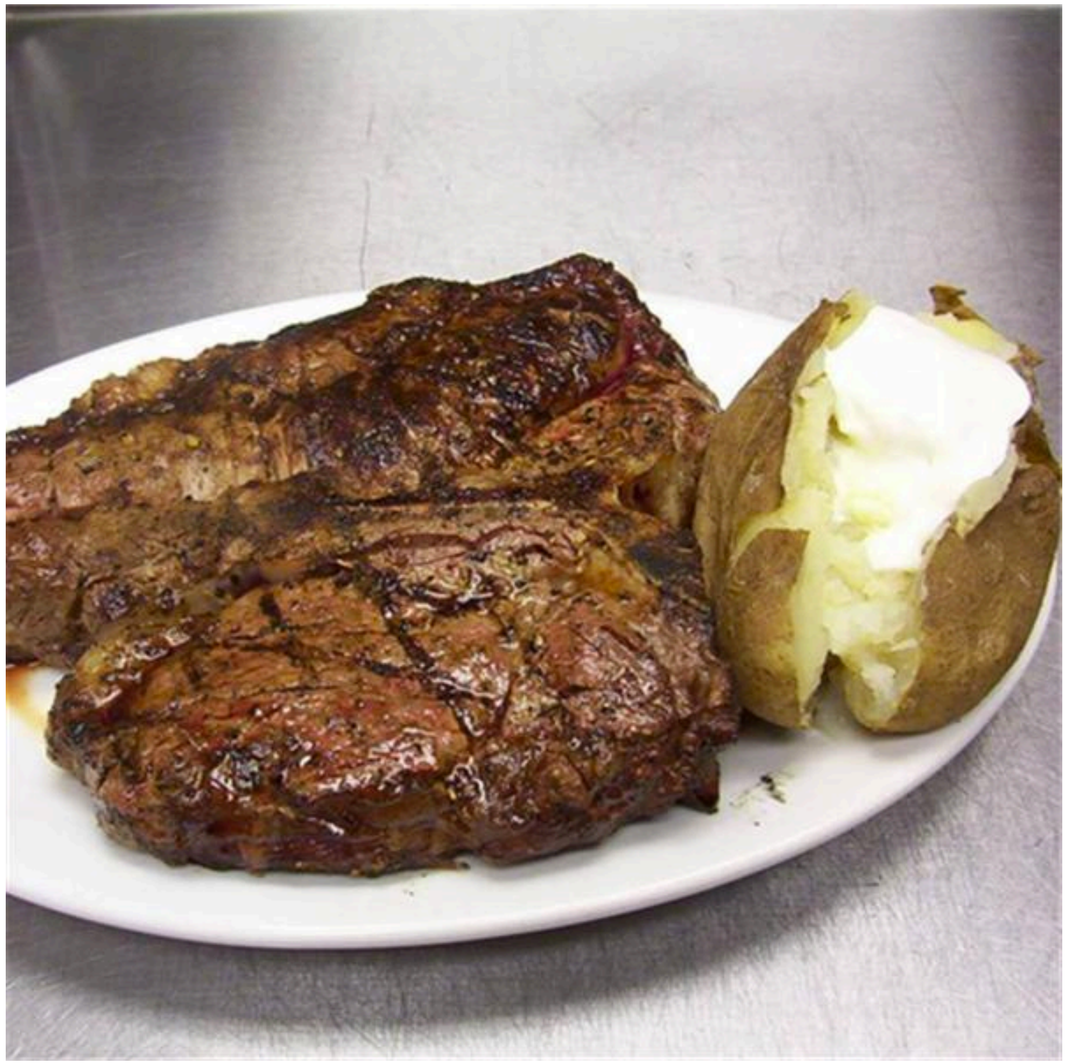
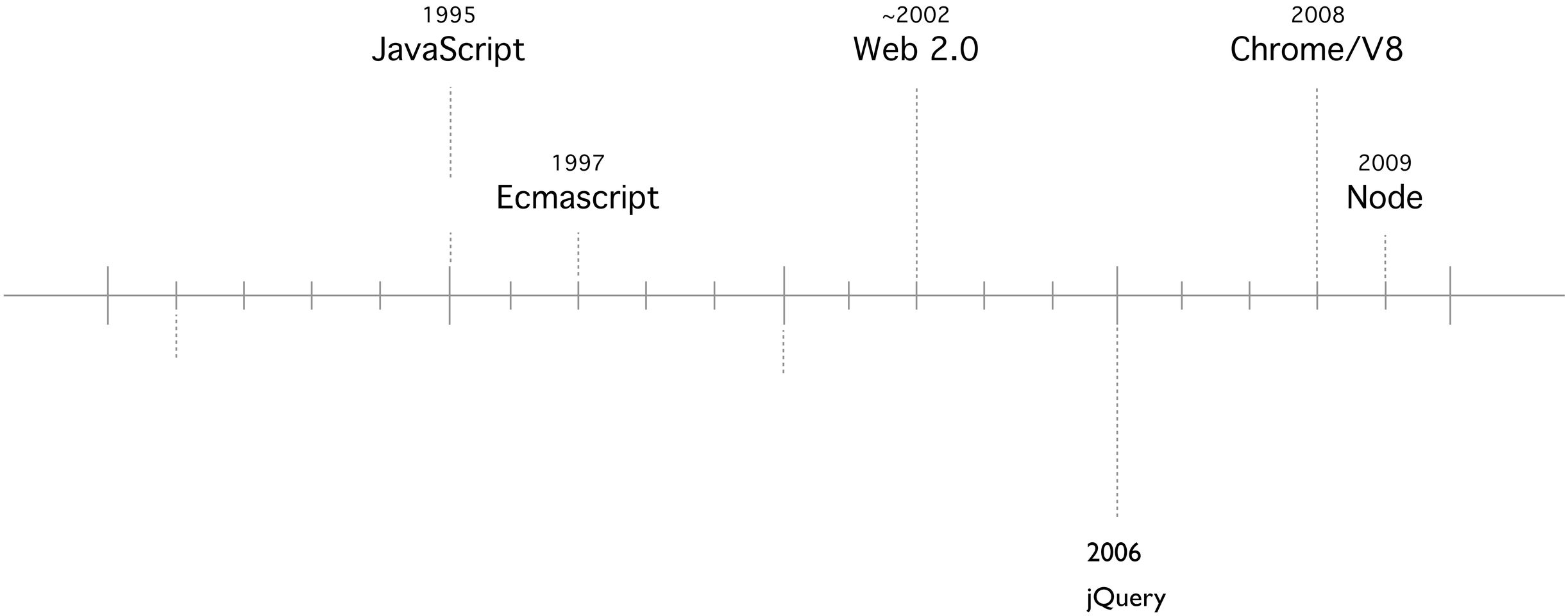
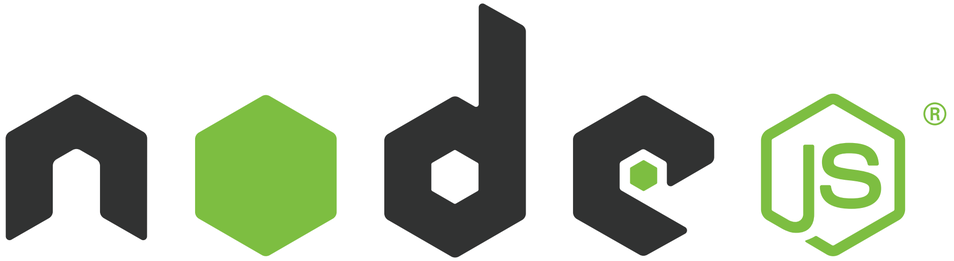



