Memory Diagrams

More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

- Draw a memory diagram for this program

More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

Stack	Heap

- Start by setting up the diagram
- Separate columns for stack and heap memory

More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

- Separate the stack into name and value

Stack		Heap
Name	Value	

More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

- Add a section wherever you have room for inputs and outputs
  - This is where you what what's printed to the screen

Stack		Heap
Name	**Value**	
		<u>in/out</u>

More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

Stack		Heap
Name	Value	
x	5	
y	2	
		in/out

- Start tracing the program from main
- First 2 lines add variables to the stack

More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

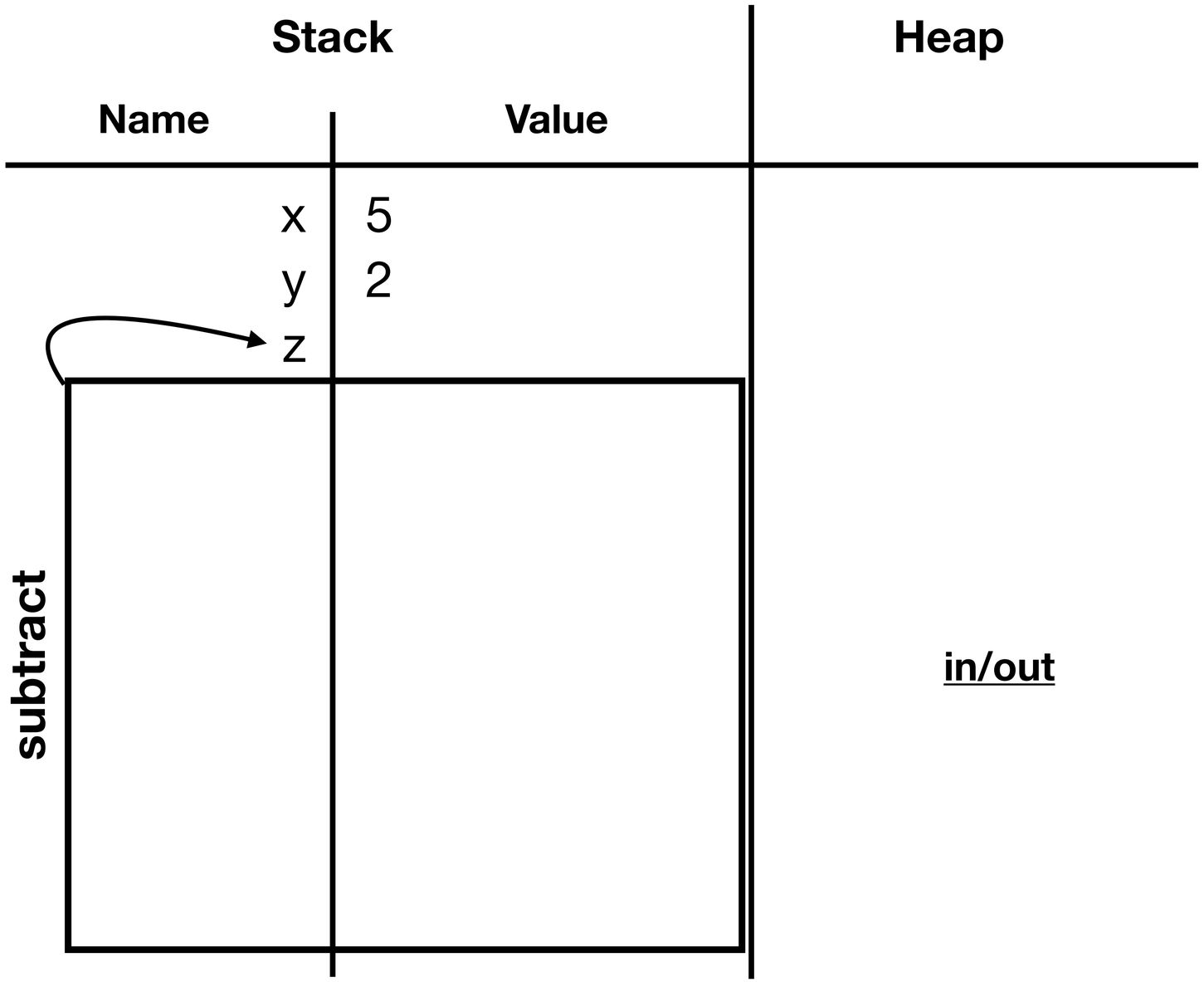
- We call a method named subtract
  - Add the return variable to the stack

Stack		Heap
Name	**Value**	
x	5	
y	2	
z		
		<u>in/out</u>

More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

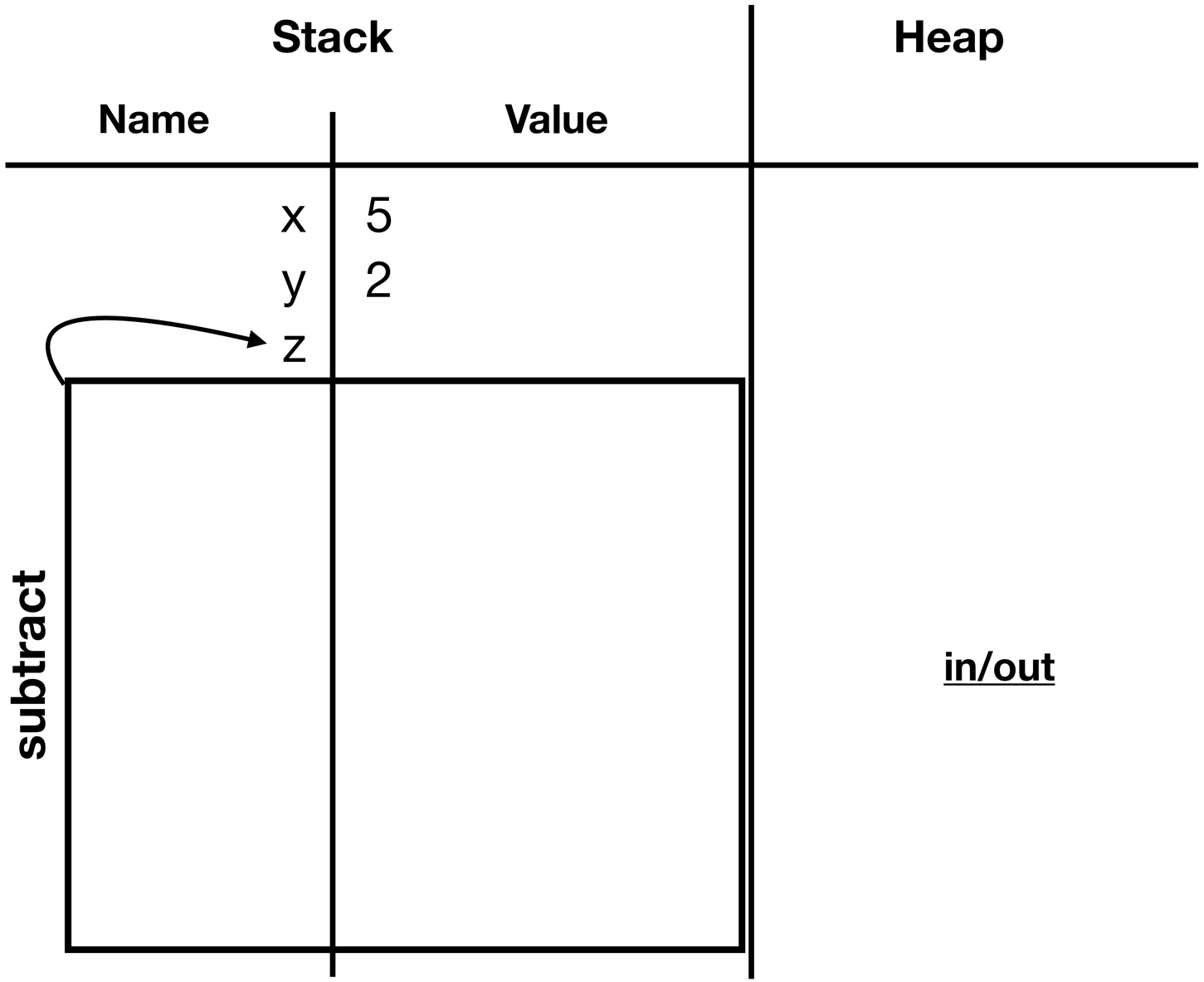


- Draw a solid box for the **Stack Frame**
- Arrow to the return variable and name of method being called

More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

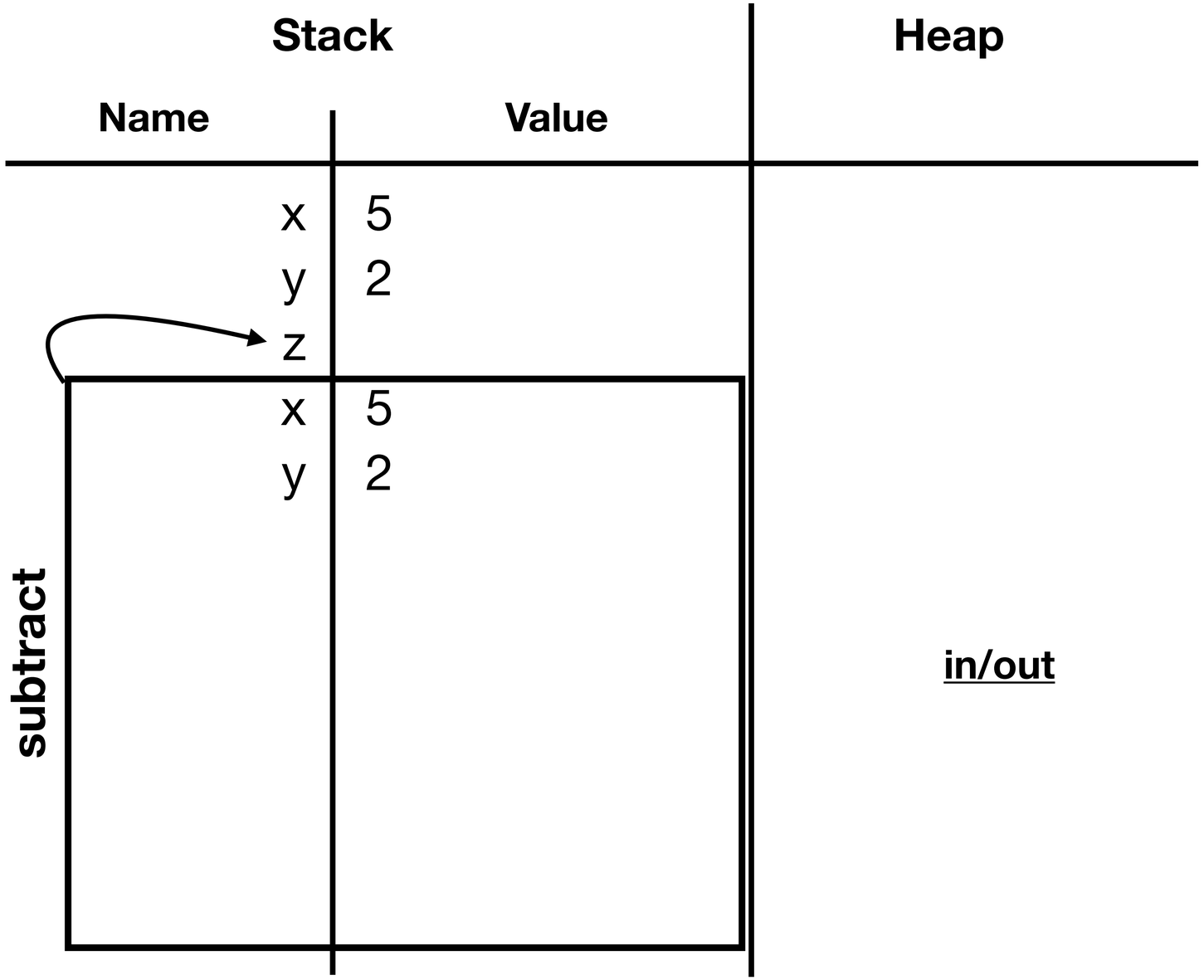


- The stack frame cannot be crossed
- Can't access variables across the solid box

More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

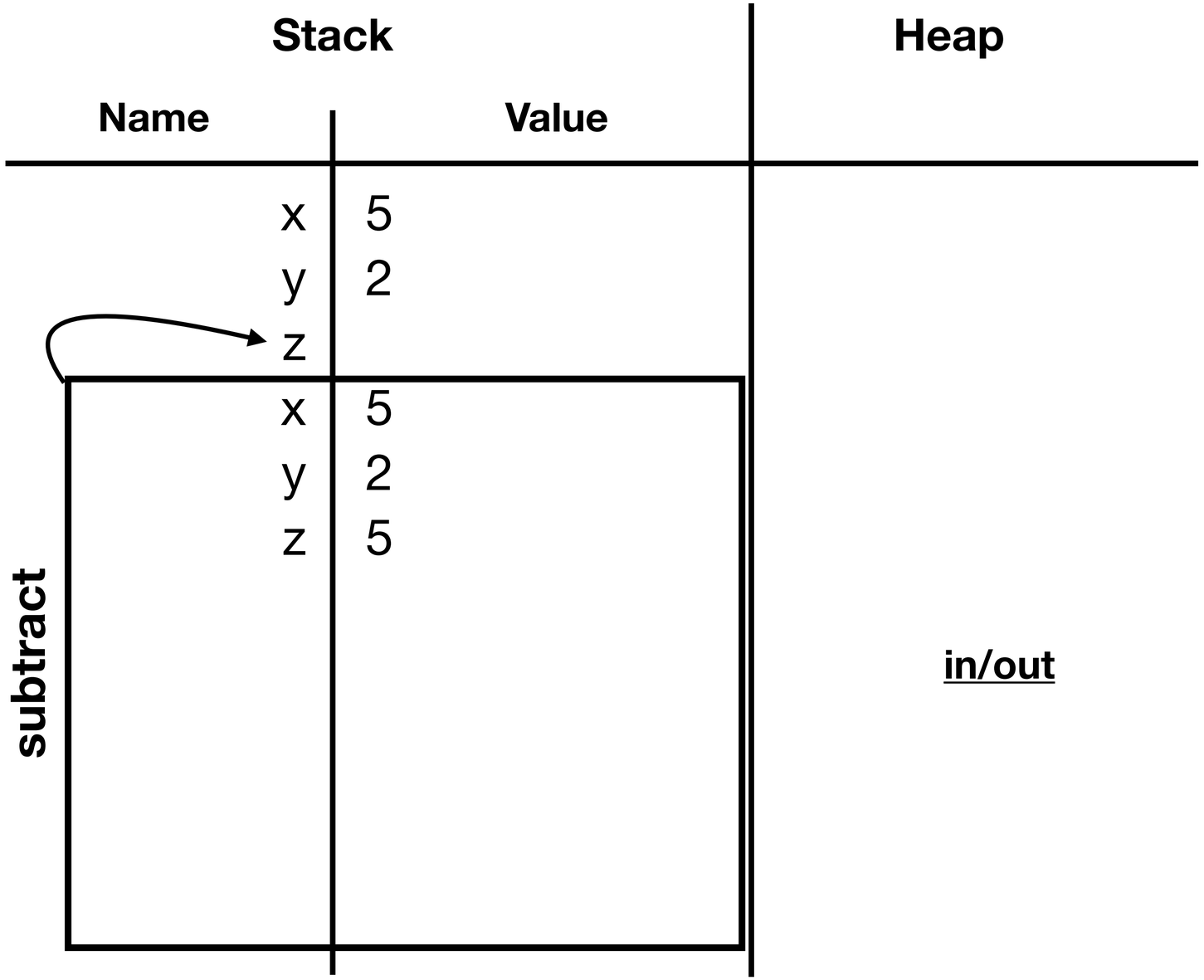


- Add the names of the parameters with the values of the arguments inside the stack frame

