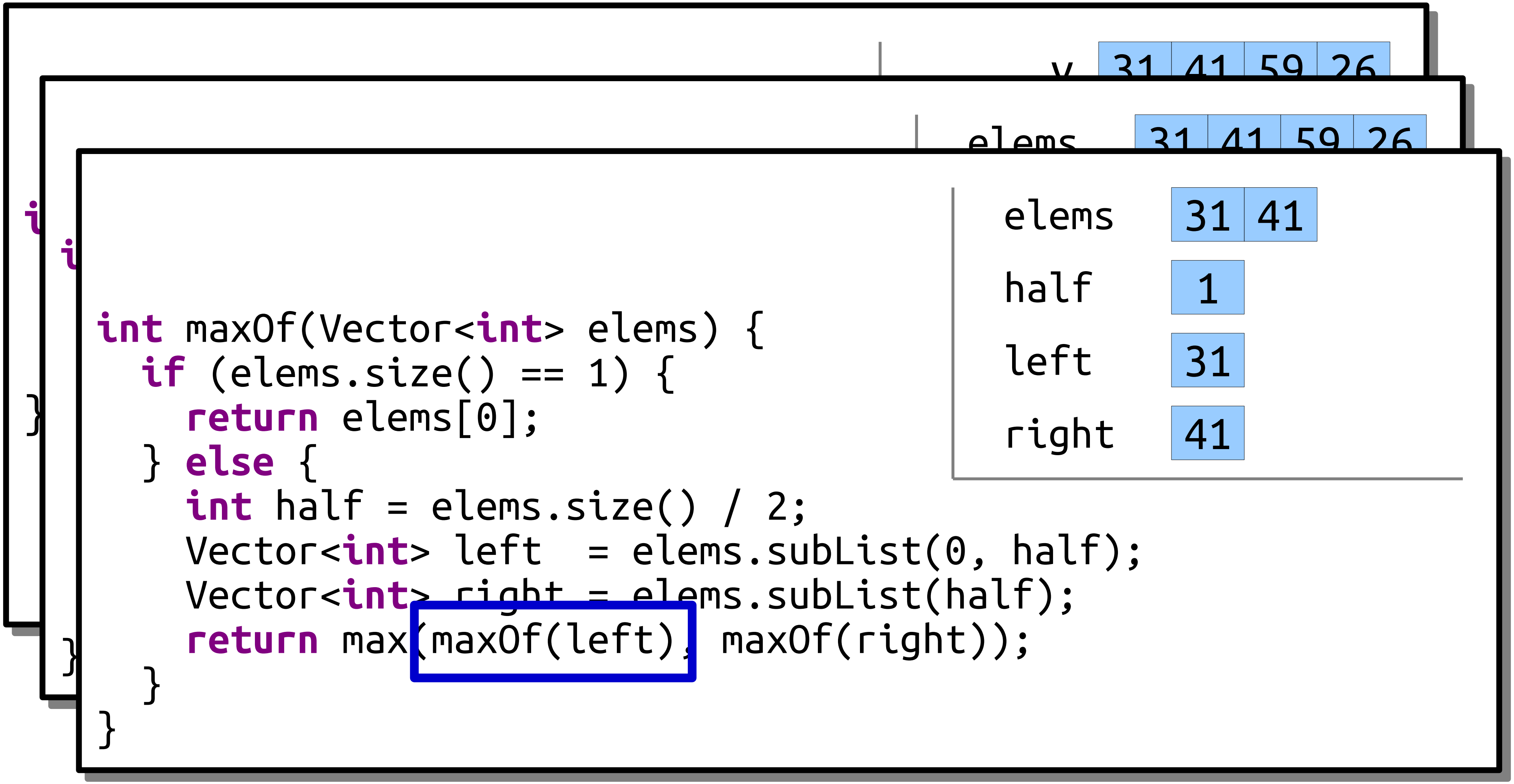
Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

elems	31 41
half	1
left	31
right	41

Tracing the Recursion



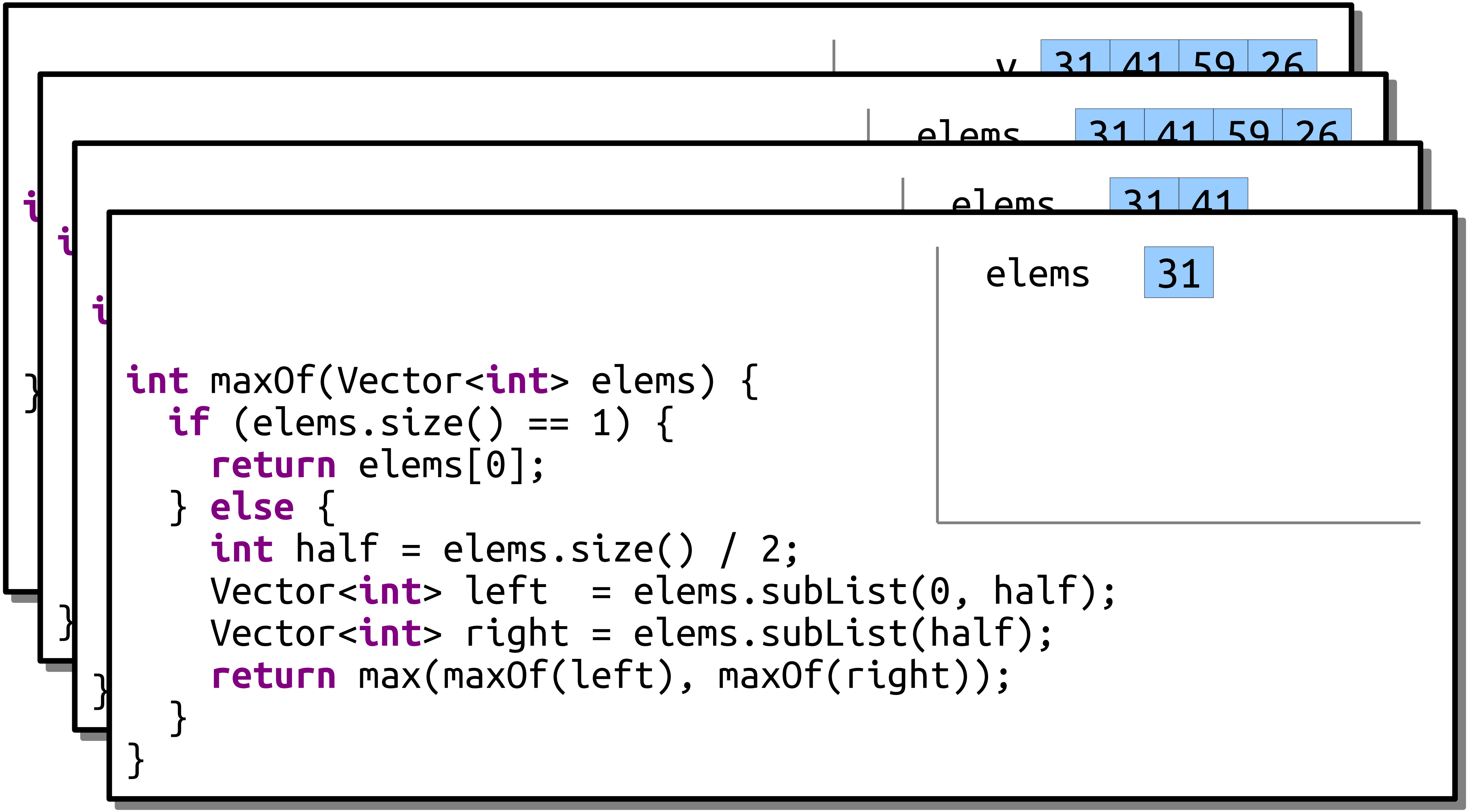
v 31 41 59 26

elems 31 41 59 26

```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

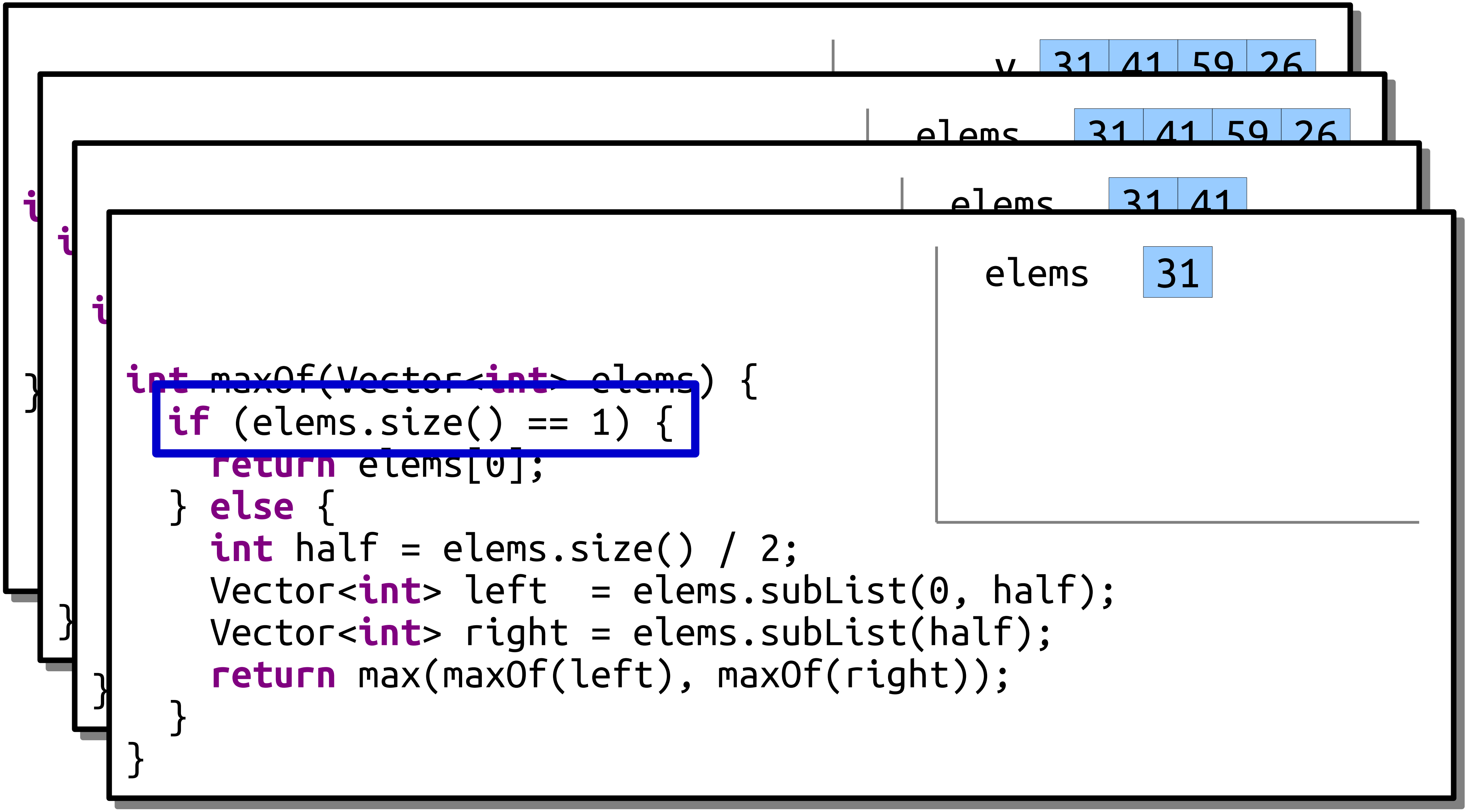
elems	31 41
half	1
left	31
right	41

Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

Tracing the Recursion



v 31 41 59 26

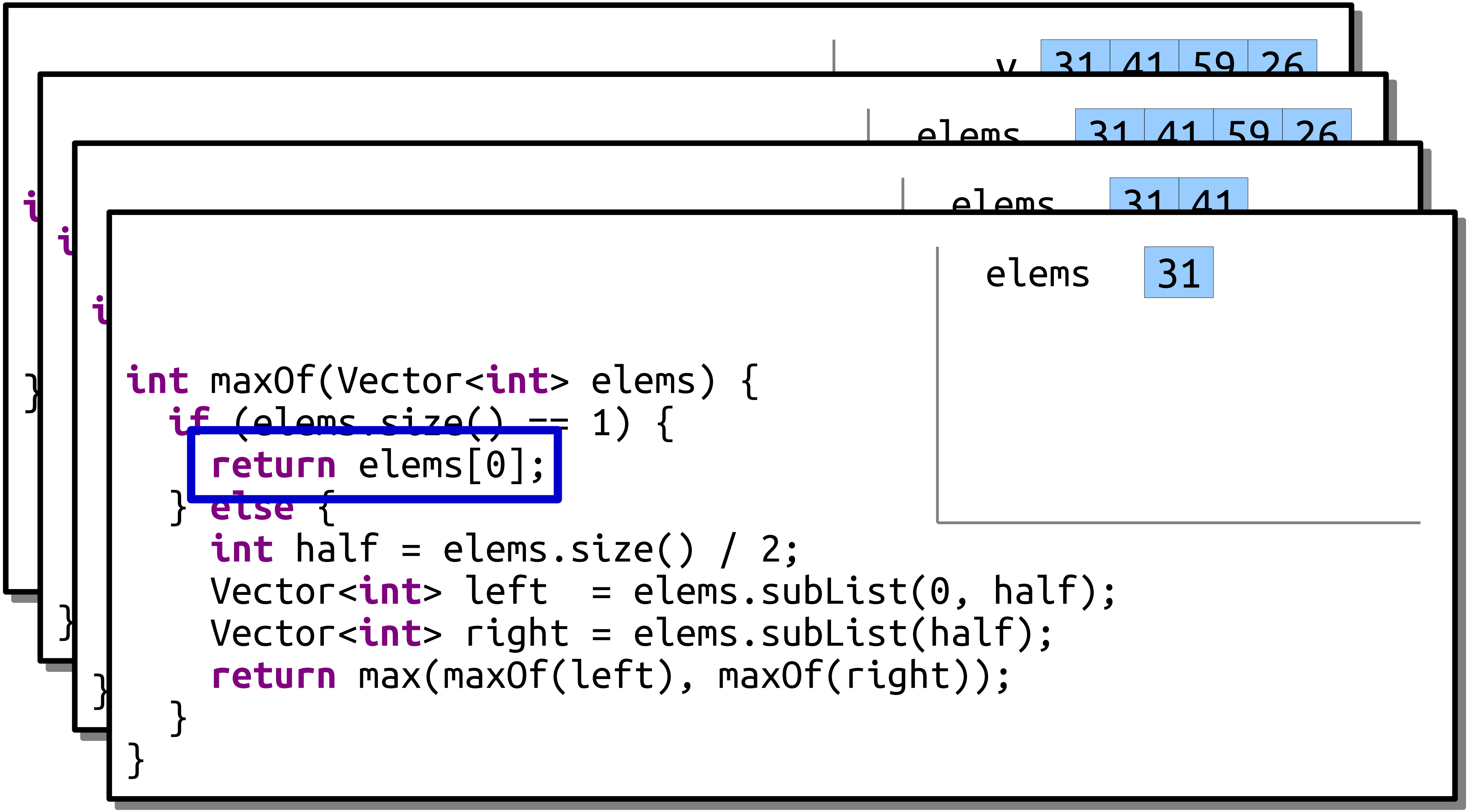
elems 31 41 59 26

elems 31 41

elems 31

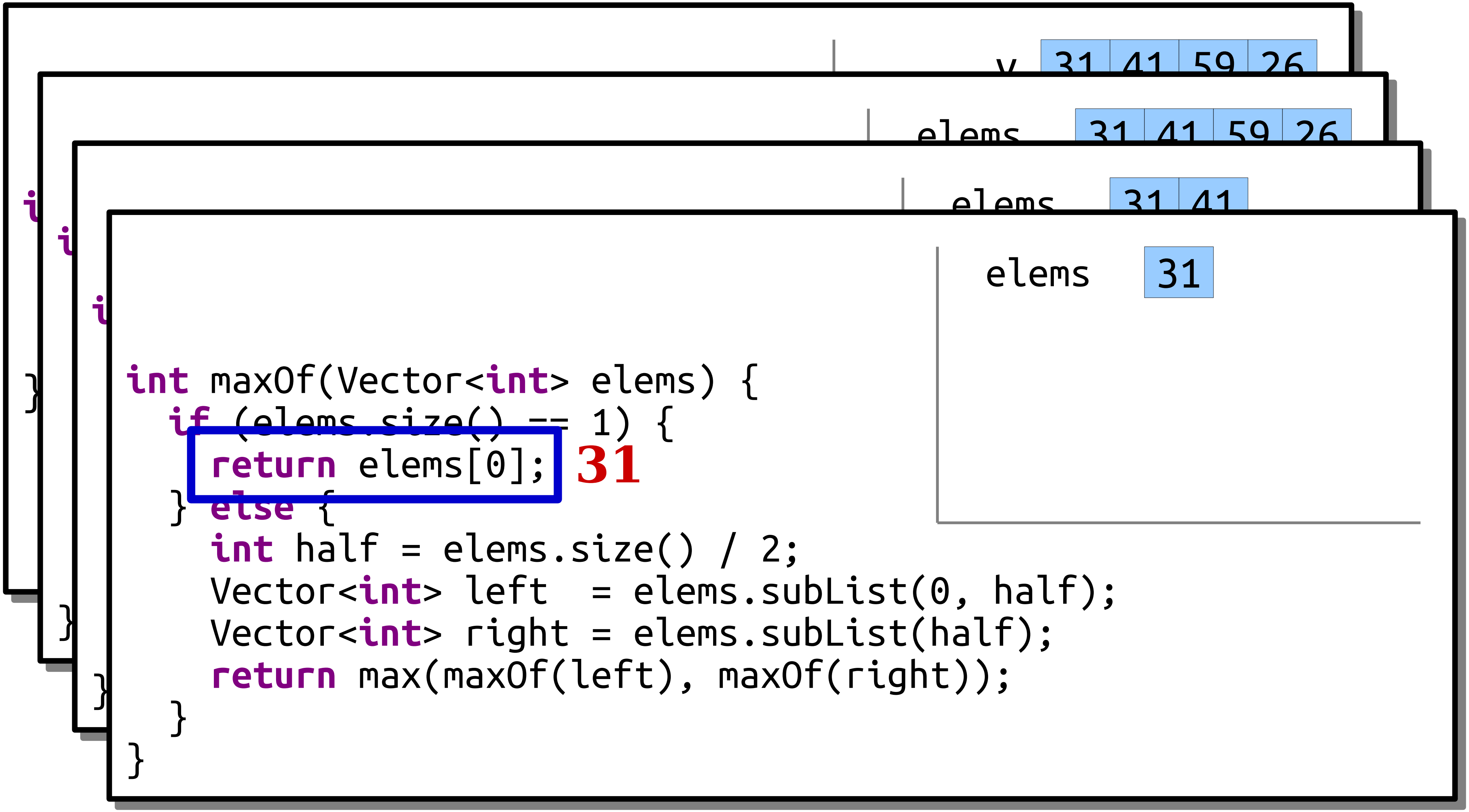
```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

