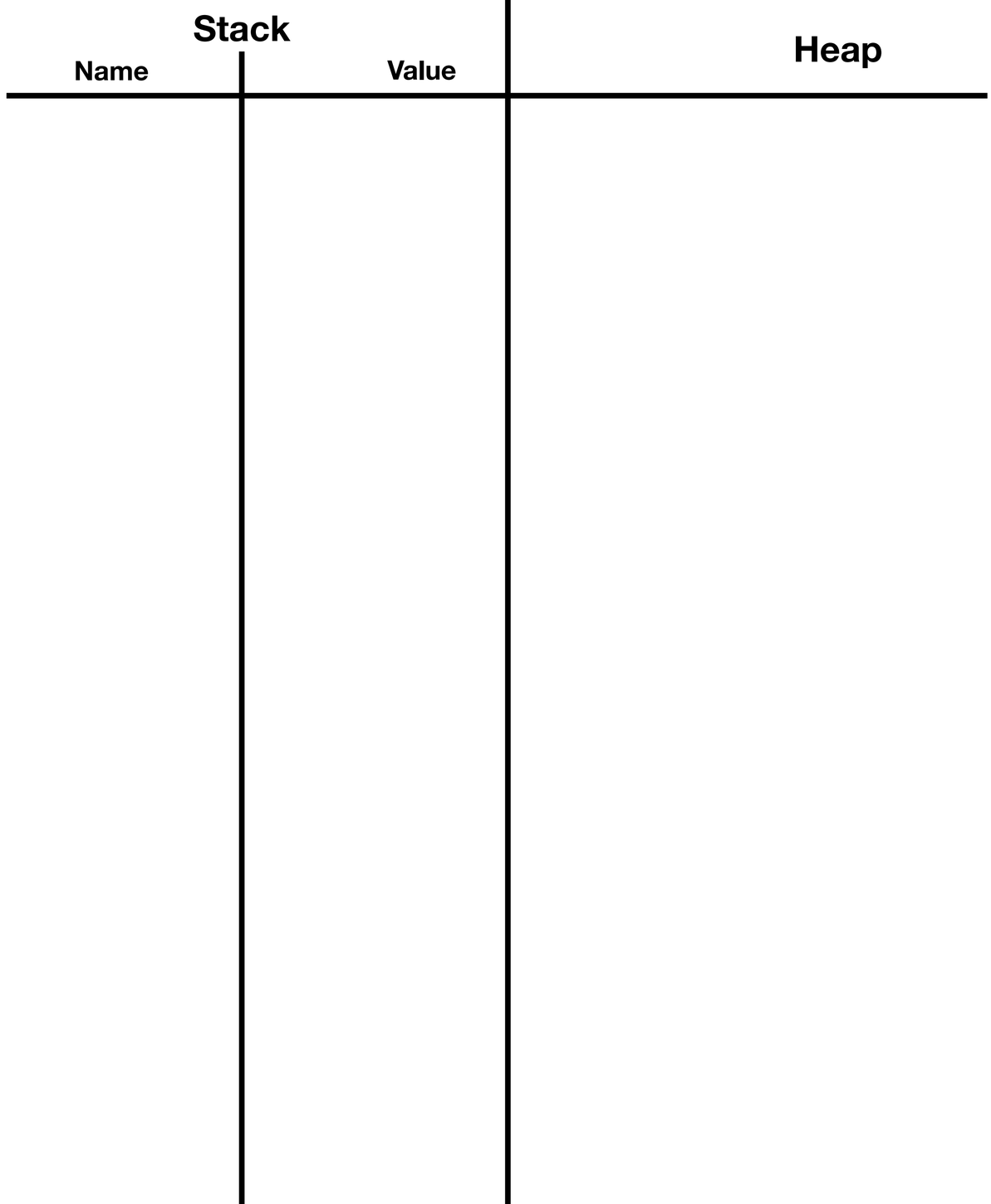
Tree Memory Diagram

```scala
class BTNode[A](var value: A, var left: BTNode[A], var right: BTNode[A])
```

```scala
def traversal[A](node: BTNode[A]): Unit = {
  if (node != null) {
    traversal(node.left)
    traversal(node.right)
    print(node.value + " ")
  }
}

def main(args: Array[String]): Unit = {
  val root = new BTNode("life", null, null)
  root.left = new BTNode("Scala", null, null)
  root.right = new BTNode("for", null, null)
  root.right.left = new BTNode("coding", null, null)
  traversal(root)
}
```

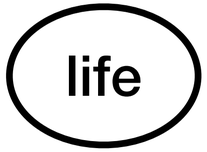
in/out

- Build a tree
- Visit every node with a post-order traversal

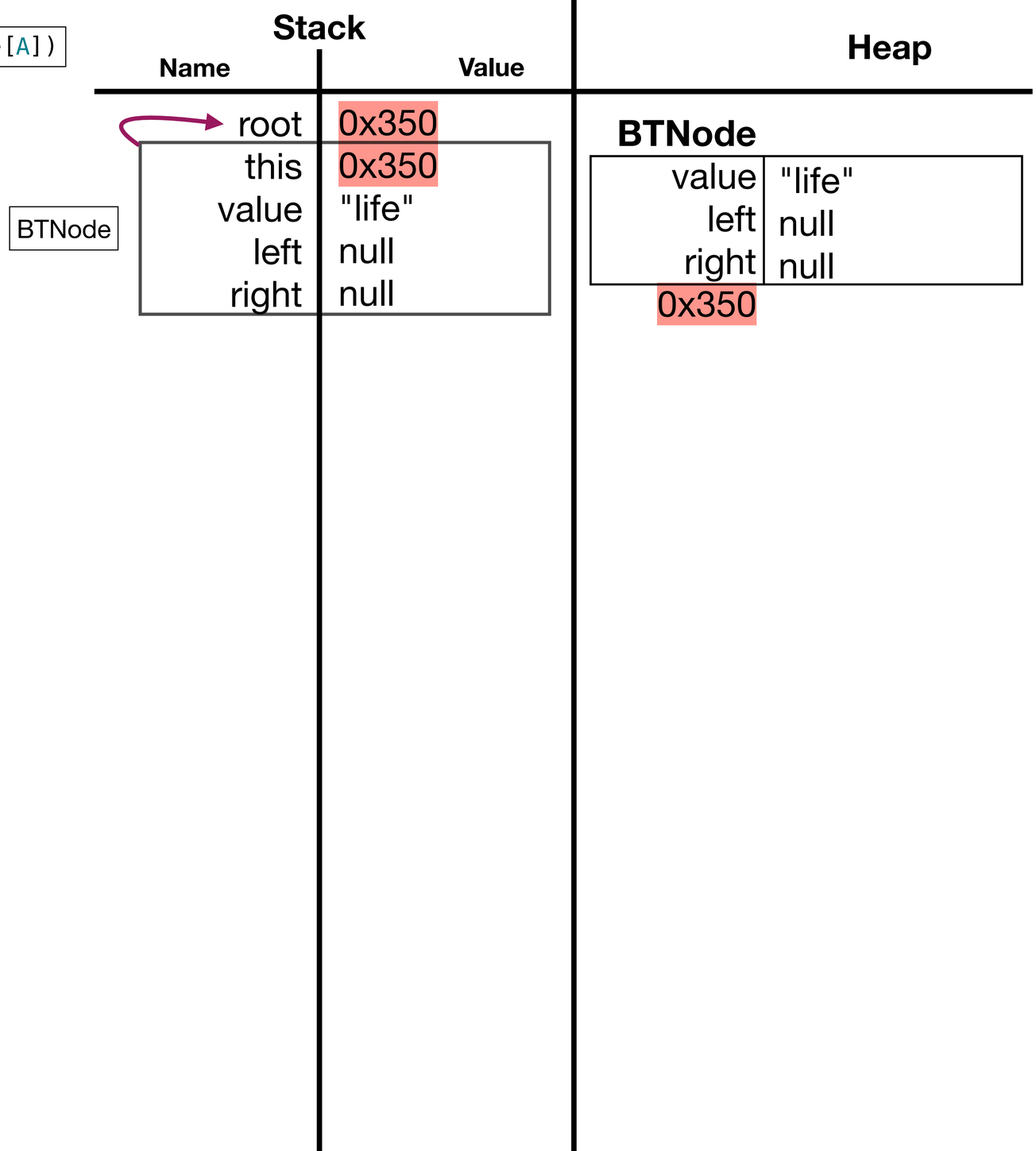


```scala
class BTNode[A](var value: A, var left: BTNode[A], var right: BTNode[A])
```

```scala
def traversal[A](node: BTNode[A]): Unit = {
  if (node != null) {
    traversal(node.left)
    traversal(node.right)
    print(node.value + " ")
  }
}

def main(args: Array[String]): Unit = {
  val root = new BTNode("life", null, null)
  root.left = new BTNode("Scala", null, null)
  root.right = new BTNode("for", null, null)
  root.right.left = new BTNode("coding", null, null)
  traversal(root)
}
```



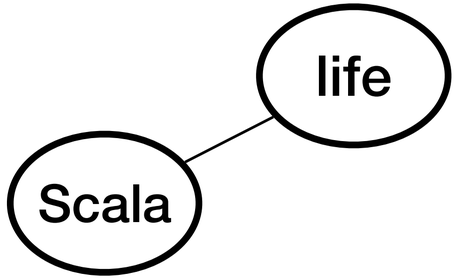
in/out



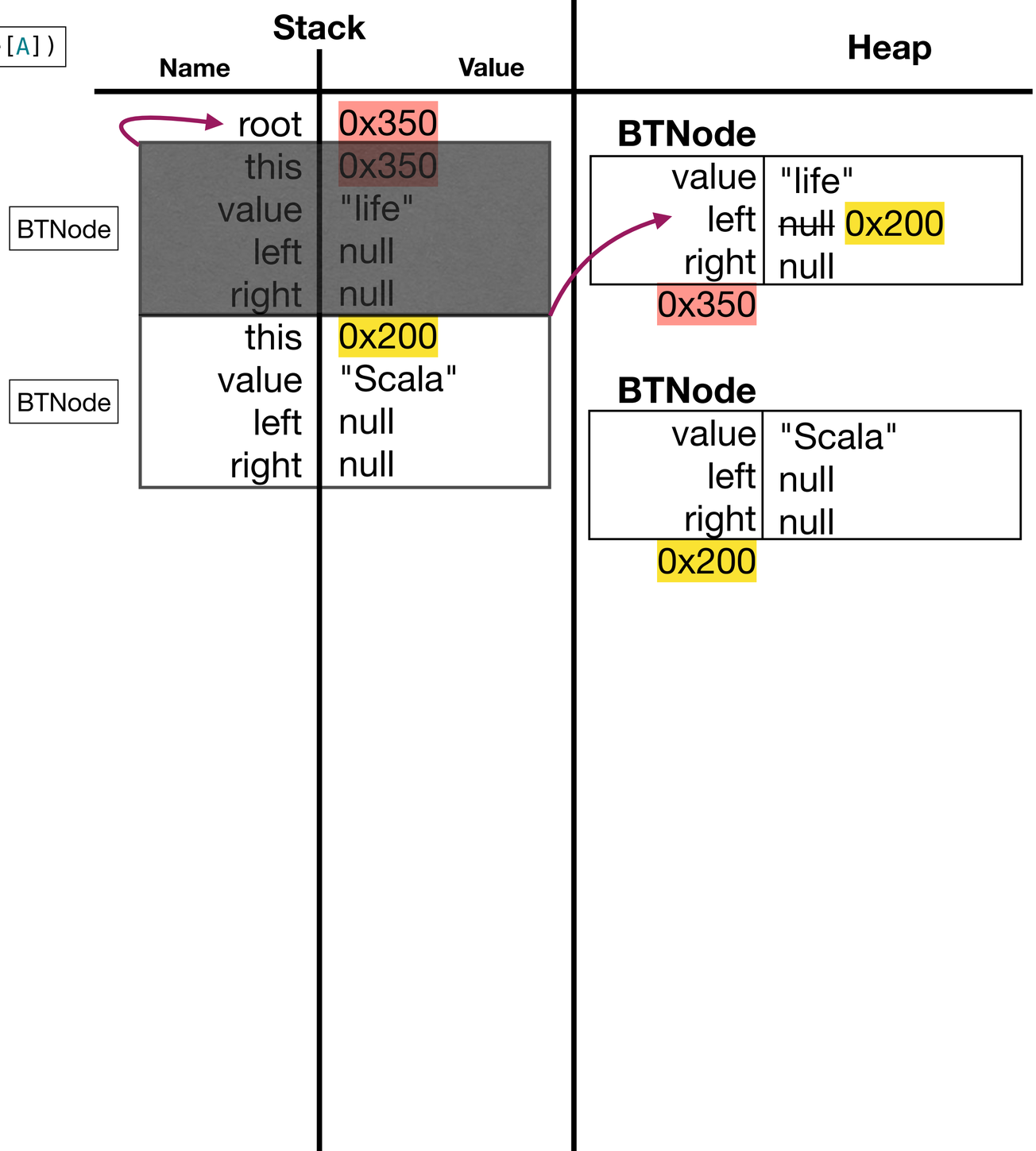
- Create the root node with the value "life"

```
class BTNode[A](var value: A, var left: BTNode[A], var right: BTNode[A])
```

```
def traversal[A](node: BTNode[A]): Unit = {
  if (node != null) {
    traversal(node.left)
    traversal(node.right)
    print(node.value + " ")
  }
}

def main(args: Array[String]): Unit = {
  val root = new BTNode("life", null, null)
  root.left = new BTNode("Scala", null, null)
  root.right = new BTNode("for", null, null)
  root.right.left = new BTNode("coding", null, null)
  traversal(root)
}
```



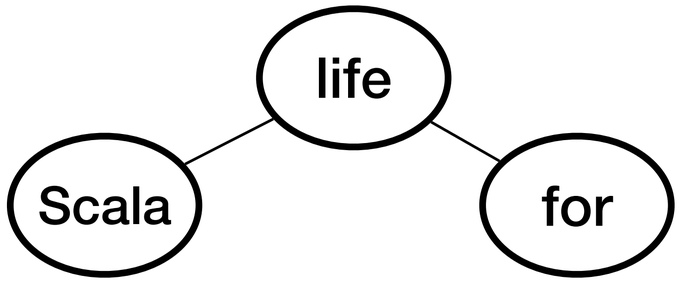
in/out



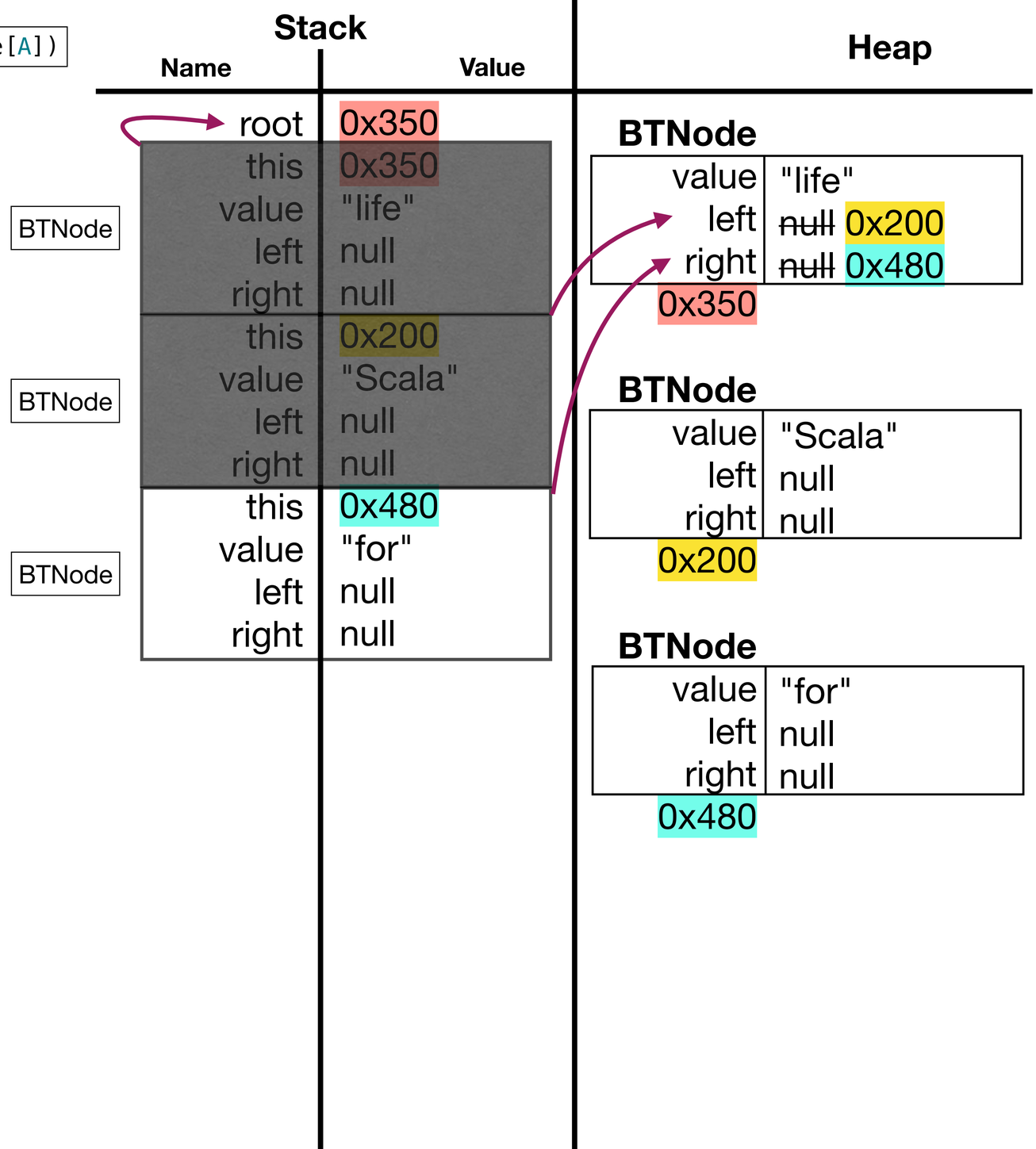
- Set the left child of the root to a node with "Scala"

```
class BTNode[A](var value: A, var left: BTNode[A], var right: BTNode[A])
```

```
def traversal[A](node: BTNode[A]): Unit = {
  if (node != null) {
    traversal(node.left)
    traversal(node.right)
    print(node.value + " ")
  }
}

def main(args: Array[String]): Unit = {
  val root = new BTNode("life", null, null)
  root.left = new BTNode("Scala", null, null)
  root.right = new BTNode("for", null, null)
  root.right.left = new BTNode("coding", null, null)
  traversal(root)
}
```



