Model of Execution

Interpretation v. Compilation

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

```scala
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

```scala
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!
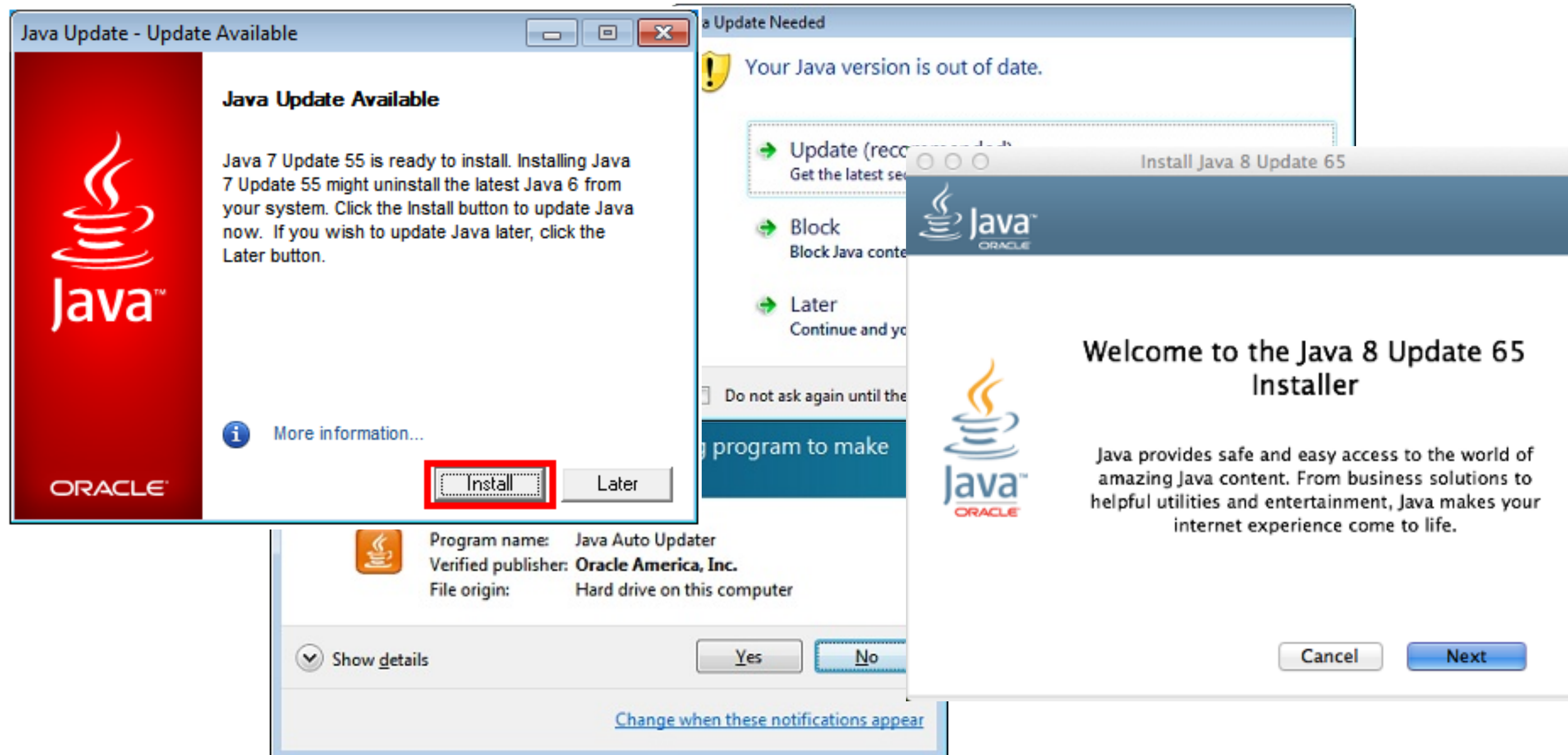
Compilation - Scala