Simple HTTP Server

A Brief History of Everything



Intro to



or: Kernals, Processes, Threads —Oh My!
or: Fun with Modules
or: The Kitchen Sink & Async
or: Six Degrees from Ryan Dahl

Outline

- What is Node?
- Program vs Process
- History / context
- Node as process
- Node modules
- Node asynchronicity

What is Node?

“Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world.”

Node.js: the Essentials

- Node is a program (written in C, C++, and JS)
- It can run JavaScript via the Chrome V8 engine (C++)
- It provides APIs for your OS's file / network system (slow I/O)
- In other words, Node lets you run JS on a computer *outside of a browser* and interact with stuff on that computer
- Why?

The Practical Dev
@ThePracticalDev

Follow

So NASA uses Node.js in their space-suits. The logical follow up question: Do NASA space-suits depend on left-pad?

Benjamin Coe @BenjaminCoe 6/3/16

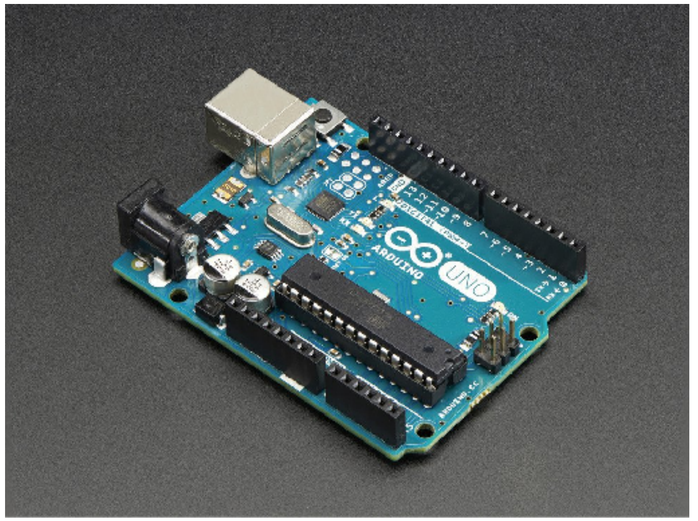
@CollinEstes I heard a rumor that NASA uses Node.js for space-suits. I'm curious, do you use the npm ecosystem to develop these apps?

57 69

Collin Estes @CollinEstes 6/3/16

@BenjaminCoe You heard correctly, and yes we do.

36 74



Program vs. Process

- **Program is data**
  - machine code (pre-compiled)
  - bytecode
  - text file (can be interpreted)
- **Inert — not doing anything**
- **Ready to be run as a process**

- **Process is execution**
  - memory allocated
  - CPU performing steps
- **"Live"**
- **Produces results**
- **Interactive**
- **Can be started/stopped**
- **Multiple processes from one program…**

Node as Process

(Demo)

Different Environments, Same Engine

fs

process

net

window

history

document

Browser Tool Examples

- **Synchronous stuff:**
  - console
  - window
  - document
- **Asynchronous stuff:**
  - setTimeout(callback, delay) (note: not in ECMAScript, "not JS" — an API!)
  - setInterval(callback, delay) — ditto above
  - new XMLHttpRequest() / .open, etc. — making HTTP requests
  - element.onclick(callback) // slightly different example

Node Tool Examples

- **Synchronous stuff:**
  - console (same name, but a wrapper for process.stdout)
  - global (no window!)
  - module (we'll get to this)
- **Asynchronous stuff:**
  - setTimeout(callback, delay) — same name for convenience's sake
  - setInterval(callback, delay) — ditto
  - http.request(options, callback) — uses built-in `http` JS module
  - etc.

Modularity

What is the purpose of a modular system?

- **Organized and maintainable vehicle for code splitting**
- **Why separate code?**
  - Single files have more defined responsibility
  - Easier collaboration
  - Visible structure
  - Testing
  - Maintainability
  - Reusability

Built-in Modules

- **There are MANY, here are a few:**
  - url
  - path
  - fs
  - http
  - crypto
  - net
  - child_process

How do we get more?