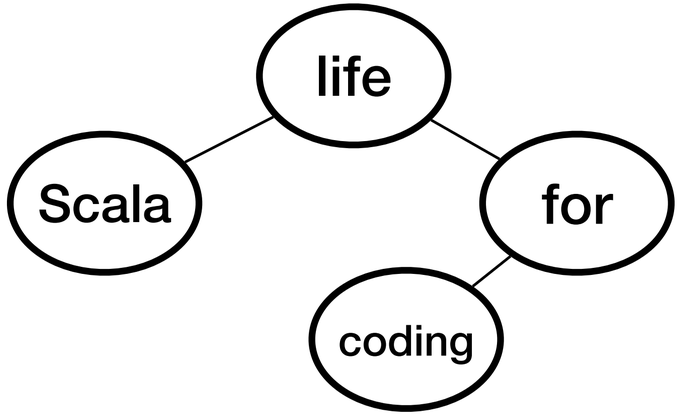
in/out



- Set the right child of the root to a node with "for"

```
class BTNode[A](var value: A, var left: BTNode[A], var right: BTNode[A])
```

```
def traversal[A](node: BTNode[A]): Unit = {
  if (node != null) {
    traversal(node.left)
    traversal(node.right)
    print(node.value + " ")
  }
}

def main(args: Array[String]): Unit = {
  val root = new BTNode("life", null, null)
  root.left = new BTNode("Scala", null, null)
  root.right = new BTNode("for", null, null)
  root.right.left = new BTNode("coding", null, null)
  traversal(root)
}
```



in/out

- Set the left child of the right child of the root to a node with "coding"

Stack

Name	Value
root	0x350
BTNode this	0x350
value	"life"
left	null
right	null
BTNode this	0x200
value	"Scala"
left	null
right	null
BTNode this	0x480
value	"for"
left	null
right	null
BTNode this	0x936
value	"coding"
left	null
right	null

Heap

BTNode 0x350

value	"life"
left	~~null~~ 0x200
right	~~null~~ 0x480

BTNode 0x200

value	"Scala"
left	null
right	null

BTNode 0x480

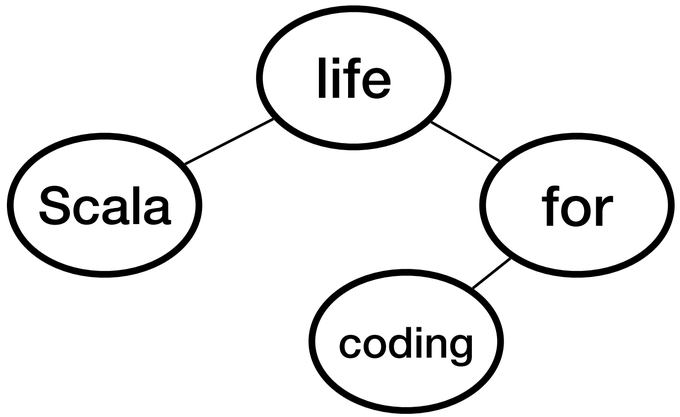
value	"for"
left	~~null~~ 0x936
right	null

BTNode 0x936

value	"coding"
left	null
right	null

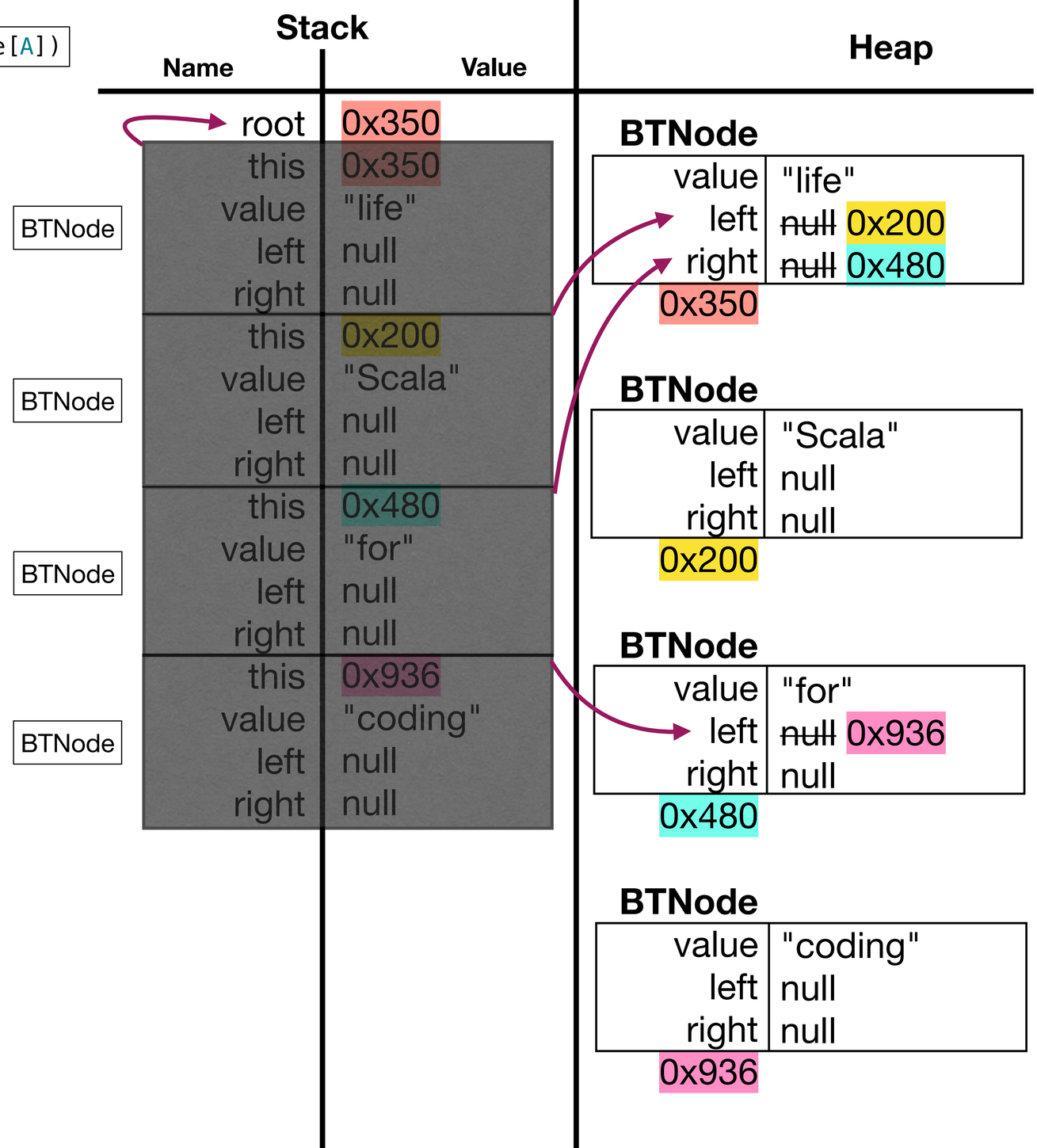
```
class BTNode[A](var value: A, var left: BTNode[A], var right: BTNode[A])
```

```
def traversal[A](node: BTNode[A]): Unit = {
  if (node != null) {
    traversal(node.left)
    traversal(node.right)
    print(node.value + " ")
  }
}

def main(args: Array[String]): Unit = {
  val root = new BTNode("life", null, null)
  root.left = new BTNode("Scala", null, null)
  root.right = new BTNode("for", null, null)
  root.right.left = new BTNode("coding", null, null)
  traversal(root)
}
```



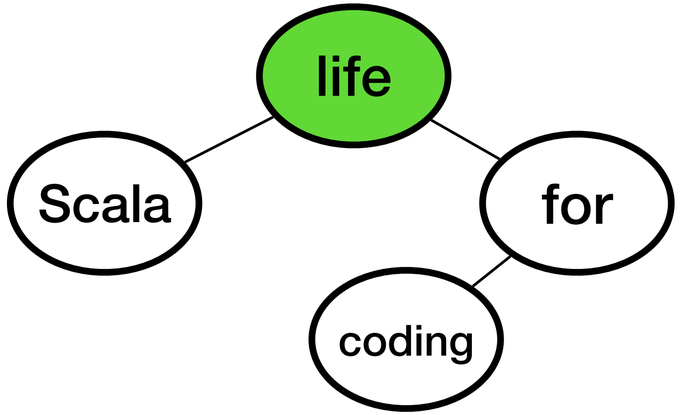
in/out

- Time to start the post-order traversal

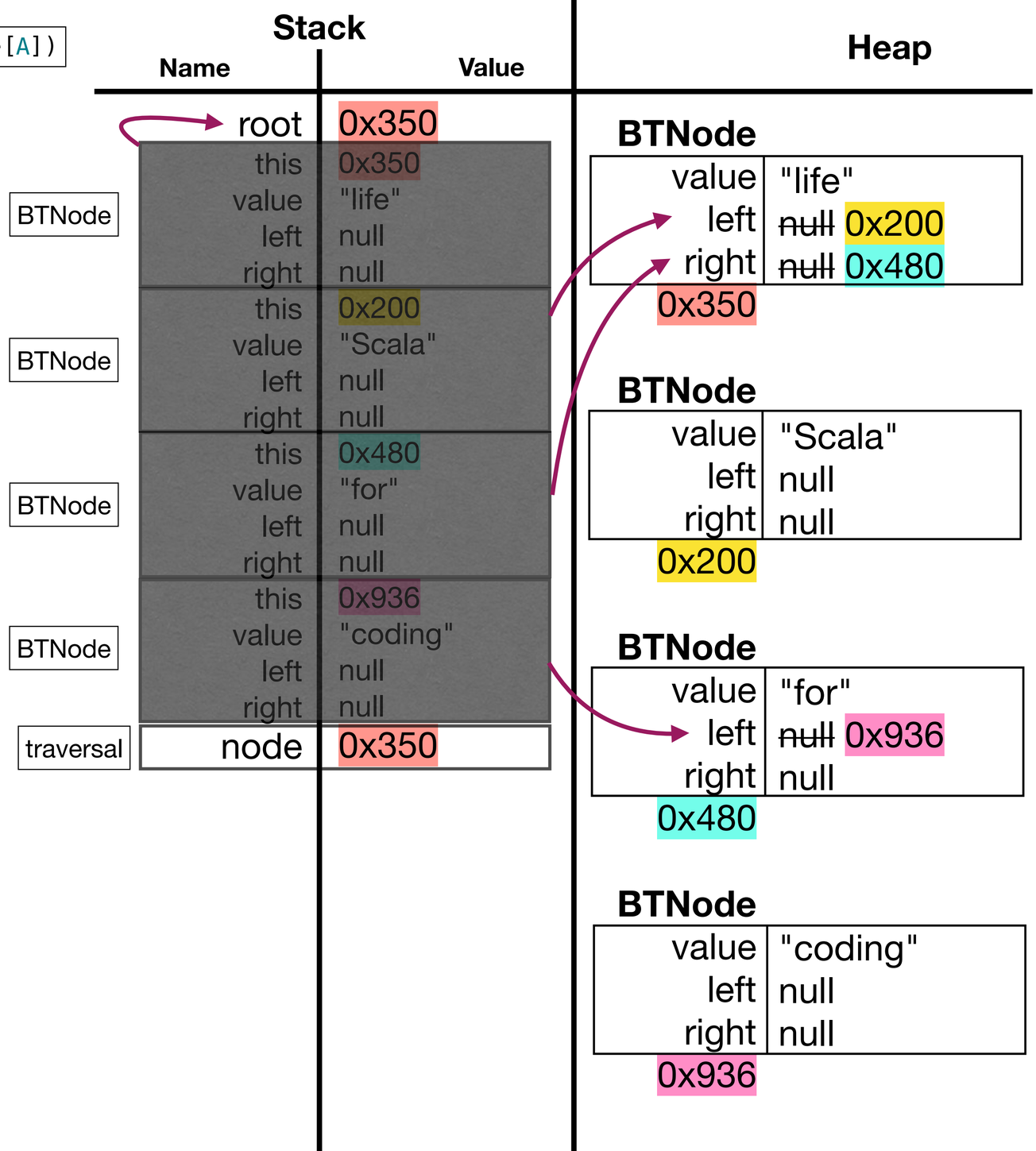


```
class BTNode[A](var value: A, var left: BTNode[A], var right: BTNode[A])
```

```
def traversal[A](node: BTNode[A]): Unit = {
  if (node != null) {
    traversal(node.left)
    traversal(node.right)
    print(node.value + " ")
  }
}

def main(args: Array[String]): Unit = {
  val root = new BTNode("life", null, null)
  root.left = new BTNode("Scala", null, null)
  root.right = new BTNode("for", null, null)
  root.right.left = new BTNode("coding", null, null)
  traversal(root)
}
```



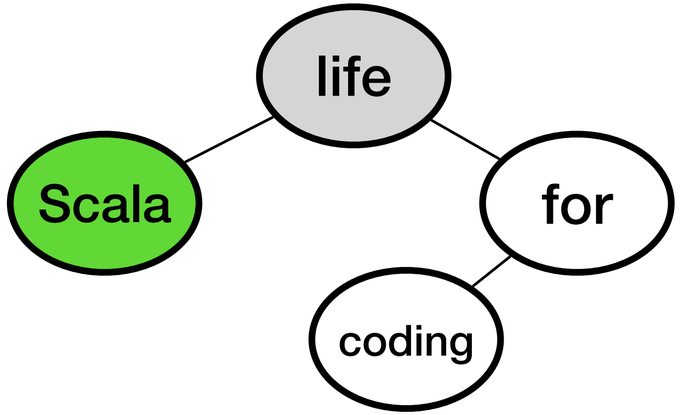
in/out



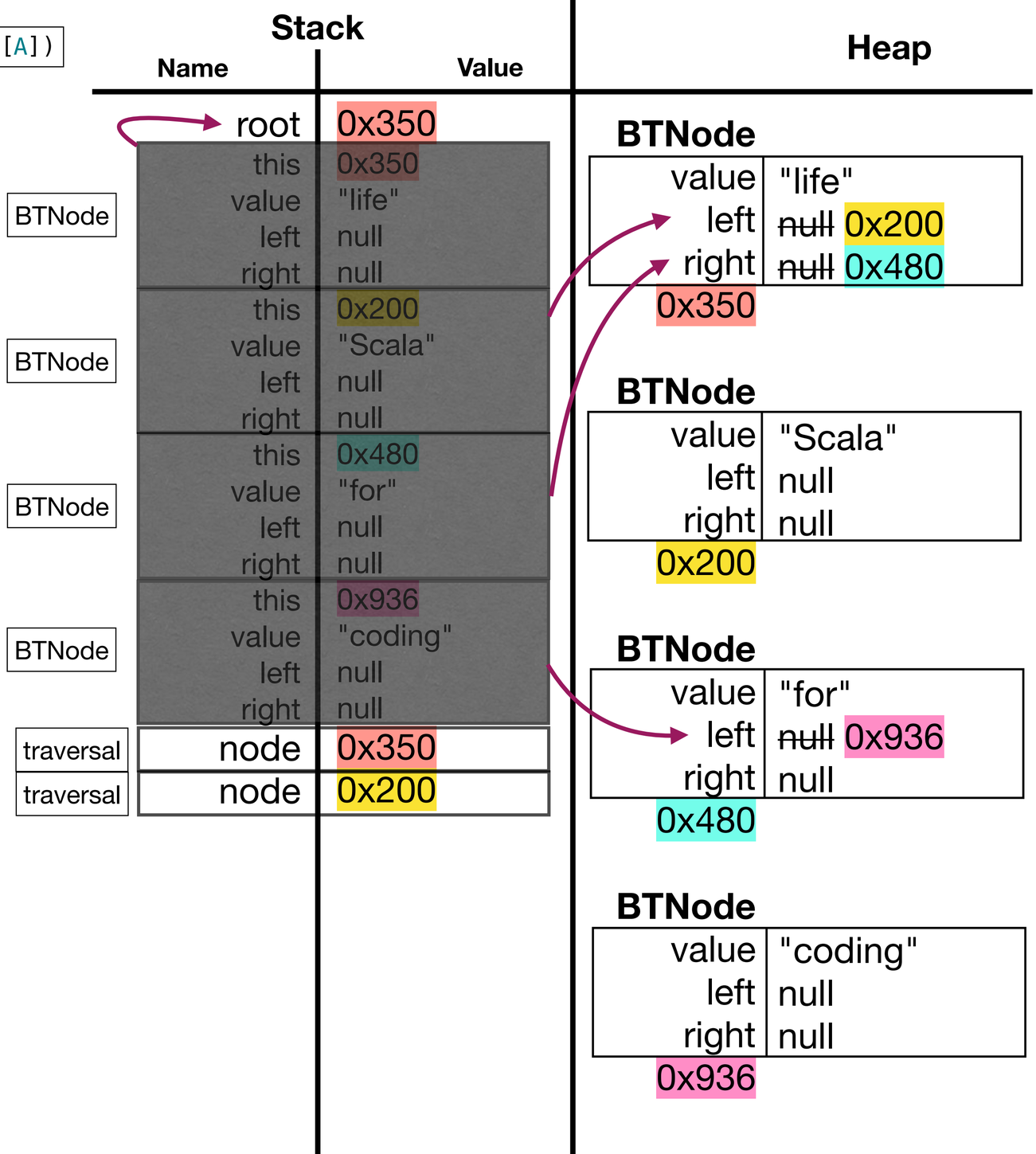
- Initial call of traversal is on the root node (0x350)

```
class BTNode[A](var value: A, var left: BTNode[A], var right: BTNode[A])
```

```
def traversal[A](node: BTNode[A]): Unit = {
  if (node != null) {
    traversal(node.left)
    traversal(node.right)
    print(node.value + " ")
  }
}

def main(args: Array[String]): Unit = {
  val root = new BTNode("life", null, null)
  root.left = new BTNode("Scala", null, null)
  root.right = new BTNode("for", null, null)
  root.right.left = new BTNode("coding", null, null)
  traversal(root)
}
```