- 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

  - Math in Scala is Java's Math class

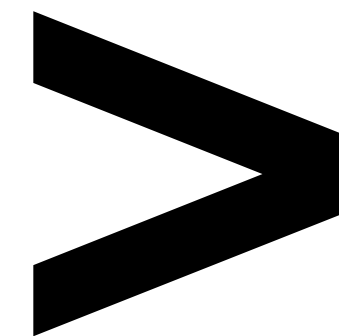 - We'll sometimes use Java libraries in this course
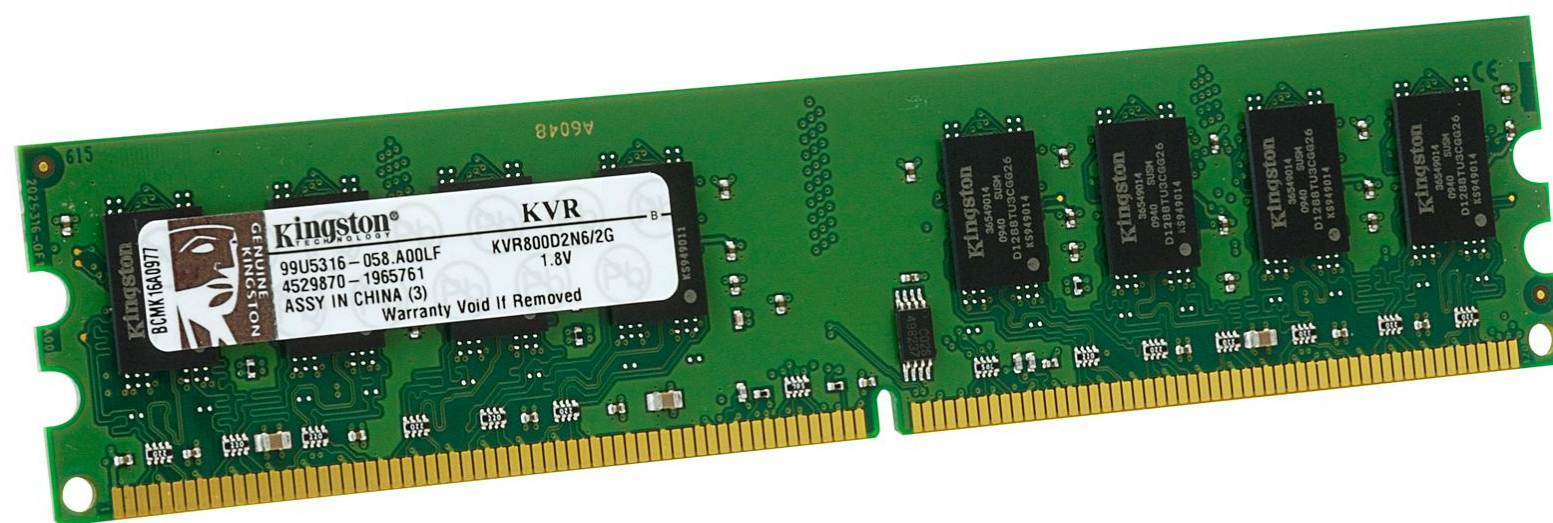
Memory

Let's Talk About Memory

- Random Access Memory (RAM)