- Author-defined
- Third-party

Asynchronous / Non-blocking I/O

Big Use Case: Server

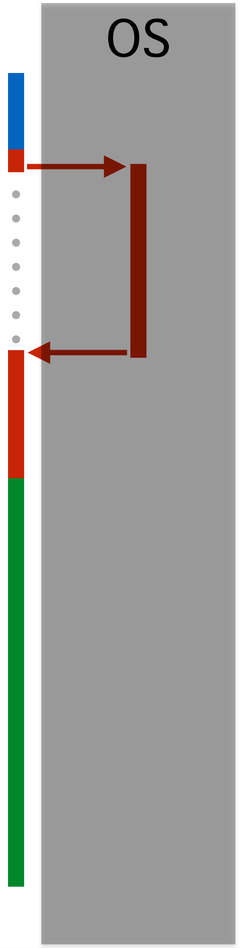
- A program running on a computer
- Listens on a *port* for *requests*
- Sends *responses* back
- Communication must follow agreed-upon *protocol* (e.g. HTTP)
- [DEMO]
- Problem: communication over network is super-slow! Should be able to field other requests / do other stuff in meantime.

Slow *blocking* I/O is handed off to a thread

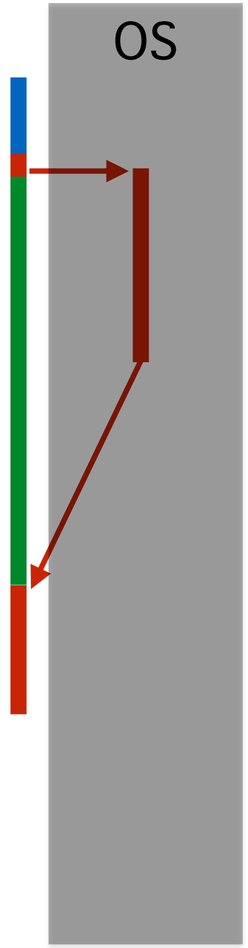
- **CPU clock cycles per access level (*Dahl 2010):**
  - Dynamic memory: "non-blocking"
    - L1 cache: 3
    - L2 cache: 14
    - RAM: 250
  - "Blocking" I/O:
    - Disk: 41,000,000
    - Network: 240,000,000

Concurrency

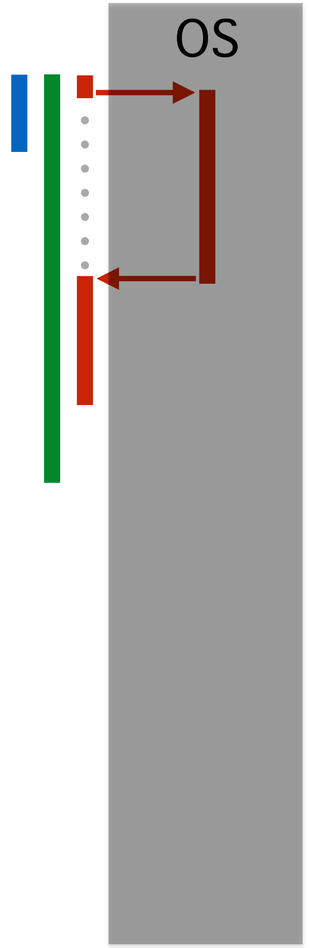
blocking



non-blocking



parallel (blocking)



"Threads are Hard"

- Every thread comes with performance overhead — memory, CPU, etc.
- Switching threads within a process consumes clock cycles (have to schedule jumps)
- Switching threads between processes consumes clock cycles AND waits for memory (have to load process state)
- Communicating between threads in the same process is very tricky — synchronizing results, being efficient
- Thread programming best left to experts

Server Strategies

- Apache: every new connection gets its own thread (wasteful of resources, doesn't scale well, lots of work to synchronize)

- Nginx: an event loop which queues *callbacks* back into the main execution stack, once thread managing blocking code finishes

"Browsers got it right — abstract that to callbacks with the DOM API."

