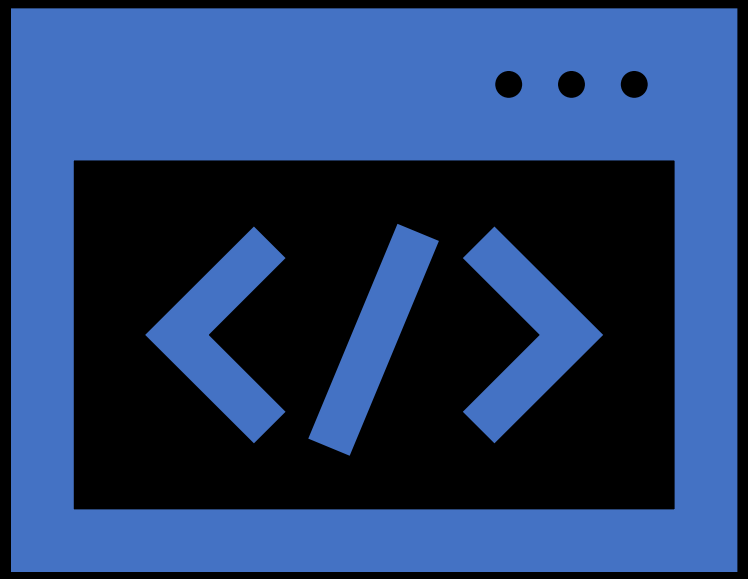


G. Mikołaj Boć

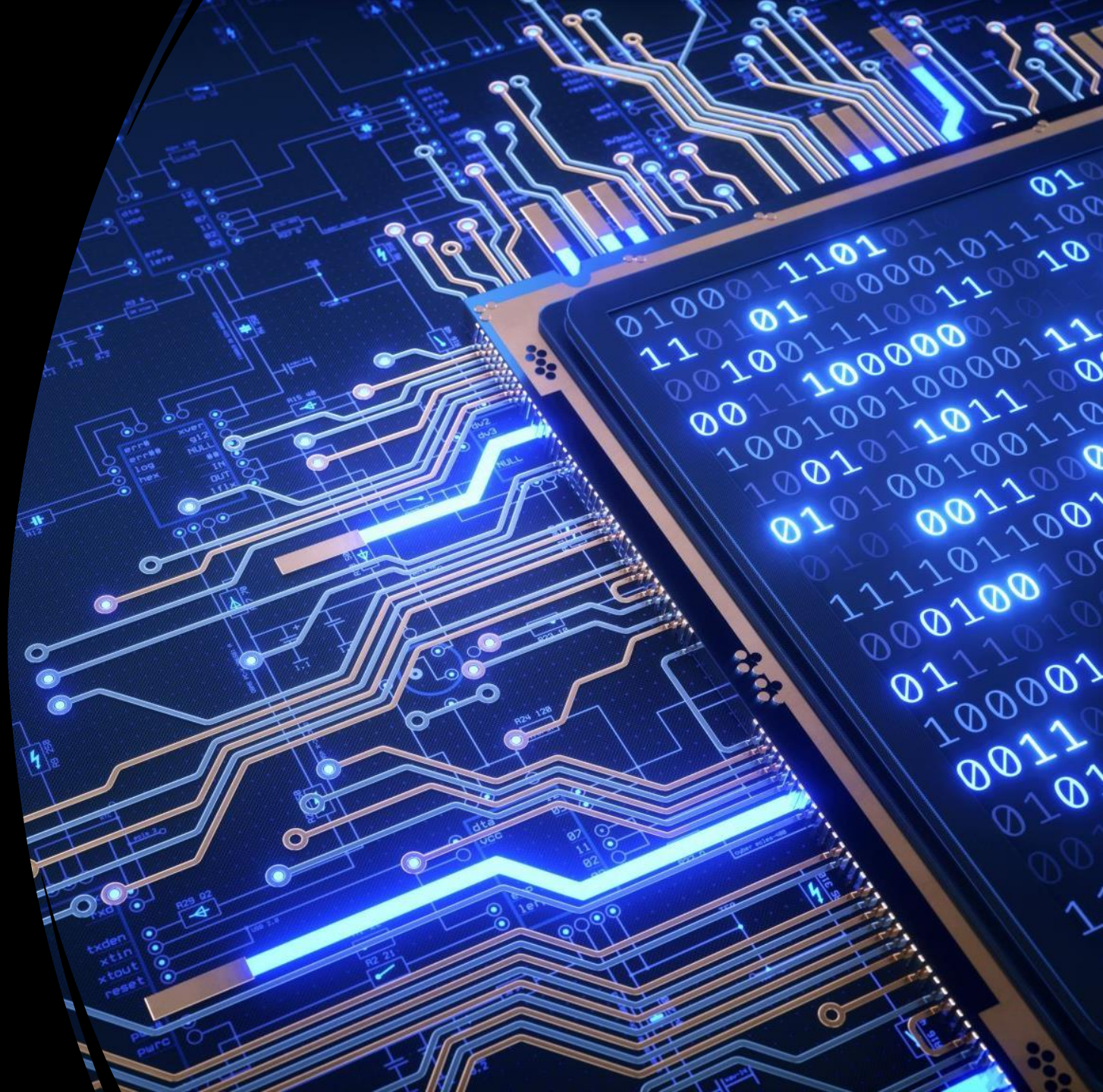
WASM - doing C++ the web way

Agenda