  - Access any value by index

  - Effectively a giant array

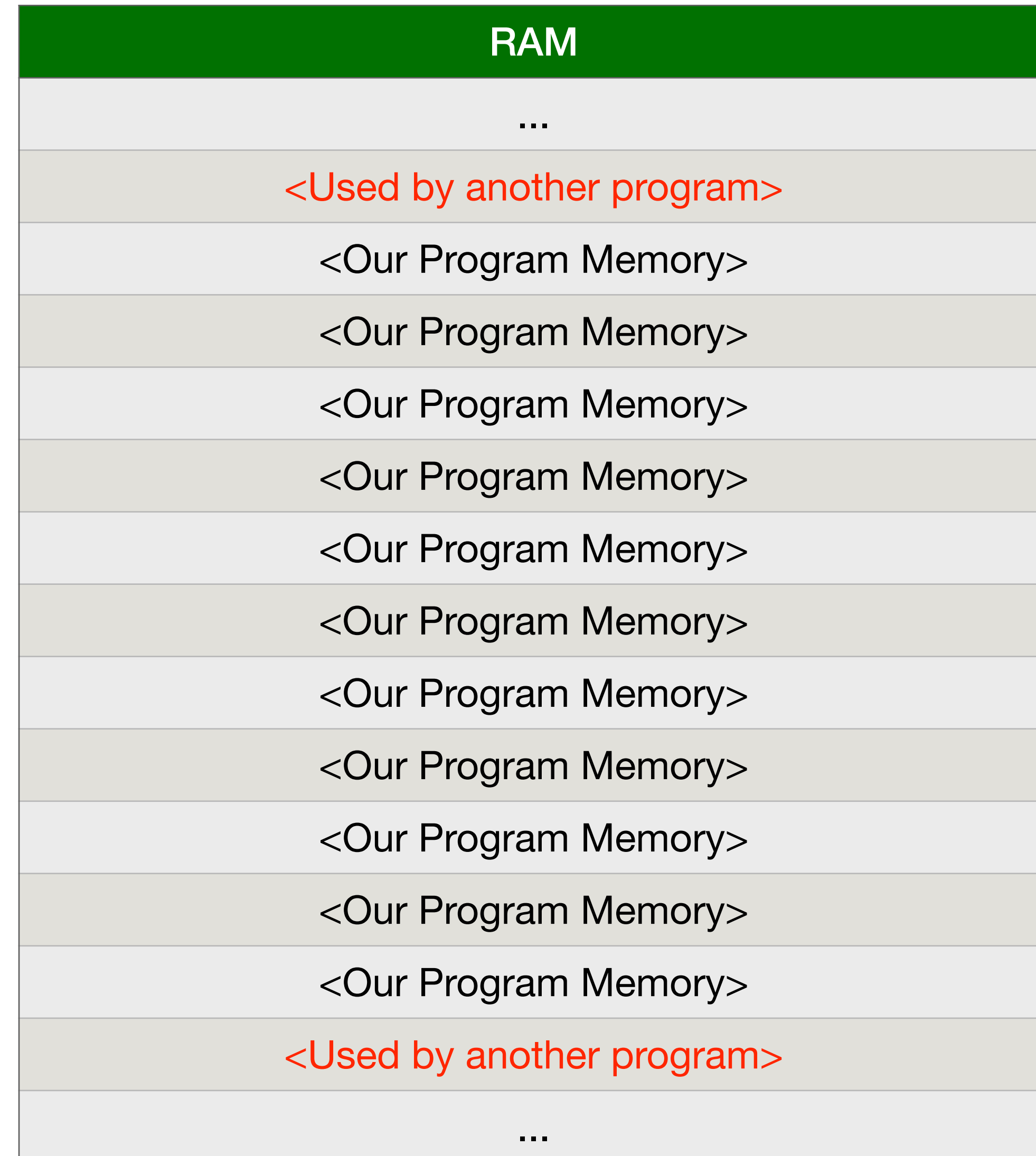- All values in your program are stored here
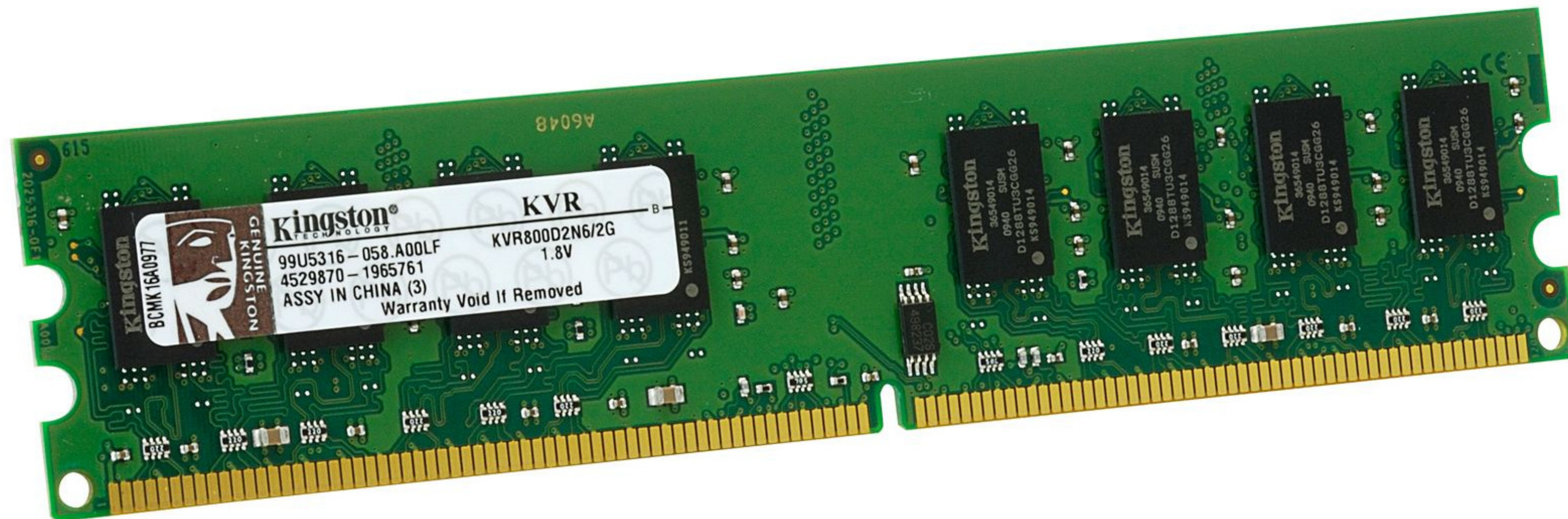
Let's Talk About Memory

- 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>
...

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

Variable Scope

- 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

Variable Scope

- 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** (Removed from the stack)

Stack Memory

- Only "primitive" types are stored in stack memory

  - Double/Float

  - Int/Long/Short

  - Char

  - Byte

  - Boolean

  - String

- All other objects are stored in heap memory*

  - Store reference to these object on the stack

*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