threading model). However, green-threaded JVMs have provided the starting point for a number of Java operating system projects that could be able to avoid some of the performance pathologies of green threading on top of a standard operating system.

As processor performance and operating system implementations have improved, the advantage of managing threads in the JVM has diminished—although it is still advantageous to manage uncontended locking operations within the JVM.

Using Java, a clean threading API has been created that allows Jikes RVM to have different underlying threading models, such as those provided by different operating systems. In the future, having a flexible interface between the language and operating system threads can allow Jikes RVM to adapt to new programmer behaviors (for example, by supporting thousands of threads).

Native Interface

Unfortunately, staying in Java code isn't always possible for Jikes RVM. Accessing operating system routines is necessary to allocate pages of memory. The class library interfaces Java code with existing libraries, such as those for windowing. Jikes RVM provides two means of accessing native code:

Java Native Interface (JNI)
: JNI is a standard that allows Java applications to integrate with native applications typically written in C. A feature of JNI is that Java objects can be handled, and Java methods can be called. To do the bookkeeping, a list of objects accessed in native code is required to prevent these objects from being garbage collected. This introduces an overhead when using JNI for native methods.

SysCalls
: These are similar to native methods in their declaration, except they have an extra annotation. They allow a more efficient transition in and out of native code, with the restriction that the native code is not able to call back into the VM via the JNI interfaces. Jikes RVM implements a SysCall as a simple procedure call to a native method using the default operating-system calling conventions.

Class Loaders and Reflection

The class loader is responsible for loading classes into Jikes RVM and interfacing this process with the object model. Reflection allows an application to query the types of objects and even perform method calls on objects whose types were not known at static compile time. Reflection occurs through applications using objects such as `java.lang.Class` or those in the package `java.lang.reflect`, which are API wrappers to routines within Jikes RVM's runtime.