- What is it?
- Where/how can I run it?
- How fast is it?
- What are the potential challenges?

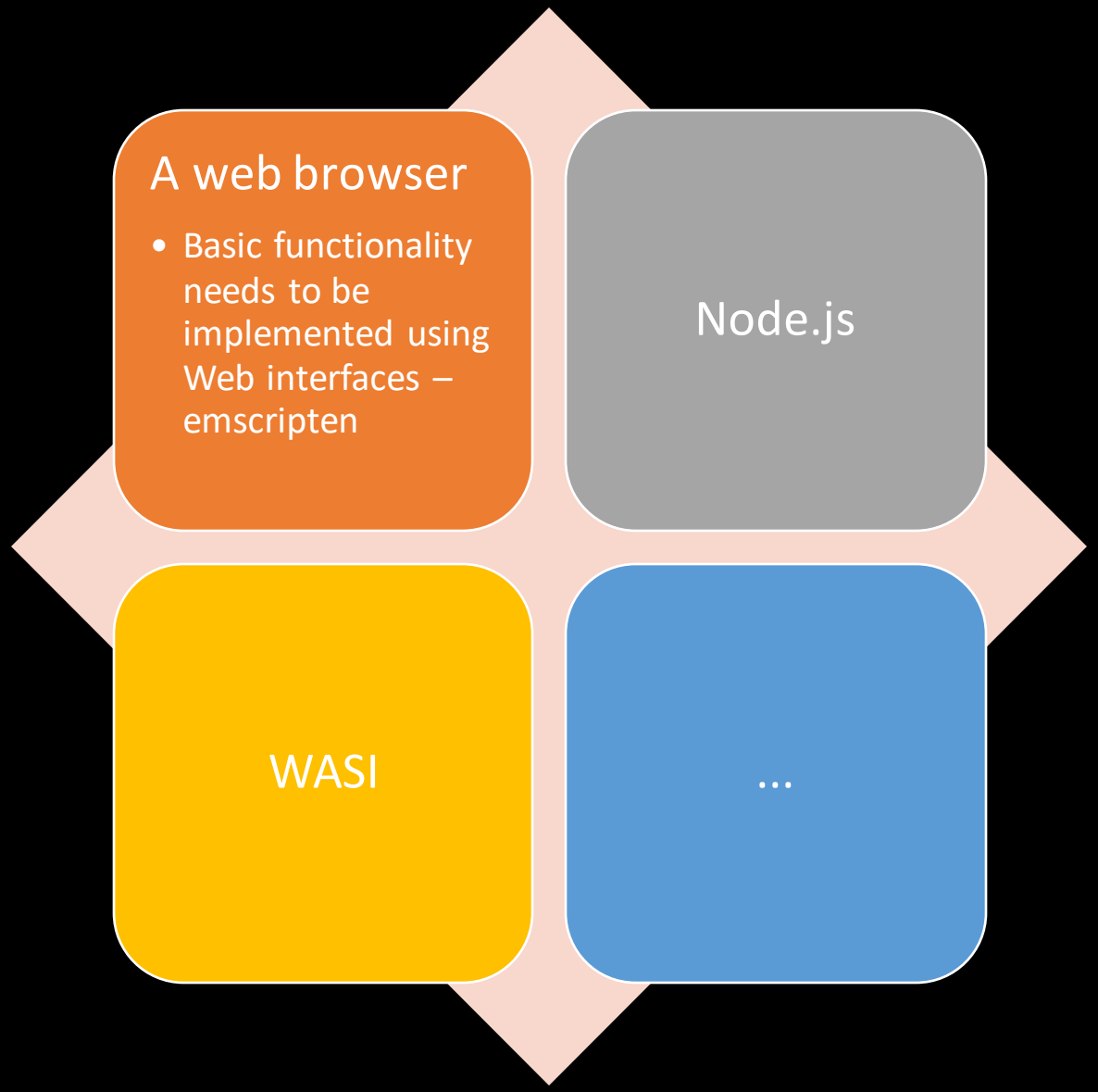
What is it?

