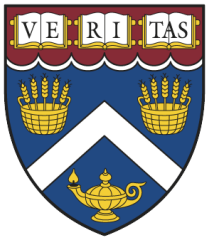
Assignment 1

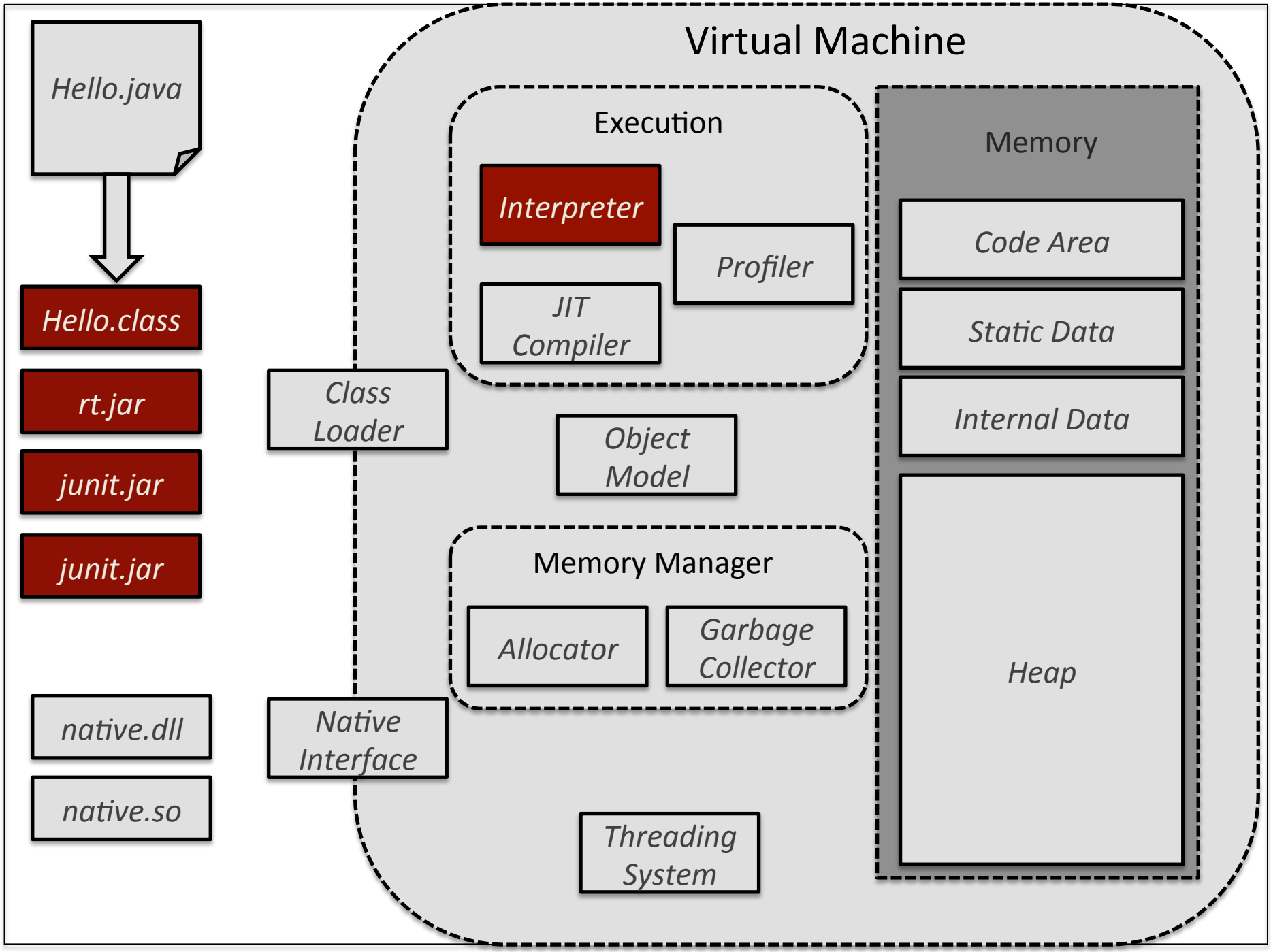
- Will be posted during the week
- We will talk next week about the SimpleJava VM
- Due on Sunday October 5th at 11:59pm

Assignment 1

- You will implement parts of the interpreter
  - IADD
  - IINC
  - GOTO
  - GETFIELD
  - PUTFIELD
  - INVOKESTATIC
  - INVOKEVIRTUAL

Bytecode and Interpretation




Hello.java
Hello.class
rt.jar
junit.jar
junit.jar
native.dll
native.so
Virtual Machine
Execution
Interpreter
Profiler
JIT Compiler
Class Loader
Object Model
Memory Manager
Allocator
Garbage Collector
Native Interface
Threading System
Memory
Code Area
Static Data
Internal Data
Heap

Java VM Architecture

- Java bytecode targets an idealized architecture
  - Allowed the designers to define their own semantics
  - Delegate translation to machine code to the VM
- Bytecode targets a stack architecture
  - Reverse Polish vs. Arithmetic calculators

The JVM Stack

- Each thread has a JVM stack
  - Similar to the stack in native executables
- Each stack frame represents a method
- The stack frame has three data structures
  - An operand stack
  - A set of local variables
  - An instruction pointer

Bytecode Format

- A method's code is a stream of unsigned bytes
- Every instruction starts with an opcode
  - One byte
- Optionally, instructions contain additional bytes
  - Constant values
  - Constant pool indices
  - Jump targets

Computational Types

- Not all types are the same size
  - Java specifies the sizes of its primitive types

boolean	True or False
byte	8-bit signed
char	16-bit unicode
double	64-bit floating point
float	32-bit floating point
int	32-bit signed
long	64-bit signed
short	16-bit signed

Computational Types

- Most data types are padded to 32 bits internally
  - Computational Type 1
- `long` and `double` values are stored as 64 bits
  - Computational Type 2
- The computational type affects the stack frame
  - Type 1 takes one local slot or operand stack entry
  - Type 2 takes two local slots or operand stack entries
- Some bytecodes have different semantics
  - See the VM spec for the `DUP` instructions

Stack Execution

```
public static int add() {
   final int i = 2;
   final int j = 3;
   return i + j;
 }
```

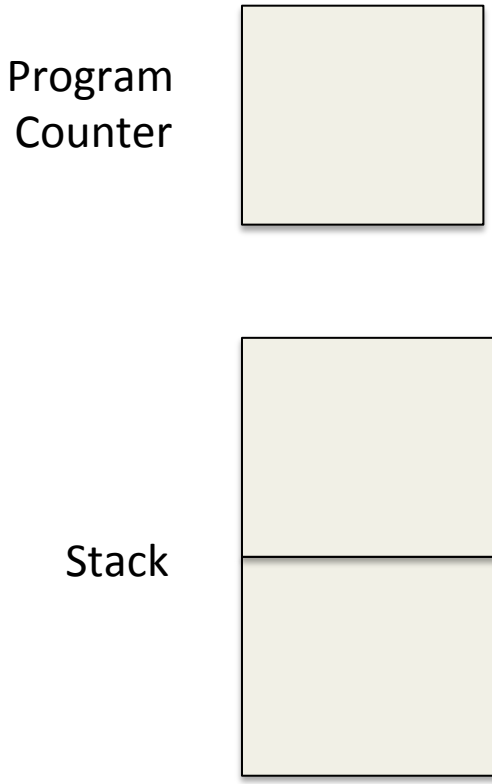
Stack Execution

```
public static int add() {
   final int i = 2;
   final int j = 3;
   return i + j;
 }
```

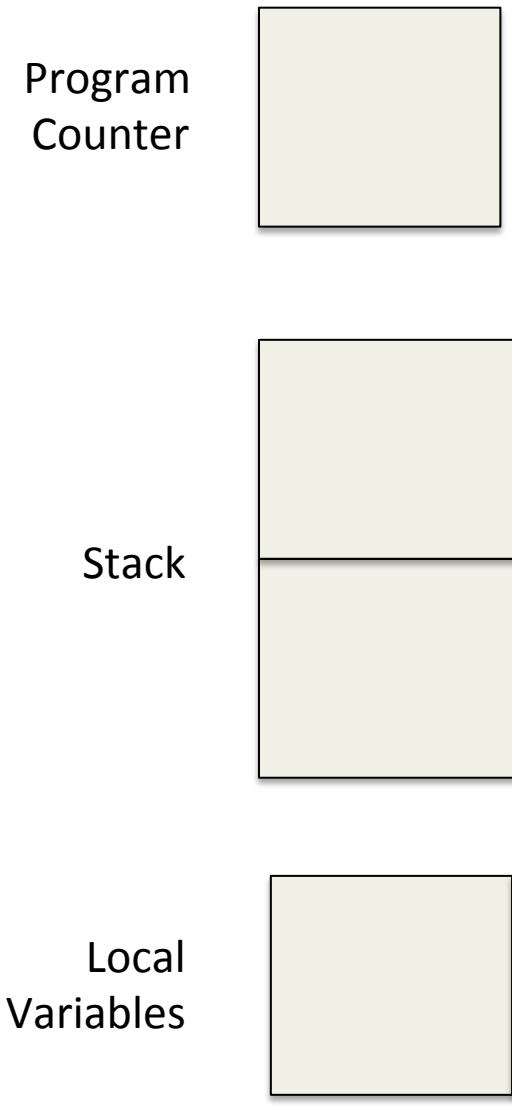
```
public static int add();
    flags: ACC_PUBLIC, ACC_STATIC
    Code:
      stack=2, locals=2, args_size=0
      0: iconst_2
      1: istore_0
      2: iconst_3
      3: istore_1
      4: iload_0
      5: iload_1
      6: iadd
      7: ireturn
    LineNumberTable:
      line 15: 0
      line 16: 2
      line 17: 4
    LocalVariableTable:
    Start  Length  Slot  Name   Signature
      1       7     0     i          I
      3       5     1     j          I
```

Program Counter

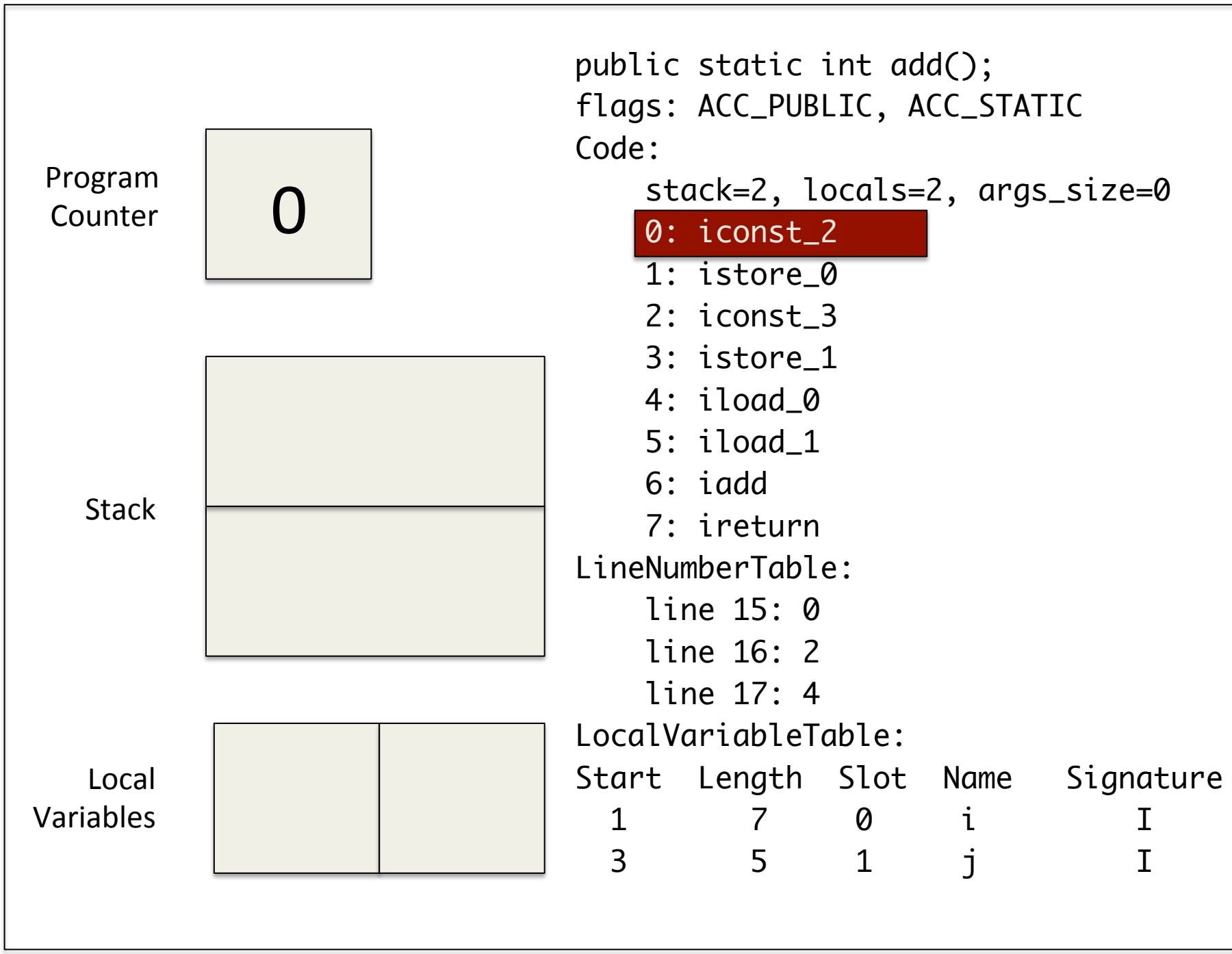
```
public static int add();
flags: ACC_PUBLIC, ACC_STATIC
Code:
    stack=2, locals=2, args_size=0
    0: iconst_2
    1: istore_0
    2: iconst_3
    3: istore_1
    4: iload_0
    5: iload_1
    6: iadd
    7: ireturn
LineNumberTable:
    line 15: 0
    line 16: 2
    line 17: 4
LocalVariableTable:
Start  Length  Slot  Name   Signature
  1       7     0     i          I
  3       5     1     j          I
```

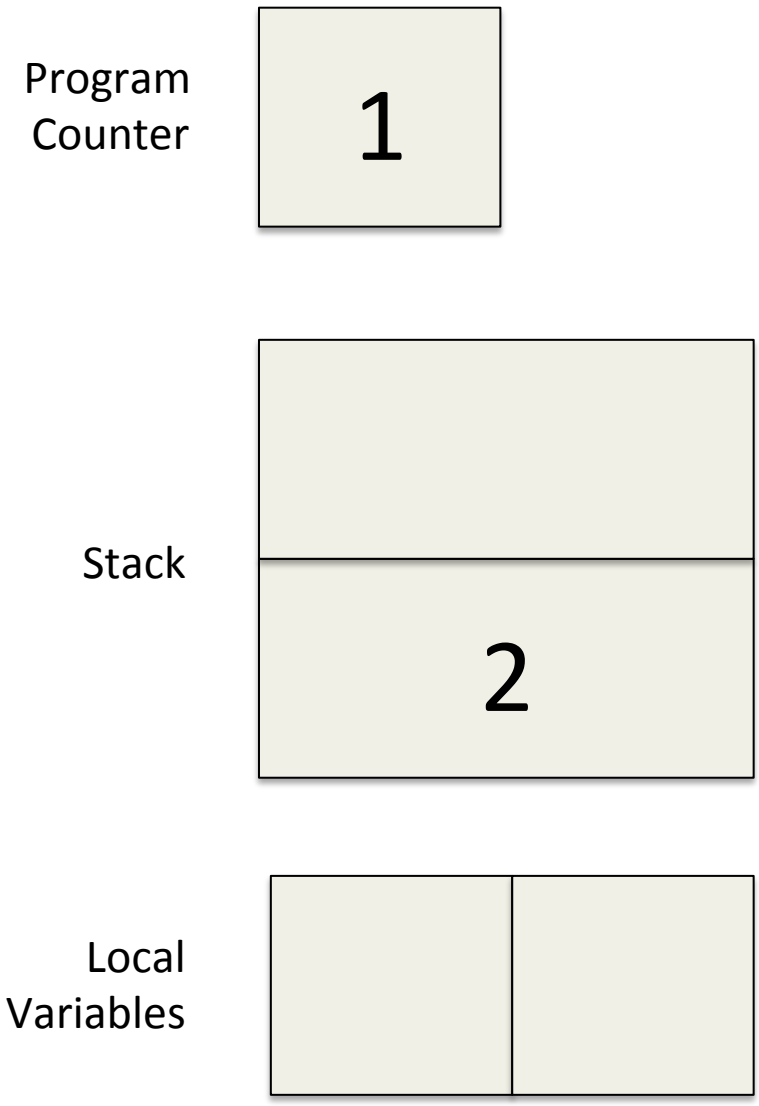



```
public static int add();
flags: ACC_PUBLIC, ACC_STATIC
Code:
    stack=2, locals=2, args_size=0
    0: iconst_2
    1: istore_0
    2: iconst_3
    3: istore_1
    4: iload_0
    5: iload_1
    6: iadd
    7: ireturn
LineNumberTable:
    line 15: 0
    line 16: 2
    line 17: 4
LocalVariableTable:
Start  Length  Slot  Name   Signature
  1       7     0     i          I
  3       5     1     j          I
```

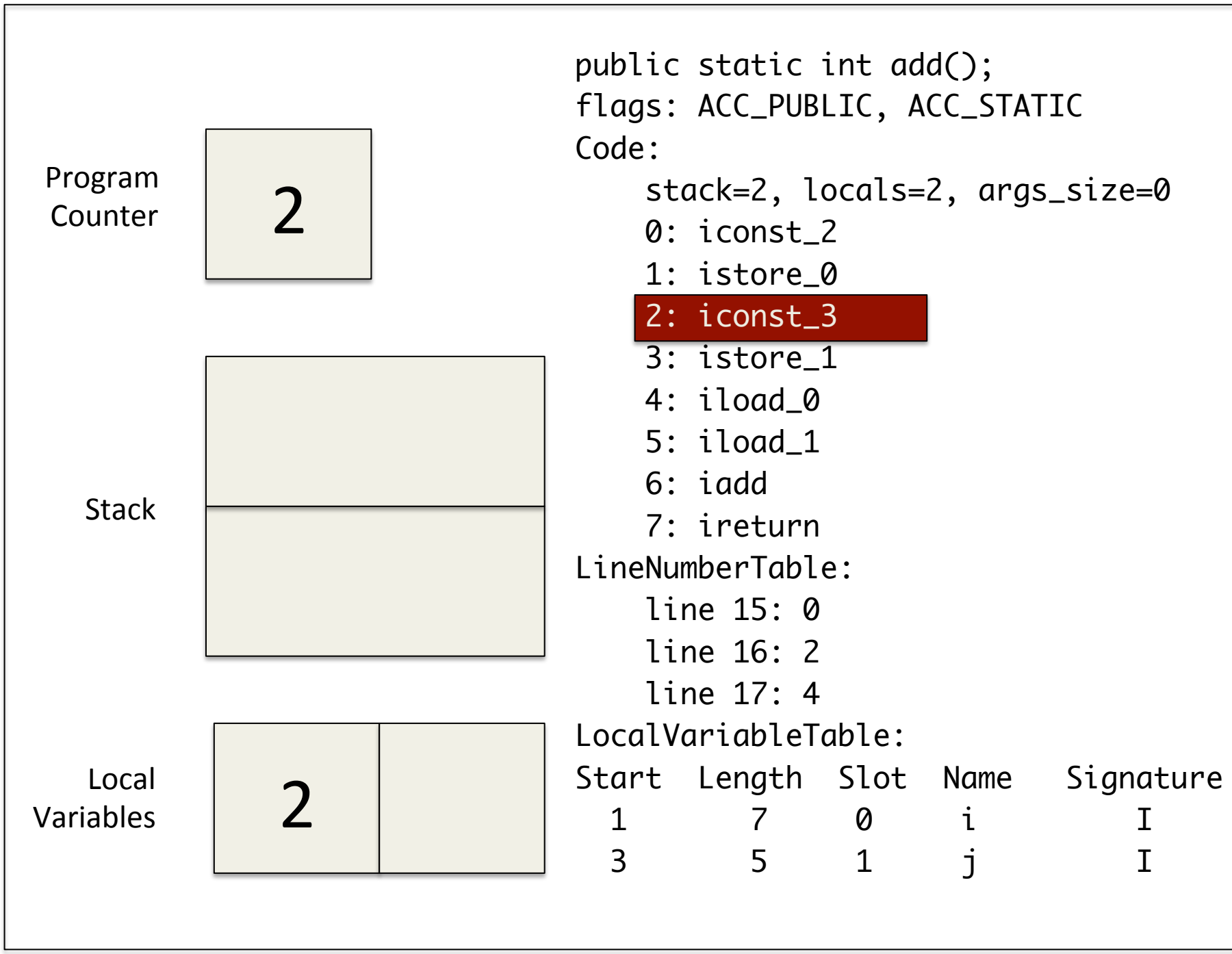


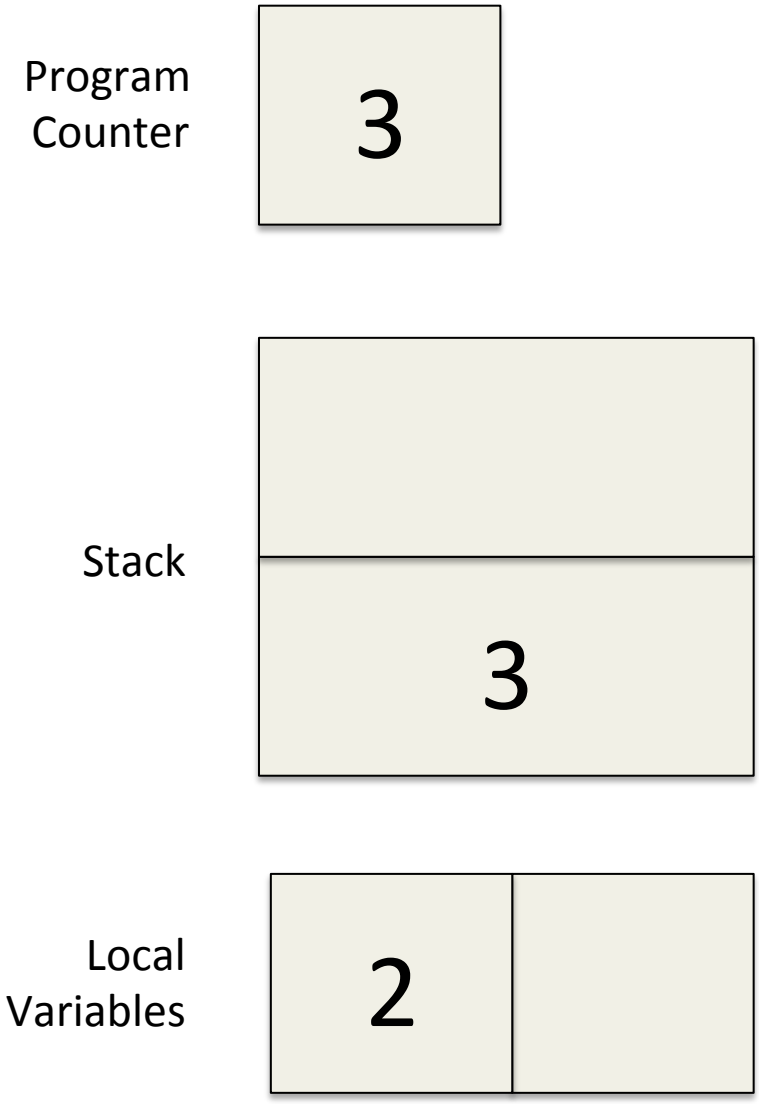

```
public static int add();
flags: ACC_PUBLIC, ACC_STATIC
Code:
    stack=2, locals=2, args_size=0
    0: iconst_2
    1: istore_0
    2: iconst_3
    3: istore_1
    4: iload_0
    5: iload_1
    6: iadd
    7: ireturn
LineNumberTable:
    line 15: 0
    line 16: 2
    line 17: 4
LocalVariableTable:
Start  Length  Slot  Name   Signature
  1       7     0     i          I
  3       5     1     j          I
```


Program Counter
0
Stack
Local Variables
public static int add();
flags: ACC_PUBLIC, ACC_STATIC
Code:
stack=2, locals=2, args_size=0
0: iconst_2
1: istore_0
2: iconst_3
3: istore_1
4: iload_0
5: iload_1
6: iadd
7: ireturn
LineNumberTable:
line 15: 0
line 16: 2
line 17: 4
LocalVariableTable:
Start Length Slot Name Signature
1 7 0 i I
3 5 1 j I

