COMP 110/L Lecture 5

Kyle Dewey

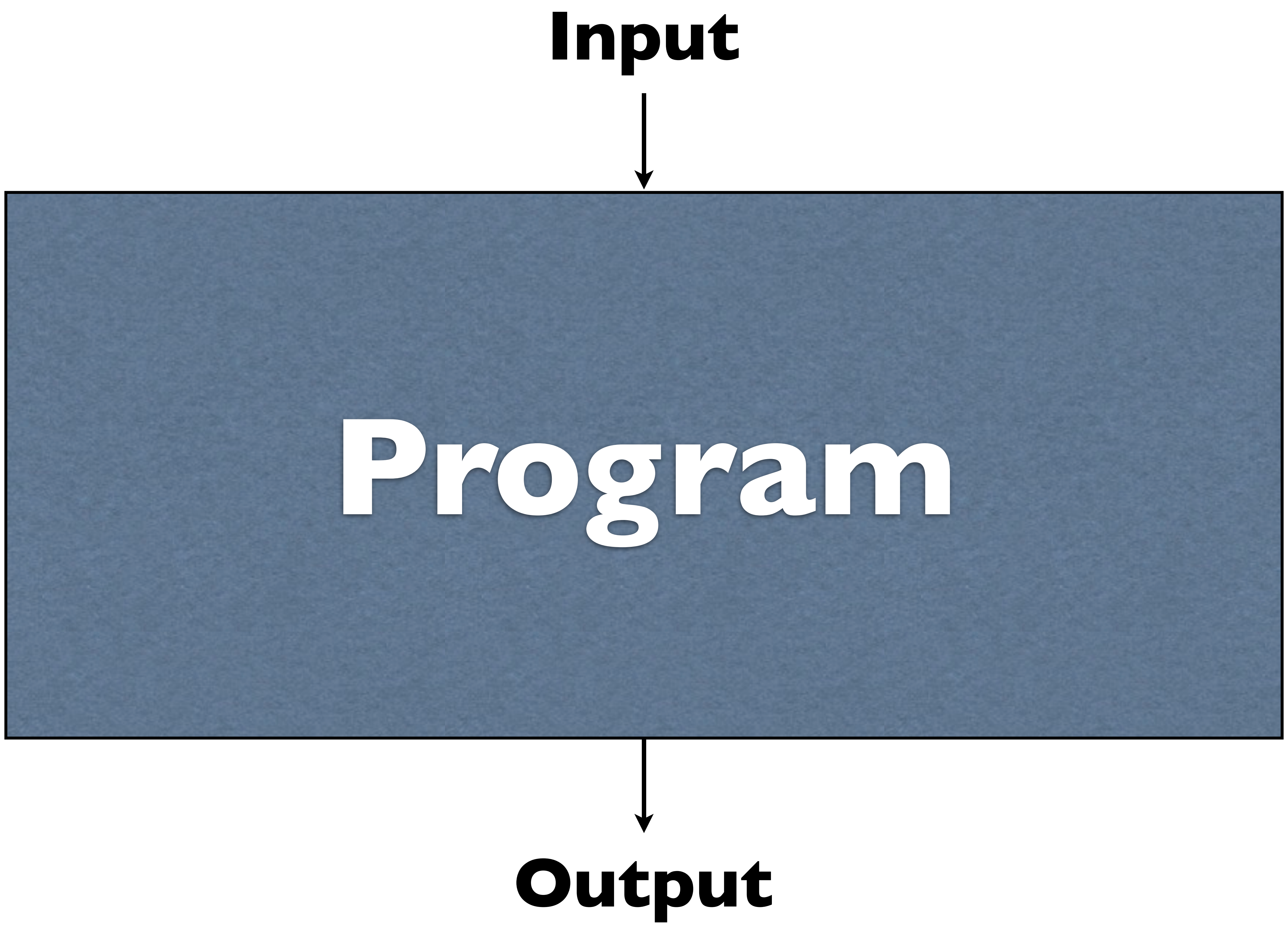
Outlines

- Methods
    - Defining methods
    - Calling methods

Methods

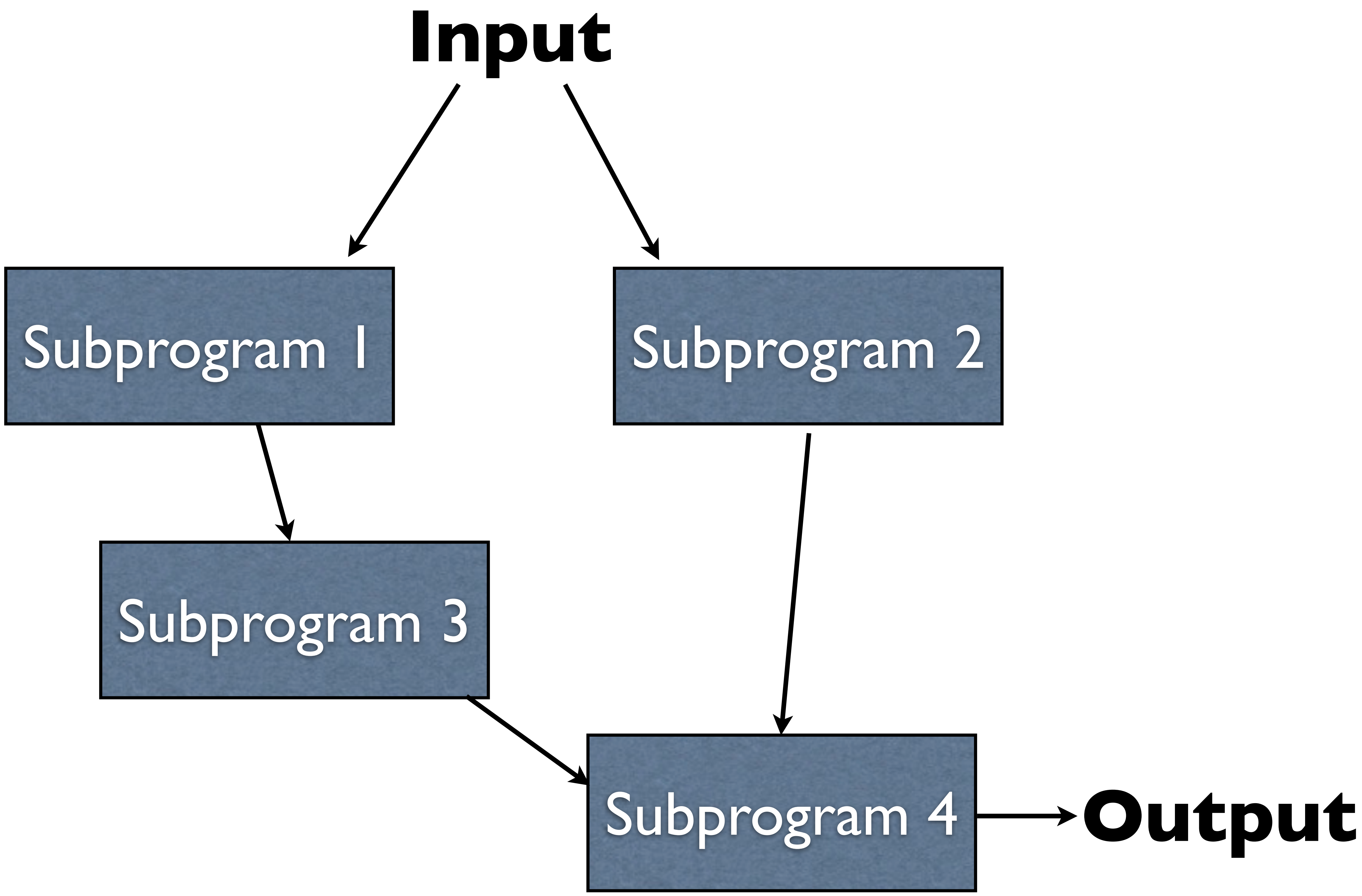
Motivation

Motivation



-Start off with some high-level motivation
-You write your program, and it's one giant block
-This is difficult to reason about

Motivation



-Simpler approach: write a bunch of smaller programs, and stitch them together
-Each program is a lot easier to reason about than the one big program
-If we're careful about how these different pieces interact with each other, then we only ever have to think about the small programs

Code Reuse

Code Reuse

```
System.out.println(...)
```

–You’re already familiar with these

Code Reuse

```
System.out.println(...)
nextInt()
```

–You're already familiar with these

Code Reuse

```
System.out.println(...)
nextInt()
nextLong()
```

–You're already familiar with these

Code Reuse

```
System.out.println(...)
nextInt()
nextLong()
nextDouble()
```

–You're already familiar with these

Code Reuse

```
System.out.println(...)
nextInt()
nextLong()
nextDouble()
```

You have used all of these multiple times.

–You’re already familiar with these, and you’ve used them a bunch of times

Code Reuse

```
System.out.println(...)
nextInt()
nextLong()
nextDouble()
```

You have used all of these multiple times.

These are all *methods*.

