defined by the receiver and the argument. Although pithy, this is not the most optimal solution; Squeak provides an alternative one:

```
intersects: aRectangle
    "Answer whether aRectangle intersects the receiver anywhere."
    "Optimized; old code answered:
         (origin max: aRectangle origin) < (corner min: aRectangle corner)"

    | rOrigin rCorner |
    rOrigin := aRectangle origin.
    rCorner := aRectangle corner.
    rCorner x <= origin x ifTrue: [^ false].
    rCorner y <= origin y ifTrue: [^ false].
    rOrigin x >= corner x ifTrue: [^ false].
    rOrigin y >= corner y ifTrue: [^ false].
    ^ true
```

Faster, but less beautiful. It gives us the opportunity, though, to introduce some Smalltalk syntax. Variables local to a method are declared inside `| |`. The assignment operator is `:=`, `^` is the equivalent of `return` in C++ and Java, and the period (`.`) separates statements. The code inside square brackets (`[ ]`) is called a *block*, a key concept in the Smalltalk architecture. A block is a *closure*—that is, a piece of code that can access the variables defined in its surrounding scope. Blocks in Smalltalk are represented by the class `BlockContext`. The contents of a block are executed when the block object receives the message `value`, and in most cases (like here), the message is sent implicitly. Comments in Smalltalk are inside double quotes; single quotes are used for strings.

A `BlockContext` shares the receiver, the arguments, the temporary variables, and the sender of the context that creates it. There is a similar class, `MethodContext`, representing all the dynamic state associated with the execution of a method (which, as we saw, is represented by `CompiledMethod`, an array of bytecodes). As we would expect from an object-oriented language, both `BlockContext` and `MethodContext` are subclasses of class `ContextPart`. The `ContextPart` class adds execution semantics to its superclass, `InstructionStream`. On its part, `InstructionStream` is the class whose instances can interpret Smalltalk code. The superclass of `InstructionStream` is `Object`, where the chain ends.

Apart from the `value` selector, blocks also have the `fork` selector, which implements concurrency in the language. As everything in Smalltalk is an object, processes are instances of the `Process` class. The `Delay` class allows us to suspend an execution of a process for a specified period of time; a `Delay` object will suspend the current executing process when it is sent the `wait` message. Combining all this together, a trivial clock can be implemented with the following code (Goldberg and Robson 1989, p. 266):

```
[[true] whileTrue:
    [Time now printString displayAt: 100@100.
    (Delay forSeconds: 1) wait]] fork
```

The `whileTrue:` selector will execute its block argument as long as its own receiver block is true. The `@` character is a selector of class `Number` that constructs instances of class `Point`.