Model of Execution

Lecture Question

Question: In a package named "physics" create a Scala **class** named "PhysicsVector" with the following:

- A constructor that takes 3 **var**iables of type Double named "x", "y", and "z"
- A method named "multiplyByConstant" that takes a Double and returns Unit. This method multiplies x, y, and z by the input
    - Be sure to update the state variables of the object when this method is called
    - Example: If a vector with x, y, and z of (2.0, 0.0, -1.5) has multiplyByConstant(2.0) called on it, it's state will become (4.0, 0.0, -3.0)
- A method named "addVector" that takes a PhysicsVector and returns Unit. This method adds the values of x, y, and z of the input vector to the state variables of the calling vector
    - Example: If a vector with x, y, and z of (2.0, 0.0, -1.5) has addVector(otherVector) called on it where otherVector is (-3.5, 0.4, -1.0), it's state will become (-1.5, 0.4, -2.5)

Testing: In a package named "tests" create a Scala class named "TestVector" as a test suite that tests **all** the functionality listed above

Interpretation v. Compilation

- Interpretation
  - Code is read and executed one statement at a time
- Compilation
  - Entire program is translated into another language
  - The translated code is interpreted

Interpretation

- Python and JavaScript are interpreted languages
- Run-time errors are common
  - Program runs, but crashes when a line with an error is interpreted

This program runs without error

```python
class RuntimeErrorExample:

    def __init__(self, initial_state):
        self.state = initial_state

    def add_to_state(self, to_add):
        print("adding to state")
        self.state += to_add


if __name__ == '__main__':
    example_object = RuntimeErrorExample(5)
    example_object.add_to_state(10)
    print(example_object.state)
```

This program crashes with runtime error

```python
class RuntimeErrorExample:

    def __init__(self, initial_state):
        self.state = initial_state

    def add_to_state(self, to_add):
        print("adding to state")
        self.state += to_add


if __name__ == '__main__':
    example_object = RuntimeErrorExample(5)
    example_object.add_to_state("ten")
    print(example_object.state)
```

Compilation

- Scala, Java, C, and C++ are compiled languages
- Compiler errors are common
  - Compilers will check all syntax and types and alert us of any errors (Compiler error)
  - Program fails to be converted into the target language
  - Program never runs
  - The compiler can help us find errors before they become run-time errors

Compiles and runs without error

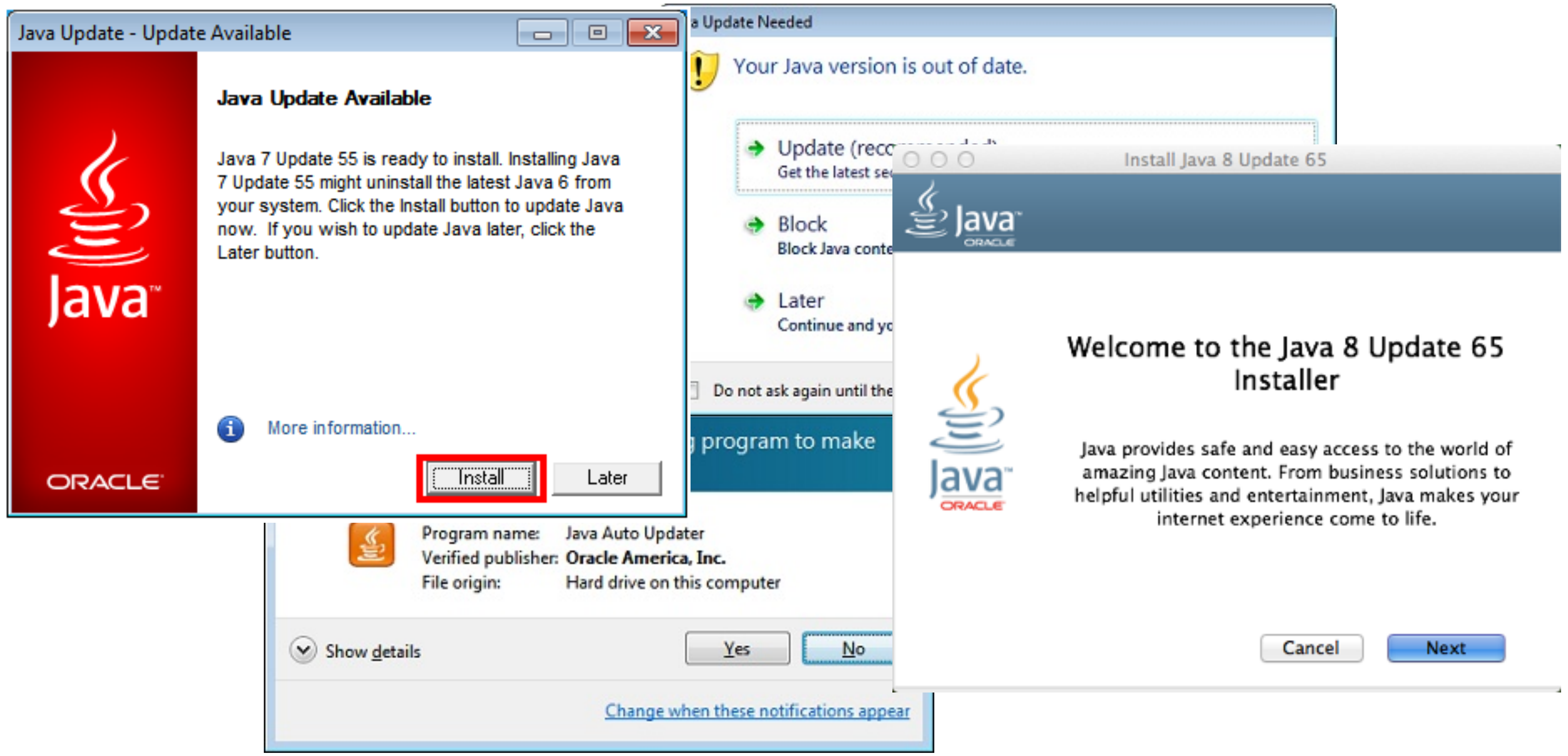
```
class CompilerError(var state: Int) {

  def addToState(toAdd: Int): Unit ={
    println("adding to state")
    this.state += toAdd
  }

}

object Main {
  def main(args: Array[String]): Unit = {
    val exampleObject = new CompilerError(5)
    exampleObject.addToState(10)
    println(exampleObject.state)
  }
}
```

Does not compile. Will not run any code

```
class CompilerError(var state: Int) {

  def addToState(toAdd: Int): Unit ={
    println("adding to state")
    this.state += toAdd
  }

}

object Main {
  def main(args: Array[String]): Unit = {
    val exampleObject = new CompilerError(5)
    exampleObject.addToState("ten")
    println(exampleObject.state)
  }
}
```

Compilation - Scala

- Scala compiles to Java Byte Code
- Executed by the Java Virtual Machine (JVM)
  - Installed on Billions of devices!



Java Update - Update Available

Java Update Available

Java 7 Update 55 is ready to install. Installing Java 7 Update 55 might uninstall the latest Java 6 from your system. Click the Install button to update Java now. If you wish to update Java later, click the Later button.

More information...

Install | Later

a Update Needed

Your Java version is out of date.

Update (reco
Get the latest se

Block
Block Java conte

Later
Continue and yo

Do not ask again until the

program to make

Program name: Java Auto Updater
Verified publisher: Oracle America, Inc.
File origin: Hard drive on this computer

Show details | Yes | No

Change when these notifications appear

Install Java 8 Update 65

Welcome to the Java 8 Update 65 Installer

Java provides safe and easy access to the world of amazing Java content. From business solutions to helpful utilities and entertainment, Java makes your internet experience come to life.