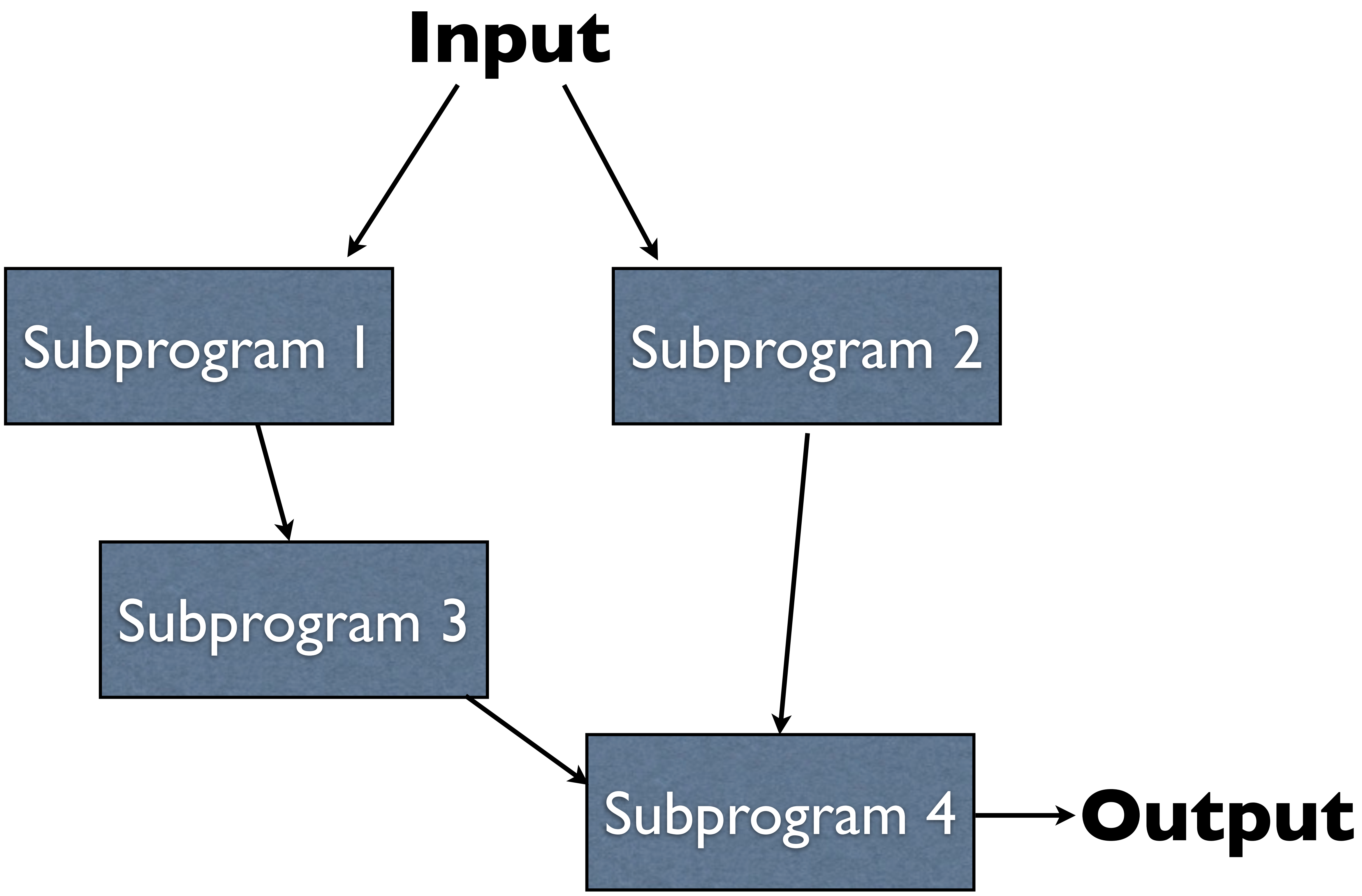
–You’re already familiar with these, and you’ve used them a bunch of times

Methods

Distinct subprograms.