Tracing the Recursion



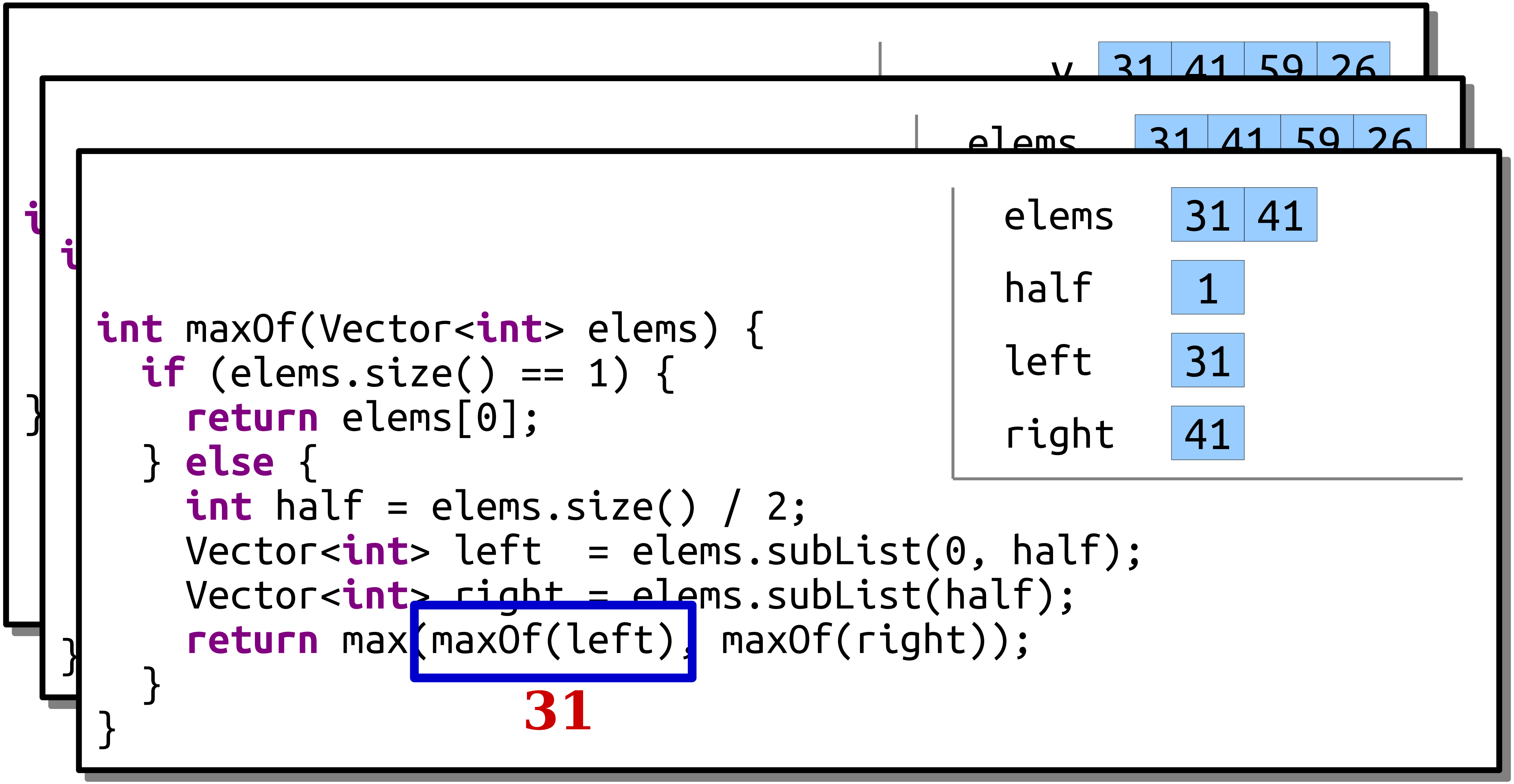
```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

Tracing the Recursion

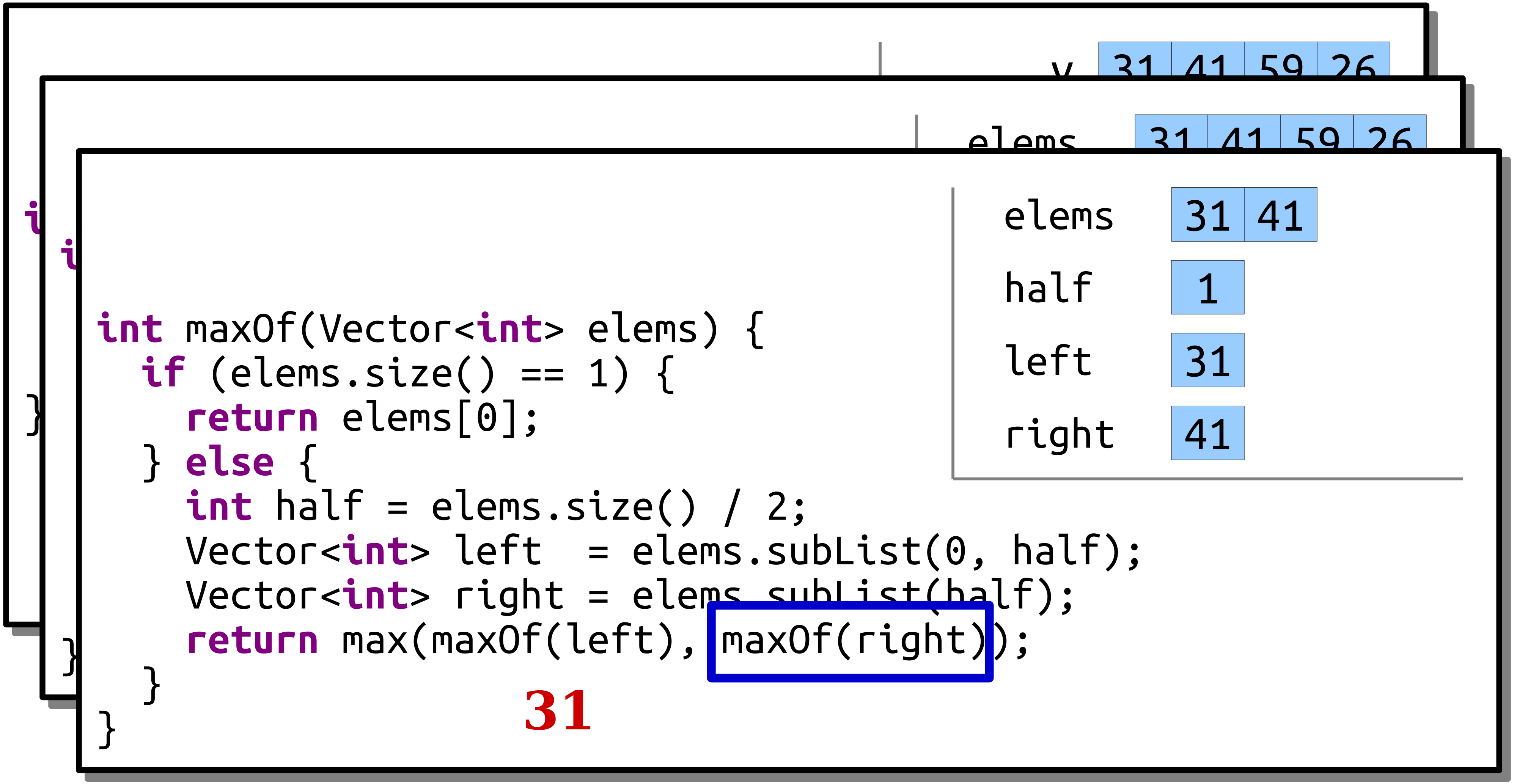


```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

31

elems	31	41
half	1	
left	31	
right	41	

Tracing the Recursion

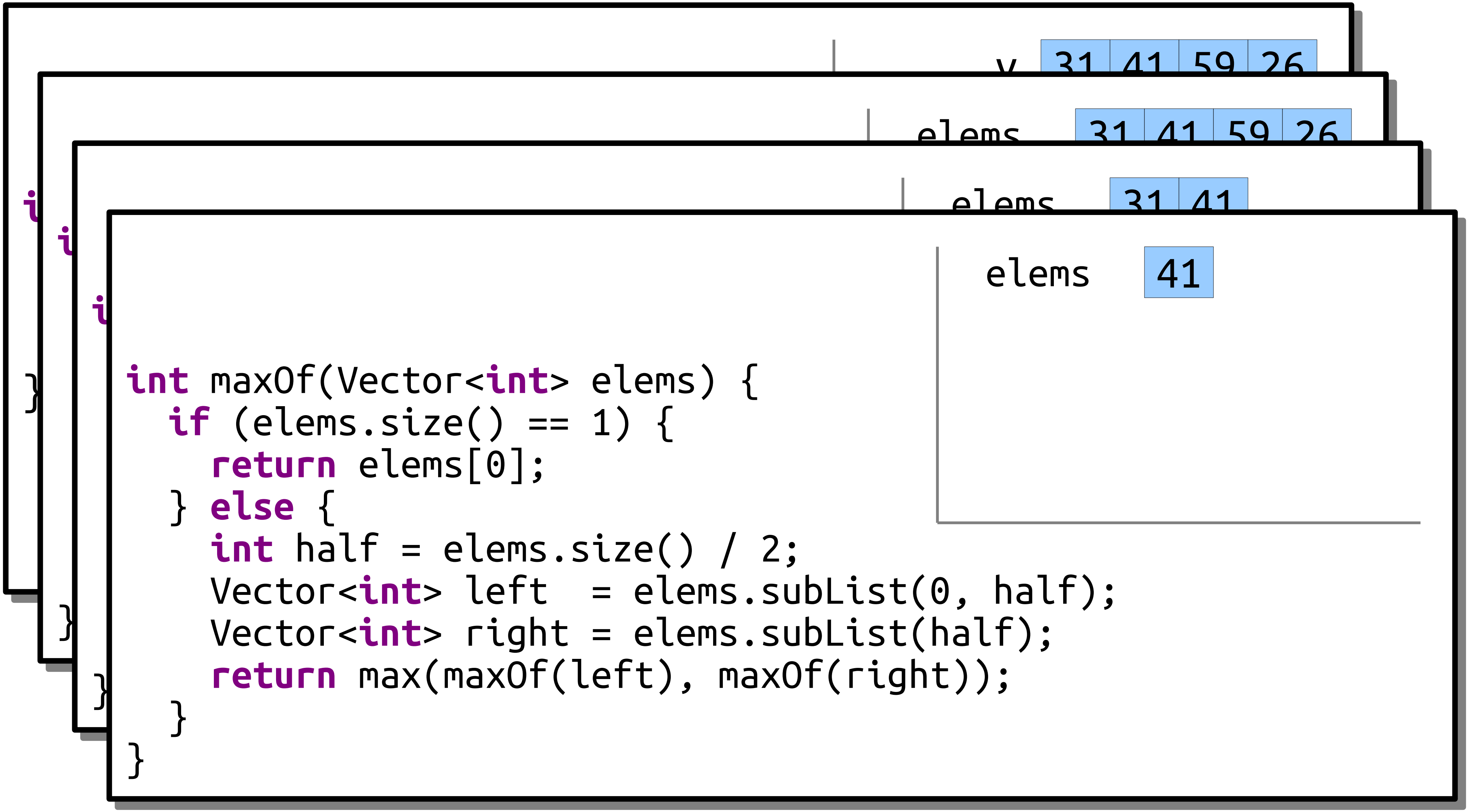


```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

31

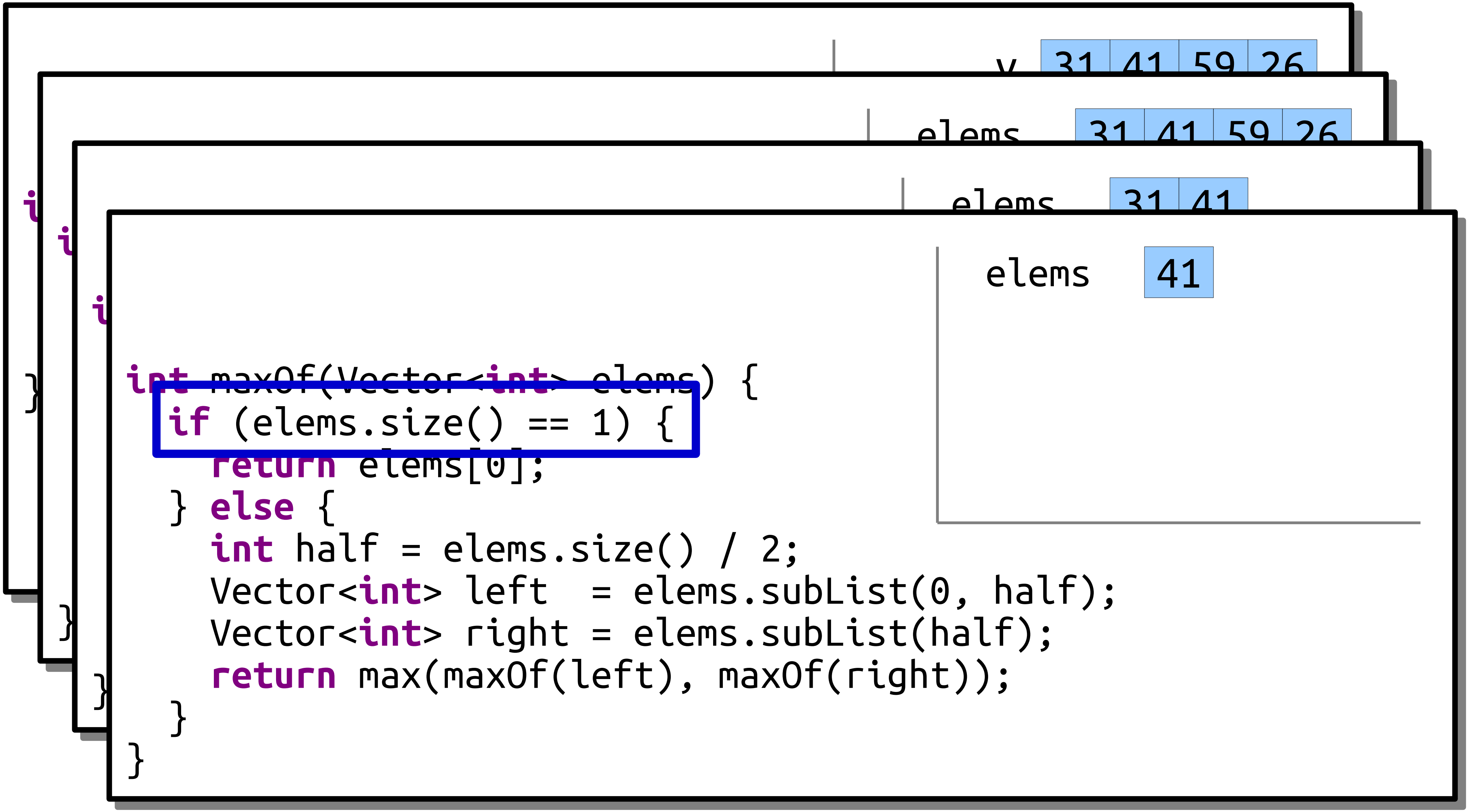
elems	31 41
half	1
left	31
right	41

Tracing the Recursion



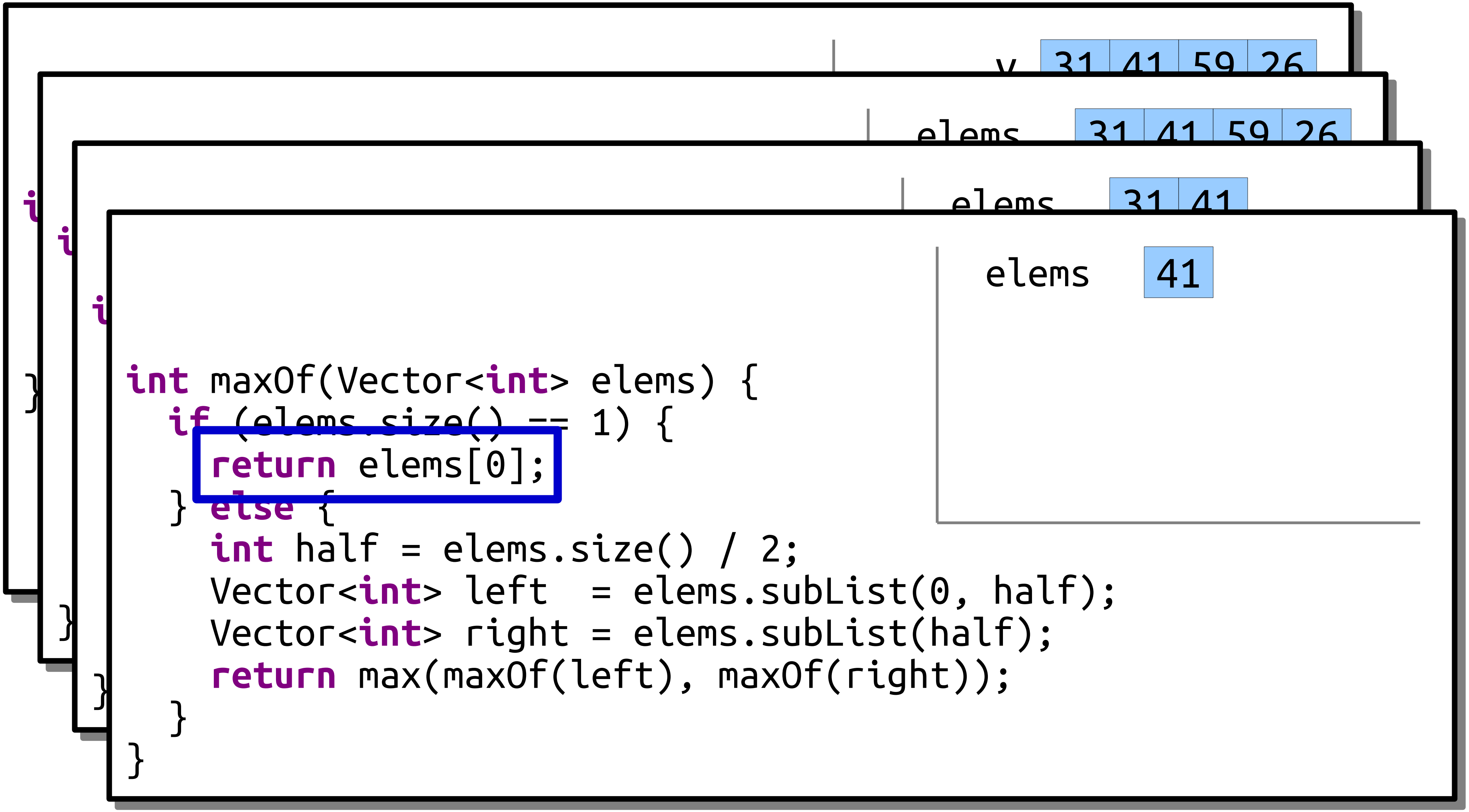
```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