- Binary instruction format
- Portable – virtual machine
- Some call it the holy grail of portability
- Tiered compilation to machine code, e.g., Chromium's **Liftoff** + **TurboFan**
- Multitude of languages supported via various toolchains (C++ among them)
- With little extra effort, as debuggable as JS on the web



Where can I run it?

"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"



WasmBoy (Web Assembly, Assemblyscript)
Frames Run: 2500
Current FPS Average: 554

WasmBoy (Typescript)
Frames Run: 2500
Current FPS Average: 475

WasmBoy (Typescript, Closure Compiled)
Frames Run: 2500
Current FPS Average: 526

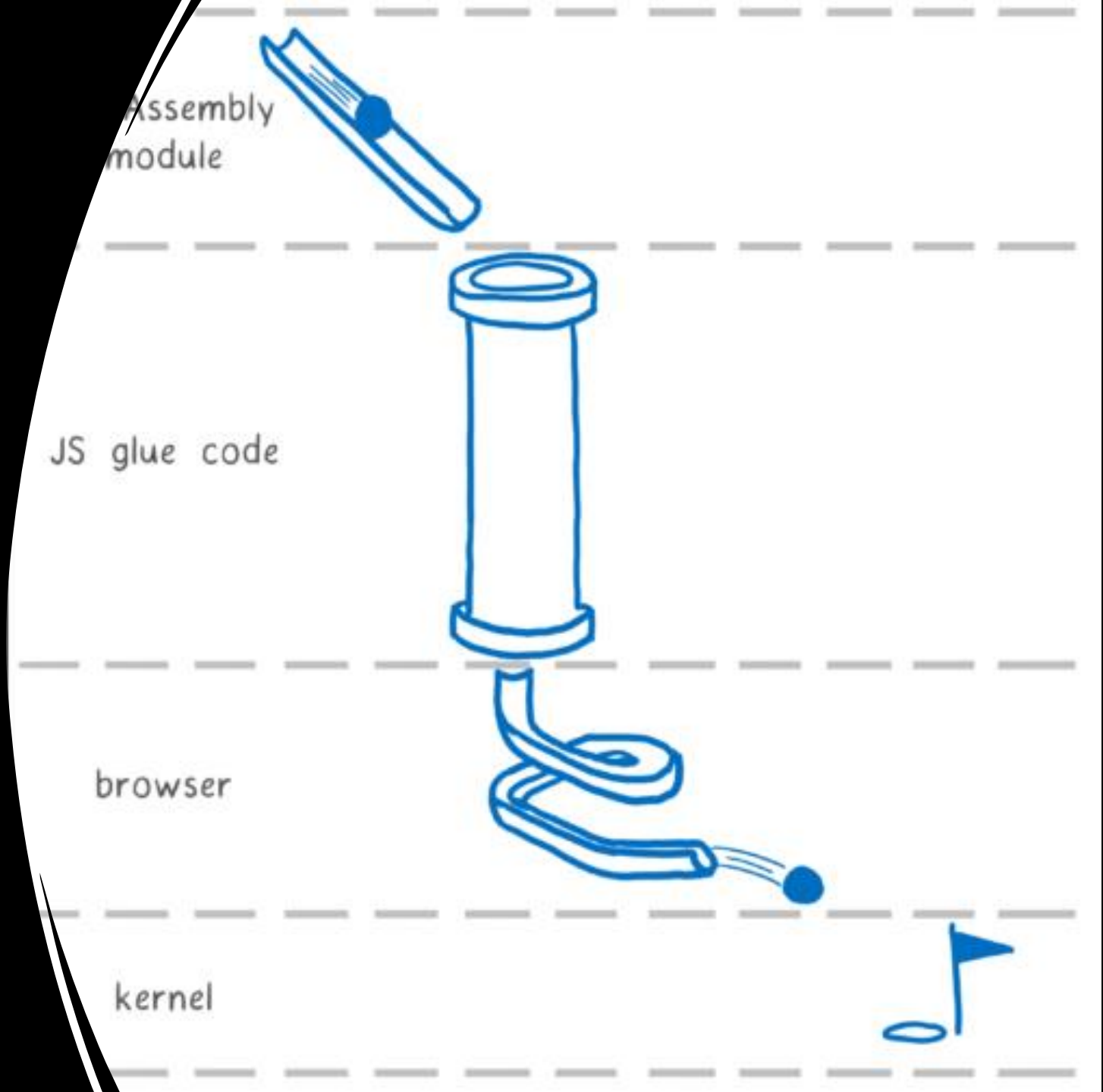
How fast is it?