Methods

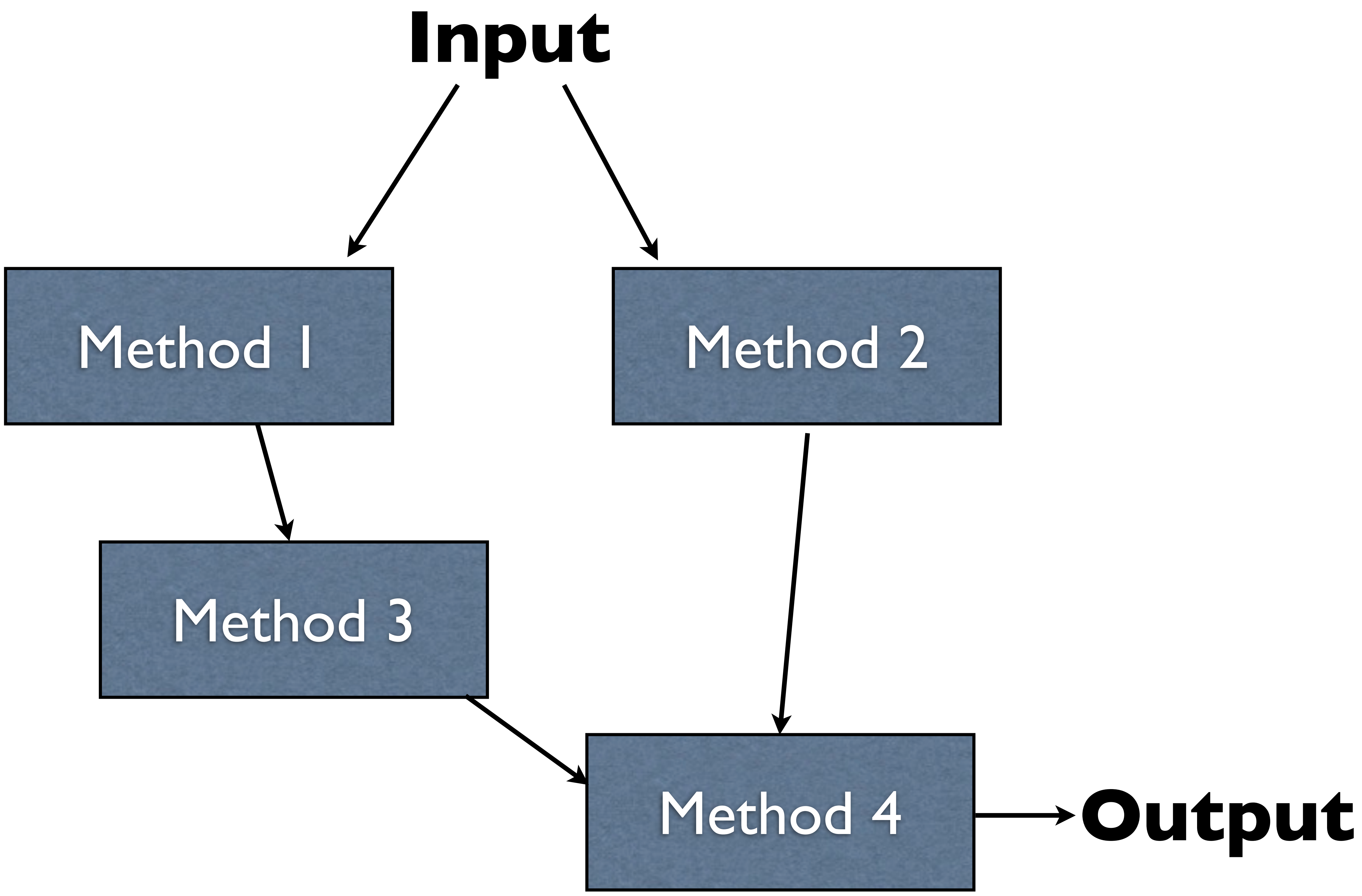
Distinct subprograms.



-Taking that illustration from before...

Methods

Distinct subprograms.



-...each one of those subprograms is a method

Method Terminology

- We can *define* a method
  - Make it available to the rest of the program
- We can *call* a method
  - Execute the subprogram

Method Anatomy

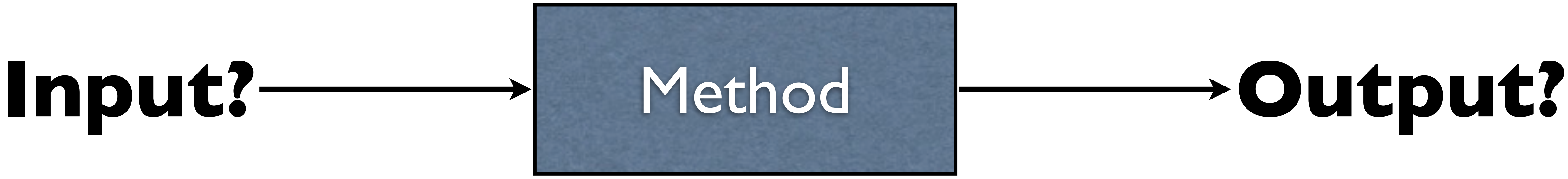
Methods take some number of inputs (can be 0).

Methods may produce an output.