Tracing the Recursion



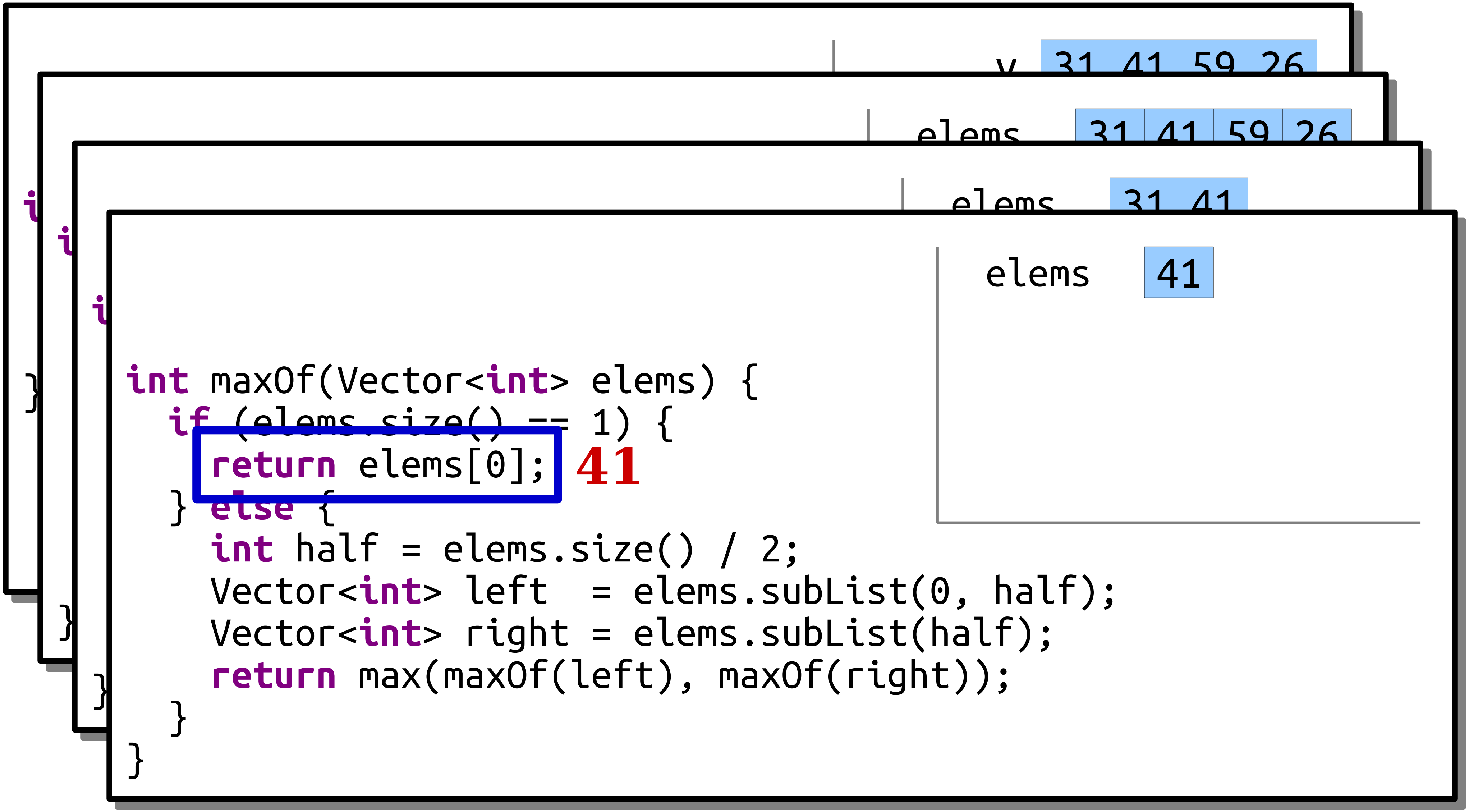
```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

Tracing the Recursion



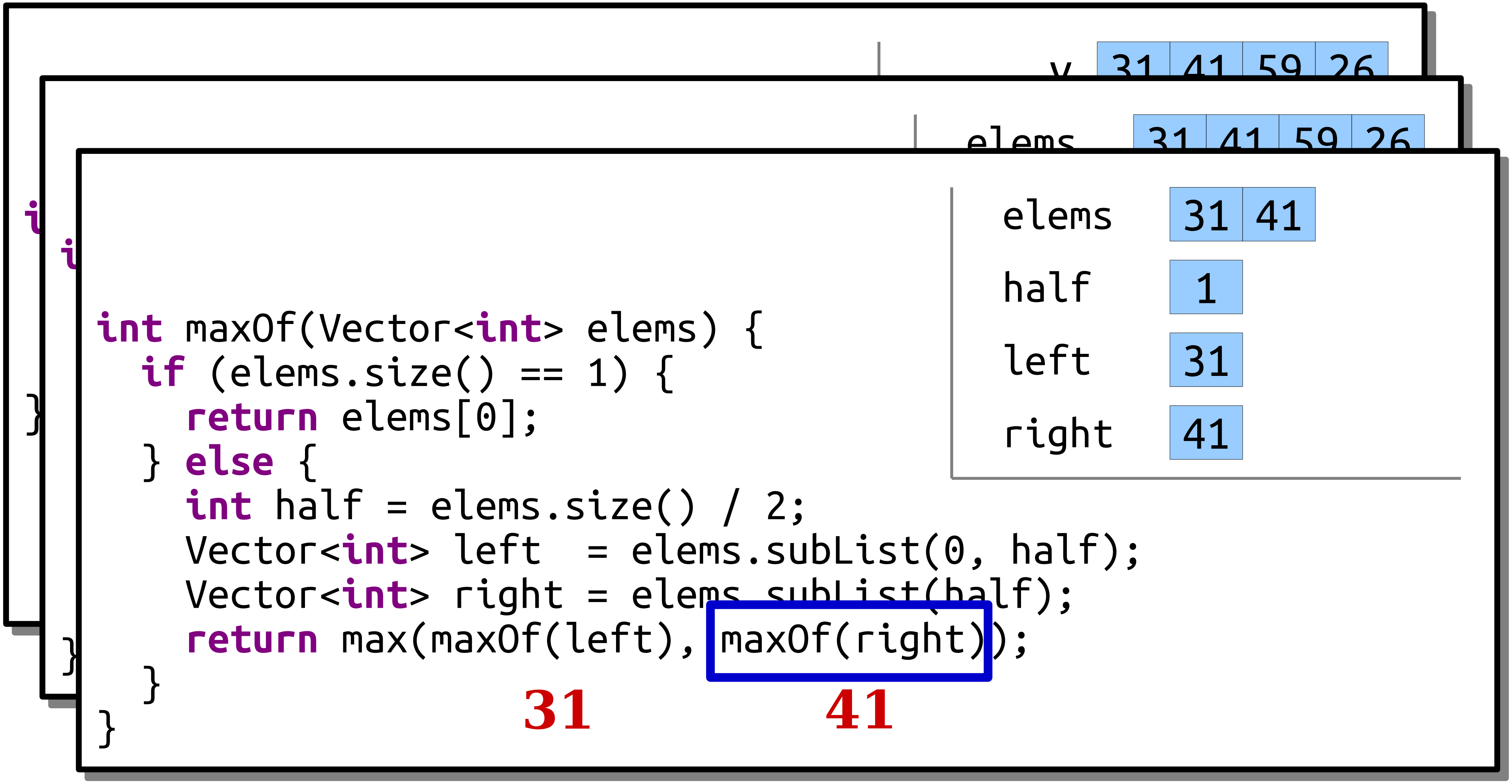
```
int maxOf(Vector<int> elems) {
   if (elems.size() == 1) {
      return elems[0];
   } else {
      int half = elems.size() / 2;
      Vector<int> left  = elems.subList(0, half);
      Vector<int> right = elems.subList(half);
      return max(maxOf(left), maxOf(right));
   }
}
```

Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

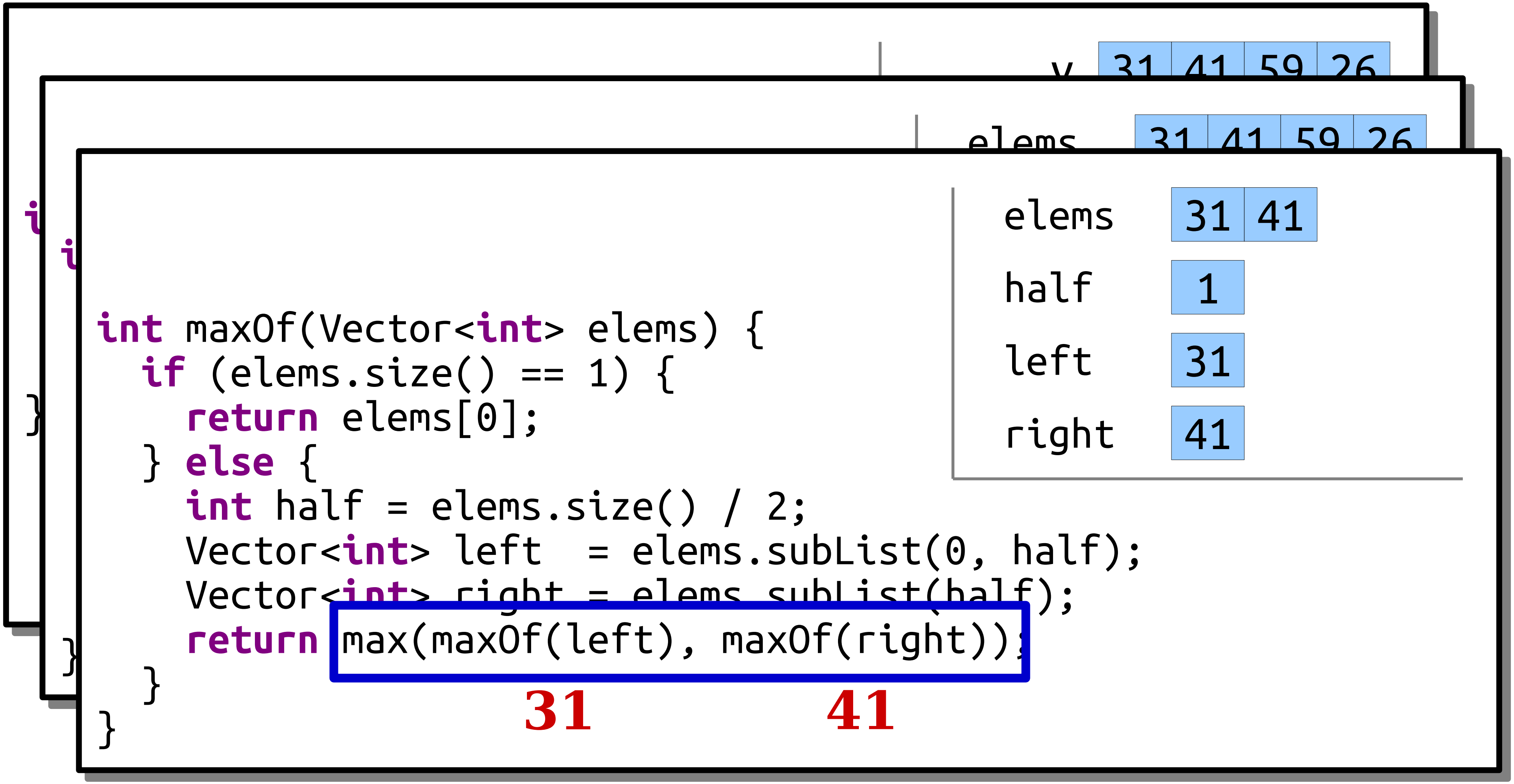
Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

elems	31 41
half	1
left	31
right	41

Tracing the Recursion

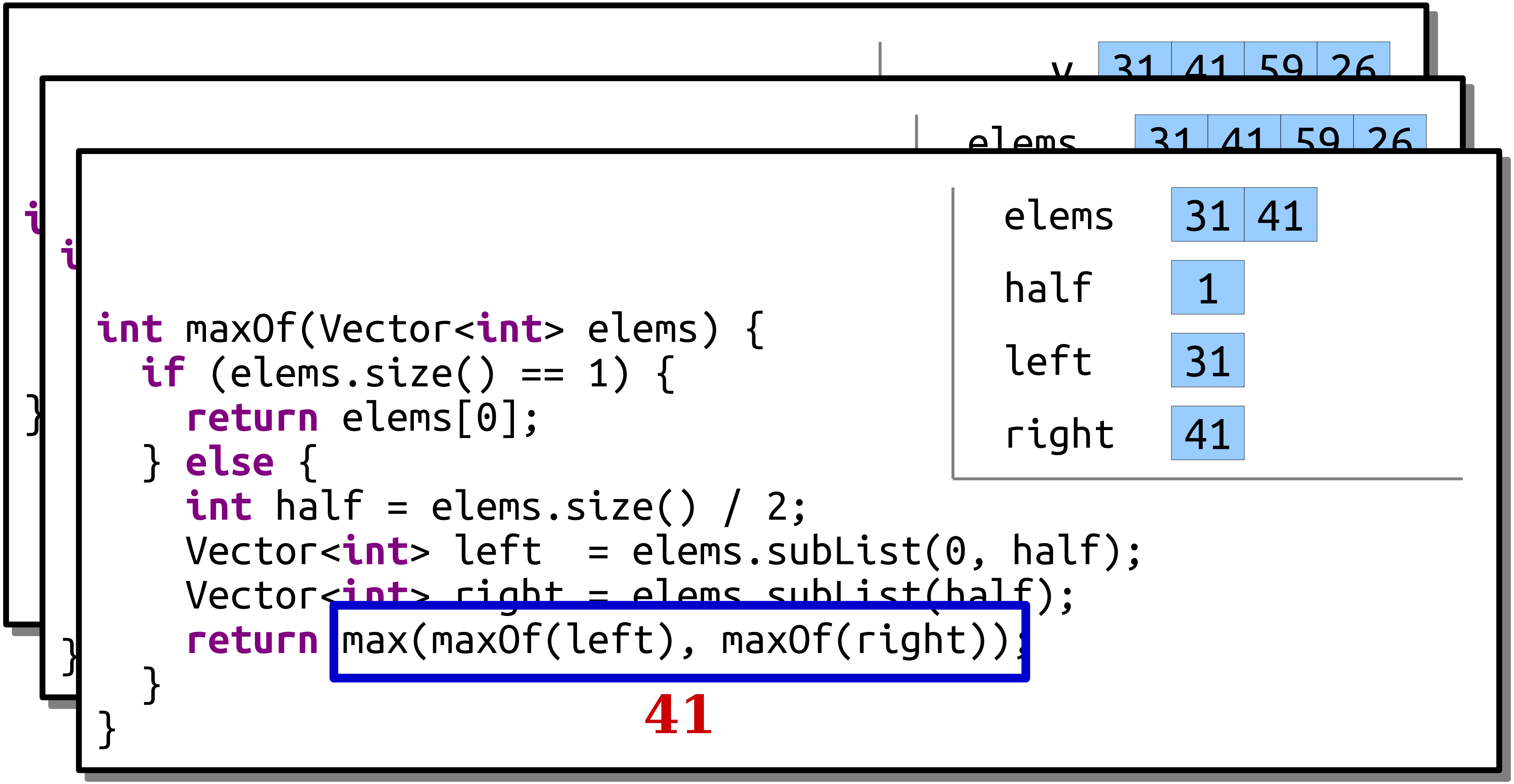


```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

31 41

elems	31	41
half	1	
left	31	
right	41	

Tracing the Recursion

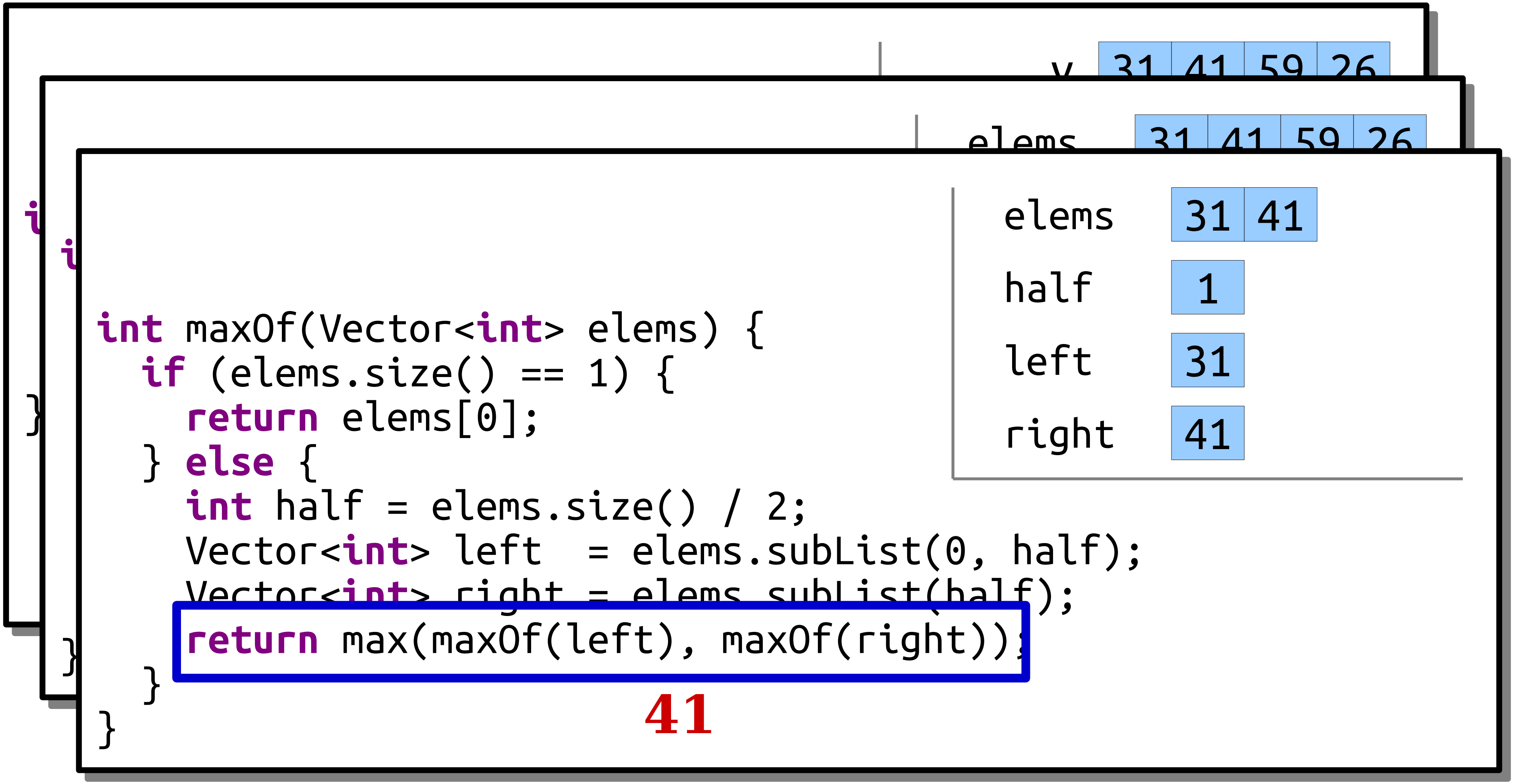


```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

elems	31	41
half	1	
left	31	
right	41	

41

Tracing the Recursion

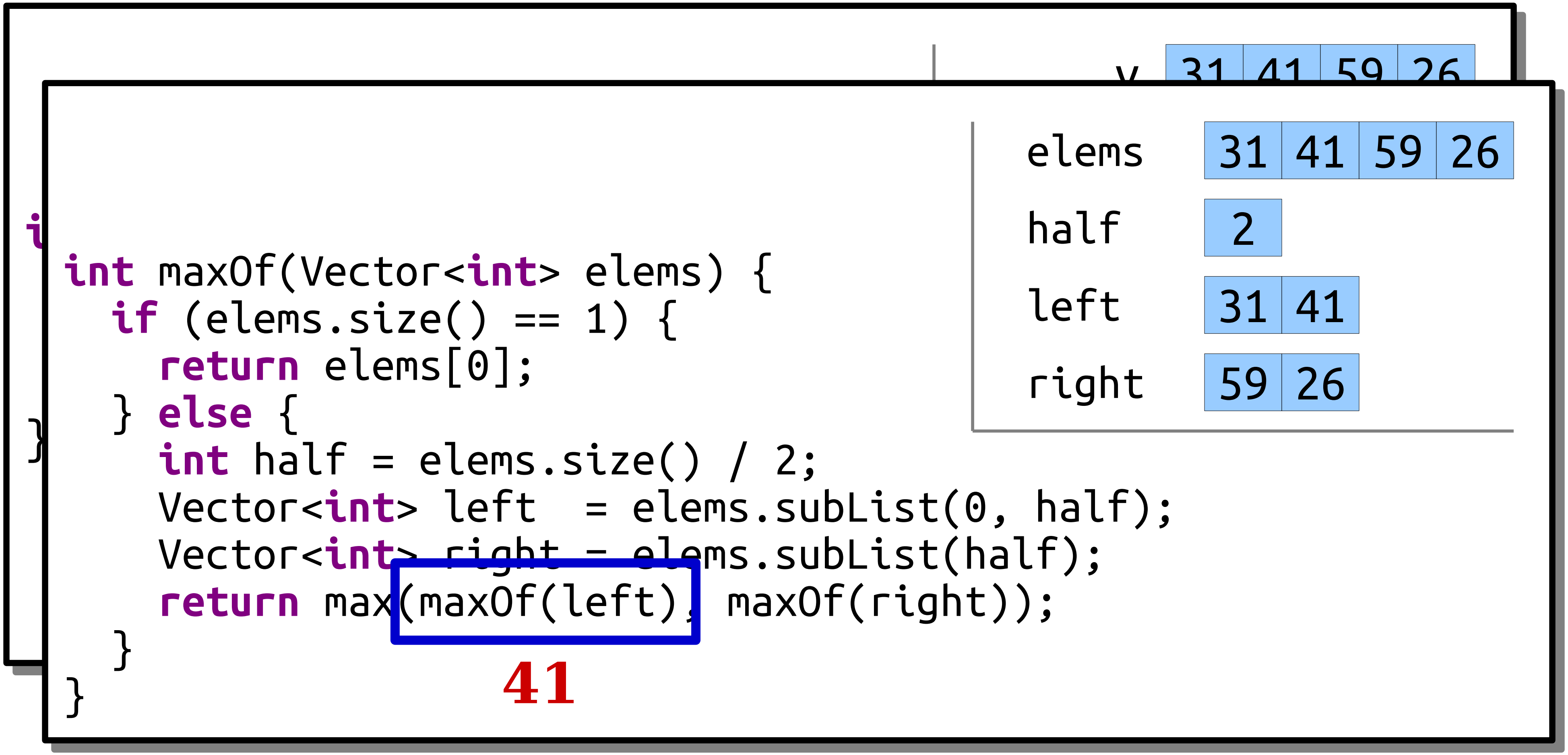