More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

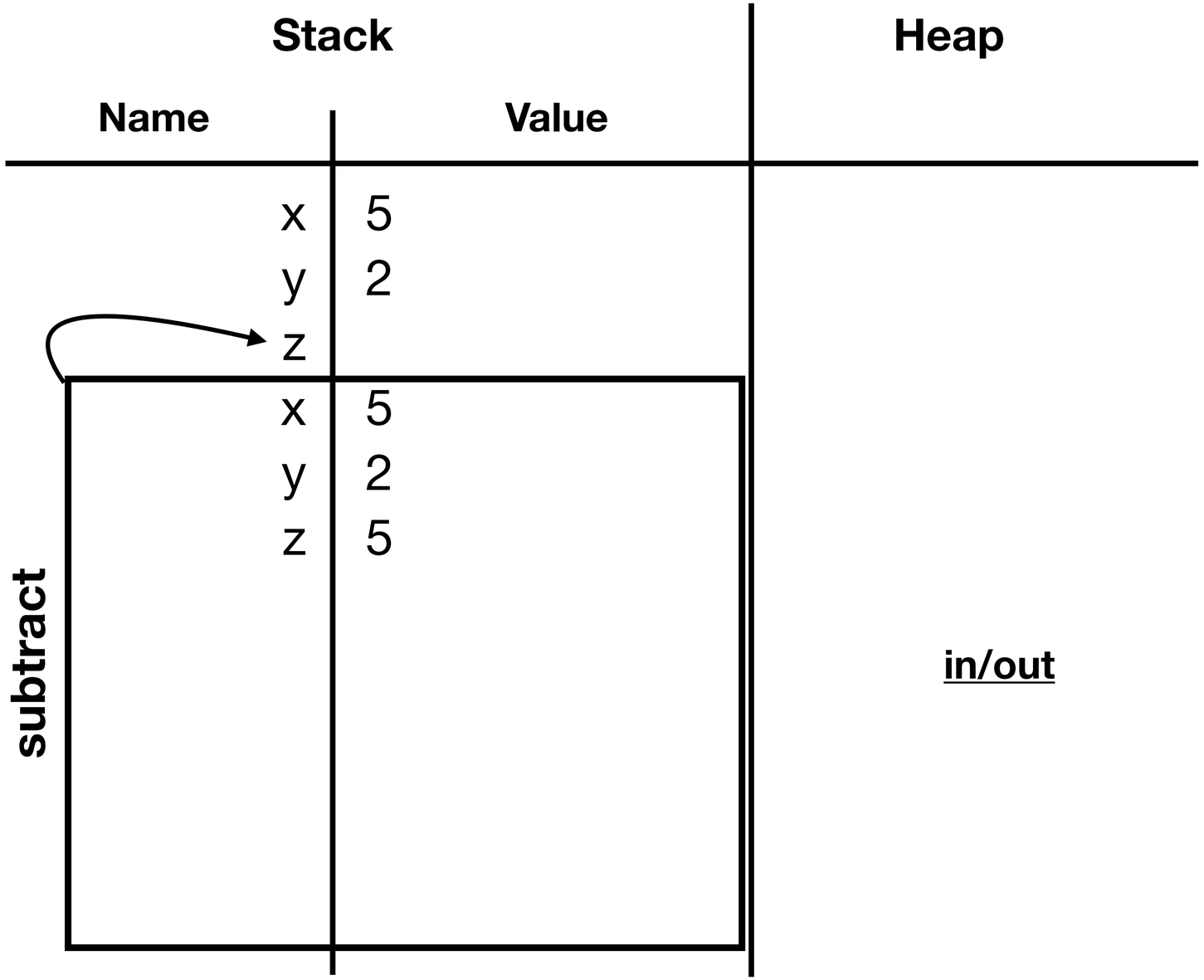


- Add z equal to the value of x
- Two x's on the stack
  - Only 1 inside this stack frame!

More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```



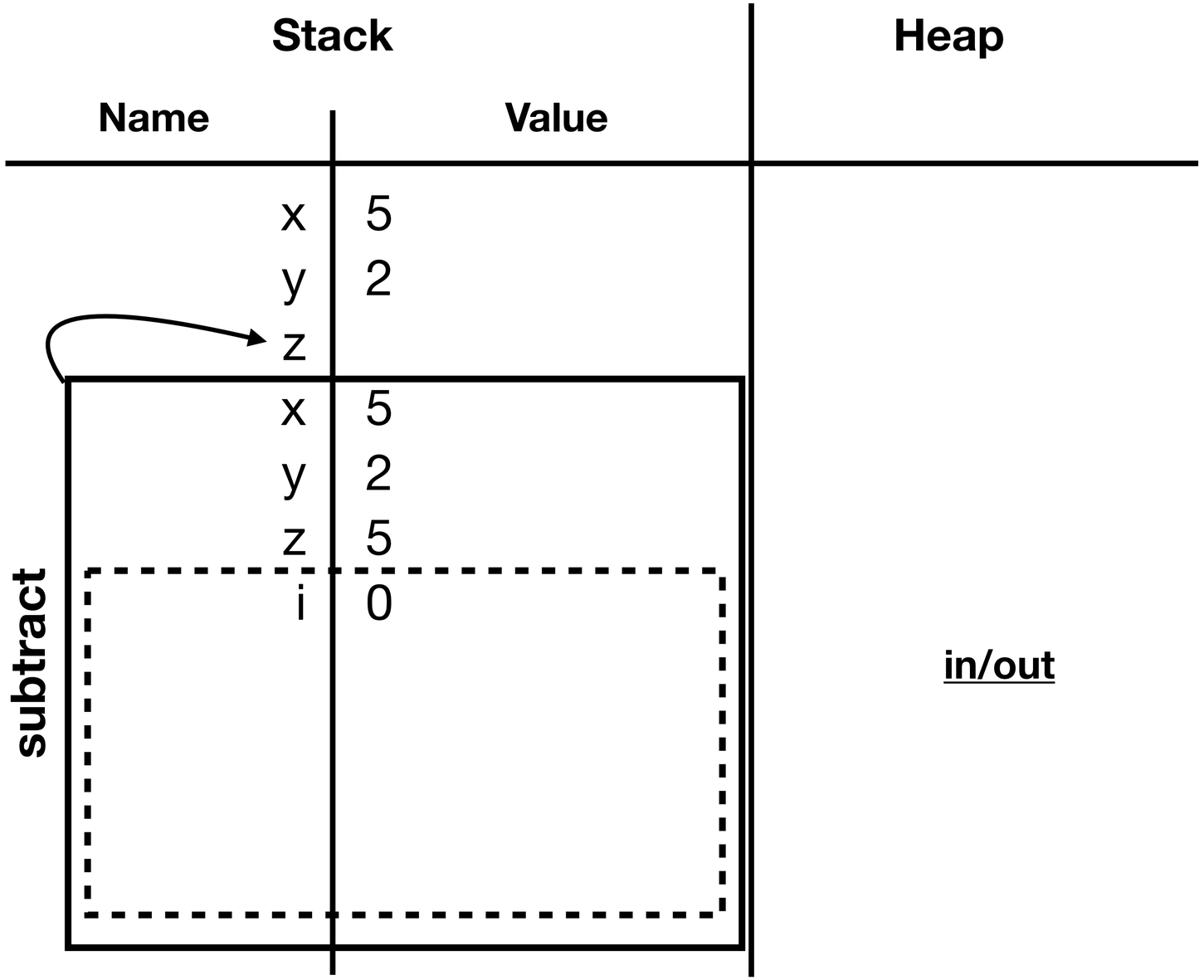
- Can reuse variable names in different stack frames

More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

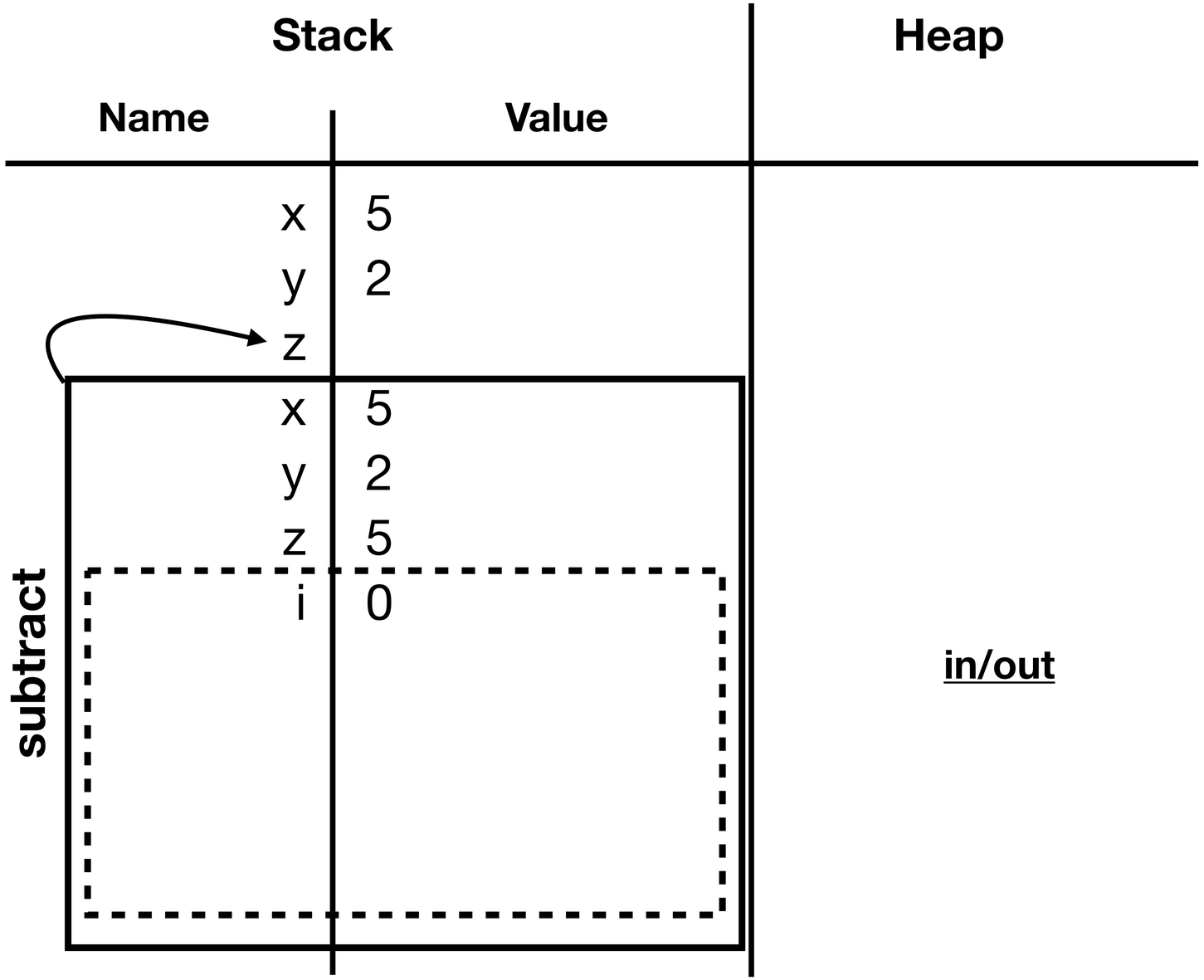
- Start of a code block for a loop
- New code block whenever there are { } that do not define a method



