[What's the difference between how virtual and non-virtual member functions are called?](http://jaiminmyinterestingcpp.blogspot.com/2012/09/what-difference-between-how-virtual-and.html)

Non-virtual member functions are resolved statically. That is, the member function is selected statically (at compile-time) based on the type of the pointer (or reference) to the object.

In contrast, virtual member functions are resolved dynamically (at run-time). That is, the member function is selected dynamically (at run-time) based on the type of the object, not the type of the pointer/reference to that object. This is called "dynamic binding." Most compilers use some variant of the following technique: if the object has one or morevirtual functions, the compiler puts a hidden pointer in the object called a "virtual-pointer" or "v-pointer." This v-pointer points to a global table called the "virtual-table" or "v-table."

The compiler creates a v-table for each class that has at least one virtual function. For example, if class Circle hasvirtual functions for draw() and move() and resize(), there would be exactly one v-table associated with classCircle, even if there were a gazillion Circle objects, and the v-pointer of each of those Circle objects would point to the Circle v-table. The v-table itself has pointers to each of the virtual functions in the class. For example, the Circlev-table would have three pointers: a pointer to Circle::draw(), a pointer to Circle::move(), and a pointer toCircle::resize().

During a dispatch of a virtual function, the run-time system follows the object's v-pointer to the class's v-table, then follows the appropriate slot in the v-table to the method code.

The space-cost overhead of the above technique is nominal: an extra pointer per object (but only for objects that will need to do dynamic binding), plus an extra pointer per method (but only for virtual methods). The time-cost overhead is also fairly nominal: compared to a normal function call, a virtual function call requires two extra fetches (one to get the value of the v-pointer, a second to get the address of the method). None of this runtime activity happens with non-virtual functions, since the compiler resolves non-virtual functions exclusively at compile-time based on the type of the pointer.