The list of primordial classes is produced during the bootstrap by searching directories and from reading an explicit list of classes to compile. The explicit list is particularly important for array types. It would be possible to produce the list of primordial classes by repeated compilation and growing the set of classes included into the boot image, but this would significantly increase the time it takes to build Jikes RVM. A proposed alternative is to use Java annotations to mark which classes are primordials.

Before traversing the object graph and writing the boot image, the boot image writer compiles the primordials. Compiling a primordial involves loading its class with Jikes RVM's class loader, which will automatically allocate space in the JTOC and the TIB as necessary, and then iterating over all the methods and compiling them with one of Jikes RVM's compilers. As this is all pure Java code, the boot image writer takes advantage of Java's concurrency API to perform this task in parallel if possible.

Once compilation of the core set of the primoridal classes is complete, the object graph in the host JVM's heap represents sufficient functionality for Jikes RVM to bootstrap itself, allocate additional objects, and start loading and executing user classes. To complete the bootstrap process, this core object graph is traversed and written out to disk in Jikes RVM's object model using the capabilities of the Java reflection API provided by the host JVM.

The Boot Image Runner and VM.boot

As mentioned in "Bootstrapping a Self-Hosting Runtime," the boot image runner is responsible for loading the compiled images into memory. The exact details of this vary depending on the operating system, but the images are set up to be demand-paged into memory. Demand paging means that pages from the boot image remain on disk until they are required.

Once in memory, the boot image runner initializes the boot record and then loads the machine registers to transfer execution over to the Jikes RVM method `org.jikesrvm.VM.boot` (or `VM.boot` for short). Jikes RVM is responsible for all memory layout, enabling efficient garbage-collection techniques and a stack organization that is efficient at dealing with Java exceptions (see the later section "Exception Model"). Once the `VM.boot` method is entered, special wrappers are needed to transfer between native code in the boot image runner and C libraries (these are described further in the upcoming section "Native Interface").

The job of `VM.boot` is to ensure that the VM is in a ready state to execute a program. It does this by initializing the components of the RVM that couldn't be initialized when the boot image was written. Some components must be started explicitly—for example, the garbage collector. The remaining components are a small subset of the primordial classes that were not fully written into the boot image because of inconsistencies in the bootstrap and Jikes RVM class files. To initialize these classes, the static initializer of the class must be run.