```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

elems	31 41
half	1
left	31
right	41

41

Tracing the Recursion

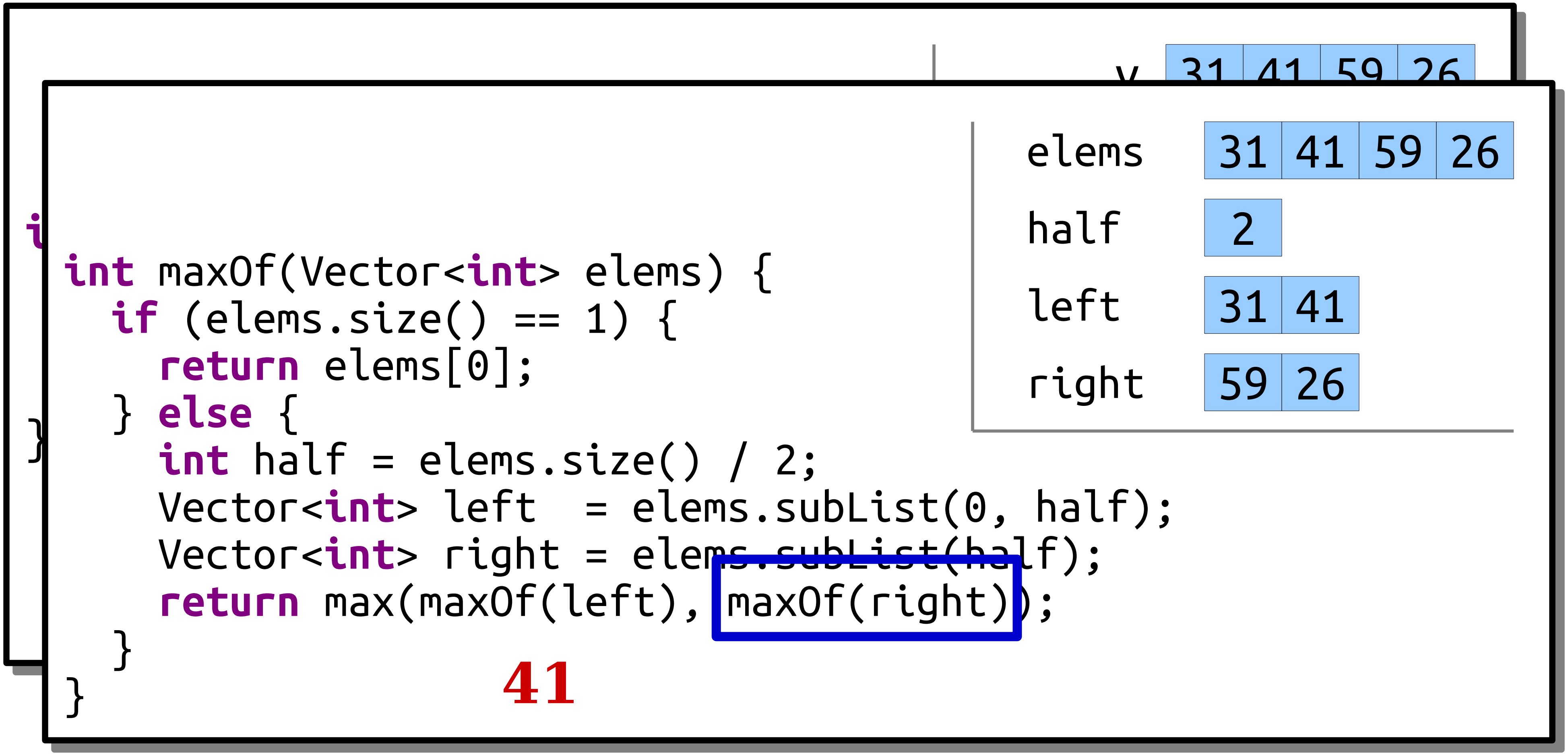


```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

elems	31 41 59 26
half	2
left	31 41
right	59 26

41

Tracing the Recursion

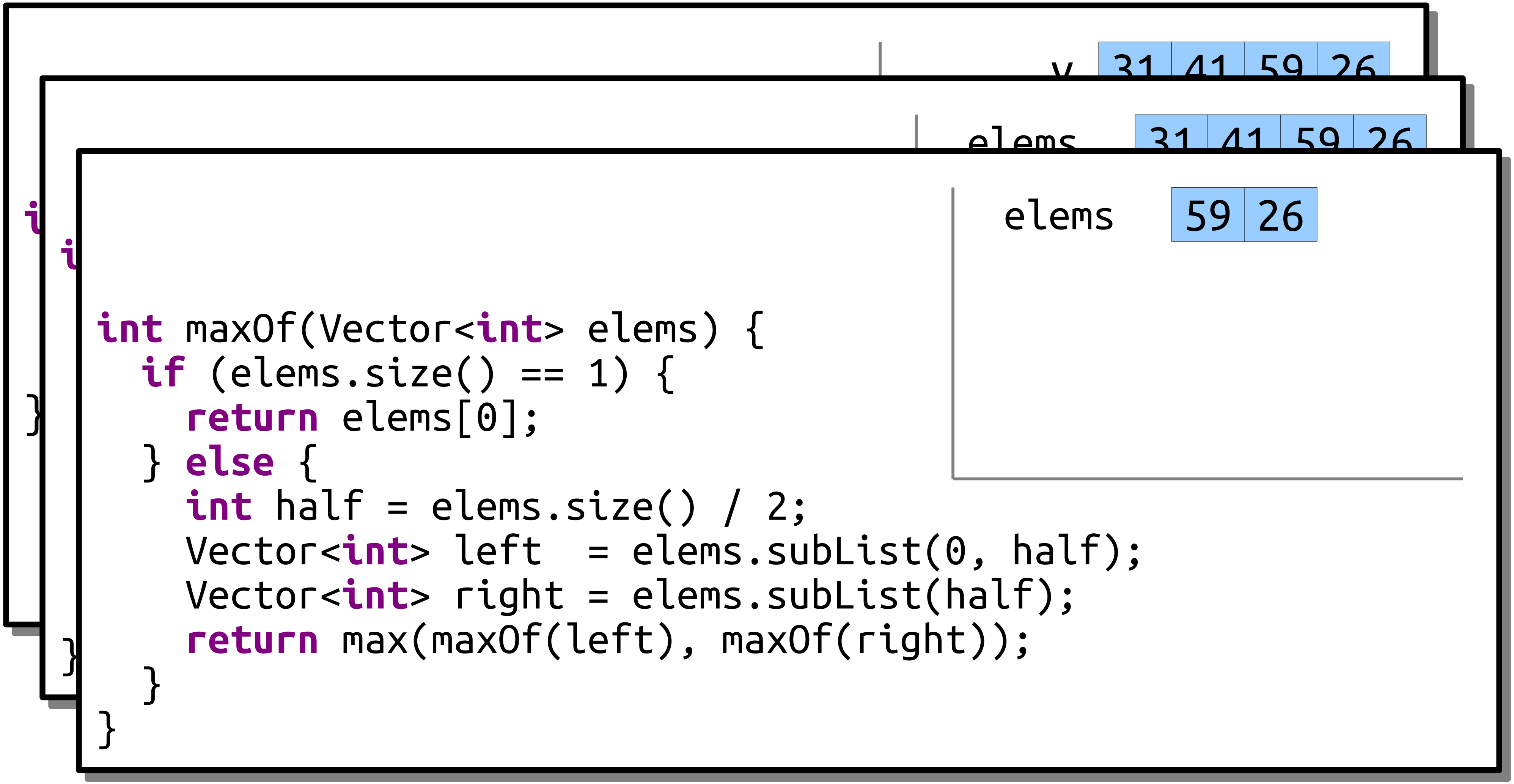


```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

elems	31	41	59	26
half	2			
left	31	41		
right	59	26		

41

Tracing the Recursion



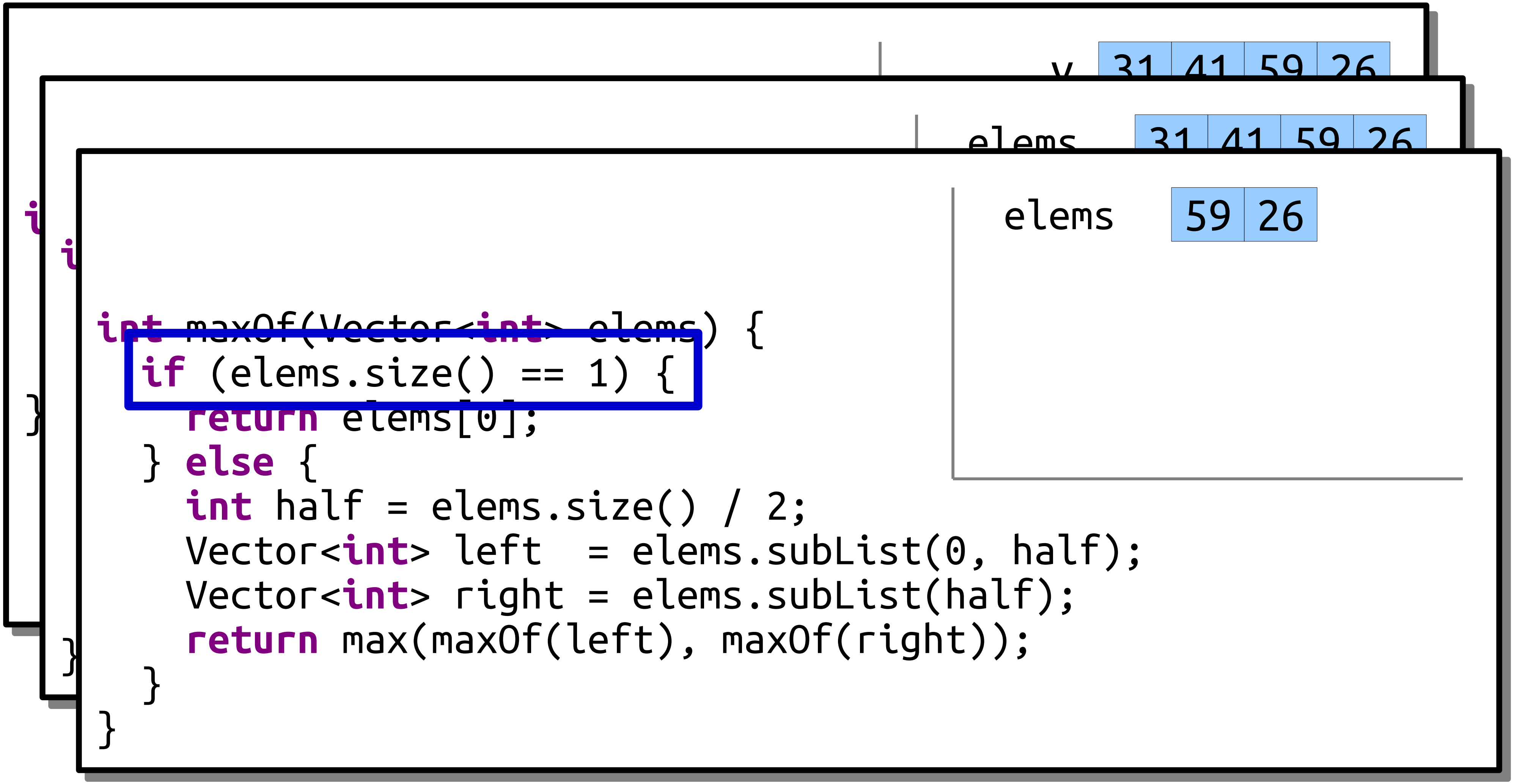
v 31 41 59 26

elems 31 41 59 26

elems 59 26

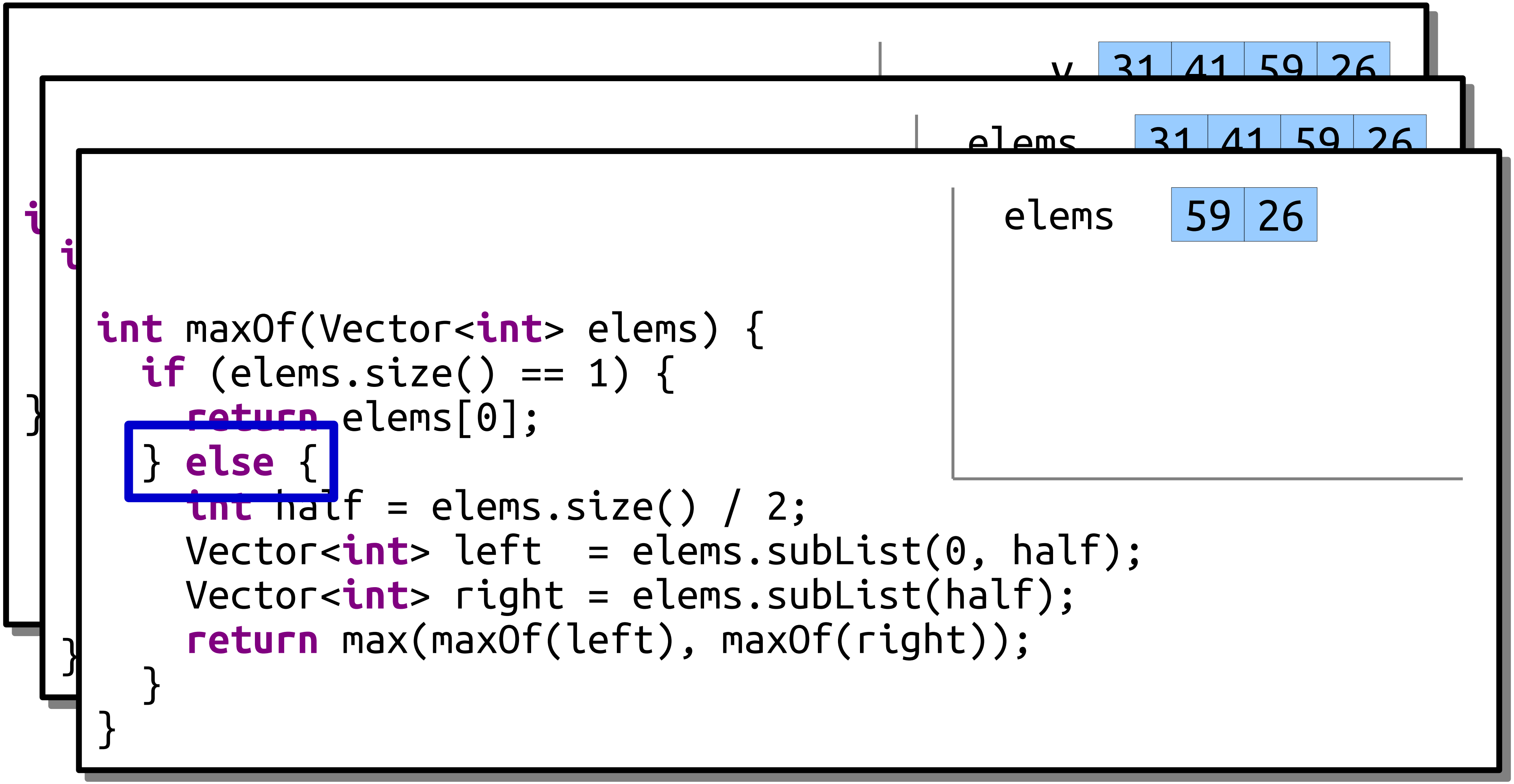
```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

Tracing the Recursion



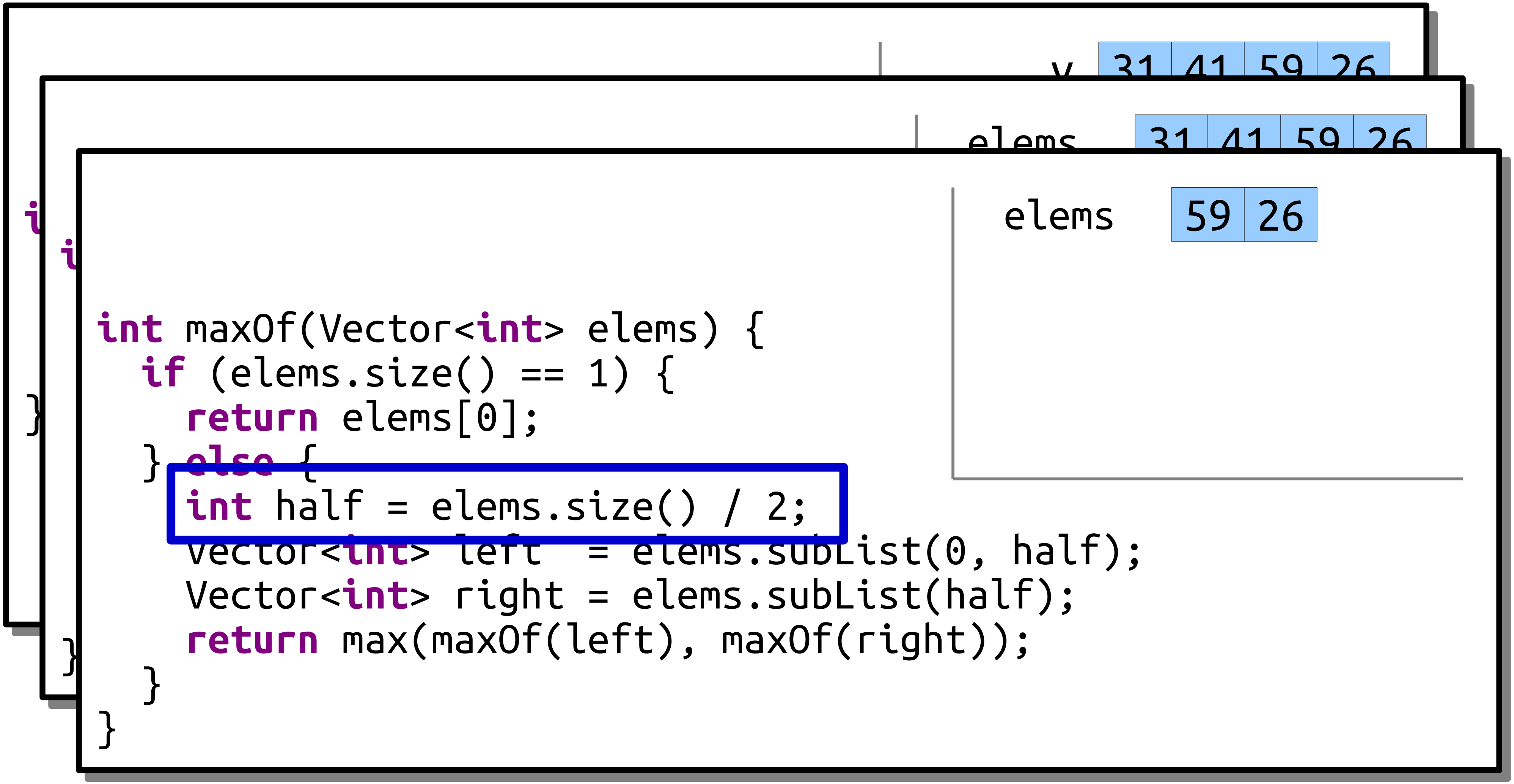
```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

Tracing the Recursion



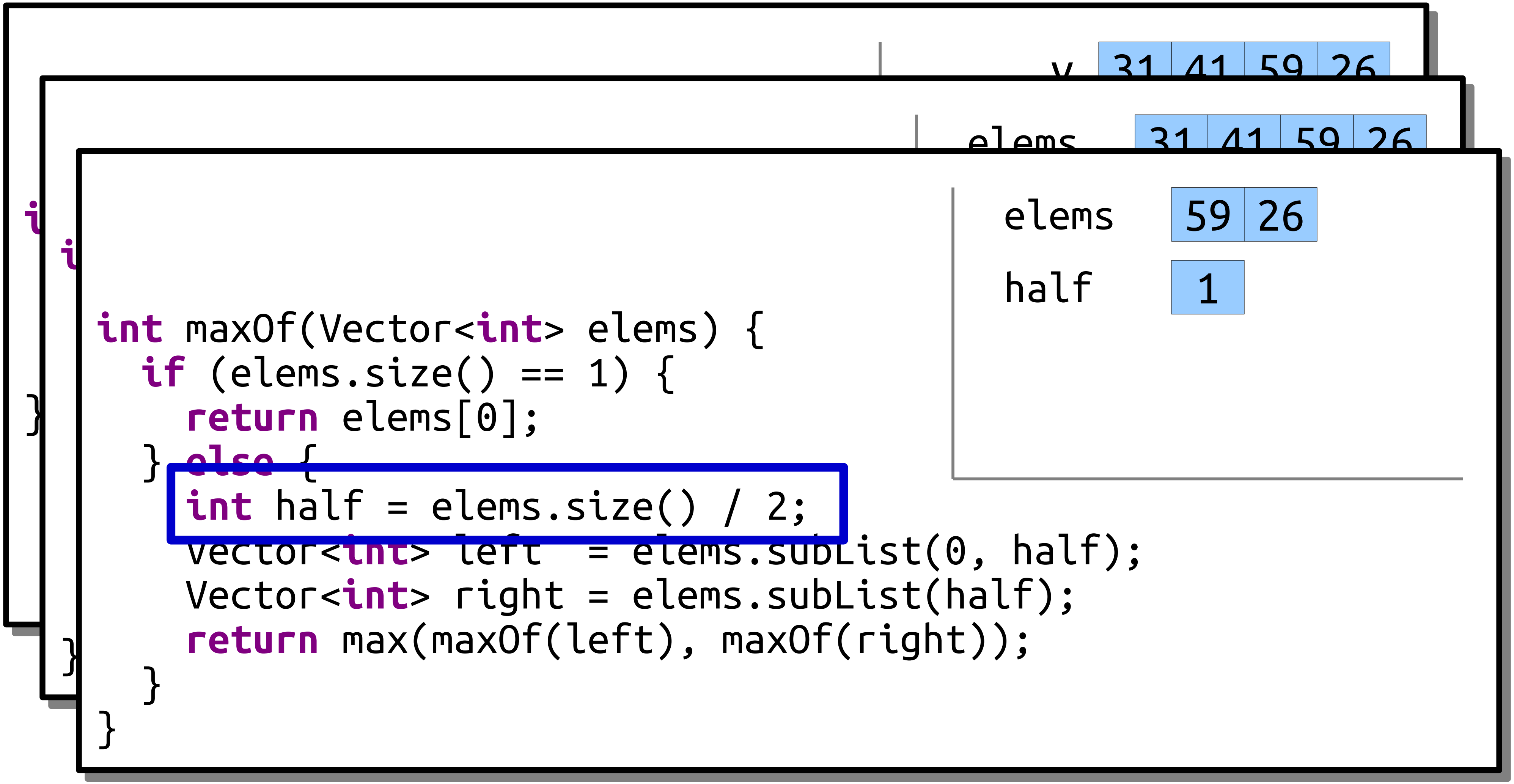
```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

Tracing the Recursion



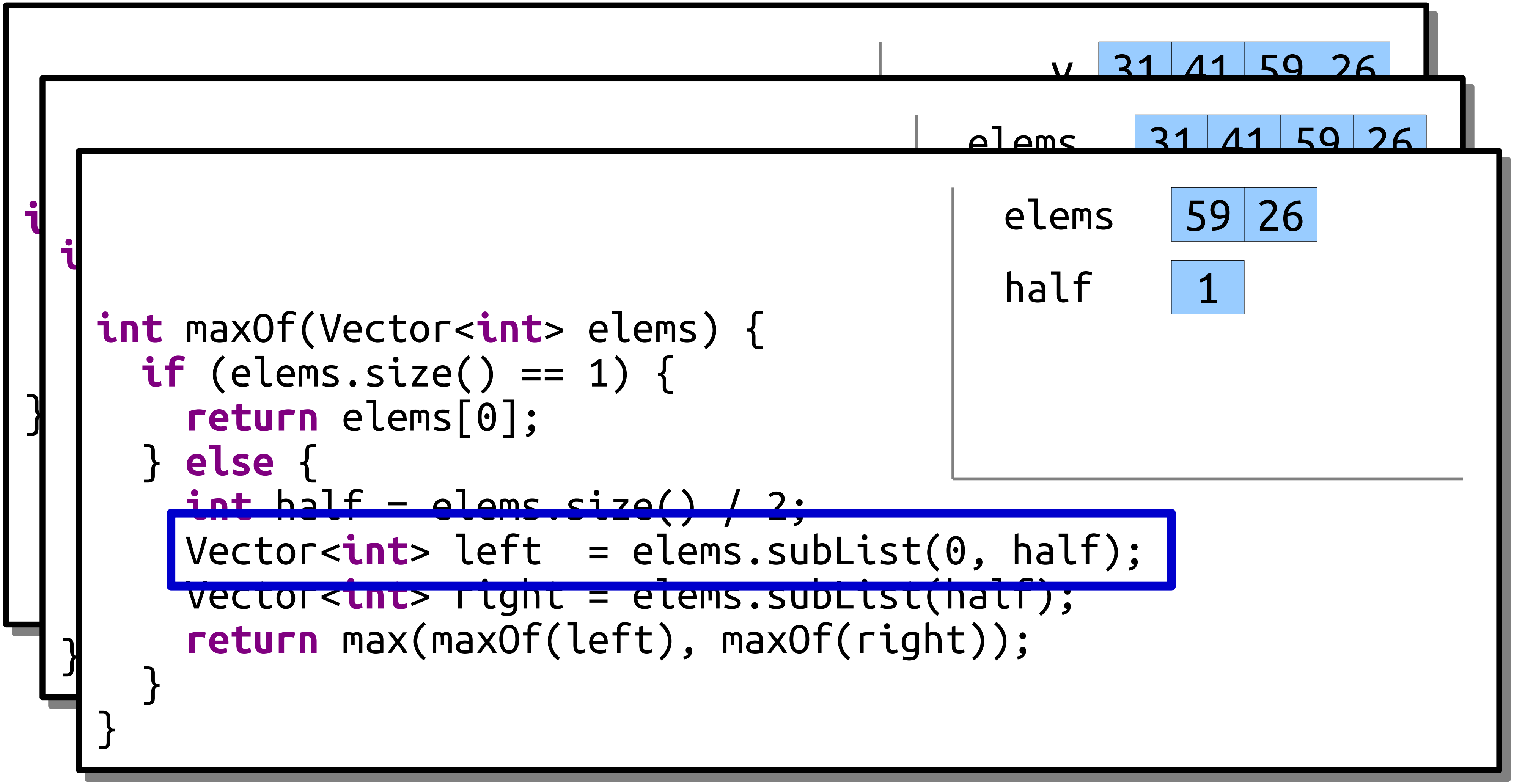
```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