More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

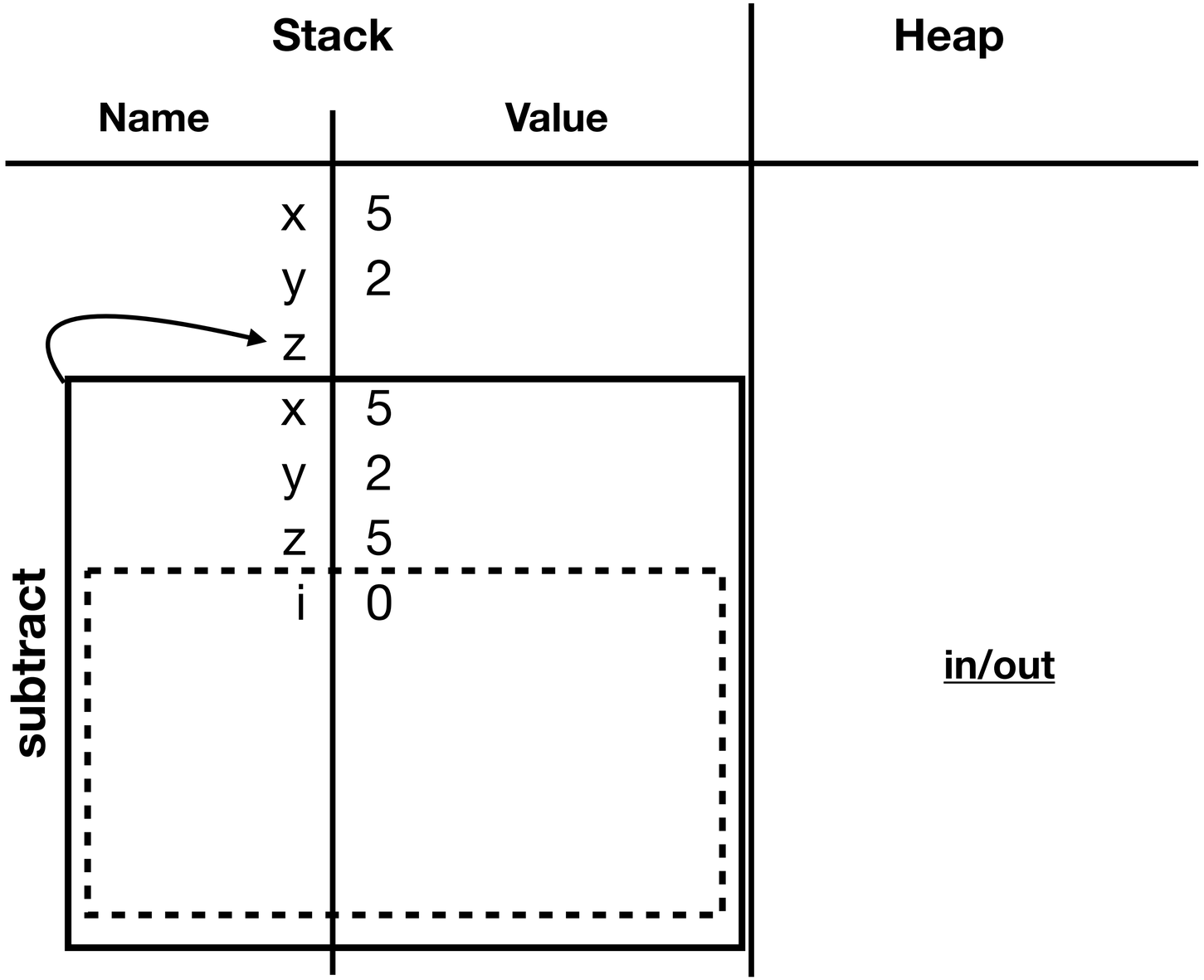


- Draw a dotted box for code blocks
- Dotted lines can *sometimes* be crossed

More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

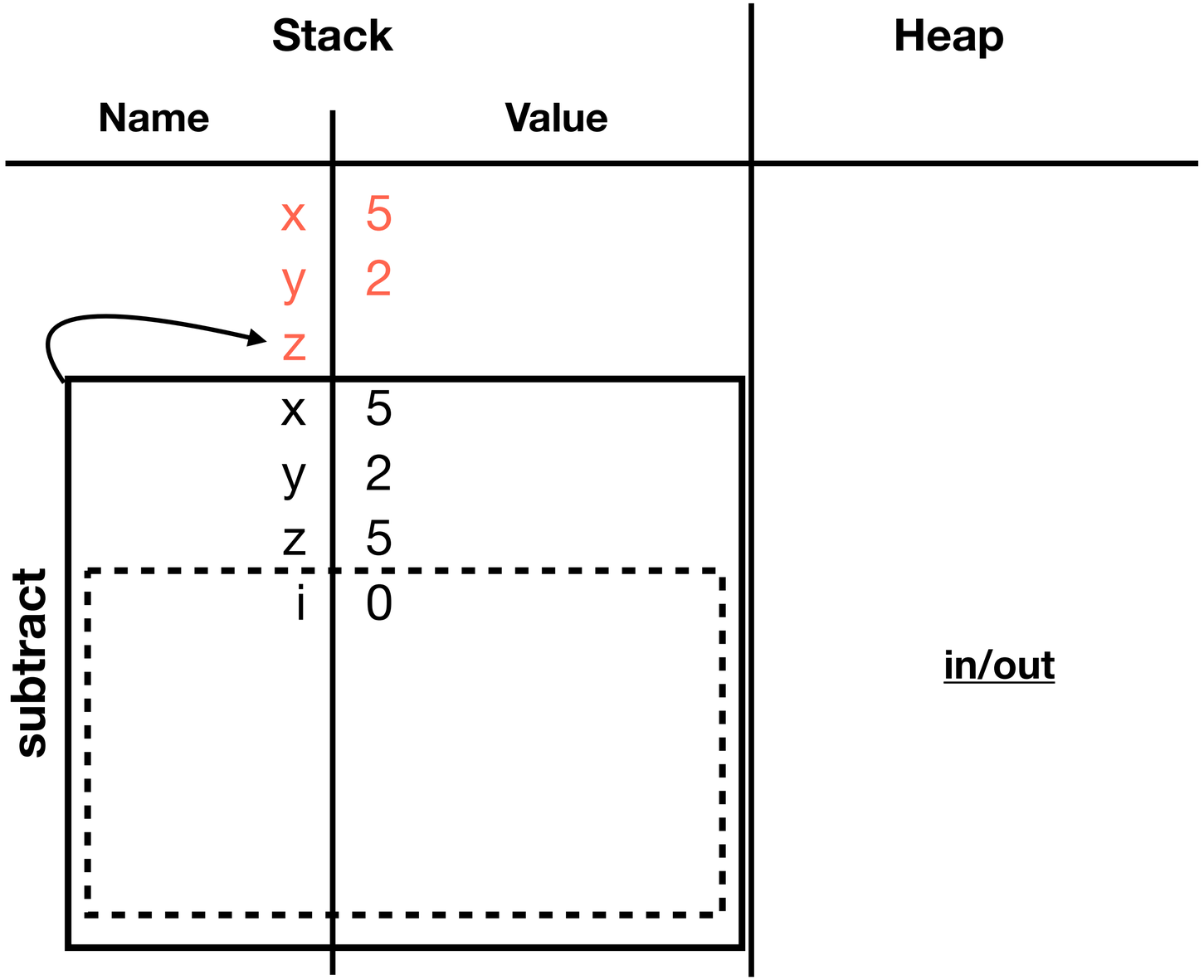


- Code blocks affect **Variable Scope**
- Variable scope determines which variables can be accessed

More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

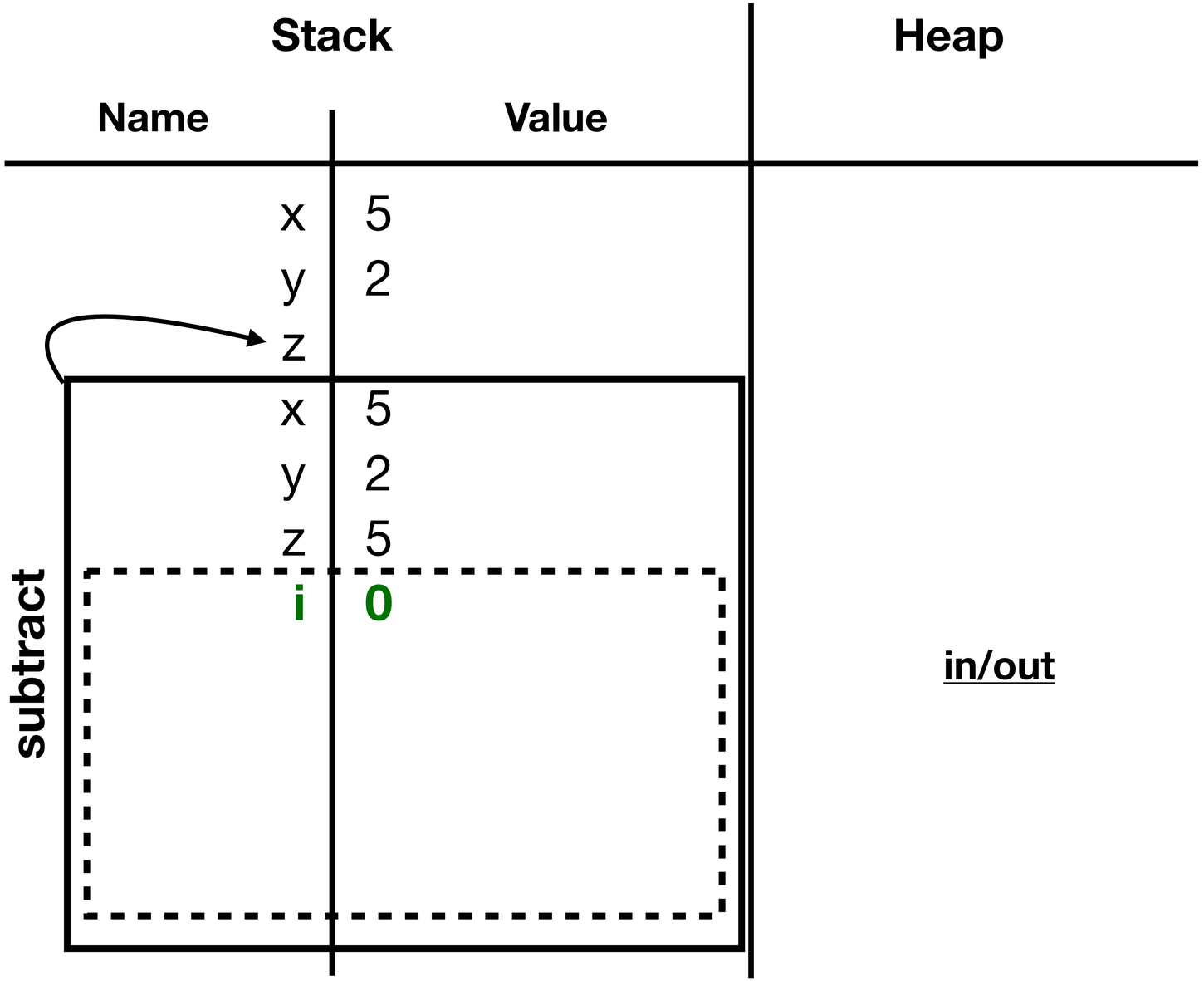


- Variables in another stack frame are always **out of scope**
- Cannot access variables across stack frames

More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

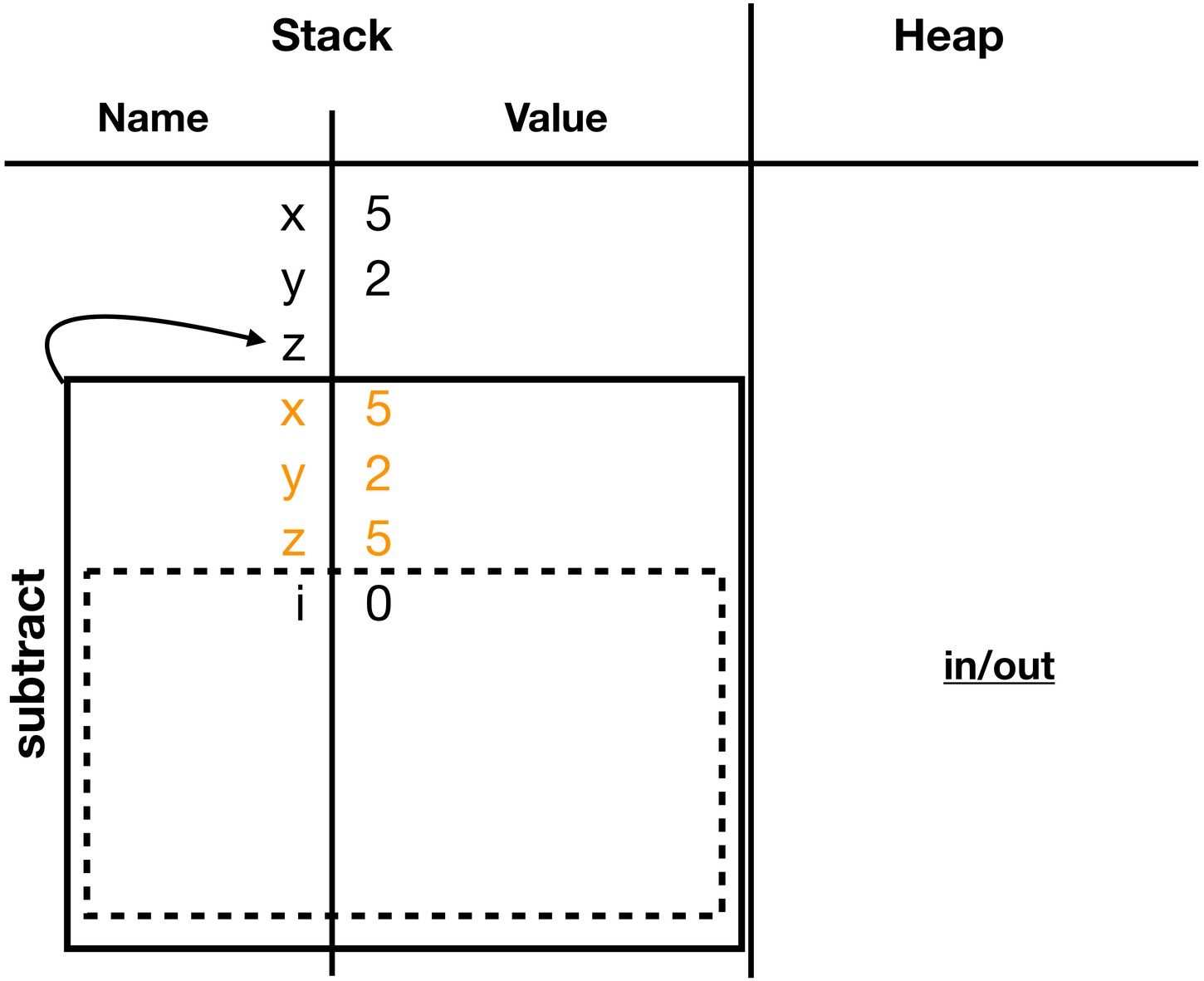


- Variables in the current code block are always **in scope**
- Can always be accessed

