RUNNING JERRYSCRIPT PROJECTS IN THE BROWSER

Martijn Thé
martijn.the@intel.com

Zidong Jiang
zidong.jiang@intel.com

Why run JerryScript projects in the browser?

- Use-case: let people without real device try out your JS runtime in a simulated device
- Browser-based simulation means
  - really low barrier to try it out, just visit the website and play around
  - ubiquitous “platform” – run the simulation on every kind of host device with a browser
  - option to embed in desktop app using something like Electron

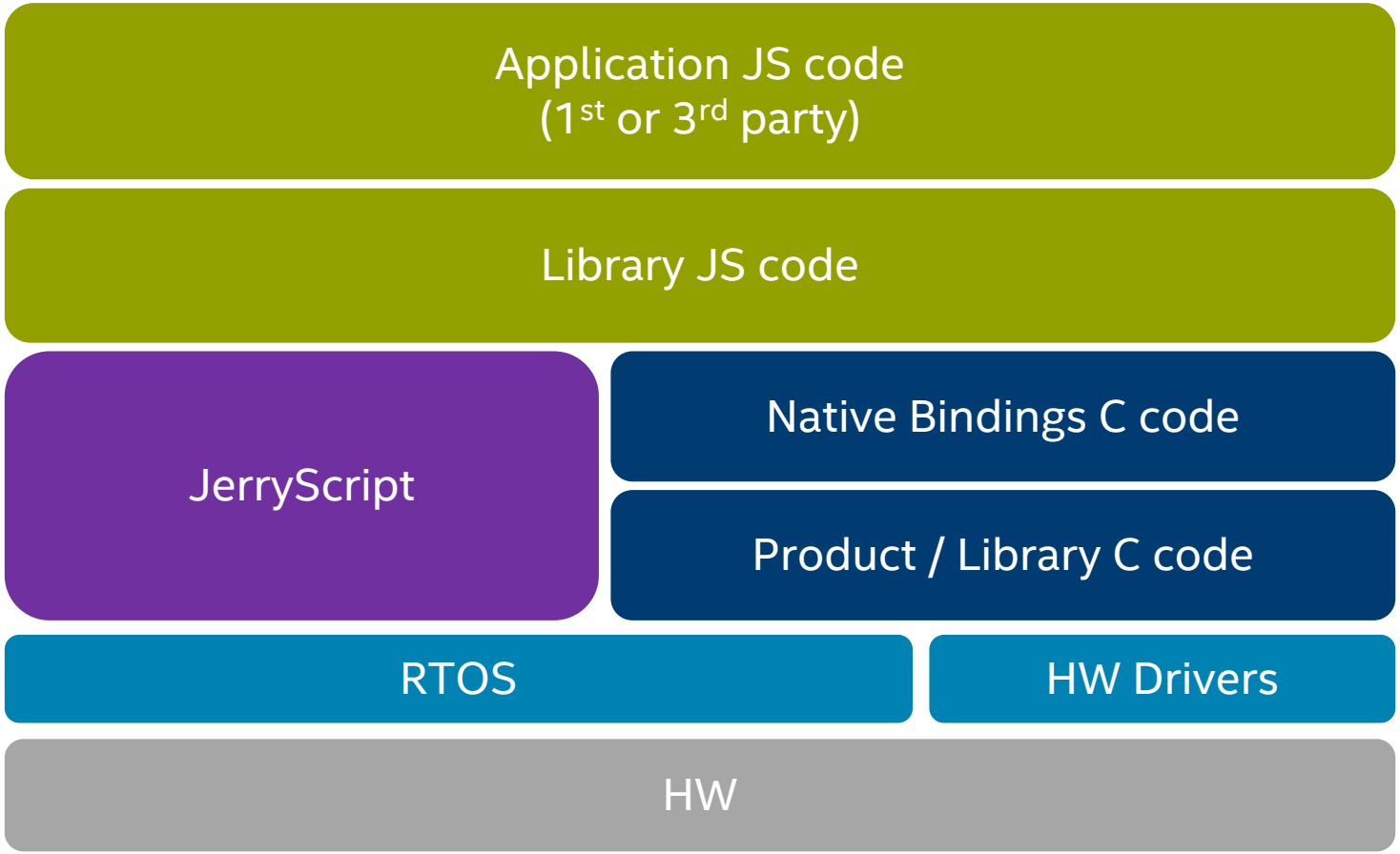
But how...?

