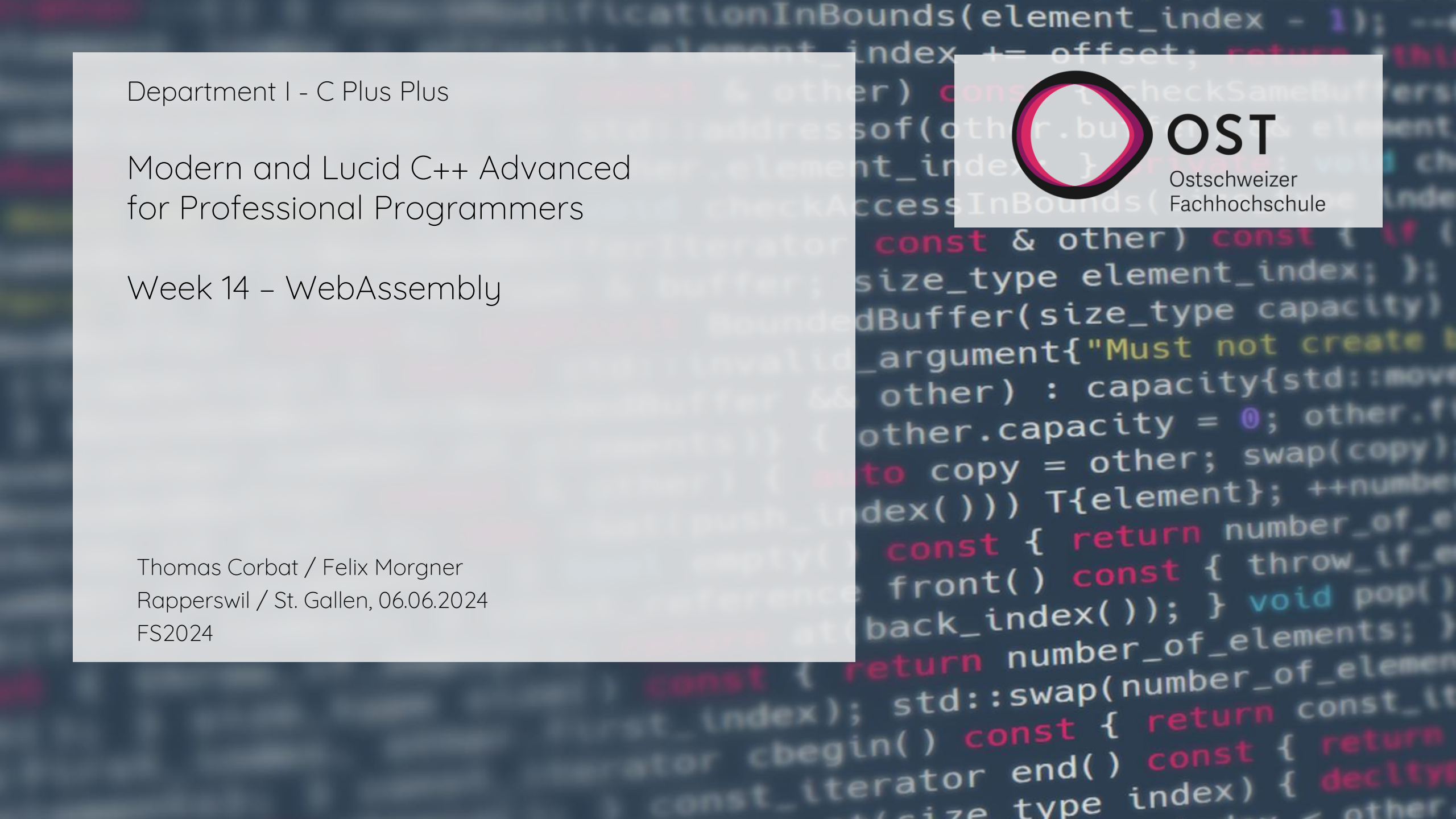
Department I - C Plus Plus

Modern and Lucid C++ Advanced for Professional Programmers

Week 14 – WebAssembly

Thomas Corbat / Felix Morgner
Rapperswil / St. Gallen, 06.06.2024
FS2024



Agenda

- **Recap Week 13**
- **Introduction to WebAssembly**

- **Participants should ...**
  - … know the basics of WebAssembly and WebAssembly Text
  - … be able to compile a C++ application to WebAssembly
  - … be able to run the application in a browser

