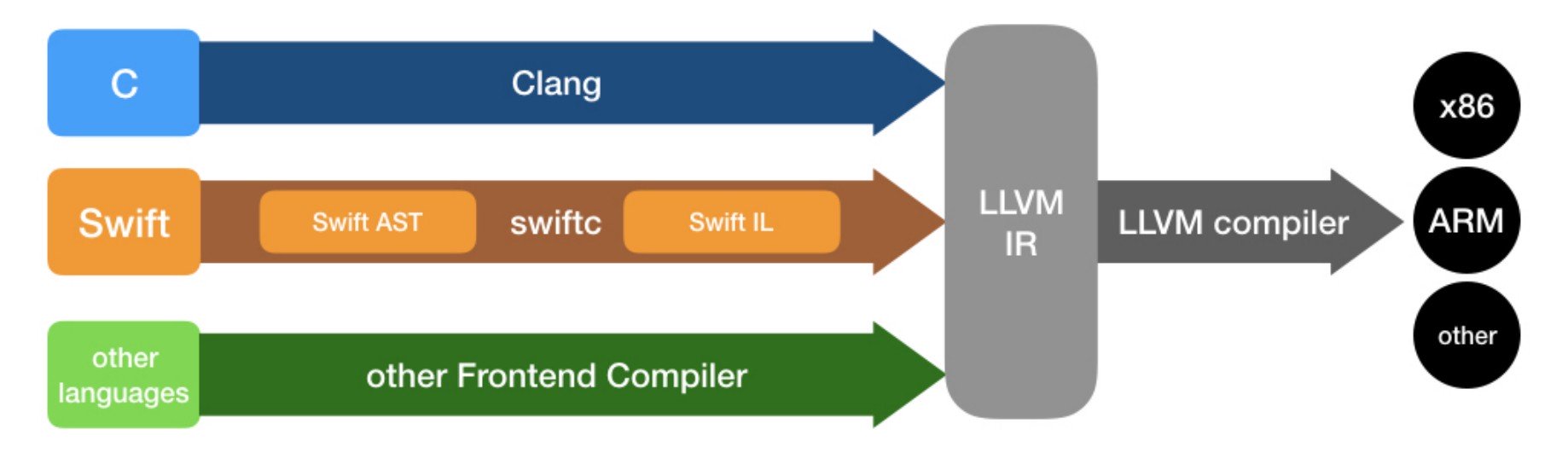
Py-Catchup

asdf

**General**

LLVM = low-level virtual machine. But the acronym is pretty inaccurate now, and LLVM is essentially a compiler made up of a number of parts, just like GCC (GNU Compiler Collection) is a compiler. The difference is that GCC supports many programming languages whereas LLVM isn’t a compiler for any given language. It’s a framework to generate object code from any kind of source code. I.e. middle man between more abstract language to assembly code.



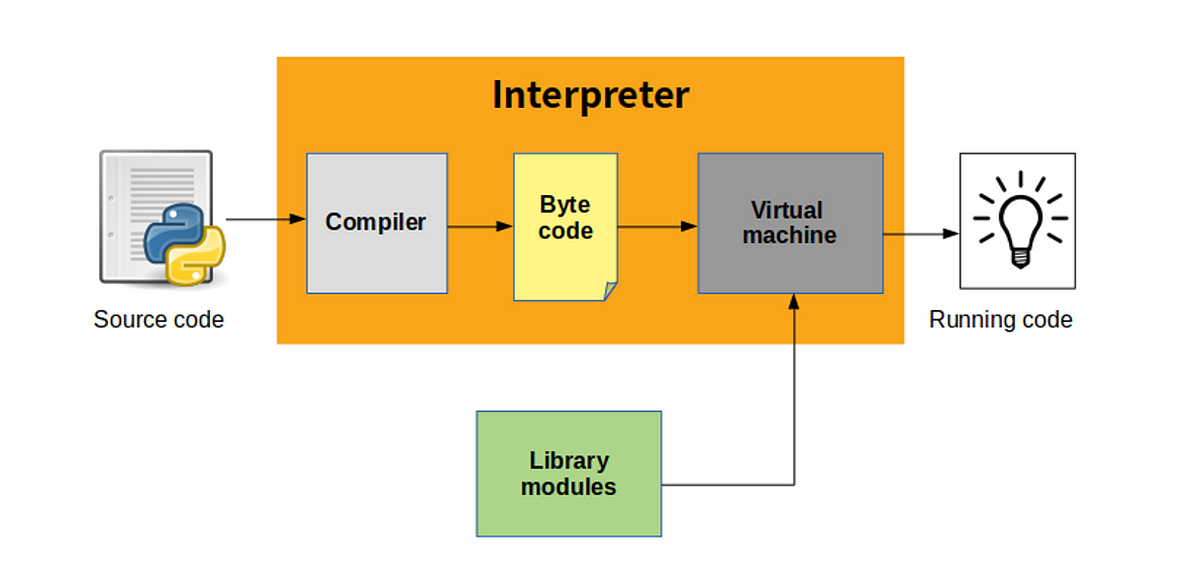
**How Does Python Actually Work?**

Python is considered an ‘interpreted’ language but in reality it’s also compiled. It really isn’t a black and white definition; Python only refers to the language, not the way it’s implemented. The default implementation of it is CPython, which, as you’d expect, is written in both C and Python. When we say ‘Python’ more generally, we are usually referring to CPython – however there are plenty of other implementations such as Microsoft’s IronPython (which compiles to .NET), Jython (which compiles to the Java Virtual Machine), and PyPy (which is written in Python itself and can compile to many back-end forms).

Now, CPython is responsible for compiling the source Python code into a Python-specific lower-level form called **bytecode.** Compared to other language compilers, CPython is designed to compile as fast as possible, as lightweight as possible, and with as little ceremony as possible. There’s very little error checking and optimization which allows it to run fast and in small amounts of memory. It’s also fairly unique in that it can be run automatically without the user even needing to be aware that compilation is occurring.

You could compare this to C for example, which is more of a ‘compiled’ language and undergoes heavier and slower compilation before execution. We call this, **ahead-of-time compilation**. This is largely due to the fact that C is more low-level than Python, so there is greater potential for errors.

Now, once the source code is compiled to Python bytecode, it is then executed by a VM. CPython is both the compiler and the VM, so well it an **interpreter.** In contrast Jython and IronPython are just the compiler, so require an extra step to actually execute the program.



The interesting thing here is that bytecode isn’t actually readable by your CPU, and that’s why the VM is required to actually execute it. The **Python Virtual Machine (PVM)**’s primary function is for this purpose only; to translate bytecode into machine code, allowing the computer to execute the program. The reason for translating first into bytecode and then into machine code, is simply because it’s faster than translating from high level Python straight into machine code.

Of note, it’s actually in the compilation stage from CPython that the .py file is checked for syntax errors. These are different from runtime errors, which only occur after the .pyc bytecode file has been translated and executed by the PVM.

Lastly, let’s cover library modules. When you import libraries or modules in your Python program, CPython first checks if it’s built-in and if so, executes the corresponding C code. If not, the interpreter checks the list of directories defined in sys.path, the directory of the input script (i.e. if you created your own modules), and directories listed in the PYTHONPATH environment variable. If a corresponding file is found, CPython creates a new module object from that file. Note that library modules can and are often in compiled form to completely skip the need for startup time, and on top of that can be written in a different programming language entirely such as FORTRAN, which runs faster than Python when compiled. This compiled module is imported directly to the PVM, which then uses the module object to execute the program.