Cancel | Next

Compilation - Scala

- Compiled Java and Scala code can be used in the same program
  - Since they both compile to Java Byte Code
- Scala uses many Java classes
  - We saw that Math in Scala is Java's Math class
  - We'll sometimes use Java libraries in this course

References

- Every class you create will be passed by reference
  - Also data structure (List, Map, Array) and other built-in classes
- Pass-by-reference means that a copy is not made when a variable is assigned a value

References

```scala
object ItemReferences {

  def increasePrice(item: Item): Unit = {
    item.price += 0.25
  }

  def main(args: Array[String]): Unit = {

    val cereal: Item = new Item("cereal", 3.0)

    // pass-by-reference
    increasePrice(cereal)

    // assignment-by-reference
    val cereal2: Item = cereal

    increasePrice(cereal2)

    // 3.5
    println(cereal.price)
  }

}
```

- increasePrice returns Unit, yet it is able to modify an item
- cereal and cereal2 "refer" to the same object
  - Changes made to one will change both variables

References: Warning

```
def updateObject(dynamicObject: DynamicObject, deltaTime: Double, magnitudeOfGravity: Double): Unit = {
  dynamicObject.previousLocation = dynamicObject.location

  // ... rest of the method

}
```

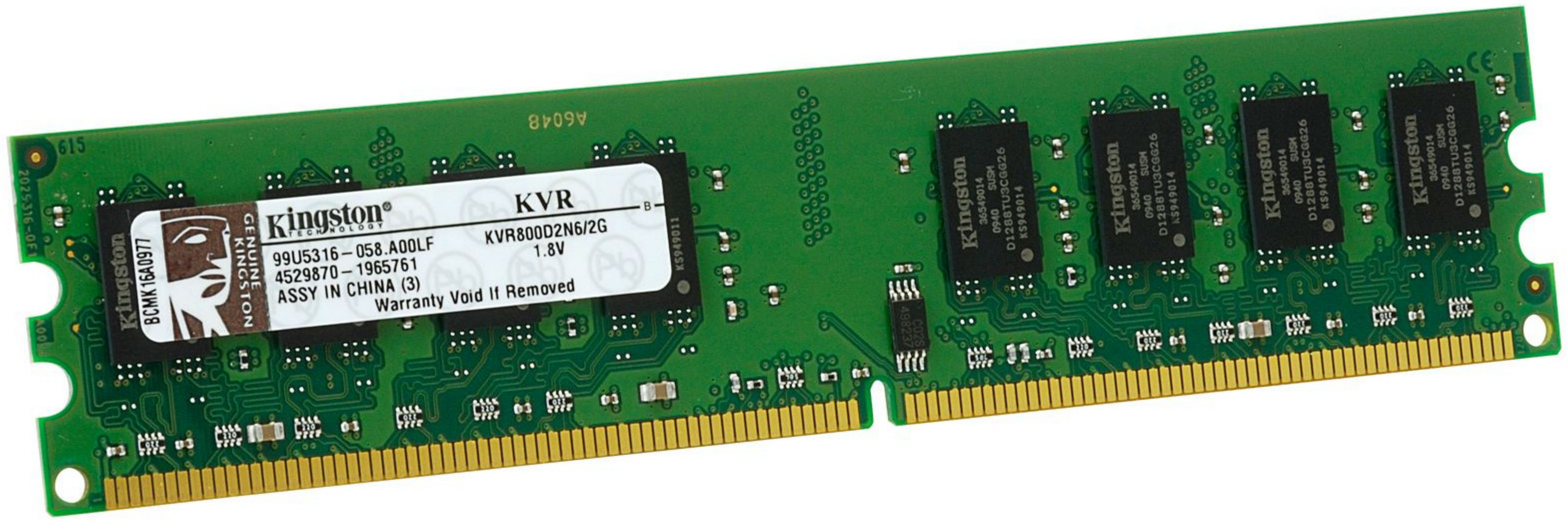
- previousLocation and location are the same object!!
  - Changing location will change previousLocation!

- Create a new PhysicsVector for previousLocation or copy x, y, z one at a time

Memory

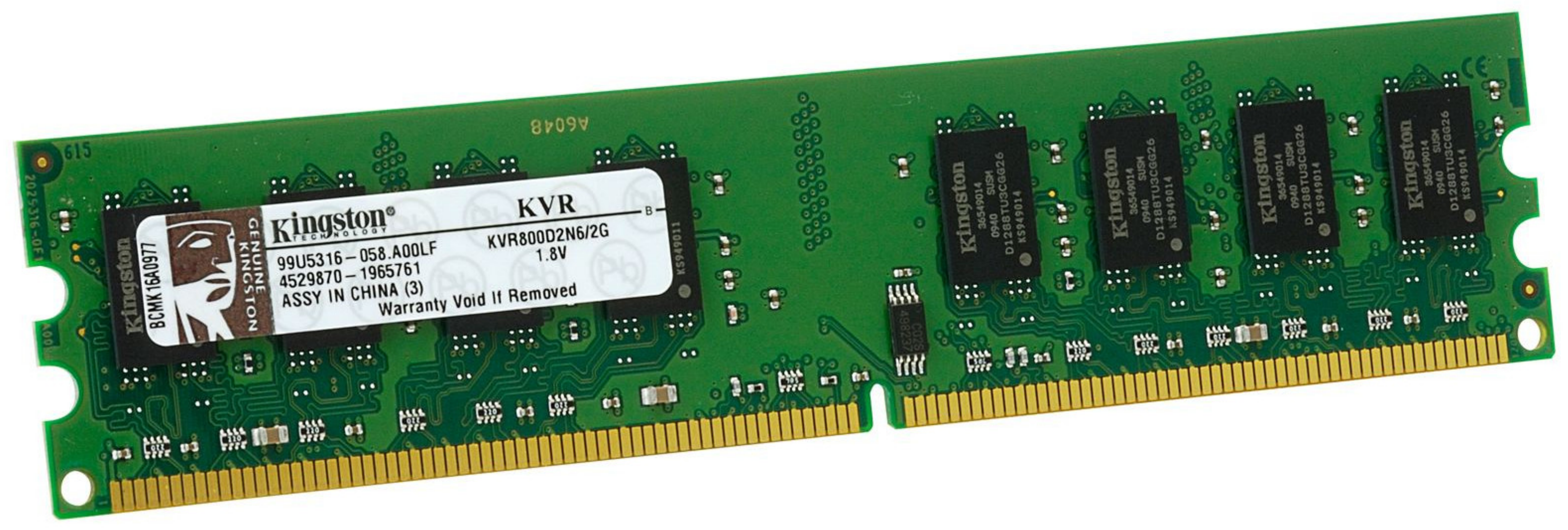
Let's Talk About Memory

- Random Access Memory (RAM)
  - Access any value by index
  - Effectively a giant array
- All values in your program are stored here



Let's Talk About Memory

- Significantly faster than reading/writing to disk
  - Even with an SSD
- Significantly more expensive than disk space



>

Let's Talk About Memory

- Operating System (OS) controls memory
- On program start, OS allocates a section of memory for our program
  - Gives access to a range of memory addresses/indices

RAM
...
<Used by another program>
<Our Program Memory>
<Our Program Memory>
<Our Program Memory>
<Our Program Memory>
<Our Program Memory>
<Our Program Memory>
<Our Program Memory>
<Our Program Memory>
<Our Program Memory>
<Our Program Memory>
<Our Program Memory>
<Used by another program>
...

Program Memory

- Some space is reserved for program data
- Details not important to CSE116
- The rest will be used for our data
- Data stored in the **memory stack**

RAM
...
<Used by another program>
<Command line args>
<Our Program Memory>
<Our Program Memory>
<Our Program Memory>
<Our Program Memory>
<Our Program Memory>
<Our Program Memory>
<Our Program Memory>
<Program Data>
<Program Data>
<Program Data>
<Used by another program>
...