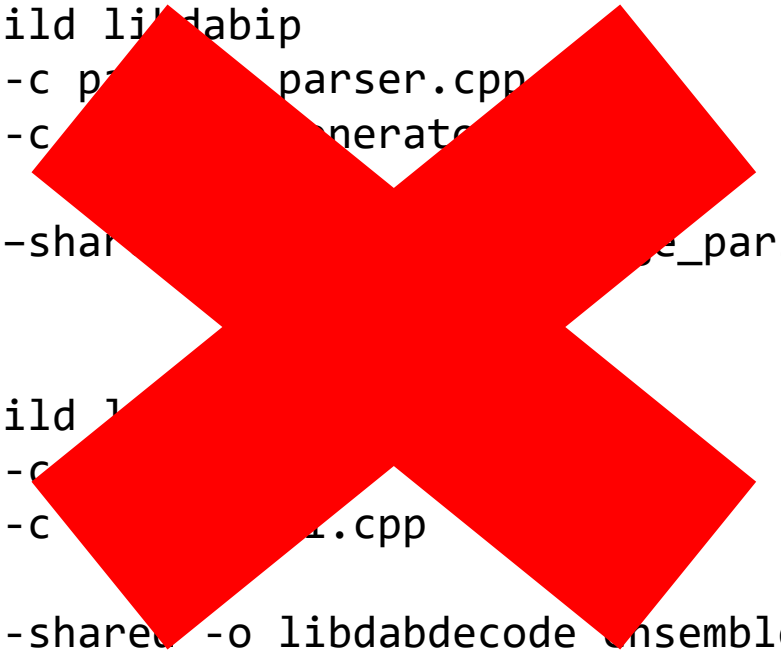
Recap Week 13

OST
Ostschweizer
Fachhochschule

- **Five libraries**
  - All depending on a common infrastructure library
- **Two executables**
  - Depend on some or all of the libraries
- **Two target-platforms**
  - Linux on x86_64 and armv7
  - OS X
- **Code will change owners**
- **4 months time-frame**

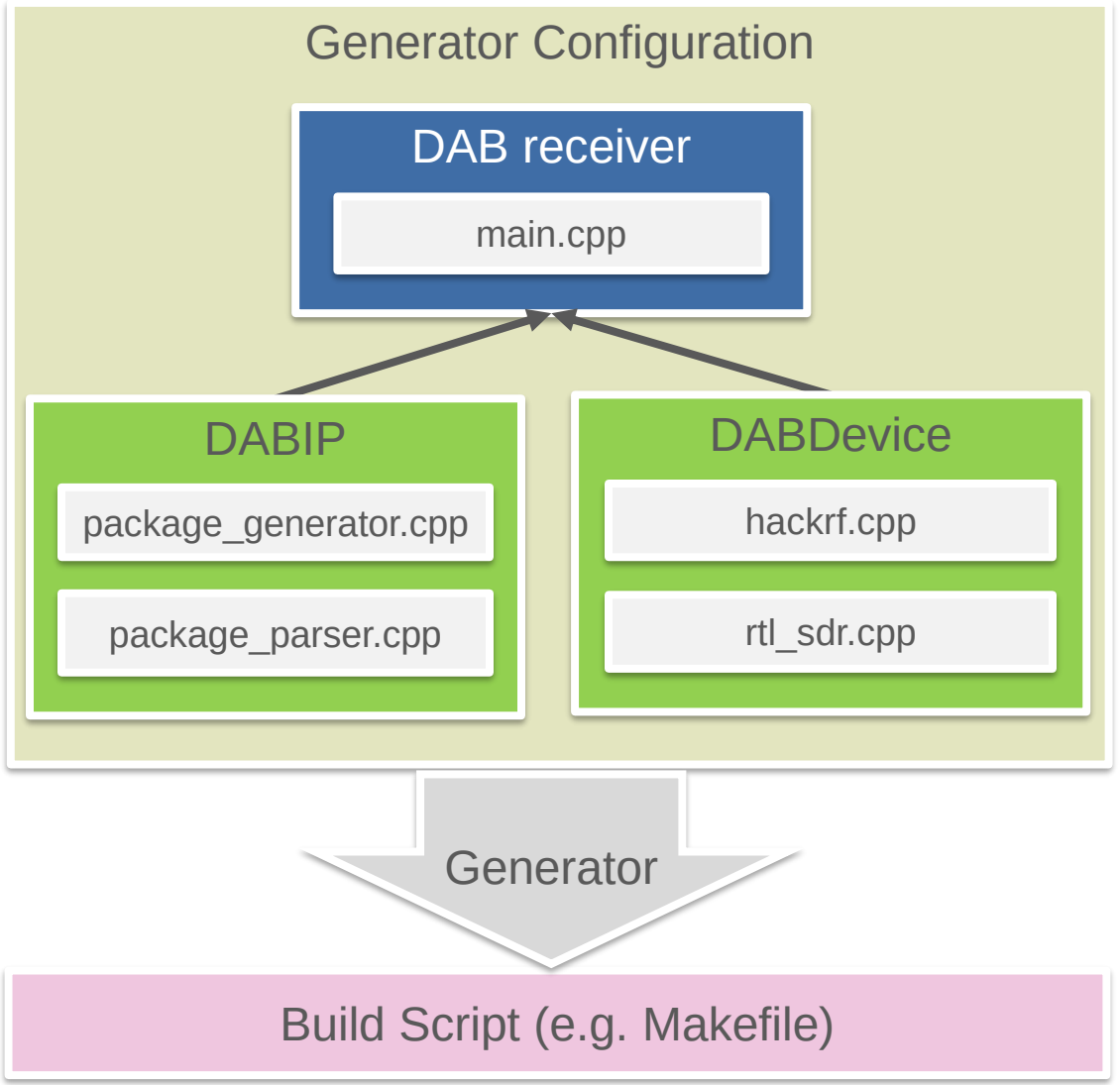
- **Write a script that…**
  - Compiles each source file
  - Links all object files together
  - Repeats that for every target
- **DON'T! Because…**
  - … every source file get built every time!
  - … the commands tend to be platform specific
  - … build order must be managed manually
  - … scripts tend to become messy over time

```
#!/bin/bash

# Build li dabip
gcc -c p      parser.cpp
gcc -c        nerat
…
gcc –shar                 e_parser.o
…

# Build l
gcc -c
gcc -c        1.cpp
…
gcc -share -o libdabdecode nsemble.o …

# Build <you get the idea>
```