- Emscripten: “transpiles” C/C++ code into JS/webasm that NodeJS / Browsers / etc. can run
  - Emscripten also comes with a C standard library, some POSIX APIs, simulated file system, etc.
  - Generated JS code emulates a 32-bit machine.
  - Pointers / memory access by address is implemented as offsets into a huge JS typed array.
  - Wasm / asm.js output, browsers are optimizing for running it quickly.
  - https://kripken.github.io/emscripten-site/index.html

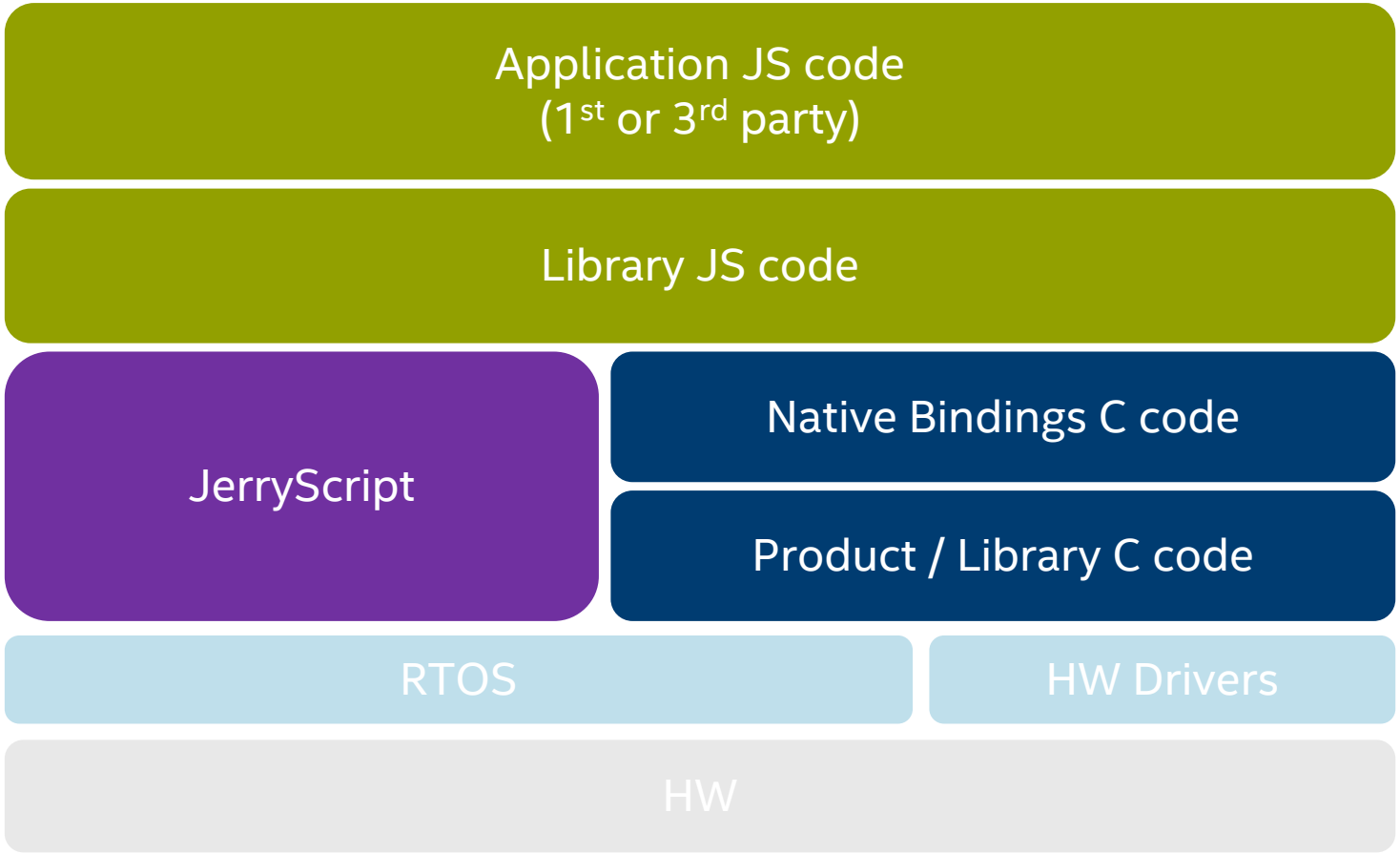
- `$ emcc main.c -o simulator`
  - `and out comes simulator.js !`