Method Anatomy

Methods take some number of inputs (can be 0).

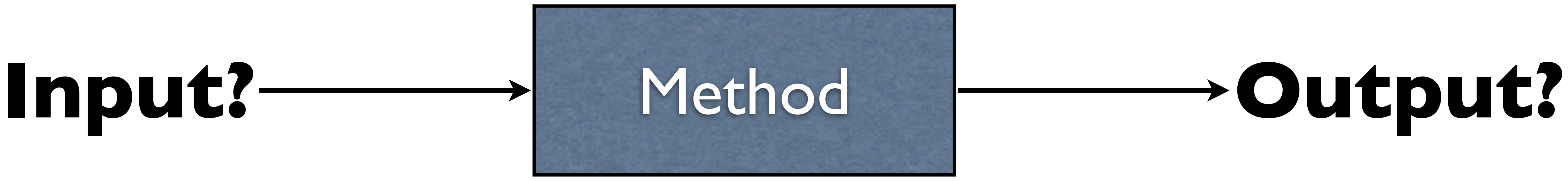
Methods may produce an output.



Method Anatomy

Methods take some number of inputs (can be 0).

Methods may produce an output.

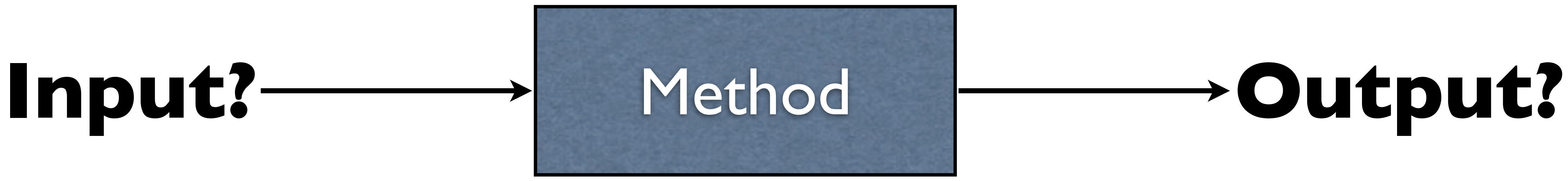


```
System.out.println("Hello");
```

Method Anatomy

Methods take some number of inputs (can be 0).

Methods may produce an output.



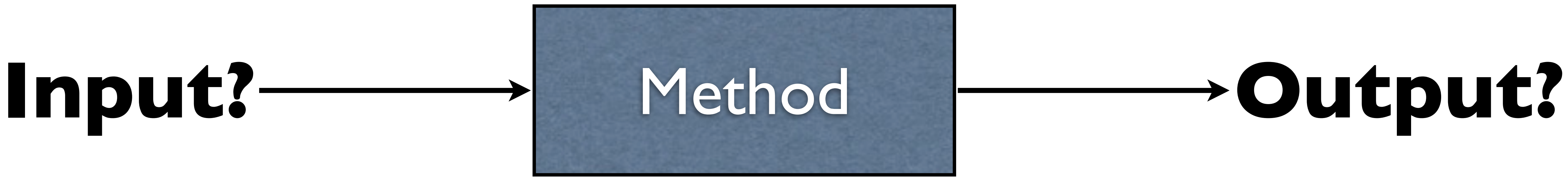
```
System.out.println("Hello");
```

One input, no outputs (cannot assign to a variable).

Method Anatomy

Methods take some number of inputs (can be 0).

Methods may produce an output.



```
System.out.println("Hello");
```

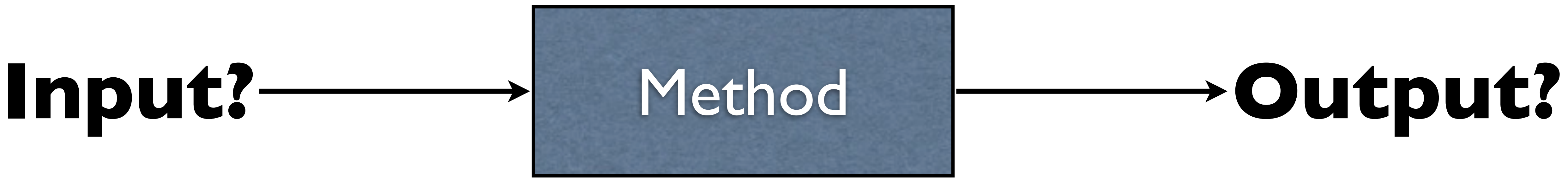
One input, no outputs (cannot assign to a variable).

```
Math.pow(2, 3);
```

Method Anatomy

Methods take some number of inputs (can be 0).