- **Make-style Build Tools**
  - Run build scripts
  - Produce your final products
  - Often verbose
  - Use a language agnostic configuration language
- **Build Script Generators**
  - Generate configurations for Make-style Build Systems or Build Scripts
  - Configuration independent of actual build tool
  - Advanced features (download dependencies, etc.)

- **Idea: Take a step back**
  - Define what we want to achieve, not how to do it
  - Work on a higher level
  - Let the create the actual build configurations
- **Platform independent build specification**
- **Tool Independent**
  - Often can generate IDE projects
  - Support multiple build tools



- **CMake includes CTest**
  - Enable CTest using `enable_testing()`
  - Create a “Test Runner” executable
    - Make sure to include your suite sources!
  - Configure build environment:

```
$ cmake ..
```

  - Build the project:

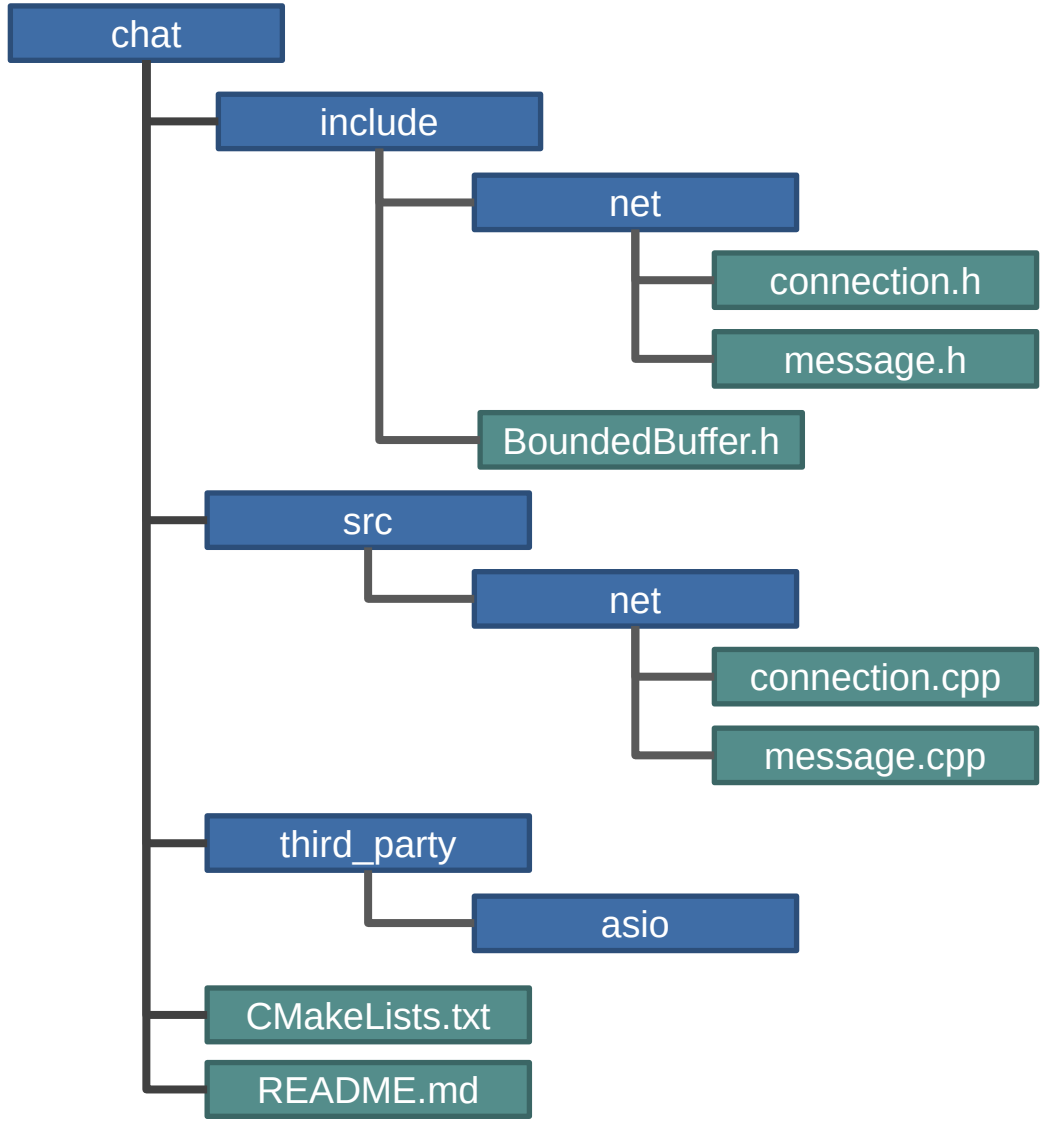
```
$ cmake --build .
```

  - Run `ctest`

```
$ ctest --output-on-failure
```

```
cmake_minimum_required(VERSION "3.12.0")

project("answer" LANGUAGES CXX)