- Vs JavaScript, almost always positive performance impact, sometimes substantial (7% - 50% faster in various studies)
- Vs native, various studies
  - Small, focused scientific programs ~10% slower
  - Larger programs up to 50%-250% slower

emscripten

- Complete toolchain to compile C/C++ to WASM (LLVM + Binaryen + Closure and others)
- Implementation of system APIs
  - File access
  - pthreads using web workers
  - Emulated POSIX TCP sockets
  - OpenAL with Web Audio
  - Wrappers for use in C++ codebase
    - emscripten_fetch, emscripten_set_X_callback
  - ...
- Inline JS support
- JS-to-C++ transliteration via emscripten::val
- Convenience code for using web APIs
  - WebGL

emscripten



What are the potential challenges?

- Permissions – sandbox in renderer process
- File access
- The async vs sync dilemma
- Linking – system libraries
- Aggressive symbol removal
- Executing apps for output
- Starving the main loop
- Threading <-> web workers
- Copying data between memory areas

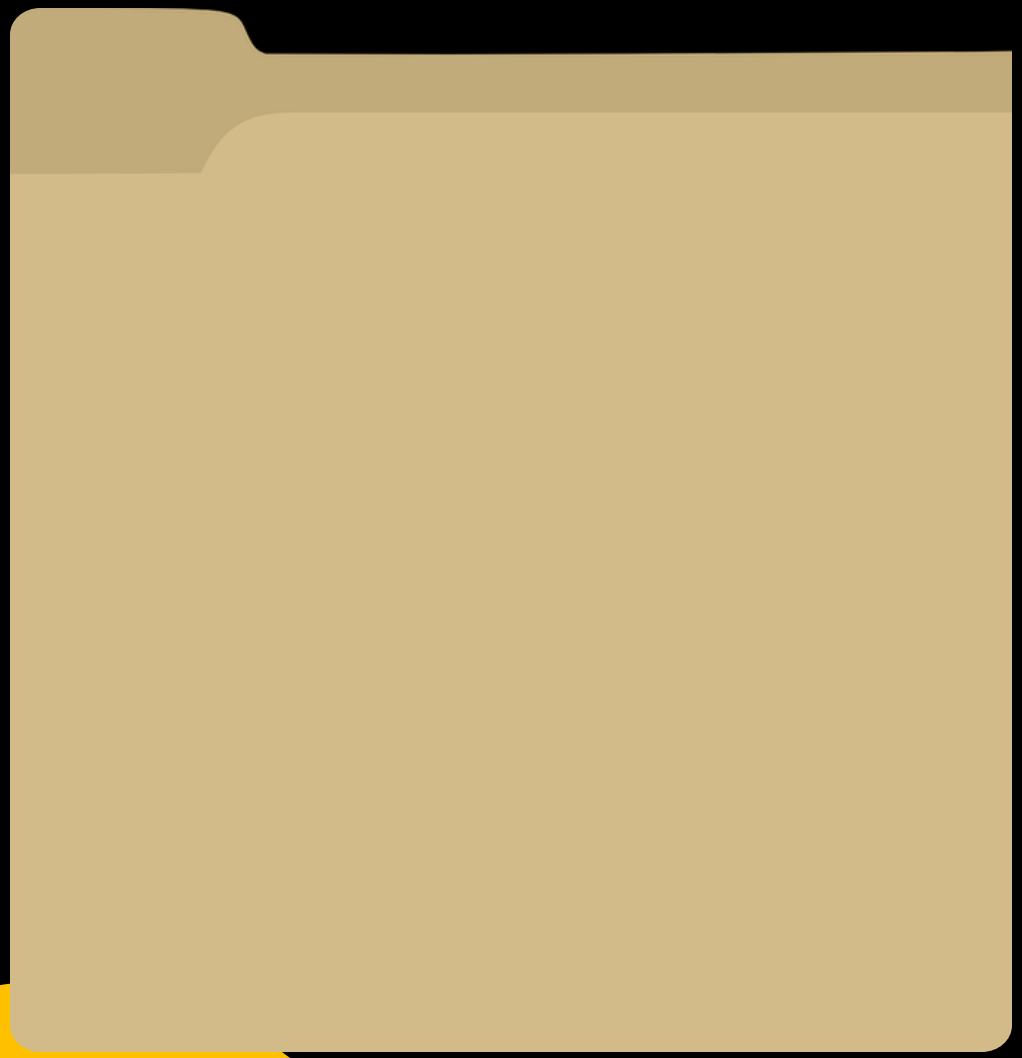
Permissions

Accessing certain APIs requires special permissions

Video recording, audio recording, notifications, clipboard access etc.