- **Enables re-using a lot of your project’s C code...**

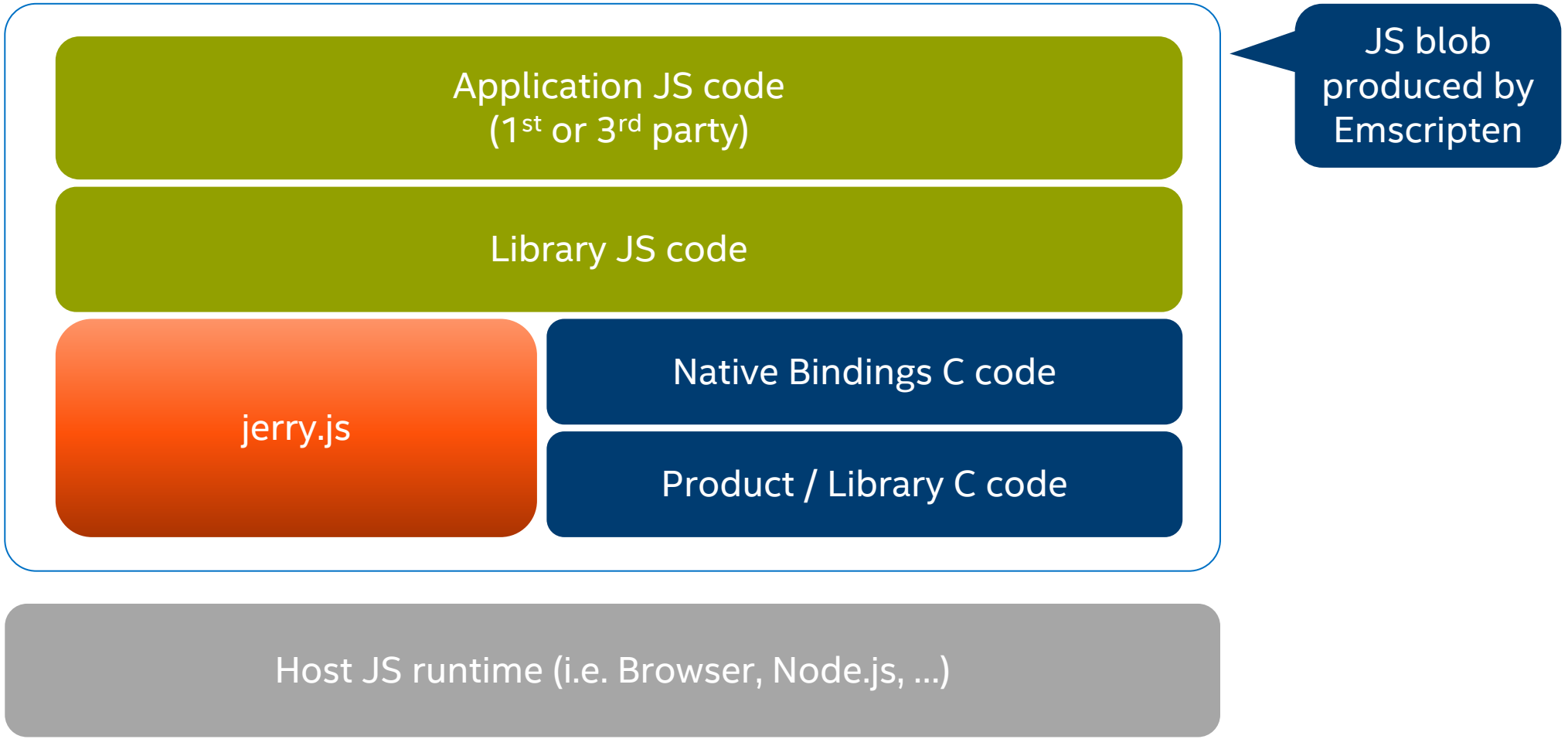
But how...?


Application JS code
(1st or 3rd party)
Library JS code
JerryScript
Native Bindings C code
Product / Library C code
RTOS
HW Drivers
HW

Making the cut. JerryScript in or out?



Reimplement jerryscript.h APIs in JS itself



jerryscript.h API in JS, some examples

```
jerry_value_t jerry_create_object (void)
  {
    return (jerry_value_t) EM_ASM_INT_V (
      {
        return __jerry.ref (new Object ());
      }
    );
  } /* jerry_create_object */
```

```
uint32_t jerry_get_array_length (const
    jerry_value_t value)
    {
      if (!jerry_value_is_array (value))
      {
        return 0;
      }
      return (uint32_t) EM_ASM_INT (
        {
          return __jerry.get ($0).length;
        }
        , value);
    } /* jerry_get_array_length */
```

- __jerry.ref() maps JS value from the host JS VM to jerry_value_t
- __jerry.get() maps jerry_value_t to host JS value
- jerry_value_t: monotonically increasing numeric ”IDs”, unlike real JrS jerry_value_t

Why reimplement jerryscript.h?