More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

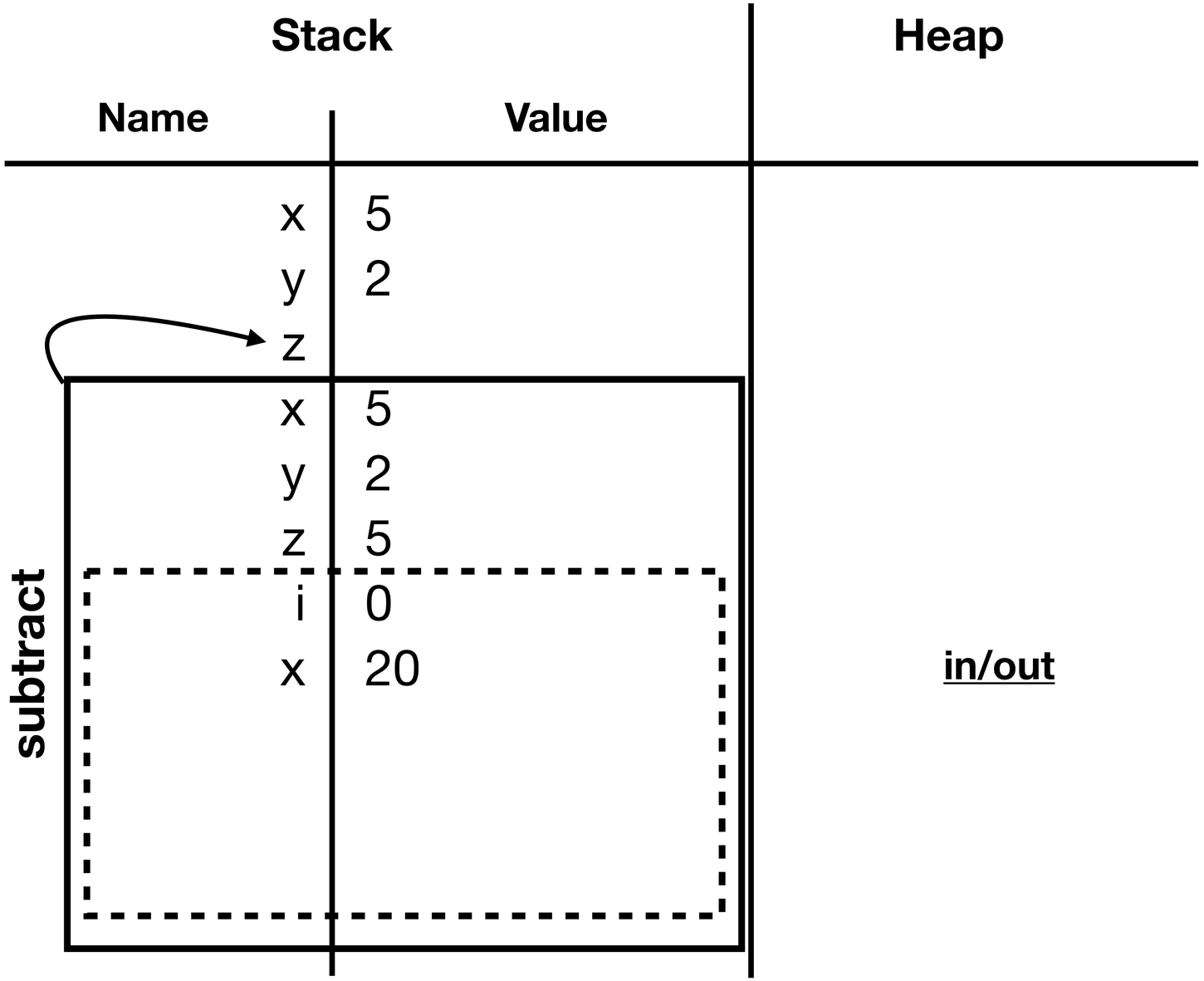


- Variable in the current stack frame, but not in the current code block, are *usually* **in scope**
  - Except when another variable with the same name is in scope

More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

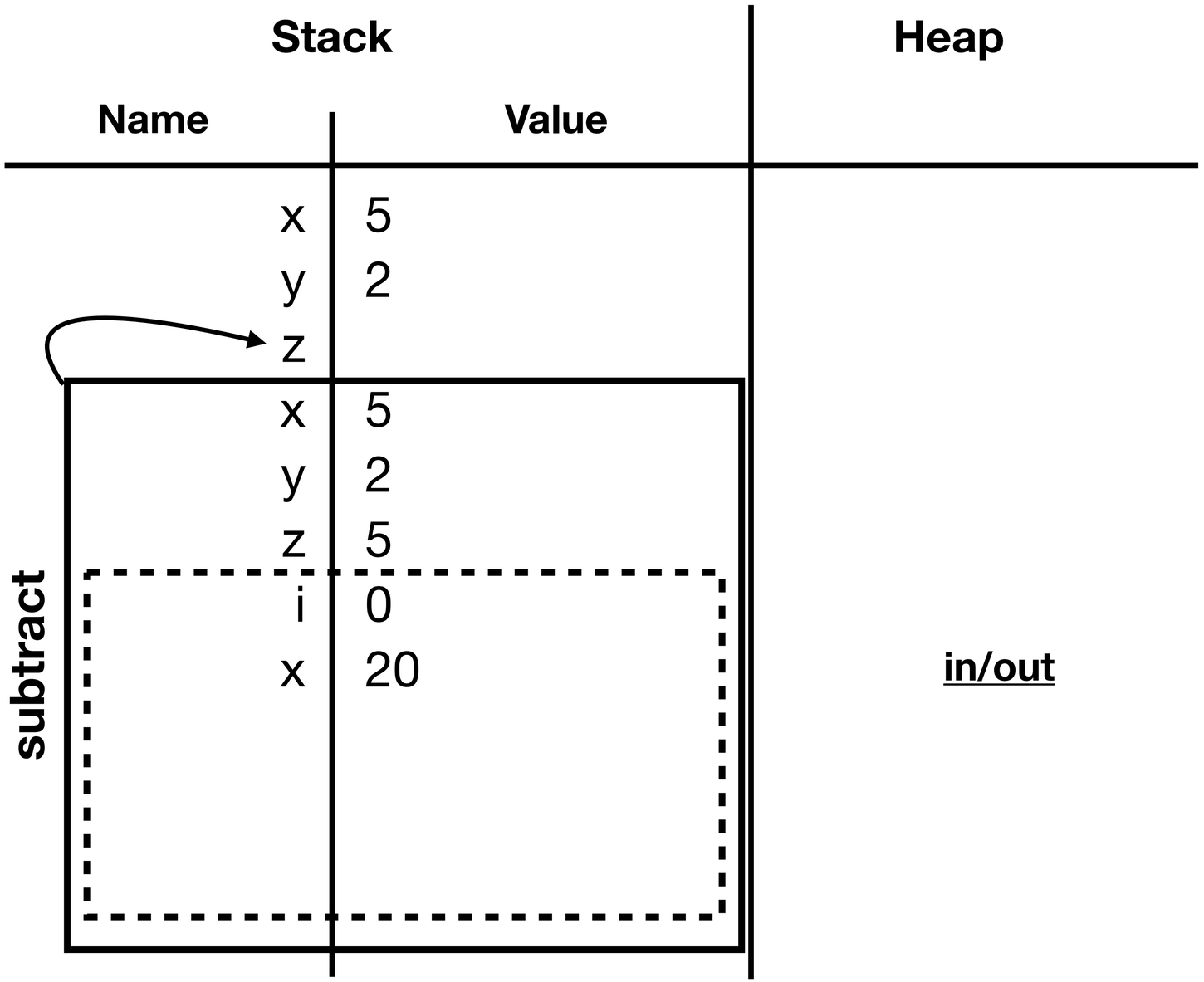


- Whenever val or var are used, a new variable is being added to the stack
- We can have multiple variables with the same name in the same stack frame!

More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

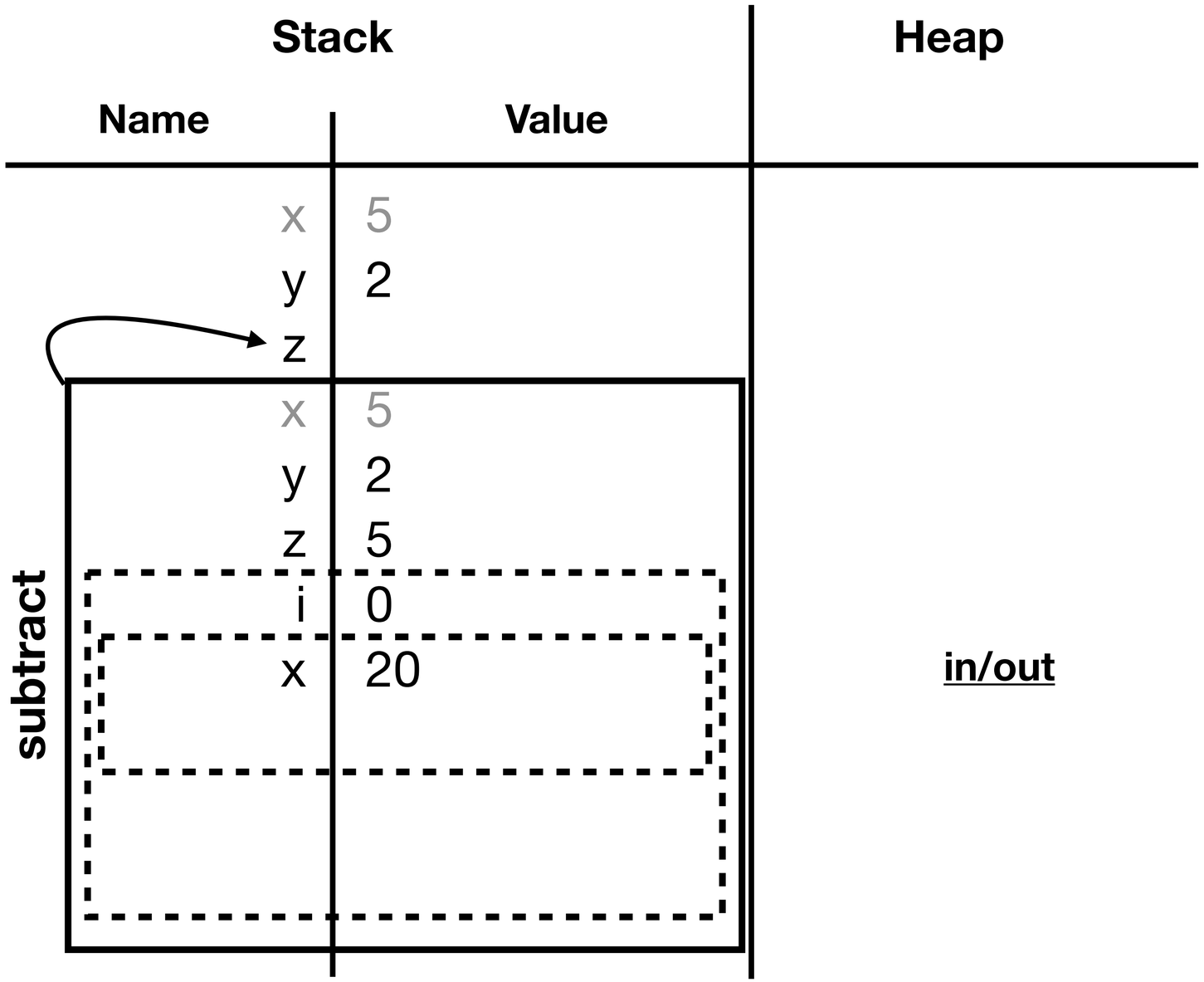


- We can have multiple variables with the same name in the same **stack frame**!
  - As long as they are not in the same **code block**

More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

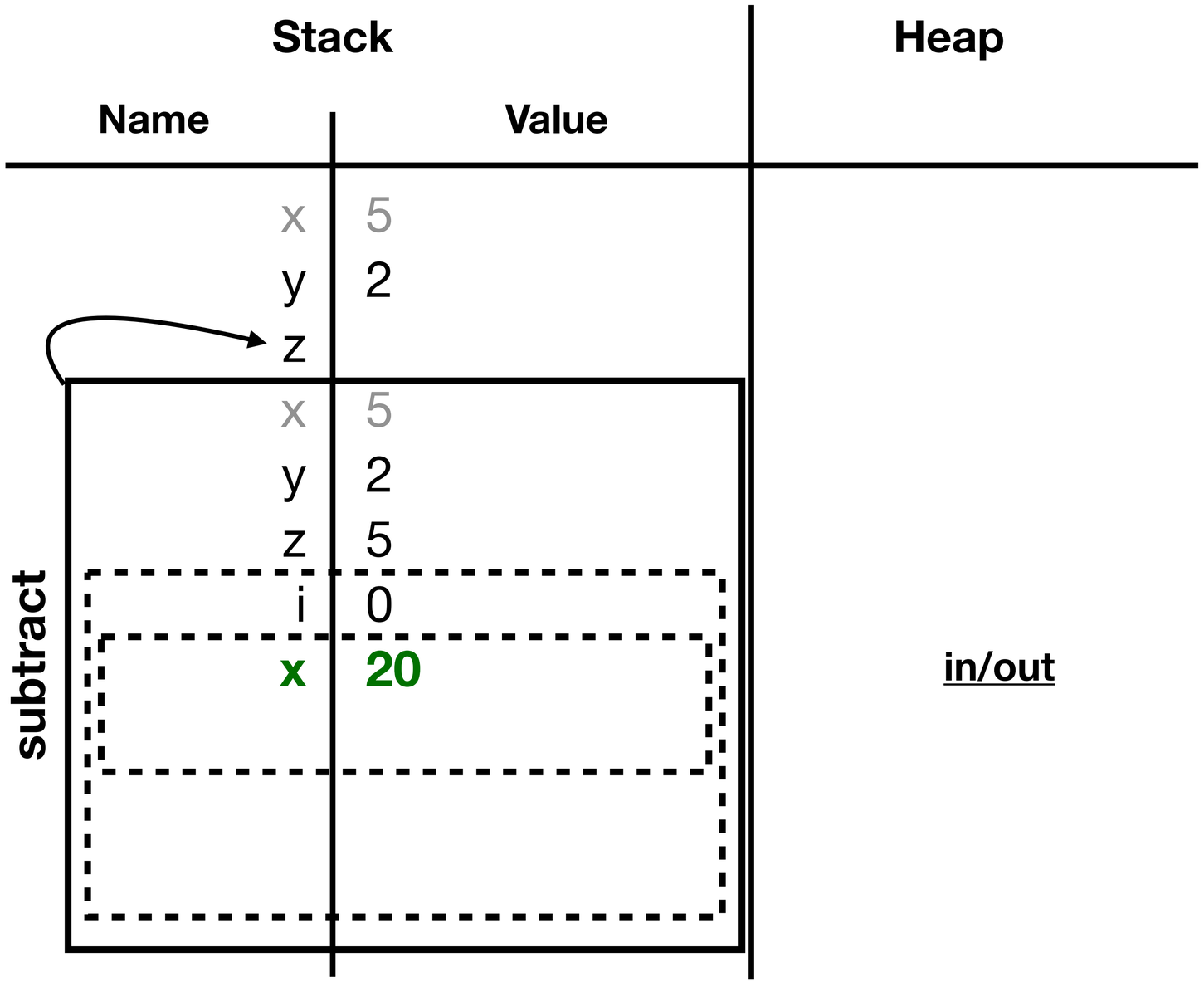


- For each iteration of the loop, draw a new dotted box
- Each iteration is a separate block with separate variables

