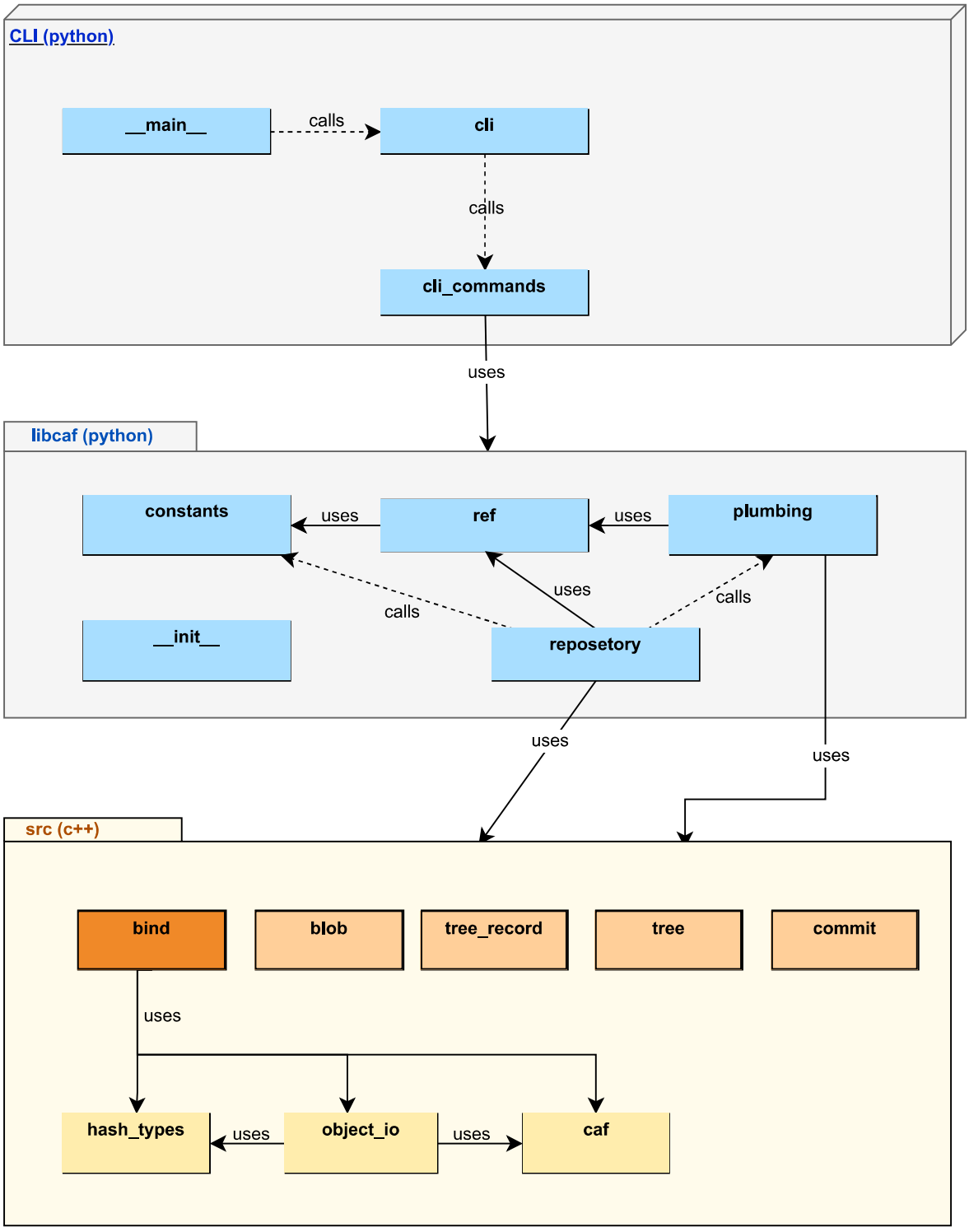
CLI (python)
__main__
calls
cli
calls
cli_commands
uses
libcaf (python)
constants
uses
ref
uses
plumbing
uses
calls
calls
__init__
repository
uses
uses
src (c++)
bind
blob
tree_record
tree
commit
uses
hash_types
uses
object_io
uses
caf

Part 2: Architectural Explanation

The CAF system is designed with a three-layered architecture that separates concerns effectively.

Command Processing flows top-down: The user interacts with the top-level CLI (Python) layer, where **main** serves as the entry point. cli handles argument parsing, and cli_commands orchestrates the high-level logic for each user command. This layer then delegates all control to the middle libcaf (Python) layer. Here, the repository module acts as the central brain, managing the state of the repository by using ref for reference handling and plumbing to access low-level functionality. The final layer, src (C++), contains the high-performance logic, and is responsible for data persistence – this is the layer where serialization of commit and tree objects is taking place, as well as file I/O operations.

The libcaf layer communicates with the src layer through the bind module, which exposes C++ functions and data structures to Python – making language integration possible. Lastly, error handling propagates upwards: a low-level C++ error is converted by pybind11 into a Python exception, which is caught by the cli_commands layer to provide a user-friendly error message (for instance, a runtime failure to open a file can propagate upward and appear to the user as a repository error). On top of that, errors associated with user's input are being caught instantly – without further assessment at deeper layers.