enable_testing()

add_library("${PROJECT_NAME}"
  "answer.cpp"
)

add_executable("test_runner"
  "Test.cpp"
)

target_link_libraries("test_runner" PRIVATE
  "answer“
)

target_include_directories("test_runner" SYSTEM PRIVATE
  "cute“
)

add_test("tests" "test_runner")
```

- **Headers live in the “include” folder**
  - Add subfolders for separate subsystems if needed
- **Implementation files live in the “src” folder**
  - Make sure that subfolder layout matches the “include” folder (**consistency**)
- **Put third-party projects/sources in a “third_party” or “lib” folder**
- **Test resource live in the “test” folder**
  - The test folder will have src, include, and third_party subfolders if required
- **Build configuration files should live in the root of your project**



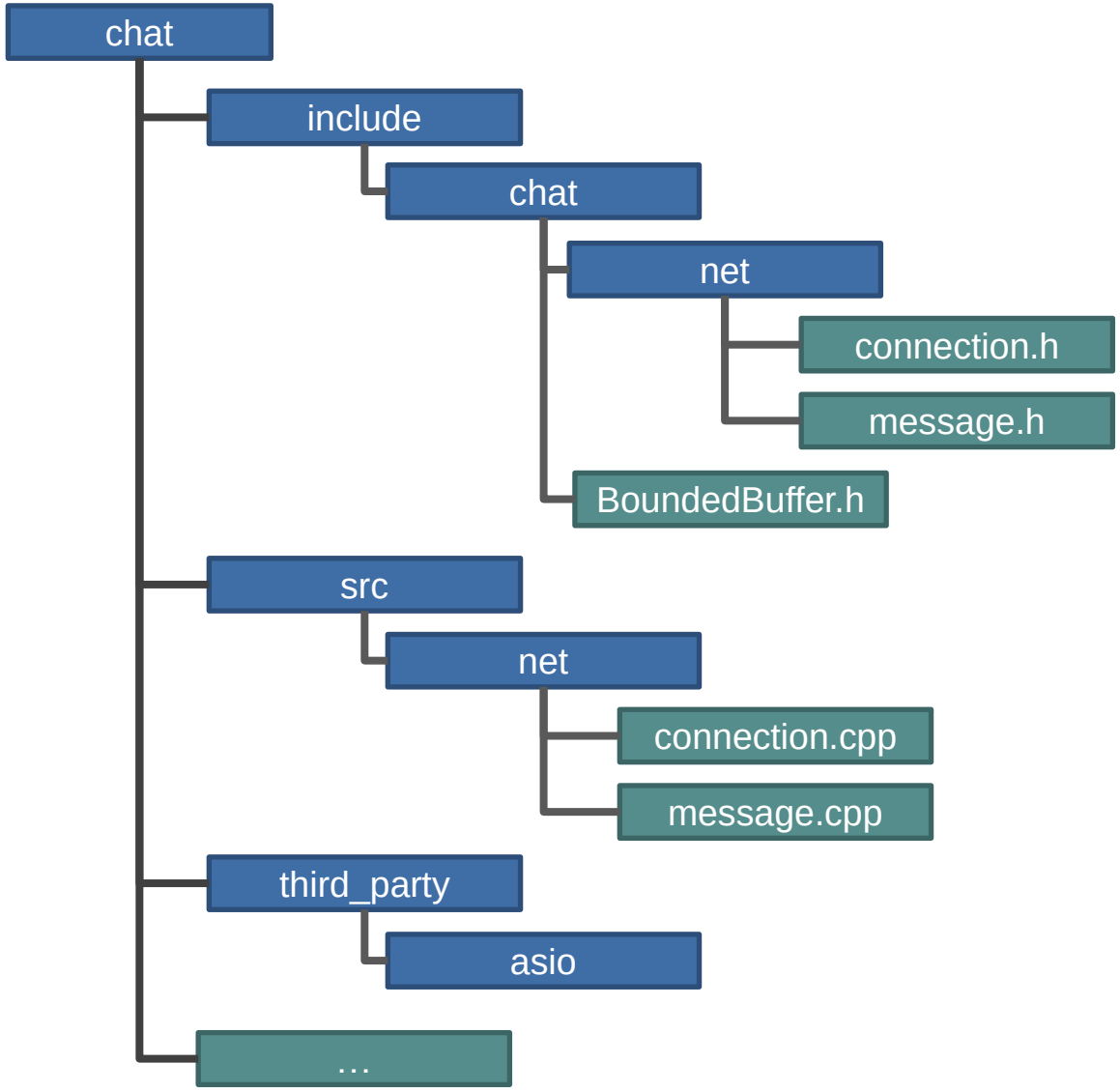
- **Libraries may benefit from a slightly different layout**
  - You will need to ship your headers
  - Your headers might have very generic names
- **Idea: Introduce another nesting level for your headers**
  - Use the name of your project

```
#include "net/message.h"
```

