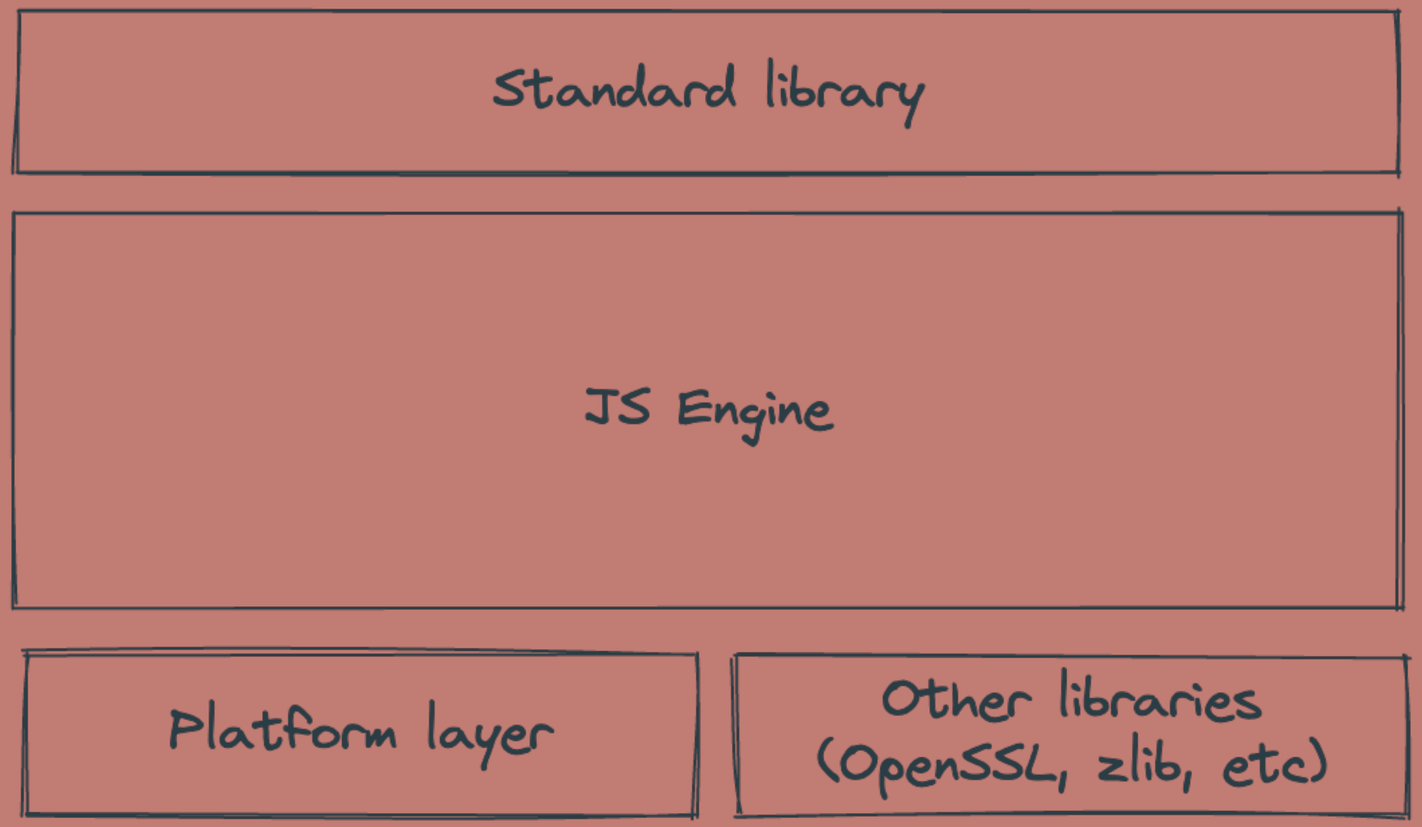
Building a tiny JavaScript runtime with QuickJS

How I built a JavaScript runtime with Duktape
and then rebuilt it with QuickJS

Me

- Not a *real* JavaScript developer
- Not a language runtime expert
- A tinkerer
- Likes to write good-old-unsafe C
- Likes *small* libraries

What's in a runtime



Let’s build one!