RYAN DAHL, CREATOR OF NODE

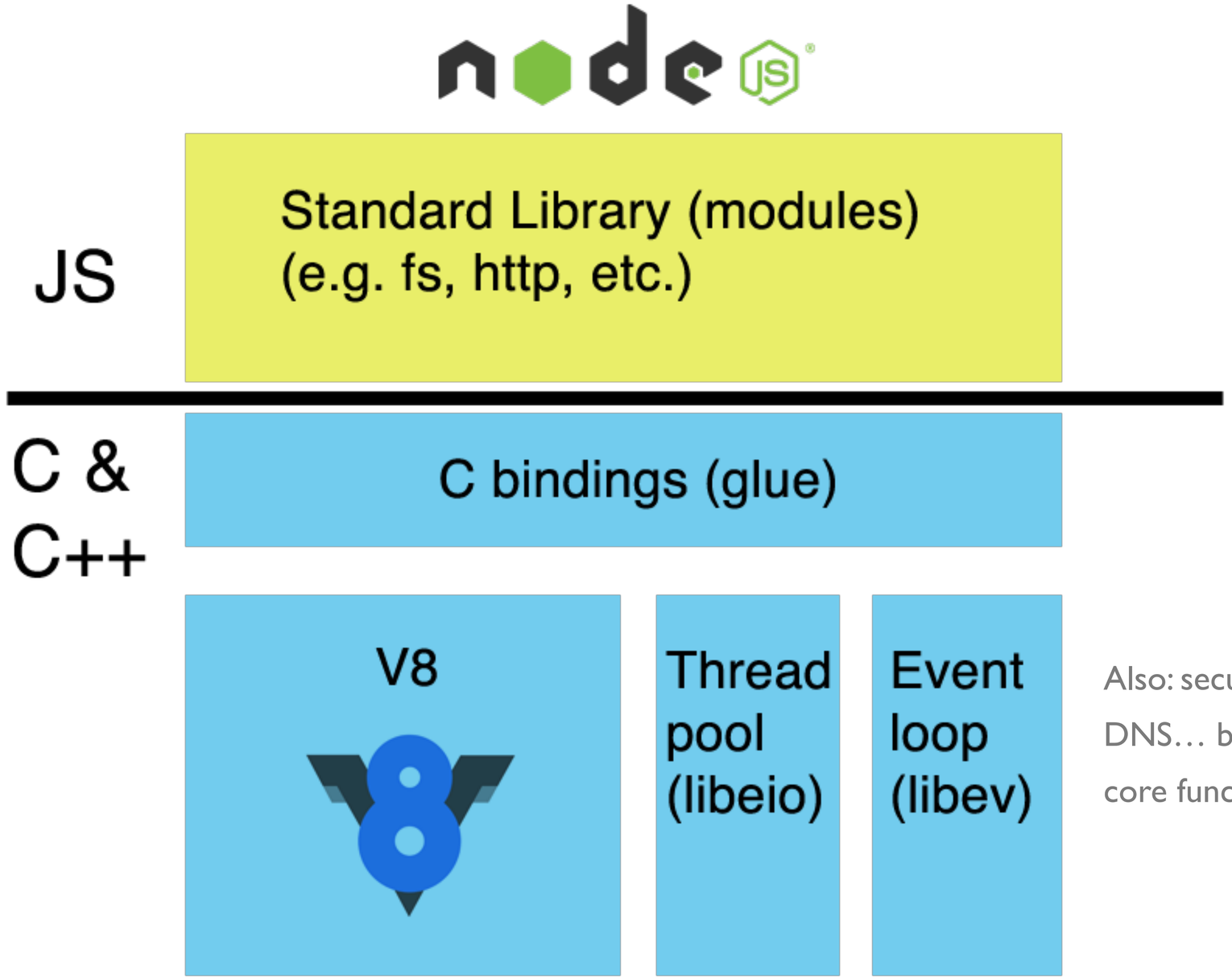
Er, not exactly

"Node.js is a ~~single-threaded~~, event-driven, non-blocking I/O platform"

