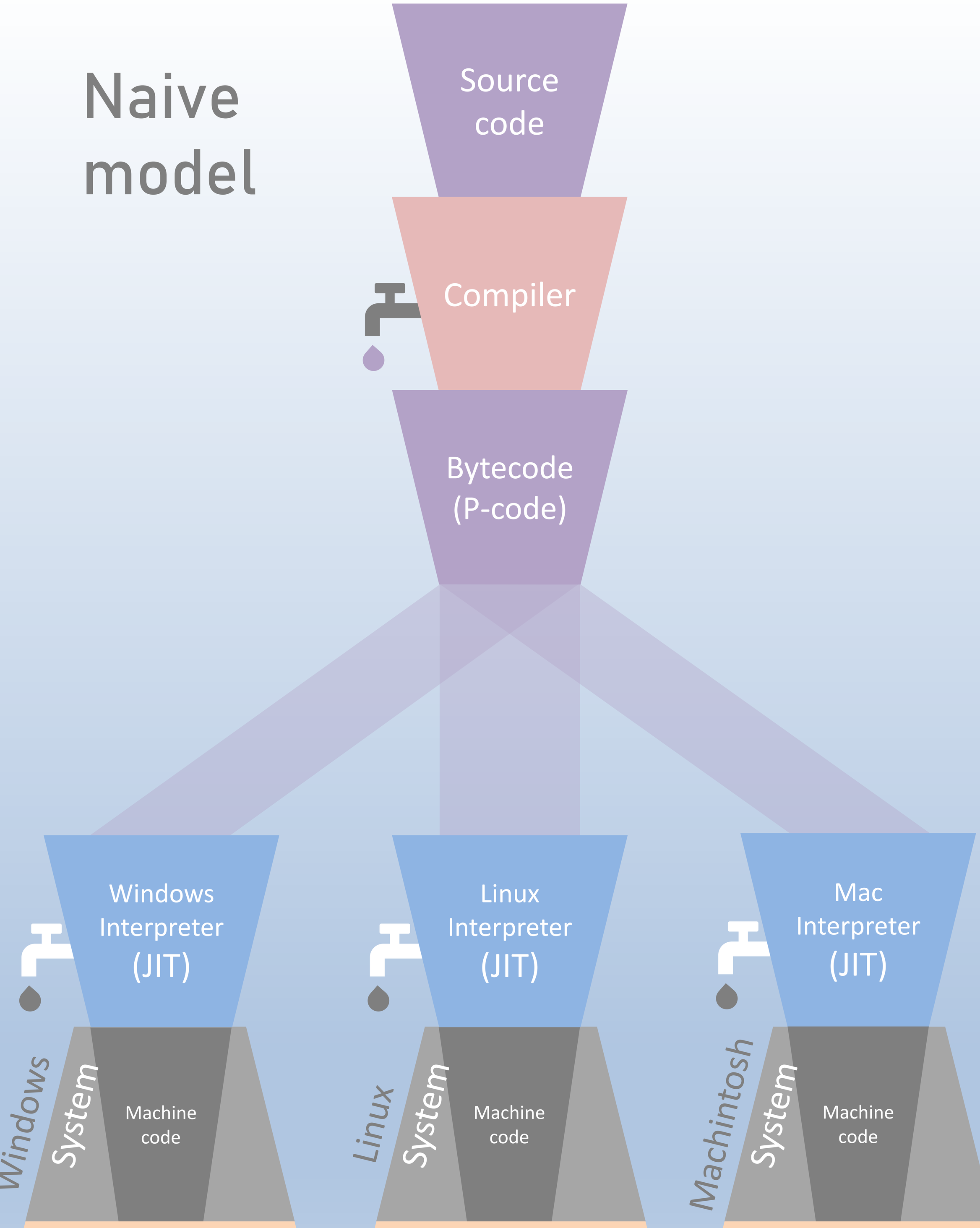
Naive model


Source code
Compiler
Bytecode (P-code)
Windows Interpreter (JIT)
Linux Interpreter (JIT)
Mac Interpreter (JIT)
Windows System
Machine code
Linux System
Machine code
Machintosh System
Machine code
Hardware
Hardware
Hardware

Bytecode portability and compilation vs interpretation. In an abstract fashion, it shows how most interpreted computer languages work today. It starts from the source code written by the programmer, which is assumed to be compiled to bytecode. The bytecode represents an abstraction of the initial source code. Bytecode is then used as it is on any platform, because there, whatever the platform is, it is met by an adaptation of the same virtual machine. This virtual machine makes a combination between interpretation and sporadic compilation (Just In Time compilation - JIT) to increase the execution speed of the software implementation. Note that "native code" and "machine code" have the exact same meaning across all figures that are alike. This particular figure contains the words "Native code" instead of "Machine code" in order to fit the text inside the horizontal compressed shapes. Note also that in a different context, "native code" may refer to the only language understood by some abstract object. For instance, Java bytecode is the "native code" to the Java Virtual Machine. As it was the case in the old days, some interpreters of lower performance (not necessarily VMs) made a direct interpretation of source code, without an intermediate step like the use of bytecode. In principle, virtual machines could be designed to directly interpret high-level source code, short circuiting the source code security through obscurity or the multi-step optimization, or both. Thus, in such a case the "native code" would be the Java high-level source code. Also, please note that the abstract representation of the modules shown in the figure indicates a lack of extreme contrast between what is commonly called an interpreter or a compiler. That is, the compiler also does a little bit of interpreting and the interpreter also does a little bit of compiling.

Paul A. Gagniuc. *An Introduction to Programming Languages: Simultaneous Learning in Multiple Coding Environments*. Synthesis Lectures on Computer Science. Springer International Publishing, 2023, pp. 1-280.