in prototype runtime environments, most modern production virtual machines rely on some form of *selective optimization*. In selective optimization, online profiling is used to identify a subset of the executing methods to compile with an aggressive optimizing compiler; the remainder of the methods are either interpreted or compiled by a very fast nonoptimizing compiler immediately before execution. It is this selectivity that enables the use of sophisticated optimizing compilers at runtime.

Unlimited Analysis in a Static Compiler Must Mean Better Performance

As runtime environments will be heavily used by many applications, optimizing them to extract greater performance across all applications makes sense. However, a range of optimizations cannot be performed if the runtime is not created as part of a dynamic environment:

Online profiling
: Aspects of the runtime vary at runtime—for example, the average size of pieces of data, or particular coding styles that may be based on differing design patterns. Online profiling allows timely use of this information to reduce overheads, such as branch prediction, and also allow more advanced optimizations, such as value speculation. An example from Java of value speculation could be to predict that the majority of stream output operations may be occurring to the `java.lang.System.out` file stream. Value speculation is an extension to partial evaluation, which we will consider further in the later section "Partial evaluation."

Variance in the underlying system
: The range of systems an application runs on is increasingly becoming more varied. The abilities of different processors, the amount of memory, the number of different processors, the power requirements, and the load of the system that the runtime is executing upon are all important in knowing how the runtime should best adapt.

Intraprocedural analysis
: Intraprocedural analysis is an important tool for an optimizing compiler, allowing optimization across method boundaries. Although off-line analysis can be unlimited, often this can result in so much data that the compiler cannot determine which data is important. As runtime feedback is more timely, it can better guide intraprocedural and other compiler optimizations.

Runtime Analysis Uses a Lot of Resources

Having a runtime environment has an overhead for the memory required by the environment. Similar requirements exist for the standard libraries that conventional applications use. On top of these, the runtime environment must keep information that can help guide its future compilation and execution. These memory requirements are modest, and through timely and memory-efficient sampling, the runtime environment can gain the most benefit with little cost.