- I started SkookumJS in 2016
- Heavily inspired by the Python runtime
- Using Duktape as the JS engine
- POSIX style API
- CommonJS all the things (RingoJS looked cool)
- **https://github.com/saghul/sjs**

Duktape

- Embeddable JS engine, focused on portability
- ES5/5.1 with some ES6 and beyond features
- Minimal dependencies
- CommonJS style modules
- Stack style API, like Lua

Duktape

```c
#include <stdio.h>
#include "duktape.h"

int main(int argc, char *argv[]) {
  duk_context *ctx = duk_create_heap_default();
  duk_eval_string(ctx, "1+2");
  printf("1+2=%d\n", (int) duk_get_int(ctx, -1));
  duk_destroy_heap(ctx);
  return 0;
}
```

Duktape

```c
#include <stdio.h>
#include "duktape.h"

int main(int argc, char *argv[]) {
  duk_context *ctx = duk_create_heap_default();
  duk_eval_string(ctx, "1+2");
  printf("1+2=%d\n", (int) duk_get_int(ctx, -1));
  duk_destroy_heap(ctx);
  return 0;
}
```

Why I abandoned it

- ES6 promises imply an event loop
- If your runtime is not Node or browser compatible you need to adapt every library
- Lack of modern ECMAScript features in Duktape
- The stack API is not my favorite
- A complete rewrite was necessary

Time went by…

QuickJS

- **Fabrice Bellard** releases QuickJS in July 2019
- Small and embeddable JavaScript engine
- ES2020 compliant
- Small standard library with C wrappers
- “Traditional” C API like CPython

QuickJS

```c
static int fib(int n) {
    if (n <= 0)
        return 0;
    else if (n == 1)
        return 1;
    else
        return fib(n - 1) + fib(n - 2);
}

static JSValue js_fib(JSContext *ctx, JSValueConst this_val,
                      int argc, JSValueConst *argv) {
    int n, res;
    if (JS_ToInt32(ctx, &n, argv[0]))
        return JS_EXCEPTION;
    res = fib(n);
    return JS_NewInt32(ctx, res);
}

static const JSCFunctionListEntry js_fib_funcs[] = {
    JS_CFUNC_DEF("fib", 1, js_fib ),
};

static int js_fib_init(JSContext *ctx, JSModuleDef *m) {
    return JS_SetModuleExportList(ctx, m, js_fib_funcs,
                                  countof(js_fib_funcs));
}

JSModuleDef *js_init_fib(JSContext *ctx,
                         const char *module_name) {
    JSModuleDef *m;
    m = JS_NewCModule(ctx, module_name, js_fib_init);
    if (!m)
        return NULL;
    JS_AddModuleExportList(ctx, m, js_fib_funcs, countof(js_fib_funcs));
    return m;
}
```

QuickJS + libuv = ❤️

- QuickJS comes with a small event loop
  for timers and signals
- Simple API to integrate your own loop
- I started by replacing platform dependent functionality
  with libuv
- I got hooked intro writing a runtime again