Methods may produce an output.



```
System.out.println("Hello");
```

One input, no outputs (cannot assign to a variable).

```
Math.pow(2, 3);
```

Two inputs, one output.

```
inputScanner.nextInt();
inputScanner.nextInt();
```

No inputs, one output.

```
inputScanner.nextInt();
```

No inputs, one output.

```
System.out.print("Goodbye");
```

```
inputScanner.nextInt();
```

No inputs, one output.

```
System.out.print("Goodbye");
```

One input, no outputs (cannot assign to a variable)

```
inputScanner.nextInt();
```

No inputs, one output.

```
System.out.print("Goodbye");
```

One input, no outputs (cannot assign to a variable)

```
inputScanner.nextLong();
inputScanner.nextInt();
```

No inputs, one output.

```
System.out.print("Goodbye");
```

One input, no outputs (cannot assign to a variable)

```
inputScanner.nextLong();
```

No inputs, one output.

```
inputScanner.nextInt();
```

No inputs, one output.

```
System.out.print("Goodbye");
```

One input, no outputs (cannot assign to a variable)

```
inputScanner.nextLong();
```

No inputs, one output.

```
inputScanner.nextDouble();
inputScanner.nextInt();
```

No inputs, one output.

```
System.out.print("Goodbye");
```

One input, no outputs (cannot assign to a variable)

```
inputScanner.nextLong();
```

No inputs, one output.

```
inputScanner.nextDouble();
```

No inputs, one output.

Defining a Method

Easiest to see with real code.

Example:

`Return42.java`

-The `return` reserved word says that the method should end and return with a given value at this point

Example:
`ReturnParameter.java`

Example:
`MultParameters1.java`

Example:
`MultParameters2.java`

Example:
`MultParameters3.java`

Method Definition General Form

```
public static
returnType
methodName(parameter_list) {
   ...
   return expression;
}
```

Method Definition General Form

```
public static
returnType
methodName(parameter_list) {
   ...
   return expression;
}
```

Magic

Method Definition General Form

```
public static
returnType
methodName(parameter_list) {
   ...
   return expression;
}
```

Magic

Type of value produced

Method Definition General Form

```
public static
returnType
methodName(parameter_list) {
    ...
    return expression;
}
```

Magic

Type of value produced

Name given to method; same naming rules as variables

Method Definition General Form

Magic

```
public static
returnType
methodName(parameter_list) {
   ...
   return expression;
}
```

Type of value produced

Inputs to method (`int x`)

Name given to method; same naming rules as variables

Method Definition General Form

```
public static
returnType
methodName(parameter_list) {
   ...
   return expression;
}
```

Magic

Type of value produced

Inputs to method (`int x`)

Name given to method; same naming rules as variables

Method ends here, evaluates `expression`, and produces its result

Methods which Produce no Values

Methods which produce no values have a `void` return type

Example: `ReturnNothing.java`

`main` Method

`main` is just another method.

`main` serves as the entry point to your program.

`main` Method

`main` is just another method.

`main` serves as the entry point to your program.

```
public static
void
main(String[] args) {
   ...
}
```

-main’s return type is void - it produces no value (doesn’t return anything)
-String[] is actually a type, so args is a parameter
-Later on we’ll get into what the type `String[]` is (not the same as just String), along with what this parameter to main holds

Calling Methods

- Execution enters the method calls
- The method is executed
- The method returns to wherever it was called from

Calling Methods

- Execution enters the method calls
- The method is executed
- The method returns to wherever it was called from