– SOME PEOPLE ON THE INTERNET

"JavaScript is single-threaded" …arguably yes

– OTHER PEOPLE ON THE INTERNET

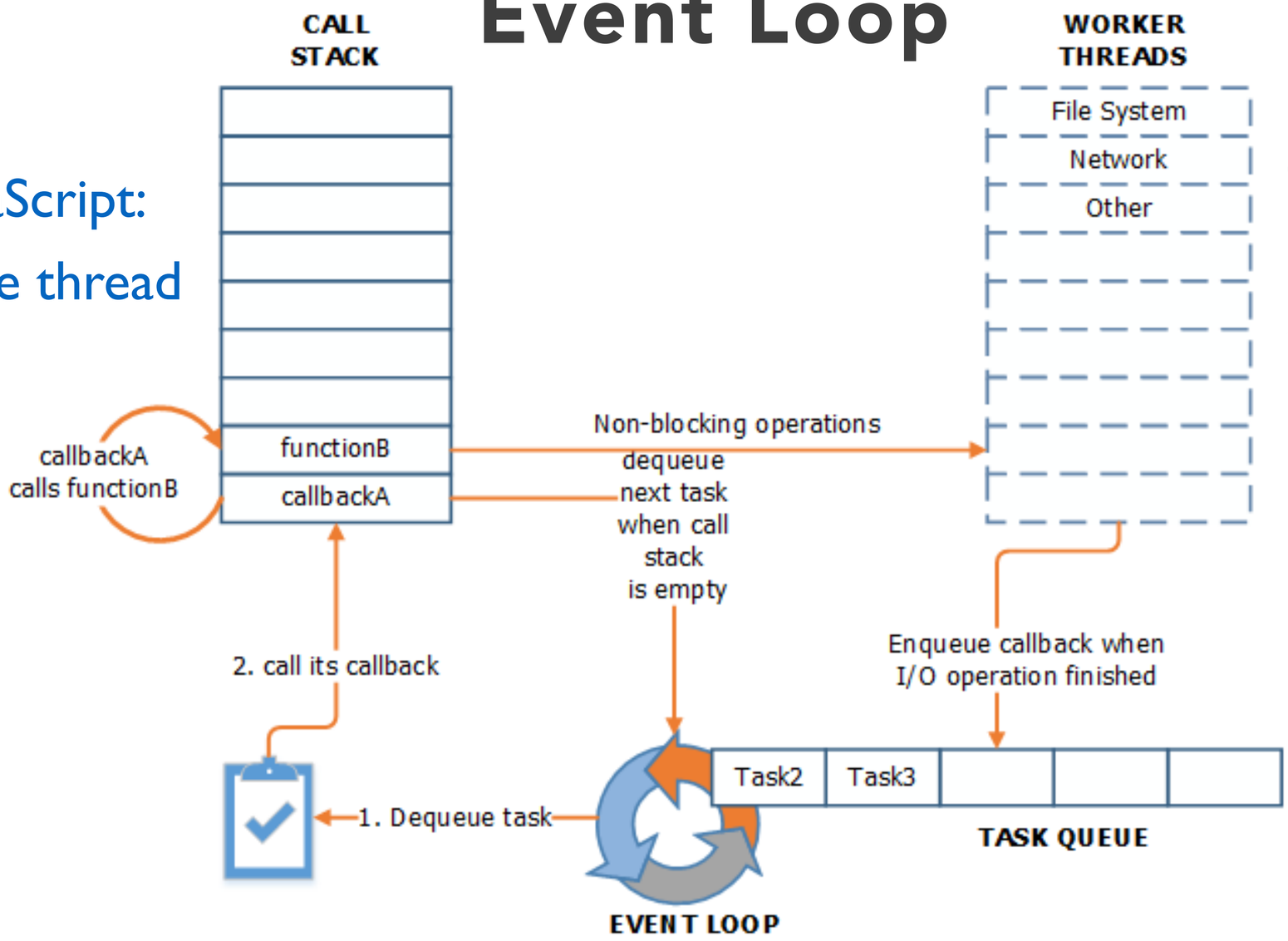


Also: secure crypto, DNS… but this is the core functionality

Event Loop

JavaScript:

One thread



Thread pool (libeio):

Slow stuff, multiple threads

Event loop (libev):

One thread