More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```



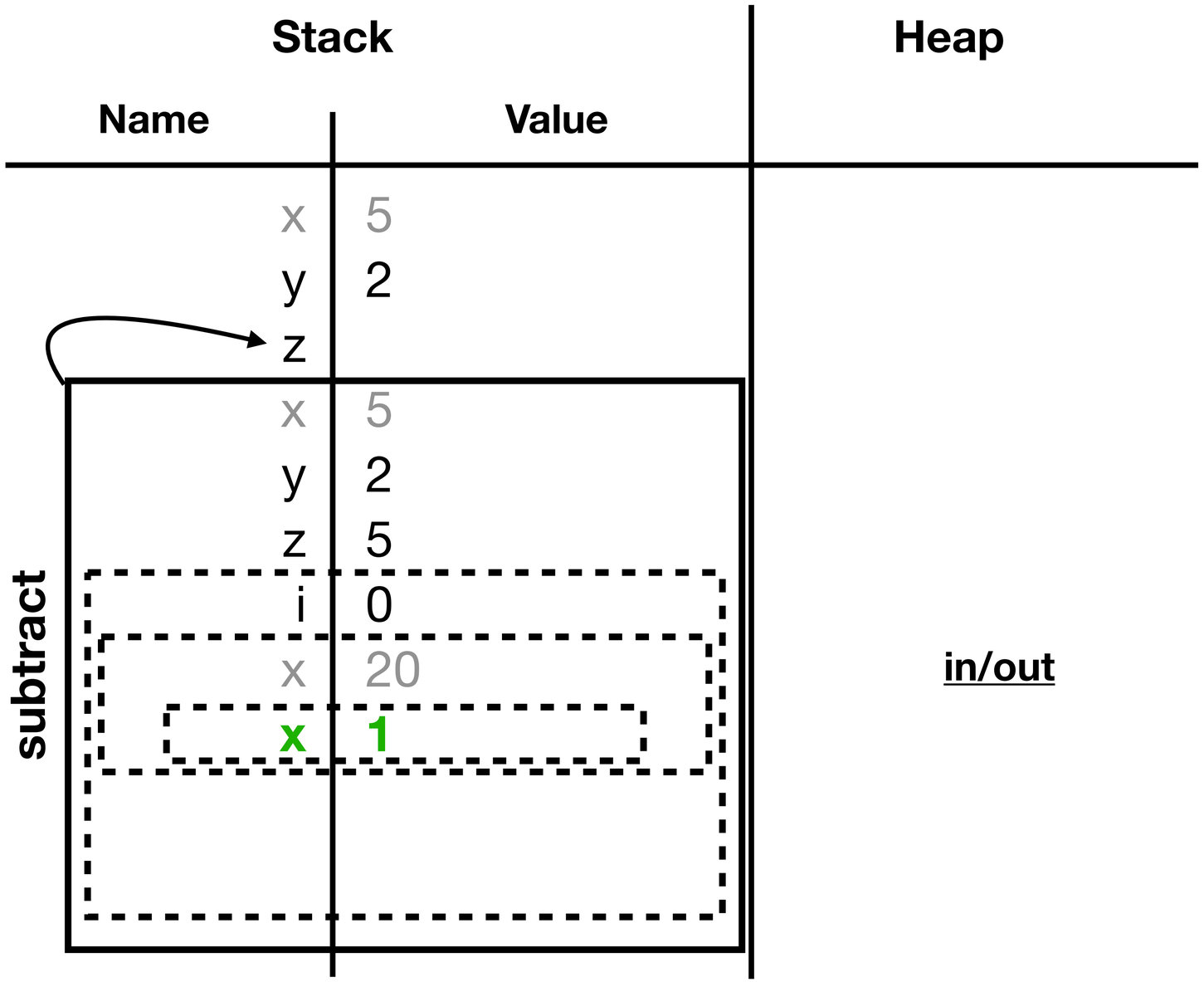
- Since variables in the current code block are always in scope, the x with 20 will be accessed if x is used
- The other 2 x's cannot be accessed

More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

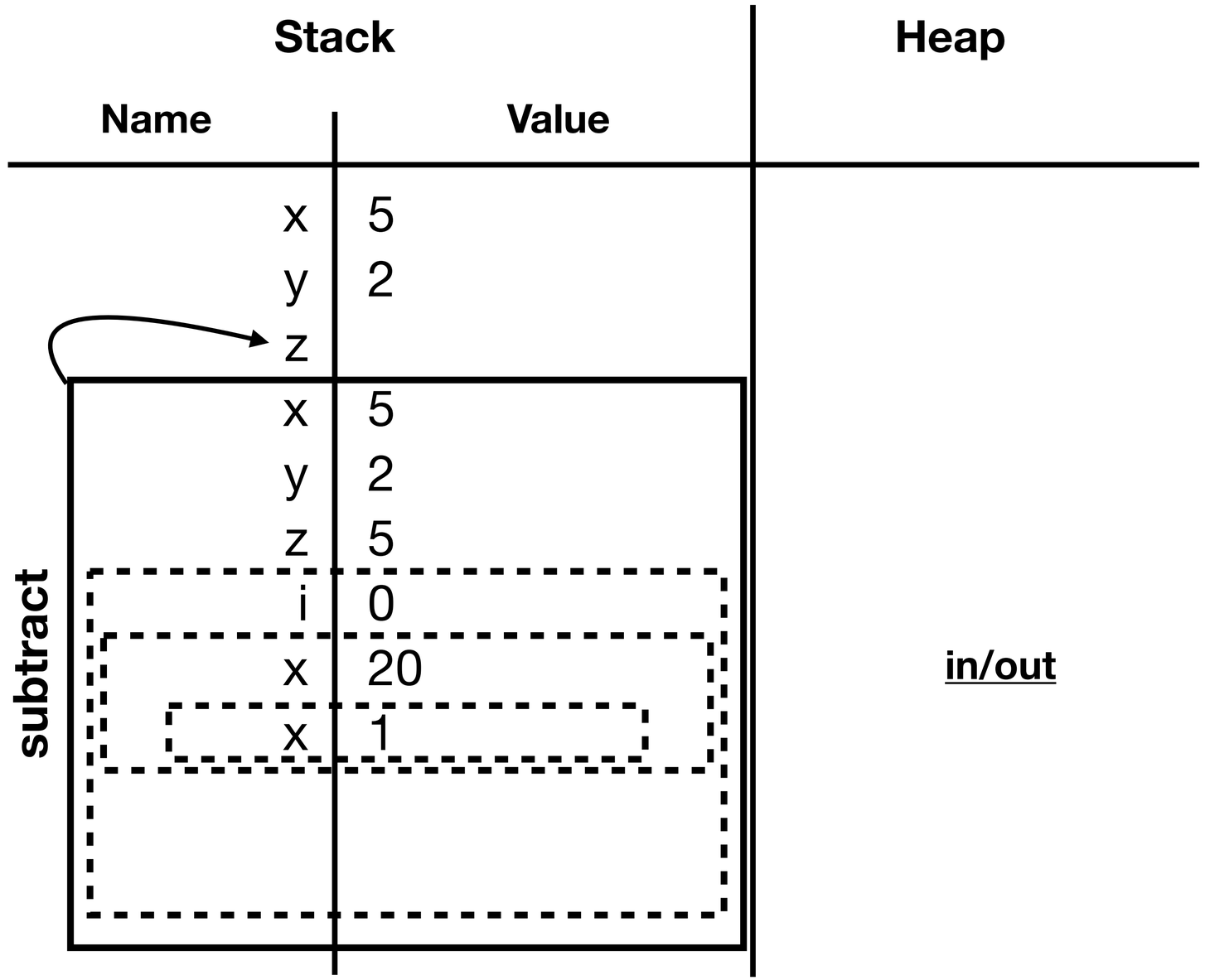
- Add another x to the stack in a new code block
- Now this is the only x in scope



More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```



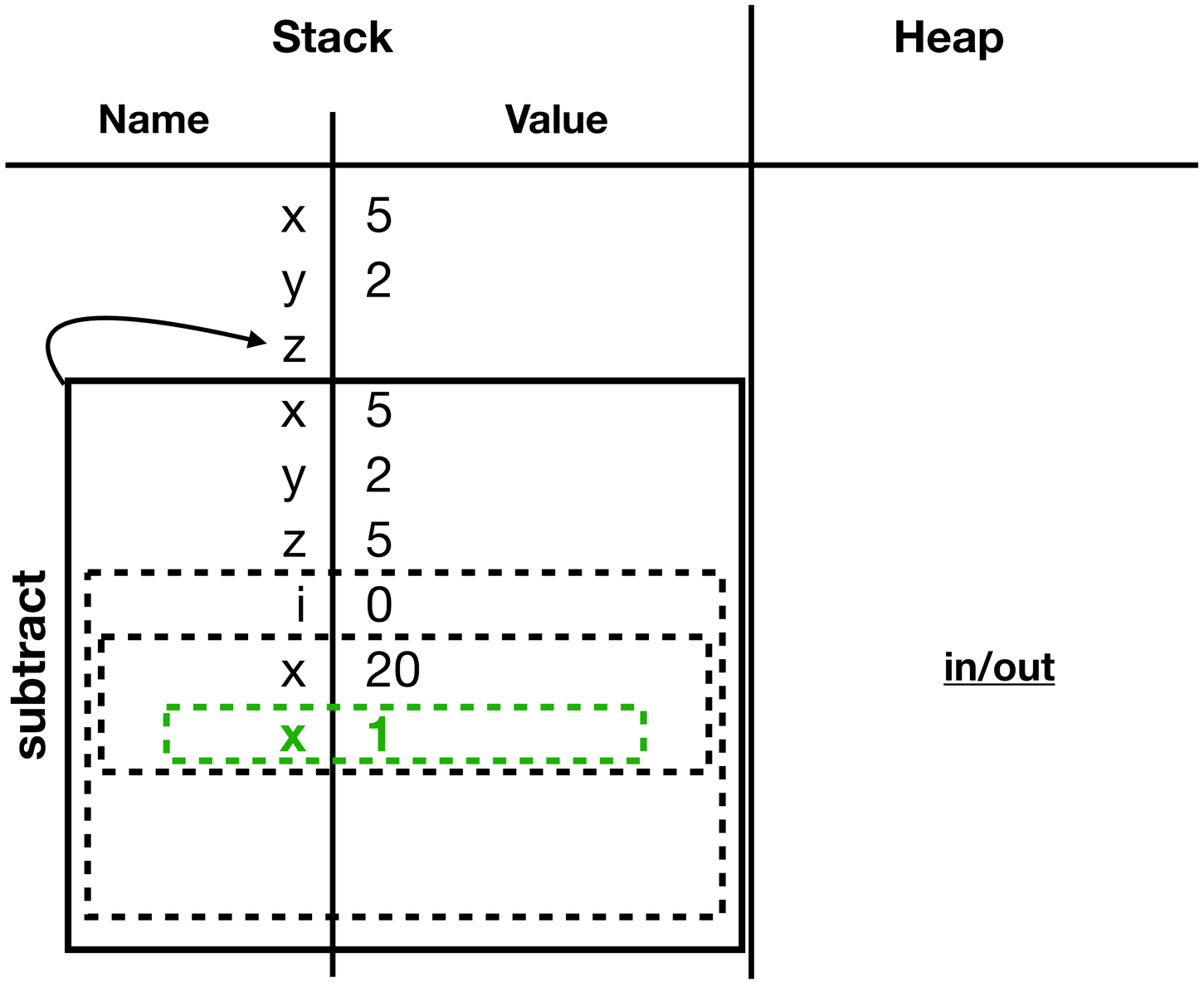
- z -= x
- But we have 4 x's on the stack!
- Which one is used??