Calling Methods

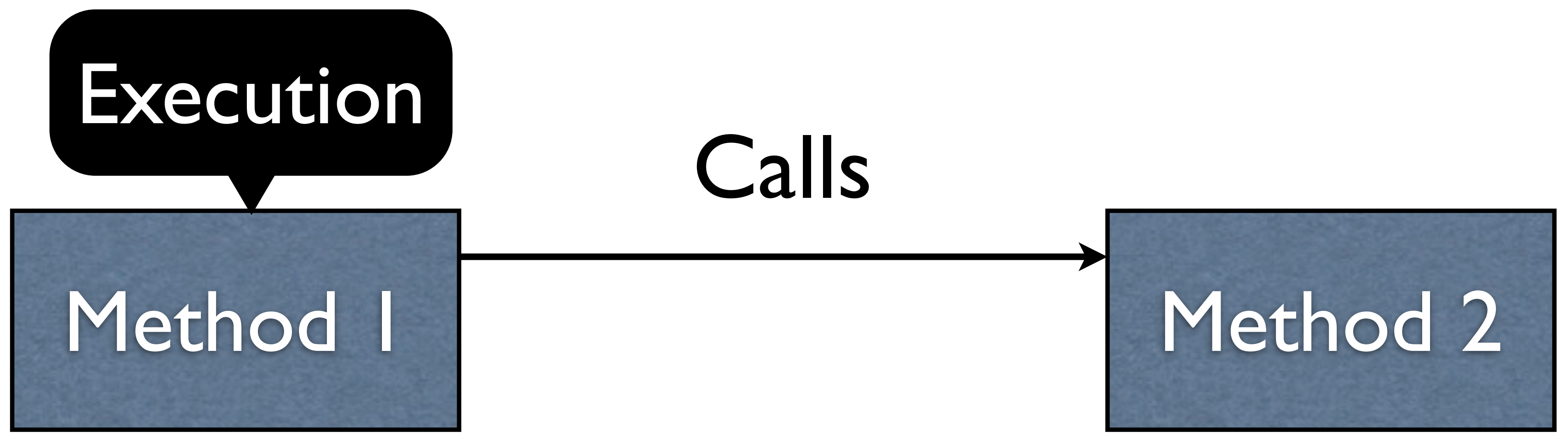
- Execution enters the method calls
- The method is executed
- The method returns to wherever it was called from

Execution
Method 1
Method 2

-Initially, execution is in method 1

Calling Methods

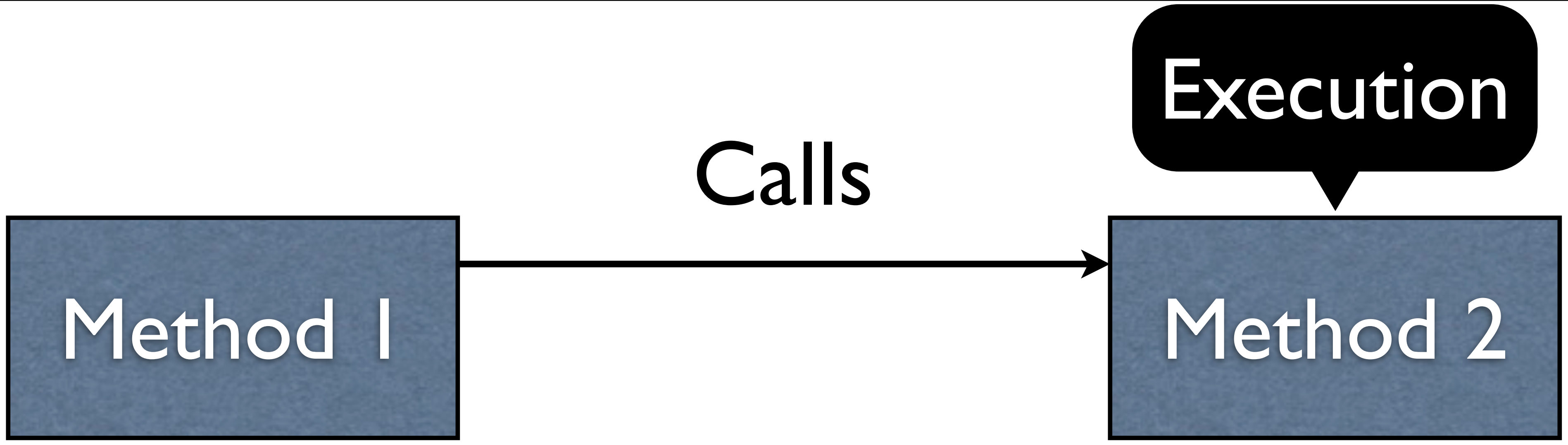
- Execution enters the method calls
- The method is executed
- The method returns to wherever it was called from