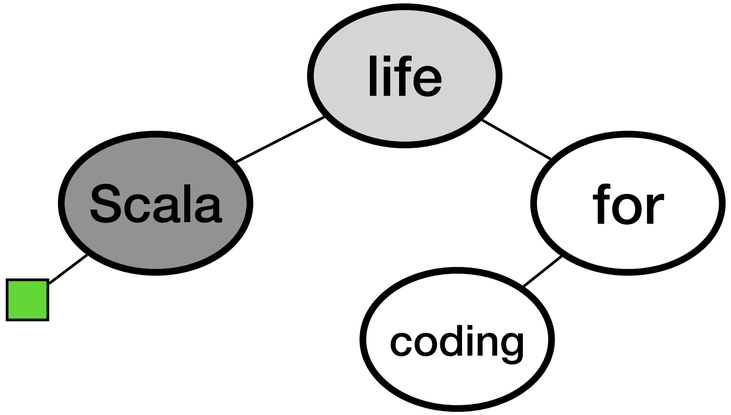
in/out



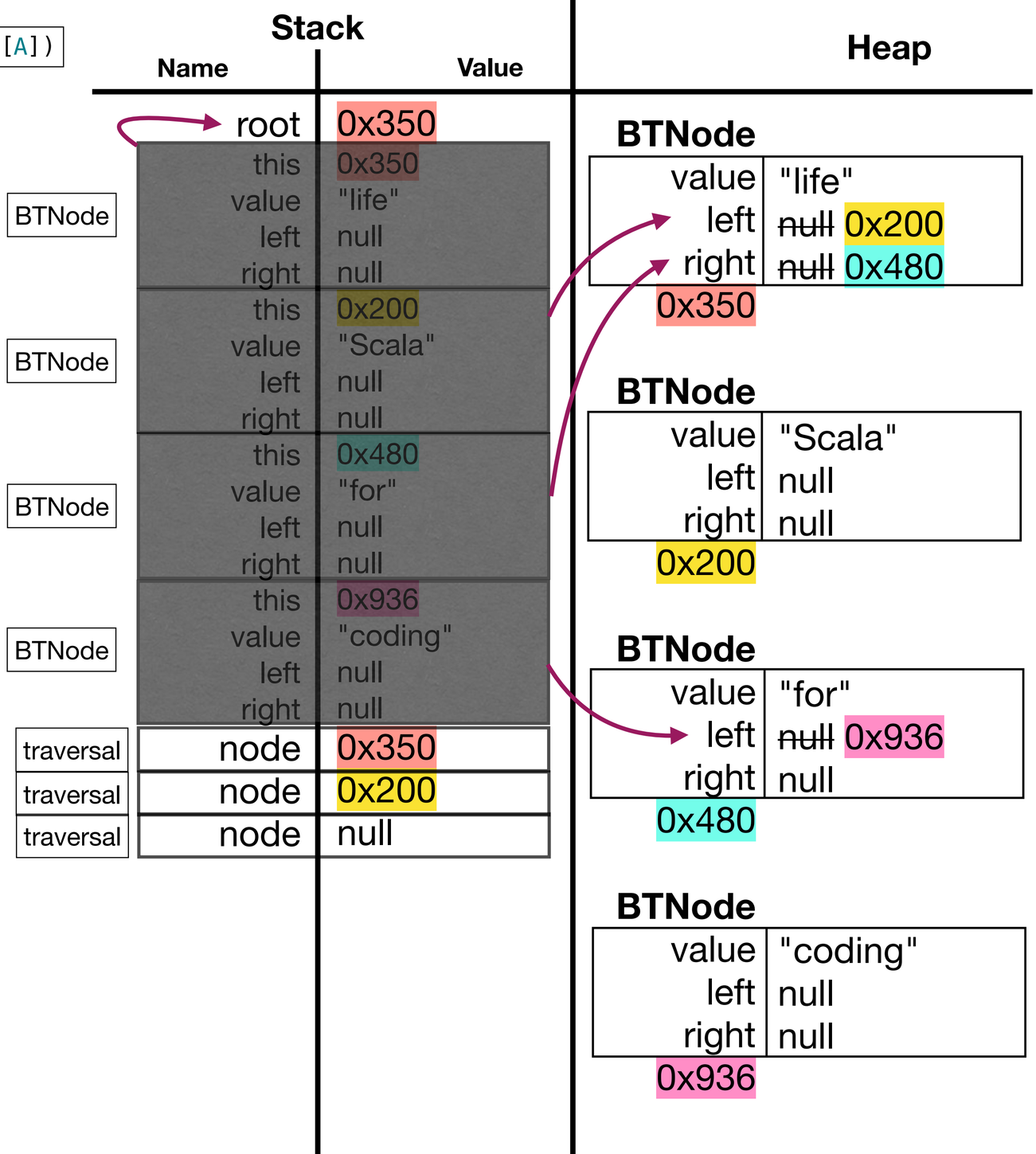
- Make a recursive call on the left child first

```
class BTNode[A](var value: A, var left: BTNode[A], var right: BTNode[A])
```

```
def traversal[A](node: BTNode[A]): Unit = {
  if (node != null) {
    traversal(node.left)
    traversal(node.right)
    print(node.value + " ")
  }
}

def main(args: Array[String]): Unit = {
  val root = new BTNode("life", null, null)
  root.left = new BTNode("Scala", null, null)
  root.right = new BTNode("for", null, null)
  root.right.left = new BTNode("coding", null, null)
  traversal(root)
}
```



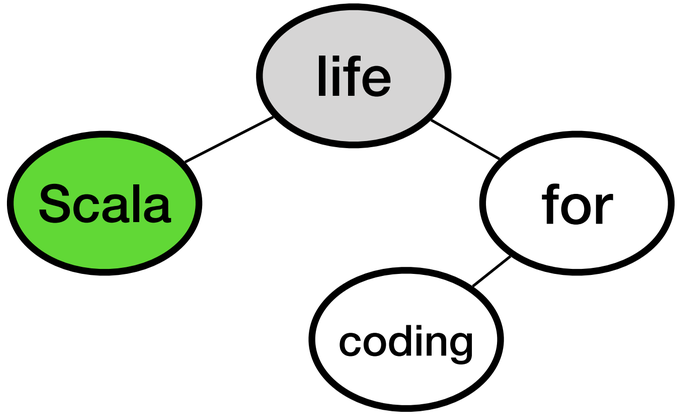
in/out



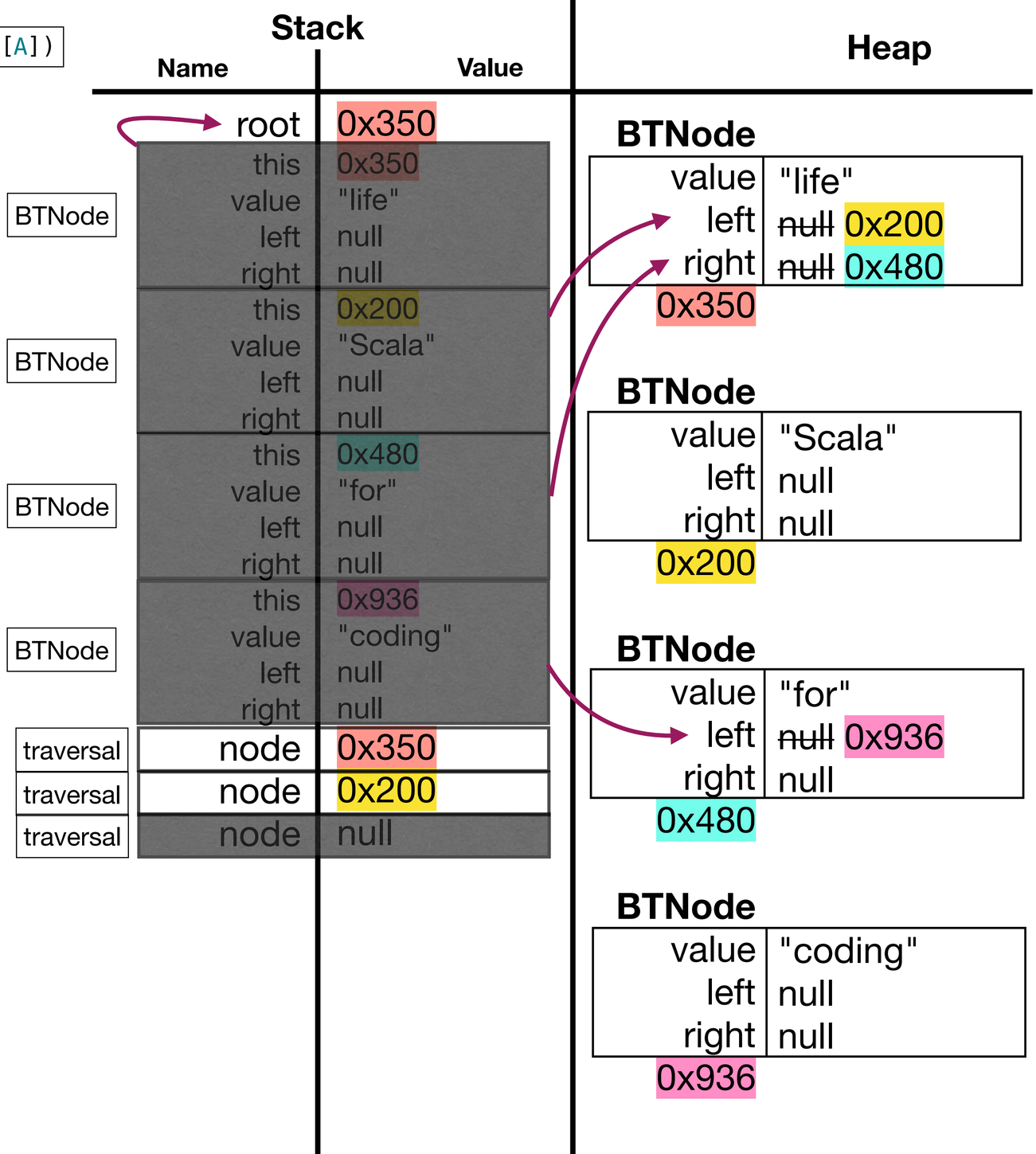
- Left node makes another recursive call on its left child

```
class BTNode[A](var value: A, var left: BTNode[A], var right: BTNode[A])
```

```
def traversal[A](node: BTNode[A]): Unit = {
  if (node != null) {
    traversal(node.left)
    traversal(node.right)
    print(node.value + " ")
  }
}

def main(args: Array[String]): Unit = {
  val root = new BTNode("life", null, null)
  root.left = new BTNode("Scala", null, null)
  root.right = new BTNode("for", null, null)
  root.right.left = new BTNode("coding", null, null)
  traversal(root)
}
```



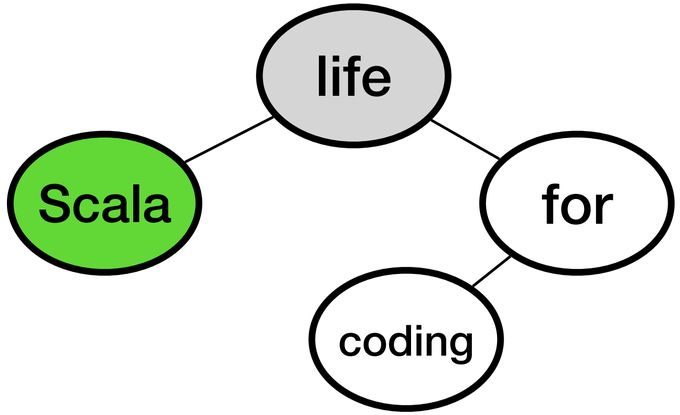
in/out



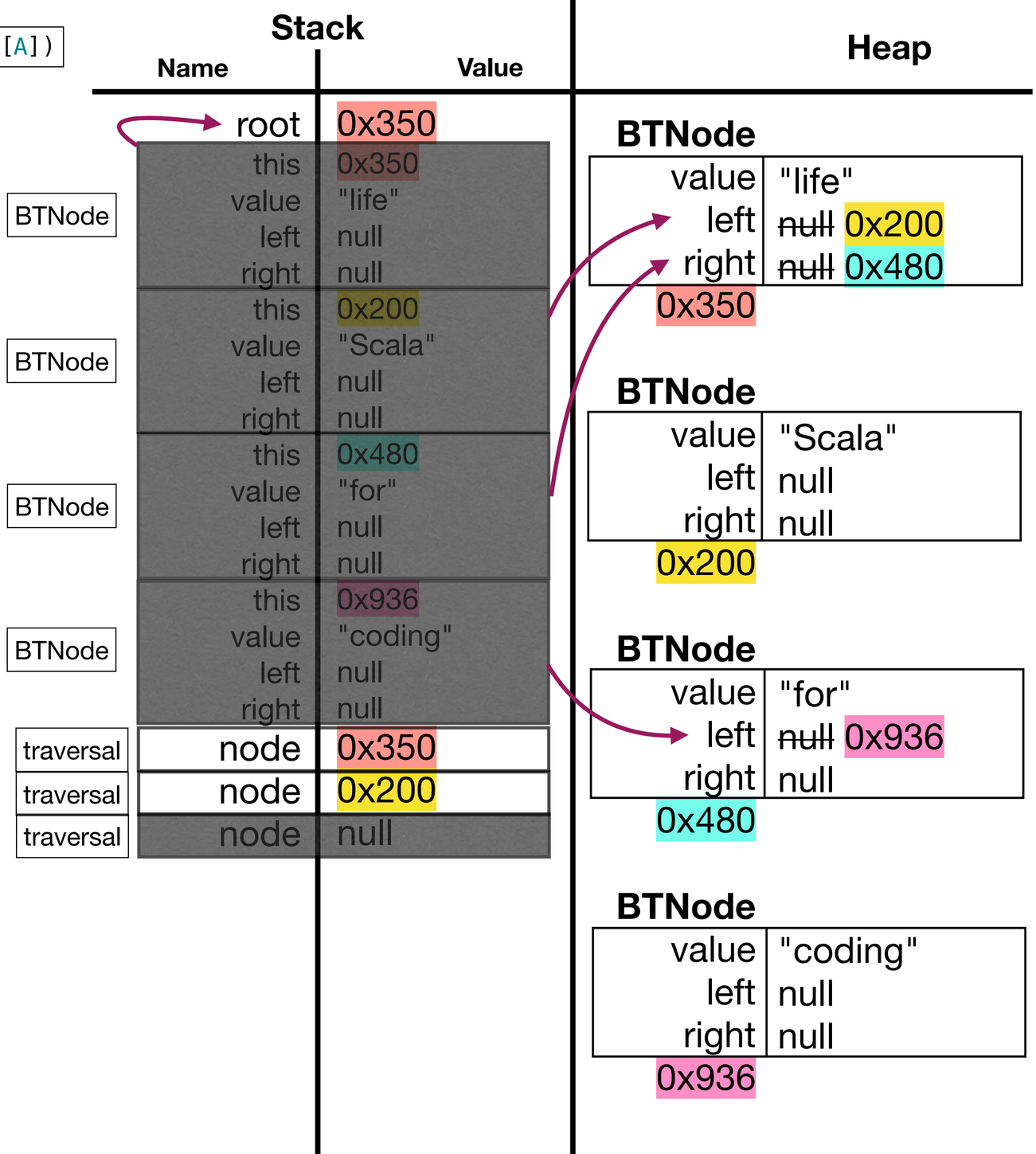
- A null node is our base case
- Do nothing and return to the previous frame on the stack

```
class BTNode[A](var value: A, var left: BTNode[A], var right: BTNode[A])
```

```
def traversal[A](node: BTNode[A]): Unit = {
  if (node != null) {
    traversal(node.left)
    traversal(node.right)
    print(node.value + " ")
  }
}

def main(args: Array[String]): Unit = {
  val root = new BTNode("life", null, null)
  root.left = new BTNode("Scala", null, null)
  root.right = new BTNode("for", null, null)
  root.right.left = new BTNode("coding", null, null)
  traversal(root)
}
```



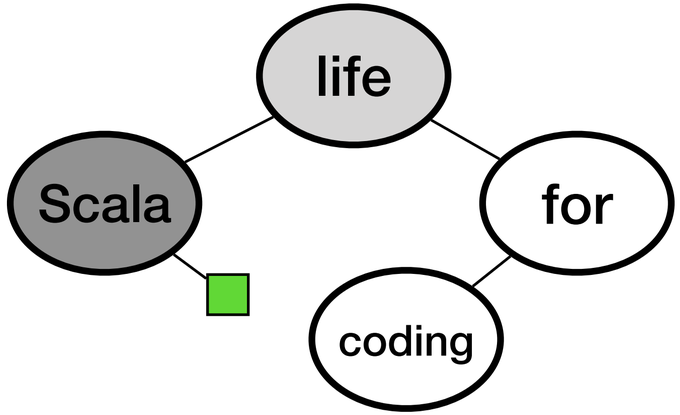
in/out



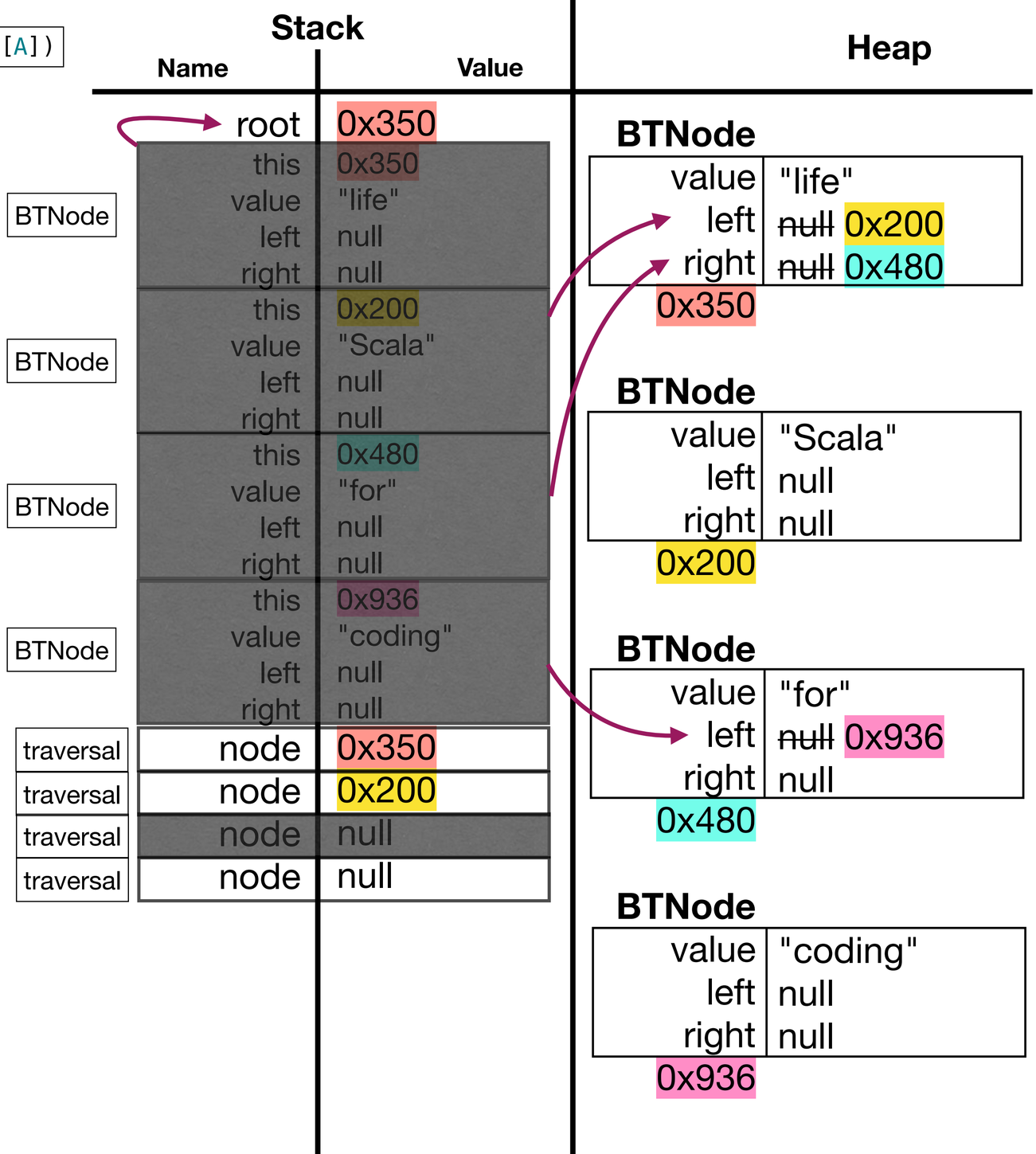
- This frame "remembers" that it made a left recursive call and needs to make the right recursive call

```
class BTNode[A](var value: A, var left: BTNode[A], var right: BTNode[A])
```

```
def traversal[A](node: BTNode[A]): Unit = {
  if (node != null) {
    traversal(node.left)
    traversal(node.right)
    print(node.value + " ")
  }
}

def main(args: Array[String]): Unit = {
  val root = new BTNode("life", null, null)
  root.left = new BTNode("Scala", null, null)
  root.right = new BTNode("for", null, null)
  root.right.left = new BTNode("coding", null, null)
  traversal(root)
}
```



