Problem:

The goal is to port DLib functions to be used on the web, except all we have is a Python file at the moment.

Issue is, Pyodide, the WASM runner for Python based WASM, will not work, because our project has cpp dependencies within DLib that cannot be cross-compiled using only CPython.

Solution:

Use the DLib library from CPP source that were used in the Python script, and compile both to WASM using Emscripten's emcc / em++ compilers.

Rewrite the Python script in CPP in order to link the Emscripten compiled libraries to the final build file. Then run the final compile using those libraries, their includes, and any other static data.

Compiling everything has its benefits. emcc has fine-grained control over execution optimization, using Emscripten's Optimization Flags and Compiler Settings.

Why WASM?

WebAssembly is a binary instruction format designed for efficient execution and compact representation. Its main goal is to enable high performance applications on the Web, but it does not make any Web-specific assumptions or provide Web-specific features, so it can be employed in other environments as well. It seemed like a rather fitting target, especially when paired with Emscripten. For more information on the WebAssembly stack machine, WASM Memory Model, and available environments, see the core spec.

Why Emscripten?

Emscripten essentially acts as a bridge between traditional C/C++ code and WebAssembly. WebAssembly itself is a low-level format designed to run in a secure and portable manner in web environments, but without Emscripten, most developers would need to write WebAssembly manually or use languages with built-in support for Wasm (like Rust). Emscripten simplifies this process, making WebAssembly accessible for C/C++ applications through replacing clang with emcc, a compiler that targets LLVM's Intermediate Representation instead of native machine code.

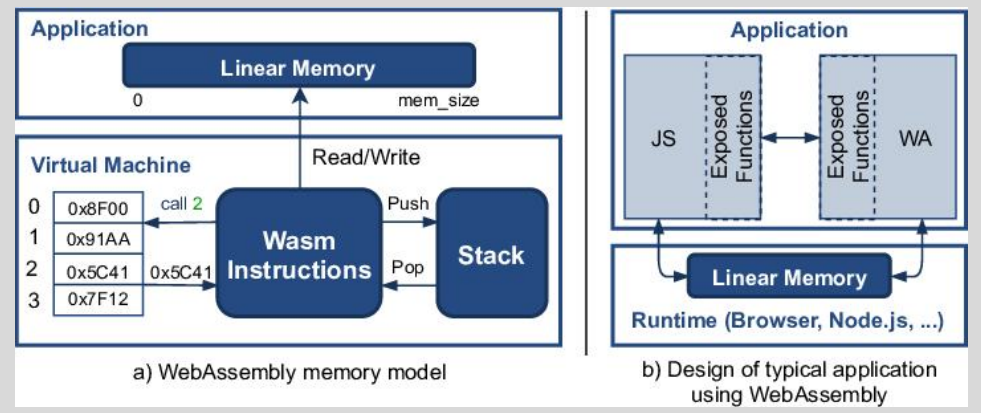
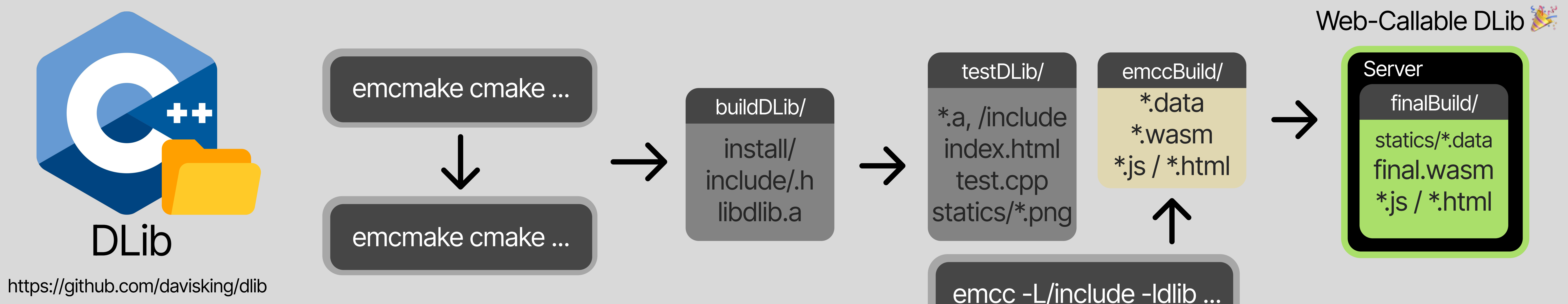


Fig 1: Web Assembly Memory & Environment.

Note: Runtime can also include Wasmtime or Wasmer for running standalone.

Compilation Flow (EMCC → LLVM IR → Binaryen optimizer → out.wasm)



```
emcmake cmake ...
emcmake cmake ...
emcc -L/include -ldlib ...
```