Need to think when to request the permissions

Transient activation needed for requesting permissions



File access

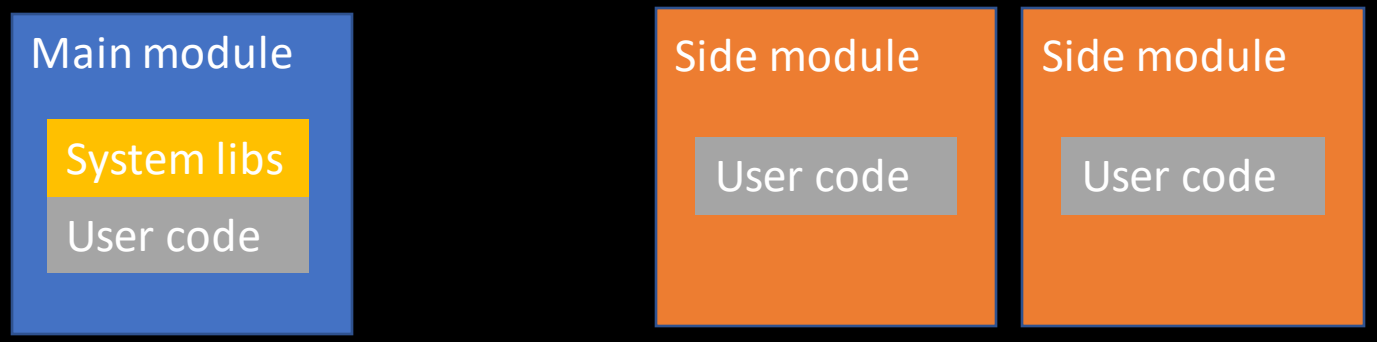
- File access is heavily limited
- Various modes that emulate a filesystem
  - MEMFS
  - NODEFS
  - IDBFS – has to be synchronized so a bit of an overhead
  - …
- Can also access the filesystem via the filesystem API
  - show(?:Open|Save)FilePicker – only available on Chrome and needs OPFS/permissions and transient activation

Async operations on the web

- Mostly async operations on the web – promises and async callbacks to events
  - Sync emulation in emscripten - hogging the main event loop
  - emscripten_set_main_loop
- Starting an event loop is a challenge
  - Asyncify
  - Moving the main thread to a worker
- emscripten::val + callbacks
  - Promise is particularly problematic

Linking

- Apps need to ship system library implementations
- Dynamic:
  - Side modules + main modules



- Load time
  - Caching helps with common libraries

Aggressive symbol removal

- Tries to keep the binary small
- Dynamically calling system functions that are deemed unused in the binary is impossible if the function is not explicitly marked for inclusion