More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

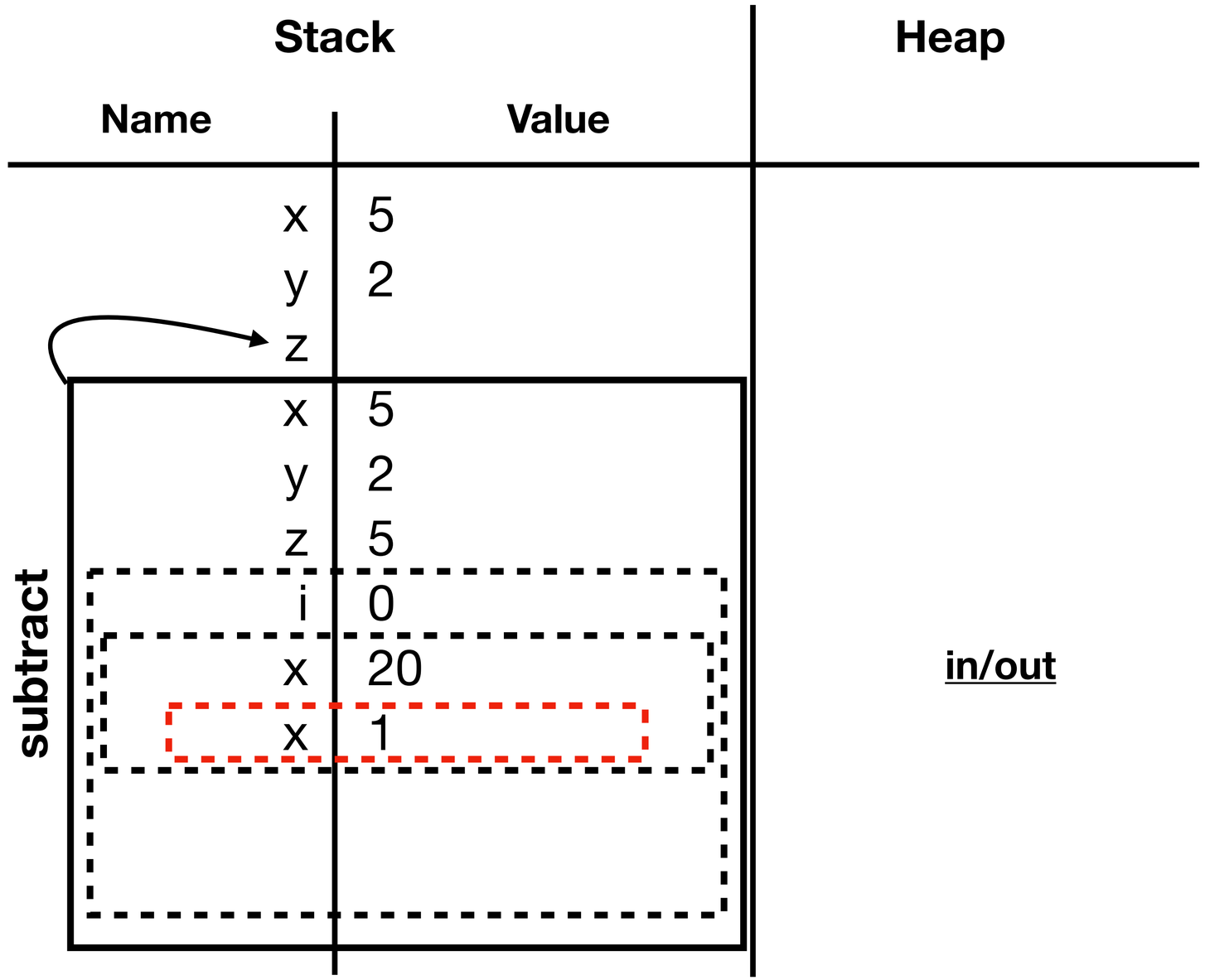
- Use the one in the inner-most code block
- The current code block has an x, use that one



More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

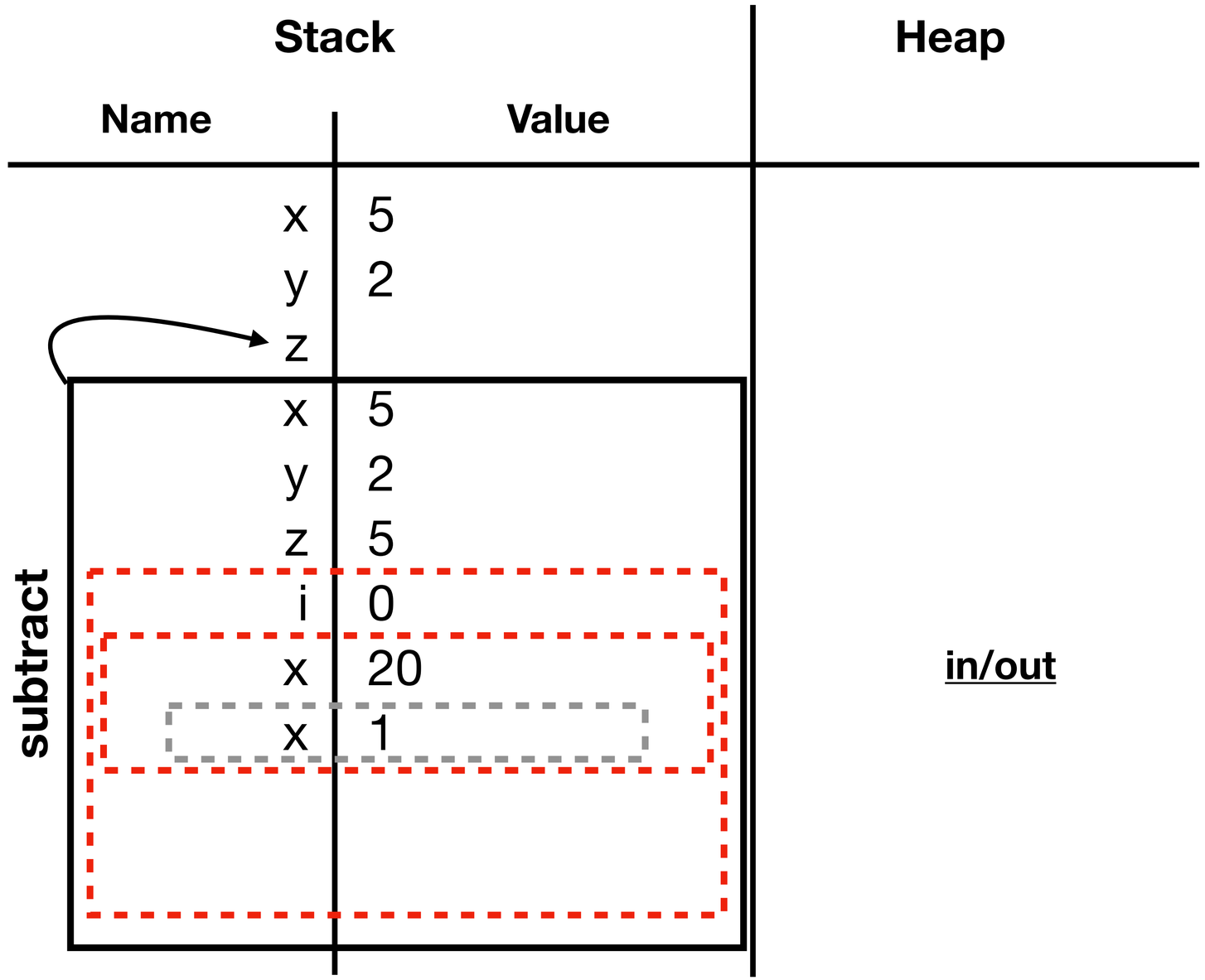


- We're also accessing z
- Current code block does not contain a z

More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

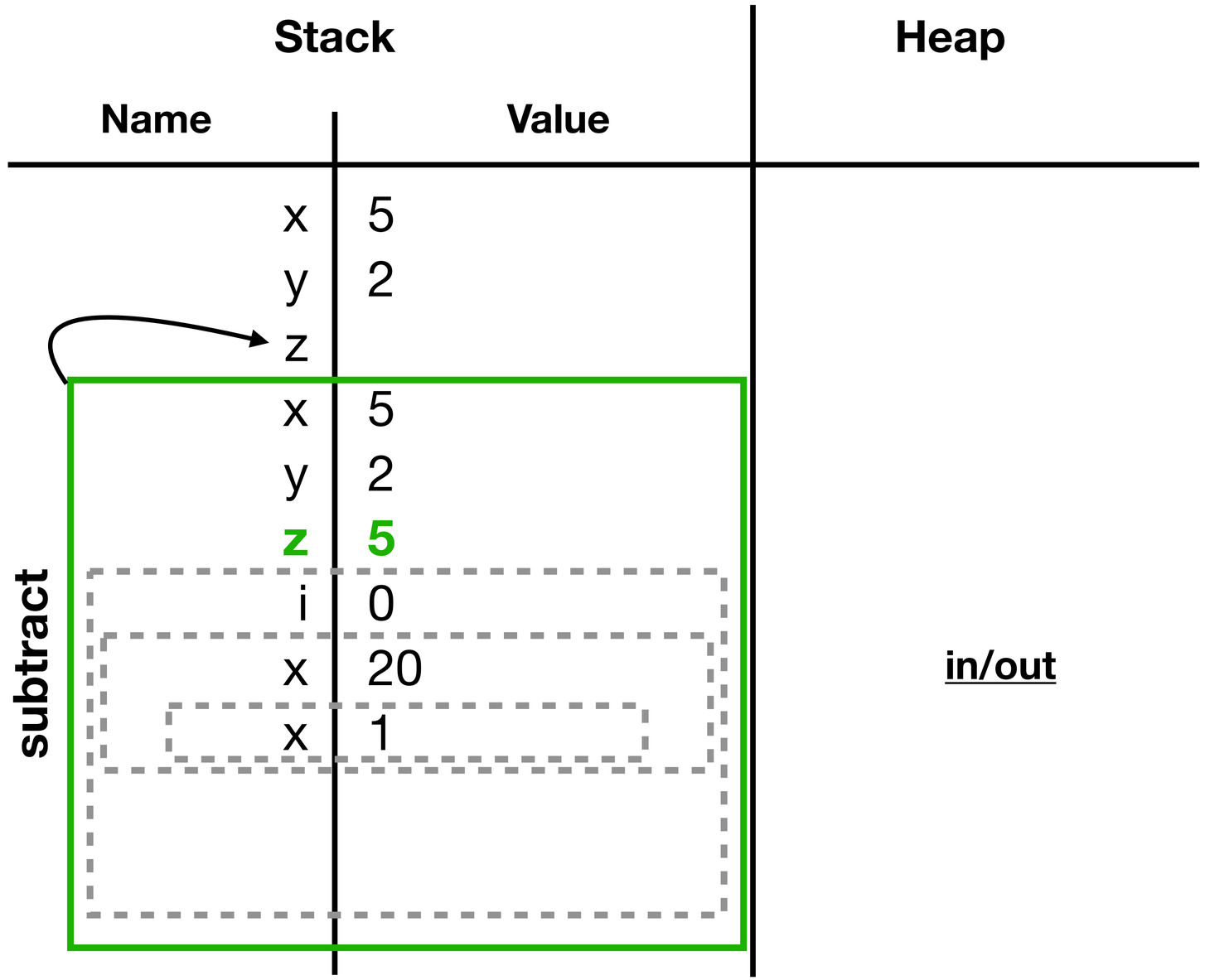


- Continue searching in the next code block
- Still no z...

More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```



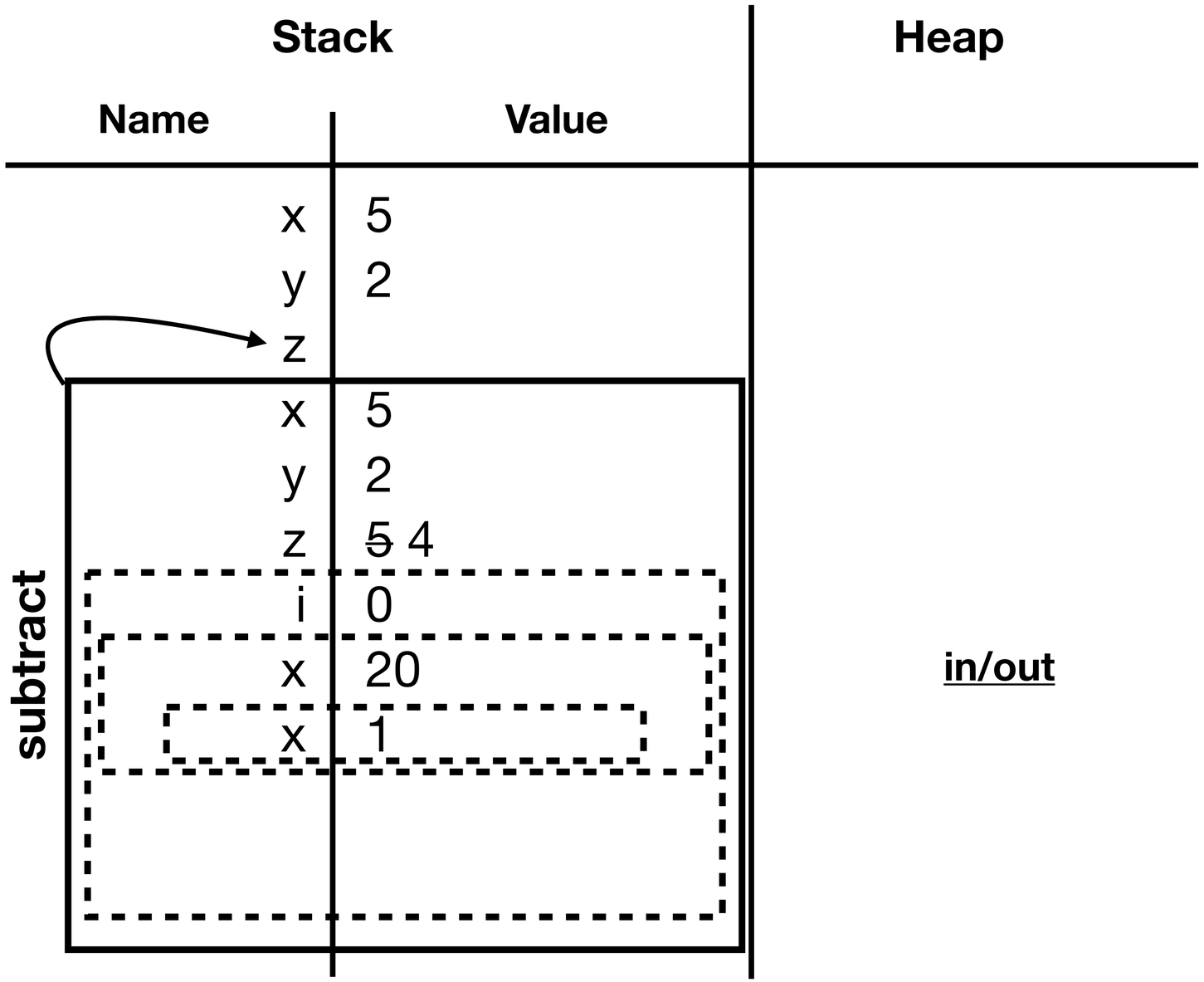
- Keep searching code blocks until we reach the stack frame
- Found a z! Use it.