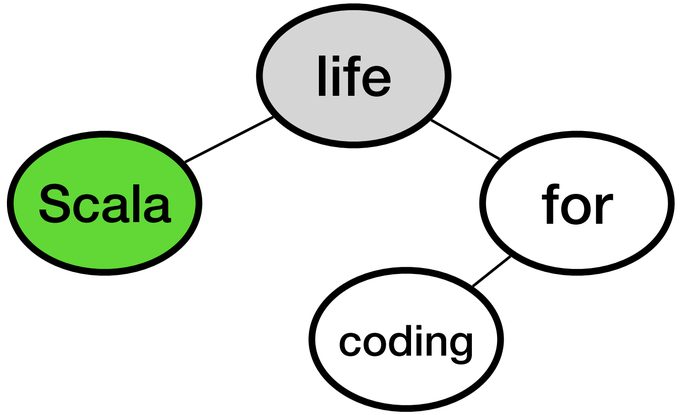
in/out



- Right recursive call is also for a null node

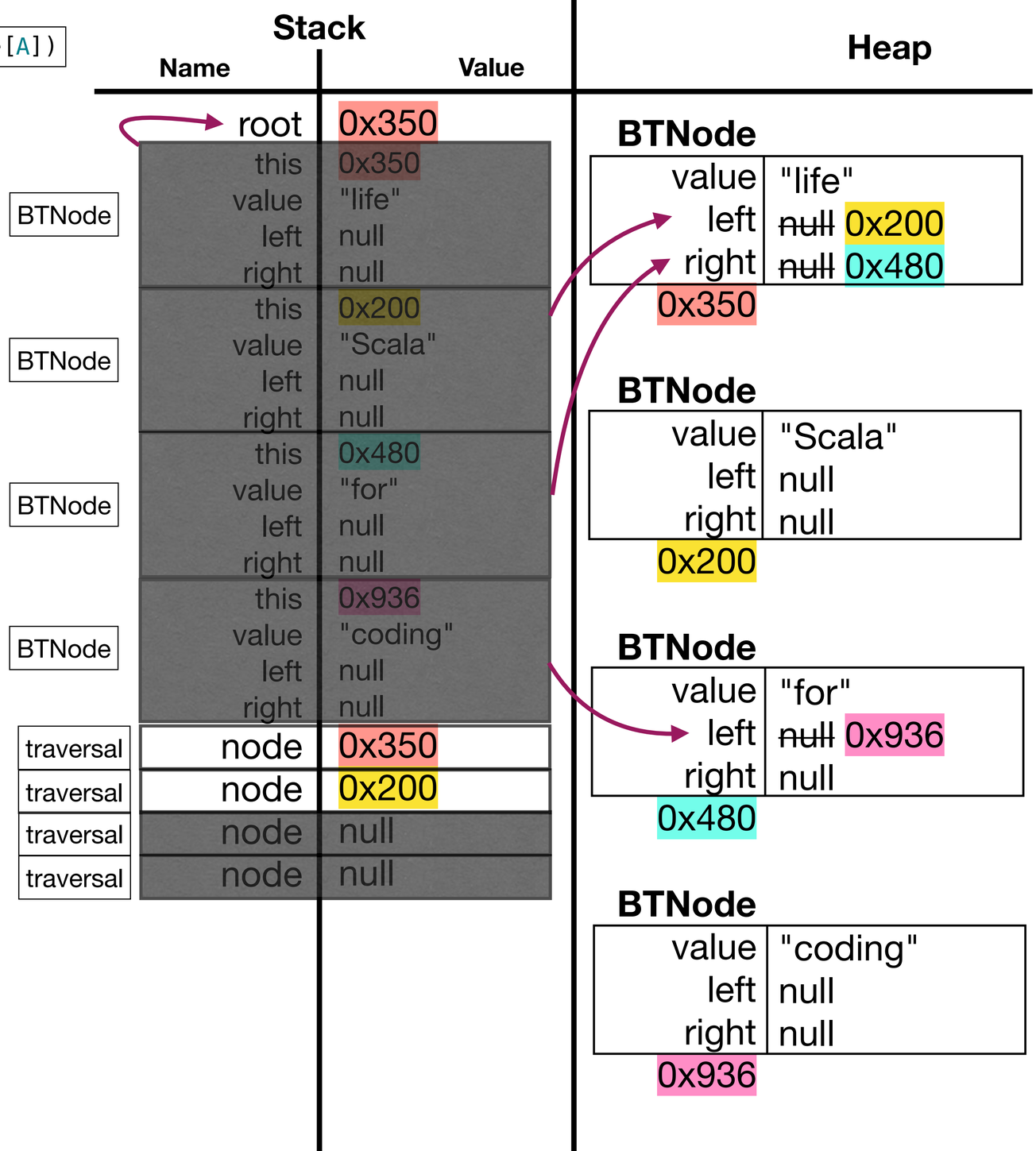
```
class BTNode[A](var value: A, var left: BTNode[A], var right: BTNode[A])
```

```
def traversal[A](node: BTNode[A]): Unit = {
  if (node != null) {
    traversal(node.left)
    traversal(node.right)
    print(node.value + " ")
  }
}

def main(args: Array[String]): Unit = {
  val root = new BTNode("life", null, null)
  root.left = new BTNode("Scala", null, null)
  root.right = new BTNode("for", null, null)
  root.right.left = new BTNode("coding", null, null)
  traversal(root)
}
```



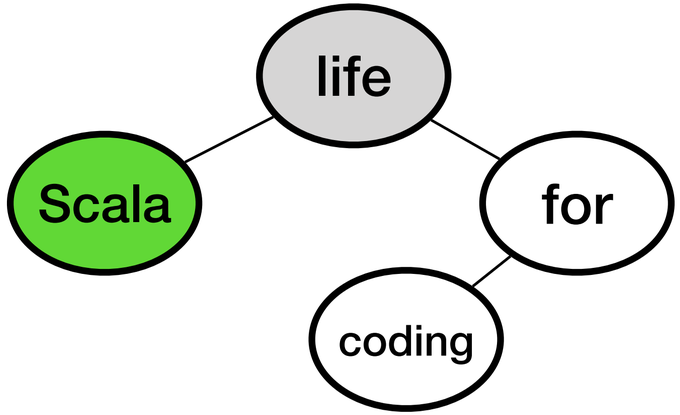
in/out

- Return to the previous frame again
- "Remembers" that it made the right recursive call



```
class BTNode[A](var value: A, var left: BTNode[A], var right: BTNode[A])
```

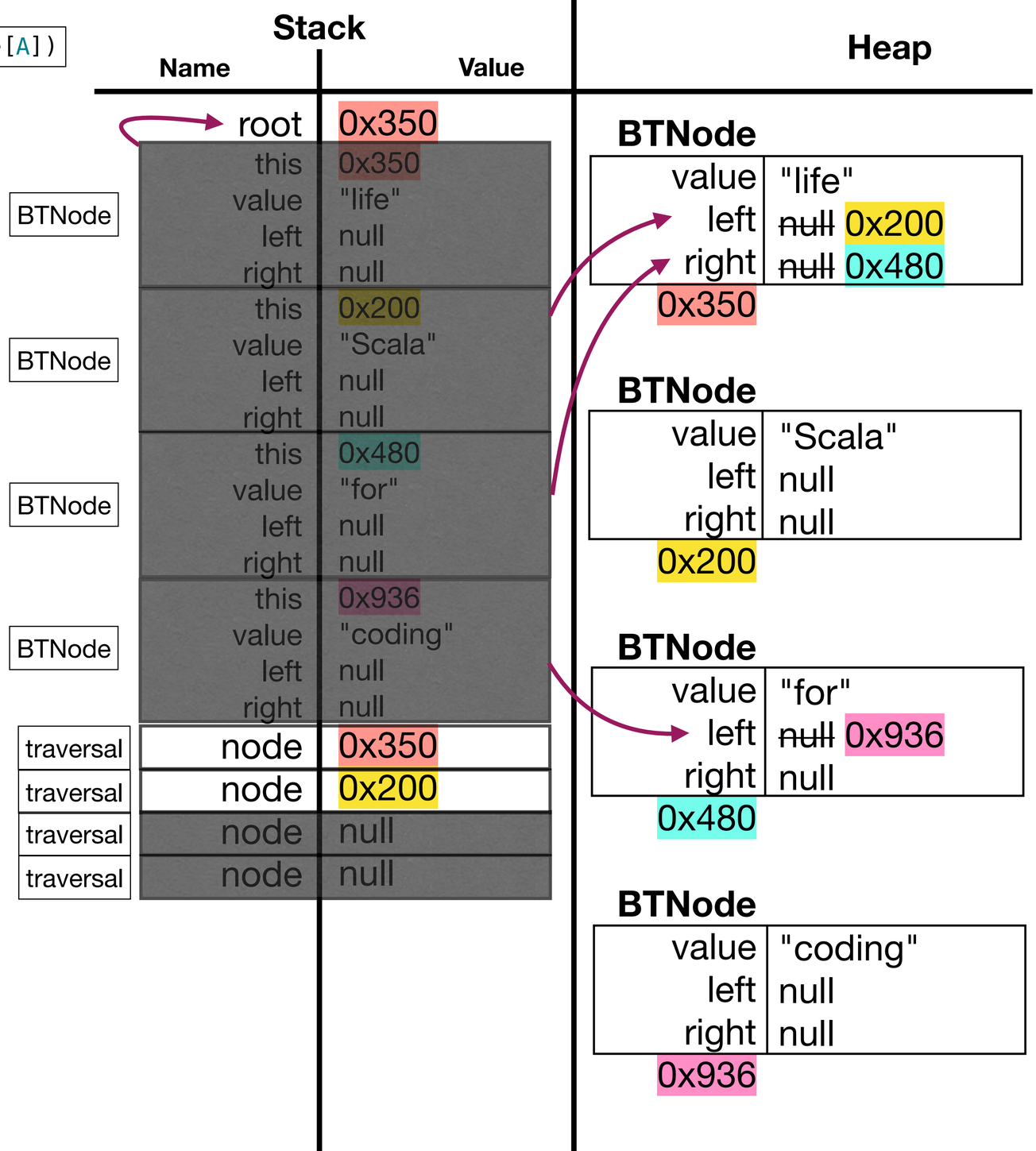
```
def traversal[A](node: BTNode[A]): Unit = {
  if (node != null) {
    traversal(node.left)
    traversal(node.right)
    print(node.value + " ")
  }
}

def main(args: Array[String]): Unit = {
  val root = new BTNode("life", null, null)
  root.left = new BTNode("Scala", null, null)
  root.right = new BTNode("for", null, null)
  root.right.left = new BTNode("coding", null, null)
  traversal(root)
}
```



in/out

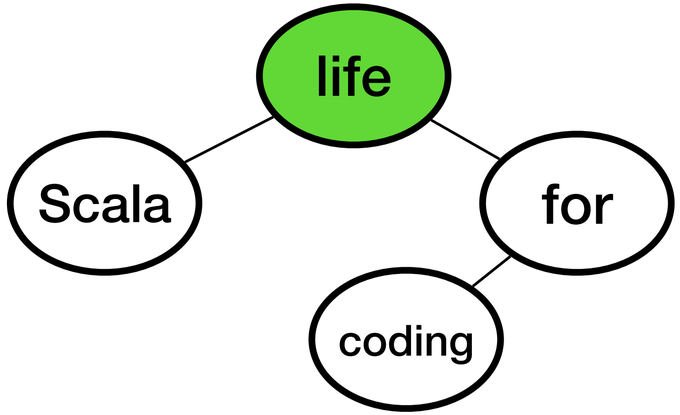
Scala

- Print Scala to the screen
- This stack frame is done and returns



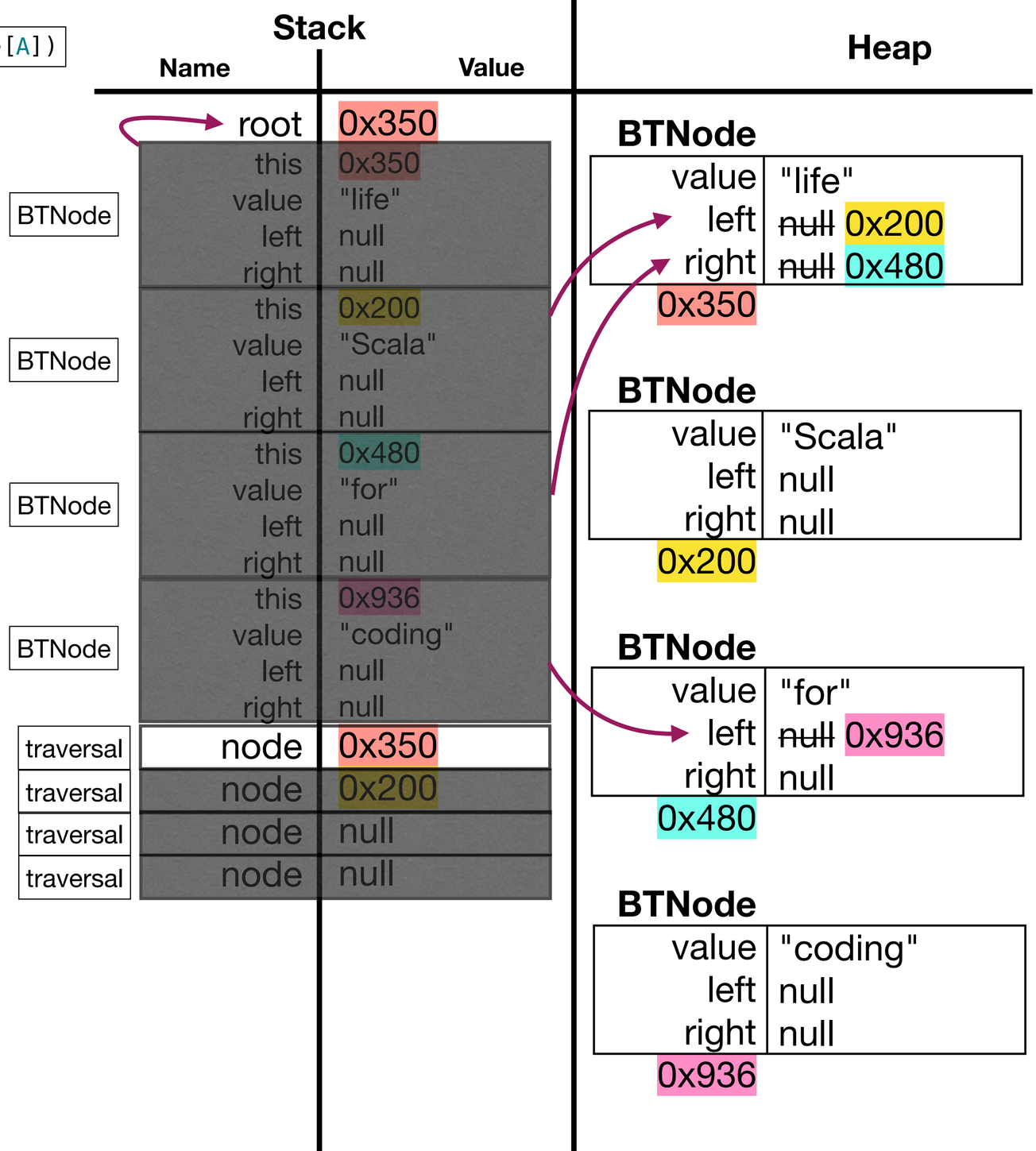
```
class BTNode[A](var value: A, var left: BTNode[A], var right: BTNode[A])
```

```
def traversal[A](node: BTNode[A]): Unit = {
  if (node != null) {
    traversal(node.left)
    traversal(node.right)
    print(node.value + " ")
  }
}

def main(args: Array[String]): Unit = {
  val root = new BTNode("life", null, null)
  root.left = new BTNode("Scala", null, null)
  root.right = new BTNode("for", null, null)
  root.right.left = new BTNode("coding", null, null)
  traversal(root)
}
```



in/out

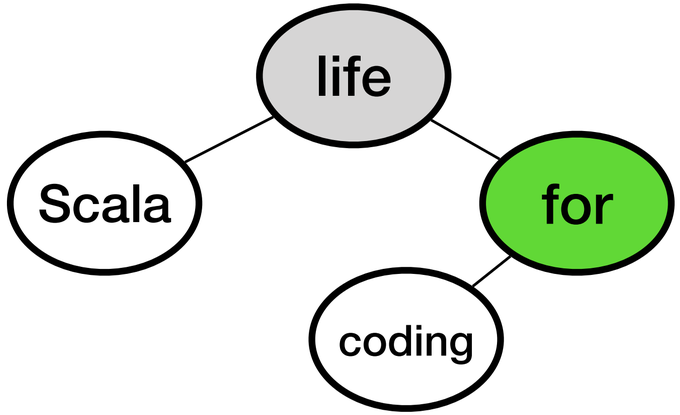
Scala



- First recursive call finally regains control (is at the top of the stack)

```
class BTNode[A](var value: A, var left: BTNode[A], var right: BTNode[A])
```