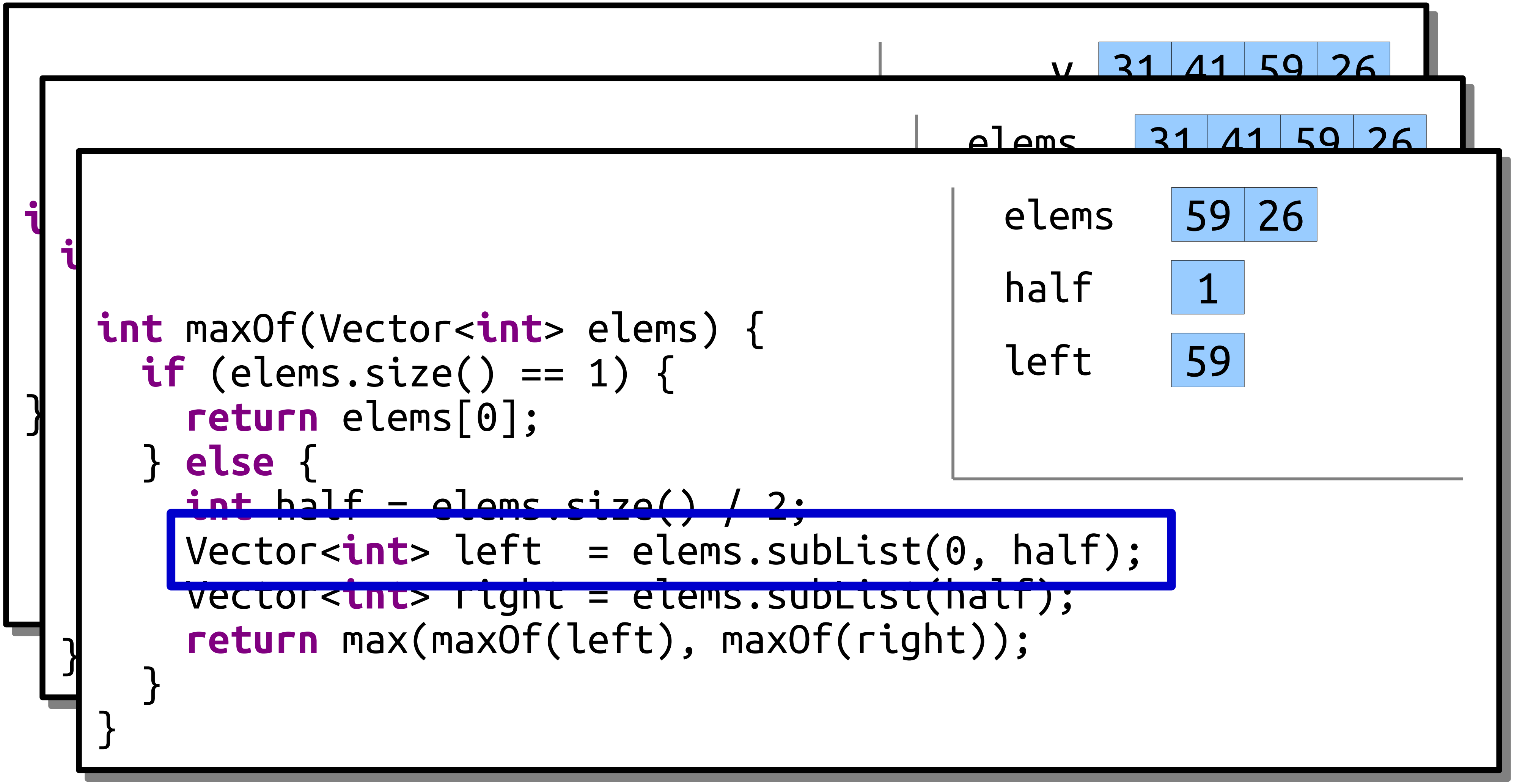
Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