… becomes …

```
#include "chat/net/message.h"
```

  - Helps avoid filename clashes



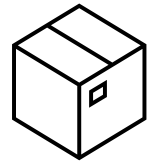
WebAssembly Basics

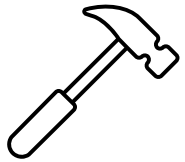
Based on The Art of WebAssembly

Why WebAssembly?

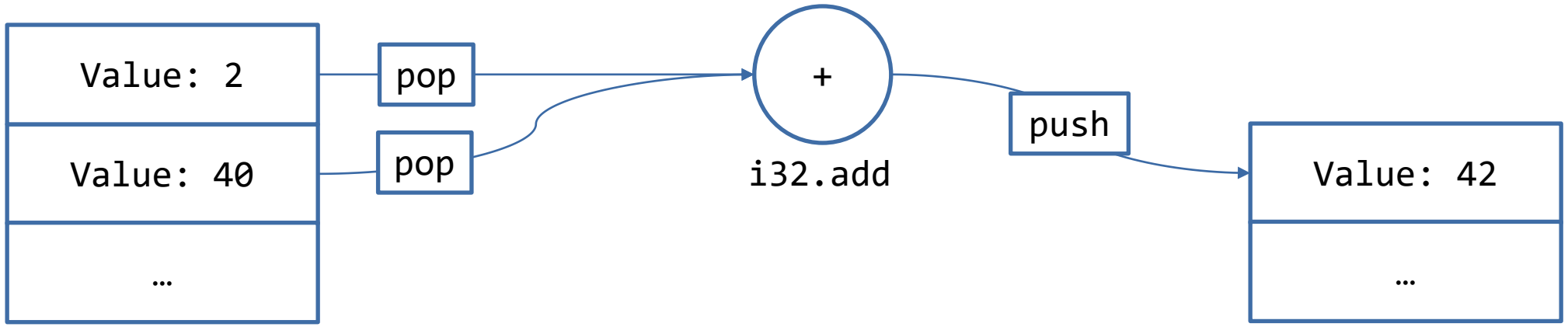
- **Can be run in modern web browsers**
- **No plug-in required (opposed to Java in browsers)**
- **Low-level language with compact representation**
- **High performance (number crunching, rendering, etc.)**
- **Enables cross-compilation of languages like C++ (Emscripten) and Rust (wasm-pack) to the web**
- **Integrates well with JavaScript**

- **Instruction Set Architecture for a stack machine (similar to JVM)**
- **Targets a virtual machine, which can be implemented for various physical machines**
- **Small binary size (for fast download) and portable**
- **Significantly faster than corresponding JavaScript code (in the browser)**
  - Compiled and optimized upfront -> binary format
  - JavaScript is text that needs to be parsed, interpreted, JIT compiled and optimized on the fly
- **Less versatile for the programmer as its features is limited compared to JavaScript**
  - Requires another high-level language as source for reasonable applications
  - Best suited for high-performance use cases





- **No registers (as in register machines)**
- **Operations are performed on the topmost elements of a stack**
  - Pop elements
  - Apply operation
  - Push result



WebAssembly Text (WAT)

- **WebAssembly is distributed in binary format**
- **WebAssembly Text is a mnemonic form of WebAssembly (its disassembly)**
  - (Super)human-readable
- **Instructions that work on the stack (push, pop, both)**
  - Instruction format `<Type>.<Operation>`
  - Example: `i32.add`
- **Declarations with keywords**
  - `Module, global, func, etc.`

```
(module
  (global $a_val (mut i32) (i32.const 1))
  (global $b_val (mut i32) (i32.const 2))
  (global $c_val (mut i32) (i32.const 0))
    (func $main (export "main")
      global.get $a_val
      global.get $b_val
      i32.add
      global.set $c_val
  )
)
```

- **Alternative syntax for structuring the code**

```
(module
  (global $a_val (mut i32) (i32.const 1))
  (global $b_val (mut i32) (i32.const 2))
  (global $c_val (mut i32) (i32.const 0))
  (func $main (export "main")
    (global.set $c_val
      (i32.add (global.get $a_val) (global.get $b_val))
    )
  )
)
```