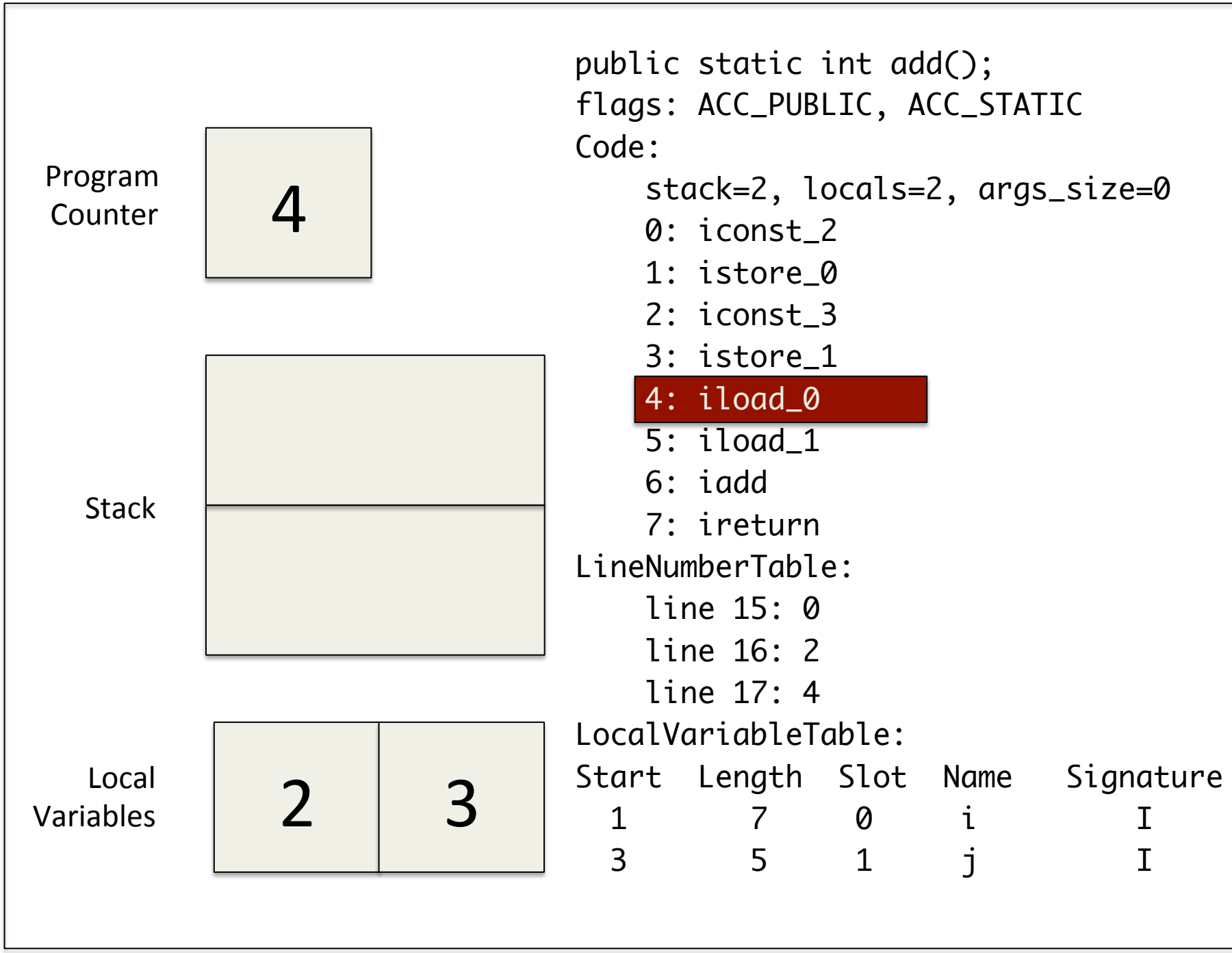


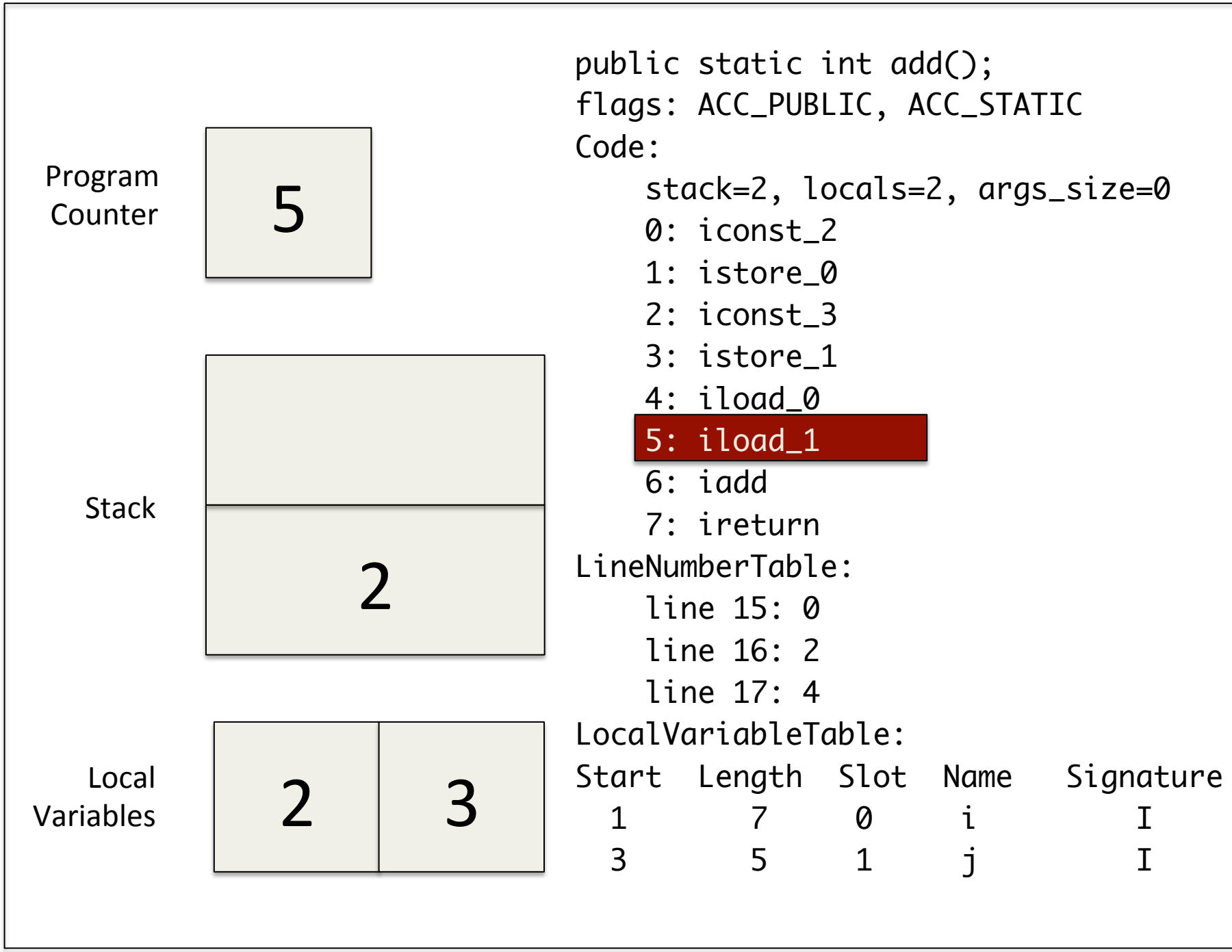

```
public static int add();
flags: ACC_PUBLIC, ACC_STATIC
Code:
    stack=2, locals=2, args_size=0
    0: iconst_2
    1: istore_0
    2: iconst_3
    3: istore_1
    4: iload_0
    5: iload_1
    6: iadd
    7: ireturn
LineNumberTable:
    line 15: 0
    line 16: 2
    line 17: 4
LocalVariableTable:
Start  Length  Slot  Name   Signature
  1       7     0     i          I
  3       5     1     j          I
```

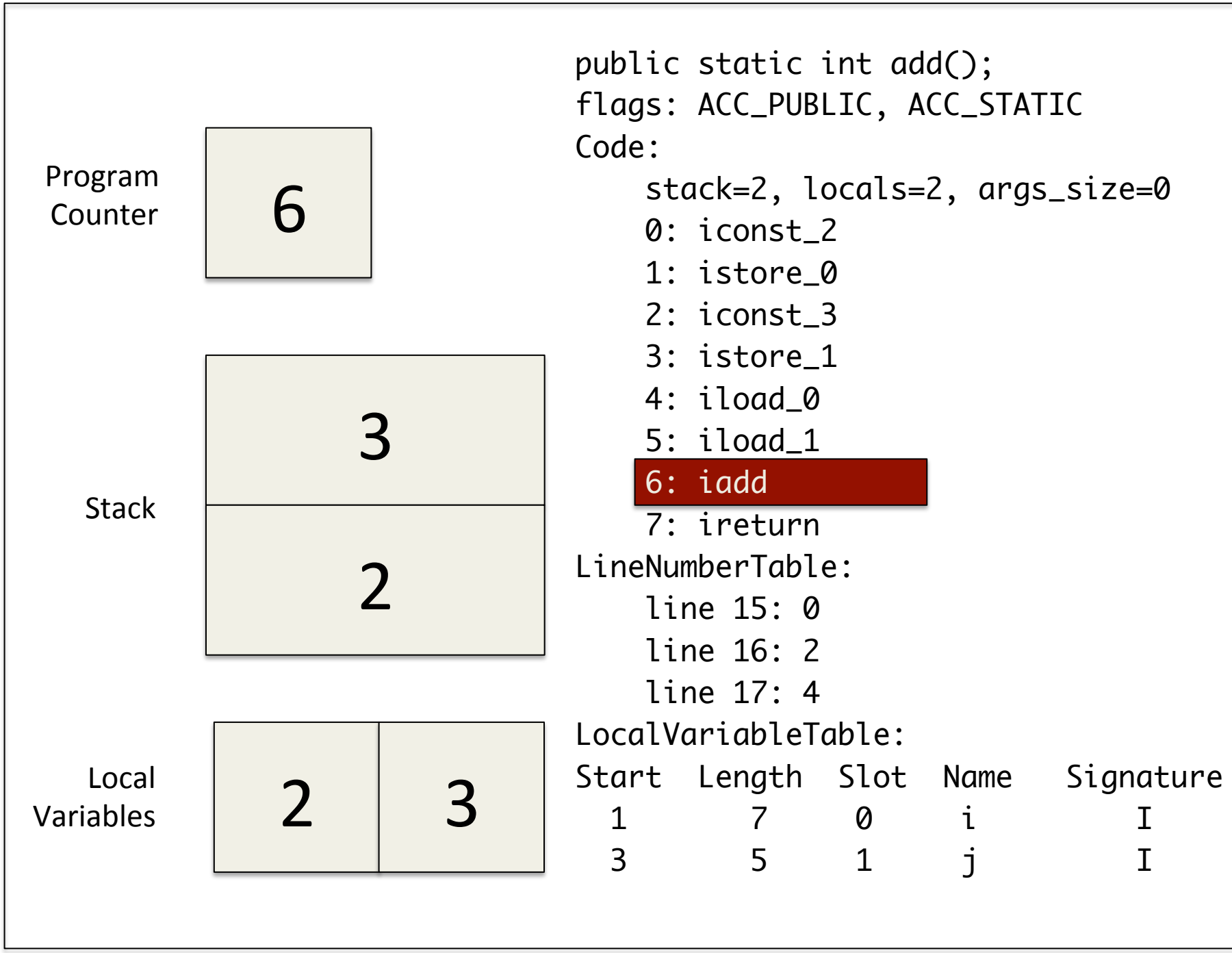

Program Counter
2
Stack
Local Variables
2
public static int add();
flags: ACC_PUBLIC, ACC_STATIC
Code:
stack=2, locals=2, args_size=0
0: iconst_2
1: istore_0
2: iconst_3
3: istore_1
4: iload_0
5: iload_1
6: iadd
7: ireturn
LineNumberTable:
line 15: 0
line 16: 2
line 17: 4
LocalVariableTable:
Start Length Slot Name Signature
1 7 0 i I
3 5 1 j I

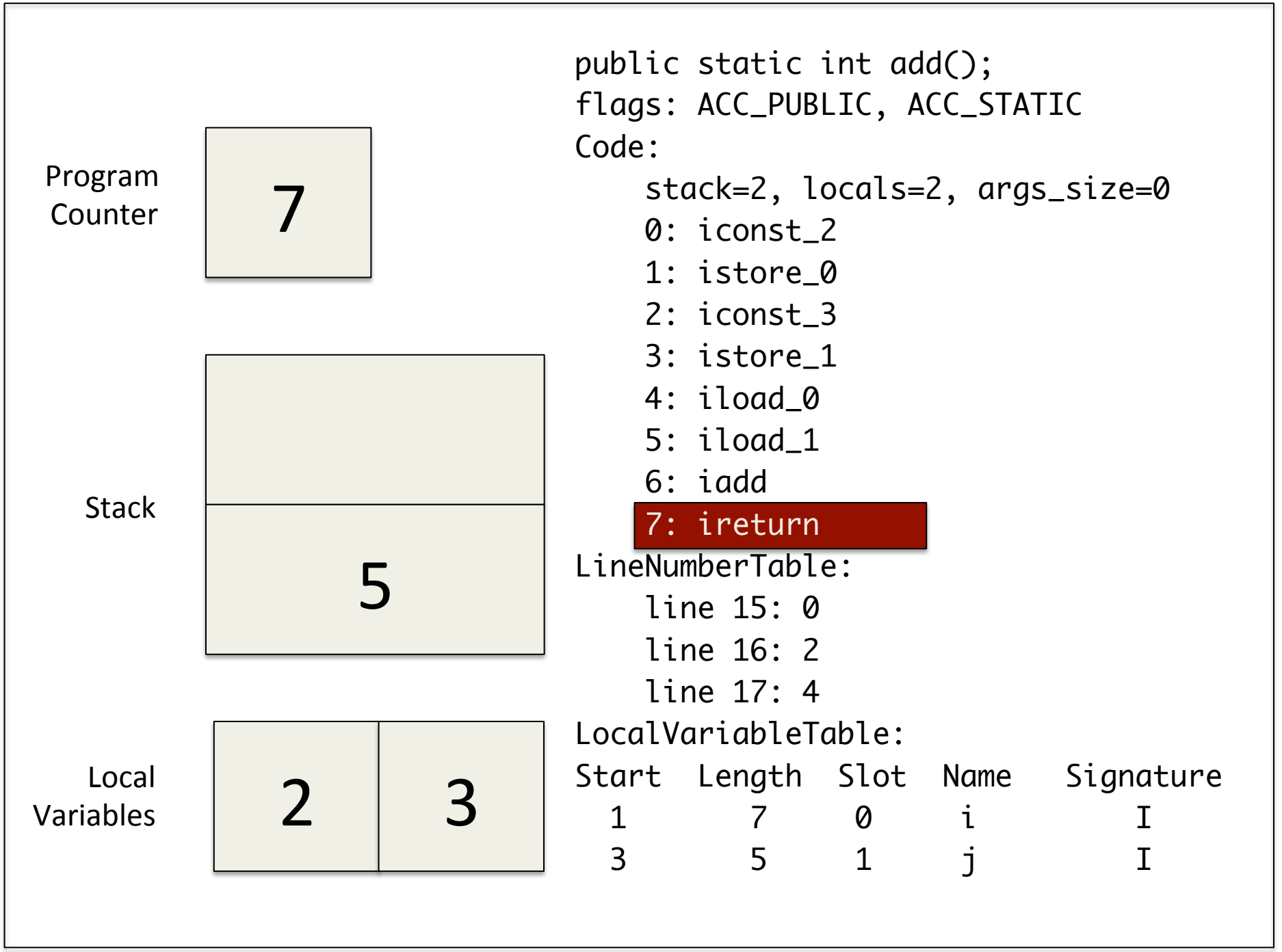



```
public static int add();
flags: ACC_PUBLIC, ACC_STATIC
Code:
    stack=2, locals=2, args_size=0
    0: iconst_2
    1: istore_0
    2: iconst_3
    3: istore_1
    4: iload_0
    5: iload_1
    6: iadd
    7: ireturn
LineNumberTable:
    line 15: 0
    line 16: 2
    line 17: 4
LocalVariableTable:
Start  Length  Slot  Name   Signature
  1       7     0     i          I
  3       5     1     j          I
```


Program Counter
4
Stack
Local Variables
2
3
public static int add();
flags: ACC_PUBLIC, ACC_STATIC
Code:
stack=2, locals=2, args_size=0
0: iconst_2
1: istore_0
2: iconst_3
3: istore_1
4: iload_0
5: iload_1
6: iadd
7: ireturn
LineNumberTable:
line 15: 0
line 16: 2
line 17: 4
LocalVariableTable:
Start Length Slot Name Signature
1 7 0 i I
3 5 1 j I


Program Counter
5
Stack
2
Local Variables
2
3
public static int add();
flags: ACC_PUBLIC, ACC_STATIC
Code:
stack=2, locals=2, args_size=0
0: iconst_2
1: istore_0
2: iconst_3
3: istore_1
4: iload_0
5: iload_1
6: iadd
7: ireturn
LineNumberTable:
line 15: 0
line 16: 2
line 17: 4
LocalVariableTable:
Start Length Slot Name Signature
1 7 0 i I
3 5 1 j I


Program Counter
6
Stack
3
2
Local Variables
2
3
public static int add();
flags: ACC_PUBLIC, ACC_STATIC
Code:
stack=2, locals=2, args_size=0
0: iconst_2
1: istore_0
2: iconst_3
3: istore_1
4: iload_0
5: iload_1
6: iadd
7: ireturn
LineNumberTable:
line 15: 0
line 16: 2
line 17: 4
LocalVariableTable:
Start Length Slot Name Signature
1 7 0 i I
3 5 1 j I


Program Counter
7
Stack
5
Local Variables
2
3
public static int add();
flags: ACC_PUBLIC, ACC_STATIC
Code:
stack=2, locals=2, args_size=0
0: iconst_2
1: istore_0
2: iconst_3
3: istore_1
4: iload_0
5: iload_1
6: iadd
7: ireturn
LineNumberTable:
line 15: 0
line 16: 2
line 17: 4
LocalVariableTable:
Start Length Slot Name Signature
1 7 0 i I
3 5 1 j I

Calling Methods

- Each thread starts with a single method
  - `main(String[] args)`
  - `run()`
  - `call()`
- Subsequent methods are pushed to the stack
- The thread exits when that method exits

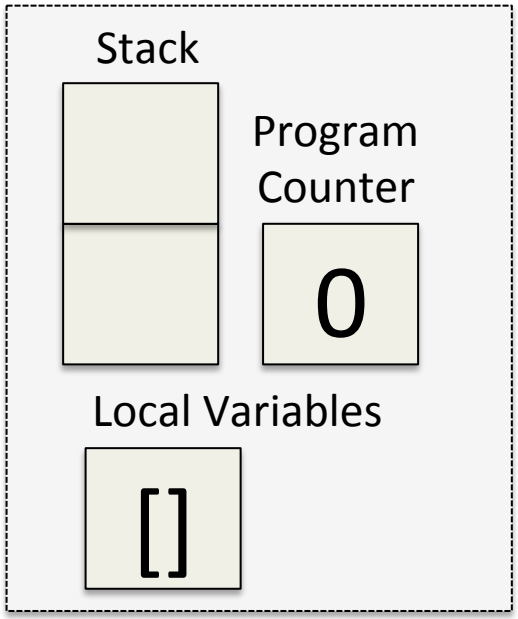
Calling Convention

- How values are passed between methods
  - All programming languages define a convention
- Parameters are stored in local variables
  - Top value stored in the first slot
- Return value pushed to the caller's stack

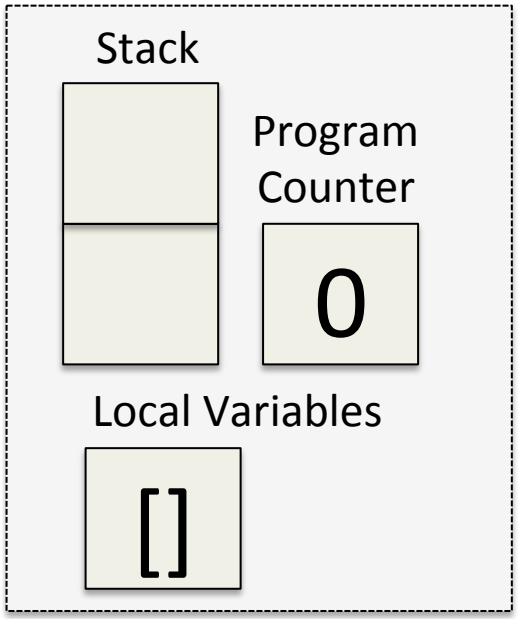
```
public static void main(final String[] args) {
   add(2, 3);
 }

 public static int add(final int i, final int j) {
   return i + j;
 }
```

```java
public static void main(final String[] args) {
   add(2, 3);
 }

 public static int add(final int i, final int j) {
   return i + j;
 }
```

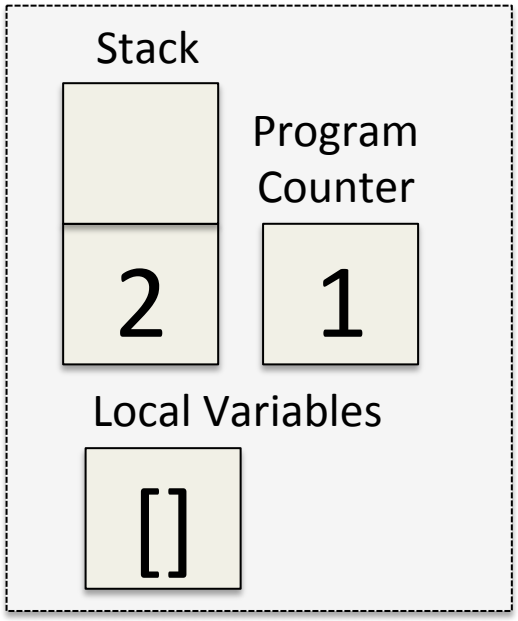
```
public static void main(java.lang.String[]);
    0: iconst_2
    1: iconst_3
    2: invokestatic  #16  // Method add:(II)I
    5: pop
    6: return

public static int add(int, int);
    0: iload_0
    1: iload_1
    2: iadd
    3: ireturn
```



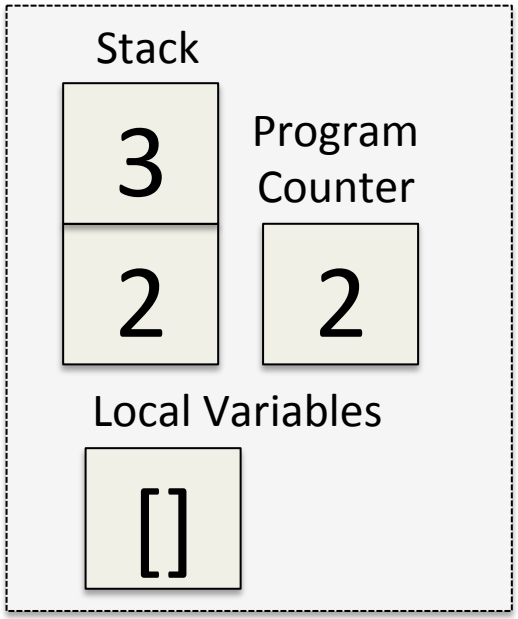

```
public static void main(java.lang.String[]);
    0: iconst_2
    1: iconst_3
    2: invokestatic  #16  // Method add:(II)I
    5: pop
    6: return
```



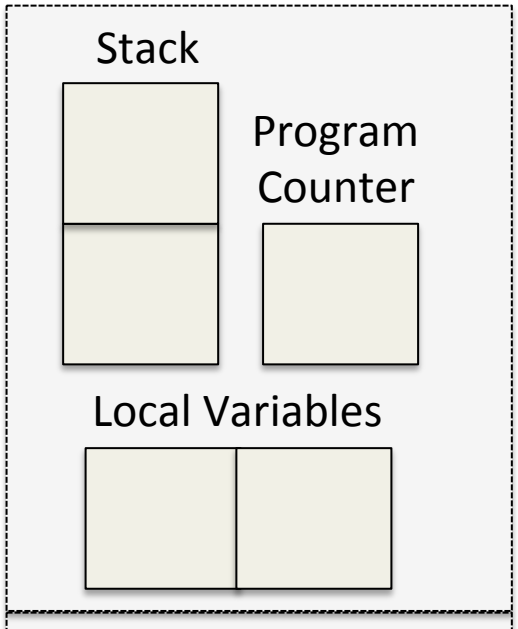

```
public static void main(java.lang.String[]);
   0: iconst_2
   1: iconst_3
   2: invokestatic  #16  // Method add:(II)I
   5: pop
   6: return
```



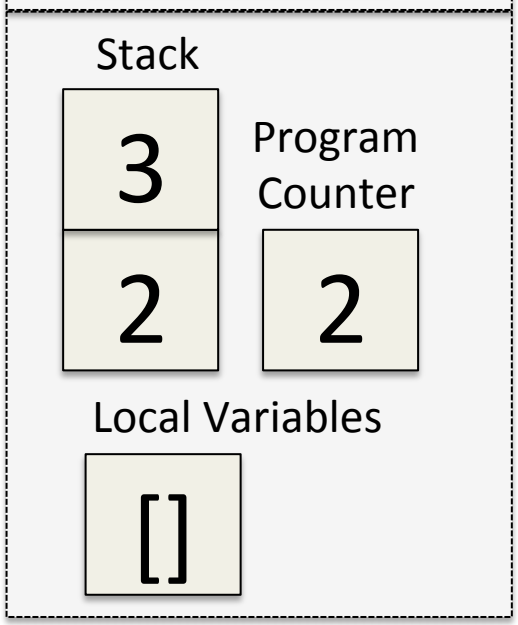

```
public static void main(java.lang.String[]);
    0: iconst_2
    1: iconst_3
    2: invokestatic  #16  // Method add:(II)I
    5: pop
    6: return
```



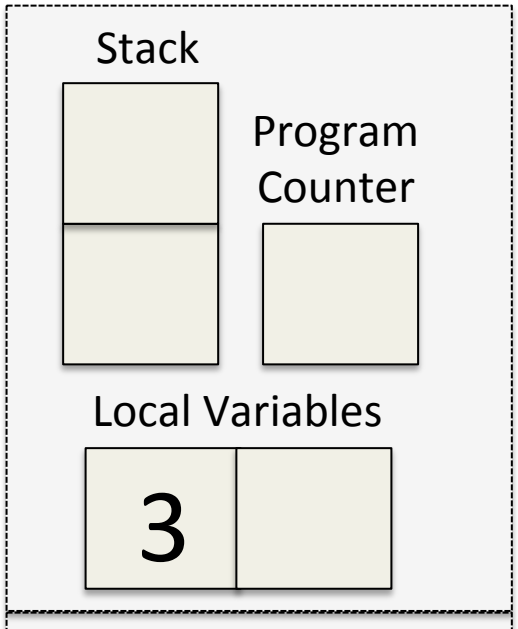

```
public static void main(java.lang.String[]);
    0: iconst_2
    1: iconst_3
    2: invokestatic  #16  // Method add:(II)I
    5: pop
    6: return
```



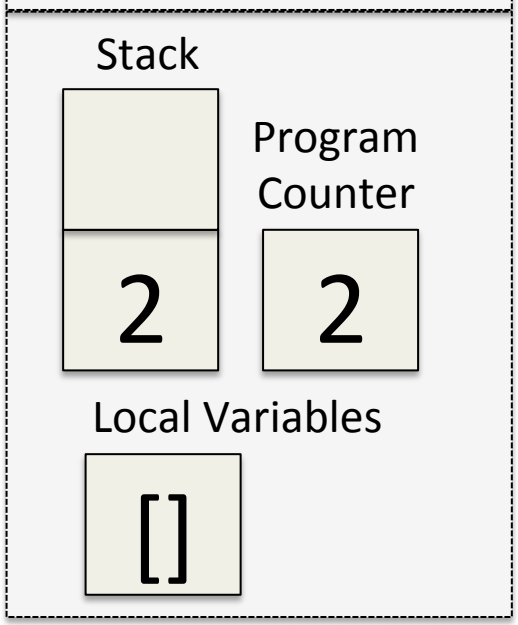

```
public static int add(int, int);
    0: iload_0
    1: iload_1
    2: iadd
    3: ireturn
```



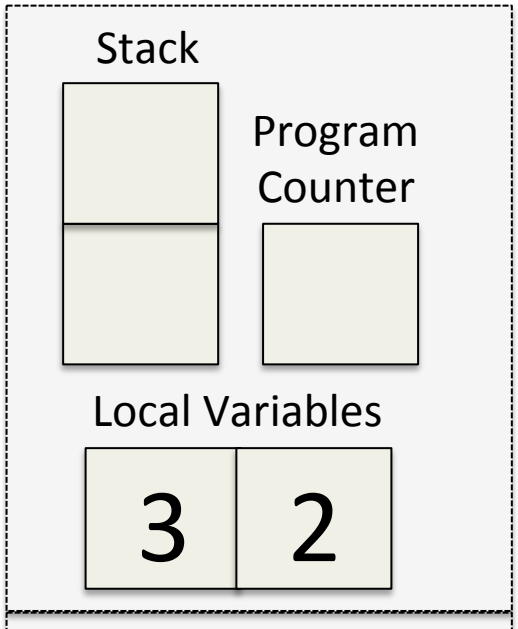

```
public static void main(java.lang.String[]);
    0: iconst_2
    1: iconst_3
    2: invokestatic  #16  // Method add:(II)I
    5: pop
    6: return
```



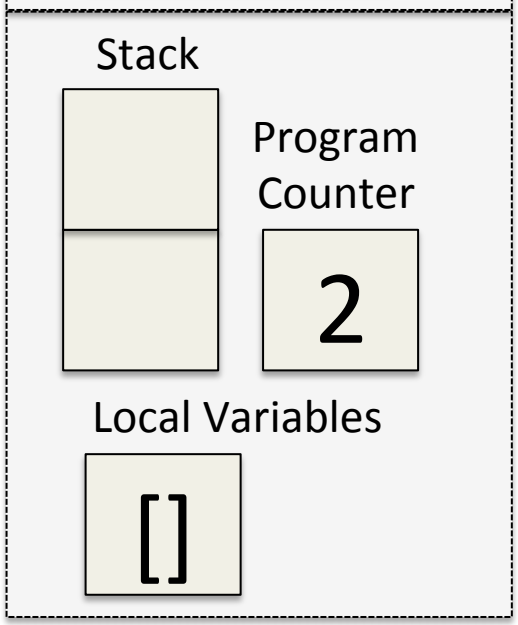

```
public static int add(int, int);
    0: iload_0
    1: iload_1
    2: iadd
    3: ireturn
```



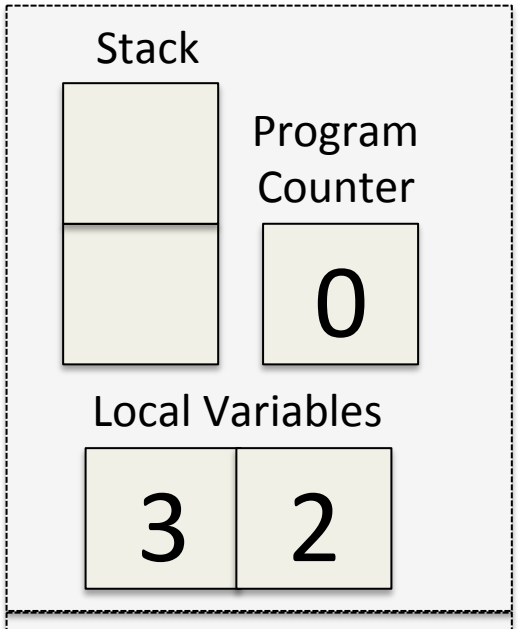

```
public static void main(java.lang.String[]);
    0: iconst_2
    1: iconst_3
    2: invokestatic  #16  // Method add:(II)I
    5: pop
    6: return
```