elems	59	26
half	1	

Tracing the Recursion



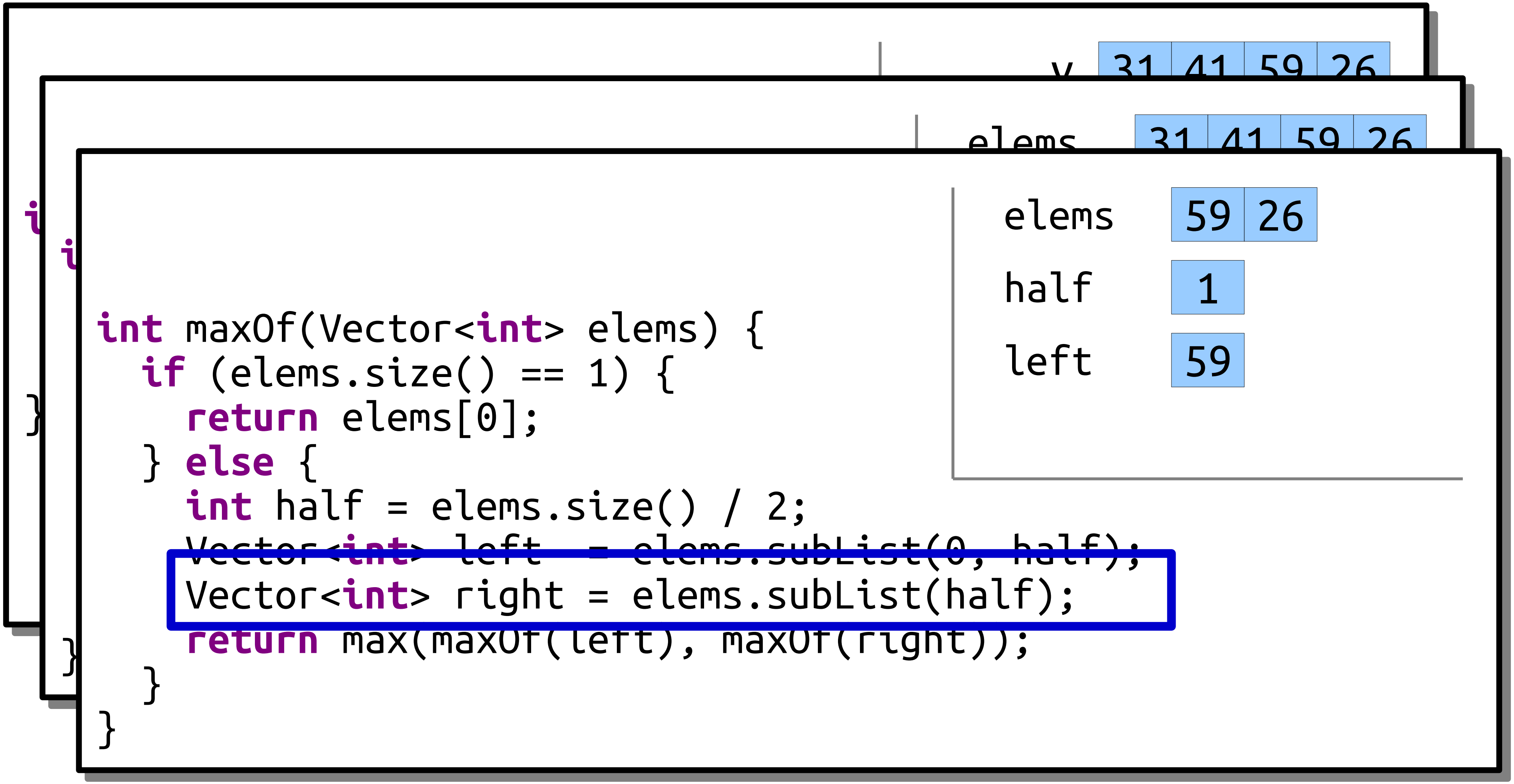
v 31 41 59 26

elems 31 41 59 26

```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

elems	59	26
half	1	
left	59	

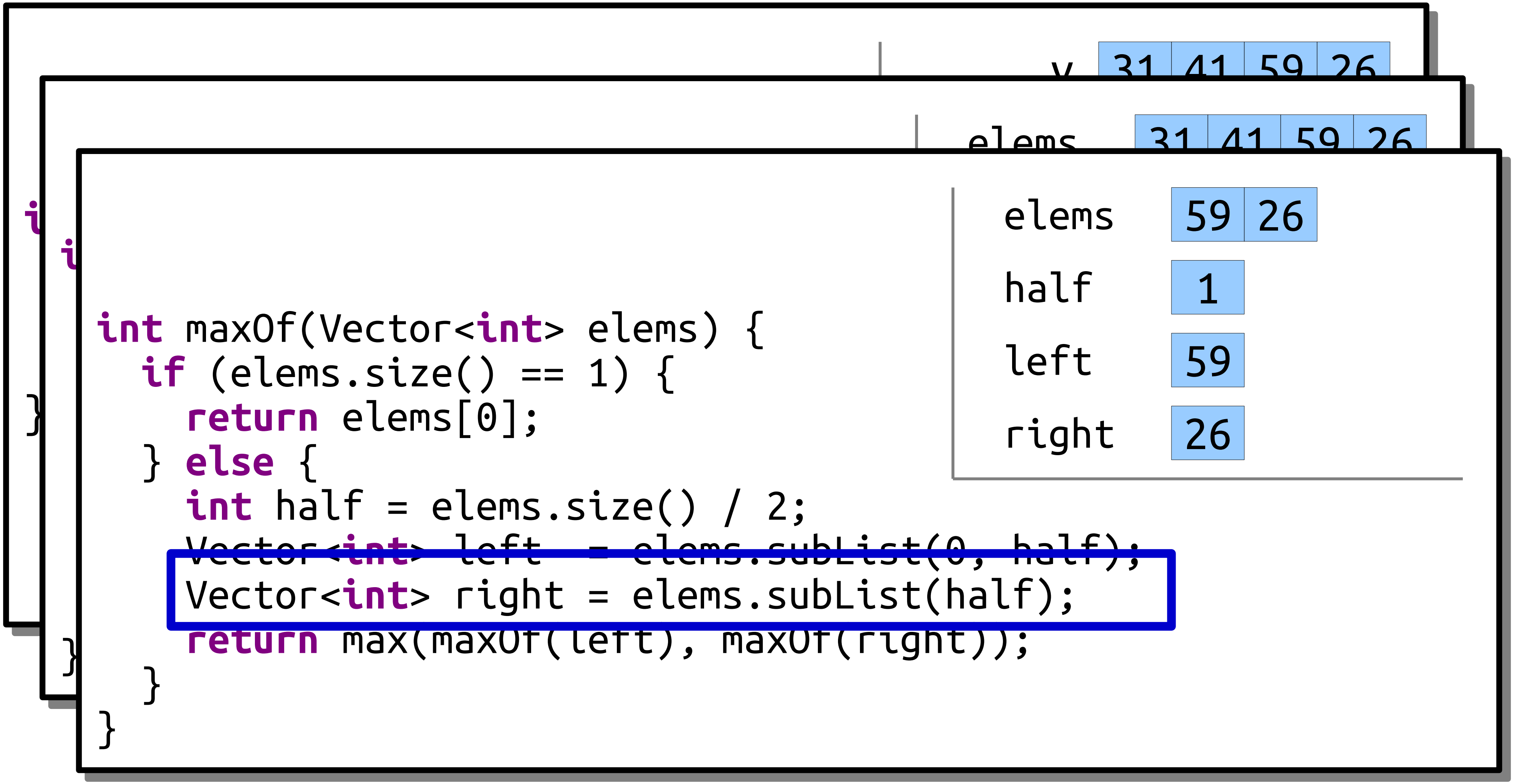
Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

elems	59	26
half	1	
left	59	

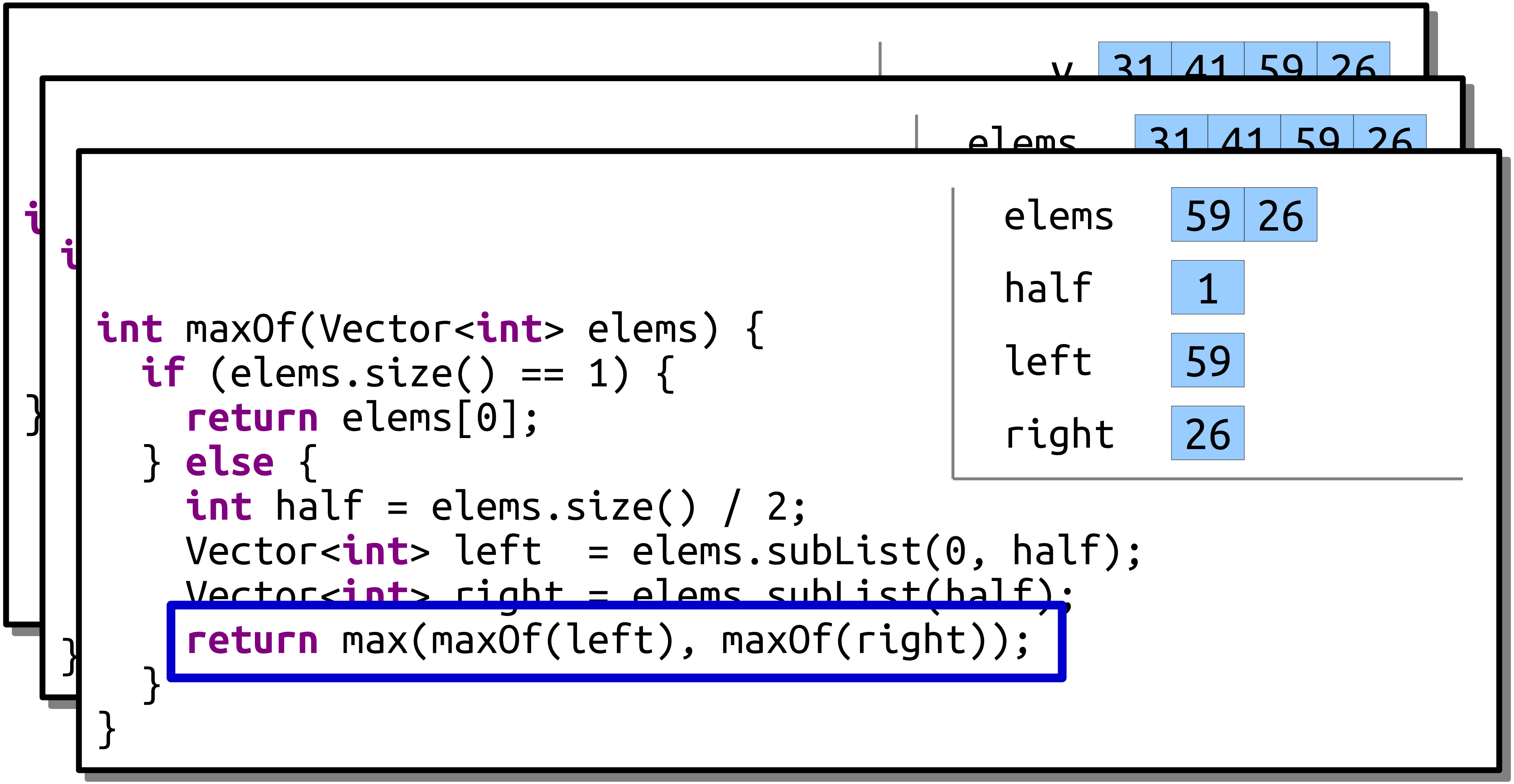
Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

elems	59 26
half	1
left	59
right	26

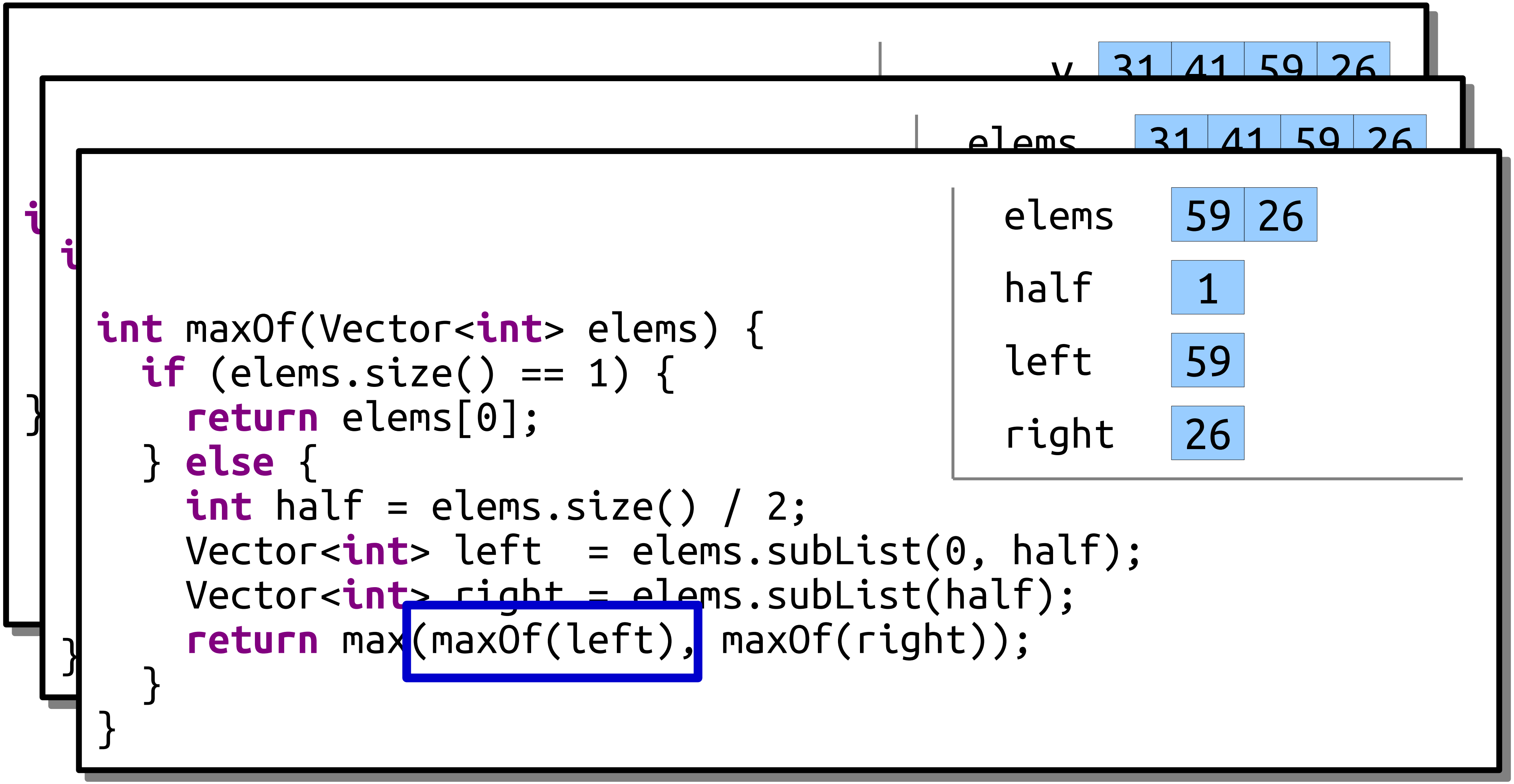
Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

elems	59	26
half	1	
left	59	
right	26	

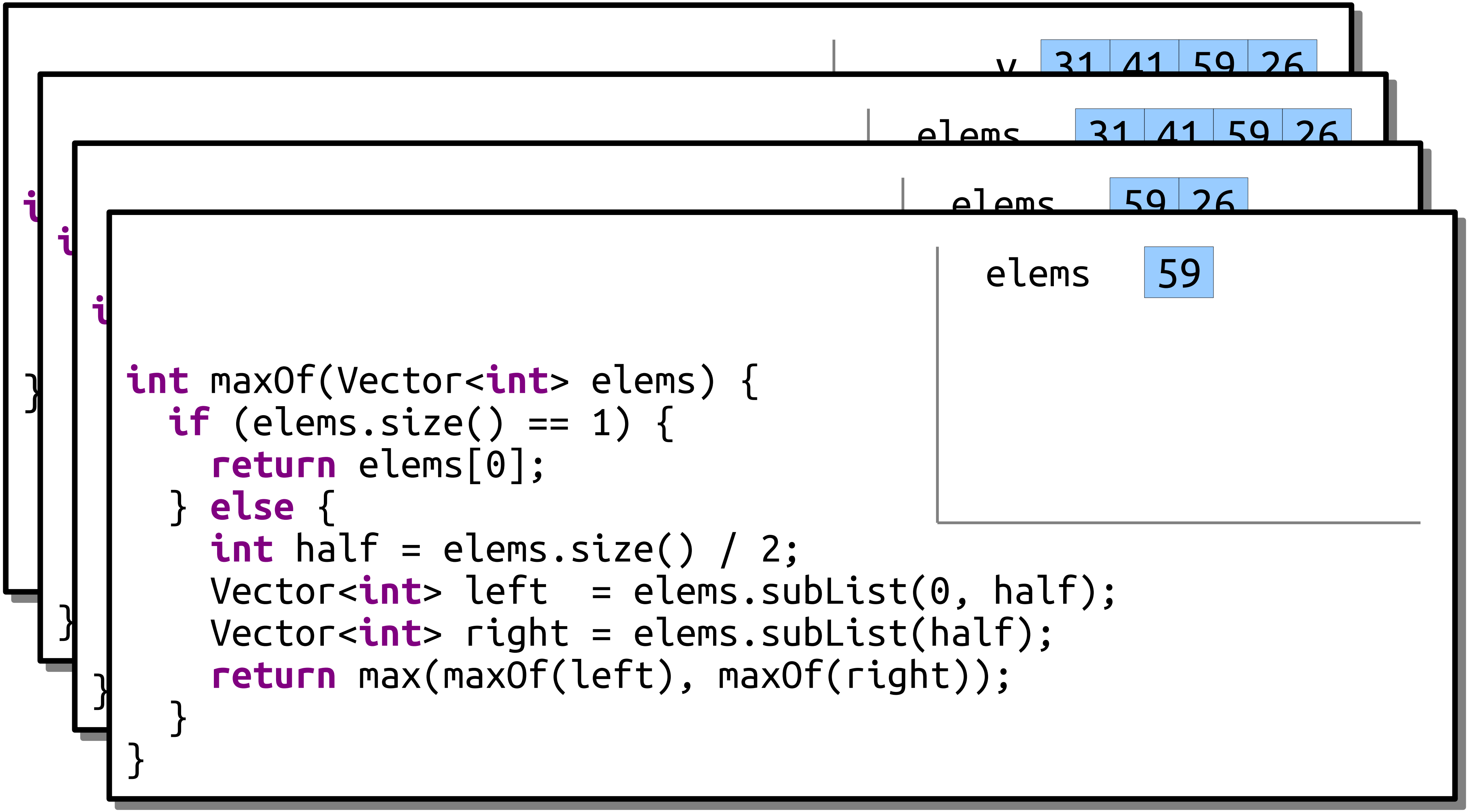
Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

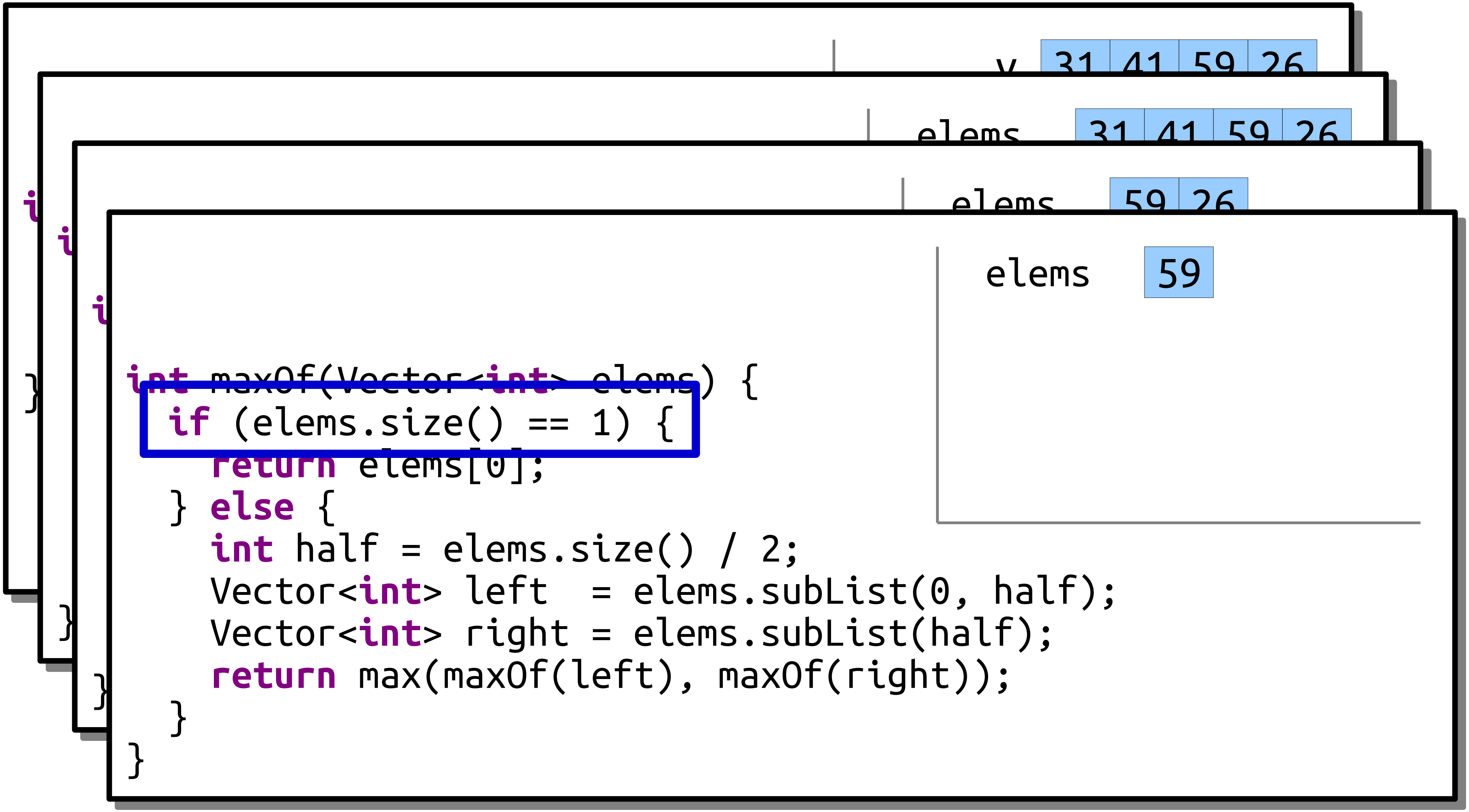
elems	59 26
half	1
left	59
right	26

Tracing the Recursion



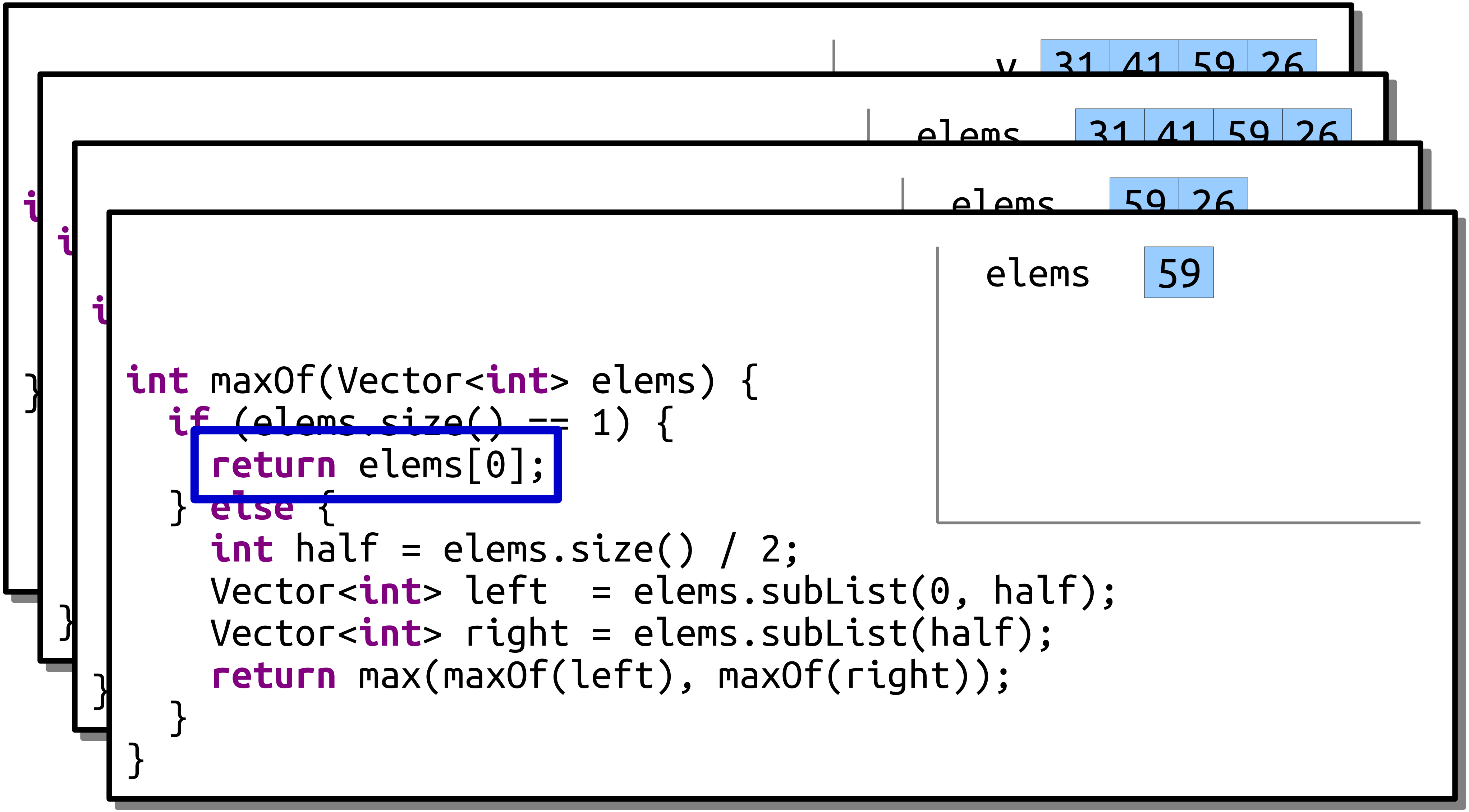
```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

Tracing the Recursion



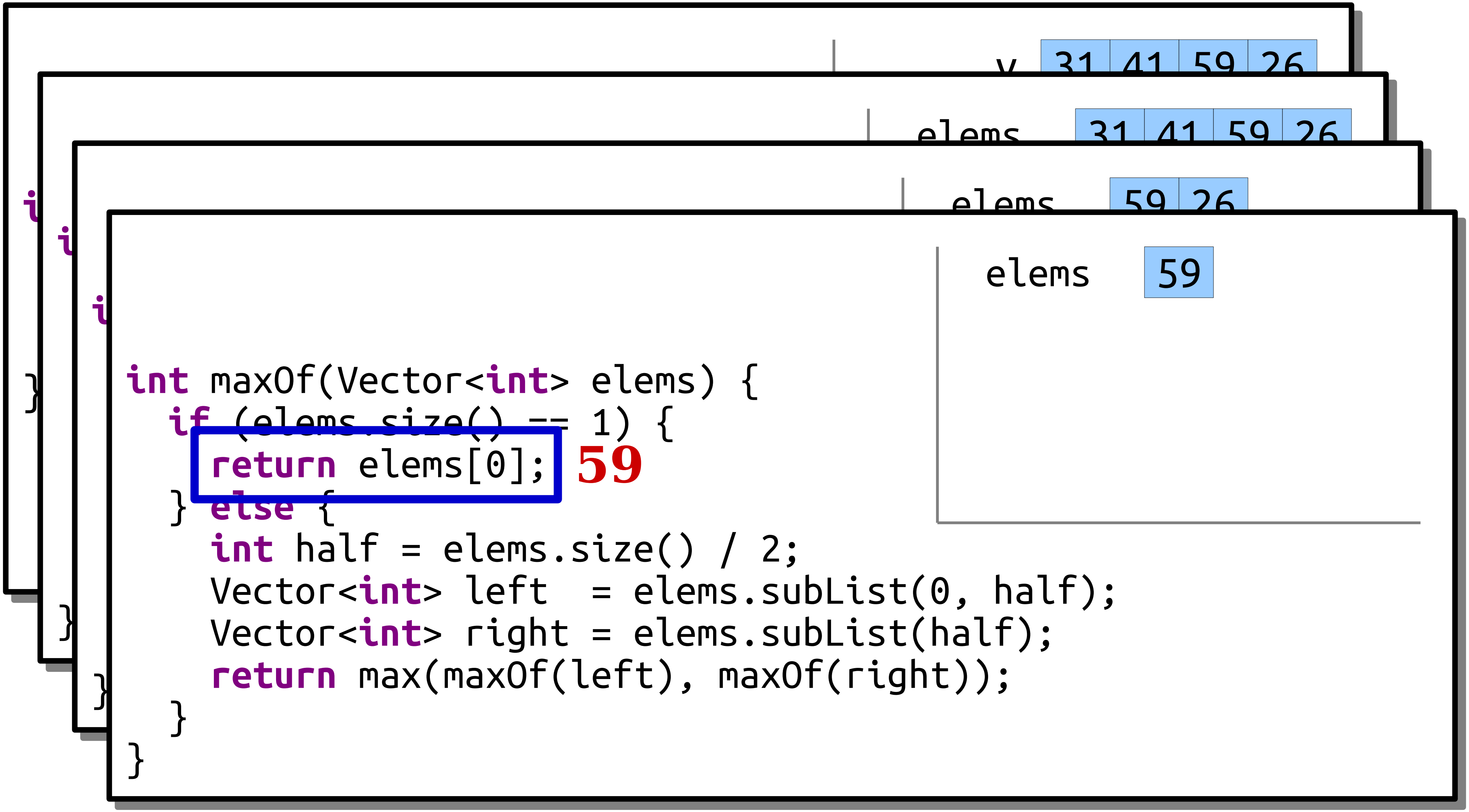
```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

Tracing the Recursion



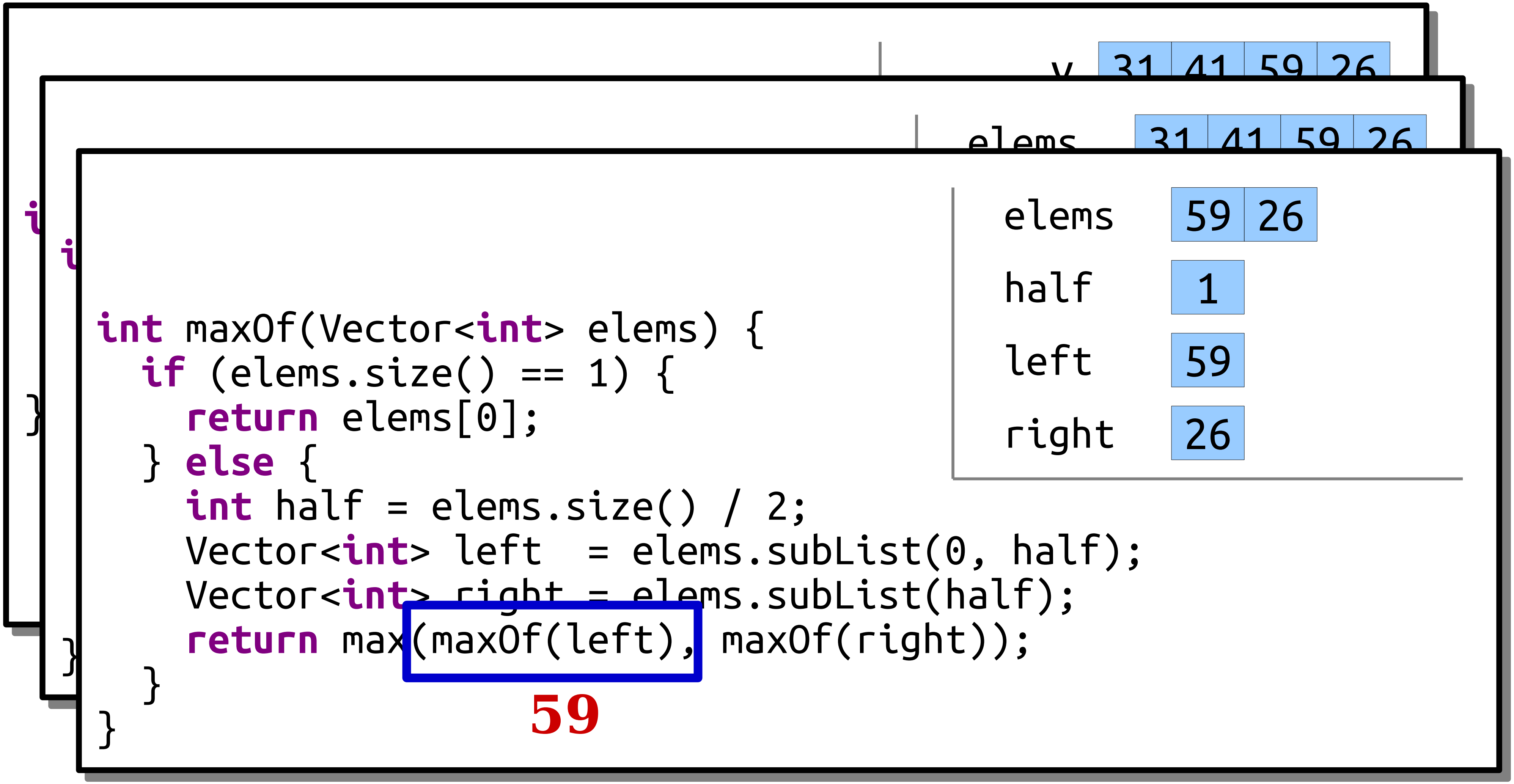
```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