- Transpiling original JerryScript C sources leads to gigantic .js output from Emscripten.
  - You'd end up running a JS engine inside a JS engine... Imagine debugging that.
- Ability to use existing JS tooling, such as debuggers (i.e. the browser's debugger) to step through app.js code, etc.
- Mix-and match your runtime's APIs (output from Emscripten) with Node.js and all the APIs it brings along.

jerry.js downsides

- Heap footprint of app.js code not accurately simulated
- Runtime APIs from the host JS runtime available to app.js
  - If needed, most of the times it's possible to just delete the interfaces or override them.
- Runtime ECMAScript version probably newer than ES5.1, cannot be "downgraded" easily
- With JS alone: not possible to implement free_cb for `jerry_set_object_native_pointer()`
  - For Node.js, we use the "weak" native module: weak reference + hook when object is GC'd to correctly implement the free_cb behavior for `jerry_set_object_native_pointer()`
  - For Browser: values that are mapped to jerry_value_t (i.e. go through a jerryscript.h API) stay in the map forever (i.e. they appear to leak) – not a big deal for our simulation purposes.

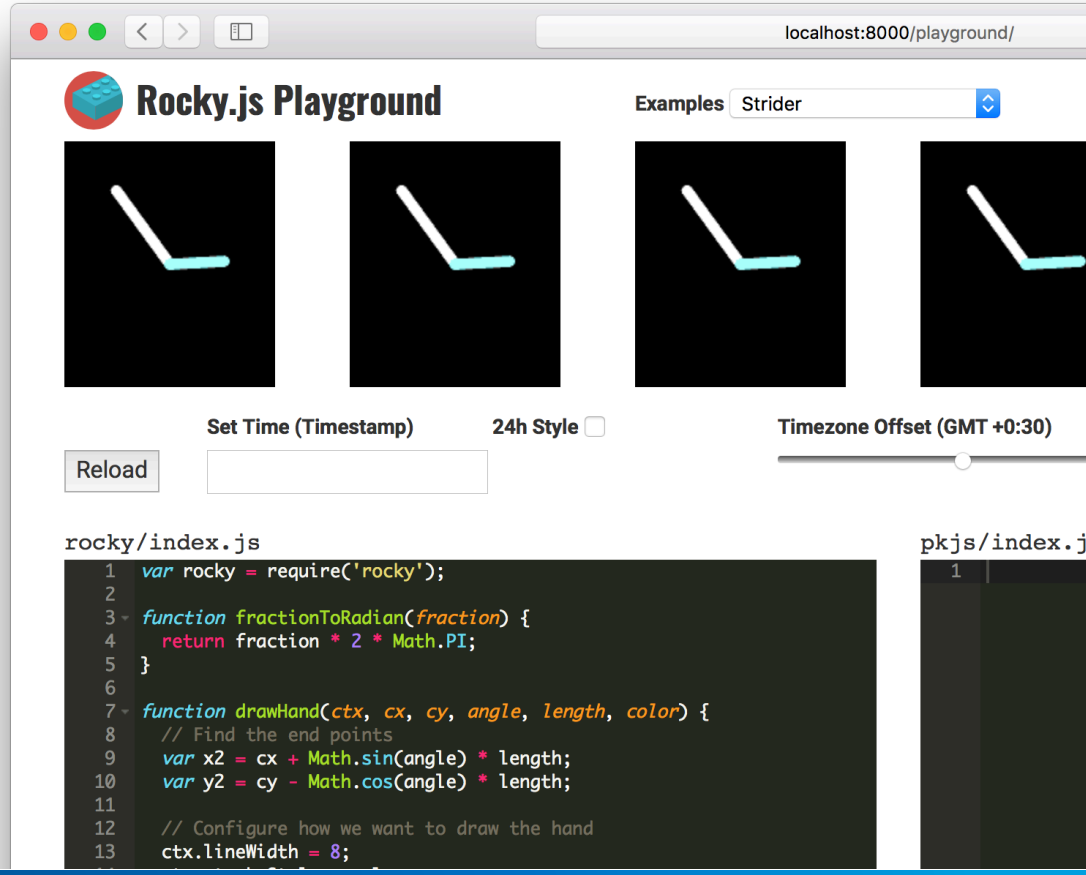
Demo

- Running (most of) the JerryScript unit tests:
- $ source ~/emsdk/emsdk_env.sh
- $ ./tools/run-tests.py --unittests --buildoptions=--emscripten-simulated-jerry-api=ON

- Building “emx-demo” example target:
- $ ./tools/build.py --emscripten-simulated-jerry-api=ON

History of the work: Pebble

- Wanted to create a browser-based “playground” for Pebble’s Rocky.js runtime
- Original JerryScript PR #1432 – Nov 2016
  - Martijn Thé & Marc Jessome, then at Pebble
  - Incomplete, just the APIs Pebble needed
  - JerryScript unit tests not building nor passing



PR #1901 – Status

- Almost done.
- Add Emscripten dependencies to Travis CI .yml
- Code review
  - Any volunteers?
- Deferring:
  - Promise & C API
    - Can't just use host's Promise because need control over execution / "job queue".
    - Doable, but yet another reasonable chunk of work.

- Not doing:
  - Anything snapshot related

Q&A