Memory Stack

- Stores the variables and values for our programs
- LIFO - Last In First Out
    - New values are added to the end of the stack
    - Only values at the end of the stack can be removed

Memory Stack

- Method calls create new stack frames
  - Active stack frame is the currently executing method
  - Only stack values in the current stack frame can be accessed
  - A stack frame is isolated from the rest of the stack
- Program execution begins in the main method stack frame

Memory Stack

- Code blocks control variable scope
  - Code executing within a code block (ex. if, for, while) begins a new section on the stack
- Similar to stack frames, but values outside of the code block can be accessed
- Variables/Values in the same code block cannot have the same name
  - If variables in different blocks have the same name, the program searches the inner-most code block first for that variable
- When the end of a code block is reached, all variables/values created within that block are destroyed

Memory Stack Example

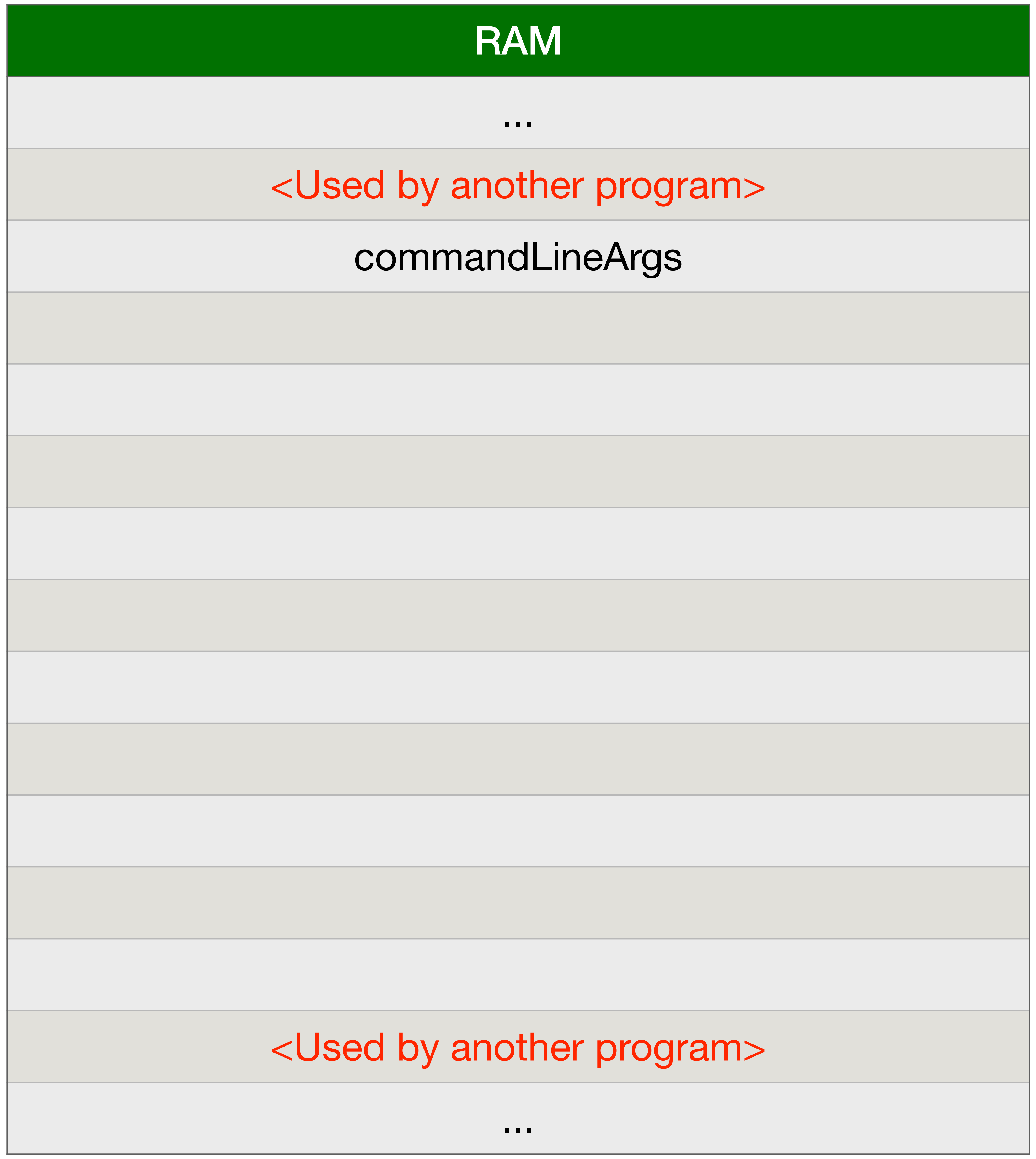
```
function computeFactorial(n){
  result = 1
  for (i=1; i<=n; i++) {
    result *= i
  }
  return result
}

function main(commandLineArgs){
  i = 5
  n = computeFactorial(i)
  print(n)
}
```

Note: This example is language independent and will focus on the concept of memory. Each language will have differences in how memory is managed

Memory Stack Example

```
function computeFactorial(n){
  result = 1
  for (i=1; i<=n; i++) {
    result *= i
  }
  return result
}

function main(commandLineArgs){
  i = 5
  n = computeFactorial(i)
  print(n)
}
```

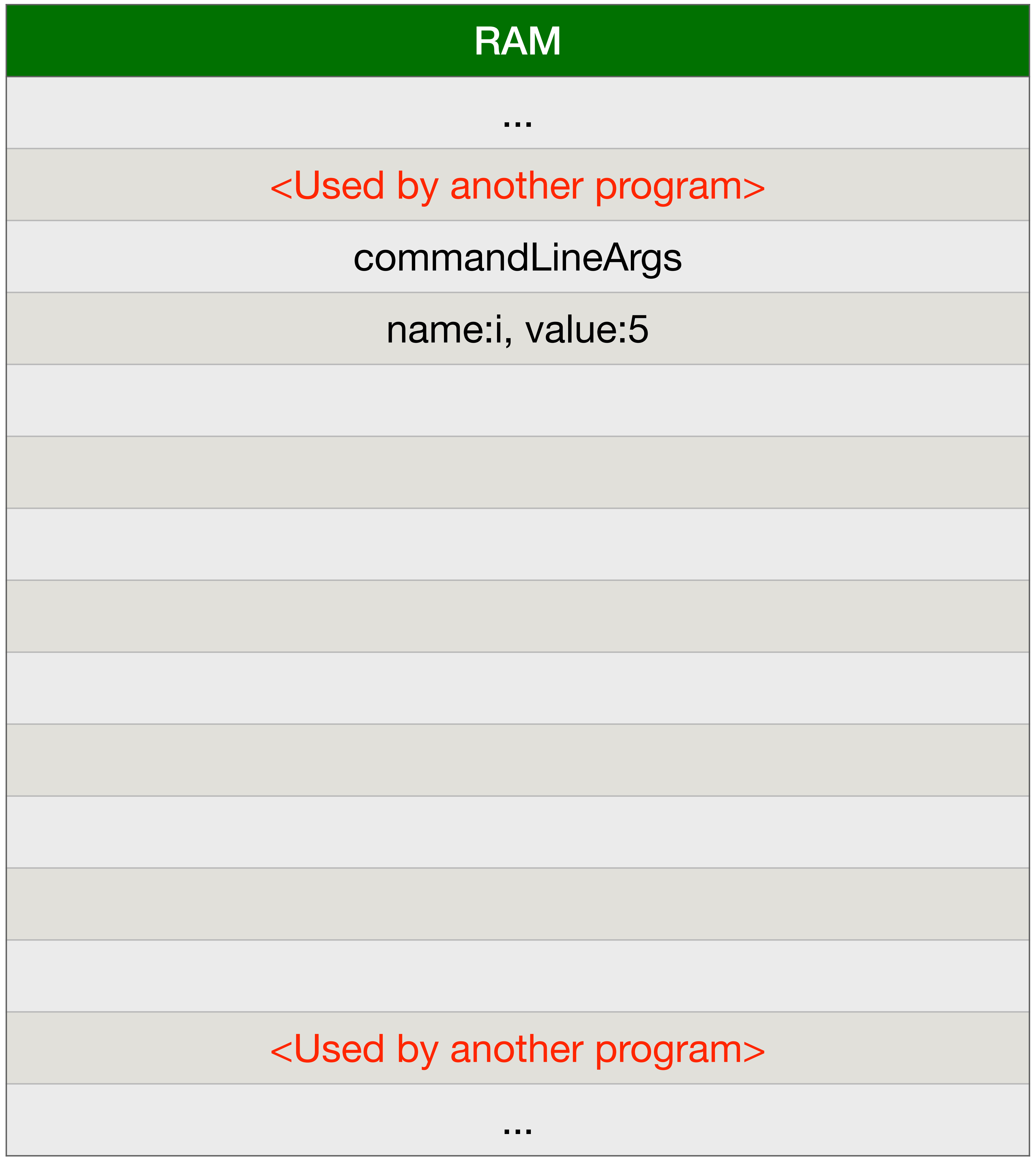
- Command line arguments added to the stack