Tracing the Recursion

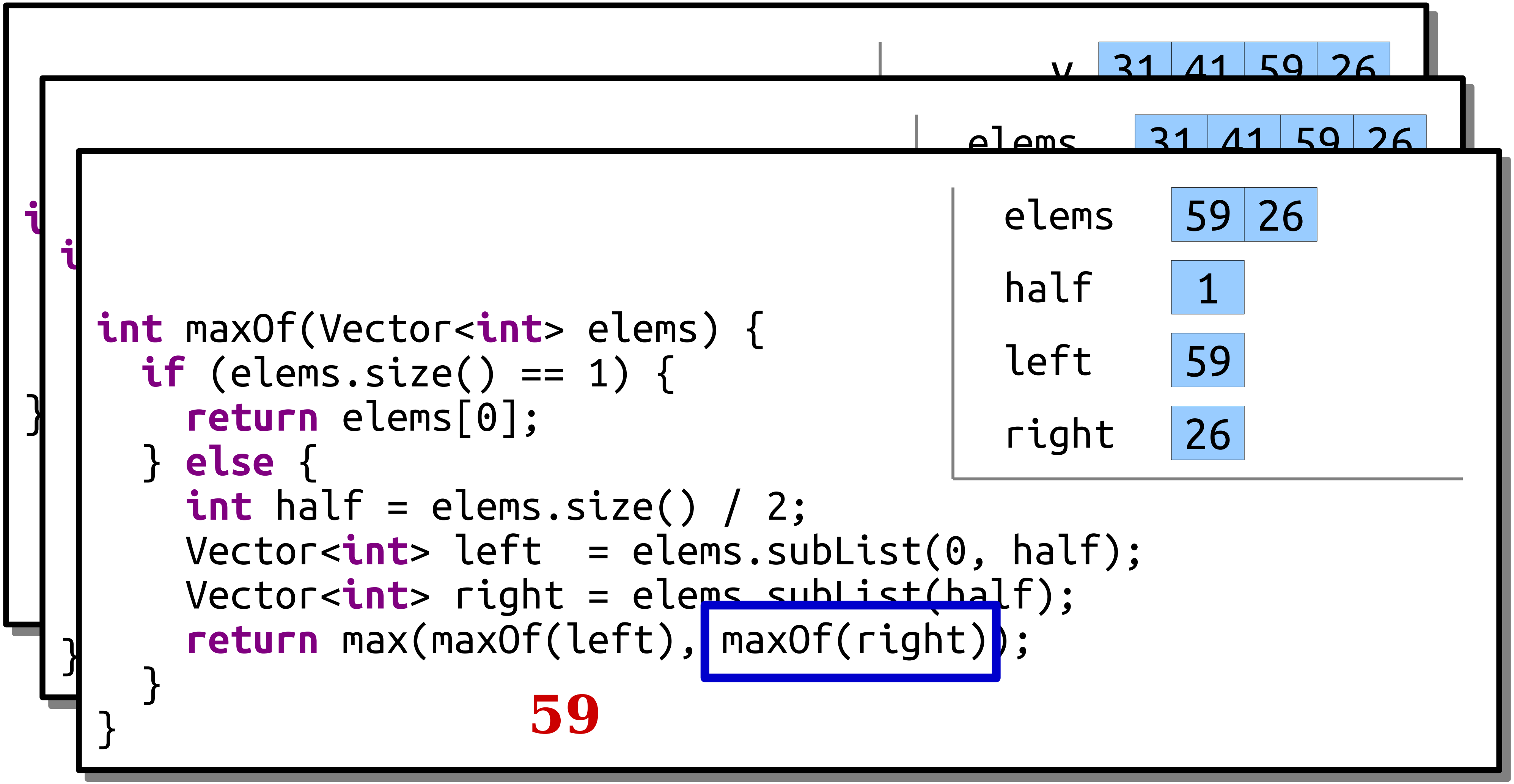


```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

elems	59 26
half	1
left	59
right	26

59

Tracing the Recursion

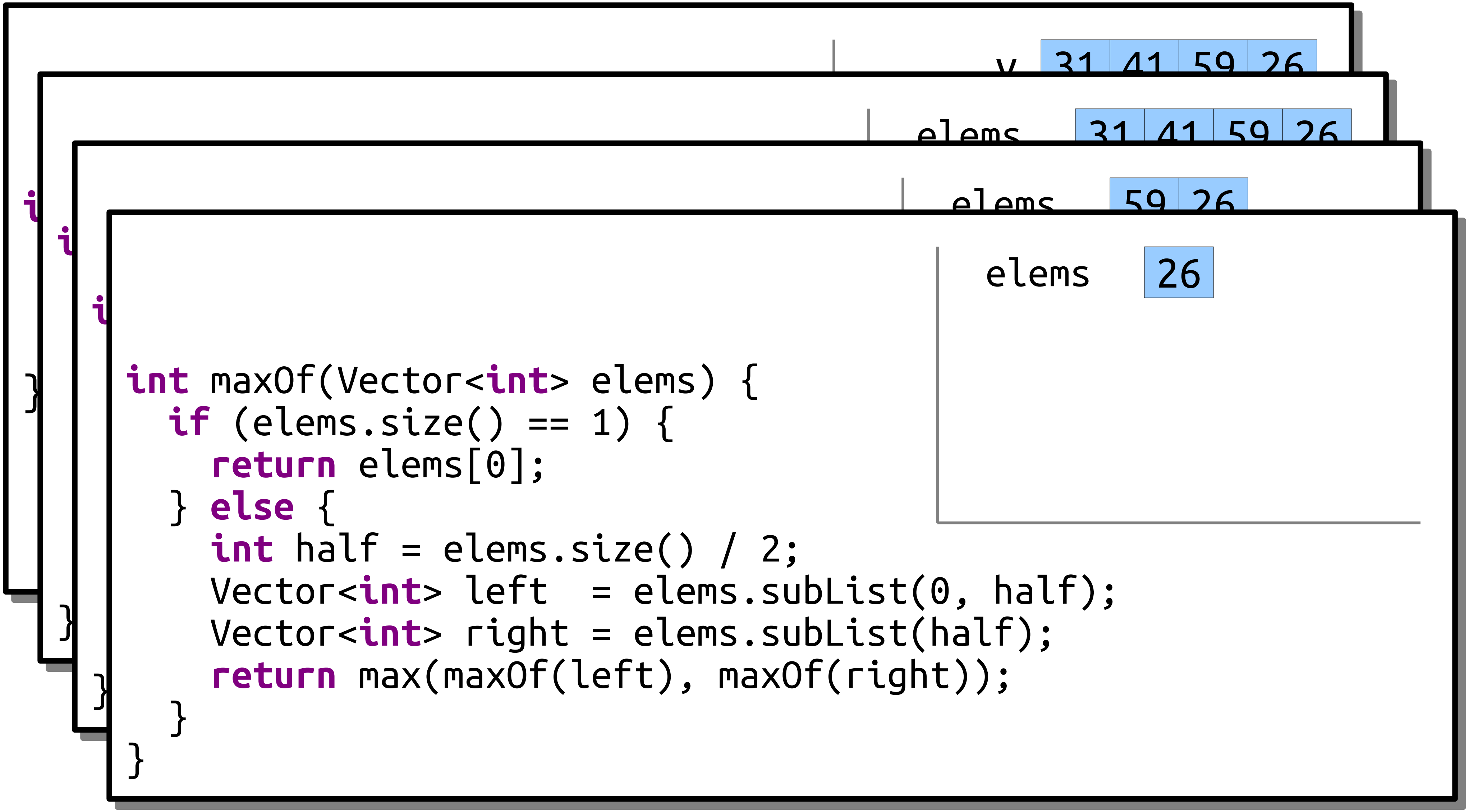


```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

59

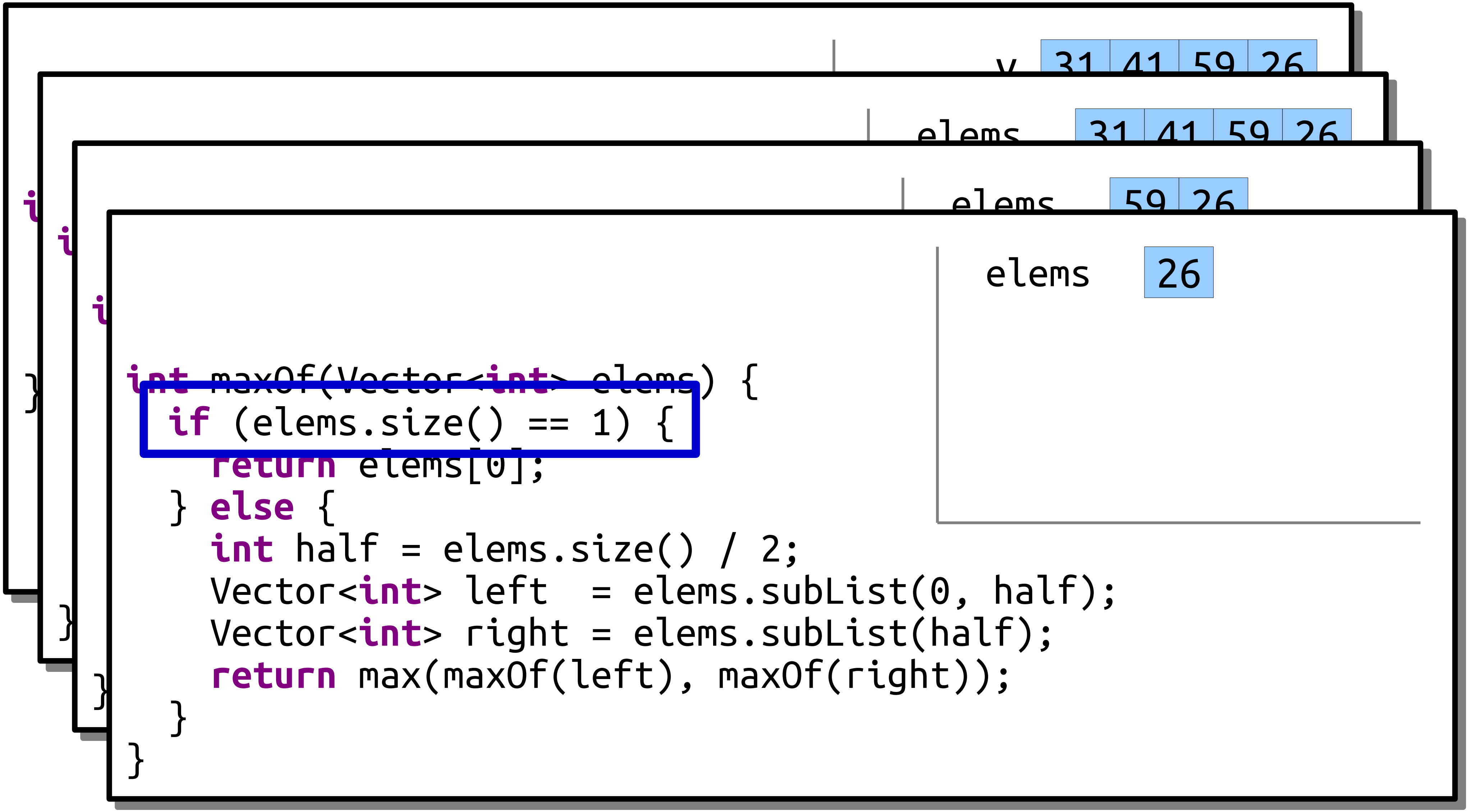
elems	59 26
half	1
left	59
right	26

Tracing the Recursion



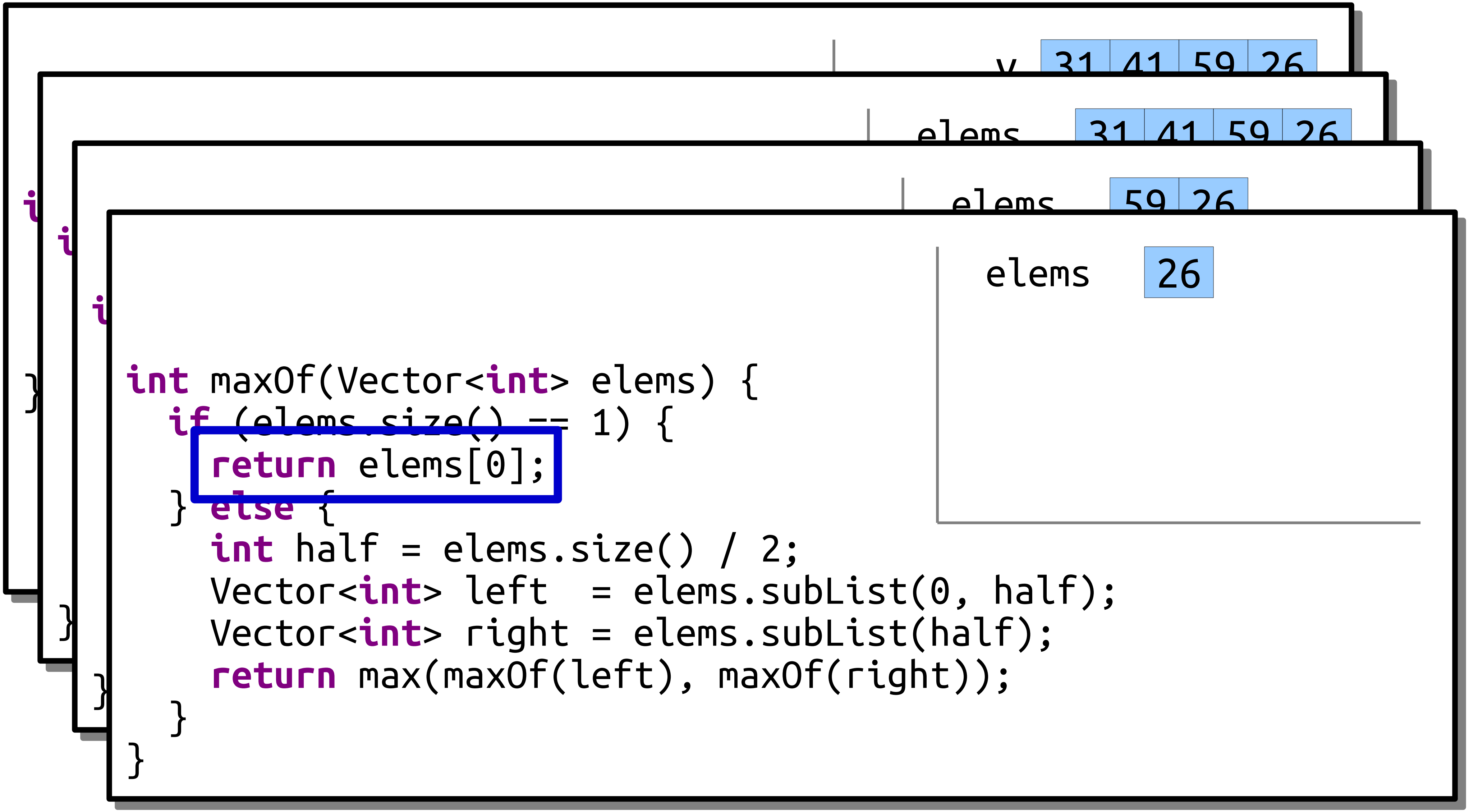
```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

Tracing the Recursion



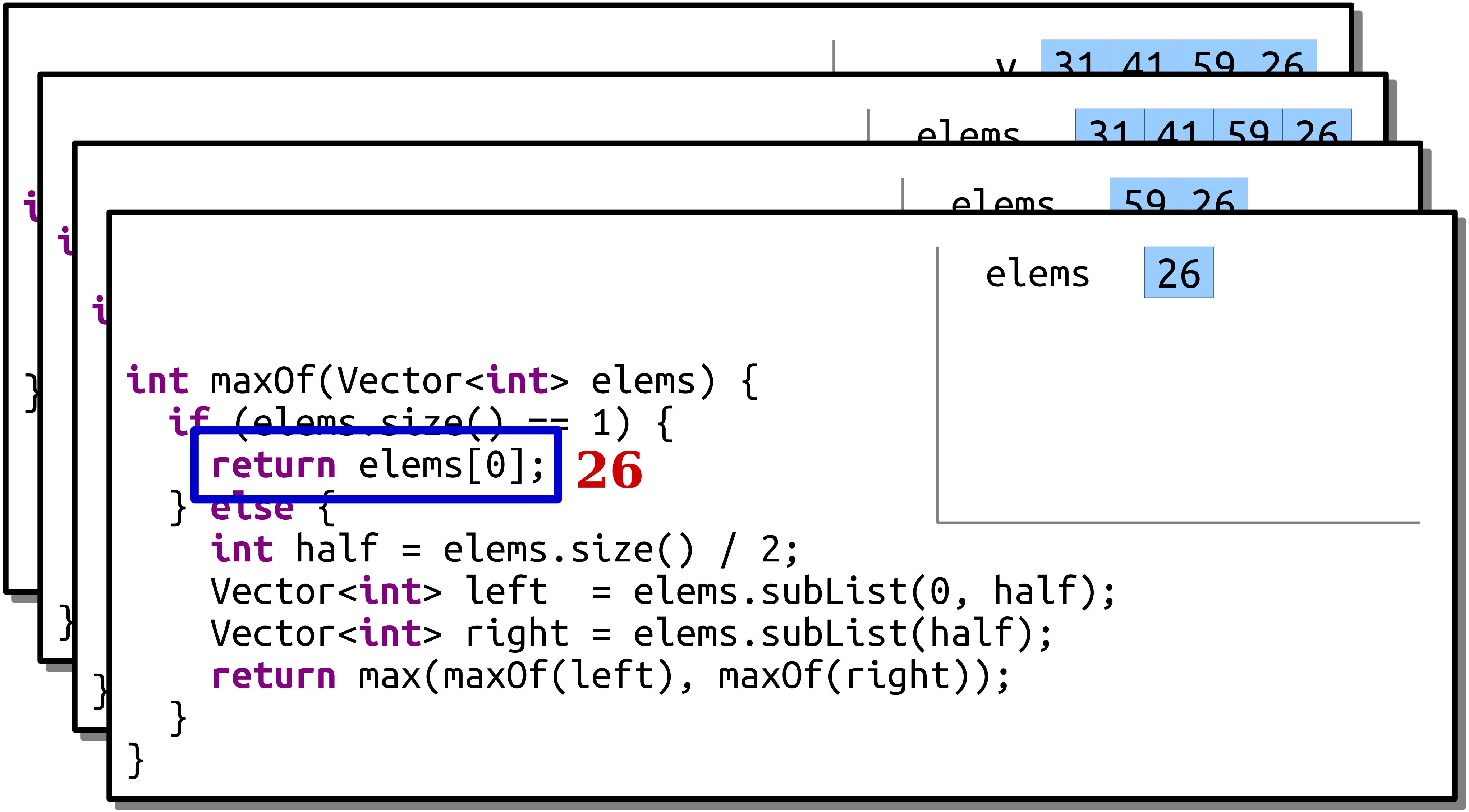
```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

Tracing the Recursion



v 31 41 59 26

elems 31 41 59 26

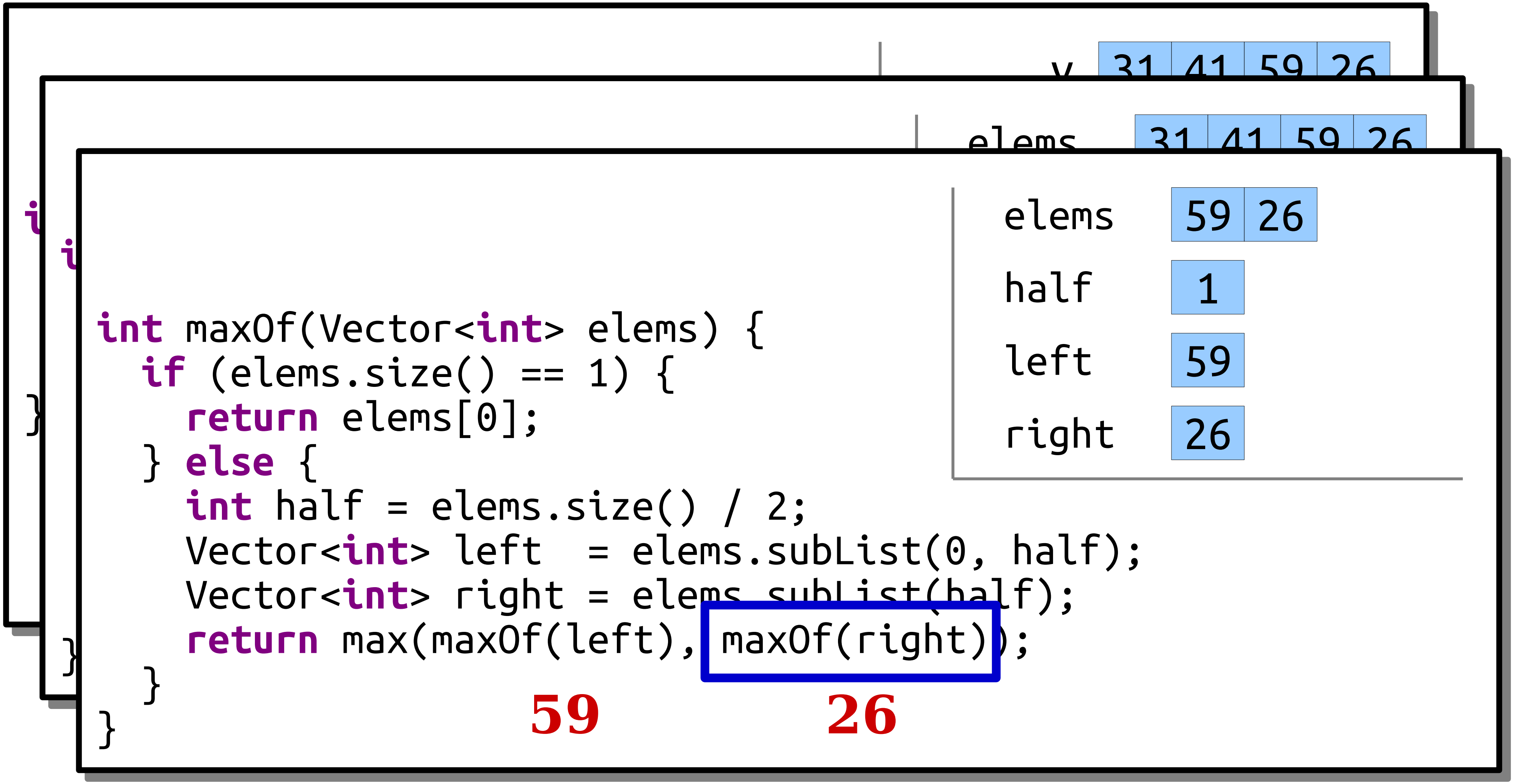
elems 59 26

elems 26

```
int maxOf(Vector<int> elems) {
   if (elems.size() == 1) {
      return elems[0];
   } else {
      int half = elems.size() / 2;
      Vector<int> left  = elems.subList(0, half);
      Vector<int> right = elems.subList(half);
      return max(maxOf(left), maxOf(right));
   }
}
```