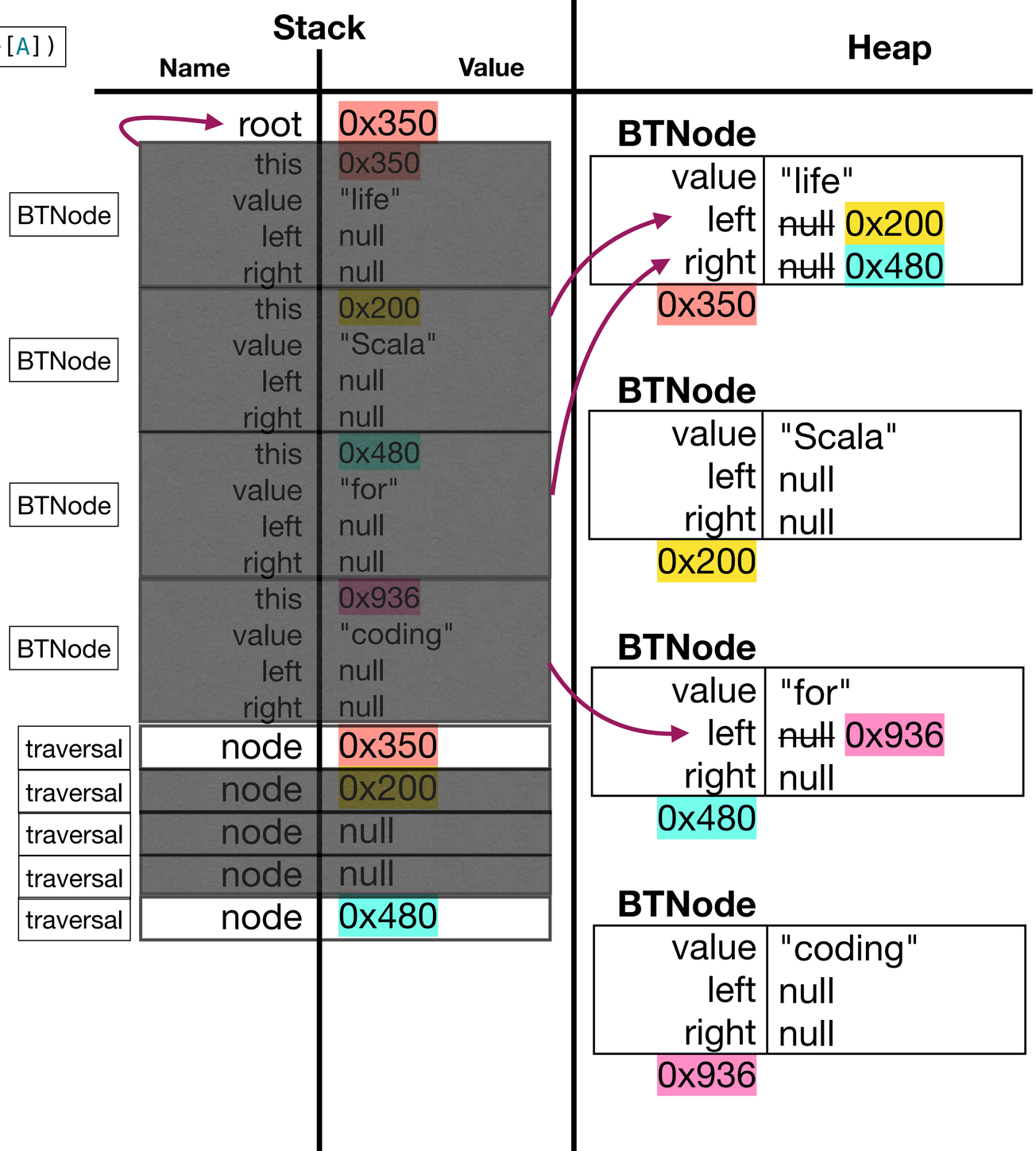
```
def traversal[A](node: BTNode[A]): Unit = {
  if (node != null) {
    traversal(node.left)
    traversal(node.right)
    print(node.value + " ")
  }
}

def main(args: Array[String]): Unit = {
  val root = new BTNode("life", null, null)
  root.left = new BTNode("Scala", null, null)
  root.right = new BTNode("for", null, null)
  root.right.left = new BTNode("coding", null, null)
  traversal(root)
}
```



in/out

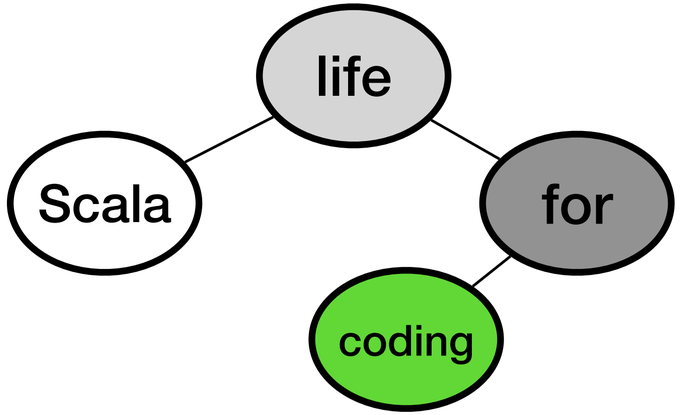
Scala

- Make a recursive call to the right of the root



```scala
class BTNode[A](var value: A, var left: BTNode[A], var right: BTNode[A])
```

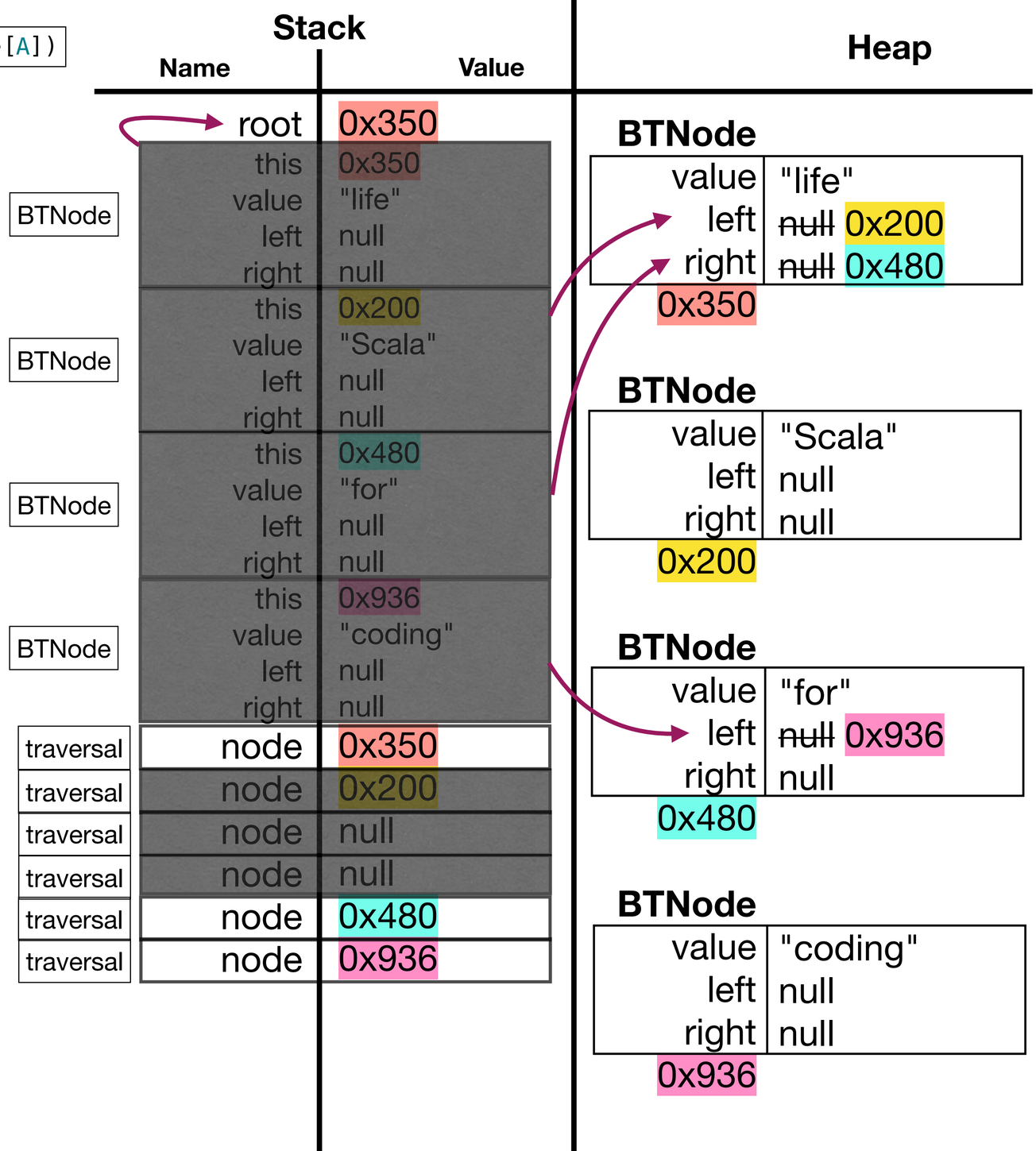
```scala
def traversal[A](node: BTNode[A]): Unit = {
  if (node != null) {
    traversal(node.left)
    traversal(node.right)
    print(node.value + " ")
  }
}

def main(args: Array[String]): Unit = {
  val root = new BTNode("life", null, null)
  root.left = new BTNode("Scala", null, null)
  root.right = new BTNode("for", null, null)
  root.right.left = new BTNode("coding", null, null)
  traversal(root)
}
```



in/out

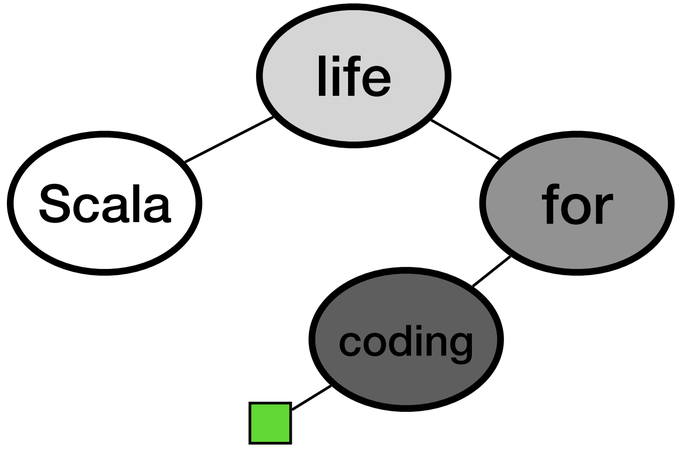
Scala

- Recursive call to the left



```scala
class BTNode[A](var value: A, var left: BTNode[A], var right: BTNode[A])
```

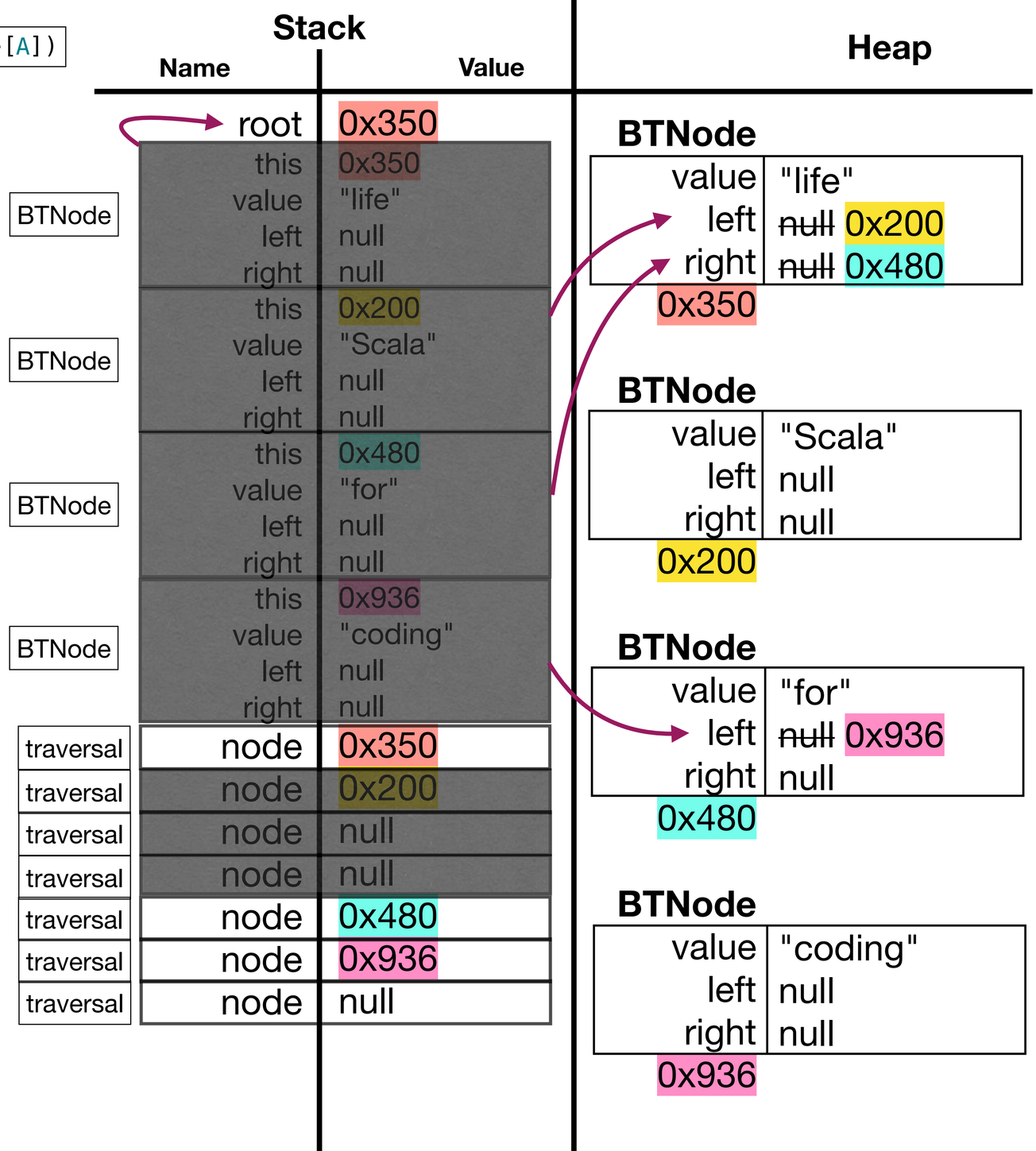
```scala
def traversal[A](node: BTNode[A]): Unit = {
  if (node != null) {
    traversal(node.left)
    traversal(node.right)
    print(node.value + " ")
  }
}

def main(args: Array[String]): Unit = {
  val root = new BTNode("life", null, null)
  root.left = new BTNode("Scala", null, null)
  root.right = new BTNode("for", null, null)
  root.right.left = new BTNode("coding", null, null)
  traversal(root)
}
```



in/out

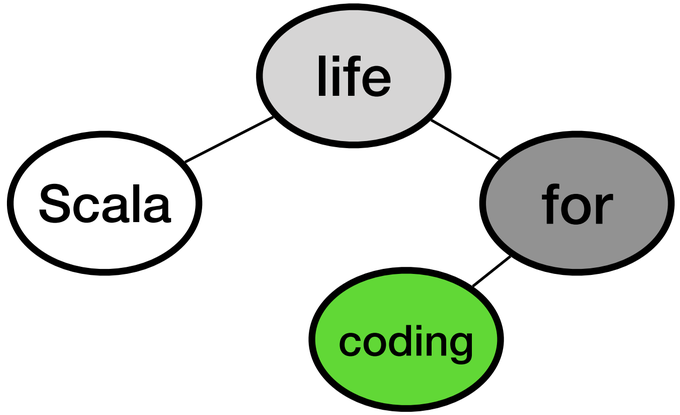
Scala

- Recursive call of null to the left



```
class BTNode[A](var value: A, var left: BTNode[A], var right: BTNode[A])
```