S-Expression

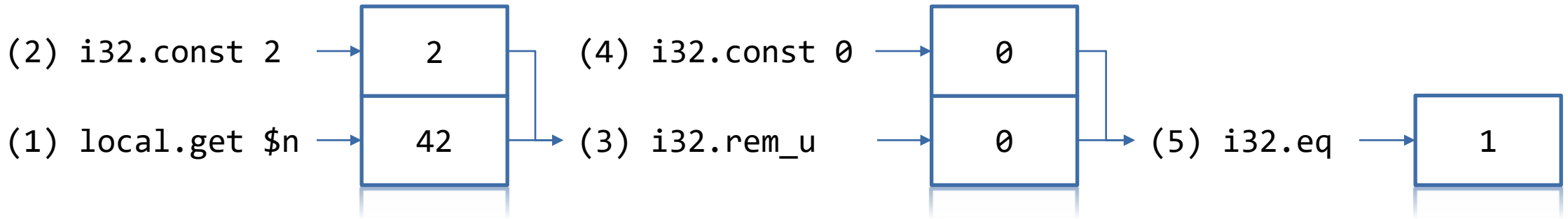
⇔

```
global.get $a_val
global.get $b_val
i32.add
global.set $c_val
```

Linear Syntax

- **Introduced by func**
  - Can be declared as “export”
- **Name after $**
- **Param(s) with type**
- **Result with type**

```
(module
  (func $even_check (param $n i32) (result i32)
    local.get $n
    i32.const 2
    i32.rem_u
    i32.const 0
    i32.eq
  )
)
```



WAT: Types

- **Integers**
  - i32 – 32 bit
  - i64 – 64 bit
  - 2s Complement
- **Floating point**
  - f32 – 32 bit
  - f64 – 64 bit
  - IEEE 754 representation

- **On JavaScript side the numbers will be mapped to 64-bit float**

- `.eq` – Test for equality
- `.ne` – Test not equal
- `.lt[_s|_u]` – Test for less than
- `.le[_s|_u]` – Test for less or equal
- `.gt[_s|_u]` – Test for greater than
- `.ge[_s|_u]` – Test for greater or equal
- `.and` – Bitwise AND
- `.or` – Bitwise OR
- `.xor` – Bitwise XOR
- `.eqz` – Float equals zero

- `_s` or `_u` suffix for signed and unsigned comparison
- Consumes the top two elements on the stack and pushes the result
- Example

```
local.get $x
local.get $y
i32.gt_s ;; pushes 1 on the stack if $x > $y
```

- **`.add` – Plus operation**
- **`.sub` – Minus operation**
- **`.mul` – Multiply operation**
- **`.div[_s|_u]` – Division operation**
- **`.rem[_s|_u]` – Remainder operation**

- **Consumes the top two elements on the stack and pushes the result**
- **Example**

```
local.get $x
local.get $y
i32.mul ;; pushes $x * $y on the stack
```

- `.and` – Bitwise AND
- `.or` – Bitwise OR
- `.xor` – Bitwise XOR
- `.eqz` – Float equals zero

- Consumes the top two elements on the stack and pushes the result
- Example

```
local.get $x
local.get $y
i32.xor ;; pushes $x ^ $y to the stack
```

- **if...else**

```
local.get $condition_i32
if
  ;; $condition_i32 not 0
  nop
else
  ;; $condition_i32 is 0
  nop
end
```

- **loop**

```
(loop $infinite_loop
  nop
  br $infinite_loop ;; jump to loop label
)
```

- **block**

```
(block $jump_to_end
  br $jump_to_end ;; jump to end of block
  nop ;; never executed
)
```

- **Every WAT application must be a module**
- **Top-level declaration**
- **;; is a line-comment**
- **(; ;) block comment (multi-line)**
  - Can be nested

```
(module
  ;; code
)
```

- **Allocate one page of 64KB memory**

```
(memory 1)
```

- **Maximum number of pages in an application is 65’536, i.e., 4GB memory**
  - Technical limitation as 32-bit addressing is used internally

```
(module
  (memory 1)
  (global $pointer i32 (i32.const 128))
  (func $init
    (i32.store
      (global.get $pointer) ;; store at address $pointer
      (i32.const 99) ;; value stored
    )
  )
  ;;...
  (start $init)
)
```