Memory Stack Example

```
function computeFactorial(n){
  result = 1
  for (i=1; i<=n; i++) {
    result *= i
  }
  return result
}

function main(commandLineArgs){
  i = 5
  n = computeFactorial(i)
  print(n)
}
```

- A variable named i of type Int is added to the stack
- The variable i is assigned a value of 5



Memory Stack Example

```
function computeFactorial(n){
  result = 1
  for (i=1; i<=n; i++) {
    result *= i
  }
  return result
}

function main(commandLineArgs){
  i = 5
  n = computeFactorial(i)
  print(n)
}
```

- The program enters a call to computeFactorial
- A new **stack frame** is created for this call

RAM
...
<Used by another program>
commandLineArgs
name:i, value:5
<begin computeFactorial stack frame>
<Used by another program>
...

Memory Stack Example

```
function computeFactorial(n){
  result = 1
  for (i=1; i<=n; i++) {
    result *= i
  }
  return result
}

function main(commandLineArgs){
  i = 5
  n = computeFactorial(i)
  print(n)
}
```

- Add n to the stack and assign it the value from the input argument

RAM
...
<Used by another program>
commandLineArgs
name:i, value:5
<begin computeFactorial stack frame>
name:n, value:5
<Used by another program>
...

Memory Stack Example

```
function computeFactorial(n){
  result = 1
  for (i=1; i<=n; i++) {
    result *= i
  }
  return result
}

function main(commandLineArgs){
  i = 5
  n = computeFactorial(i)
  print(n)
}
```

- Add result to the stack and assign it the value 1

RAM
...
<Used by another program>
commandLineArgs
name:i, value:5
<begin computeFactorial stack frame>
name:n, value:5
name:result, value:1
<Used by another program>
...

Memory Stack Example

```
function computeFactorial(n){
  result = 1
→ for (i=1; i<=n; i++) {
    result *= i
  }
  return result
}

function main(commandLineArgs){
  i = 5
  n = computeFactorial(i)
  print(n)
}
```

- Begin loop block
- Add i to the stack and assign it the value 1
  - This is different from the i declared in main since they are in different frames

RAM
...
<Used by another program>
commandLineArgs
name:i, value:5
<begin computeFactorial stack frame>
name:n, value:5
name:result, value:1
<begin loop block>
name:i, value:1
<Used by another program>
...

Memory Stack Example

```
function computeFactorial(n){
  result = 1
  for (i=1; i<=n; i++) {
    result *= i
  }
  return result
}

function main(commandLineArgs){
  i = 5
  n = computeFactorial(i)
  print(n)
}
```

- Iterate through the loop
- look for variable named result in current stack frame
  - Found it outside the loop block
  - Update it's value (remains 1 on first iteration)

RAM
...
<Used by another program>
commandLineArgs
name:i, value:5
<begin computeFactorial stack frame>
name:n, value:5
name:result, value:1
<begin loop block>
name:i, value:1
<Used by another program>
...

Memory Stack Example

```
function computeFactorial(n){
  result = 1
  for (i=1; i<=n; i++) {
    result *= i
  }
  return result
}

function main(commandLineArgs){
  i = 5
  n = computeFactorial(i)
  print(n)
}
```

- Iterate through the loop
- look for variable named i in current stack frame
  - Found it inside the loop block
  - *Some languages look outside the current frame

RAM
...
<Used by another program>
commandLineArgs
name:i, value:5
<begin computeFactorial stack frame>
name:n, value:5
name:result, value:1
<begin loop block>
name:i, value:2
<Used by another program>
...

Memory Stack Example

```
function computeFactorial(n){
  result = 1
  for (i=1; i<=n; i++) {
→   result *= i
  }
  return result
}

function main(commandLineArgs){
  i = 5
  n = computeFactorial(i)
  print(n)
}
```

- Multiply result by i

RAM
...
<Used by another program>
commandLineArgs
name:i, value:5
<begin computeFactorial stack frame>
name:n, value:5
name:result, value:2
<begin loop block>
name:i, value:2
<Used by another program>
...

Memory Stack Example

```
function computeFactorial(n){
  result = 1
  for (i=1; i<=n; i++) {
    result *= i
  }
  return result
}

function main(commandLineArgs){
  i = 5
  n = computeFactorial(i)
  print(n)
}
```

- Iterate through the loop until conditional is false

RAM
...
<Used by another program>
commandLineArgs
name:i, value:5
<begin computeFactorial stack frame>
name:n, value:5
name:result, value:120
<begin loop block>
name:i, value:5
<Used by another program>
...

Memory Stack Example

```
function computeFactorial(n){
  result = 1
  for (i=1; i<=n; i++) {
    result *= i
  }
  return result
}

function main(commandLineArgs){
  i = 5
  n = computeFactorial(i)
  print(n)
}
```