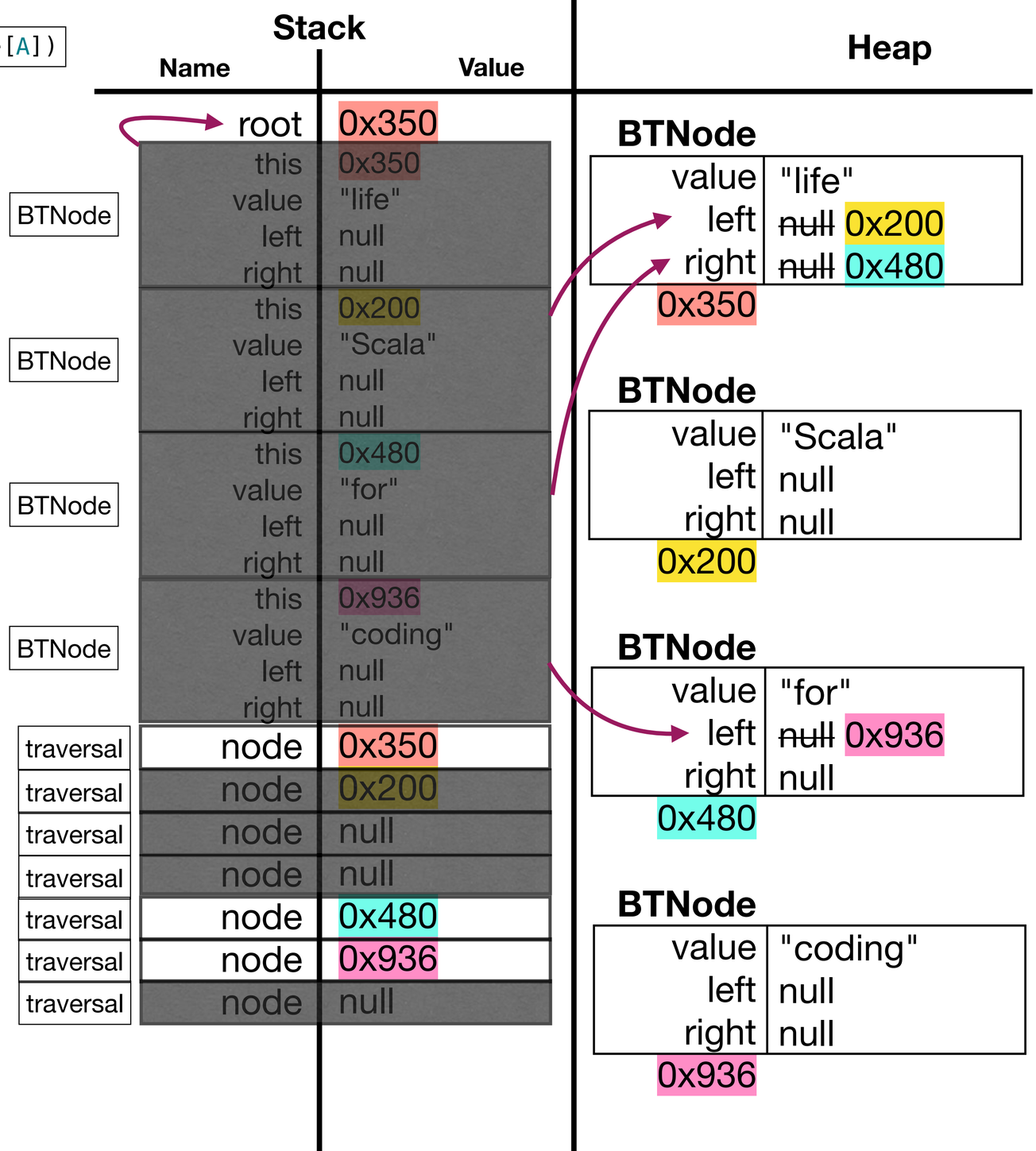
```
def traversal[A](node: BTNode[A]): Unit = {
  if (node != null) {
    traversal(node.left)
    traversal(node.right)
    print(node.value + " ")
  }
}

def main(args: Array[String]): Unit = {
  val root = new BTNode("life", null, null)
  root.left = new BTNode("Scala", null, null)
  root.right = new BTNode("for", null, null)
  root.right.left = new BTNode("coding", null, null)
  traversal(root)
}
```



in/out

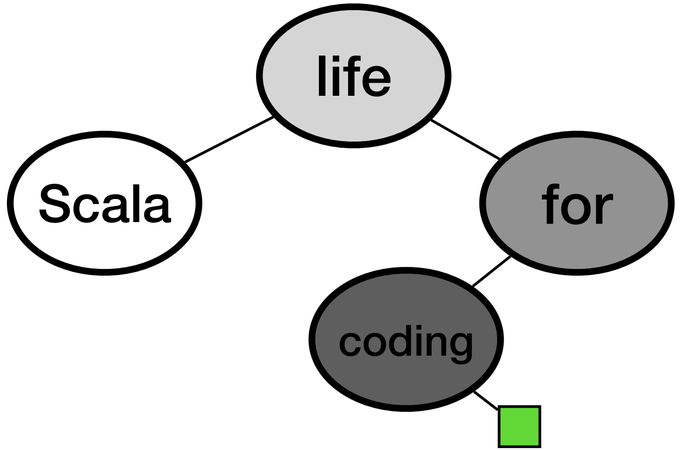
Scala

- Return to the previous frame and make the right recursive call



```
class BTNode[A](var value: A, var left: BTNode[A], var right: BTNode[A])
```

```
def traversal[A](node: BTNode[A]): Unit = {
  if (node != null) {
    traversal(node.left)
    traversal(node.right)
    print(node.value + " ")
  }
}

def main(args: Array[String]): Unit = {
  val root = new BTNode("life", null, null)
  root.left = new BTNode("Scala", null, null)
  root.right = new BTNode("for", null, null)
  root.right.left = new BTNode("coding", null, null)
  traversal(root)
}
```



in/out

Scala

- Make the right recursive call of null

Stack

	Name	Value
	root	0x350
BTNode	this	0x350
	value	"life"
	left	null
	right	null
BTNode	this	0x200
	value	"Scala"
	left	null
	right	null
BTNode	this	0x480
	value	"for"
	left	null
	right	null
BTNode	this	0x936
	value	"coding"
	left	null
	right	null
traversal	node	0x350
traversal	node	0x200
traversal	node	null
traversal	node	null
traversal	node	0x480
traversal	node	0x936
traversal	node	null
traversal	node	null

Heap

BTNode 0x350

value	"life"
left	~~null~~ 0x200
right	~~null~~ 0x480

BTNode 0x200

value	"Scala"
left	null
right	null

BTNode 0x480

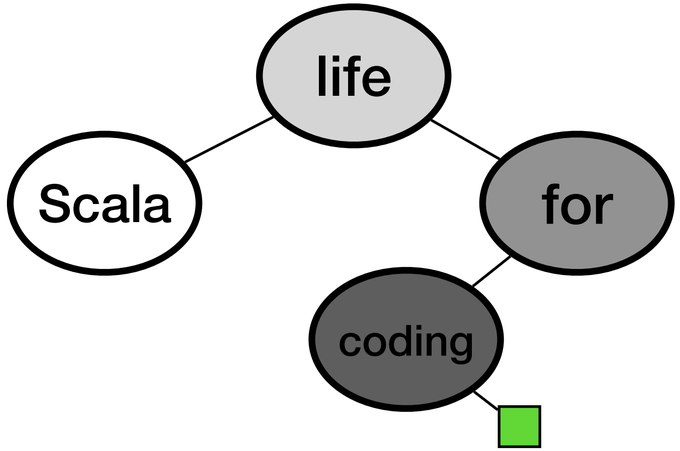
value	"for"
left	~~null~~ 0x936
right	null

BTNode 0x936

value	"coding"
left	null
right	null

```
class BTNode[A](var value: A, var left: BTNode[A], var right: BTNode[A])
```

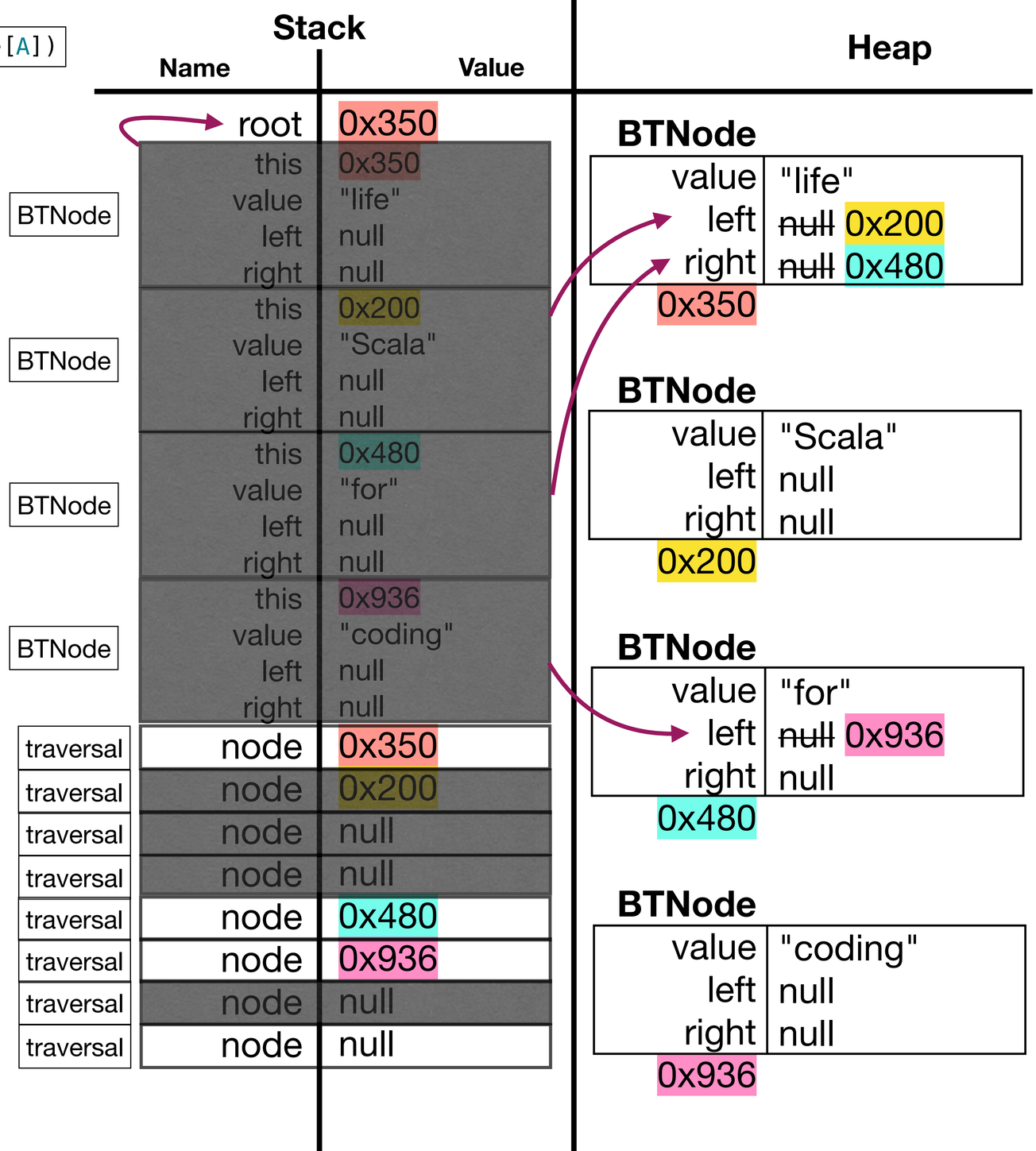
```
def traversal[A](node: BTNode[A]): Unit = {
  if (node != null) {
    traversal(node.left)
    traversal(node.right)
    print(node.value + " ")
  }
}

def main(args: Array[String]): Unit = {
  val root = new BTNode("life", null, null)
  root.left = new BTNode("Scala", null, null)
  root.right = new BTNode("for", null, null)
  root.right.left = new BTNode("coding", null, null)
  traversal(root)
}
```



in/out

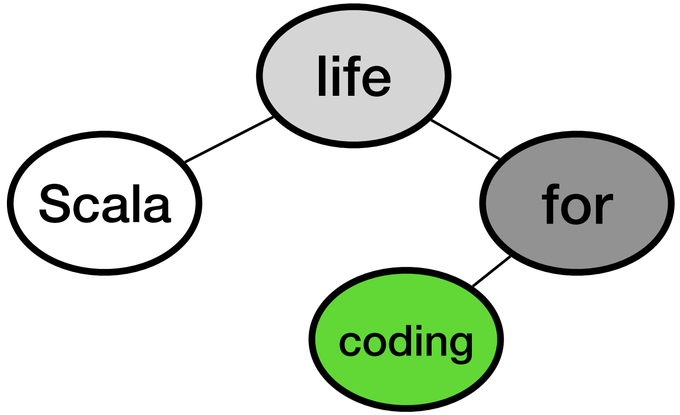
Scala

- Notice that all 4 stack frames remember what they need to do next when they regain control



```
class BTNode[A](var value: A, var left: BTNode[A], var right: BTNode[A])
```

```
def traversal[A](node: BTNode[A]): Unit = {
  if (node != null) {
    traversal(node.left)
    traversal(node.right)
    print(node.value + " ")
  }
}

def main(args: Array[String]): Unit = {
  val root = new BTNode("life", null, null)
  root.left = new BTNode("Scala", null, null)
  root.right = new BTNode("for", null, null)
  root.right.left = new BTNode("coding", null, null)
  traversal(root)
}
```



in/out

Scala

- This stack frame already made both recursive calls

Stack

	Name	Value
	root	0x350
BTNode	this	0x350
	value	"life"
	left	null
	right	null
BTNode	this	0x200
	value	"Scala"
	left	null
	right	null
BTNode	this	0x480
	value	"for"
	left	null
	right	null
BTNode	this	0x936
	value	"coding"
	left	null
	right	null
traversal	node	0x350
traversal	node	0x200
traversal	node	null
traversal	node	null
traversal	node	0x480
traversal	node	0x936
traversal	node	null
traversal	node	null

Heap

BTNode 0x350

value	"life"
left	~~null~~ 0x200
right	~~null~~ 0x480

BTNode 0x200

value	"Scala"
left	null
right	null

BTNode 0x480

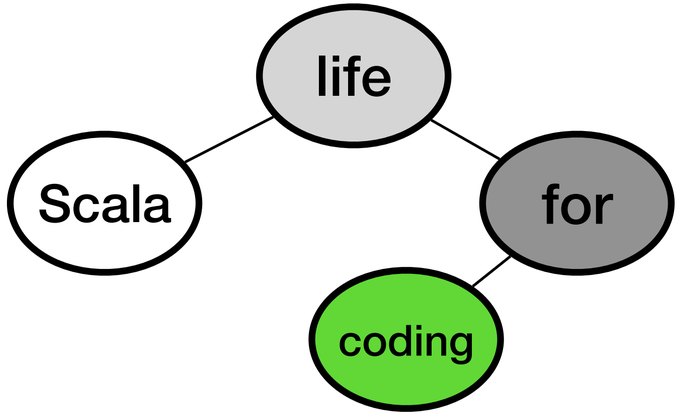
value	"for"
left	~~null~~ 0x936
right	null

BTNode 0x936

value	"coding"
left	null
right	null

```
class BTNode[A](var value: A, var left: BTNode[A], var right: BTNode[A])
```

```
def traversal[A](node: BTNode[A]): Unit = {
  if (node != null) {
    traversal(node.left)
    traversal(node.right)
    print(node.value + " ")
  }
}

def main(args: Array[String]): Unit = {
  val root = new BTNode("life", null, null)
  root.left = new BTNode("Scala", null, null)
  root.right = new BTNode("for", null, null)
  root.right.left = new BTNode("coding", null, null)
  traversal(root)
}
```



in/out

Scala coding

- Print "coding" to the screen

Stack

	Name	Value
	root	0x350
BTNode	this	0x350
	value	"life"
	left	null
	right	null
BTNode	this	0x200
	value	"Scala"
	left	null
	right	null
BTNode	this	0x480
	value	"for"
	left	null
	right	null
BTNode	this	0x936
	value	"coding"
	left	null
	right	null
traversal	node	0x350
traversal	node	0x200
traversal	node	null
traversal	node	null
traversal	node	0x480
traversal	node	0x936
traversal	node	null
traversal	node	null

Heap

BTNode 0x350

value	"life"
left	~~null~~ 0x200
right	~~null~~ 0x480

BTNode 0x200

value	"Scala"
left	null
right	null

BTNode 0x480

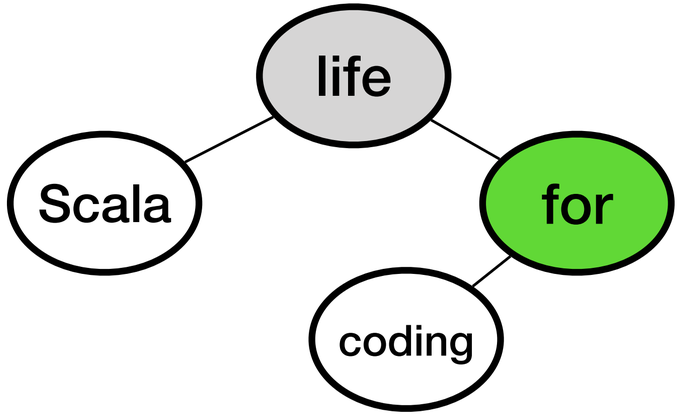
value	"for"
left	~~null~~ 0x936
right	null

BTNode 0x936

value	"coding"
left	null
right	null

```
class BTNode[A](var value: A, var left: BTNode[A], var right: BTNode[A])
```

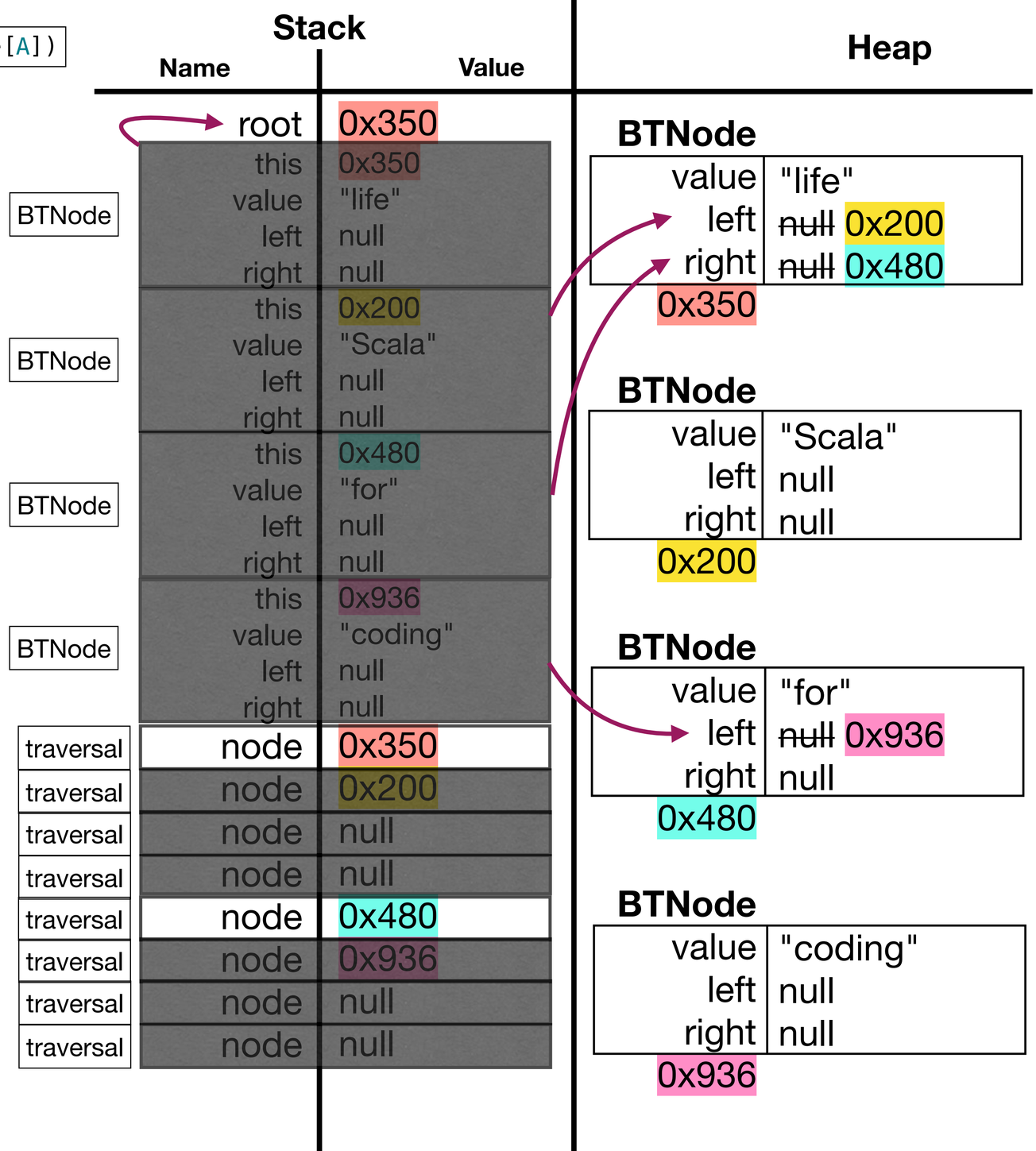
```
def traversal[A](node: BTNode[A]): Unit = {
  if (node != null) {
    traversal(node.left)
    traversal(node.right)
    print(node.value + " ")
  }
}

def main(args: Array[String]): Unit = {
  val root = new BTNode("life", null, null)
  root.left = new BTNode("Scala", null, null)
  root.right = new BTNode("for", null, null)
  root.right.left = new BTNode("coding", null, null)
  traversal(root)
}
```



in/out

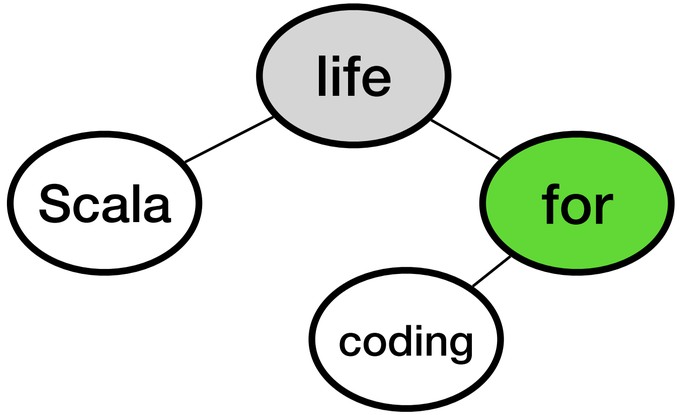
Scala coding

- Return back to the previous stack frame



```
class BTNode[A](var value: A, var left: BTNode[A], var right: BTNode[A])
```