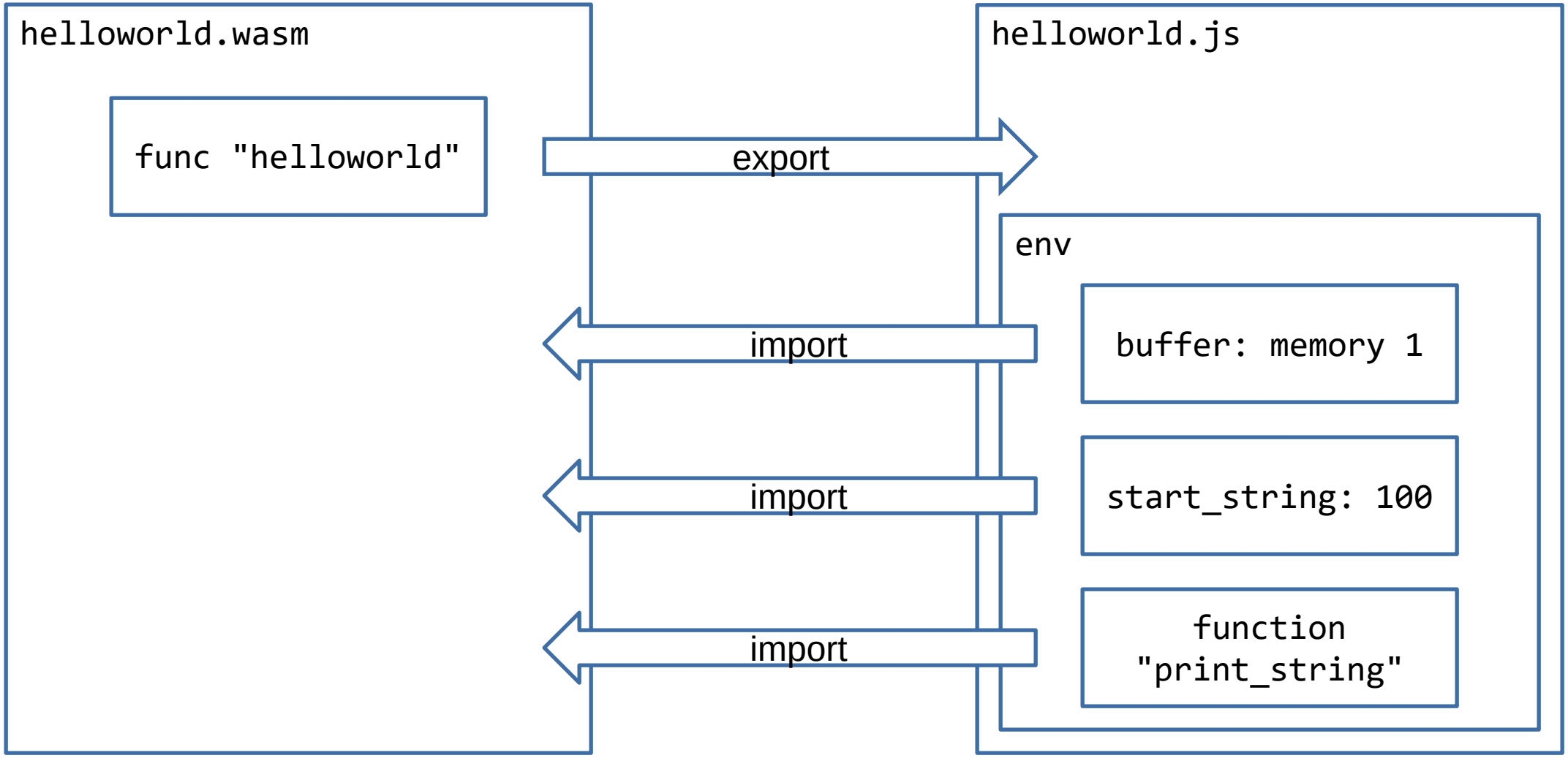
```
(module
  (import "env" "print_string" (func $print_string (param i32)))
  (import "env" "buffer" (memory 1))
  (global $start_string (import "env" "start_string") i32)
  (global $string_len i32 (i32.const 12))
  (data (global.get $start_string) "hello world!")
  (func (export "helloworld")
    (call $print_string (global.get $string_len))
  )
)
```

- **Compiled with `wat2wasm`**

Example from: The Art of WebAssembly

```
const fs = require('fs');
const bytes = fs.readFileSync(__dirname + '/helloworld.wasm');
let hello_world = null; // function will be set later
let start_string_index = 100; // linear memory location of string
let memory = new WebAssembly.Memory({ initial: 1 }); // linear memory
let importObject = {
  env: {
    buffer: memory,
    start_string: start_string_index,
    print_string: function (str_len) {
      const bytes = new Uint8Array (memory.buffer, start_string_index, str_len);
      const log_string = new TextDecoder('utf8').decode(bytes);
      console.log (log_string);
    }
  }
};
( async () => {
  let obj = await
      WebAssembly.instantiate(new Uint8Array (bytes), importObject);
  ({helloworld: hello_world} = obj.instance.exports);
  hello_world();
})();
```

Example from: The Art of WebAssembly


helloworld.wasm
func "helloworld"
helloworld.js
export
env
import
buffer: memory 1
import
start_string: 100
import
function
"print_string"

- **Emscripten: Toolchain for LLVM-based languages to WebAssembly**
- **Compiler Frontend (emcc)**
- **https://emscripten.org/**