More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

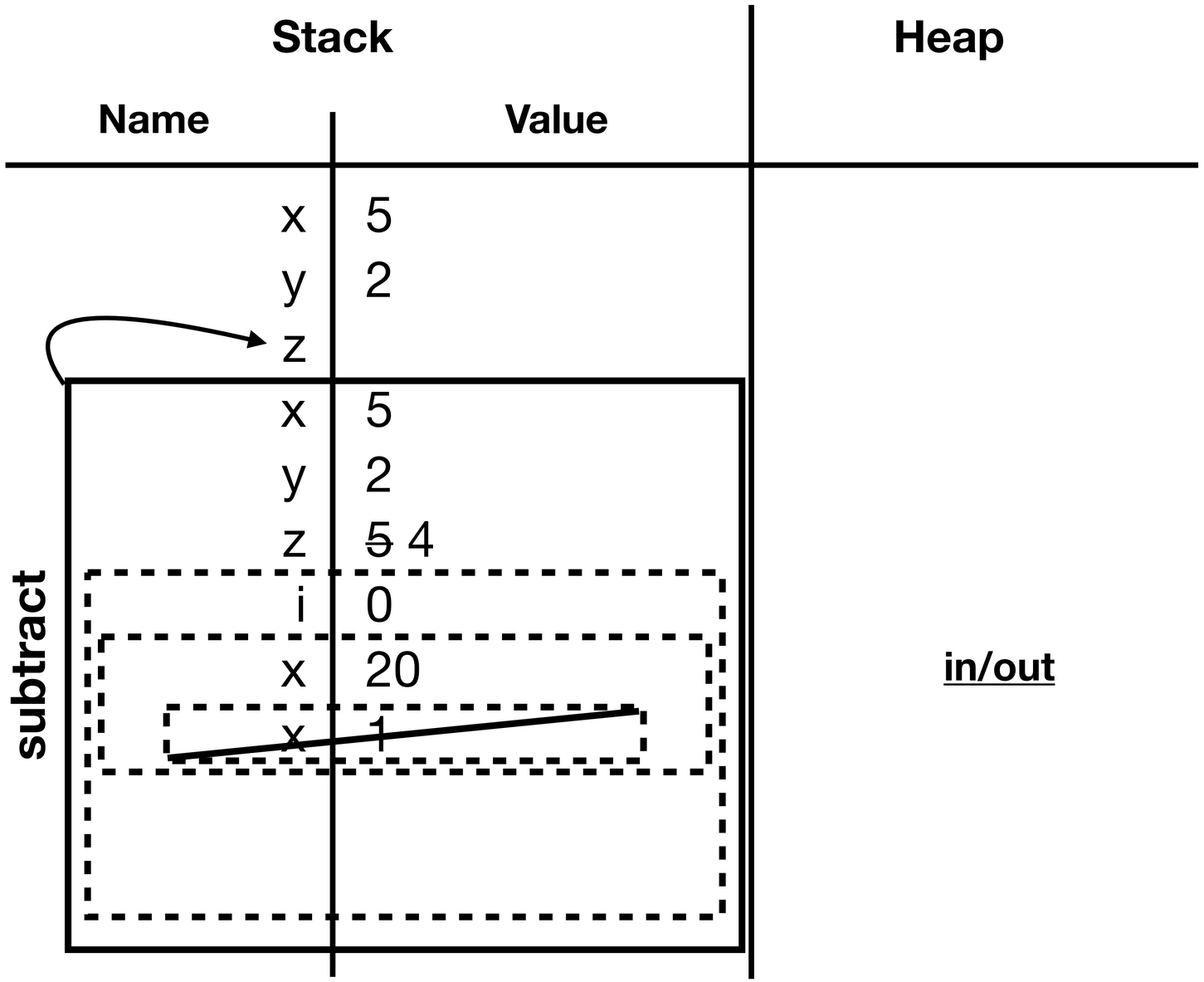
- Subtract 1 from z



More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```



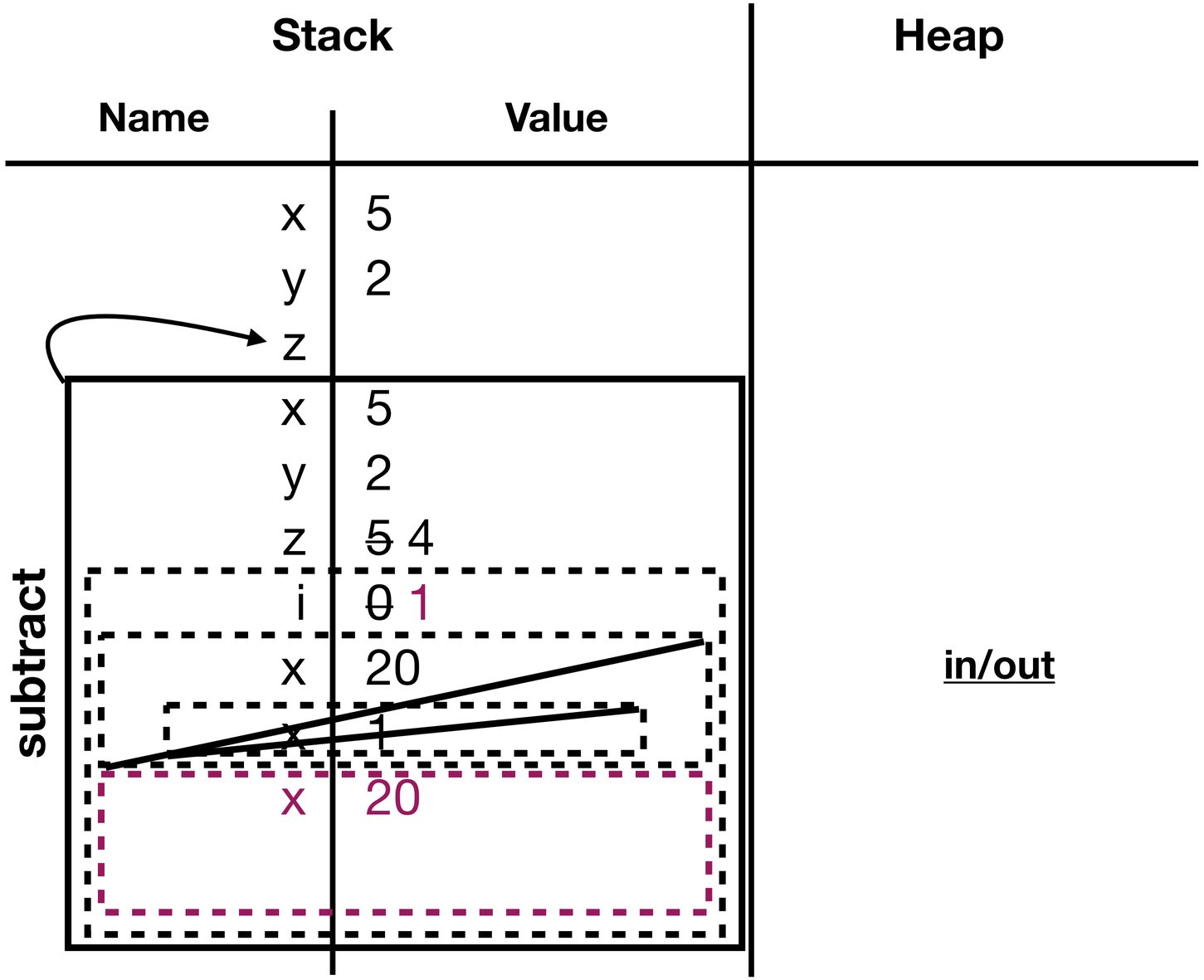
- End of the if/else code block
  - Cross it out
- The x with value 1 is no longer on the stack and the x with 20 is back in scope

More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

- This iteration of the loop ends
- Cross out the block and create a new block for the next one

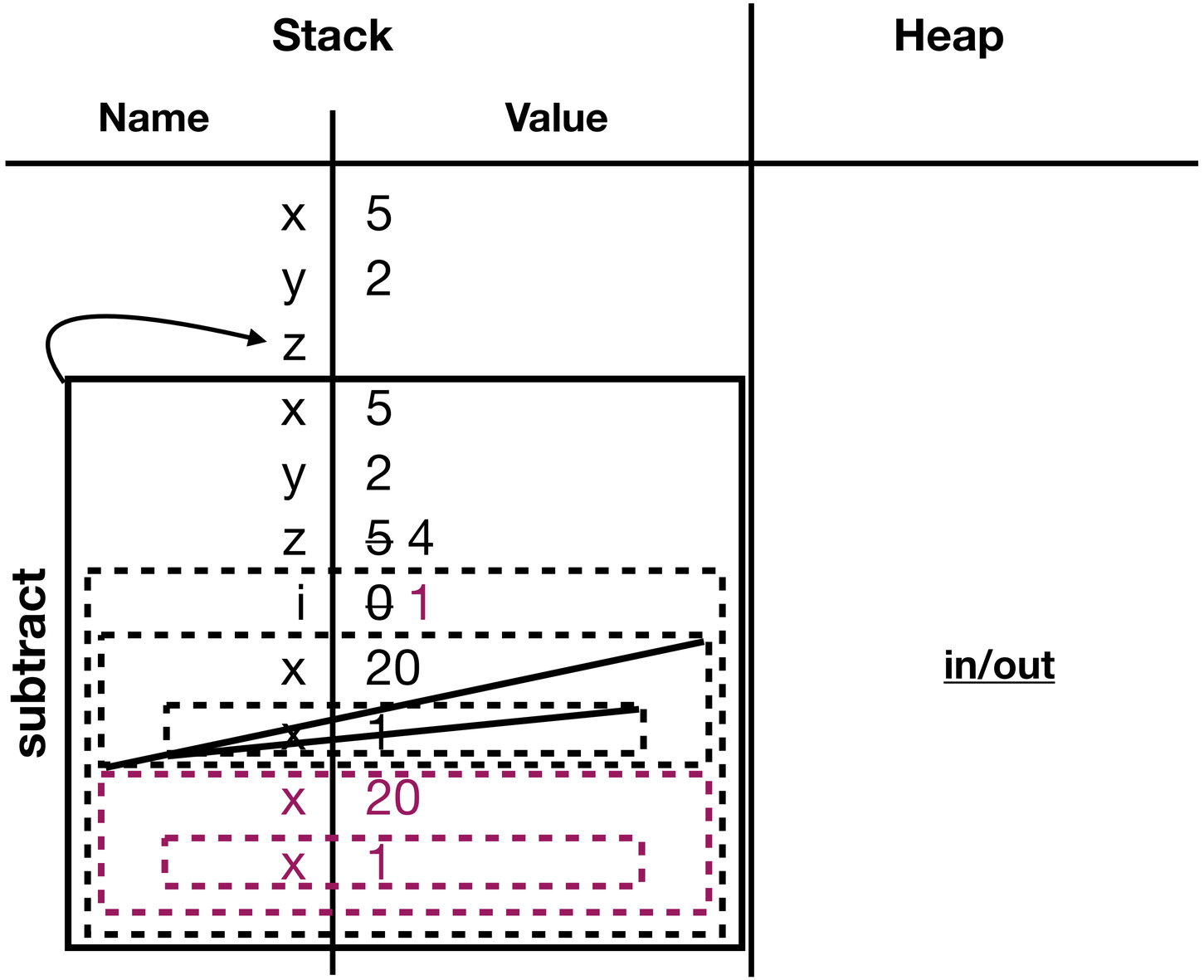


More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

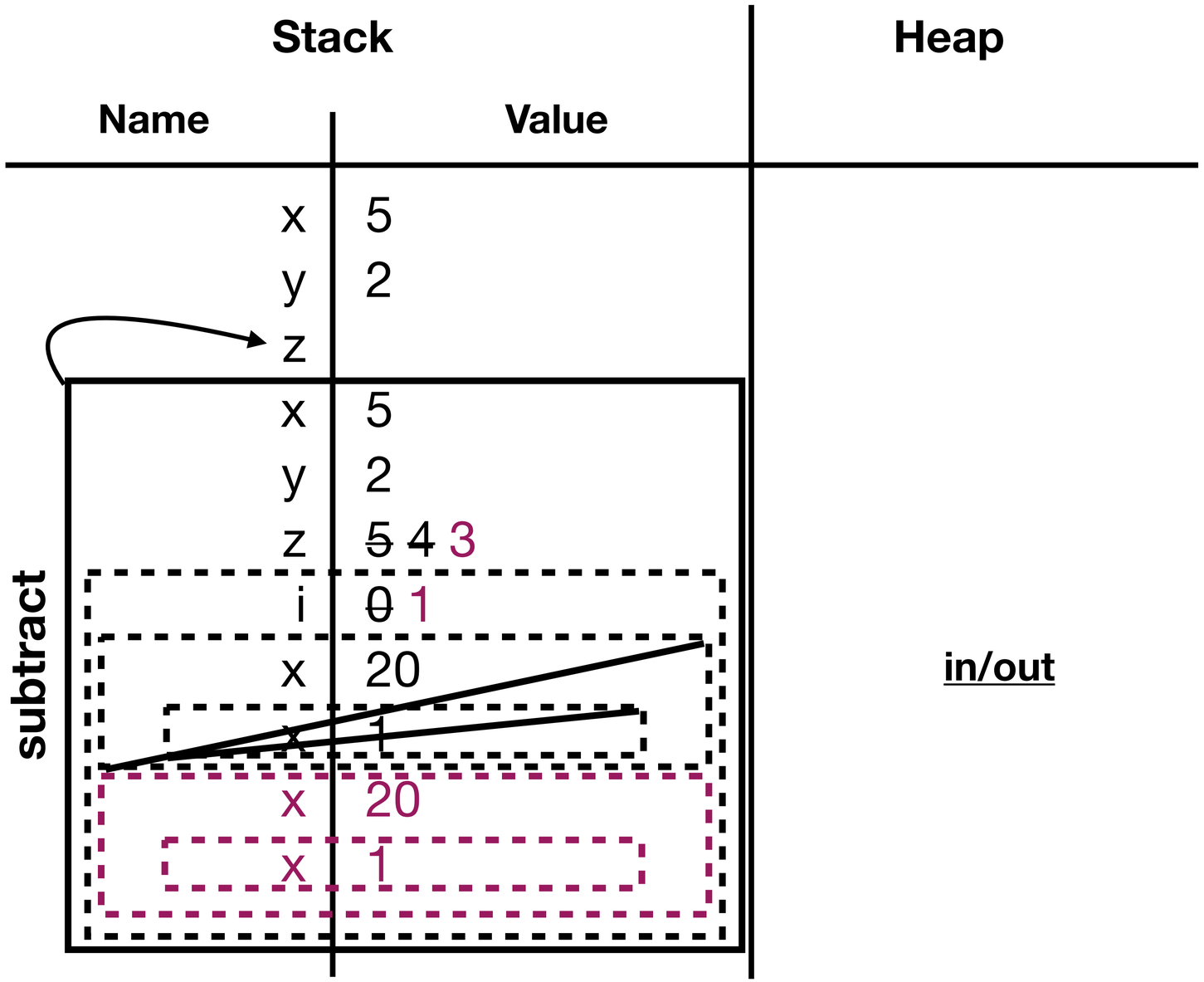
- Do the same thing for the next iteration