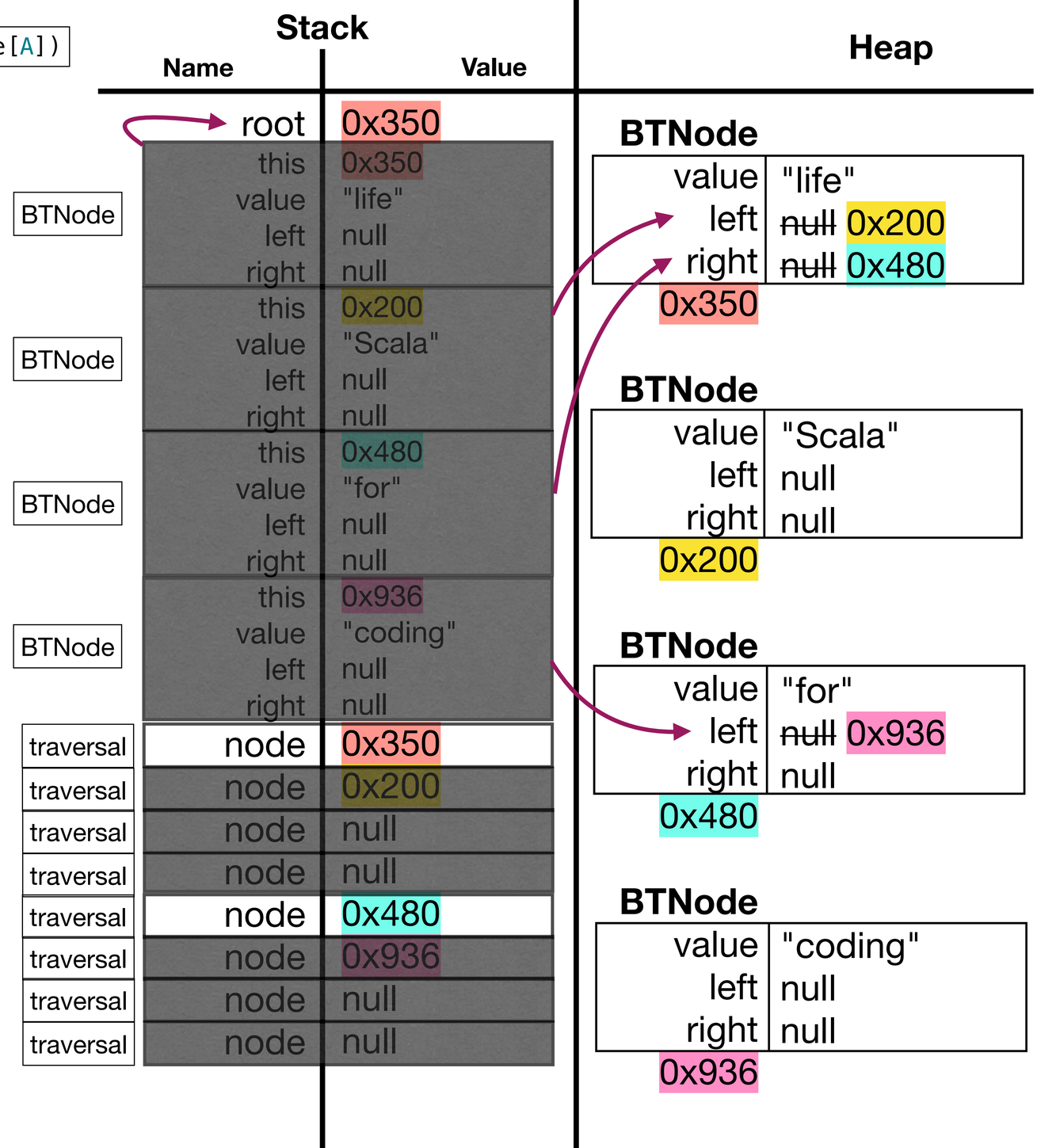
```
def traversal[A](node: BTNode[A]): Unit = {
  if (node != null) {
    traversal(node.left)
    traversal(node.right)
    print(node.value + " ")
  }
}

def main(args: Array[String]): Unit = {
  val root = new BTNode("life", null, null)
  root.left = new BTNode("Scala", null, null)
  root.right = new BTNode("for", null, null)
  root.right.left = new BTNode("coding", null, null)
  traversal(root)
}
```



in/out

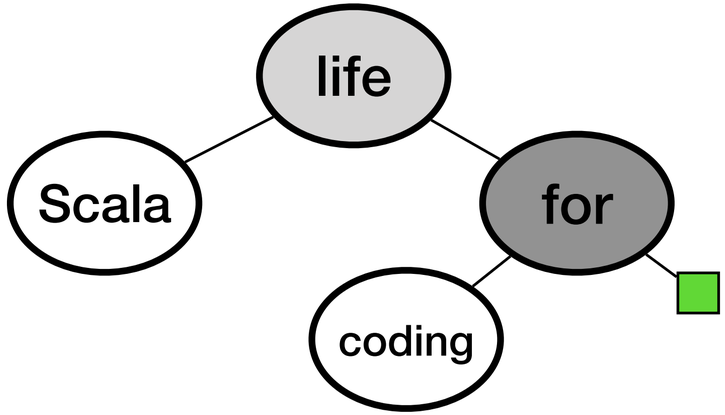
Scala coding

- Make right recursive call



```
class BTNode[A](var value: A, var left: BTNode[A], var right: BTNode[A])
```

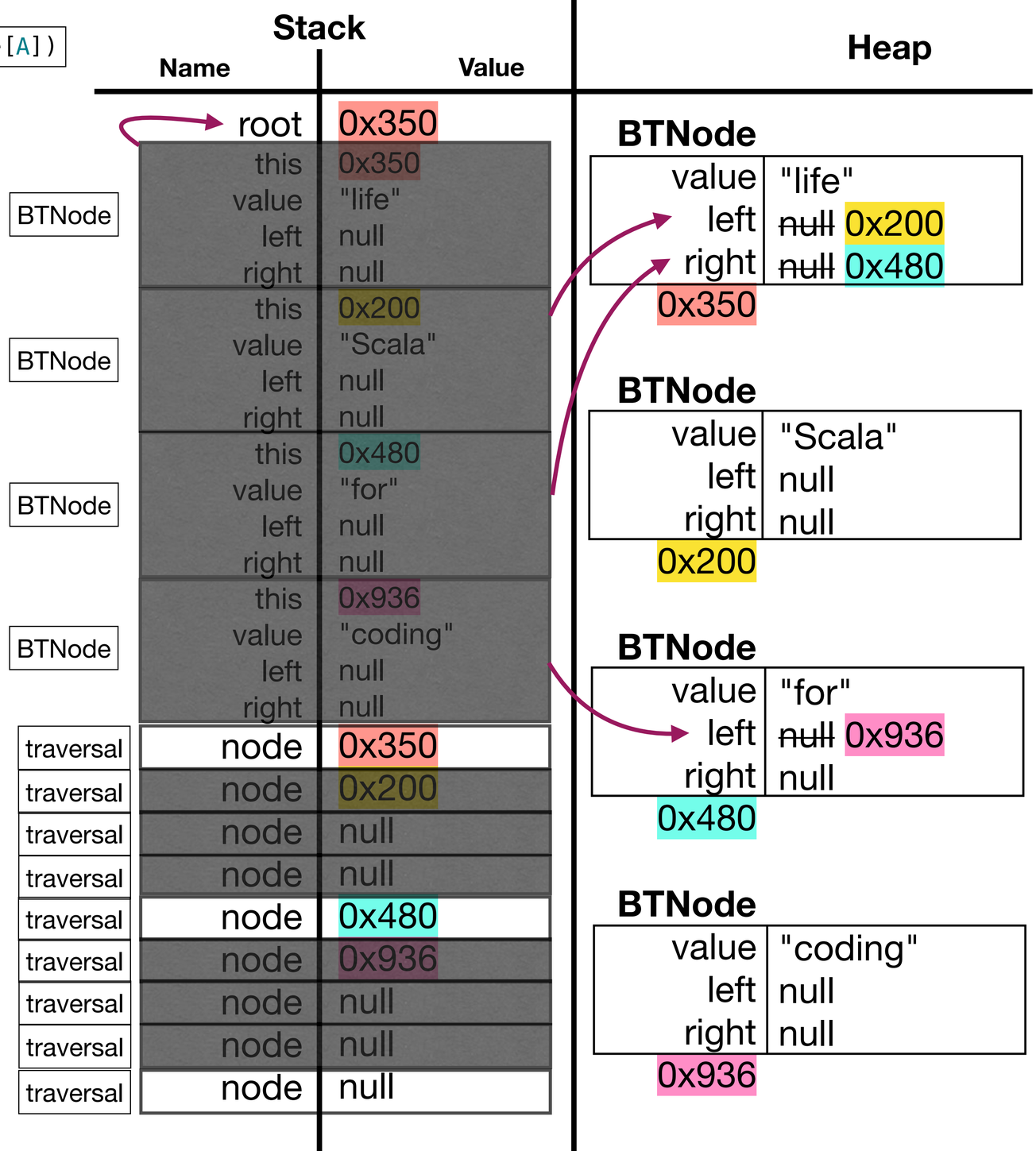
```
def traversal[A](node: BTNode[A]): Unit = {
  if (node != null) {
    traversal(node.left)
    traversal(node.right)
    print(node.value + " ")
  }
}

def main(args: Array[String]): Unit = {
  val root = new BTNode("life", null, null)
  root.left = new BTNode("Scala", null, null)
  root.right = new BTNode("for", null, null)
  root.right.left = new BTNode("coding", null, null)
  traversal(root)
}
```



in/out

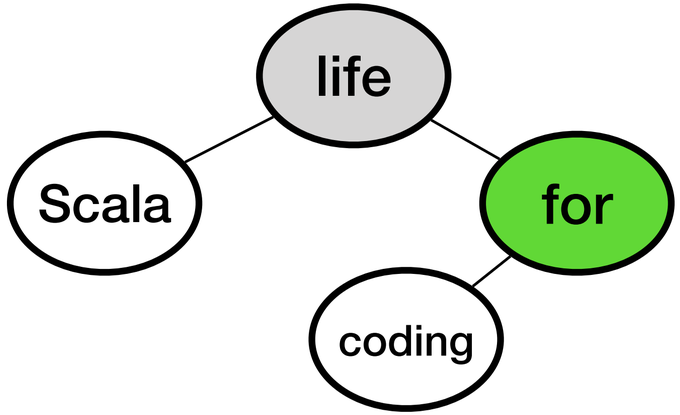
Scala coding

- Base case of null
- Just return



```
class BTNode[A](var value: A, var left: BTNode[A], var right: BTNode[A])
```

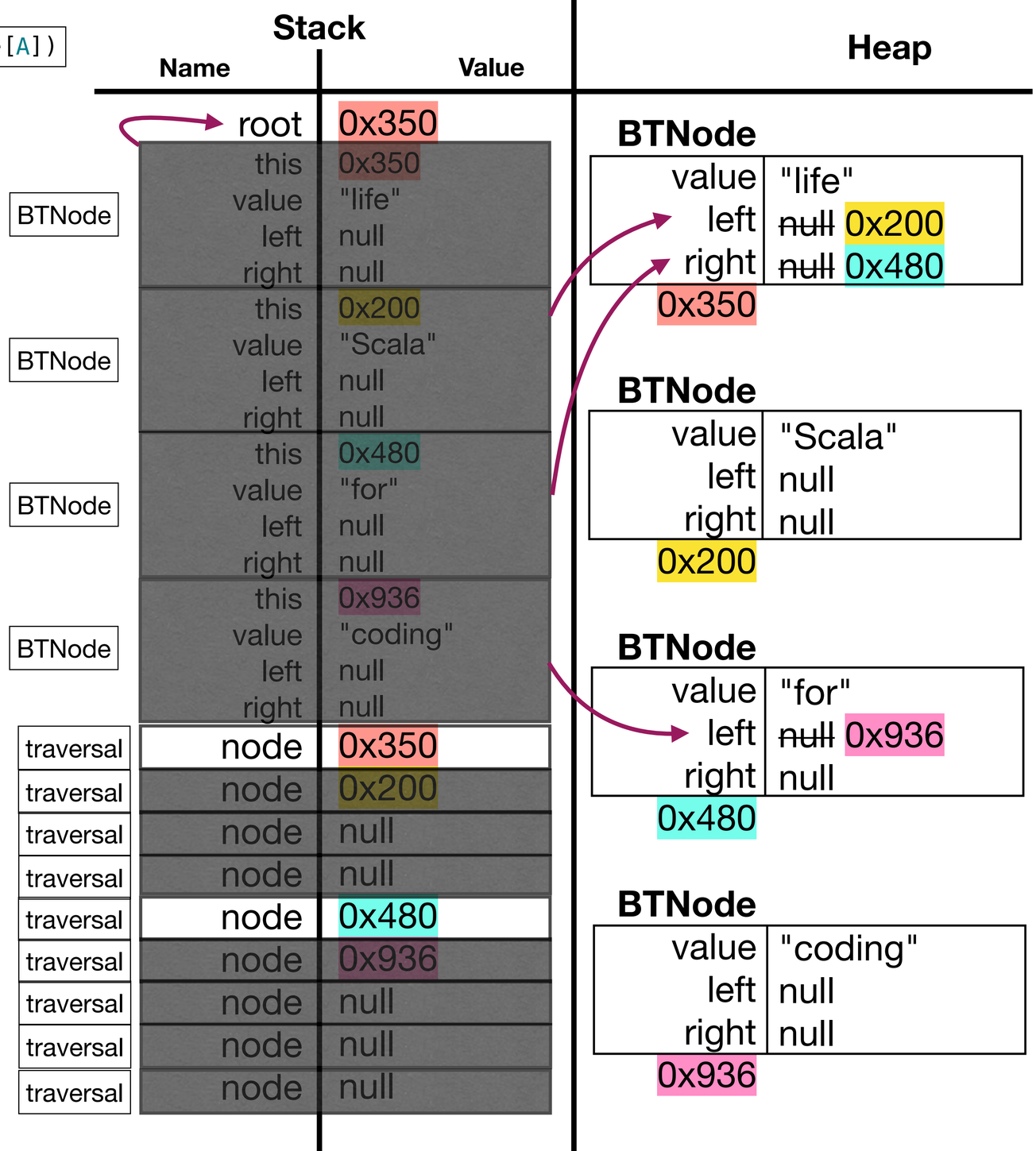
```
def traversal[A](node: BTNode[A]): Unit = {
  if (node != null) {
    traversal(node.left)
    traversal(node.right)
    print(node.value + " ")
  }
}

def main(args: Array[String]): Unit = {
  val root = new BTNode("life", null, null)
  root.left = new BTNode("Scala", null, null)
  root.right = new BTNode("for", null, null)
  root.right.left = new BTNode("coding", null, null)
  traversal(root)
}
```



in/out

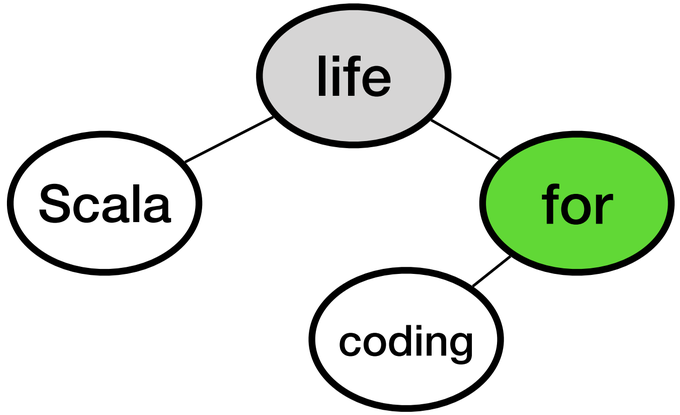
Scala coding

- Stack frame done with both recursive calls



```scala
class BTNode[A](var value: A, var left: BTNode[A], var right: BTNode[A])
```