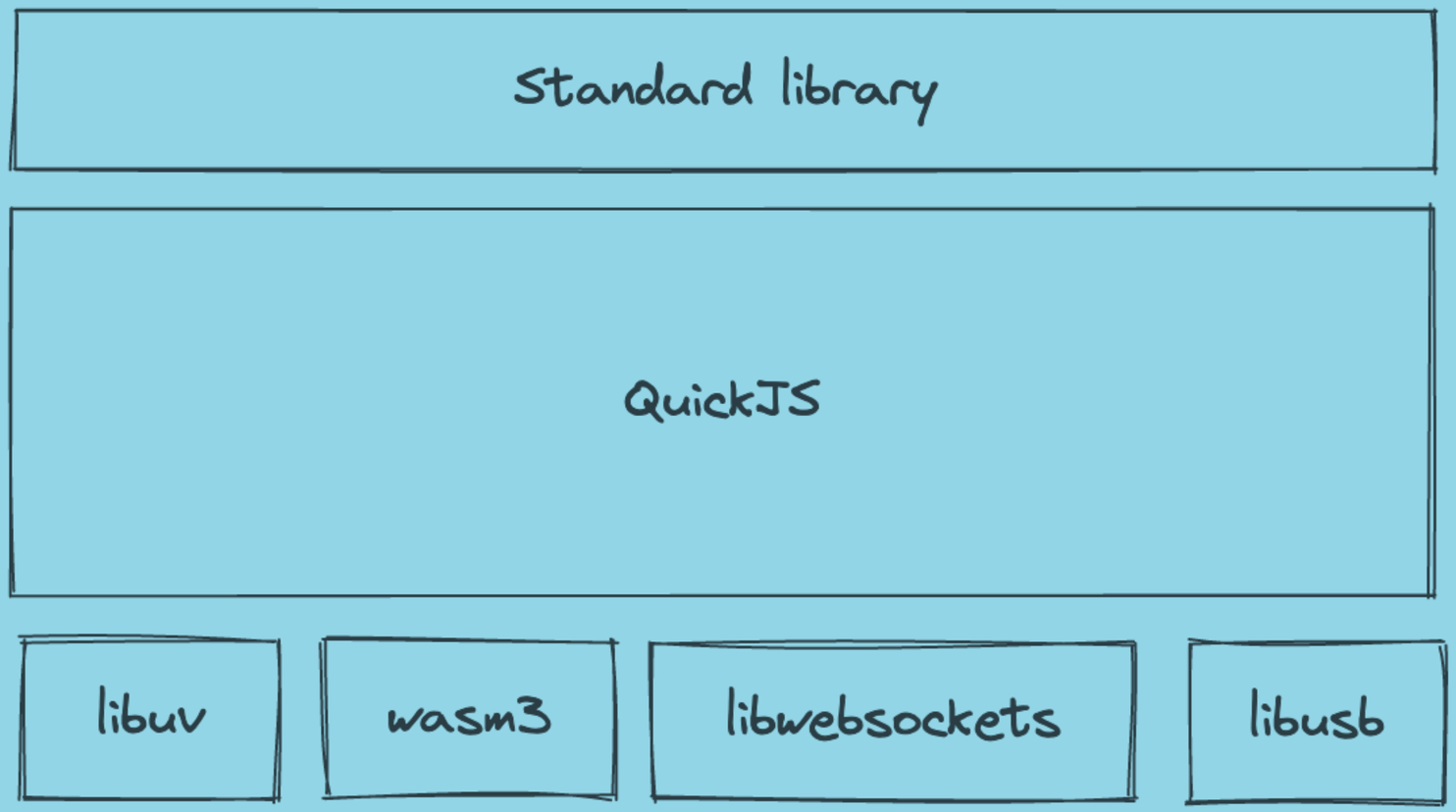
txiki.js — Let’s do it again!

- Target browser APIs
- Single global module for custom APIs
- Polyfill what QuickJS does not (yet) implement
- No shared libraries
- No CommonJS
- No package management (for now)
- **https://github.com/saghul/txiki.js**

txiki.js

- **libuv** as the platform layer
  - Socket and filesystem operations
- XHR implemented using **curl**, fetch polyfill
- WebAssembly implemented with **wasm3**

txiki.js — Looking ahead



My learnings

- Whatever engine you choose, ES2020 is a good baseline
- Targeting browser APIs is A Good Idea, thanks Deno!
- Use other tools like *esbuild* to help with bundling
- *INCBIN* made it very simple to embed JS code in the C runtime
- QuickJS can also be used for standalone apps: njk

njk

- Simple Jinja2-like templating in a single binary (< 1MB)
- Designed for generating config files in a container for example
- Uses QuickJS to embed Nunjucks and render templates from env variables 🤯
- **https://github.com/saghul/njk**

Thanks!