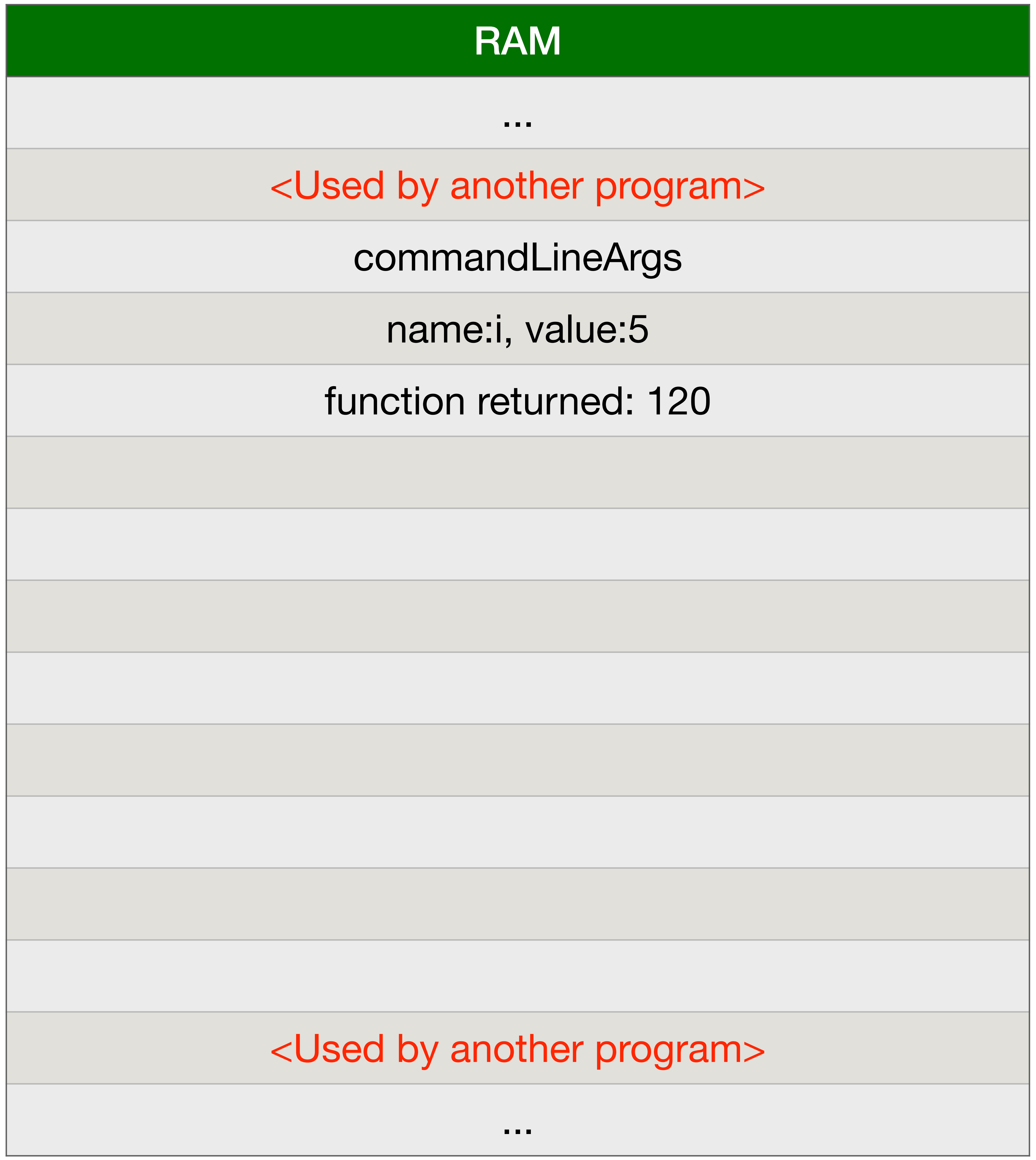
- End of a code block is reached
- Delete ALL stack storage used by that block!
  - The variable i fell out of scope and no longer exists

RAM
...
<Used by another program>
commandLineArgs
name:i, value:5
<begin computeFactorial stack frame>
name:n, value:5
name:result, value:120
<Used by another program>
...

Memory Stack Example

```
function computeFactorial(n){
  result = 1
  for (i=1; i<=n; i++) {
    result *= i
  }
  return result
}

function main(commandLineArgs){
  i = 5
  n = computeFactorial(i)
  print(n)
}
```

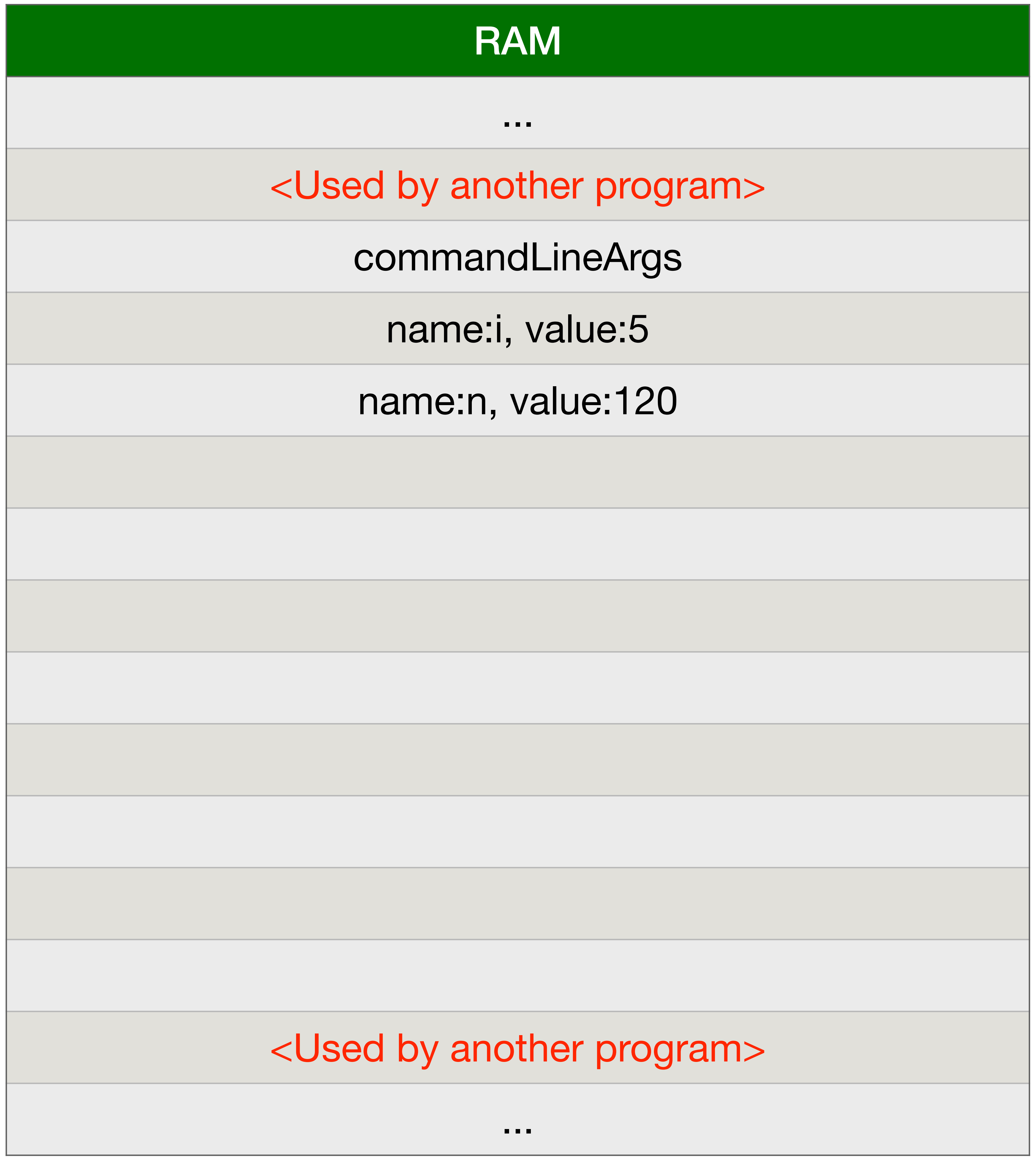
- End of a function is reached
- Delete ALL stack storage used by that stack frame!
- Replace function call with its return value



Memory Stack Example

```
function computeFactorial(n){
  result = 1
  for (i=1; i<=n; i++) {
    result *= i
  }
  return result
}

function main(commandLineArgs){
  i = 5
  n = computeFactorial(i)
  print(n)
}
```

- Declare n
- Assign return value to n



RAM
...
<Used by another program>
commandLineArgs
name:i, value:5
name:n, value:120
<Used by another program>
...

