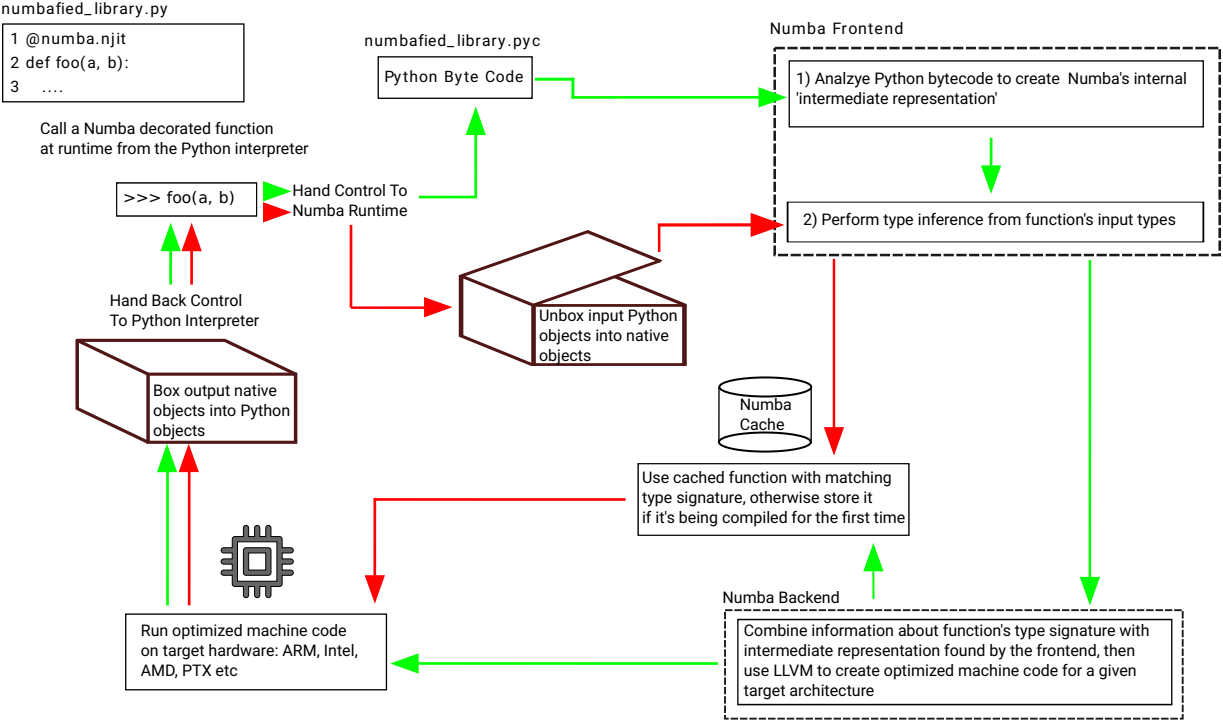
numbafied_library.py
1 @numba.njit
2 def foo(a, b):
3 ....
numbafied_library.pyc
Python Byte Code
Numba Frontend
1) Analzye Python bytecode to create Numba's internal 'intermediate representation'
2) Perform type inference from function's input types
Call a Numba decorated function at runtime from the Python interpreter
>>> foo(a, b)
Hand Control To Numba Runtime
Hand Back Control To Python Interpreter
Unbox input Python objects into native objects
Box output native objects into Python objects
Numba Cache
Use cached function with matching type signature, otherwise store it if it's being compiled for the first time
Numba Backend
Combine information about function's type signature with intermediate representation found by the frontend, then use LLVM to create optimized machine code for a given target architecture
Run optimized machine code on target hardware: ARM, Intel, AMD, PTX etc