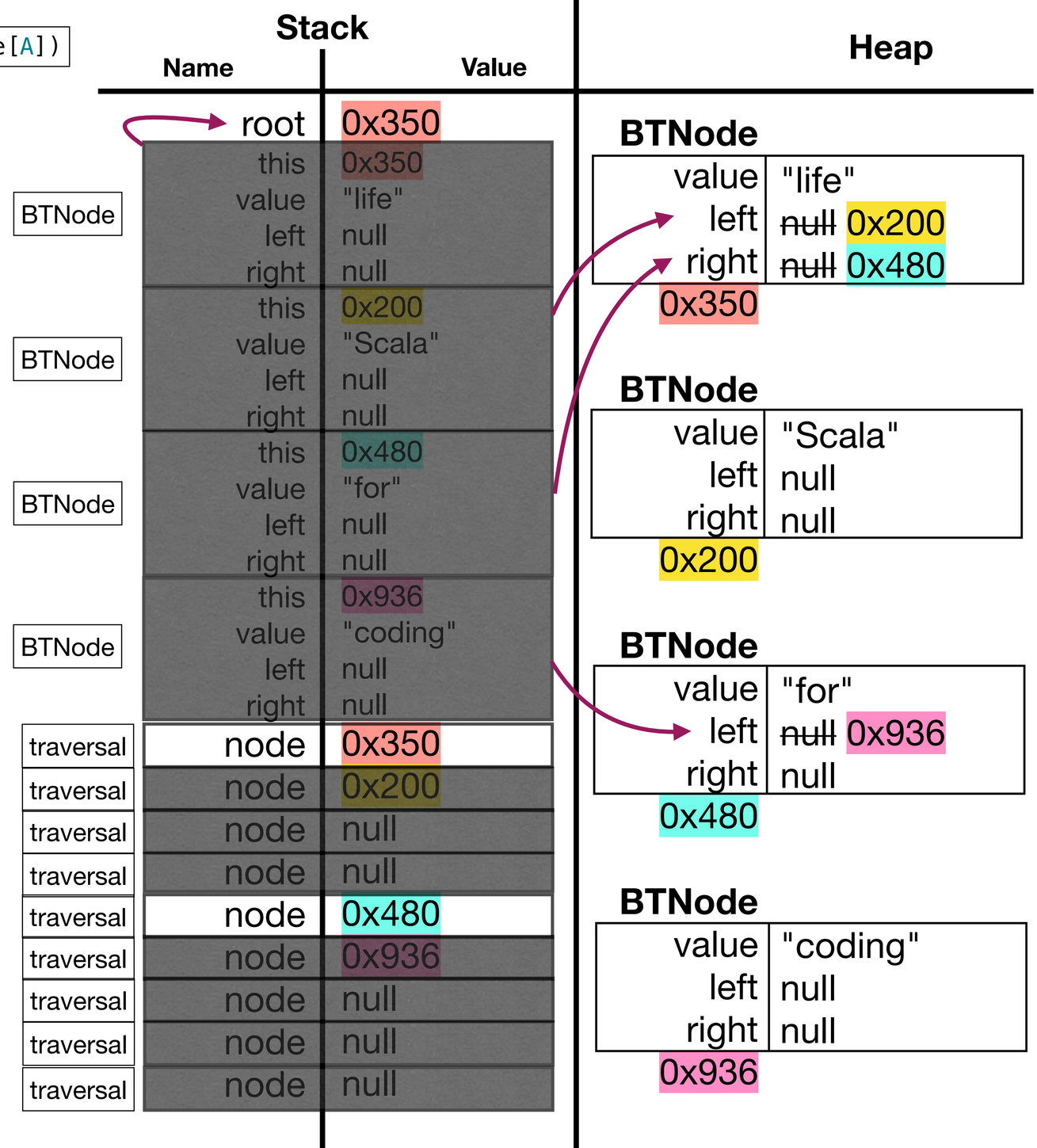
```scala
def traversal[A](node: BTNode[A]): Unit = {
  if (node != null) {
    traversal(node.left)
    traversal(node.right)
    print(node.value + " ")
  }
}

def main(args: Array[String]): Unit = {
  val root = new BTNode("life", null, null)
  root.left = new BTNode("Scala", null, null)
  root.right = new BTNode("for", null, null)
  root.right.left = new BTNode("coding", null, null)
  traversal(root)
}
```



in/out

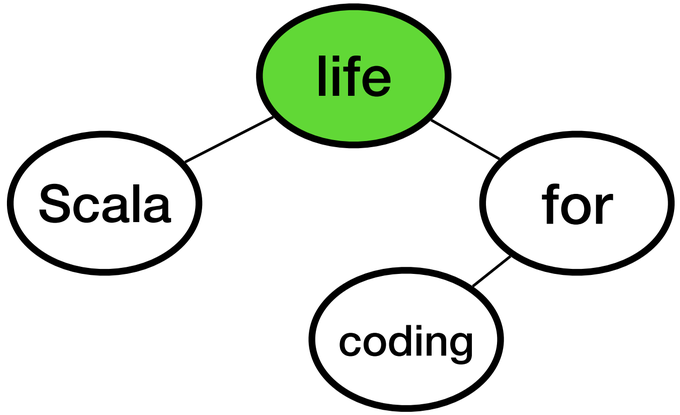
Scala coding for

- Print "for" to the screen



```
class BTNode[A](var value: A, var left: BTNode[A], var right: BTNode[A])
```

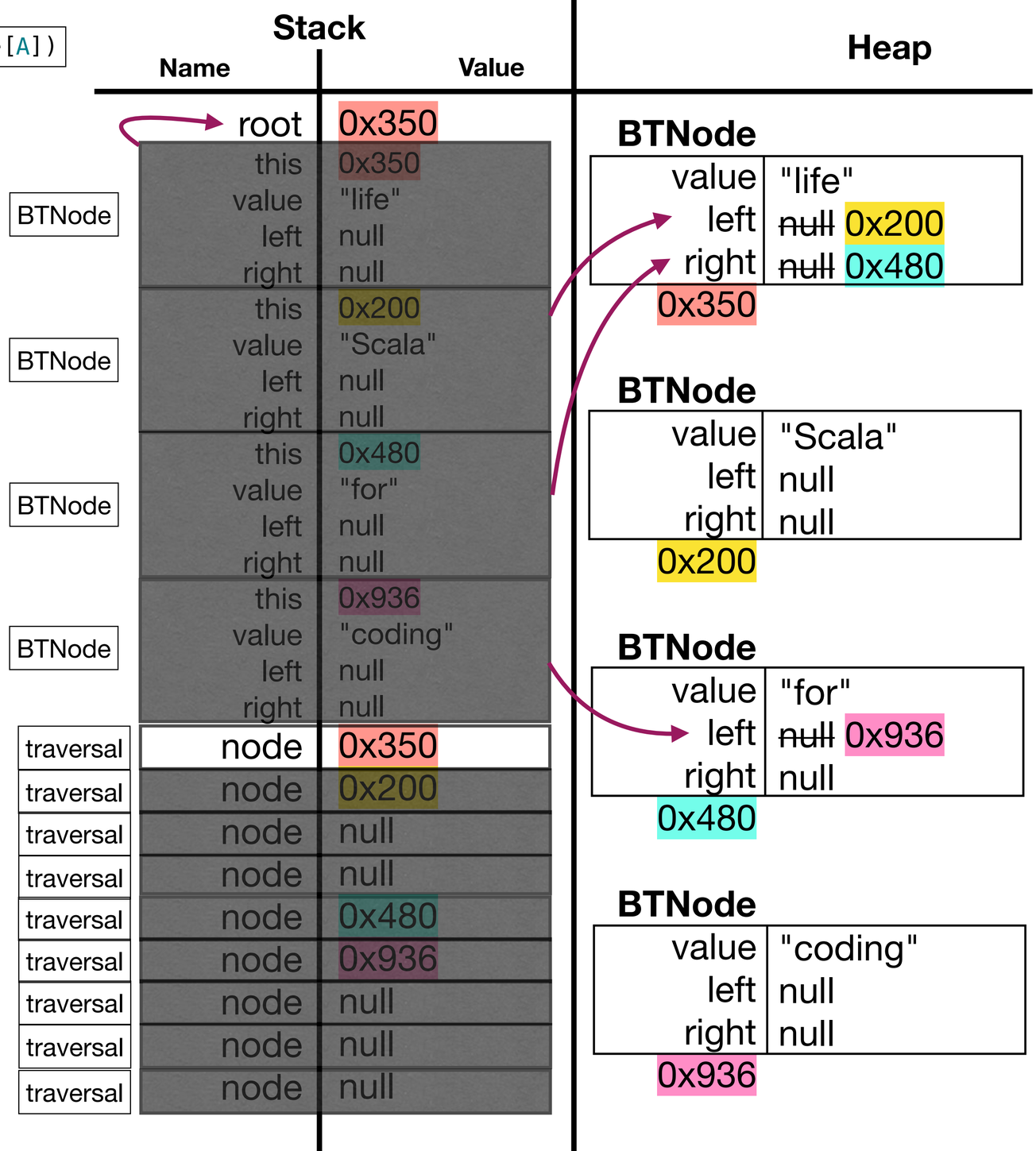
```
def traversal[A](node: BTNode[A]): Unit = {
  if (node != null) {
    traversal(node.left)
    traversal(node.right)
    print(node.value + " ")
  }
}

def main(args: Array[String]): Unit = {
  val root = new BTNode("life", null, null)
  root.left = new BTNode("Scala", null, null)
  root.right = new BTNode("for", null, null)
  root.right.left = new BTNode("coding", null, null)
  traversal(root)
}
```



in/out

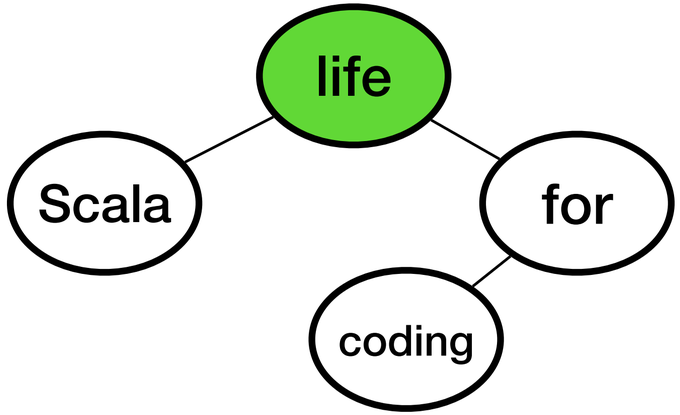
Scala coding for

- Root node is now done with both recursive calls



```
class BTNode[A](var value: A, var left: BTNode[A], var right: BTNode[A])
```

```
def traversal[A](node: BTNode[A]): Unit = {
  if (node != null) {
    traversal(node.left)
    traversal(node.right)
    print(node.value + " ")
  }
}

def main(args: Array[String]): Unit = {
  val root = new BTNode("life", null, null)
  root.left = new BTNode("Scala", null, null)
  root.right = new BTNode("for", null, null)
  root.right.left = new BTNode("coding", null, null)
  traversal(root)
}
```



in/out

Scala coding for life

- Print "life" to the screen

Stack

	Name	Value
	root	0x350
BTNode	this	0x350
	value	"life"
	left	null
	right	null
BTNode	this	0x200
	value	"Scala"
	left	null
	right	null
BTNode	this	0x480
	value	"for"
	left	null
	right	null
BTNode	this	0x936
	value	"coding"
	left	null
	right	null
traversal	node	0x350
traversal	node	0x200
traversal	node	null
traversal	node	null
traversal	node	0x480
traversal	node	0x936
traversal	node	null
traversal	node	null
traversal	node	null

Heap

BTNode 0x350

value	"life"
left	~~null~~ 0x200
right	~~null~~ 0x480

BTNode 0x200

value	"Scala"
left	null
right	null

BTNode 0x480

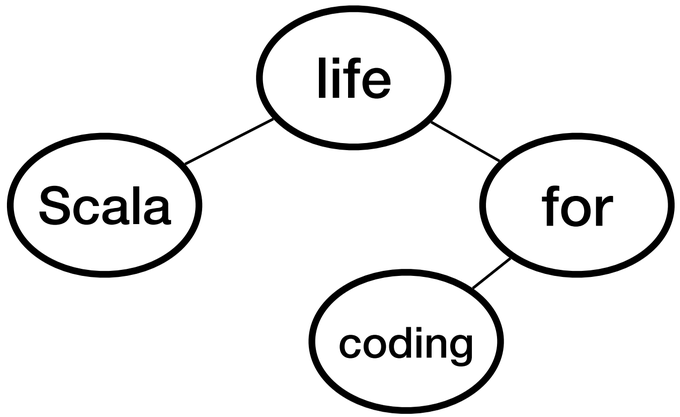
value	"for"
left	~~null~~ 0x936
right	null

BTNode 0x936

value	"coding"
left	null
right	null

```
class BTNode[A](var value: A, var left: BTNode[A], var right: BTNode[A])
```

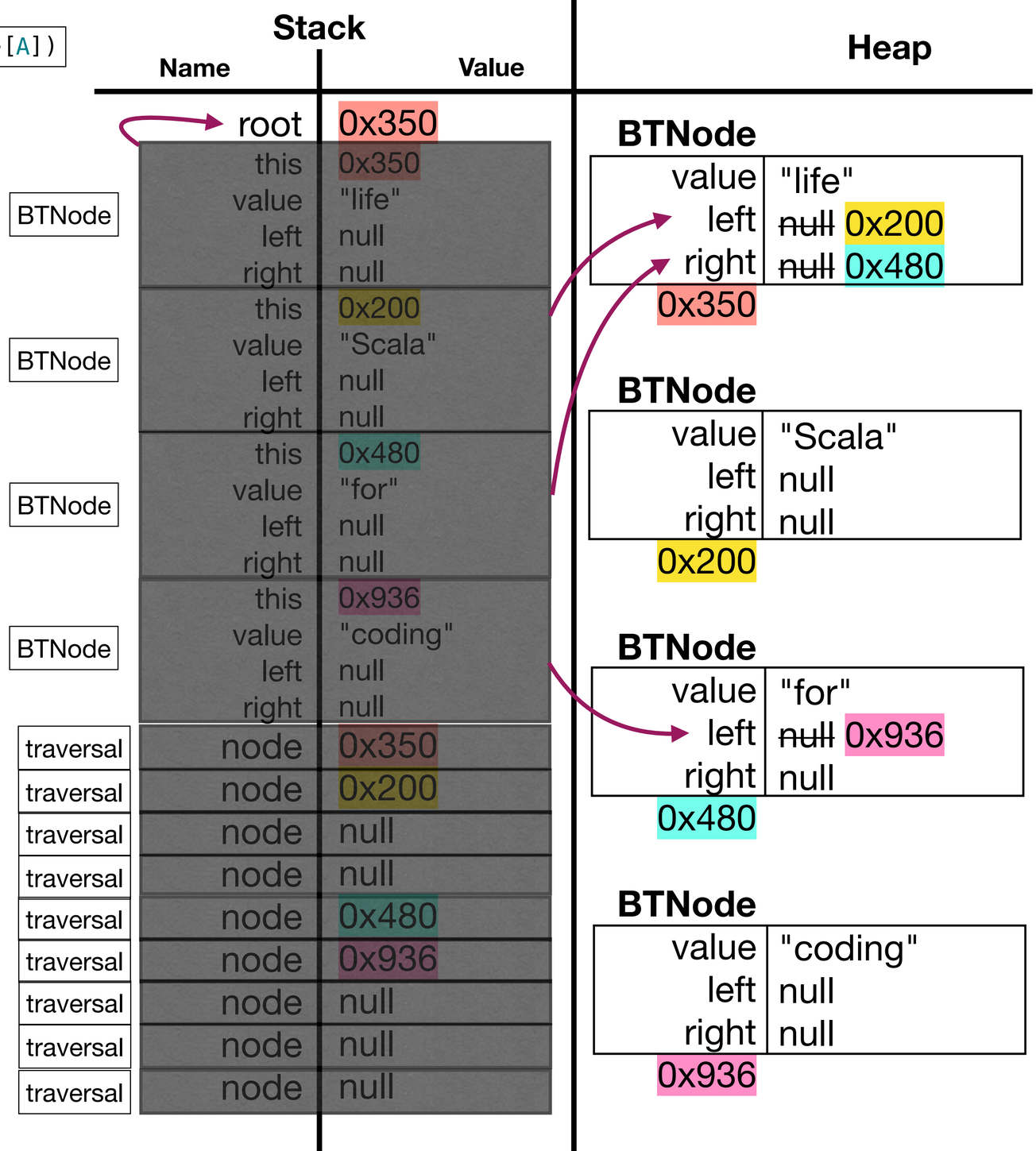
```
def traversal[A](node: BTNode[A]): Unit = {
  if (node != null) {
    traversal(node.left)
    traversal(node.right)
    print(node.value + " ")
  }
}

def main(args: Array[String]): Unit = {
  val root = new BTNode("life", null, null)
  root.left = new BTNode("Scala", null, null)
  root.right = new BTNode("for", null, null)
  root.right.left = new BTNode("coding", null, null)
  traversal(root)
}
```



in/out

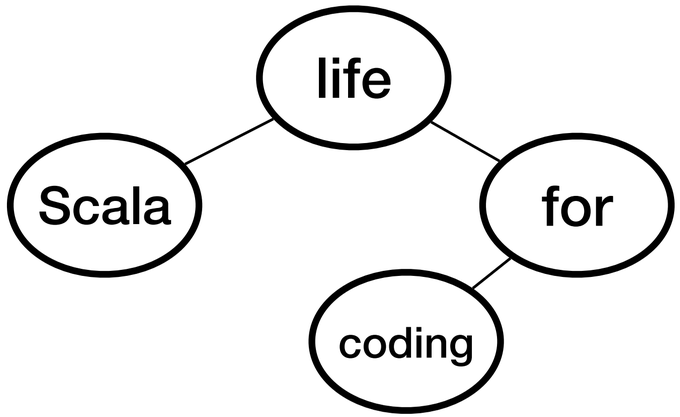
Scala coding for life

- The traversal is over
- All stack frames have returned



```
class BTNode[A](var value: A, var left: BTNode[A], var right: BTNode[A])
```

```
def traversal[A](node: BTNode[A]): Unit = {
  if (node != null) {
    traversal(node.left)
    traversal(node.right)
    print(node.value + " ")
  }
}

def main(args: Array[String]): Unit = {
  val root = new BTNode("life", null, null)
  root.left = new BTNode("Scala", null, null)
  root.right = new BTNode("for", null, null)
  root.right.left = new BTNode("coding", null, null)
  traversal(root)
}
```



in/out

Scala coding for life

- Program ends