Memory Stack Example

```
function computeFactorial(n){
  result = 1
  for (i=1; i<=n; i++) {
    result *= i
  }
  return result
}

function main(commandLineArgs){
  i = 5
  n = computeFactorial(i)
  print(n)
}
```

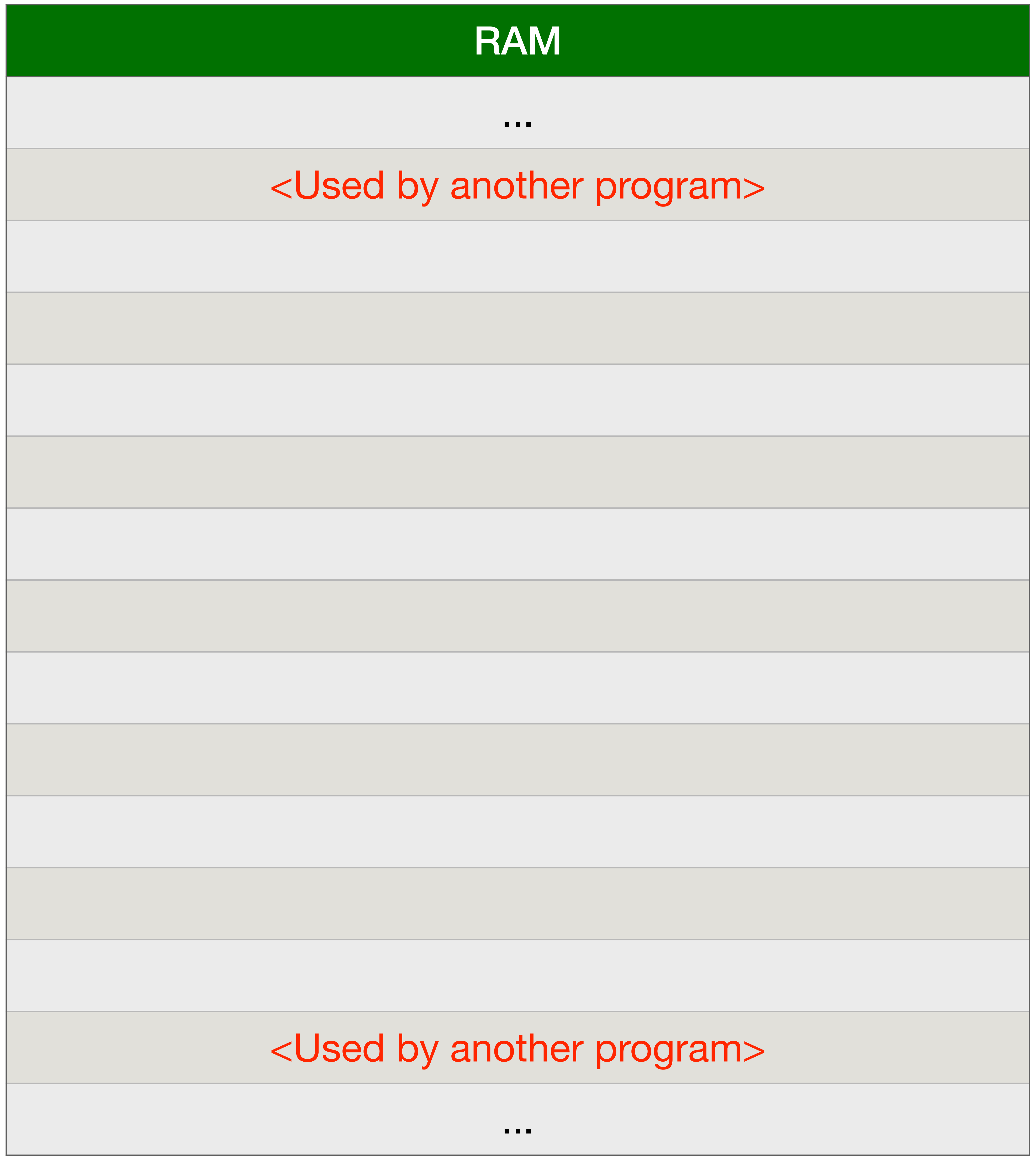
- Print n to the screen
- At this point:
  - No memory of variables n (function), i (function), or result

RAM
...
<Used by another program>
commandLineArgs
name:i, value:5
name:n, value:120
<Used by another program>
...

Memory Stack Example

```
function computeFactorial(n){
  result = 1
  for (i=1; i<=n; i++) {
    result *= i
  }
  return result
}

function main(commandLineArgs){
  i = 5
  n = computeFactorial(i)
  print(n)
}
```

- End of program
- Free memory back to the OS



Memory Stack Example

```
function computeFactorial(n){
  result = 1
  for (i=1; i<=n; i++) {
    result *= i
  }
  return result
}

function main(commandLineArgs){
  i = 5
  n = computeFactorial(i)
  print(n)
}
```

- No memory of our program

RAM
…
<Used by another program>
<Used by another program>
…

Stack Memory

- Only "primitive" types are stored in stack memory
  - Double/Float
  - Int/Long/Short
  - Char
  - Byte
  - Boolean

- Values corresponding to Java primitives
  - Compiler converts these objects to primitives in Java Byte Code

Stack Memory

- Only "primitive" types are stored in stack memory
  - Double/Float
  - Int/Long/Short
  - Char
  - Byte
  - Boolean

- All other objects are stored in heap memory*

***Stack and heap allocations vary by compiler and JVM implementations. With modern optimizations, we can never be sure where our values will be stored We'll use this simplified view so we can move on and learn Computer Science**

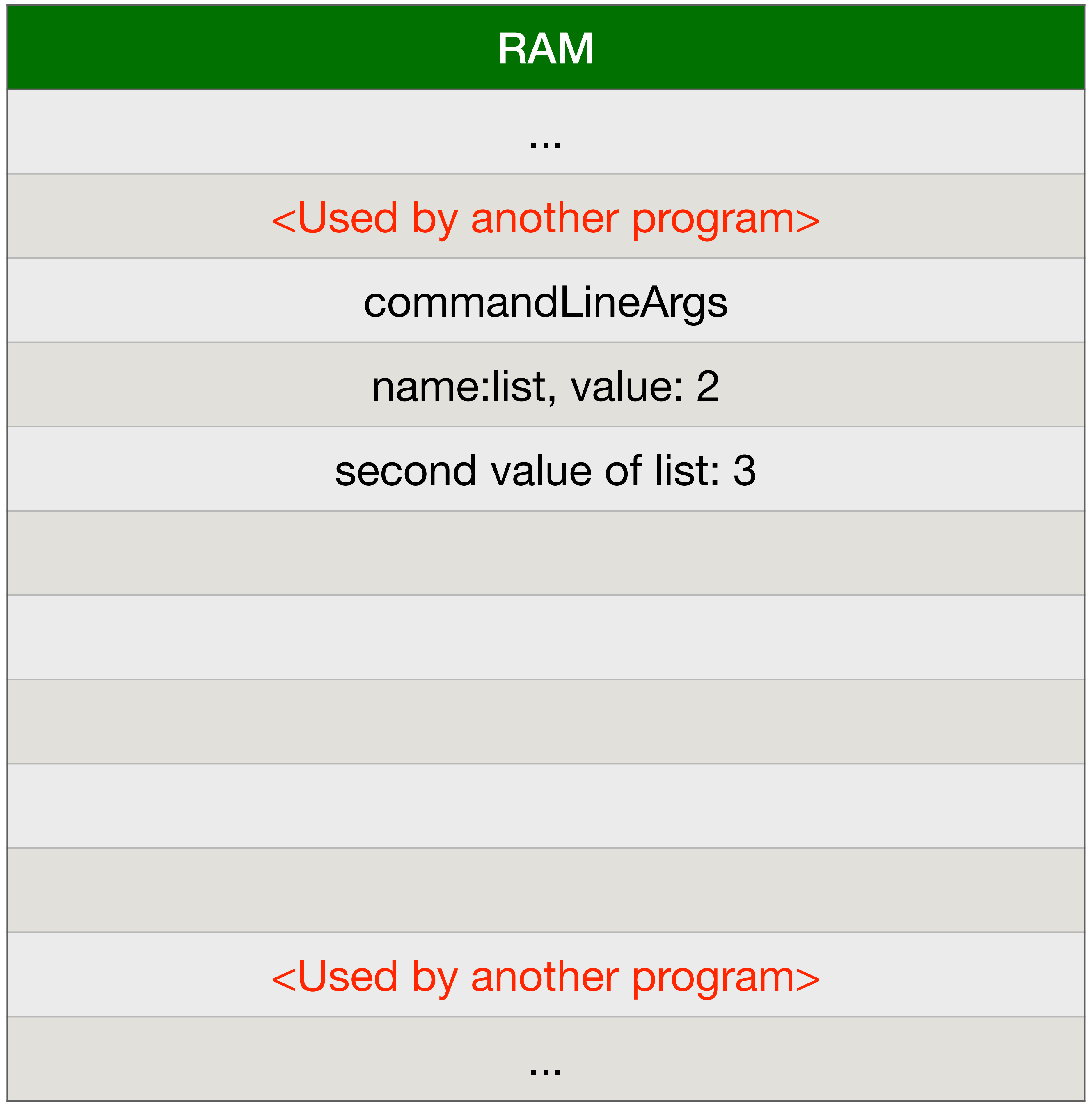
Memory Heap

The stack is very structured

What if we want more dynamic memory?

```
def main(args: Array[String]): Unit = {
  var list: List[Int] = List(2, 3)
  val x = 5
  val y = 12
  list = 1 :: list
}
```