26

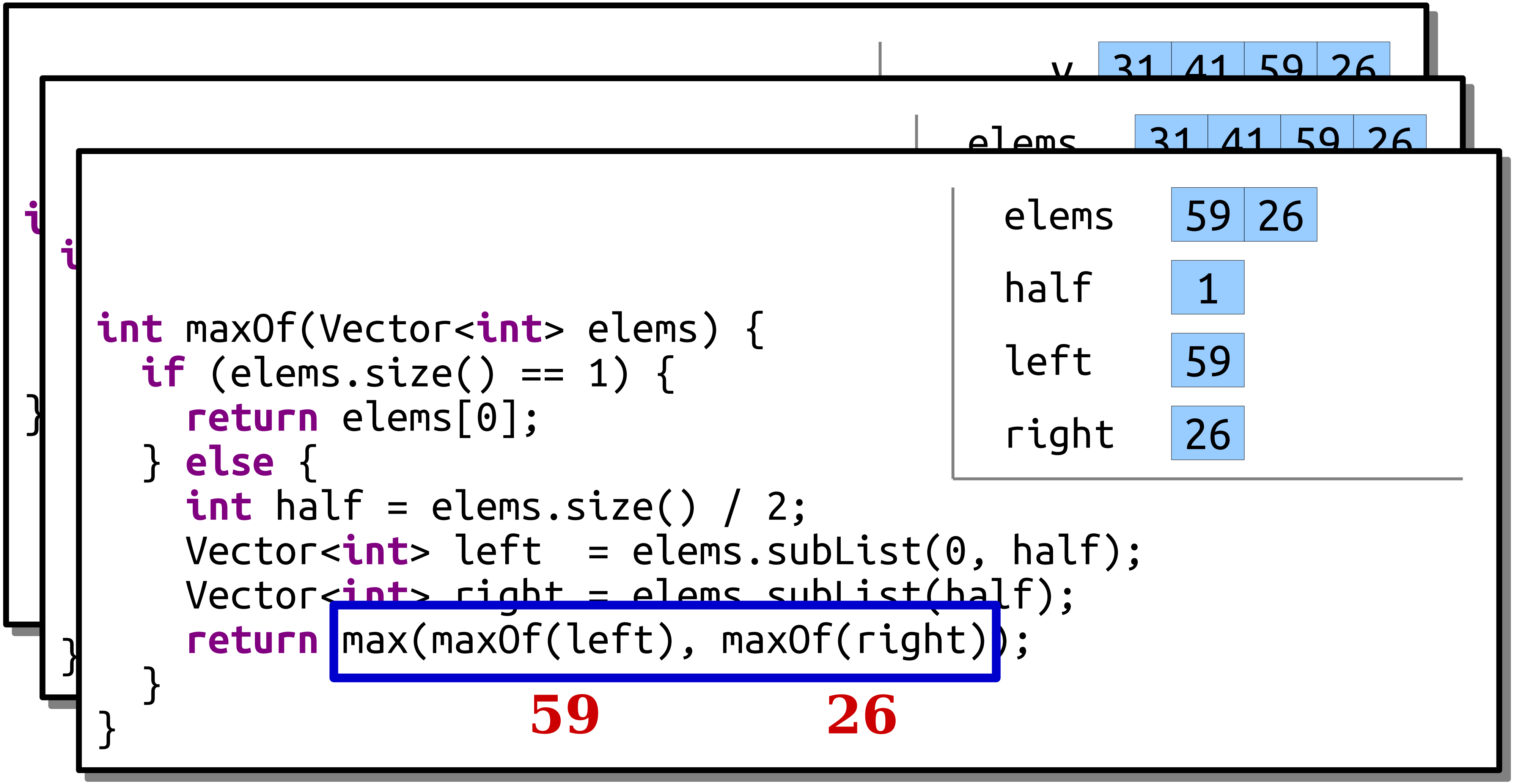
Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

elems	59 26
half	1
left	59
right	26

Tracing the Recursion

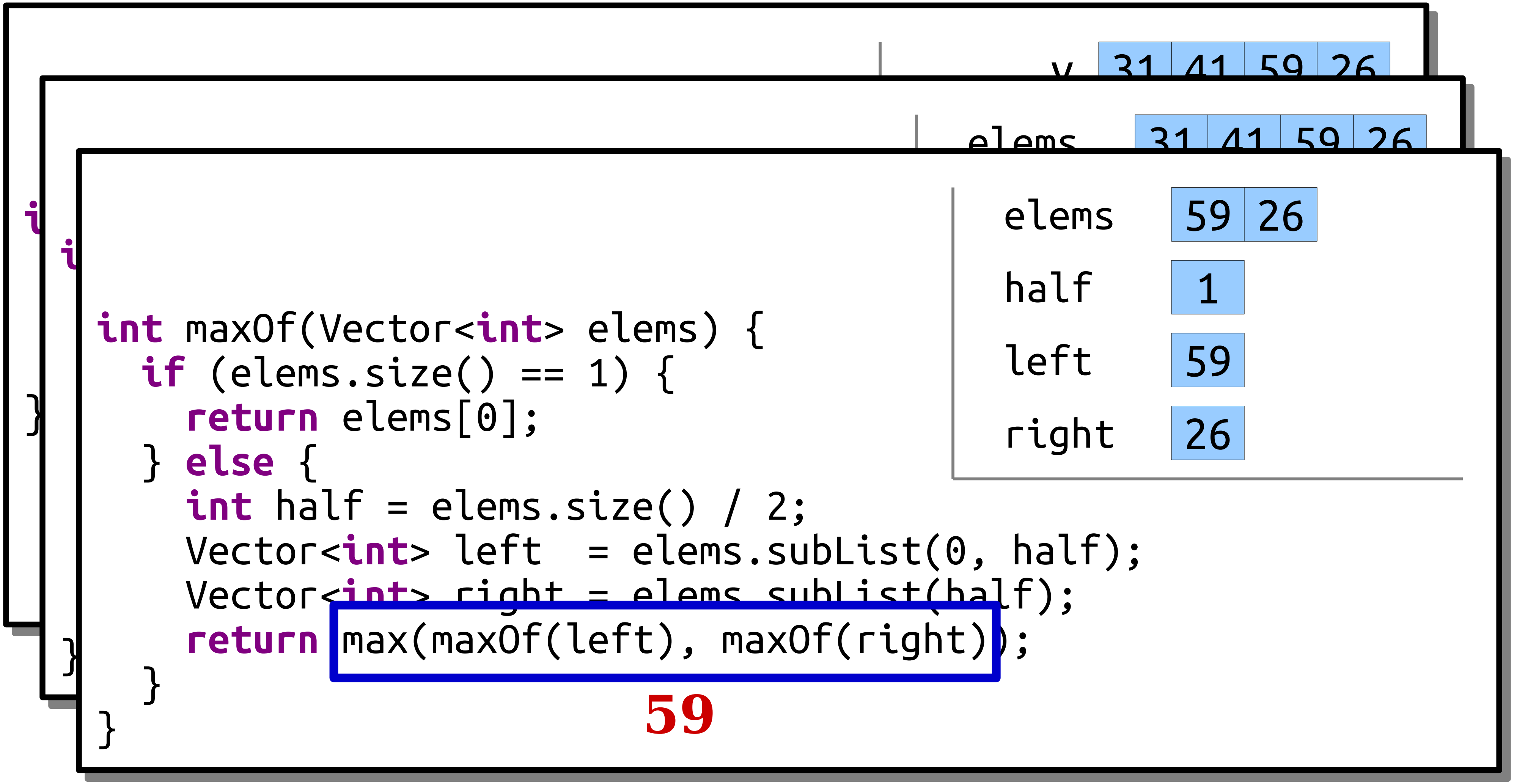


```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

elems	59	26
half	1	
left	59	
right	26	

59 26

Tracing the Recursion

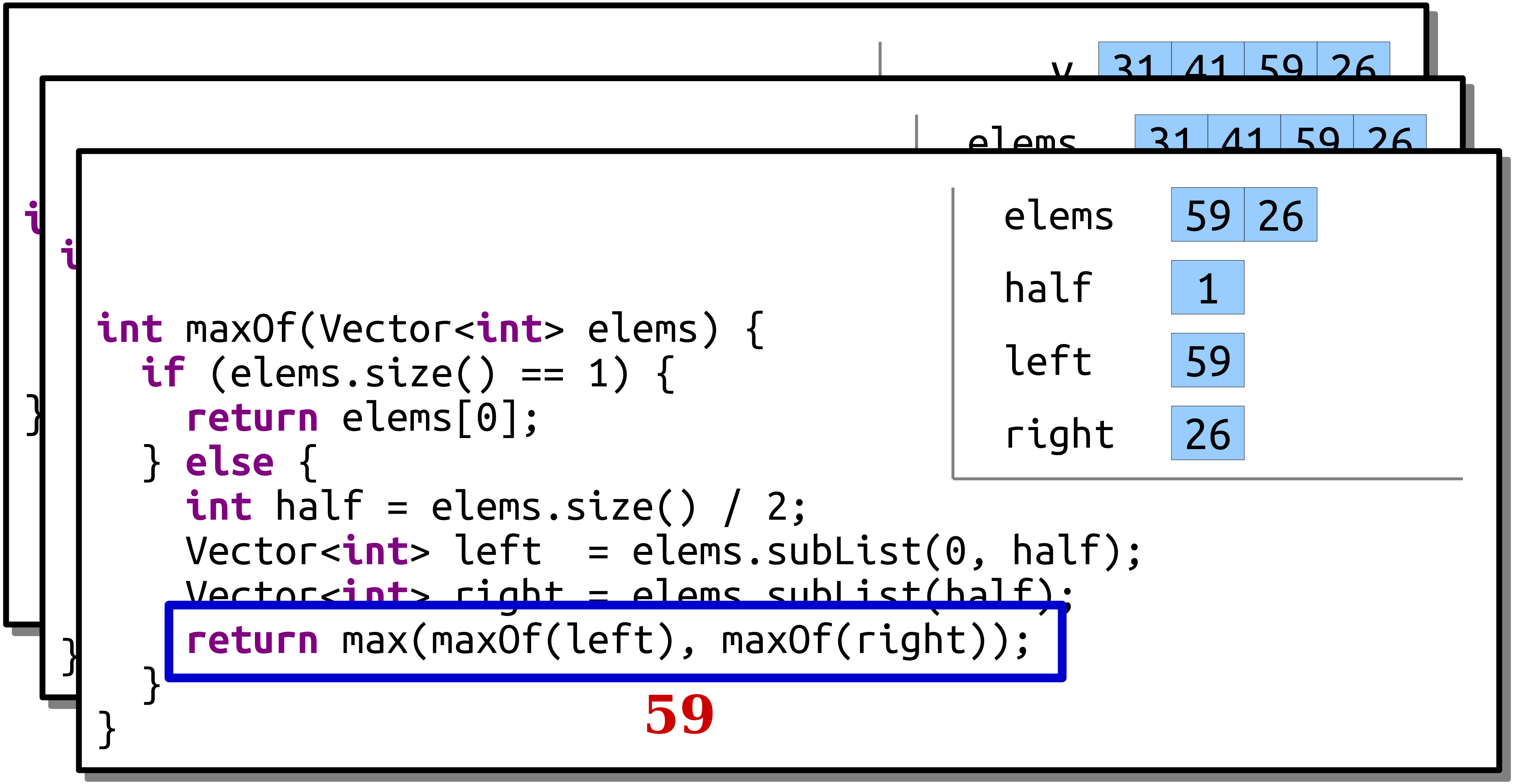


```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

59

elems	59 26
half	1
left	59
right	26

Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

elems	59	26
half	1	
left	59	
right	26	

59

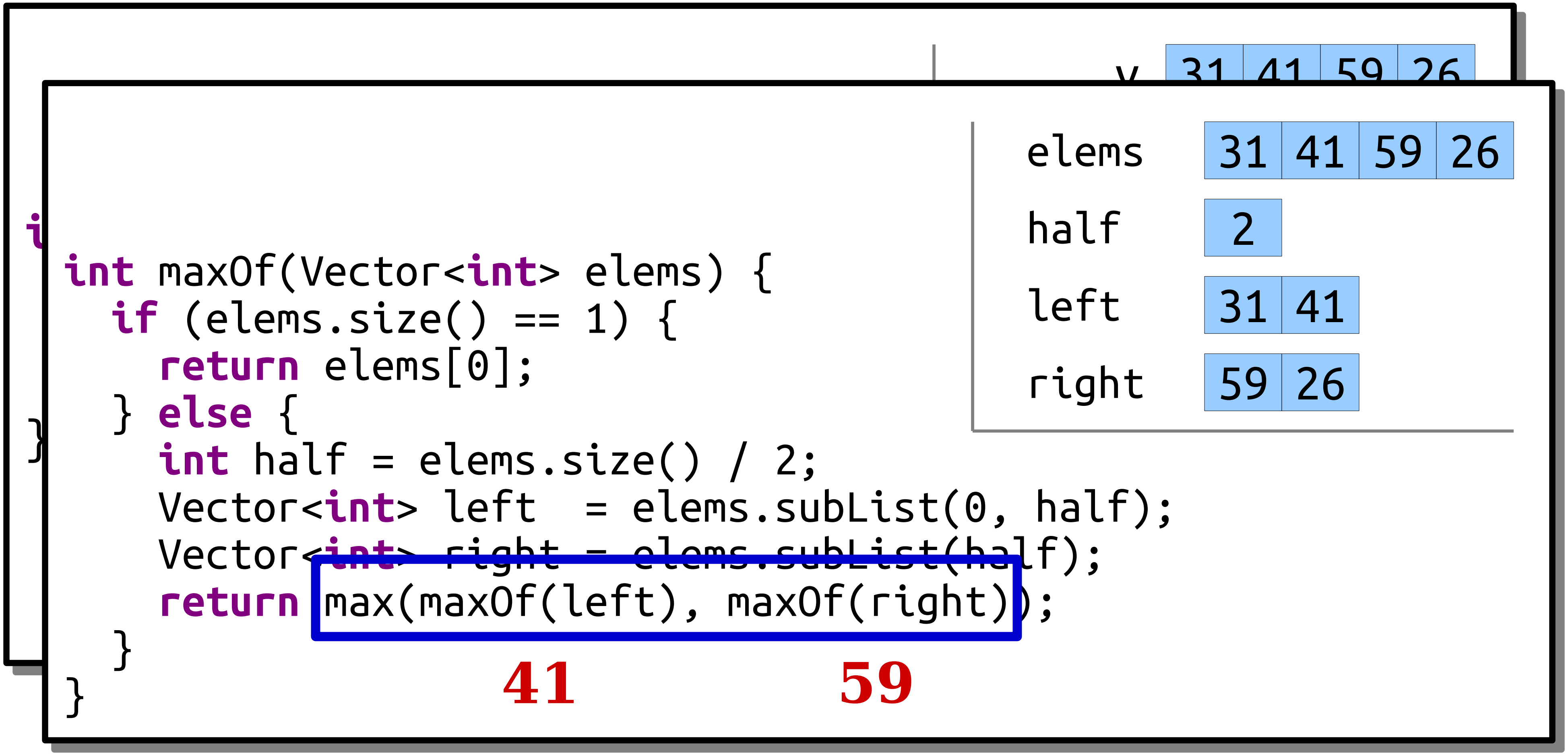
Tracing the Recursion

```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

41 59

elems	31	41	59	26
half	2			
left	31	41		
right	59	26		

Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

elems	31	41	59	26
half	2			
left	31	41		
right	59	26		

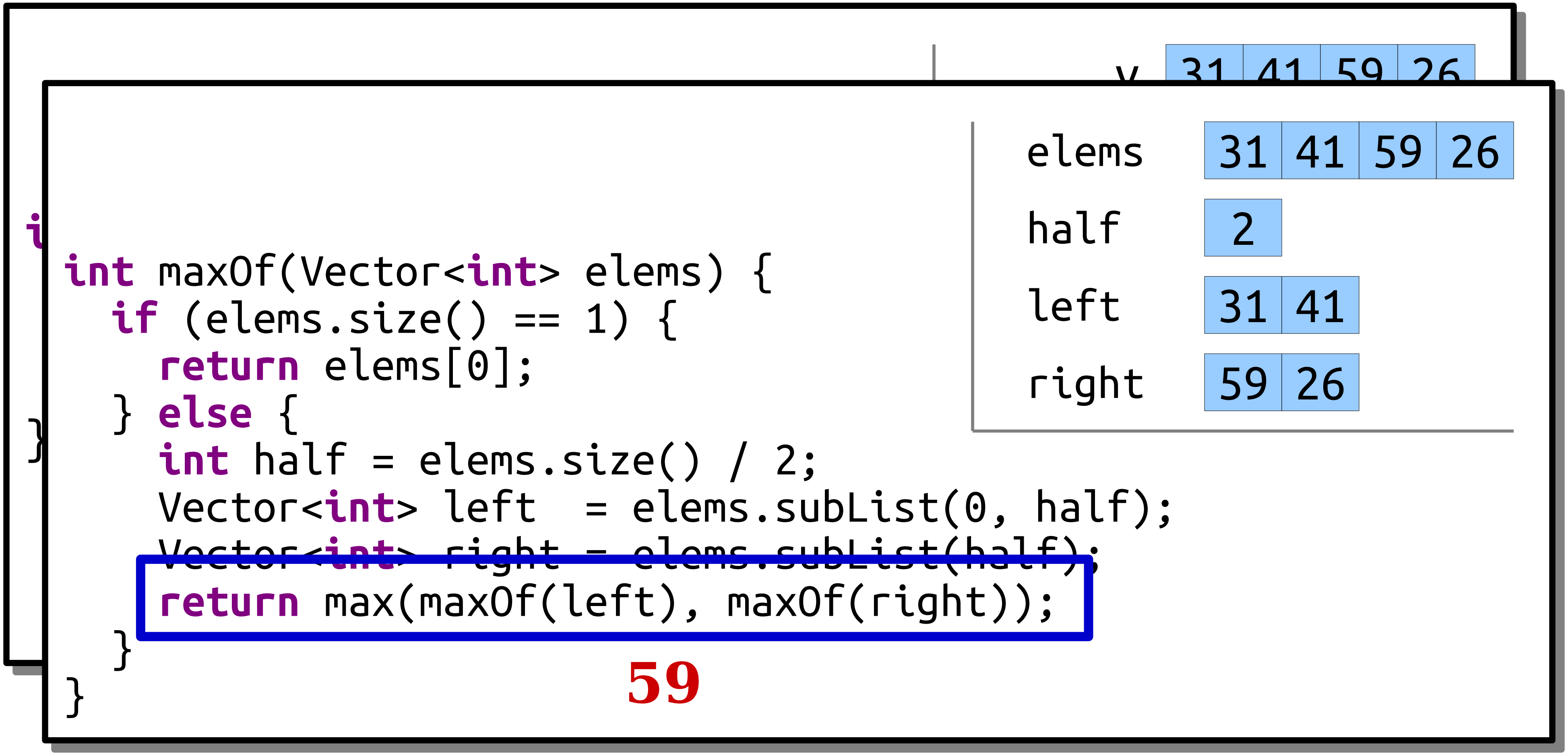
Tracing the Recursion

elems	31 41 59 26
half	2
left	31 41
right	59 26

```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

59

Tracing the Recursion



```
int maxOf(Vector<int> elems) {
  if (elems.size() == 1) {
    return elems[0];
  } else {
    int half = elems.size() / 2;
    Vector<int> left  = elems.subList(0, half);
    Vector<int> right = elems.subList(half);
    return max(maxOf(left), maxOf(right));
  }
}
```

elems	31	41	59	26
half	2			
left	31	41		
right	59	26		

Tracing the Recursion

v	31	41	59	26

```
int main() {
    Vector<int> v = { 31, 41, 59, 26 };
    cout << maxOf(v) << endl;
    return 0;
}
```

59

Summary from Today

- The `Vector<T>` type in C++ represents a sequence of elements.
- Parameters in C++ are passed by *value* by default. You can change that to use pass by *reference* if you'd like.
- You can write the same recursive function in many different ways.
- Each stack frame from a recursive function gets its own copies of all the local variables.

Your Action Items

- ***Read Chapter 5.1 of the textbook.***
  - It's all about `Vector`! There are some goodies there.
- ***Work on Assignment 1.***
  - Aim to complete all three recursion problems by Tuesday evening.
    - Not done by then? Don't worry! Stop by the LaIR to ask questions.
  - Start working on Plotter.
- ***Explore the `maxOf` example.***
  - Tinker and play around with this one. See if you can get very comfortable with how it works.

Next Time

- ***Stacks***
  - How driveways relate to parentheses.
- ***Queues***
  - And a fun application. 😃