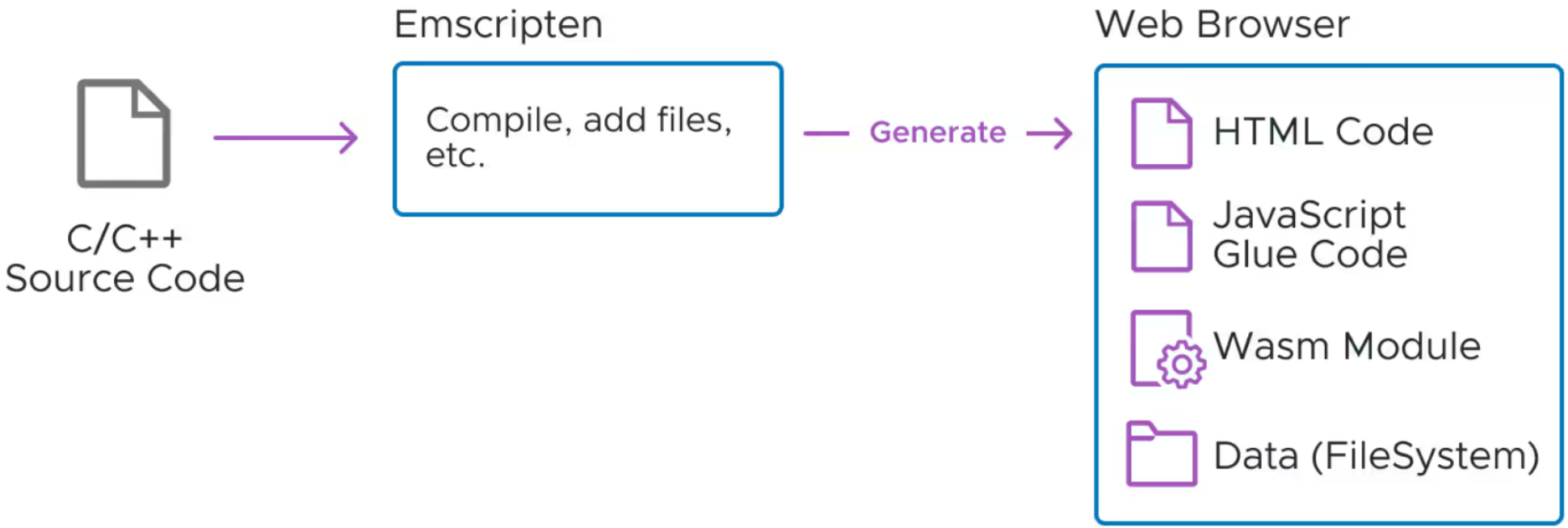


Image source: https://wasmlabs.dev/static/images/opt/CXJluOJrgw-1500.avif

- C++ Standard Library
  - Containers
  - Algorithms
  - Memory Management (`unique_ptr` etc.)
- Exceptions
  - Default disabled -> terminate!
  - Optional via JavaScript Exceptions
  - WebAssembly Exception Proposal
- POSIX Networking API via WebSockets
- Graphics APIs
  - OpenGL / EGL translated to WebGL
  - SDL
- Multithreading
  - Support for POSIX Threads
  - Atomic Operations
- File I/O via embedded file system
- OpenAL support via WebAudio

```cpp
#include <iostream>

auto main() -> int {
  std::cout << "CplA on the Web!\n";
}
```

- `em++ -std=c++20 hello.cpp`
  - Generates WASM and JS file
  - JS file provides initialization
  - Executable with NodeJS or similar
  - Useful in backend
- `em++ -std=c++20 hello.cpp -o hello.html`
  - Generates HTML shell
  - Can be loaded in browser
    - May require local webserver!
  - Useful in frontend

- **Autotools**
  - Emscripten provides “wrappers”
  - `emconfigure`
    - Wraps execution of configure scripts
    - E.g. `emconfigure ./configure`
  - `emmake`
    - Wraps execution of make itself
    - E.g. `emmake make all`

- **Cmake**
  - Emscripten provides a single “wrapper”
  - `emcmake`
    - Wraps execution of `cmake`
    - Provides additional flags under the hood
    - `emcmake cmake -S . -B build`
  - Can use libraries just as usual
  - Can generate HTML shell by specifying suffix
    - ```
      set_target_properties(tgt_name
          PROPERTIES
          SUFFIX ".html"
      )
      ```
      ```

- **WebAssembly is designed to be as close to “native” performance as possible while still being easily portable**
  - Enables high performance execution in web browsers as well as in backend applications using NodeJS.
  - However: It is currently limited to 32-Bit applications, meaning that memory is limited to 4GB.
- **The design of WebAssembly makes it possible to use different high-level languages to write applications for the web.**
- **JavaScript and WebAssembly can interact with each other**
  - JavaScript can call WebAssembly functions, and share data with it, and vice versa.
- **Emscripten provides a toolchain enabling the use of modern C++ on the web**
  - It integrates well with existing build systems like GNU Autotools or CMake.
  - Makes it possible to ship the same code natively or via the web!