Memory Heap Example



- Add the list to the stack
- The list has 2 elements
  - Allocate space for 2 Ints

```scala
def main(args: Array[String]): Unit = {
  var list: List[Int] = List(2, 3)
  val x = 5
  val y = 12
  list = 1 :: list
}
```

Memory Heap Example

RAM
...
<Used by another program>
commandLineArgs
name:list, value: 2
second value of list: 3
name:x, value: 5
name:y, value: 12
<Used by another program>
...

- Add more values to the stack
- Create a variable named "x" of type Int and assign it the value 5
- Create a variable named "y" of type Int and assign it the value 12
- Both values go on the stack in the order they are declared

```scala
def main(args: Array[String]): Unit = {
  var list: List[Int] = List(2, 3)
  val x = 5
  val y = 12
  list = 1 :: list
}
```

Memory Heap Example

RAM
...
<Used by another program>
commandLineArgs
name:list, value: 2
second value of list: 3
name:x, value: 5
name:y, value: 12
<Used by another program>
...

- Create a new list with values
  - 1, 2, 3
- Reassign the variable "list" to this new list
  - **But how??**

```
def main(args: Array[String]): Unit = {
  var list: List[Int] = List(2, 3)
  val x = 5
  val y = 12
  list = 1 :: list
}
```

Memory Heap Example

RAM
...
<Used by another program>
commandLineArgs
name:list, value: 1
second value of list: 2
[third value of list: 3] --- name:x, value: 5
name:y, value: 12
<Used by another program>
...

Option 1

- Expand the the list to contain the new element
- Conflicts with value "x"
- Could move all values down the stack
  - **Too slow**

```scala
def main(args: Array[String]): Unit = {
  var list: List[Int] = List(2, 3)
  val x = 5
  val y = 12
  list = 1 :: list
}
```

Memory Heap Example

RAM
...
<Used by another program>
commandLineArgs
name:x, value: 5
name:y, value: 12
name:list, value: 1
second value of list: 2
third value of list: 3
<Used by another program>
...

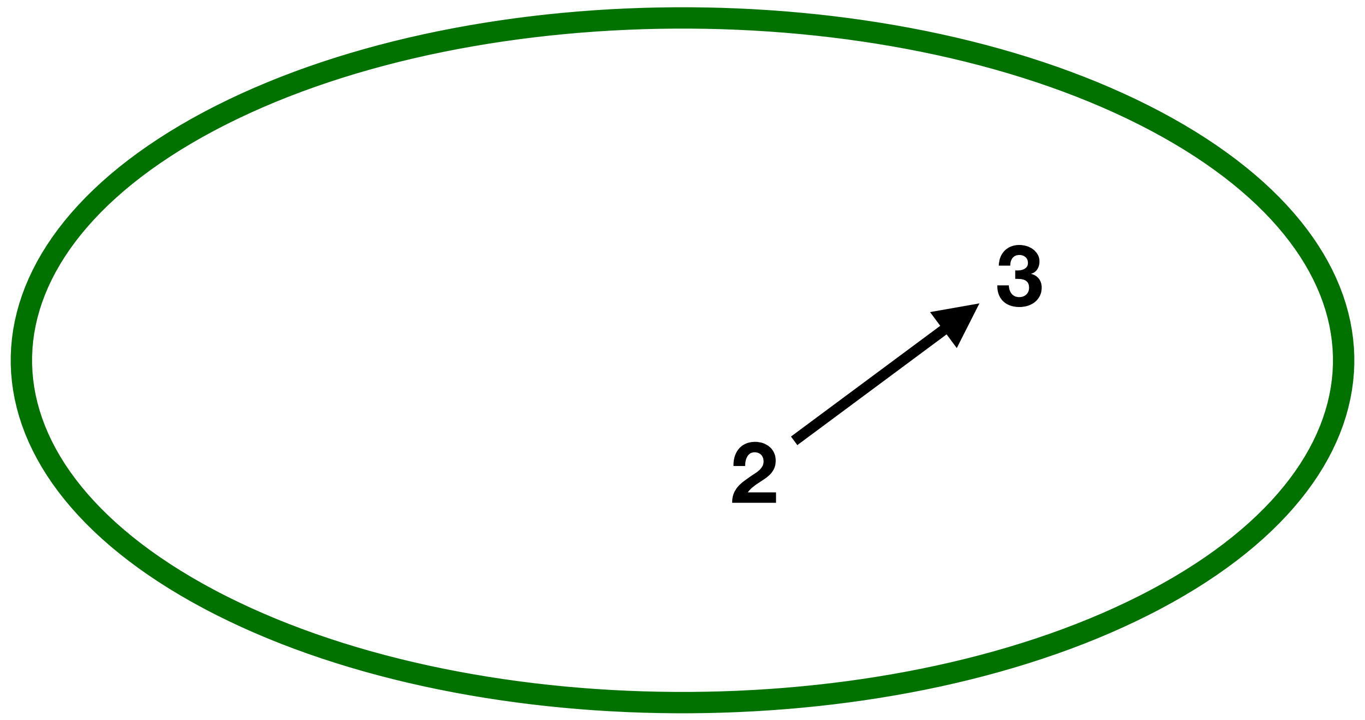
Option 2

- Move "list" to the bottom of the stack
  - Copy all values
  - Delete the older copy to avoid two "list" variables in the same stack frame
- **Too slow to copy entire list**
- **Leaves a gap in the stack**

```scala
def main(args: Array[String]): Unit = {
  var list: List[Int] = List(2, 3)
  val x = 5
  val y = 12
  list = 1 :: list
}
```

Memory Heap Example

RAM
...
<Used by another program>
commandLineArgs
name:list, value: [location of the value 2 in heap]
<Used by another program>
...



Option 3

- Allocate objects in heap memory
- When the list is created
  - Use memory that is not part of the stack (the heap) to store its values
  - Store the location, in memory, of these values in the "list" variable