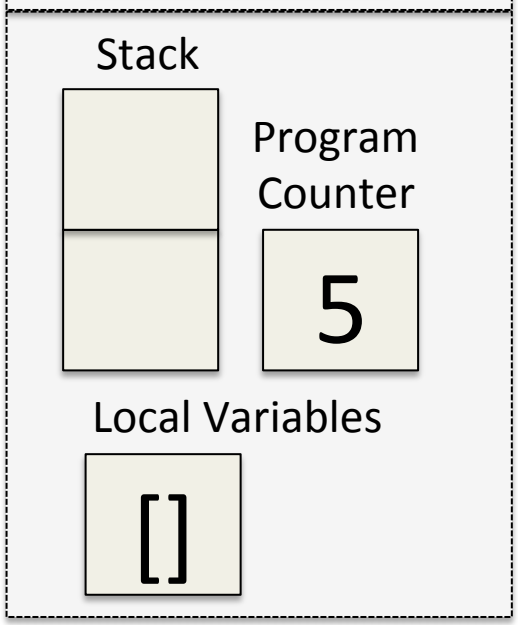

```
public static int add(int, int);
    0: iload_0
    1: iload_1
    2: iadd
    3: ireturn
```



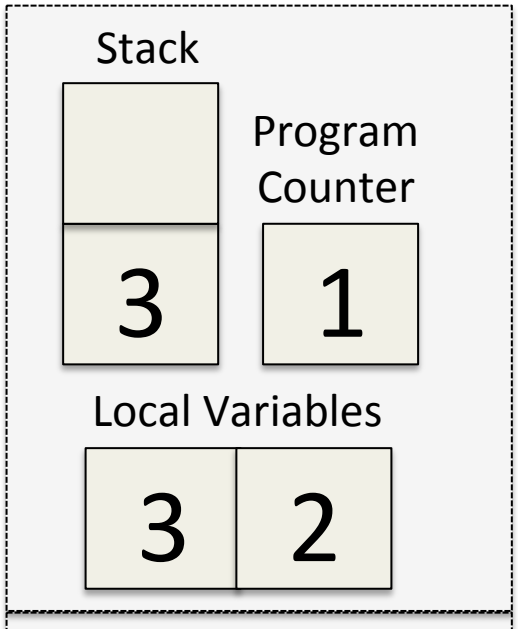

```
public static void main(java.lang.String[]);
    0: iconst_2
    1: iconst_3
    2: invokestatic  #16  // Method add:(II)I
    5: pop
    6: return
```



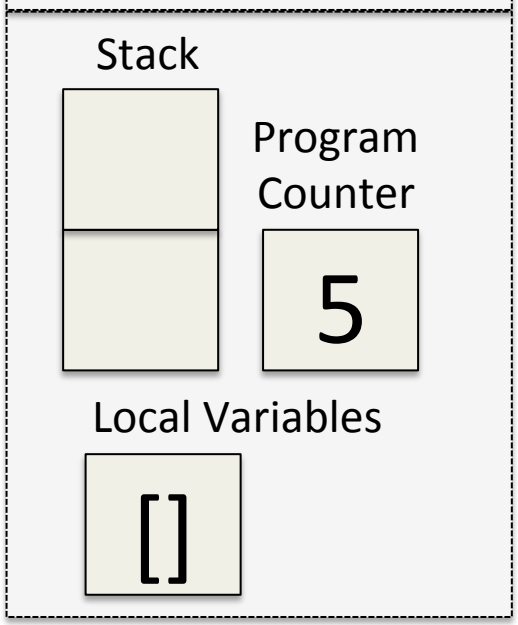

```
public static int add(int, int);
    0: iload_0
    1: iload_1
    2: iadd
    3: ireturn
```



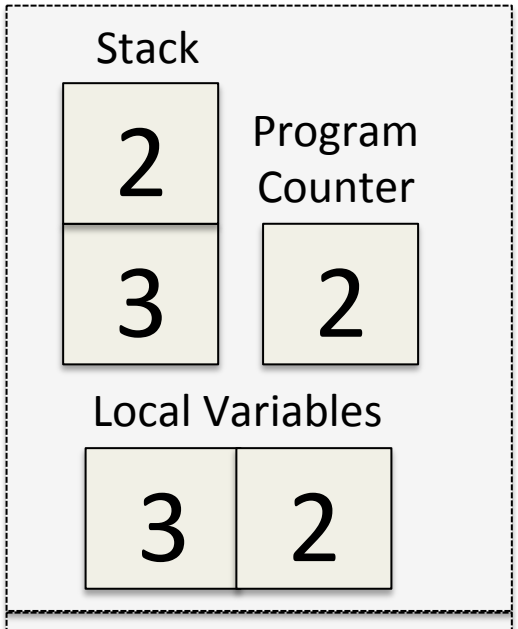

```
public static void main(java.lang.String[]);
    0: iconst_2
    1: iconst_3
    2: invokestatic  #16  // Method add:(II)I
    5: pop
    6: return
```



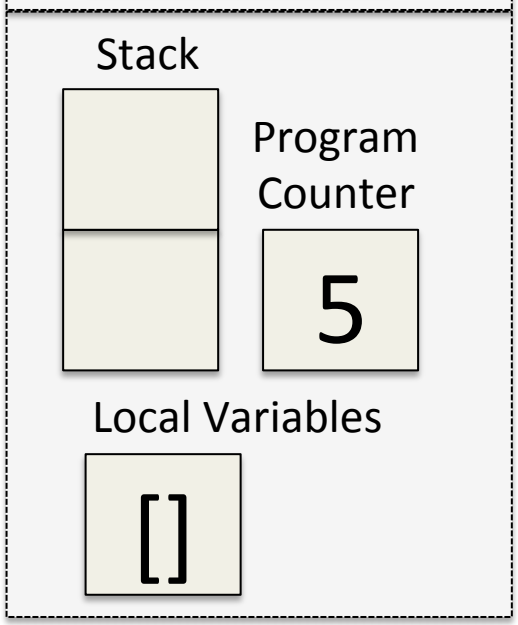

```
public static int add(int, int);
    0: iload_0
    1: iload_1
    2: iadd
    3: ireturn
```



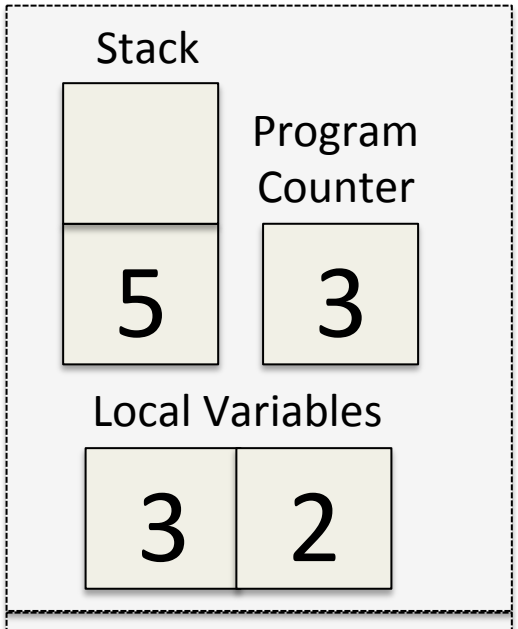

```
public static void main(java.lang.String[]);
    0: iconst_2
    1: iconst_3
    2: invokestatic  #16  // Method add:(II)I
    5: pop
    6: return
```



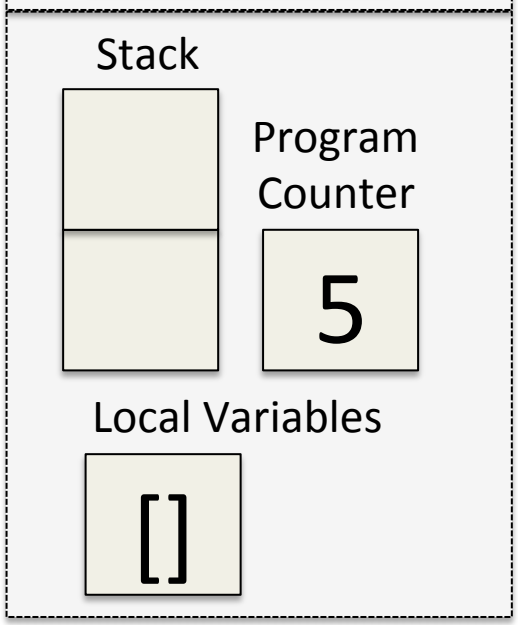

```
public static int add(int, int);
    0: iload_0
    1: iload_1
    2: iadd
    3: ireturn
```



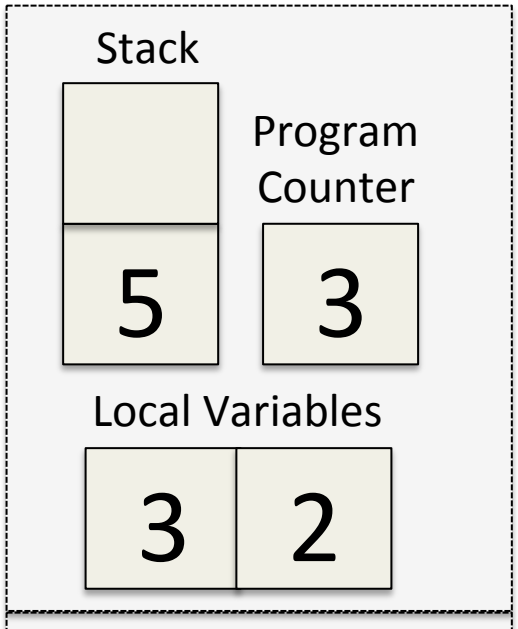

```
public static void main(java.lang.String[]);
    0: iconst_2
    1: iconst_3
    2: invokestatic  #16  // Method add:(II)I
    5: pop
    6: return
```



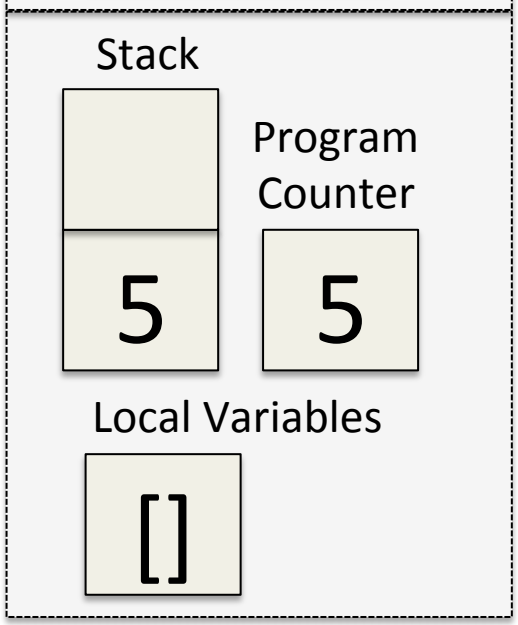

```
public static int add(int, int);
    0: iload_0
    1: iload_1
    2: iadd
    3: ireturn
```



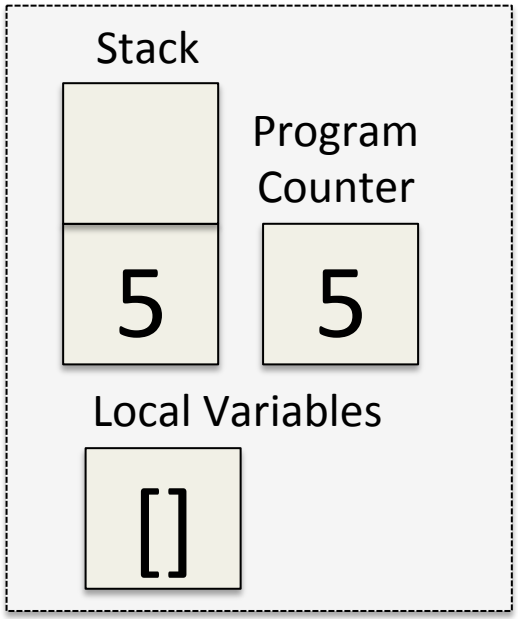

```
public static void main(java.lang.String[]);
    0: iconst_2
    1: iconst_3
    2: invokestatic  #16  // Method add:(II)I
    5: pop
    6: return
```



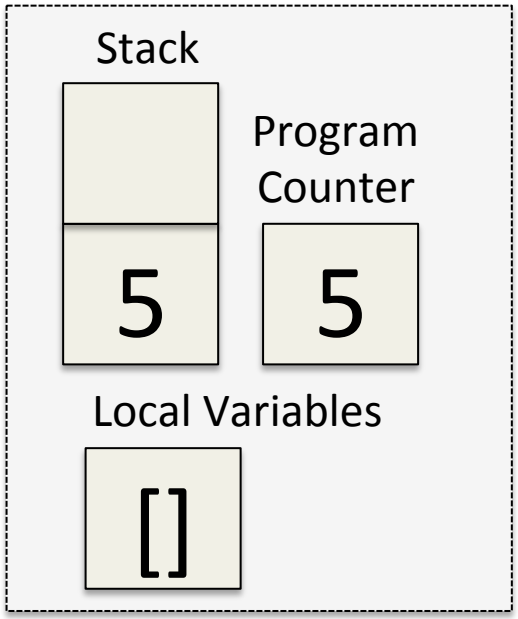

```
public static int add(int, int);
    0: iload_0
    1: iload_1
    2: iadd
    3: ireturn
```



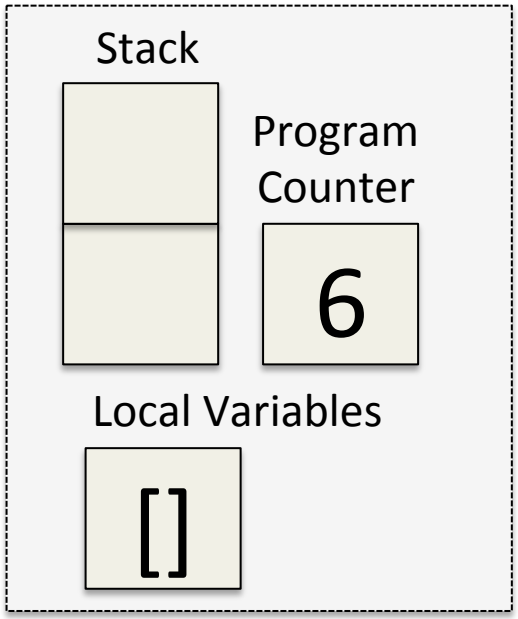

```
public static void main(java.lang.String[]);
    0: iconst_2
    1: iconst_3
    2: invokestatic  #16  // Method add:(II)I
    5: pop
    6: return
```




```
public static void main(java.lang.String[]);
    0: iconst_2
    1: iconst_3
    2: invokestatic  #16  // Method add:(II)I
    5: pop
    6: return
```




```
public static void main(java.lang.String[]);
    0: iconst_2
    1: iconst_3
    2: invokestatic  #16  // Method add:(II)I
    5: pop
    6: return
```




```
public static void main(java.lang.String[]);
    0: iconst_2
    1: iconst_3
    2: invokestatic  #16  // Method add:(II)I
    5: pop
    6: return
```

Instance Methods

- Instance methods called on a specific object
  - That object must be on the top of the stack
  - `this` pointer stored to local variable zero
- Different semantics from static method calls
  - Uses INVOKEVIRTUAL
- We'll later see other invoke instructions
  - INVOKESPECIAL
  - INVOKEDYNAMIC

Dynamic Dispatch

- Java supports single dynamic dispatch
  - Method called depends on the type of the object
  - Which method is determined at run time
  - One call site can call different methods
- Some languages support multiple dispatch (Lisp)
  - The method depends on the parameters too
  - The visitor pattern simulates it elsewhere

Method Lookup

- Determine the type of the object
  - The object's header has a pointer to the class
- If that type defines a matching method, call it
- If not, check the super type of the object
- Continue until you find the method
- If you don't find the method, there's a problem
  - Throw an `AbstractMethodError`

Flow Control

- Several types of jumps and branches

Flow Control

- Several types of jumps and branches

- Unconditional
  - `GOTO`
- Compare with zero
  - `IFEQ, IFNE, IFGT, etc`
- Compare int values directly
  - `IF_ICMPEQ, IF_ICMPGT`

Flow Control

- Several types of jumps and branches

- Unconditional
  - `GOTO`
- Compare with zero
  - `IFEQ, IFNE, IFGT, etc`
- Compare int values directly
  - `IF_ICMPEQ, IF_ICMPGT`
- Compare to null
  - `IFNULL, IFNONNULL`
- Compare references
  - `IF_ACMPEQ, IF_ACMPNE`

Flow Control

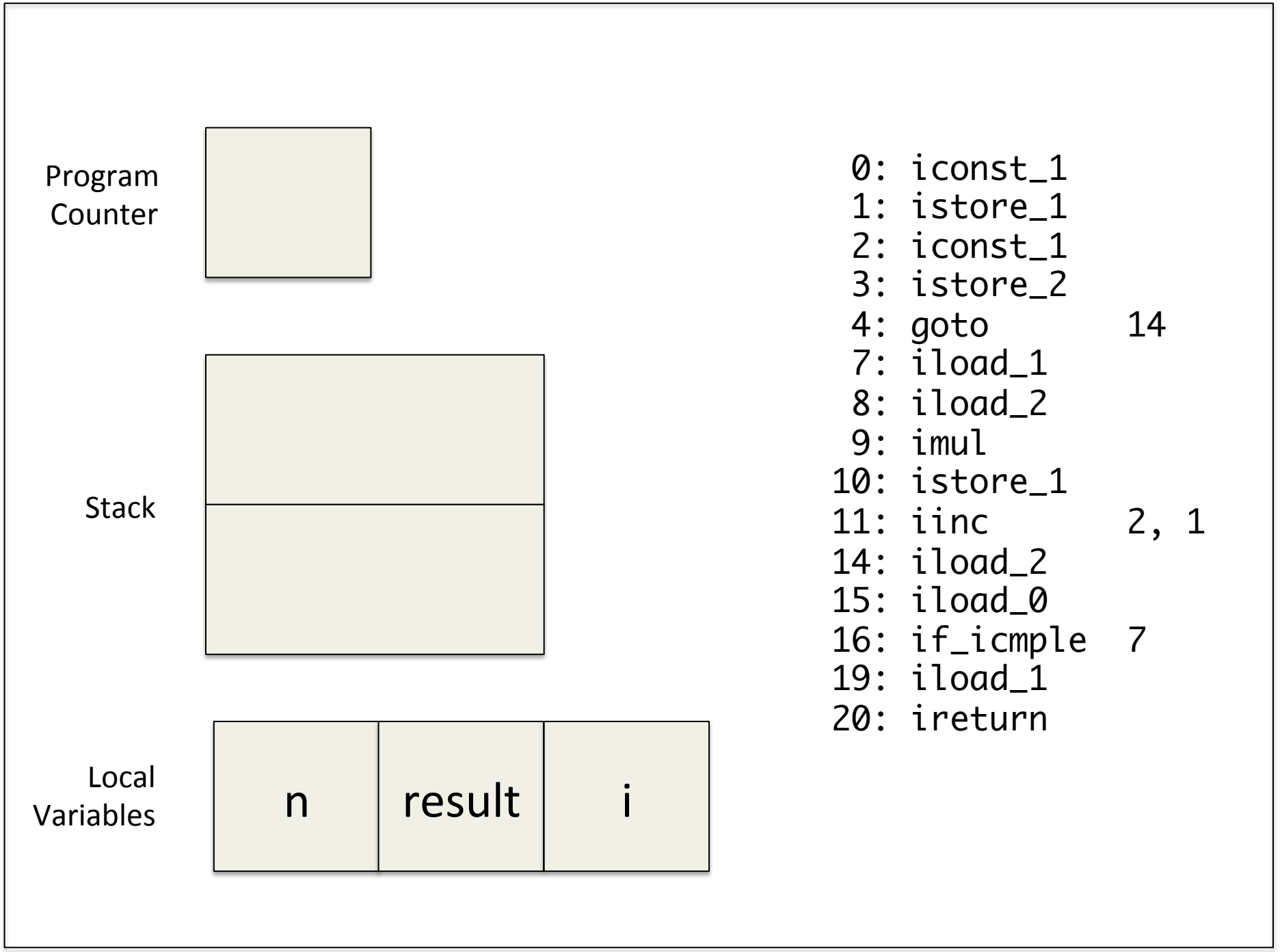
- Several types of jumps and branches

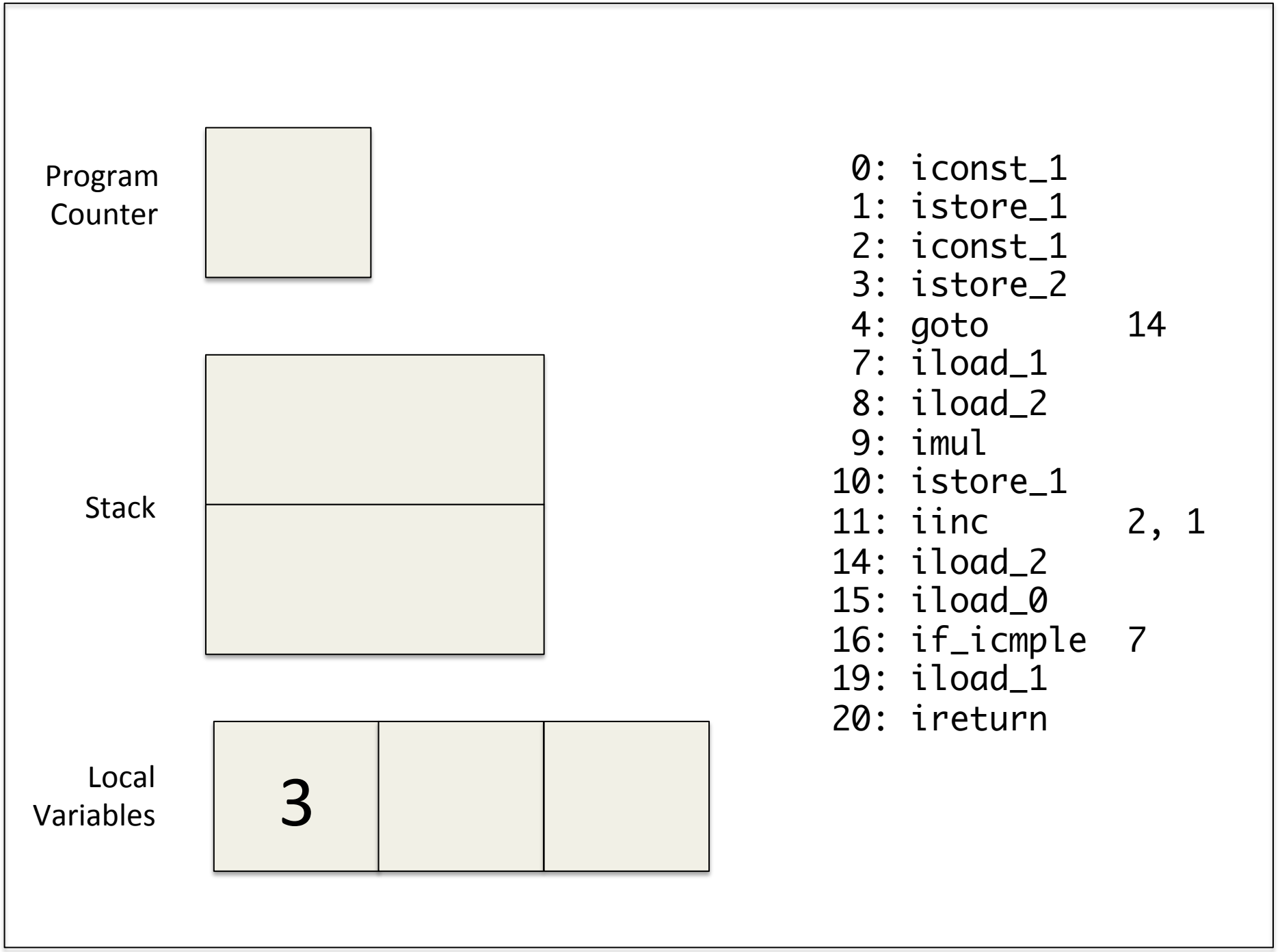
- Unconditional
  - `GOTO`
- Compare with zero
  - `IFEQ, IFNE, IFGT, etc`
- Compare int values directly
  - `IF_ICMPEQ, IF_ICMPGT`
- Compare to null
  - `IFNULL, IFNONNULL`
- Compare references
  - `IF_ACMPEQ, IF_ACMPNE`
- Compare other types
  - `FCMPEQ, FCMPLT`
  - `LCMP`

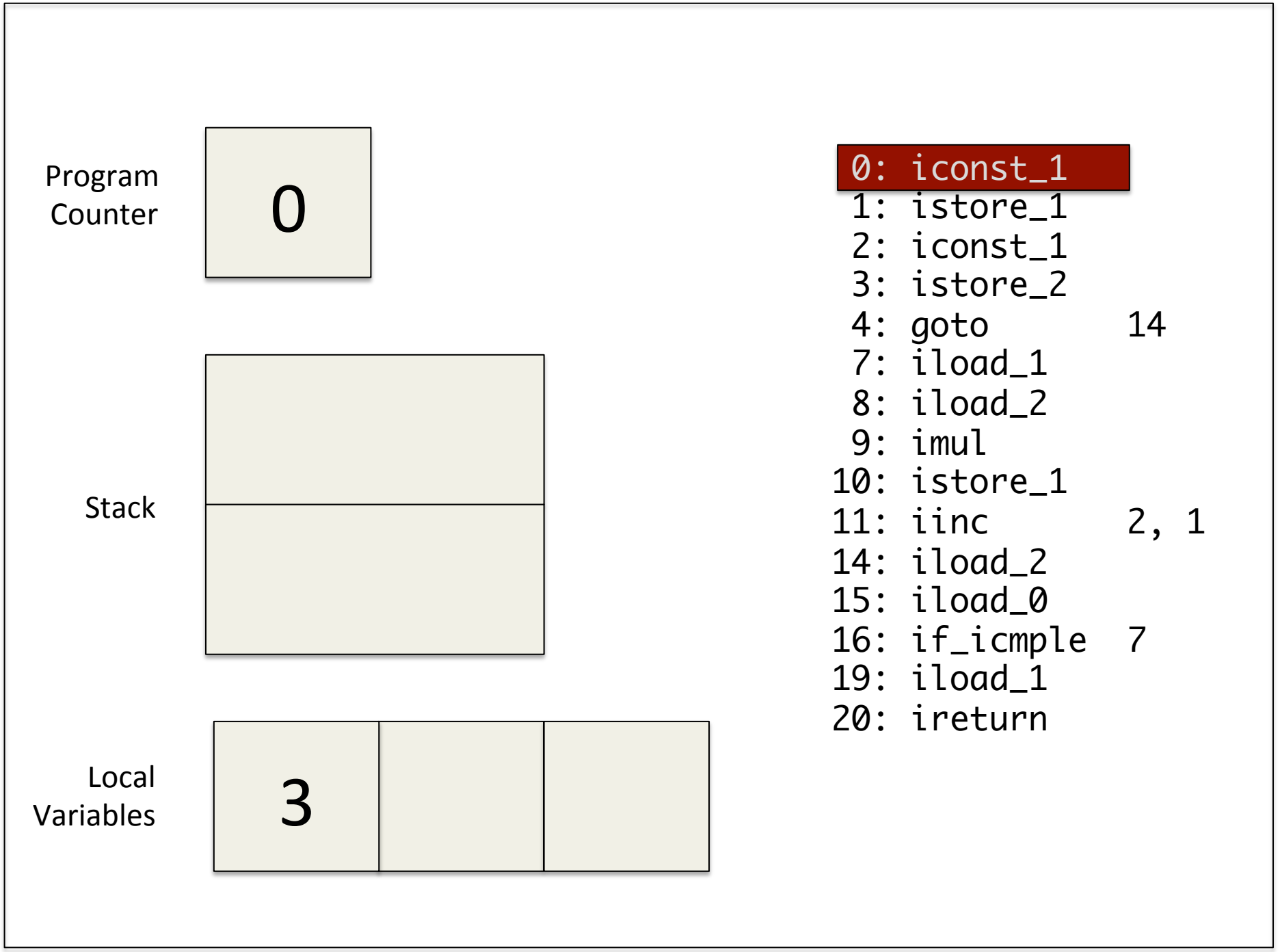
```java
public static int factorial(final int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
      result = result * i;
    }
    return result;
  }
```