More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

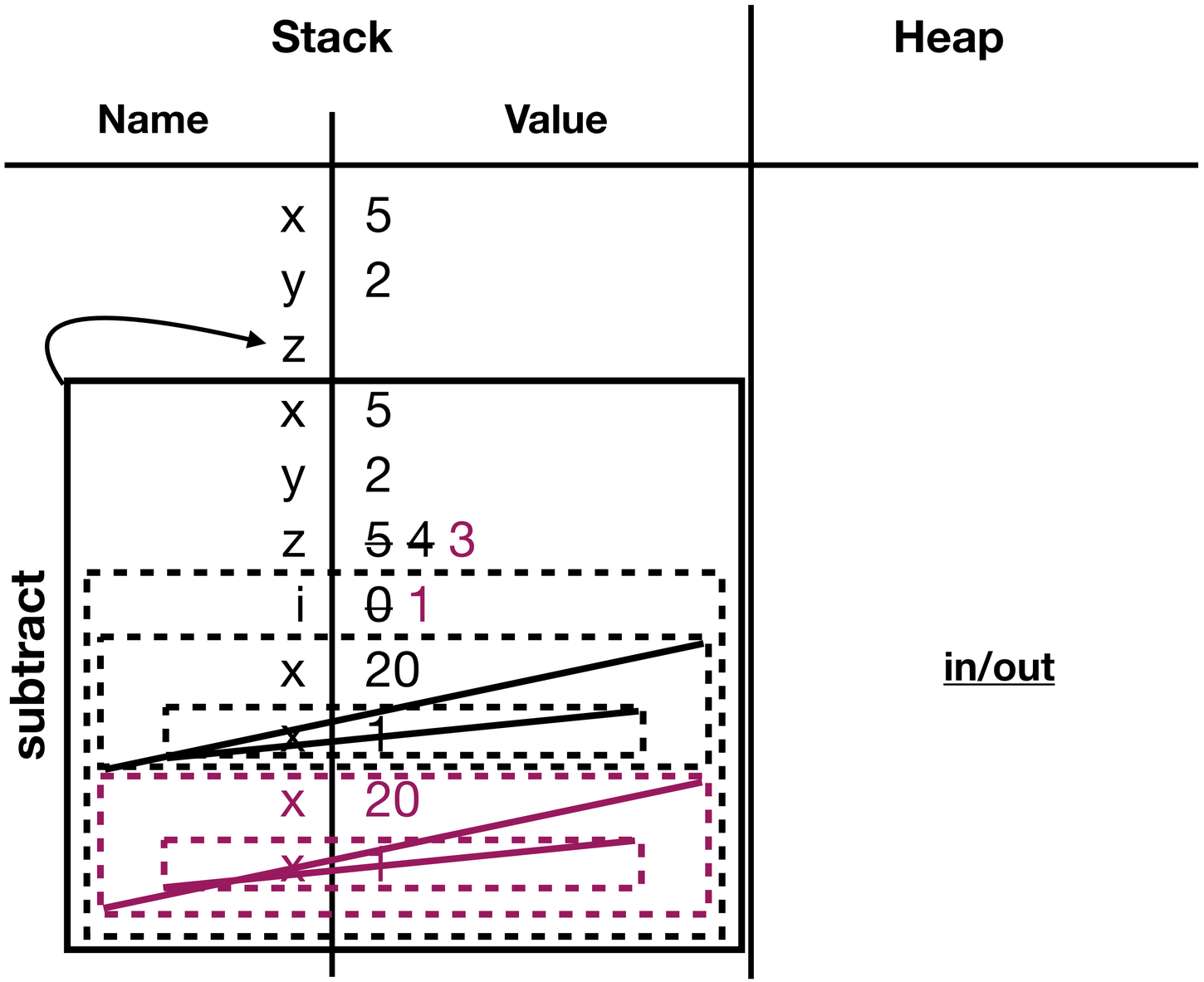


- Update z
- Found an x with value 1 in the inner-most block

More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

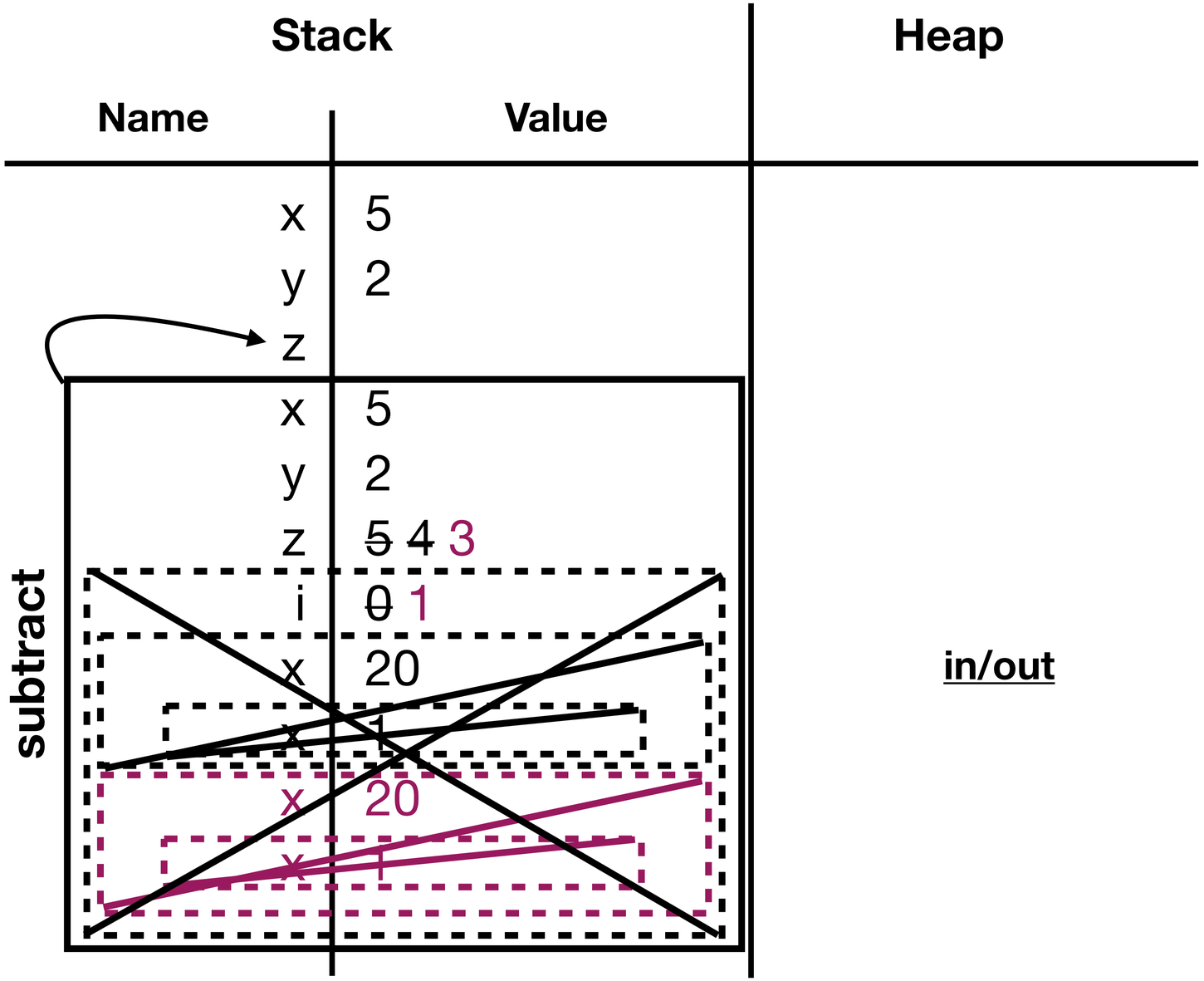


- End of the if/else block
- End of this iteration of the loop

More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

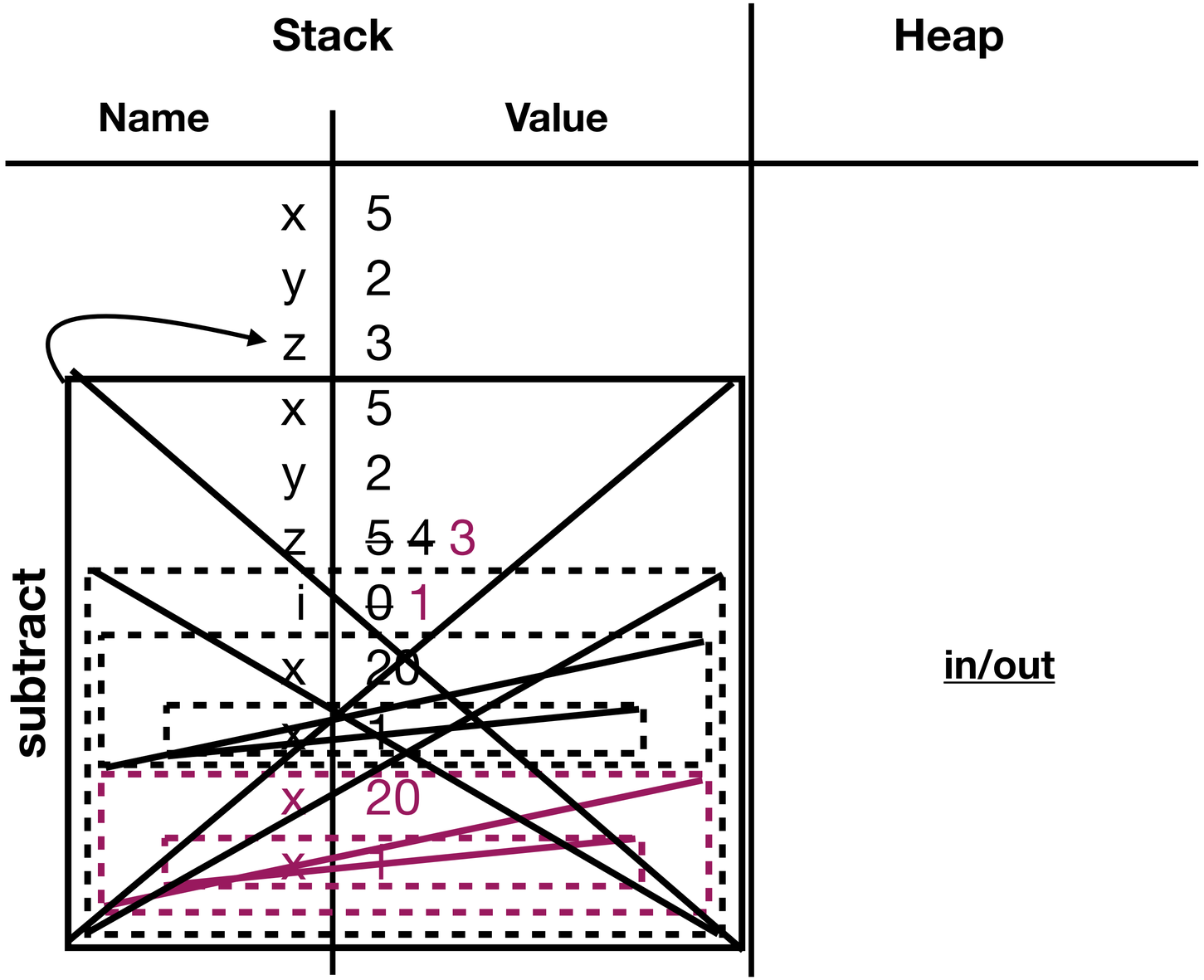


- End of the entire loop
  - i is no longer on the stack
  - x with value 5 is back in scope

More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```



- Return the value of z to the variable z in the main stack frame
- Cross out the entire stack frame

More Memory Examples

```
def subtract(x: Int, y: Int): Int = {
  var z: Int = x
  for (i <- 0 until Math.abs(y)) {
    val x: Int = 20
    if (y < 0) {
      val x: Int = 1
      z += x
    } else {
      val x: Int = 1
      z -= x
    }
  }
  z
}


def main(args: Array[String]): Unit = {
  val x: Int = 5
  val y: Int = 2
  val z: Int = subtract(x, y)
  println(z)
}
```

- Print 3 to the screen
- End of program