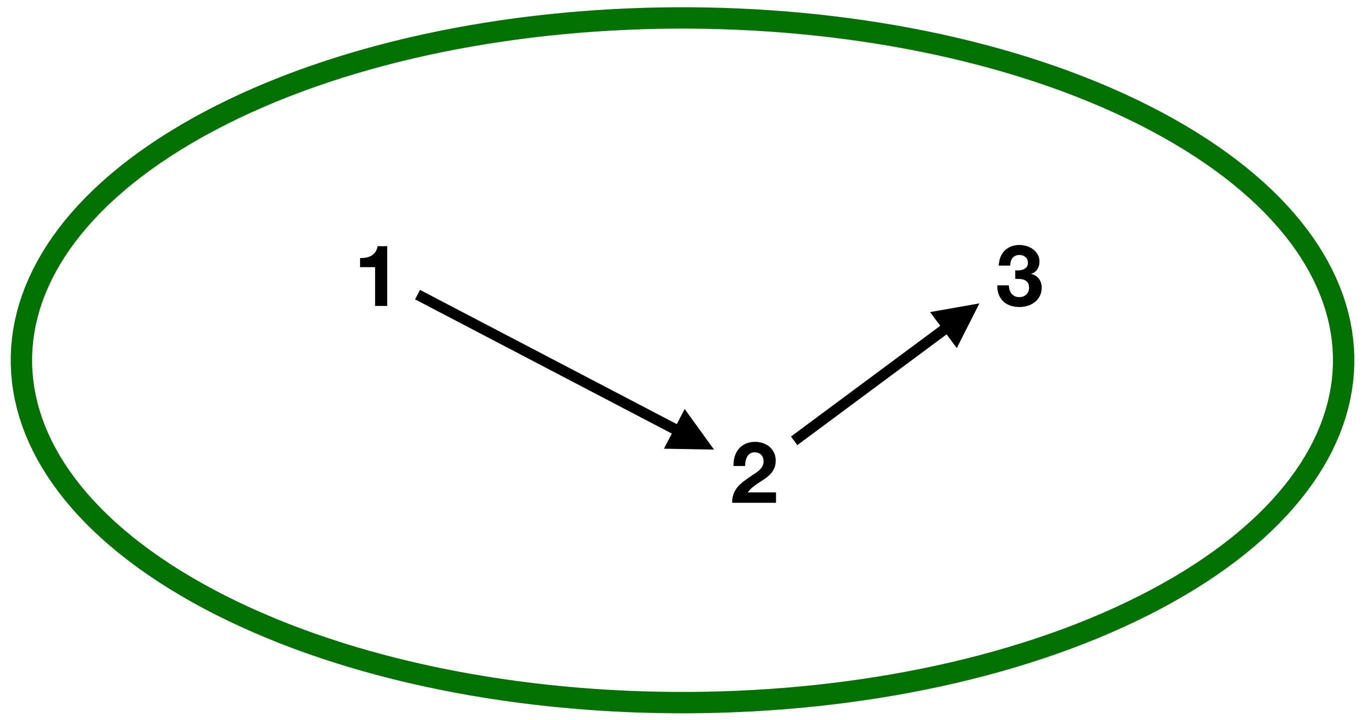
```
def main(args: Array[String]): Unit = {
  var list: List[Int] = List(2, 3)
  val x = 5
  val y = 12
  list = 1 :: list
}
```

Memory Heap Example

RAM
...
<Used by another program>
commandLineArgs
name:list, value: [location of the value 1 in heap]
name:x, value: 5
name:y, value: 12
<Used by another program>
...

Option 3

- Create a new List of Ints by adding the value 1 in heap space and have it "point" to the location of the value 2
- Reassign "list" to store the location (or reference) of the value 1



```
def main(args: Array[String]): Unit = {
  var list: List[Int] = List(2, 3)
  val x = 5
  val y = 12
  list = 1 :: list
}
```

Memory Heap

- Heap memory is dynamic
- Can be anywhere in RAM
  - Location is not important
  - Location can change
- Use references to find data
  - **Variables only store references to objects**

Memory Heap

Another language agnostic example

- Create an object on the heap using a class
- Modify the state of the object in a function
  - The object is passed by reference

```
class ClassWithState{
  int stateVar = 0;
}
```

```
function addToState(input){
  input.stateVar += 1
}

function main{
  data = new ClassWithState()
  addToState(data)
  println(data.stateVar)
}
```

Memory Heap Example

RAM
...
<Used by another program>
commandLineArgs
name:data, value: @387 (main)
<Used by another program>
...

RAM @387
...
ClassWithStateObject
-stateVar value:0
...

```
class ClassWithState{
  int stateVar = 0;
}
```

```
function addToState(input){
  input.stateVar += 1
}

function main{
  data = new ClassWithState()
  addToState(data)
  println(data.stateVar)
}
```

- Create instance of ClassWithState on the heap
- Store **memory address** of the new object in data

Memory Heap Example

RAM
...
<Used by another program>
commandLineArgs
name:data, value: @387 (main)
<addToState stack frame>
name:input, value: @387 (addToState)
<Used by another program>
...

RAM @387
...
ClassWithStateObject
-stateVar value:0
...

```
class ClassWithState{
  int stateVar = 0;
}
```

```
function addToState(input){
  input.stateVar += 1
}

function main{
  data = new ClassWithState()
  addToState(data)
  println(data.stateVar)
}
```

- Create a stack frame for the function call
- input is assigned the value in data
  - Which is a **memory address**

Memory Heap Example

RAM
...
<Used by another program>
commandLineArgs
name:data, value: @387 (main)
<addToState call stack frame>
name:input, value: @387 (addToState)
<Used by another program>
...

RAM @387
...
ClassWithStateObject
-stateVar value:1
...

```
class ClassWithState{
  int stateVar = 0;
}
```

```
function addToState(input){
  input.stateVar += 1
}

function main{
  data = new ClassWithState()
  addToState(data)
  println(data.stateVar)
}
```

- Add 1 to input.state variable
- Find the object at memory address @387
- Alter the state of the object at that address

Memory Heap Example

RAM
...
<Used by another program>
commandLineArgs
name:data, value: @387 (main)
<Used by another program>
...

RAM @387
...
ClassWithStateObject
-stateVar value:1
...

```
class ClassWithState{
  int stateVar = 0;
}
```

```
function addToState(input){
  input.stateVar += 1
}

function main{
  data = new ClassWithState()
  addToState(data)
  println(data.stateVar)
}
```

- Function call ends
- Destroy all data in the stack frame
- input is destroyed
- **Change to the object remains**

Memory Heap Example

RAM
...
<Used by another program>
commandLineArgs
name:data, value:387 (main)
<Used by another program>
...

RAM @387
...
ClassWithStateObject
-stateVar value:1
...

```
class ClassWithState{
  int stateVar = 0;
}
```

```
function addToState(input){
  input.stateVar += 1
}

function main{
  data = new ClassWithState()
  addToState(data)
  println(data.stateVar)
}
```

- Access data.stateVar
- Find the object at memory address @387
- Access the state of the object at that address

Memory Heap Example

RAM
...
<Used by another program>
<Used by another program>
...

RAM @387
...
...

```
class ClassWithState{
  int stateVar = 0;
}
```

```
function addToState(input){
  input.stateVar += 1
}

function main{
  data = new ClassWithState()
  addToState(data)
  println(data.stateVar)
}
```

- All memory freed when program ends

Lecture Question

Question: In a package named "physics" create a Scala **class** named "PhysicsVector" with the following:

- A constructor that takes 3 **var**iables of type Double named "x", "y", and "z"
- A method named "multiplyByConstant" that takes a Double and returns Unit. This method multiplies x, y, and z by the input
  - Be sure to update the state variables of the object when this method is called
  - Example: If a vector with x, y, and z of (2.0, 0.0, -1.5) has multiplyByConstant(2.0) called on it, it's state will become (4.0, 0.0, -3.0)
- A method named "addVector" that takes a PhysicsVector and returns Unit. This method adds the values of x, y, and z of the input vector to the state variables of the calling vector
  - Example: If a vector with x, y, and z of (2.0, 0.0, -1.5) has addVector(otherVector) called on it where otherVector is (-3.5, 0.4, -1.0), it's state will become (-1.5, 0.4, -2.5)

Testing: In a package named "tests" create a Scala class named "TestVector" as a test suite that tests **all** the functionality listed above

Lecture Question

Sample Usage

```
val v1 = new PhysicsVector(1.0, 2.0, -3.0)
val v2 = new PhysicsVector(-1.5, -3.5, 6.5)

v1.multiplyByConstant(3.0)
assert(Math.abs(v1.x - 3.0) < 0.0001, v1.x)
```

Commentary

Both methods for this lecture question return Unit. To test these methods you can access the state variables of the calling object after the method resolves

Since we're checking Doubles, be sure to allow enough tolerance to cover truncation errors