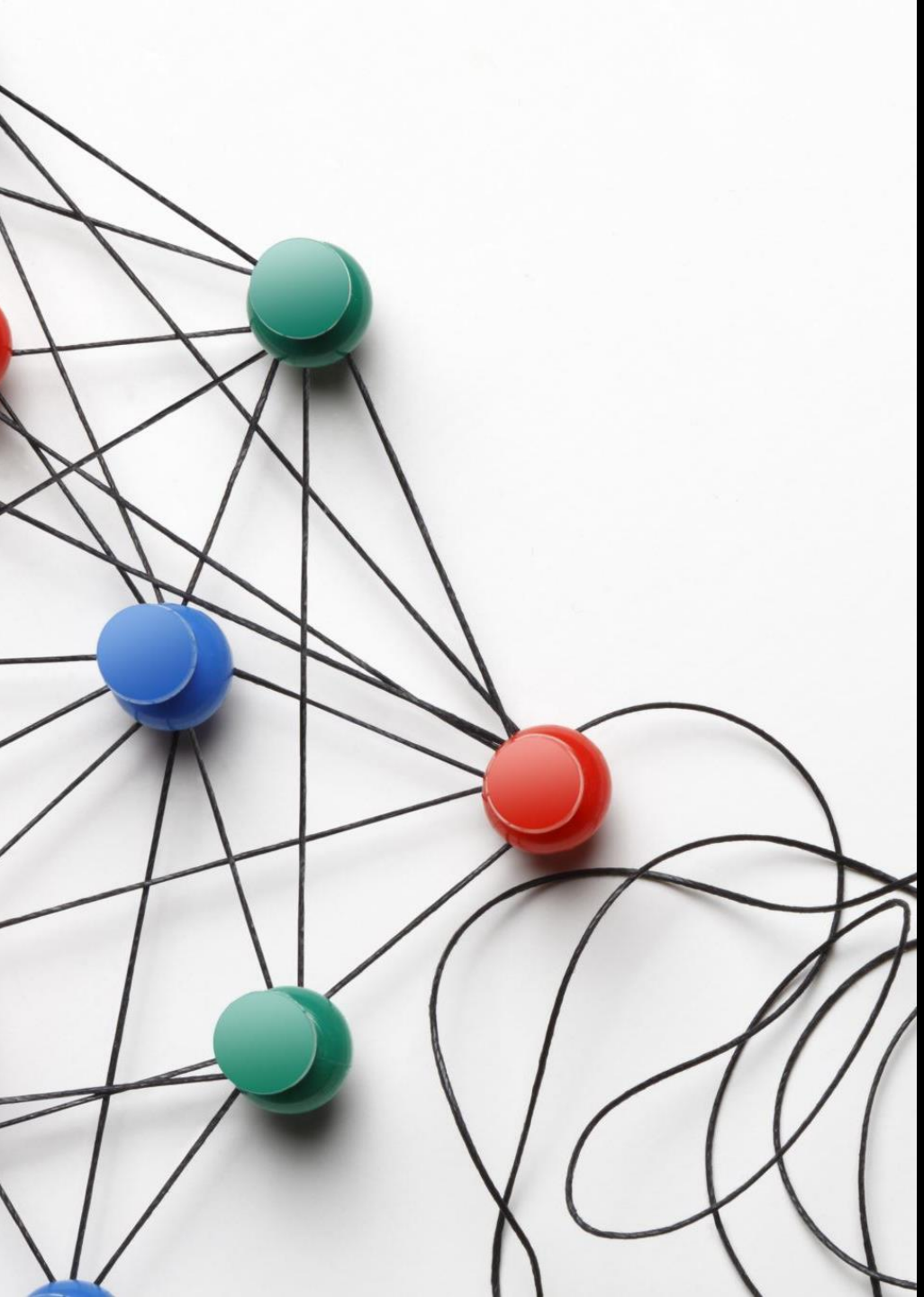
```
// Volatile is to make this unoptimizable, so that the function is referenced, but is not
// called at runtime.
volatile bool doIt = false;
if (doIt)
    emscripten_set_wheel_callback(NULL, 0, 0, NULL);
```

Executing apps

- Simply loading the generated html file
  - Cannot make the program a part of an execution pipeline
  - Difficult to get the output
- Using web APIs + JS to load the module
  - Most control
  - More work
  - emscripten quirks – more complicated functionality needs a lot of code reading
- emrun
  - Sets up a local server which reads POST data sent from the application
  - Application's output redirected to the server using a js preamble file
  - Problems:
    - Selecting a browser
    - Port clashes

Starving the main loop

- When running long operations, it is necessary to yield to the main event loop
- There's an option to move the main application thread to a secondary thread to avoid such problems
  - Target of experiments in Qt now



Threading

- Thread implementation needs SharedArrayBuffer to work
- Extra worker file generated
- Quirks
  - Proxying on child threads
    - Can lead to deadlocks!
  - Busy-waiting for phtead_join, pthread_mutex_lock, usleep, etc.
  - pthread_create needs to return to the main event loop
    - PTHREAD_POOL_SIZE – if you expect to run a certain number of threads at a time
    - PROXY_TO_PTHREAD – runs main on a worker. Introduces other problems

Memory areas

- Emscripten heap
- JS memory
- Might need to copy memory around to get the benefits of WASM
  - Data from direct calls to javascript
- Copying not always necessary – typed_memory_view

Thank you!

It doesn't end

- By default, runtime does not exit
  - Stdio streams not flushed
  - Global destructors not called
  - atexit callbacks not called
- -s EXIT_RUNTIME