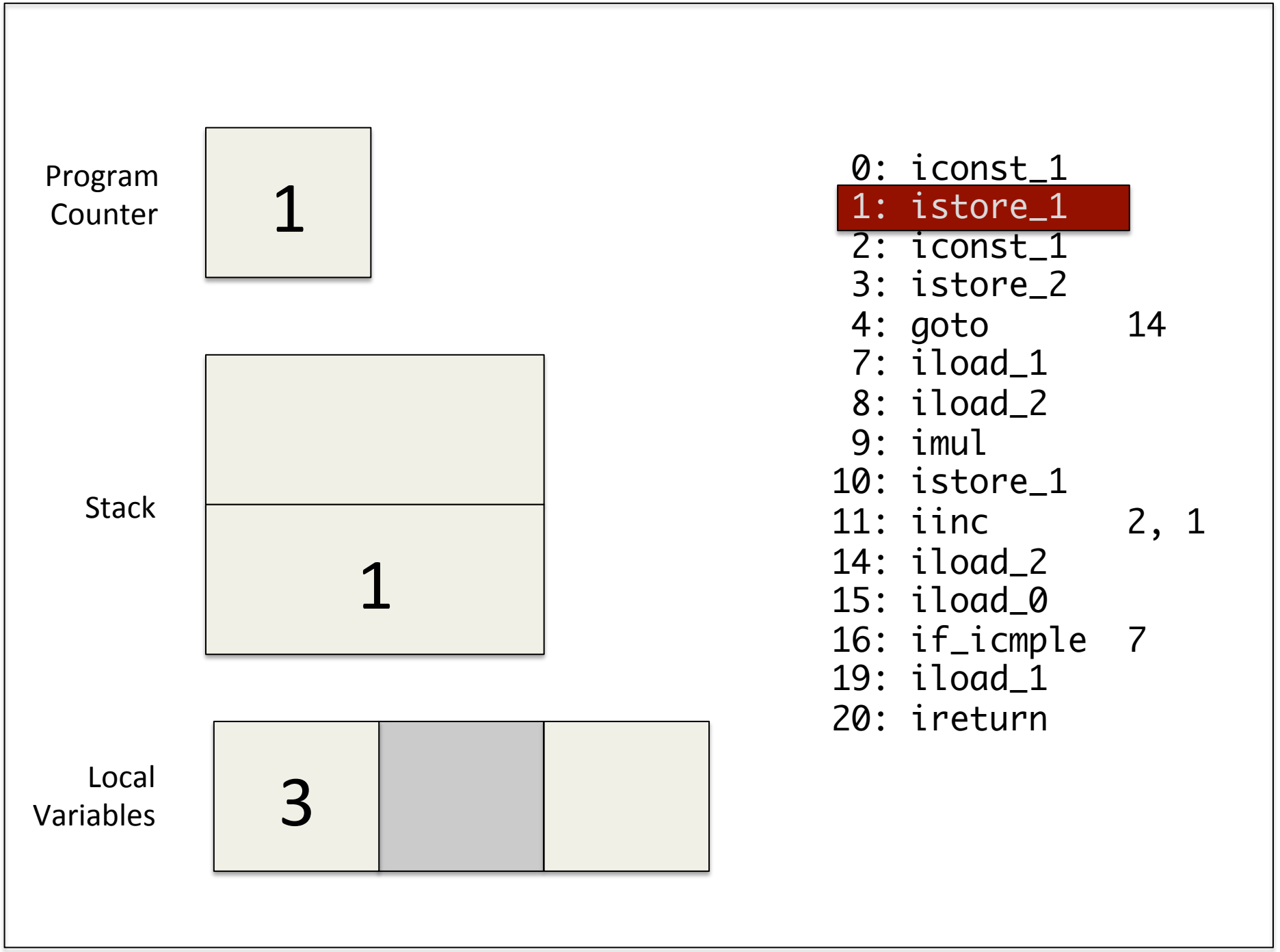
```java
public static int factorial(final int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
      result = result * i;
    }
    return result;
  }
```

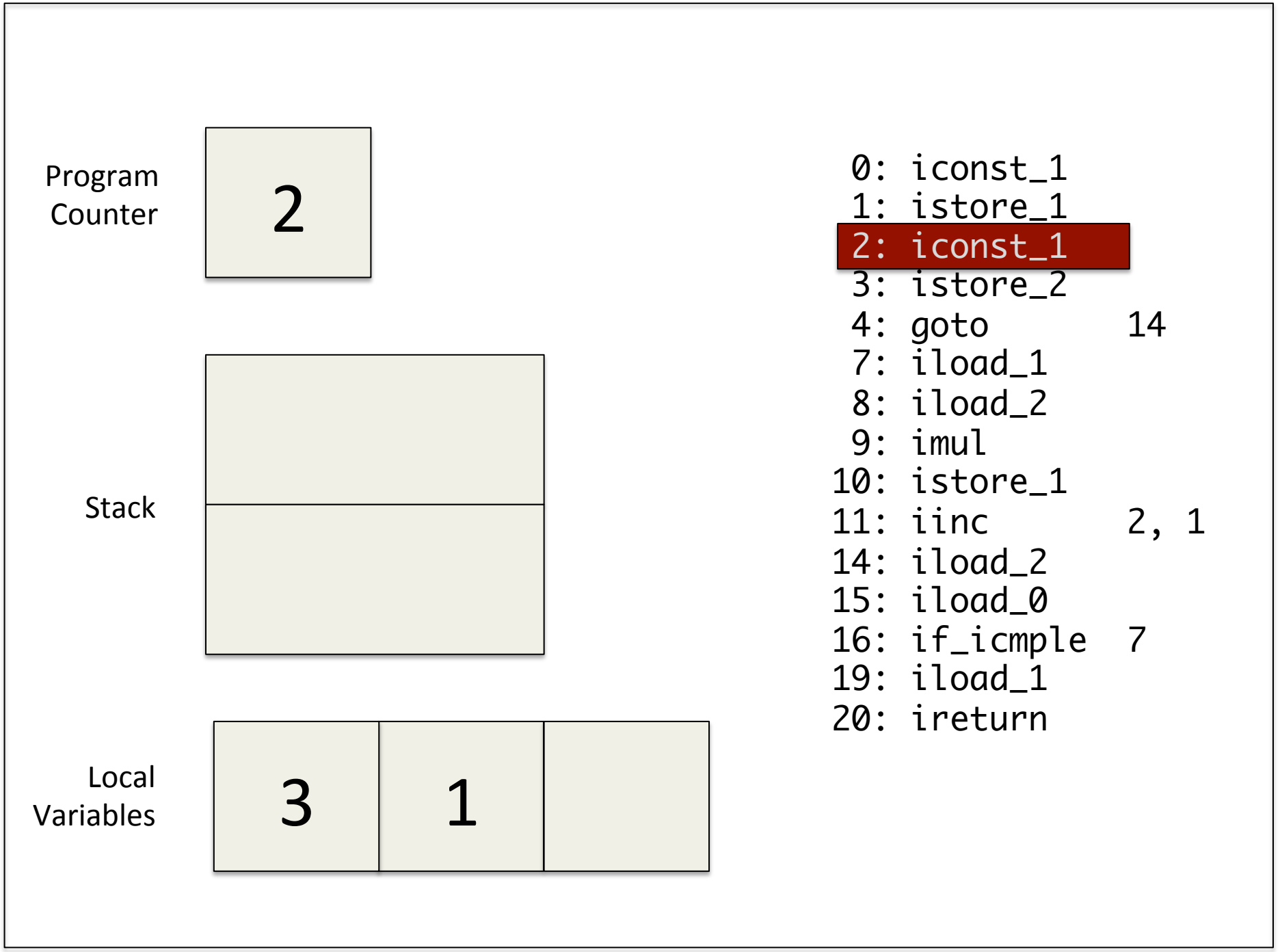
```
 0: iconst_1
 1: istore_1
 2: iconst_1
 3: istore_2
 4: goto        14
 7: iload_1
 8: iload_2
 9: imul
10: istore_1
11: iinc        2, 1
14: iload_2
15: iload_0
16: if_icmple   7
19: iload_1
20: ireturn
```

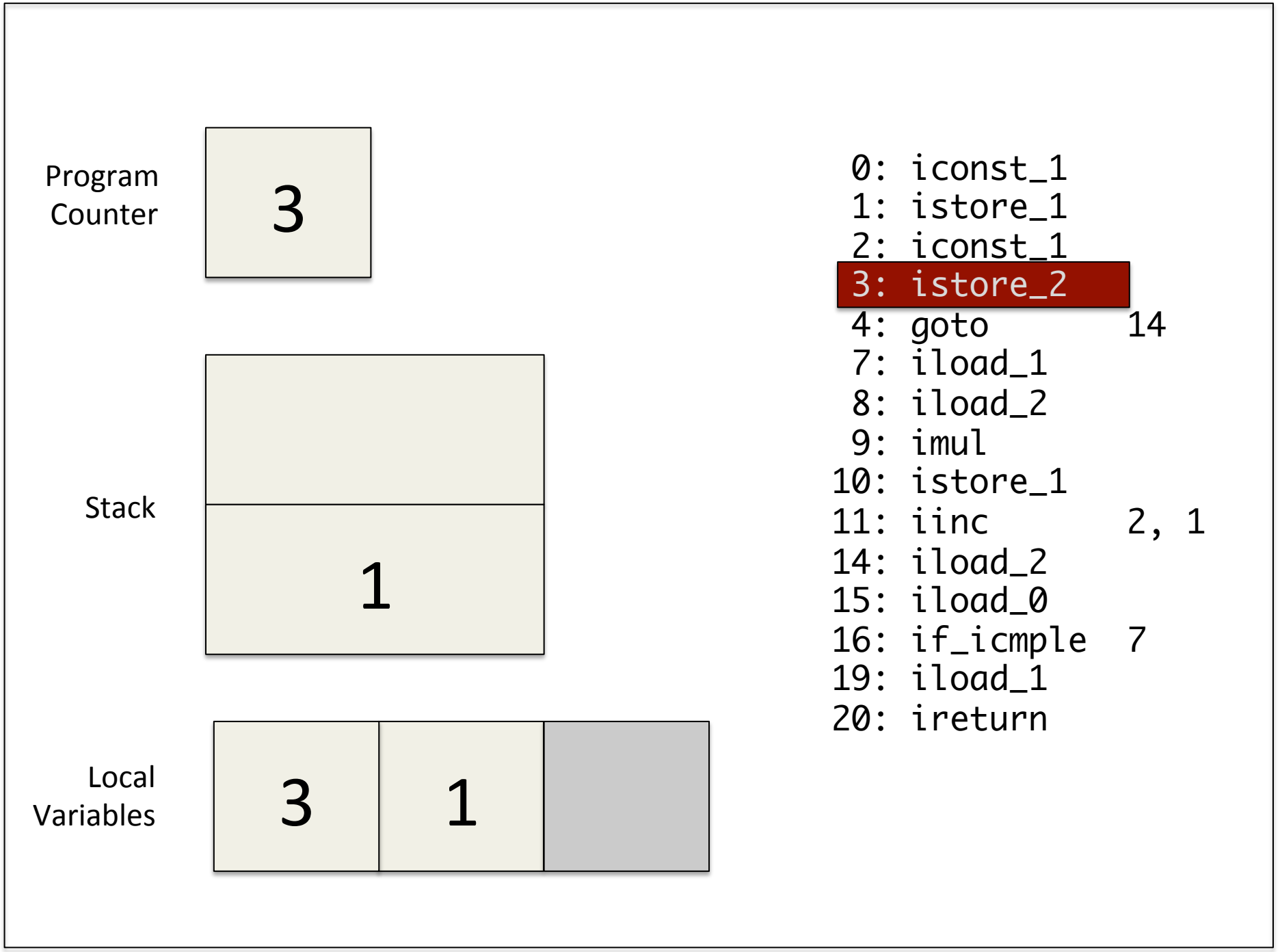

Program Counter
Stack
Local Variables
n
result
i
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn

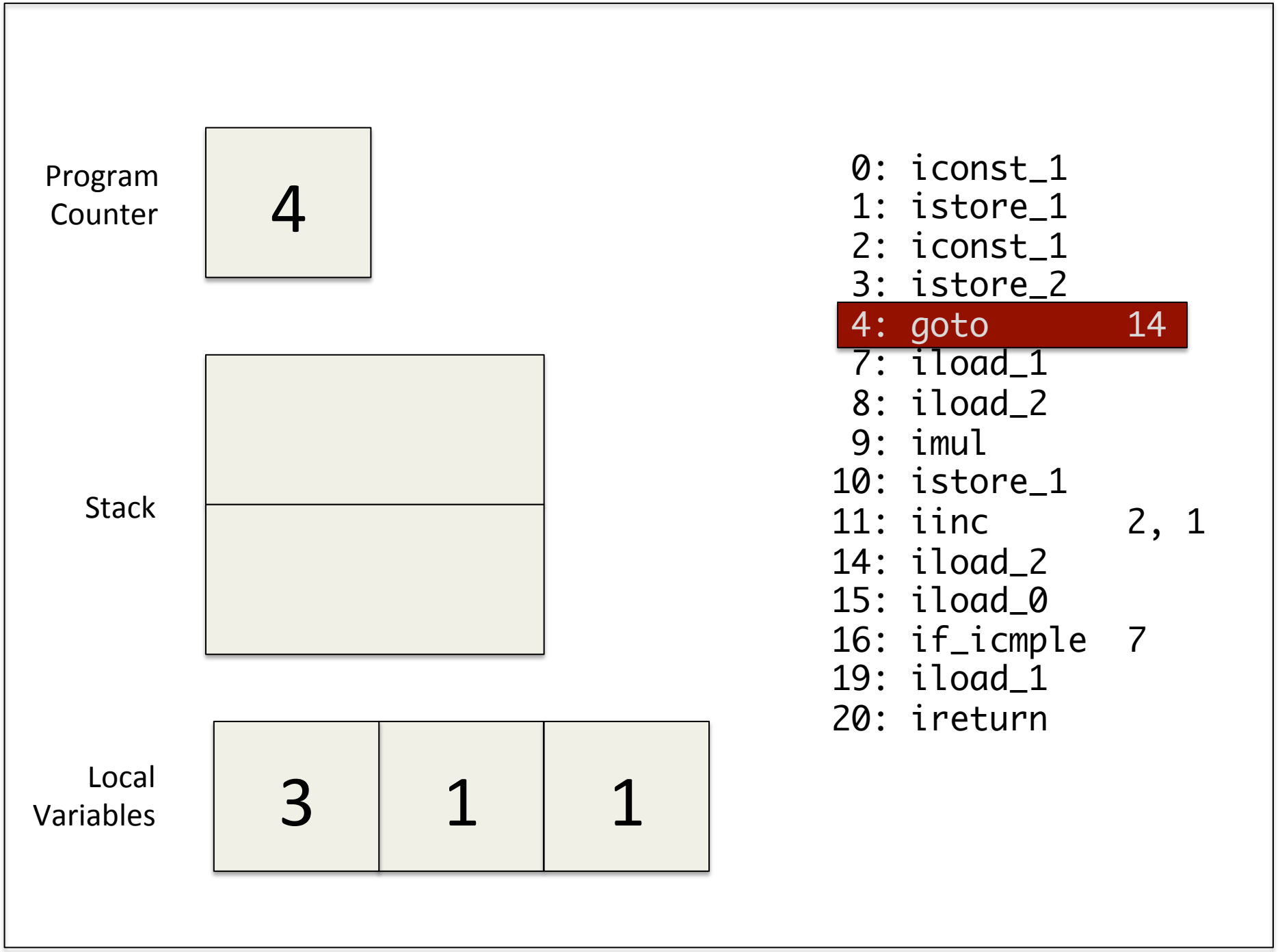

Program Counter
Stack
Local Variables
3
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn

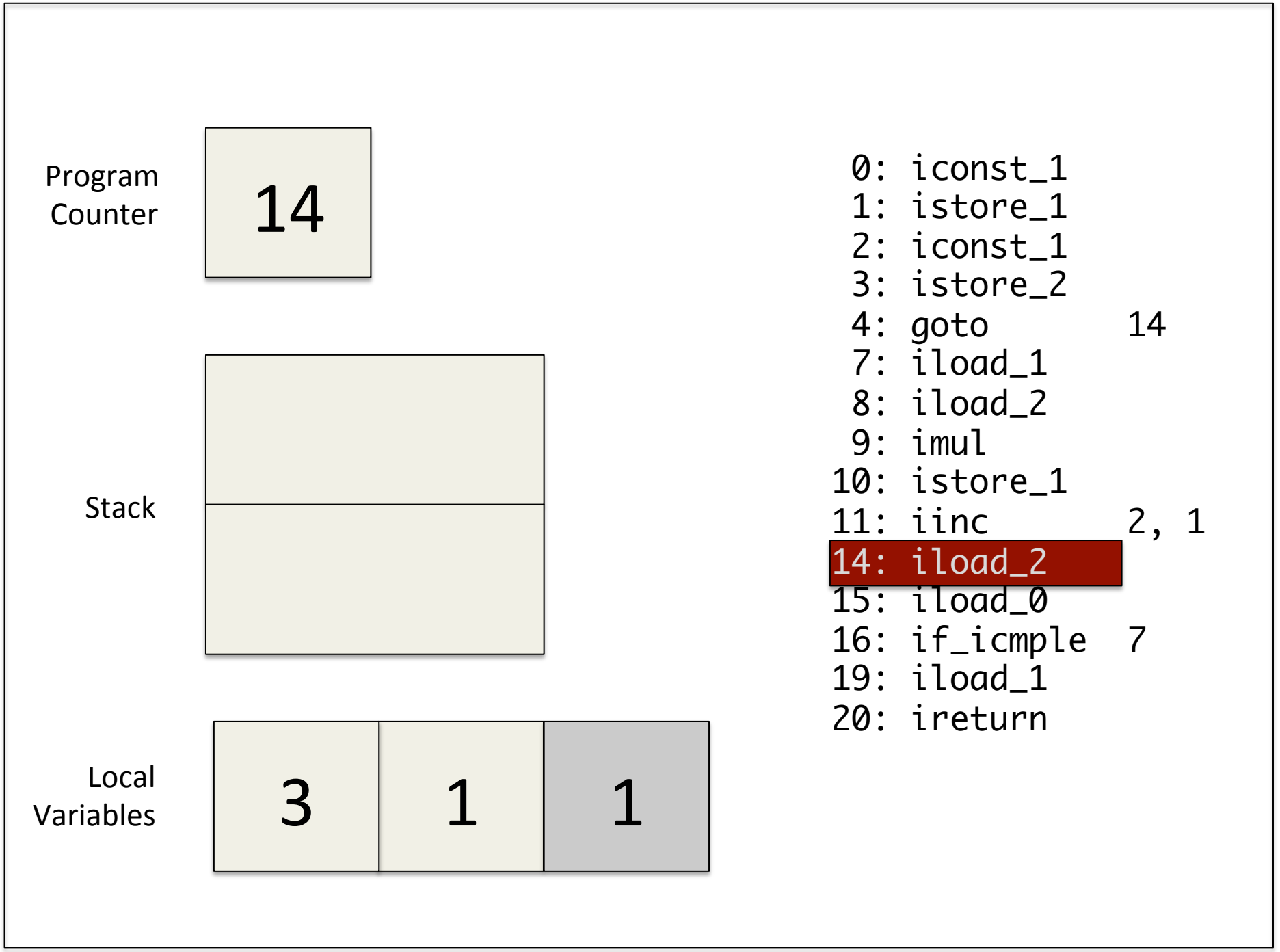

Program Counter
0
Stack
Local Variables
3
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn

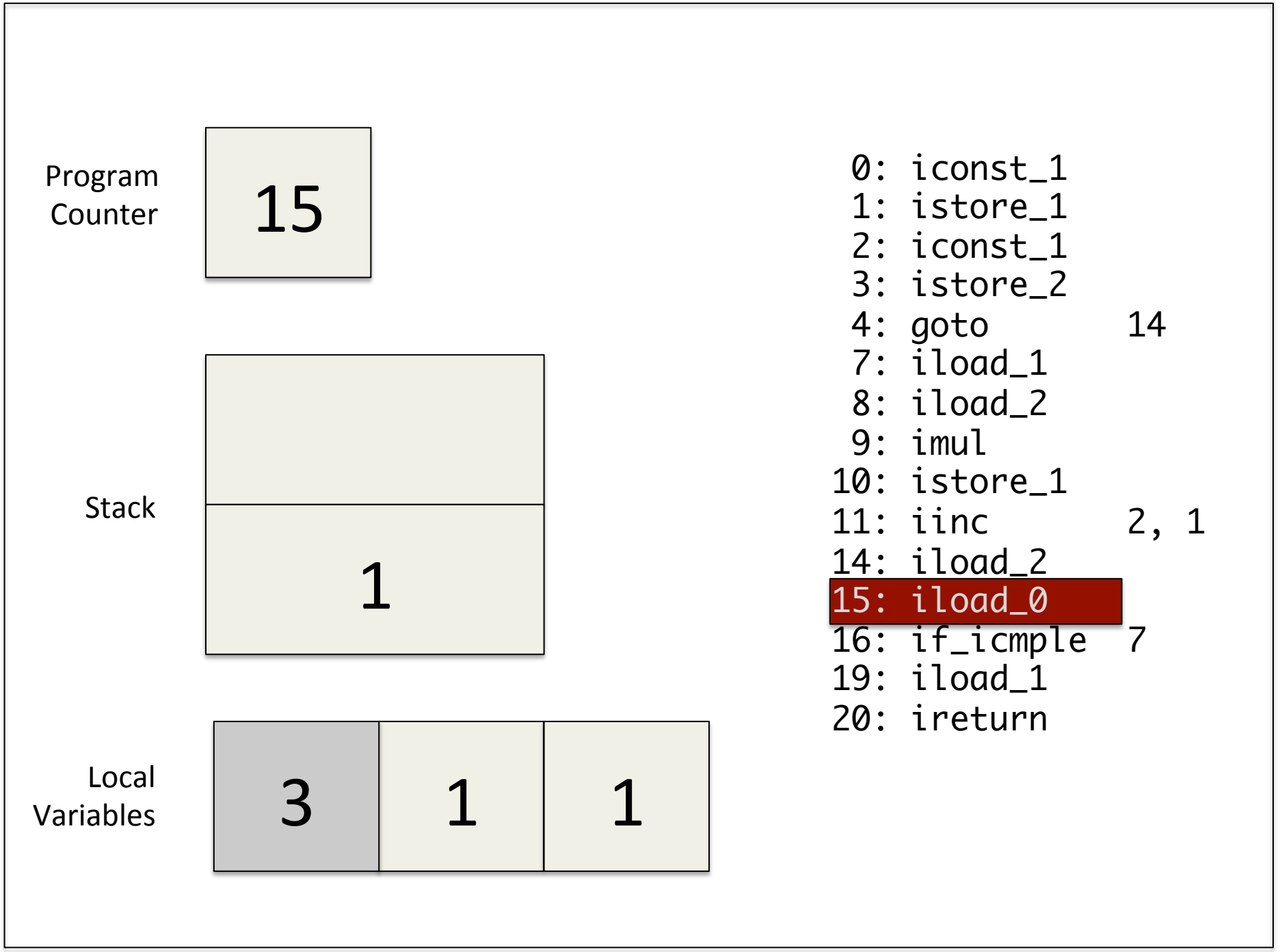

Program Counter
1
Stack
1
Local Variables
3
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn

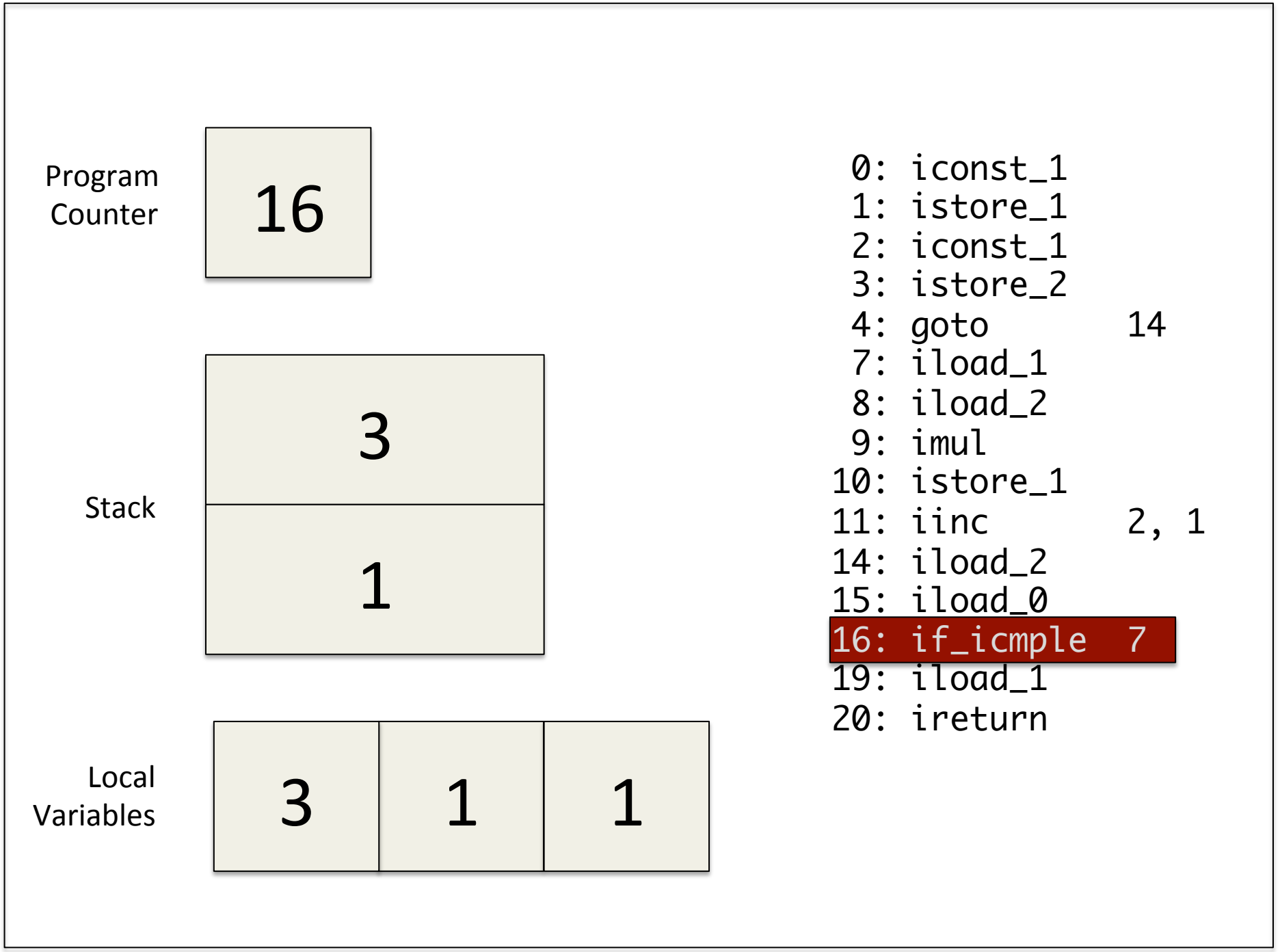

Program Counter
2
Stack
Local Variables
3
1
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn

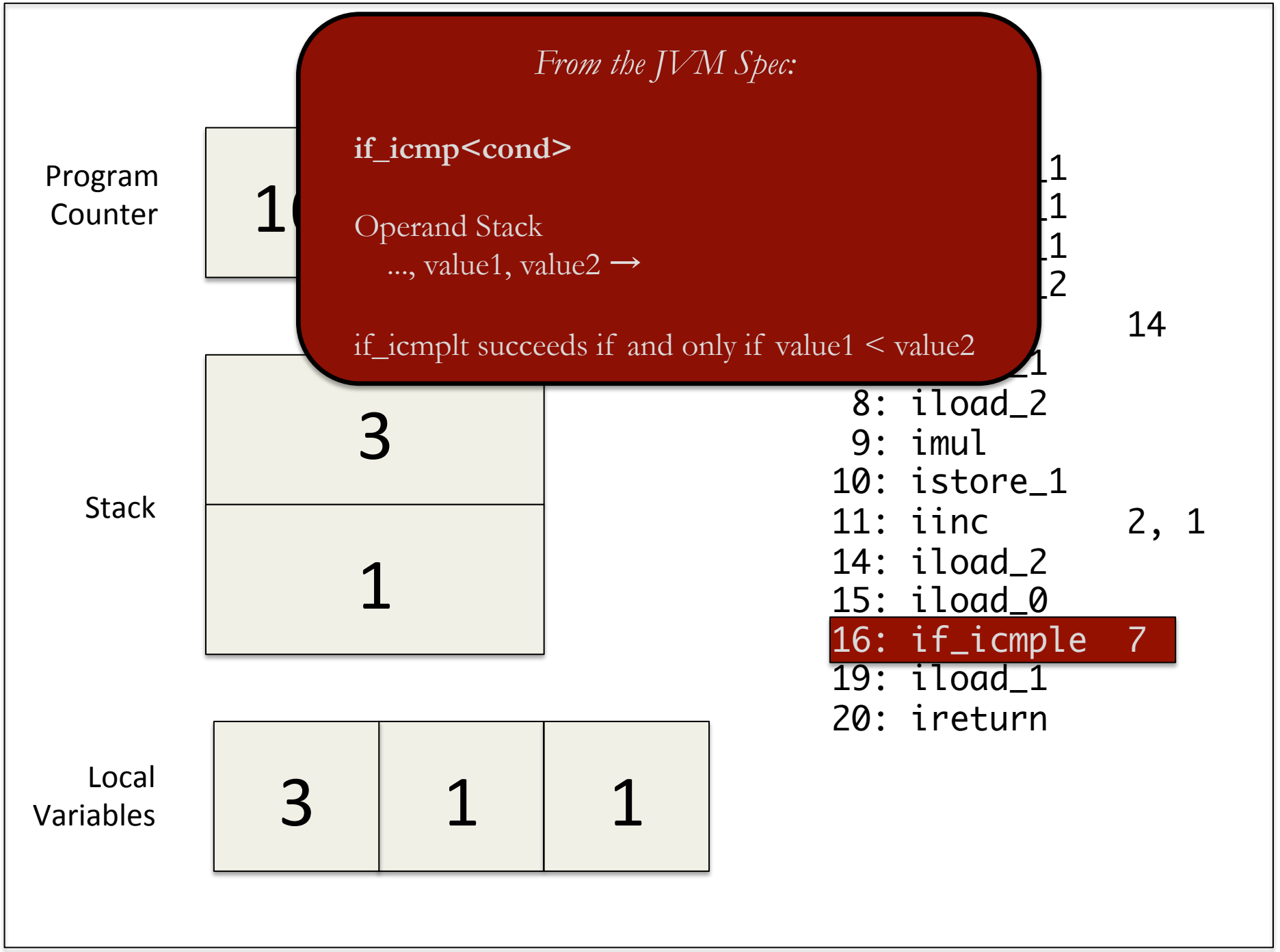

Program Counter
3
Stack
1
Local Variables
3
1
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn

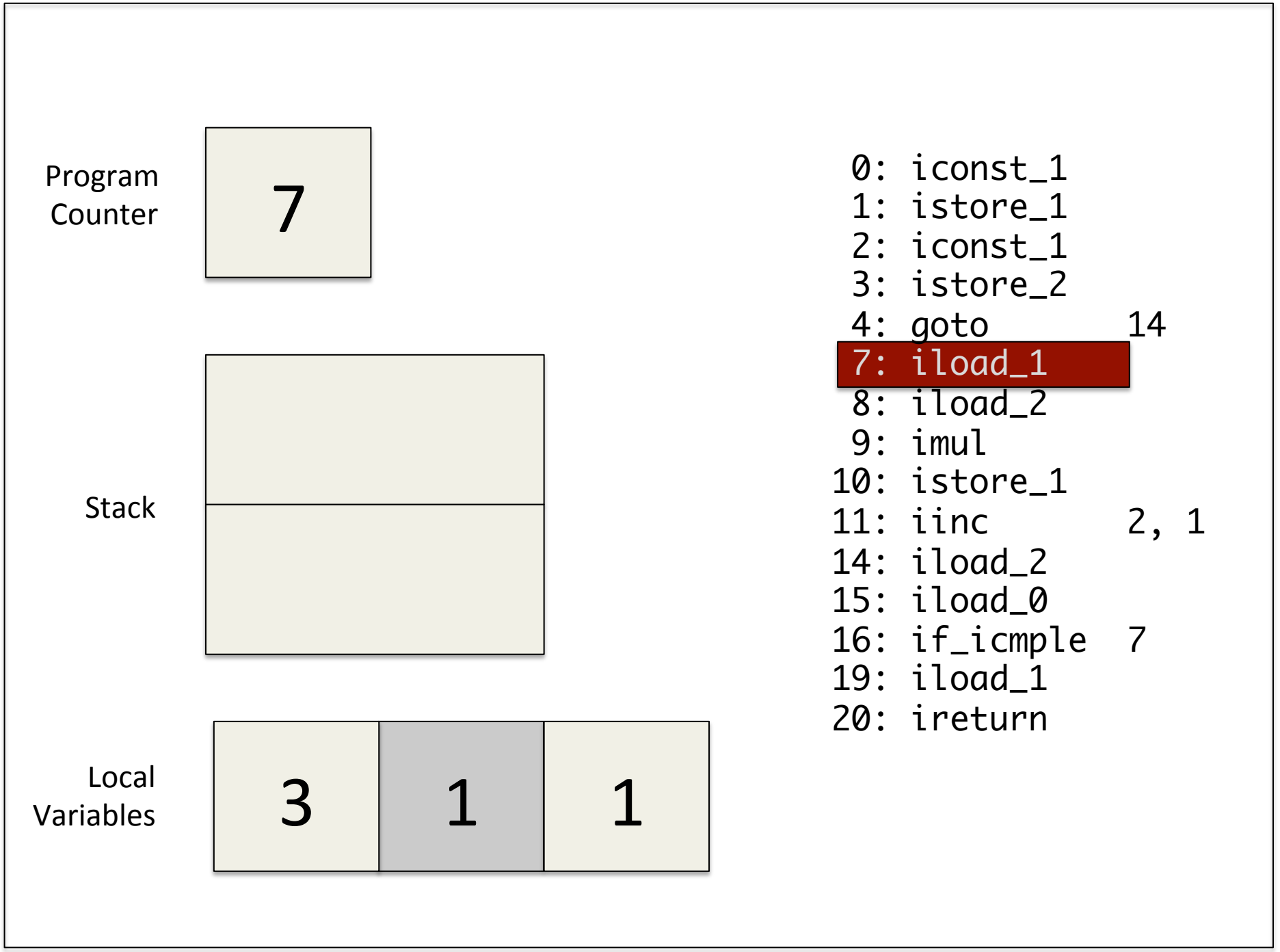

Program Counter
4
Stack
Local Variables
3
1
1
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn

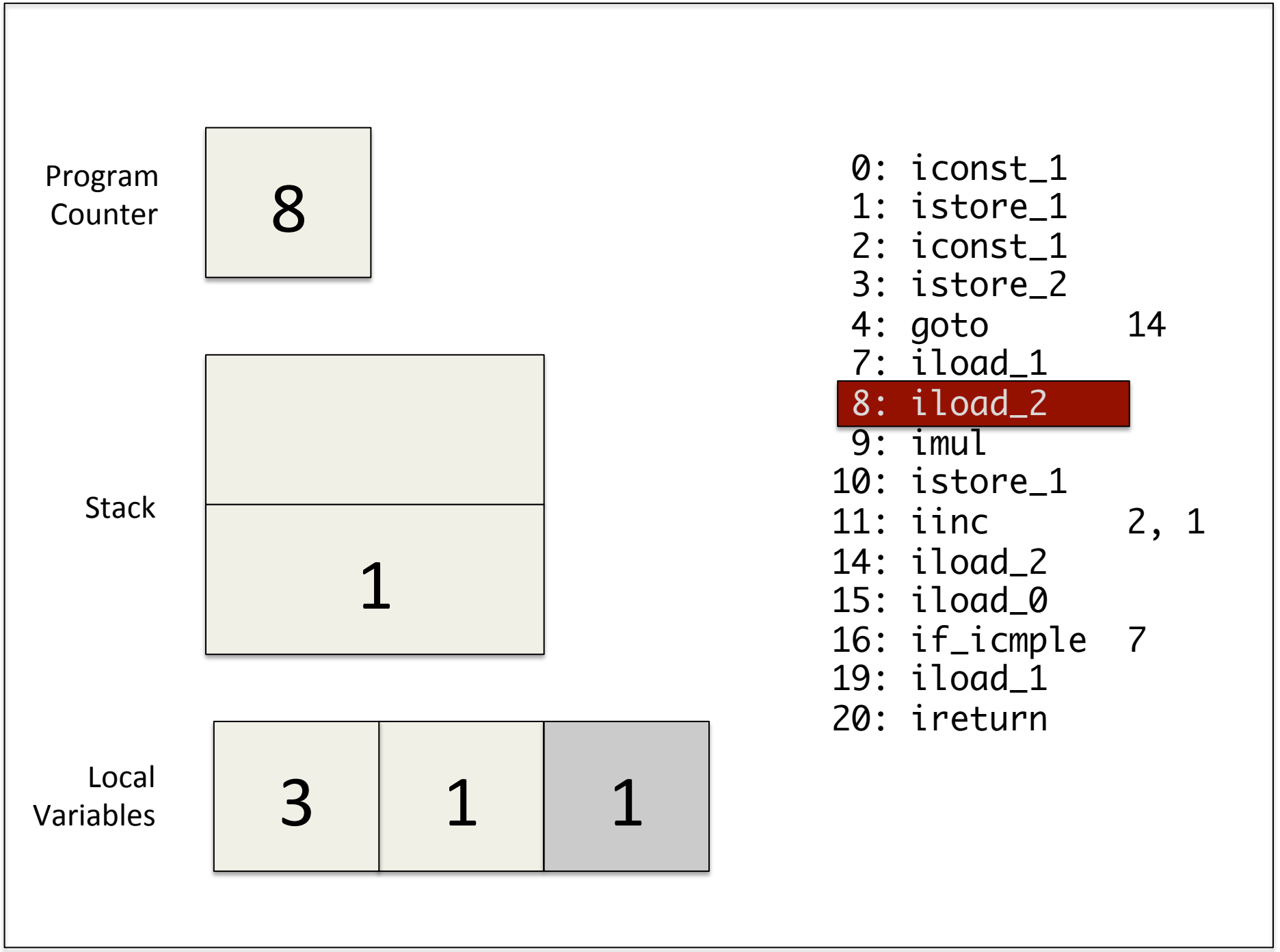

Program Counter
14
Stack
Local Variables
3
1
1
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn

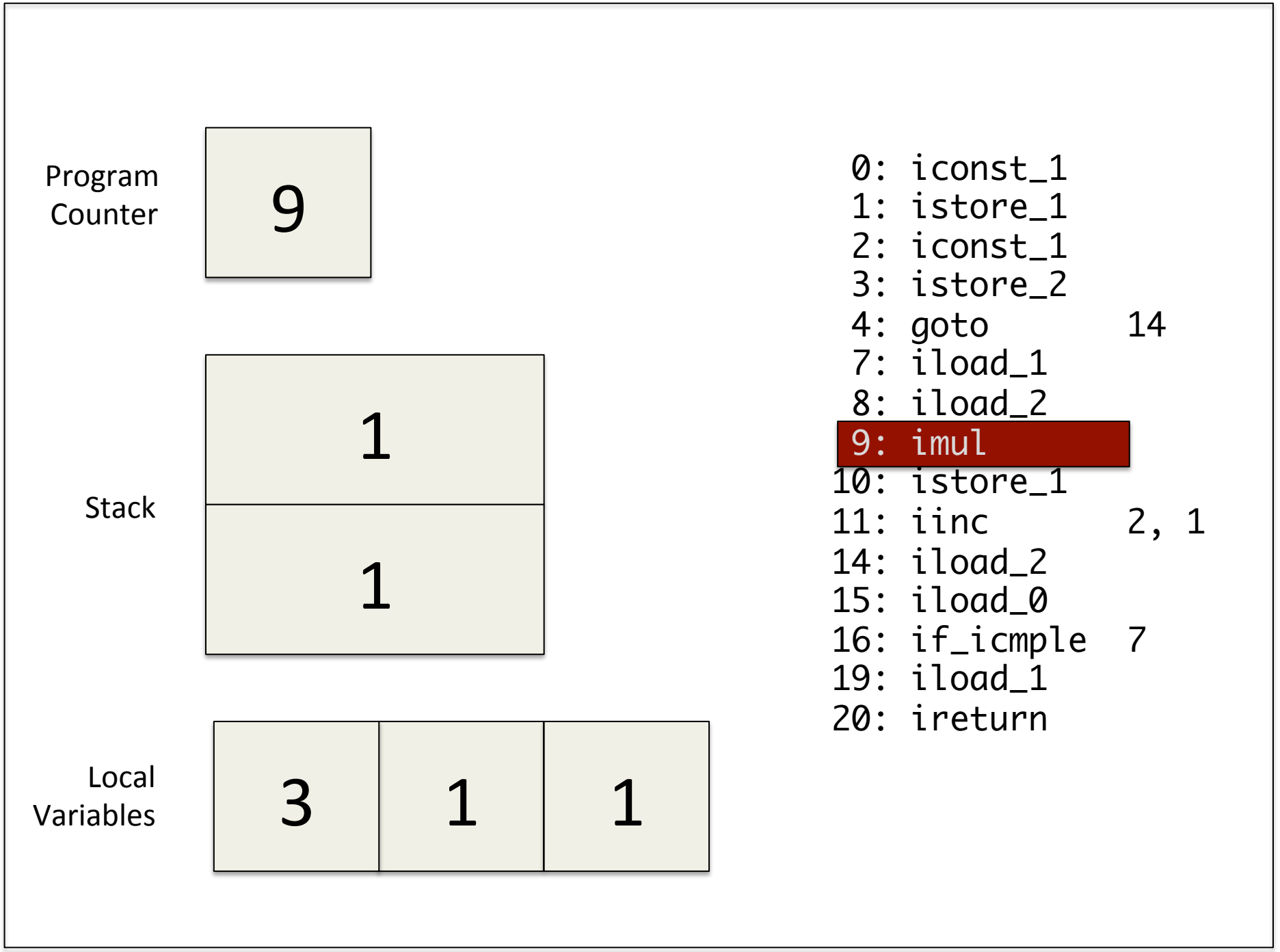

Program Counter
15
Stack
1
Local Variables
3
1
1
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn

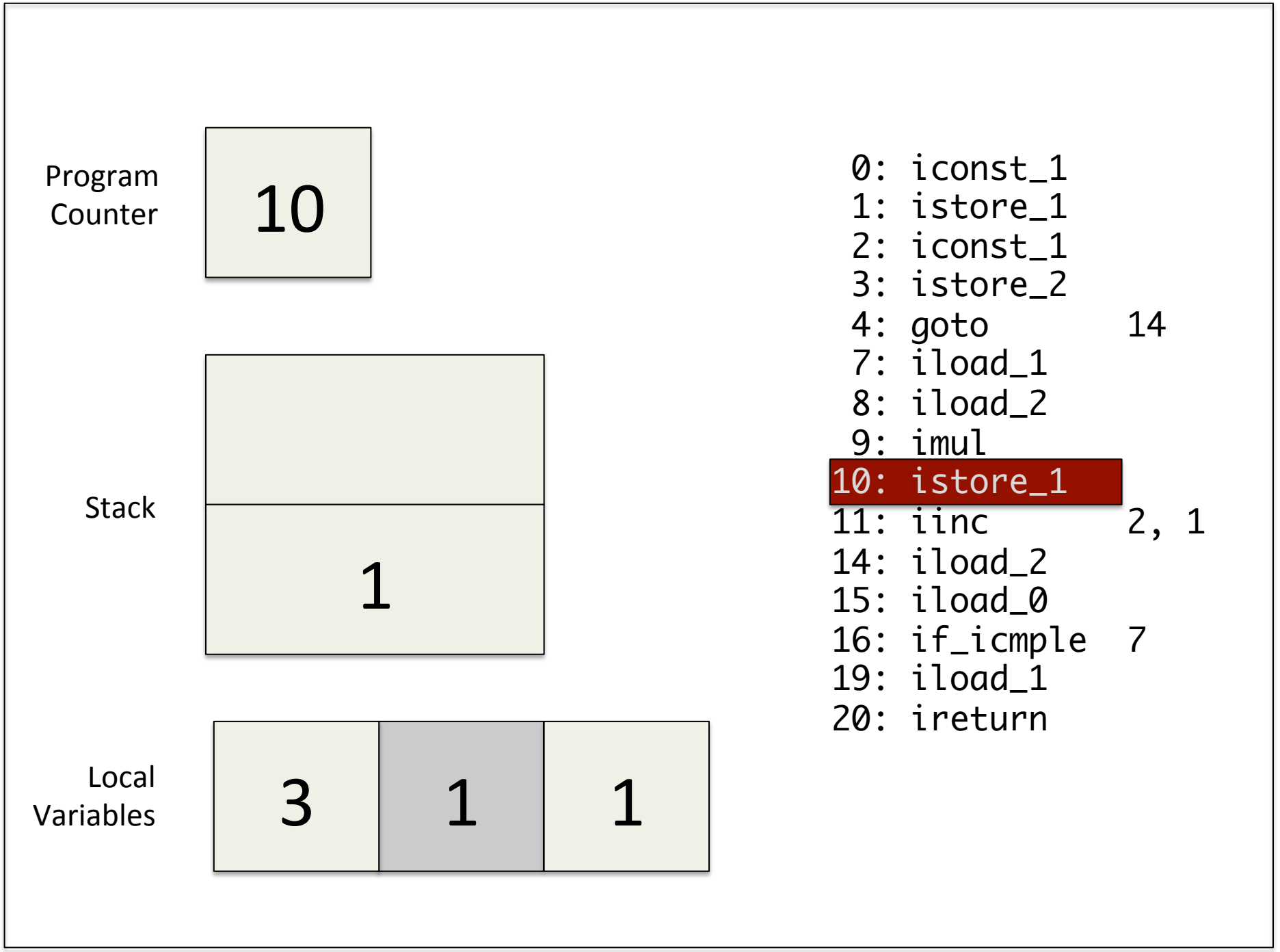

Program Counter
16
Stack
3
1
Local Variables
3
1
1
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn

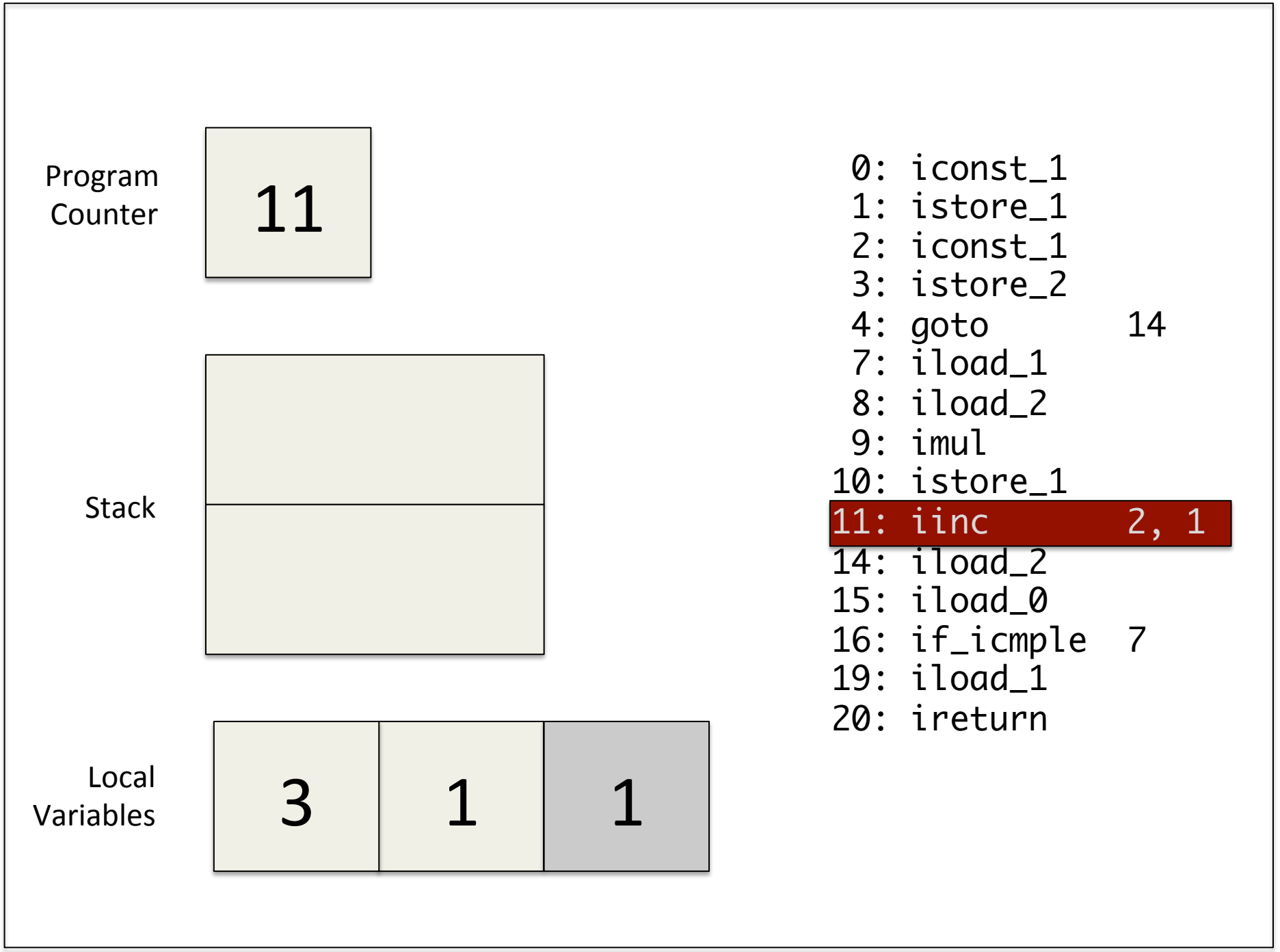

From the JVM Spec:
if_icmp<cond>
Operand Stack
..., value1, value2 →
if_icmplt succeeds if and only if value1 < value2
Program Counter
Stack
3
1
Local Variables
3
1
1
14
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn

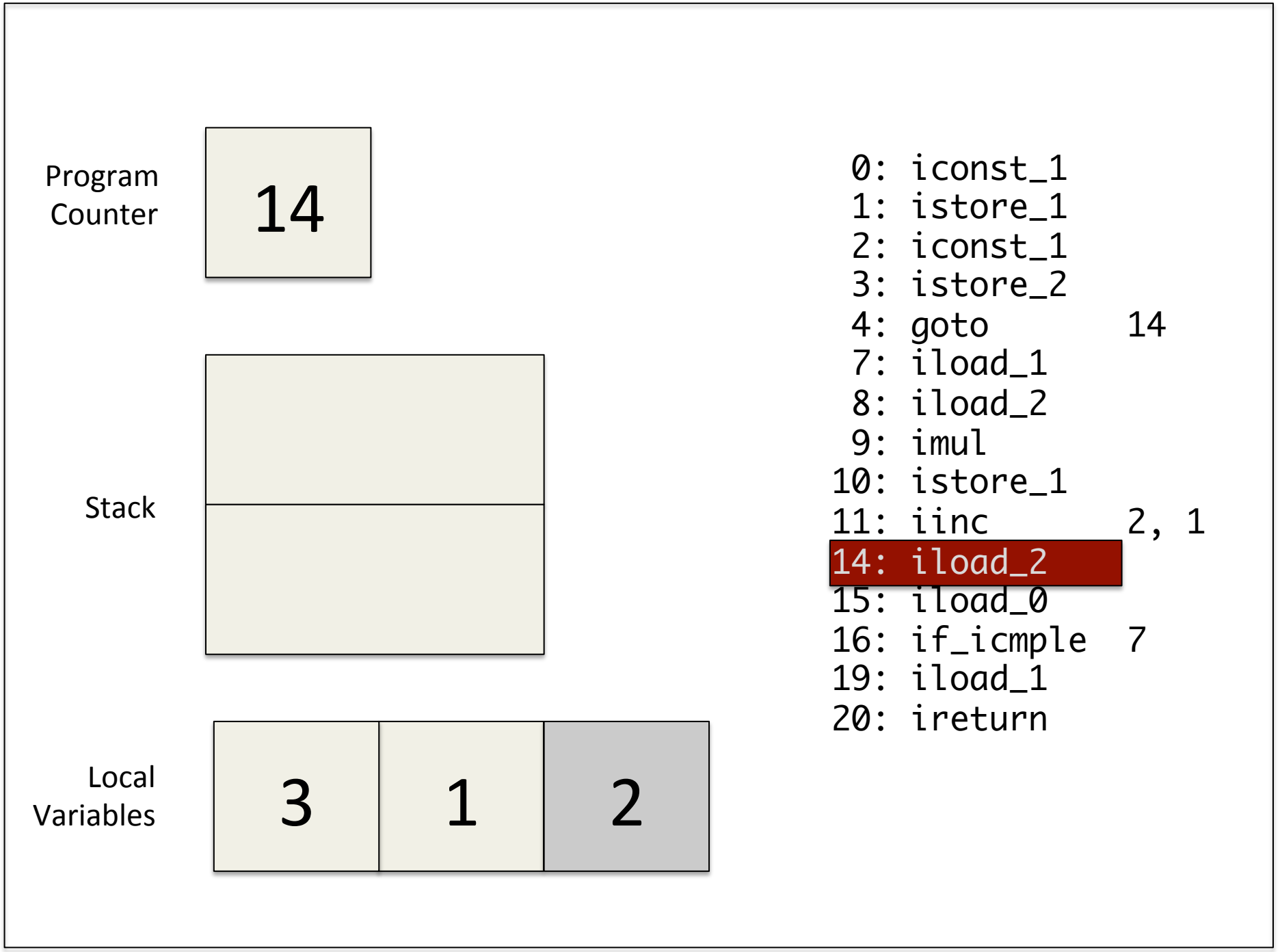

Program Counter
7
Stack
Local Variables
3
1
1
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn

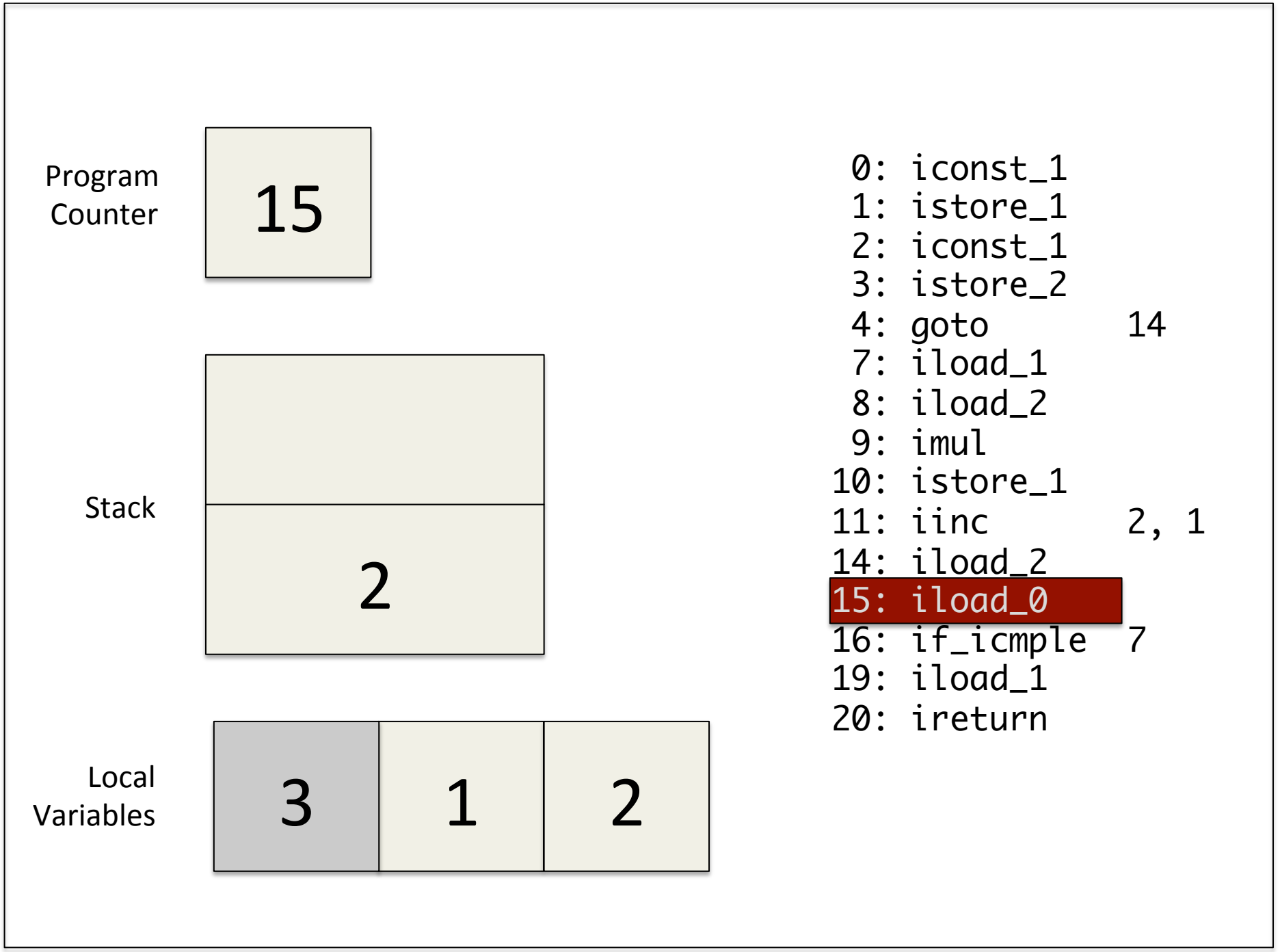

Program Counter
8
Stack
1
Local Variables
3
1
1
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn

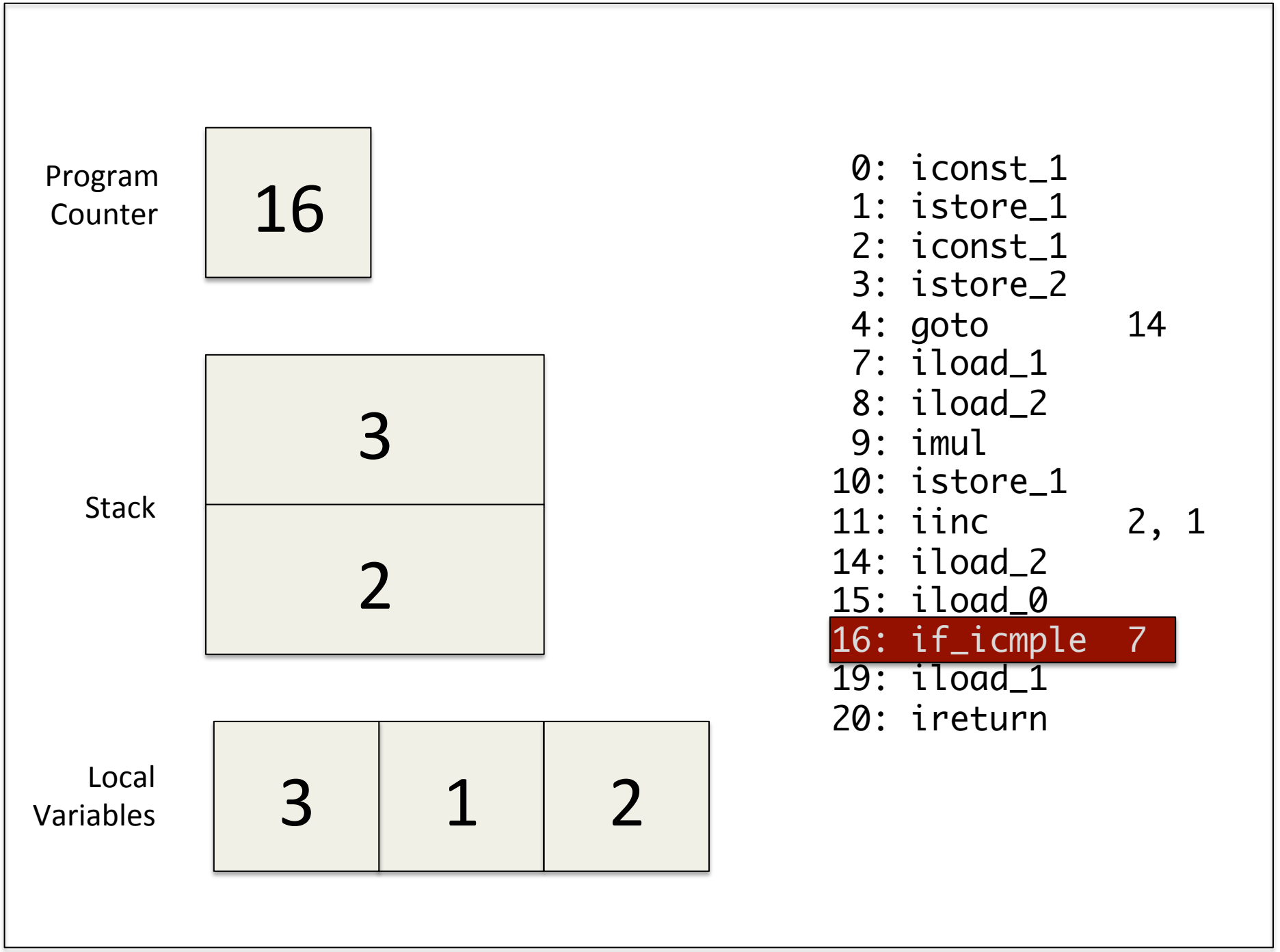

Program Counter
9
Stack
1
1
Local Variables
3
1
1
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn

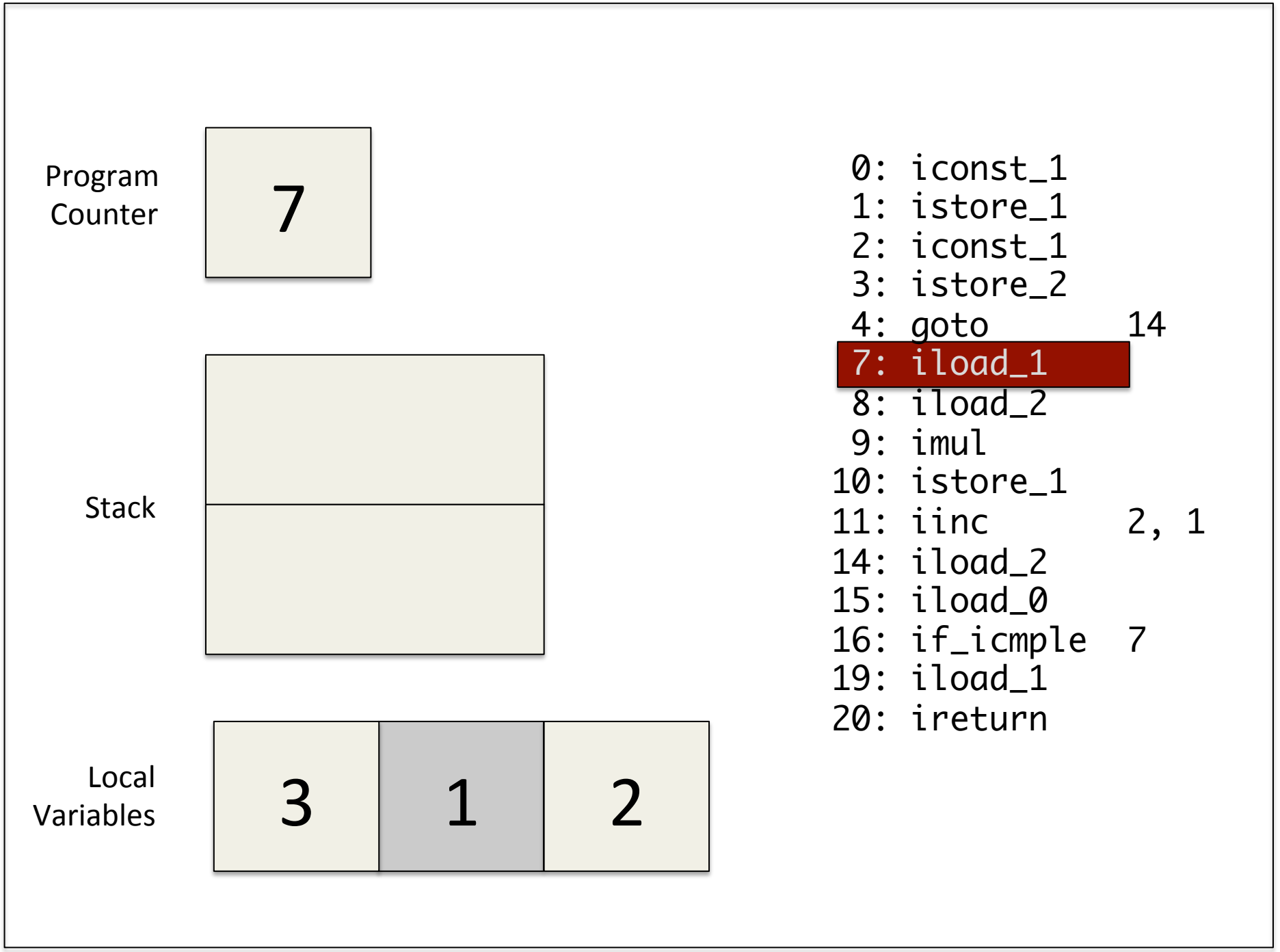

Program Counter
10
Stack
1
Local Variables
3
1
1
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn

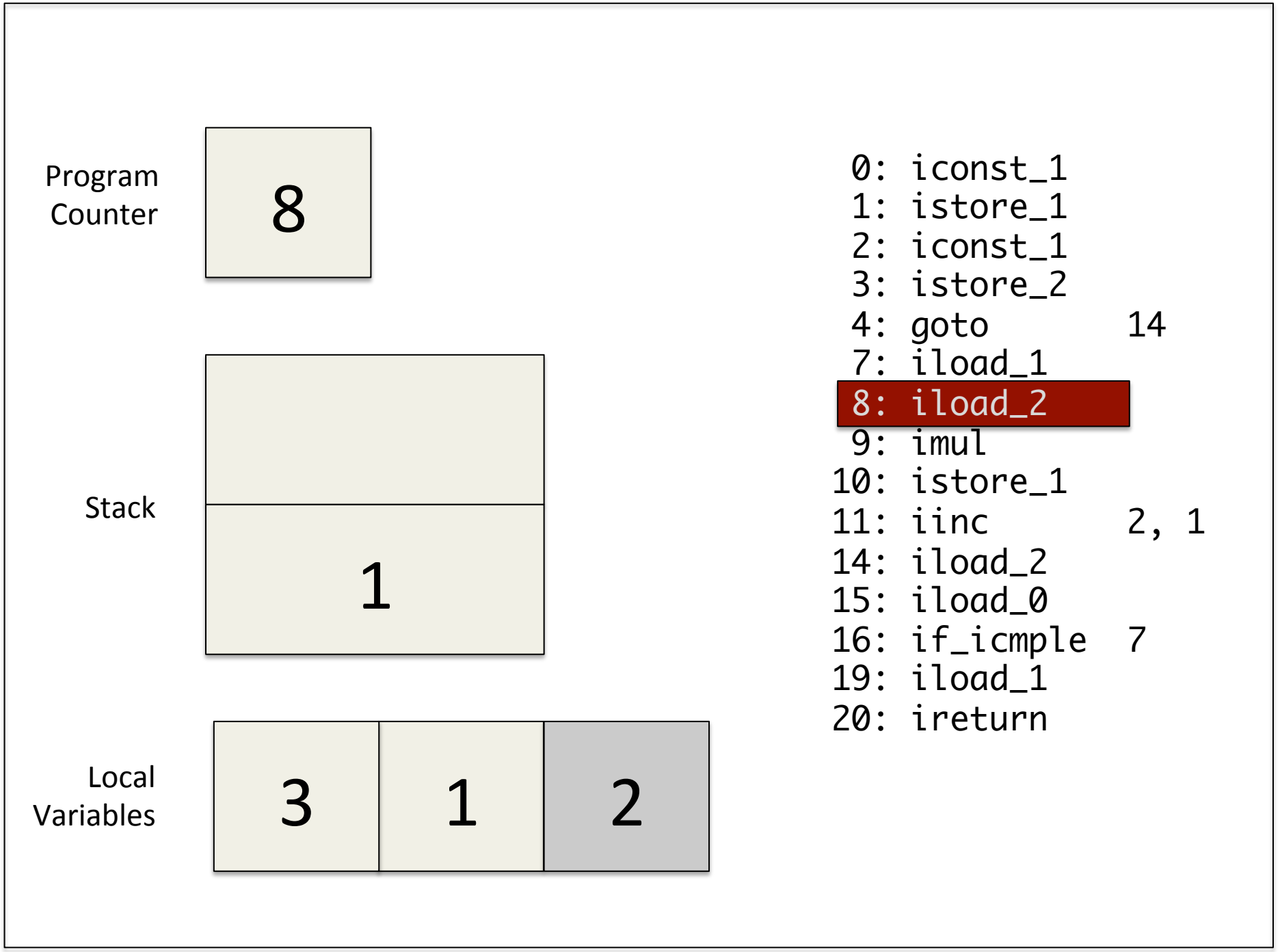

Program Counter
11
Stack
Local Variables
3
1
1
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn

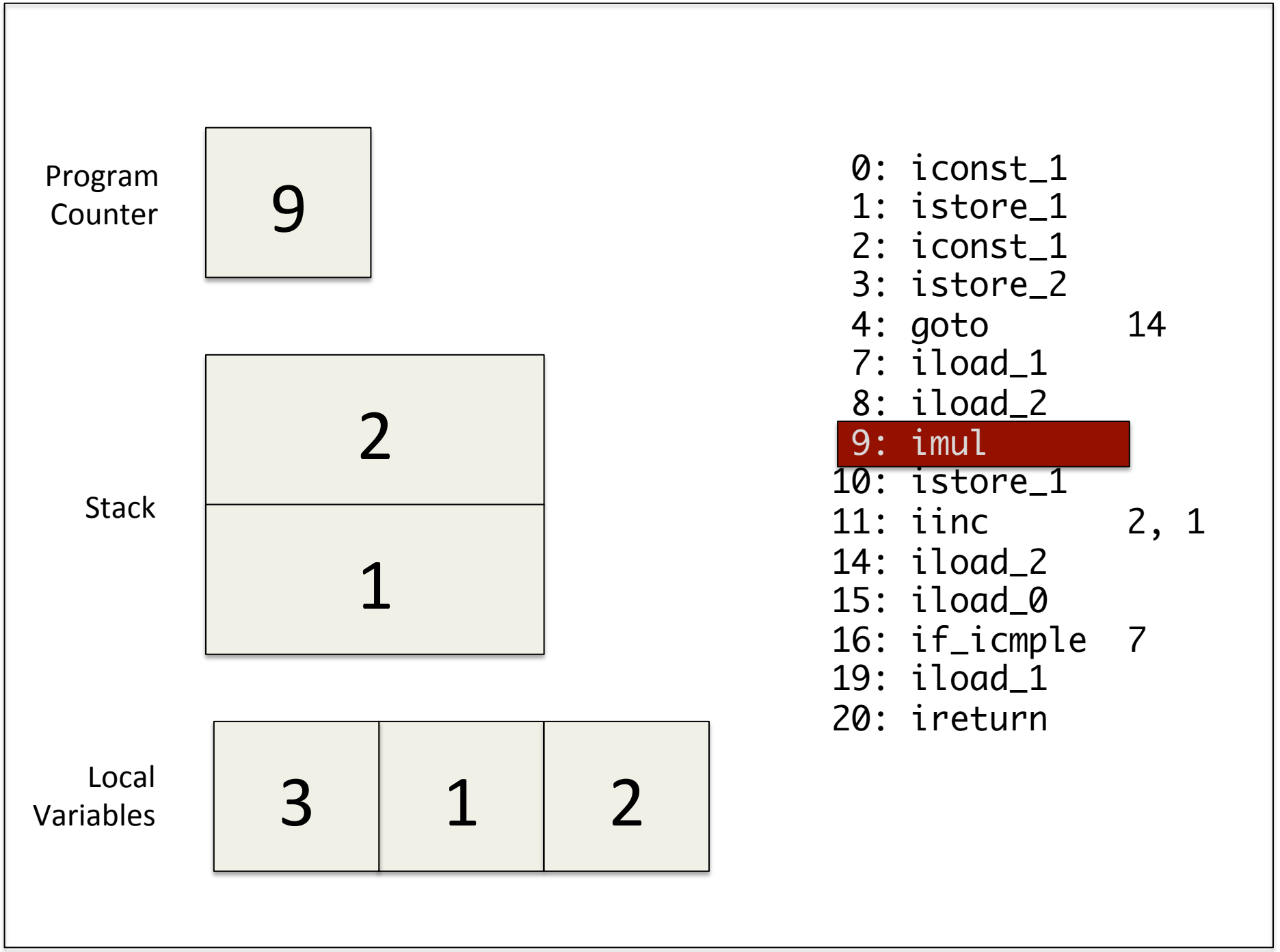

Program Counter
14
Stack
Local Variables
3
1
2
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn

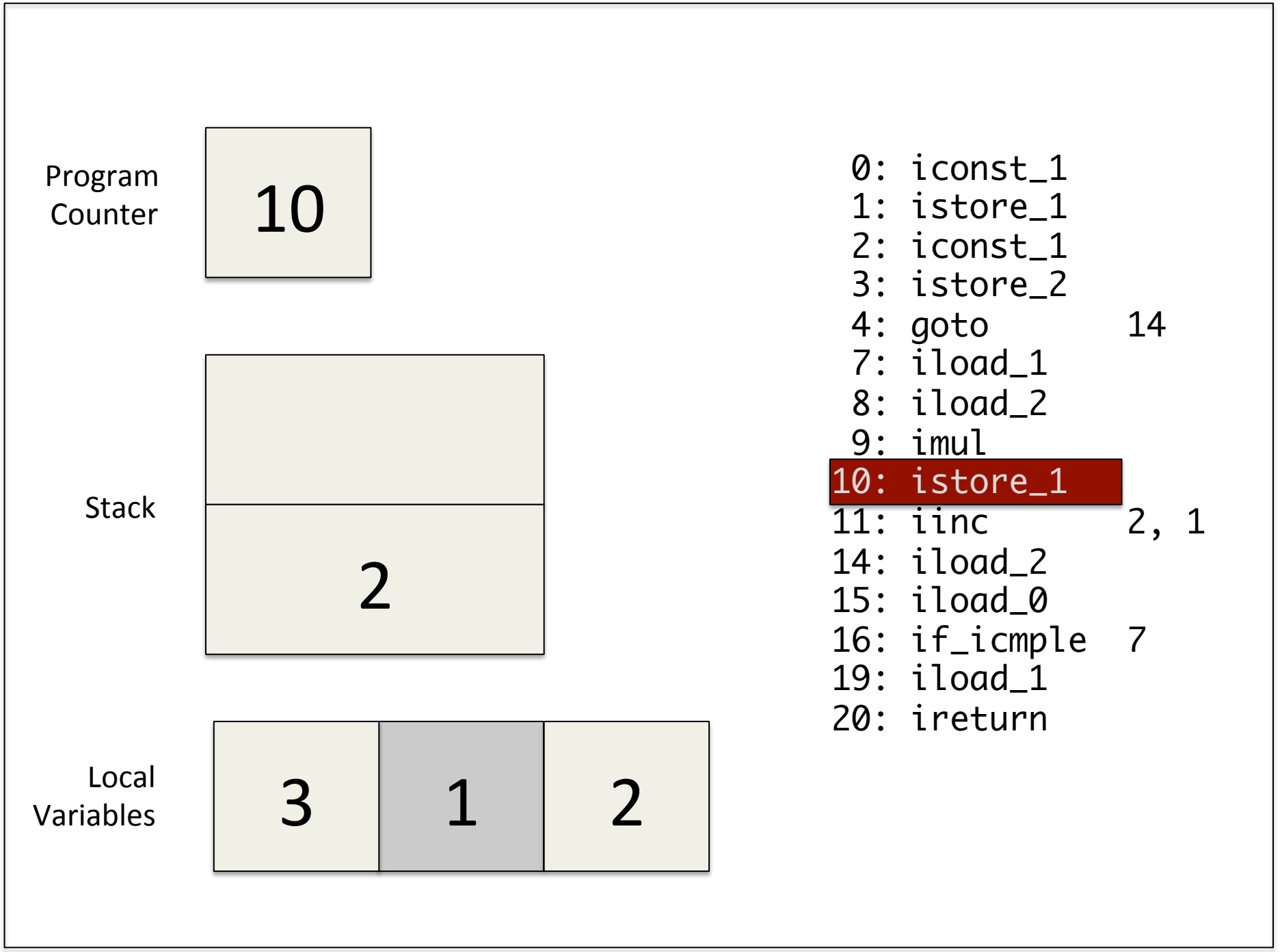

Program Counter
15
Stack
2
Local Variables
3
1
2
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn

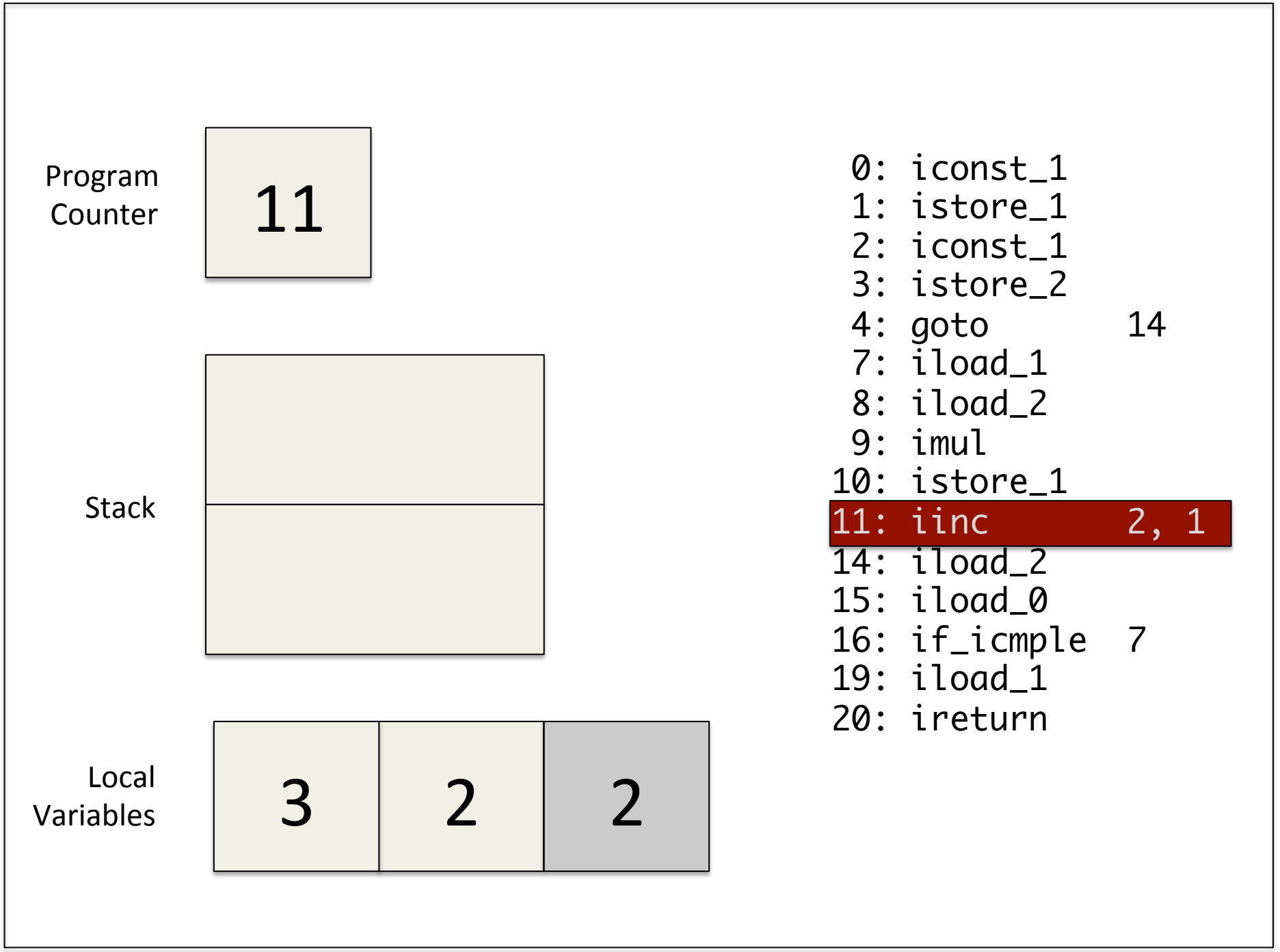

Program Counter
16
Stack
3
2
Local Variables
3
1
2
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn

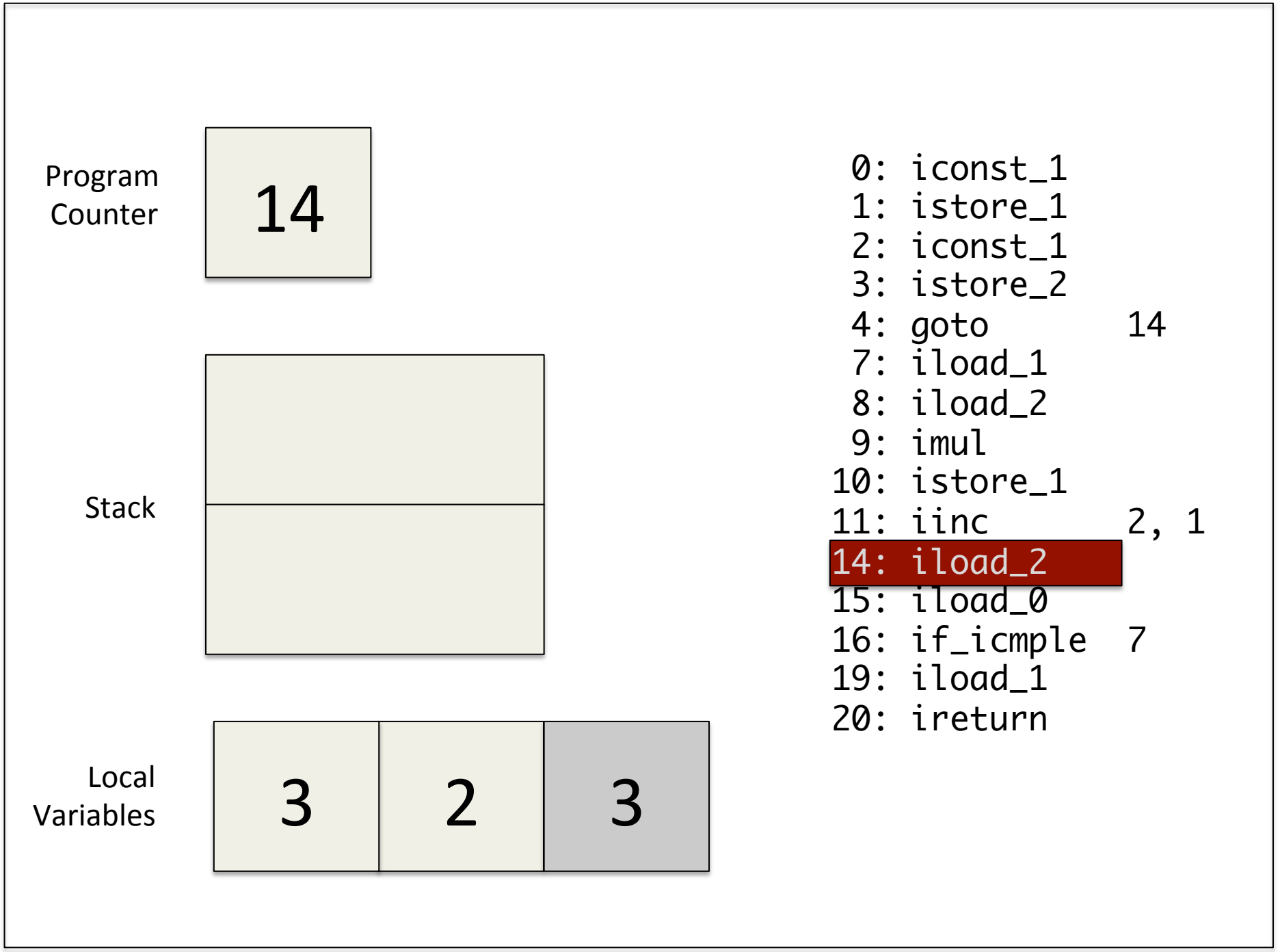

Program Counter
7
Stack
Local Variables
3
1
2
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn

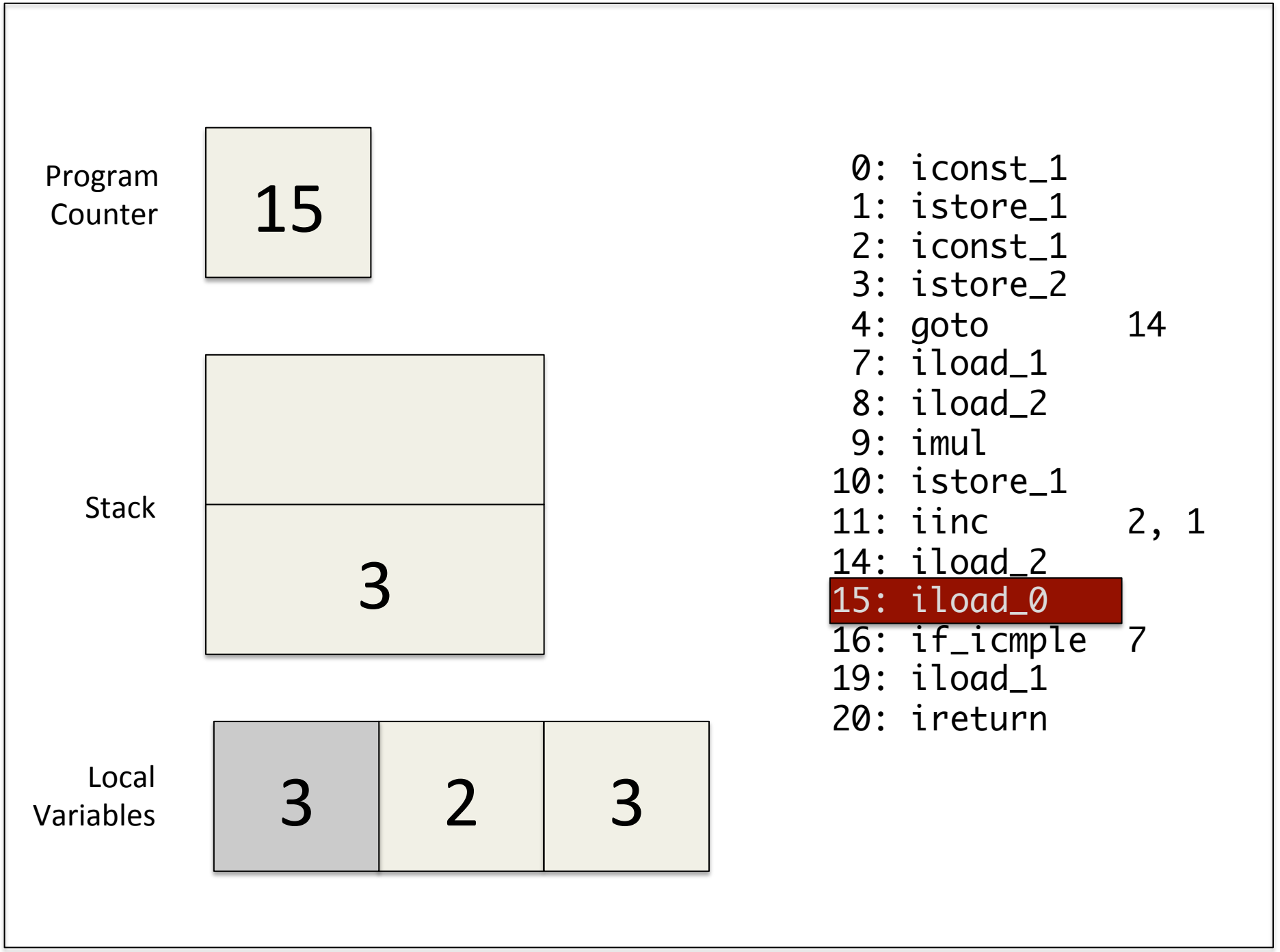

Program Counter
8
Stack
1
Local Variables
3
1
2
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn

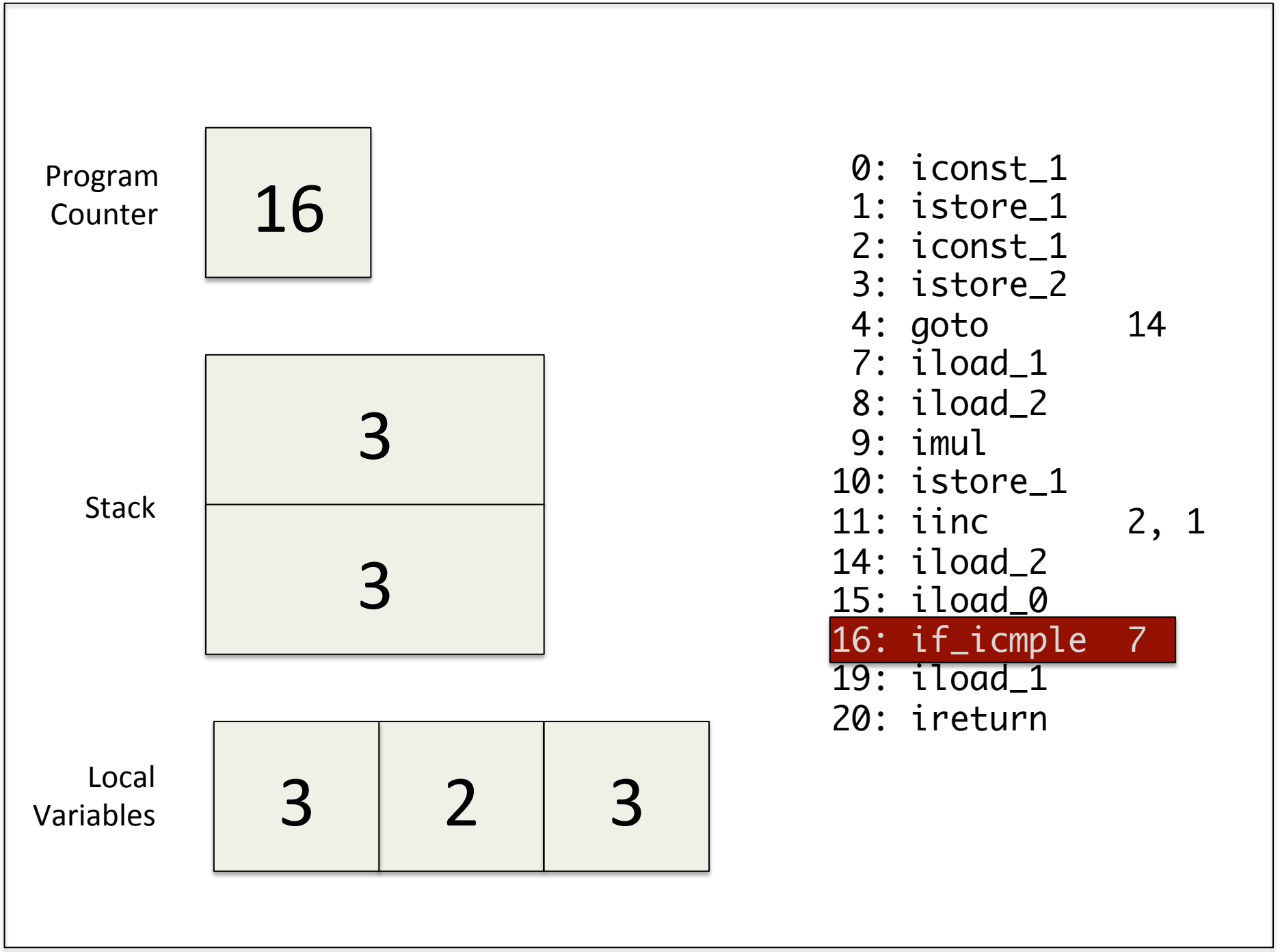

Program Counter
9
Stack
2
1
Local Variables
3
1
2
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn

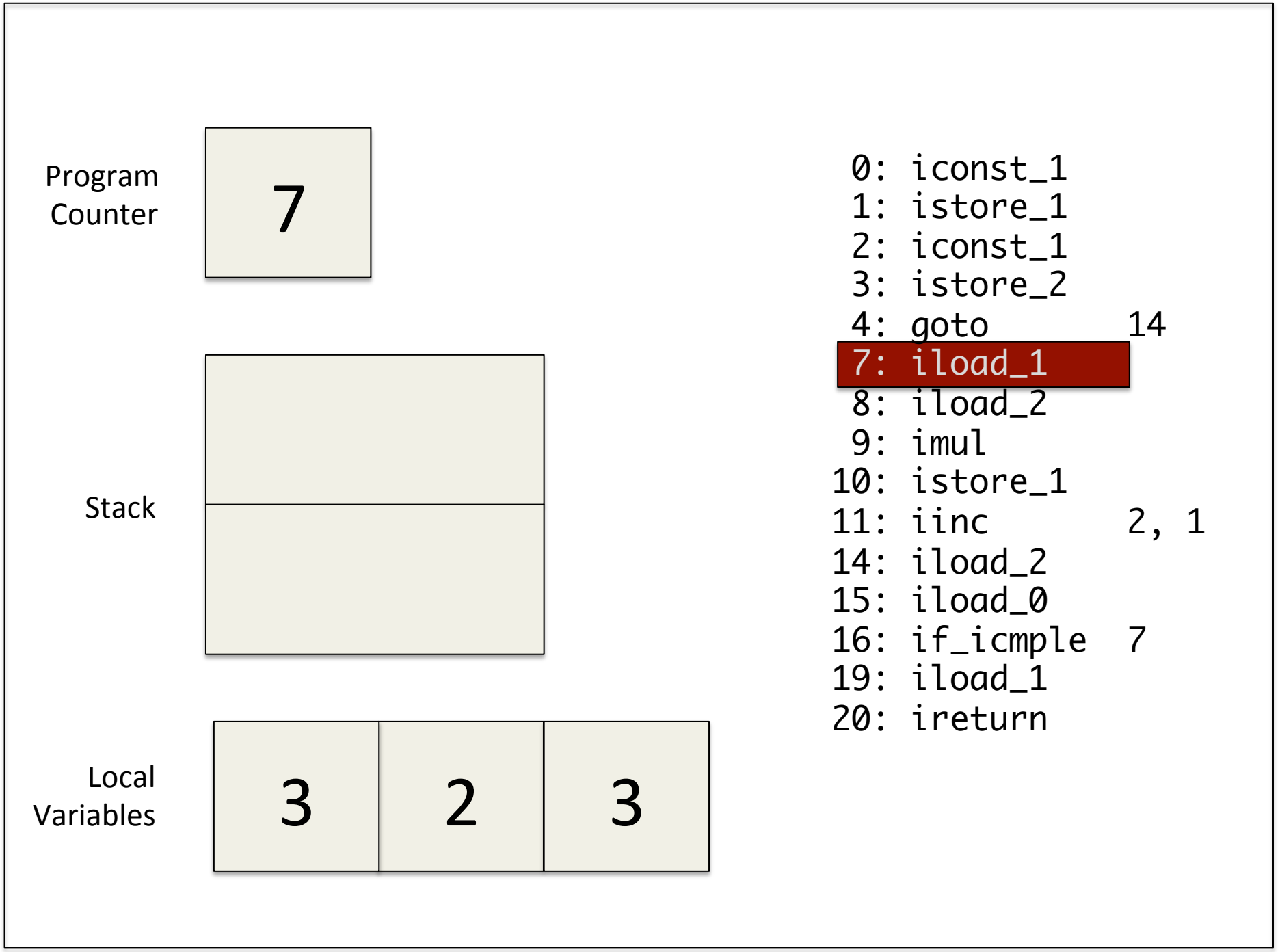

Program Counter
10
Stack
2
Local Variables
3
1
2
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn

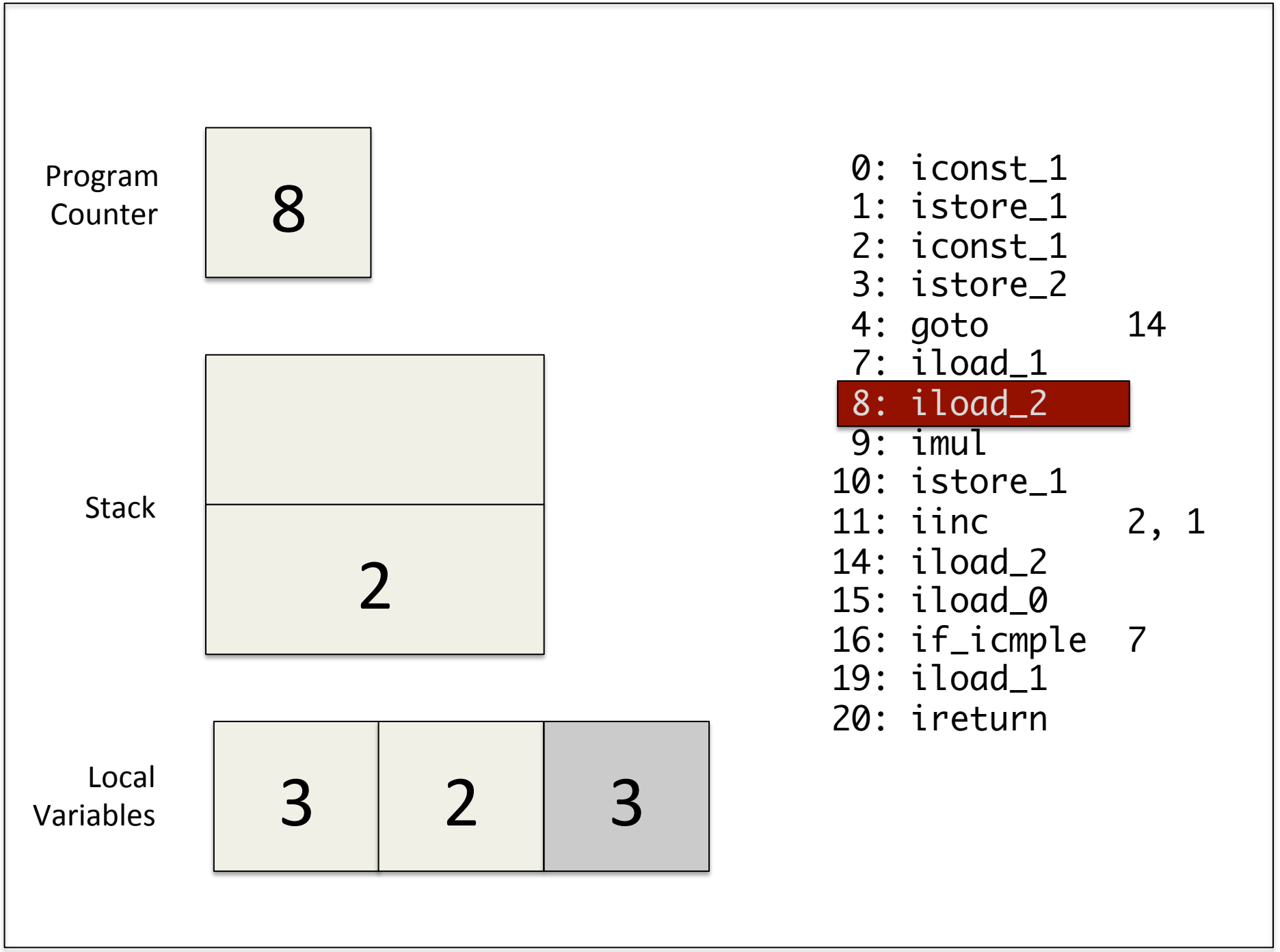

Program Counter
11
Stack
Local Variables
3
2
2
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn

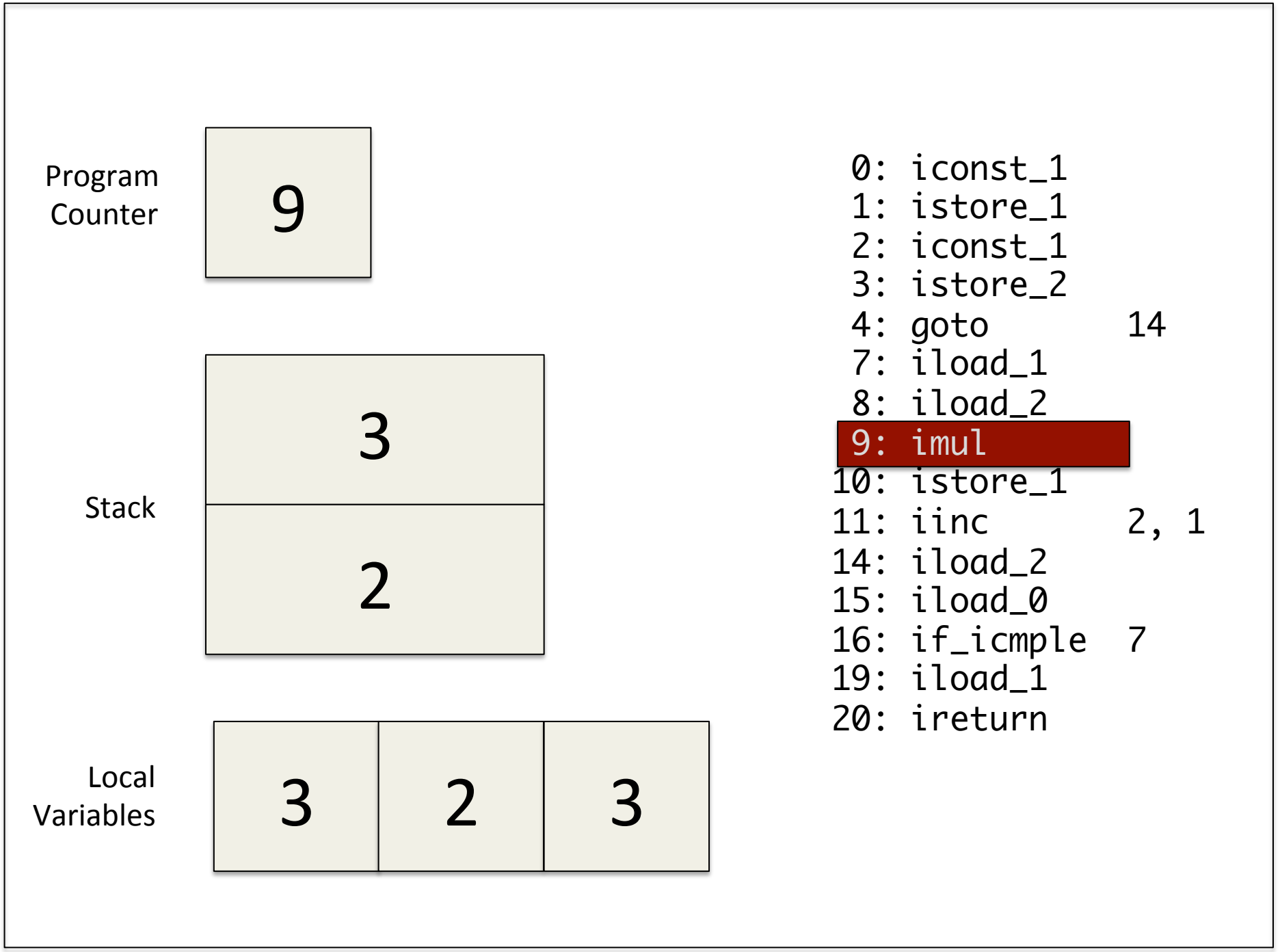

Program Counter
14
Stack
Local Variables
3
2
3
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn

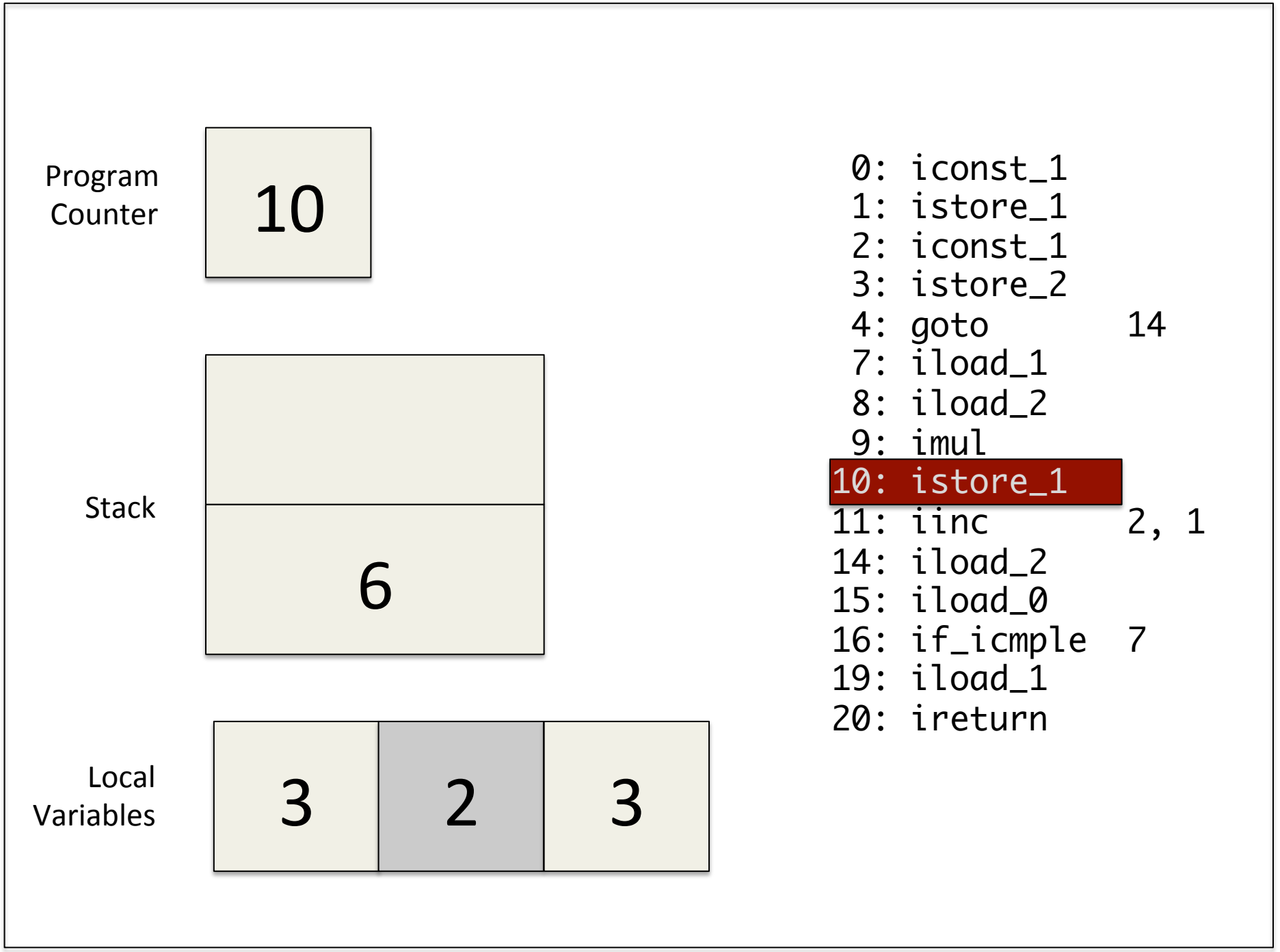

Program Counter
15
Stack
3
Local Variables
3
2
3
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn

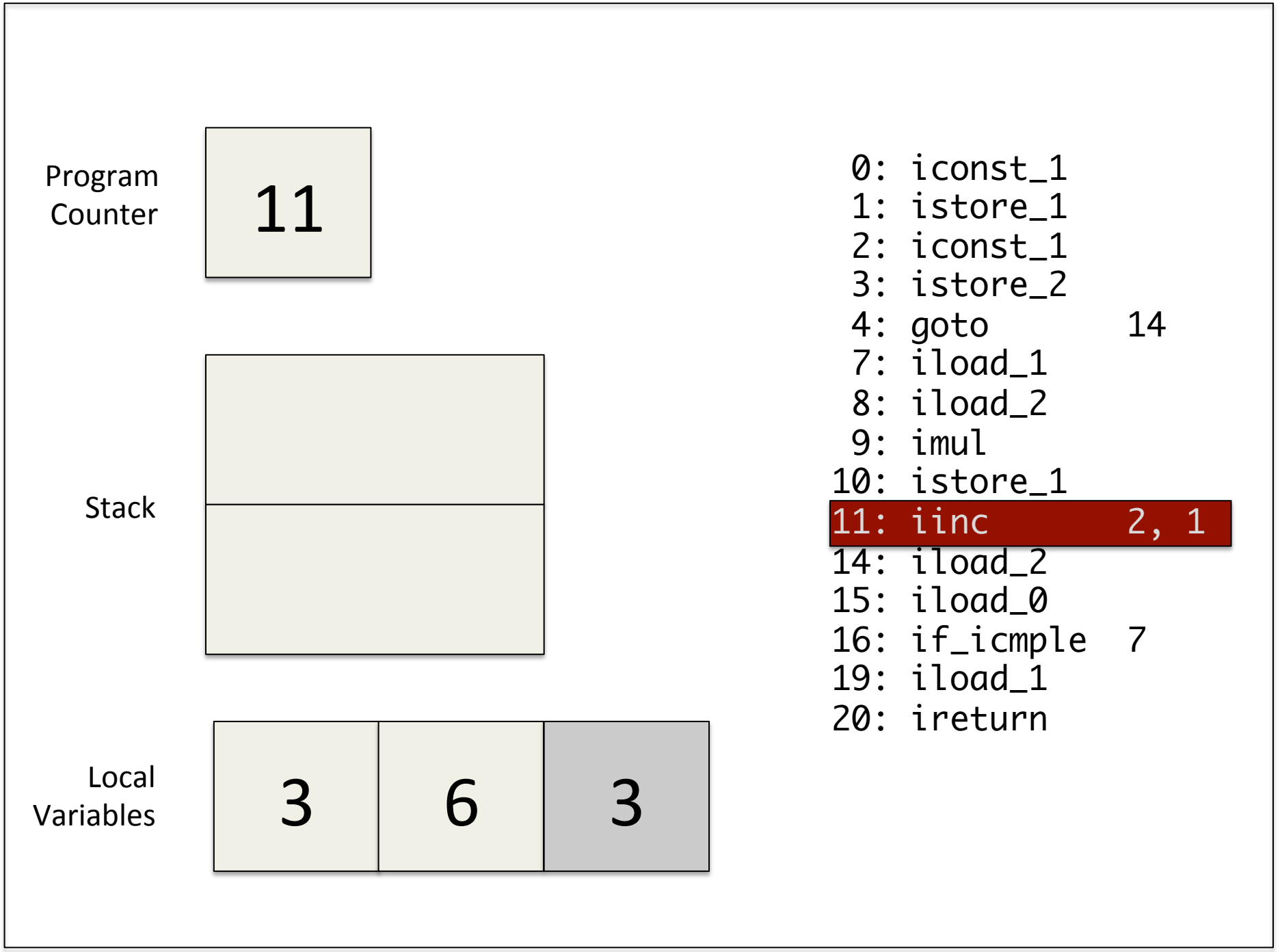

Program Counter
16
Stack
3
3
Local Variables
3
2
3
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn

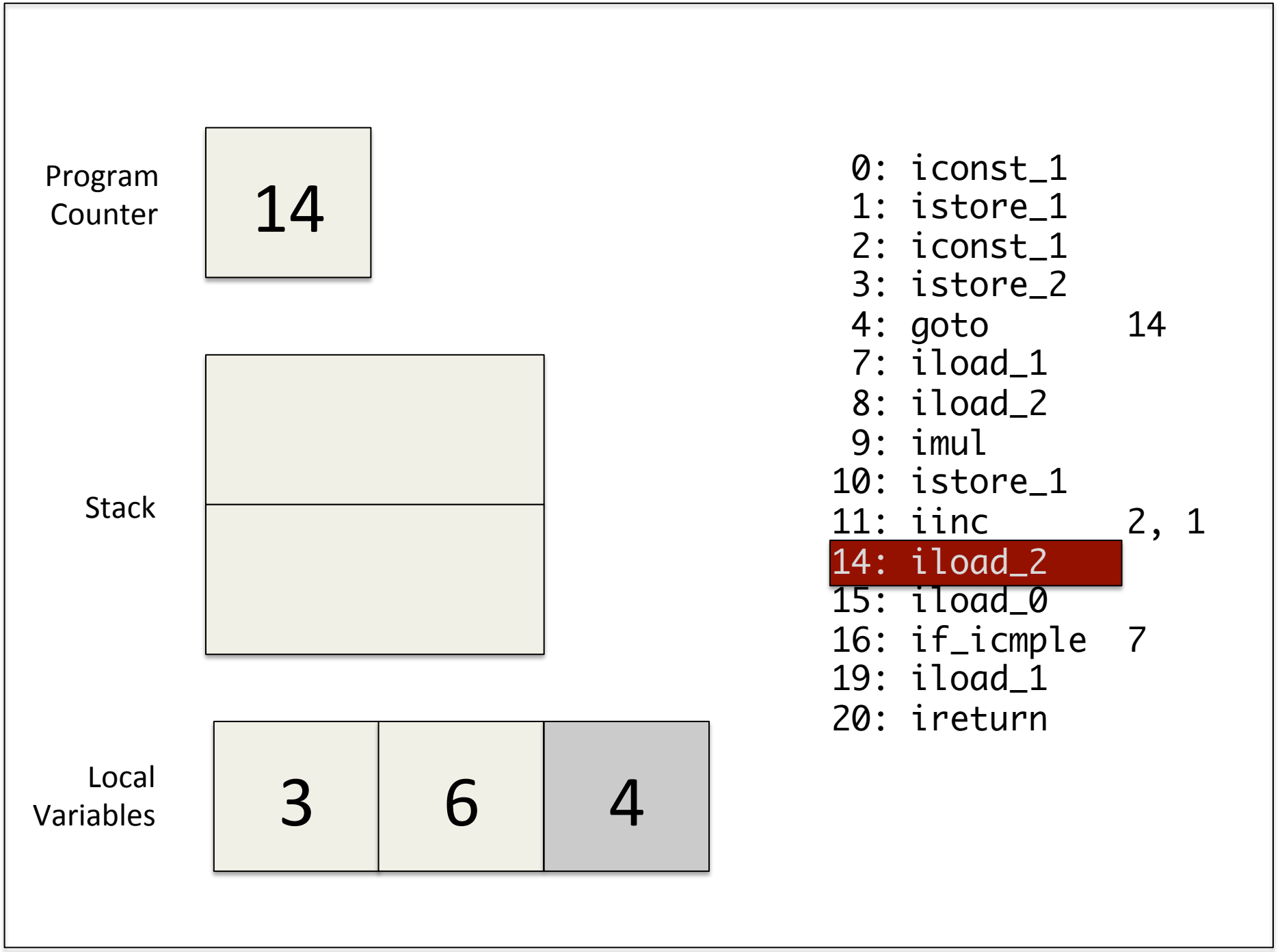

Program Counter
7
Stack
Local Variables
3
2
3
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn

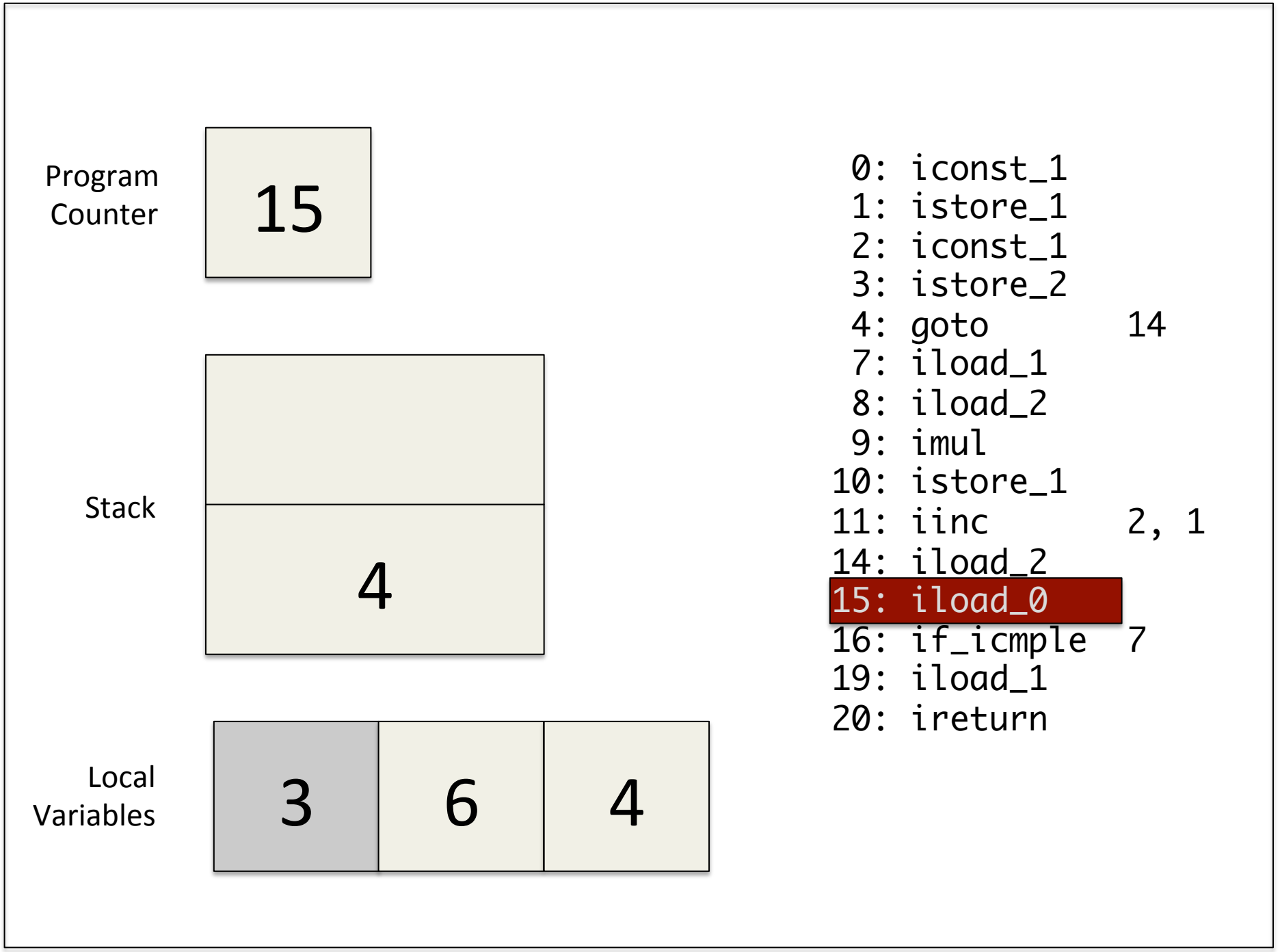

Program Counter
8
Stack
2
Local Variables
3
2
3
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn

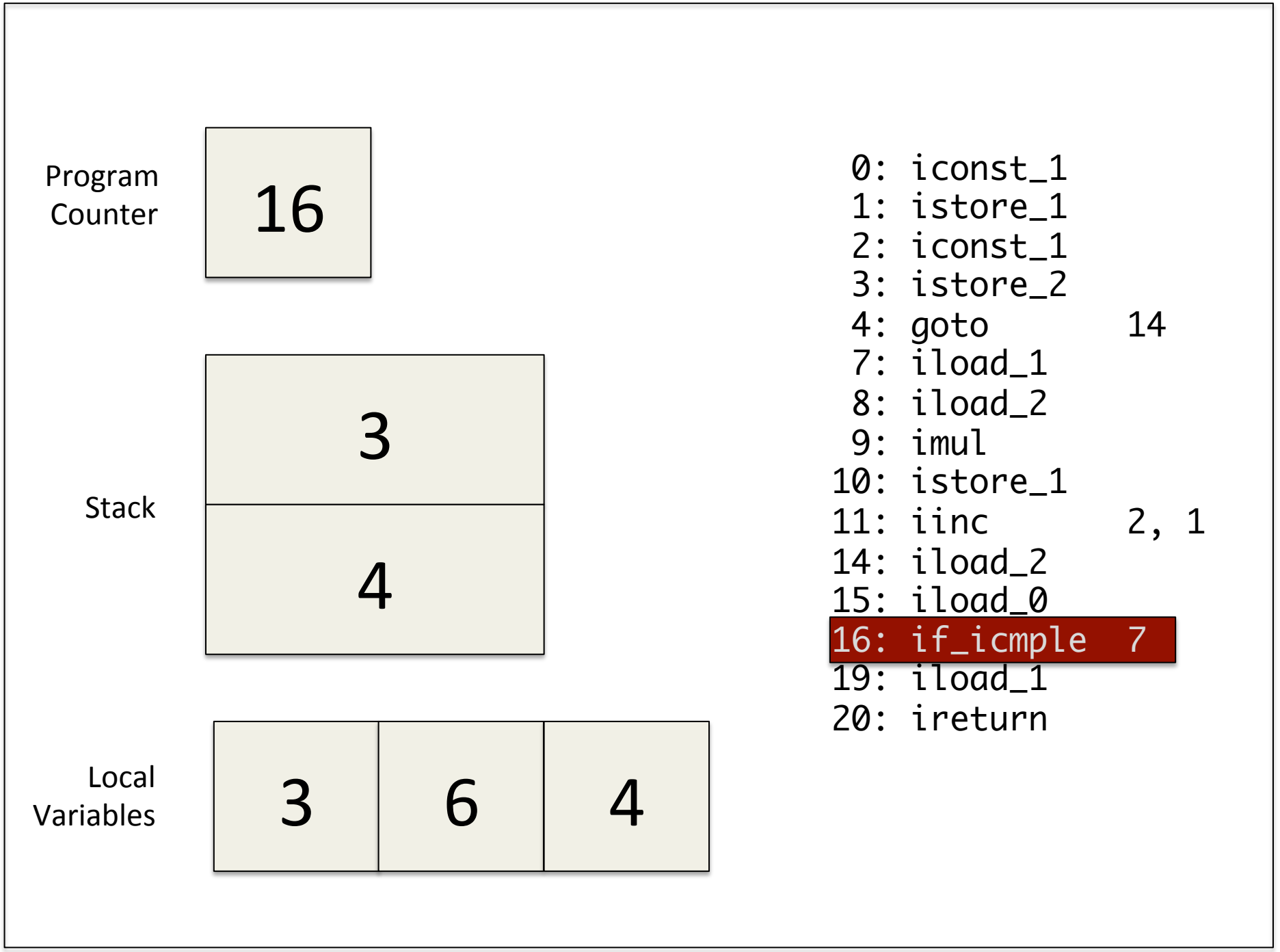

Program Counter
9
Stack
3
2
Local Variables
3
2
3
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn

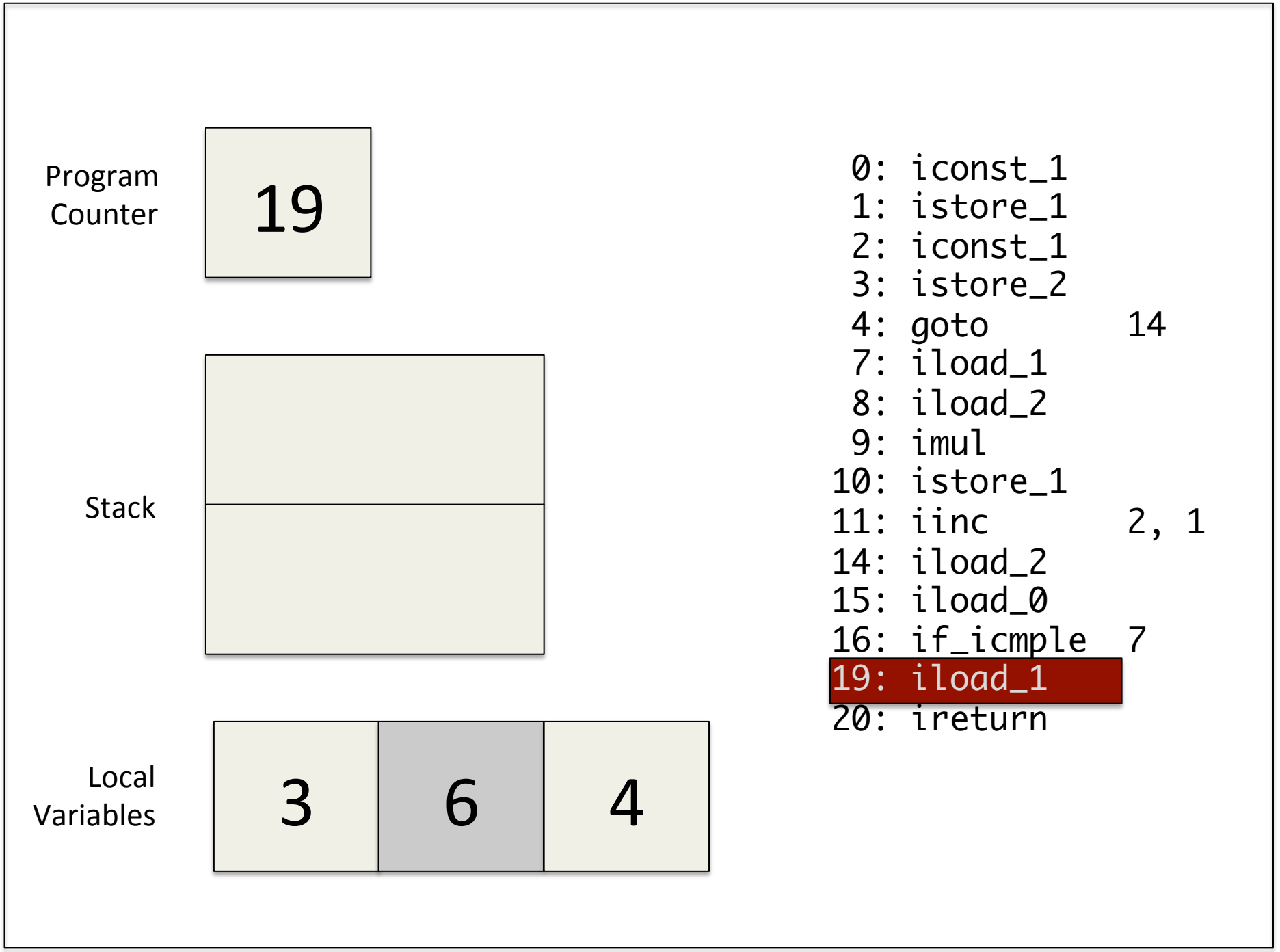

Program Counter
10
Stack
6
Local Variables
3
2
3
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn

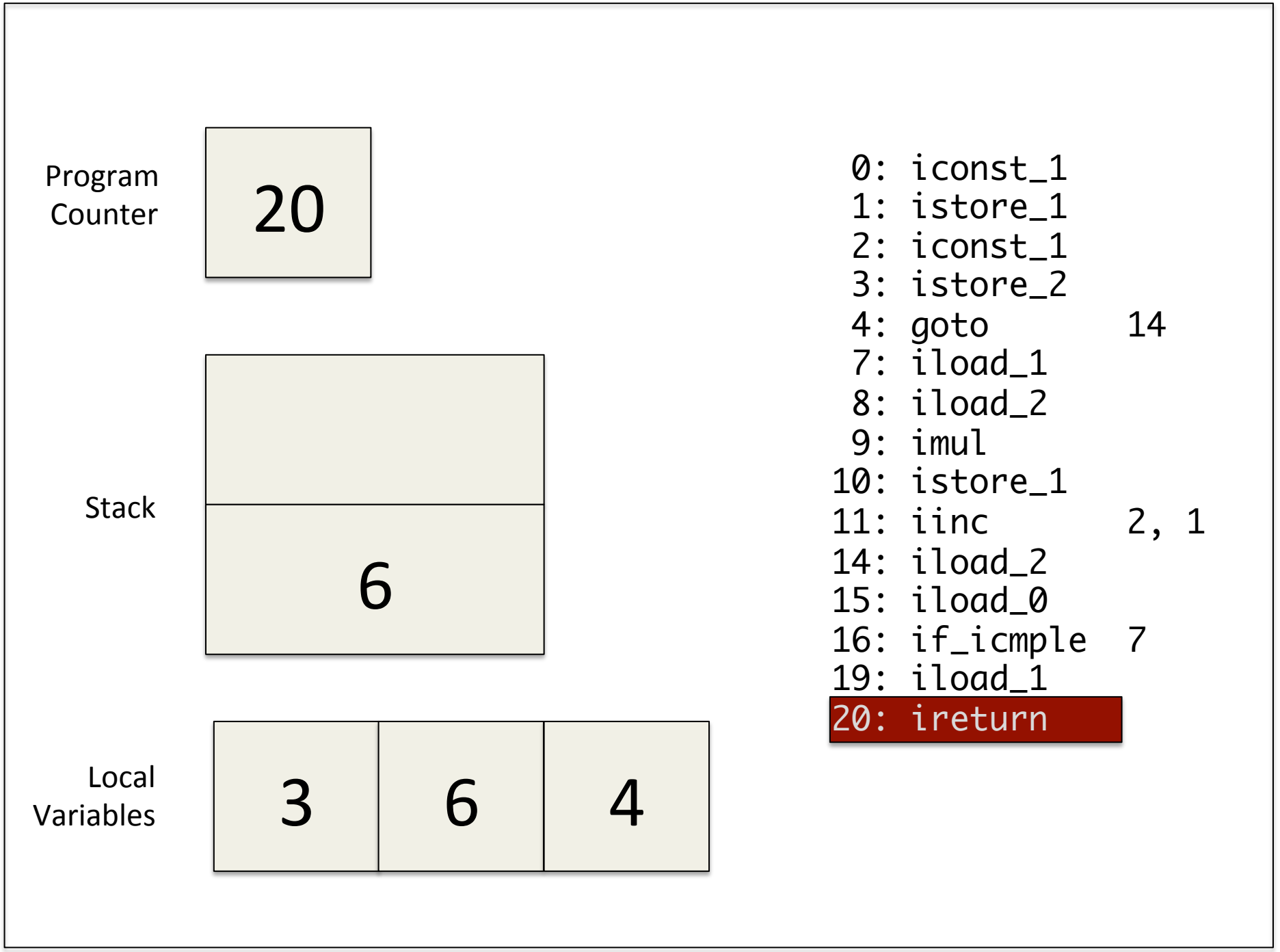

Program Counter
11
Stack
Local Variables
3
6
3
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn


Program Counter
14
Stack
Local Variables
3
6
4
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn


Program Counter
15
Stack
4
Local Variables
3
6
4
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn


Program Counter
16
Stack
3
4
Local Variables
3
6
4
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn


Program Counter
19
Stack
Local Variables
3
6
4
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn

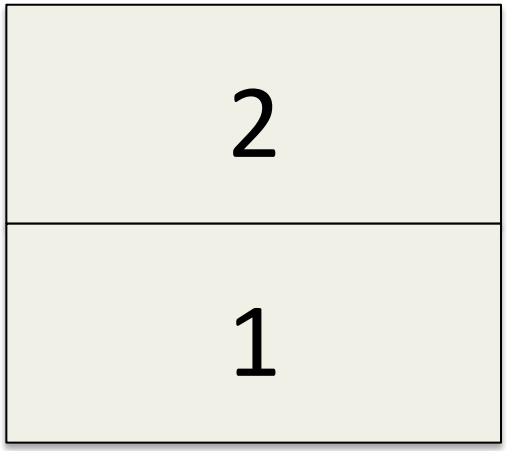

Program Counter
20
Stack
6
Local Variables
3
6
4
0: iconst_1
1: istore_1
2: iconst_1
3: istore_2
4: goto 14
7: iload_1
8: iload_2
9: imul
10: istore_1
11: iinc 2, 1
14: iload_2
15: iload_0
16: if_icmple 7
19: iload_1
20: ireturn

Stack Manipulation

- Reorder values
  - `SWAP`

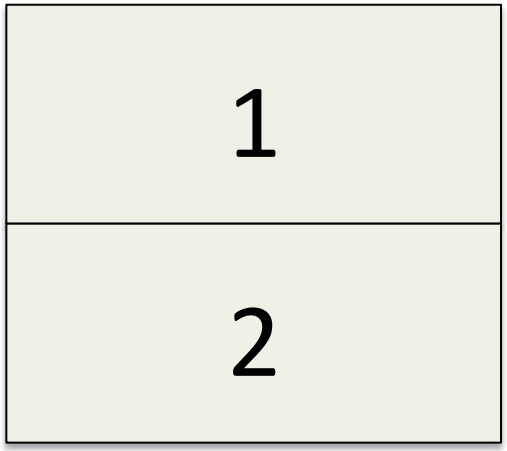
SWAP

- Swap the top two values on the operand stack



SWAP

- Swap the top two values on the operand stack

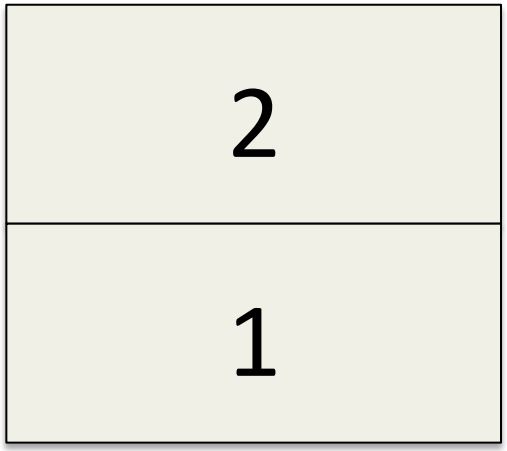


Stack Manipulation

- Reorder values
  - `SWAP`
- Remove values
  - `POP, POP2`

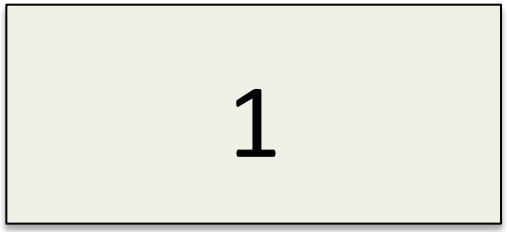
POP

- Pop the top operand stack value



POP

- Pop the top operand stack value

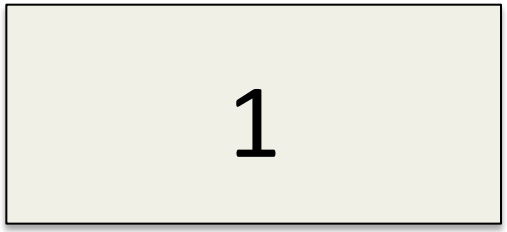


Stack Manipulation

- Reorder values
  - `SWAP`
- Remove values
  - `POP, POP2`
- Duplicate values
  - `DUP`
  - `DUP_x1, DUP_x2`
  - `DUP2`
  - `DUP2_x1, DUP2_x2`

DUP

- Duplicate the top operand stack value

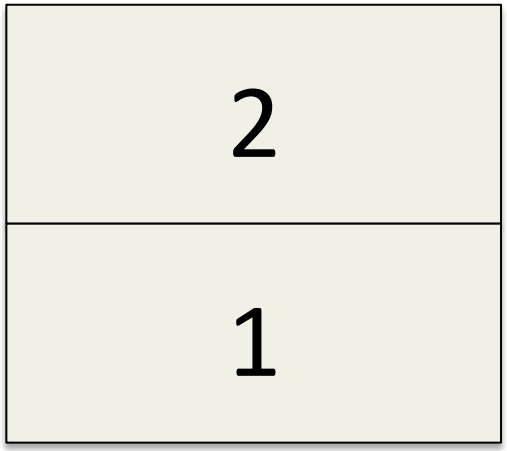


DUP

- Duplicate the top operand stack value

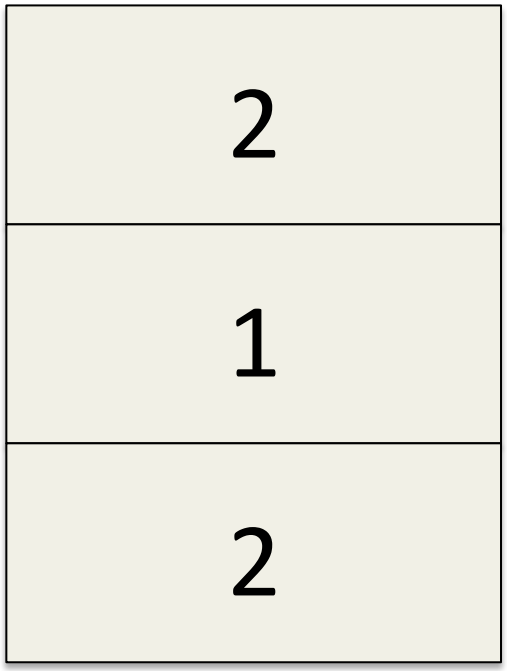
DUP_X1

- Duplicate the top operand stack value and insert two values down



DUP_X1

- Duplicate the top operand stack value and insert two values down



Interpreter Implementation

- Interpreters are a basic way of executing code
  - They don't have to be naïve
- Interpretation can be optimized
- Extra features can be added to the interpreter
  - Profiling
  - Debugging

Basic Interpretation

- Interpreter designed around a switch statement
  - Loop over each bytecode
  - Check the opcode at the current program counter
  - Jump to code to interpret that bytecode
- Simple to understand and implement
- Code sequences can be optimized

Template-Based Interpretation

- More advanced form of interpretation
- On VM startup, custom build the interpreter
  - Choose a template for each bytecode
  - Link them together to form an interpreter
- Templates are hand-optimized for the platform
  - Both the OS and the CPU
  - Code runs much closer to the hardware

Pros and Cons

- The template interpreter runs far faster
  - Assembly is optimized to the platform
  - Fine control like register allocation for the PC
- Implementation is much harder
  - 10 KLOC of Intel assembly, 14 KLOC of SPARC
  - Porting to a new platform is a major task
- Harder to implement debugging hooks

For Next Time

- Keep an eye out for Assignment 1
- First reading and discussion topic is now posted
  - Running languages other than Java on the JVM
  - Check the Canvas discussion board
- Questions or comments very welcome
  - Contact over e-mail, Canvas or office hours
  - Send any areas that you'd like to know more about