-Method 1 then calls method 2

Calling Methods

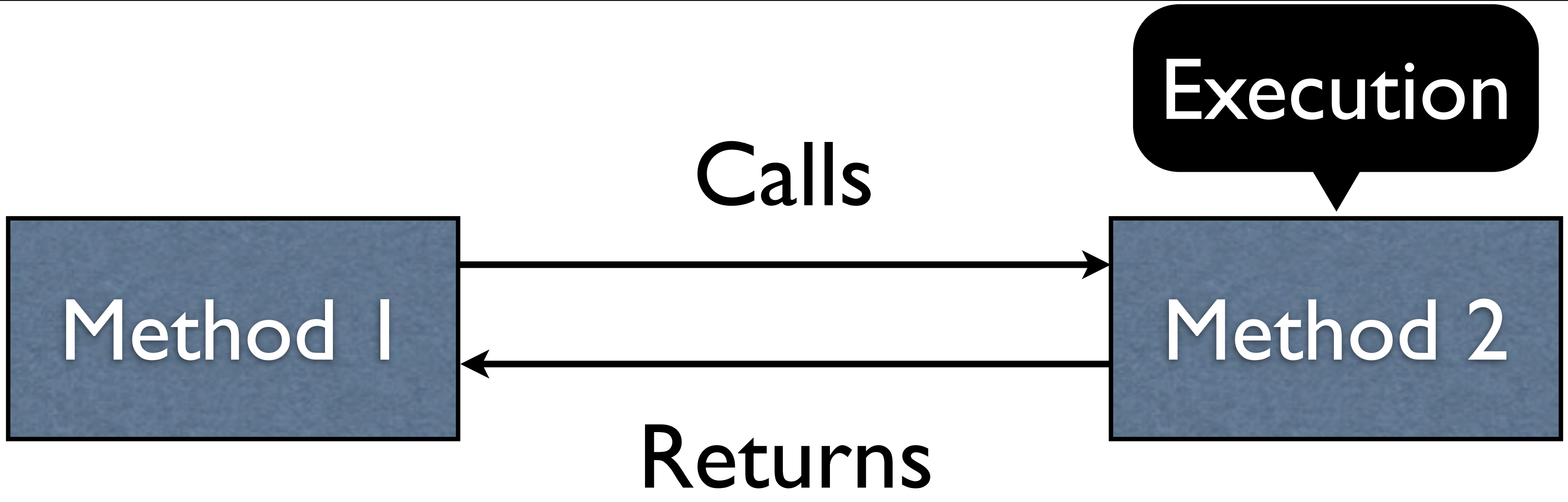
- Execution enters the method calls
- The method is executed
- The method returns to wherever it was called from



-Execution transfers to method 2 as a result of the call

Calling Methods

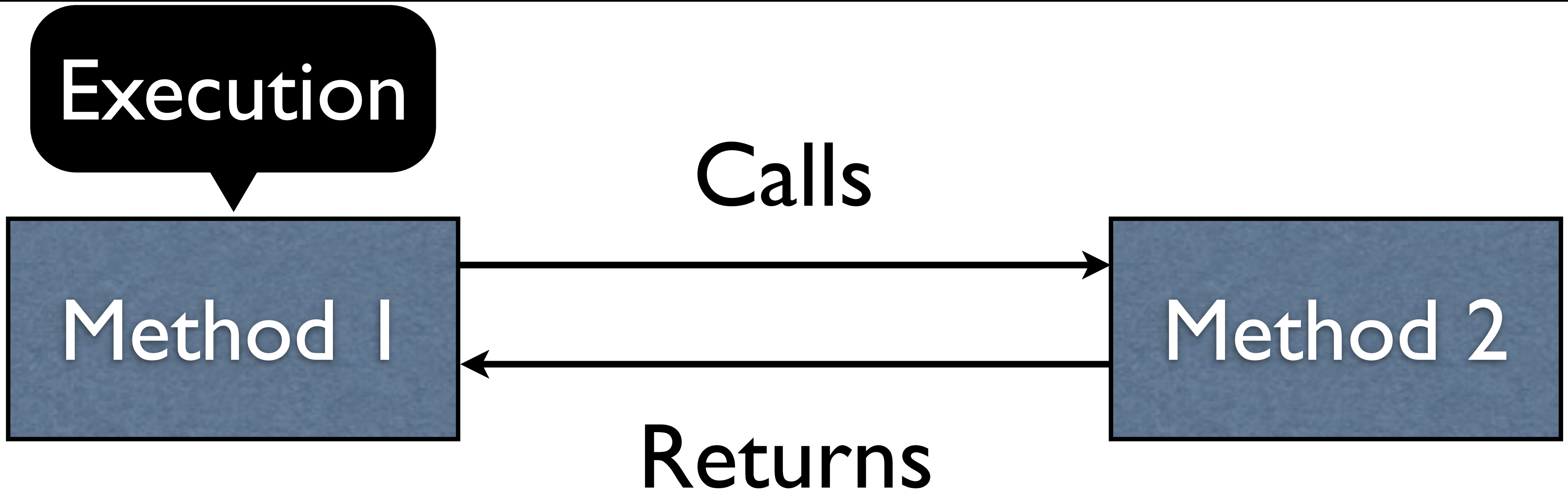
- Execution enters the method calls
- The method is executed
- The method returns to wherever it was called from



-Method 2 eventually completes, returning back to method 1

Calling Methods

- Execution enters the method calls
- The method is executed
- The method returns to wherever it was called from



-Once the return is complete, execution resumes back in method 1 wherever it left off