Brian McDaniel

Outline

- Background
  - Threads vs. events
  - Blocking vs. non-blocking I/O
- Node.js
  - What is it?
  - Why JavaScript?
  - API
  - Architecture
  - Ecosystem

Node.js

- "a purely **evented**, **non-blocking** [I/O] **infrastructure** to script highly **concurrent** programs" – Ryan Dahl

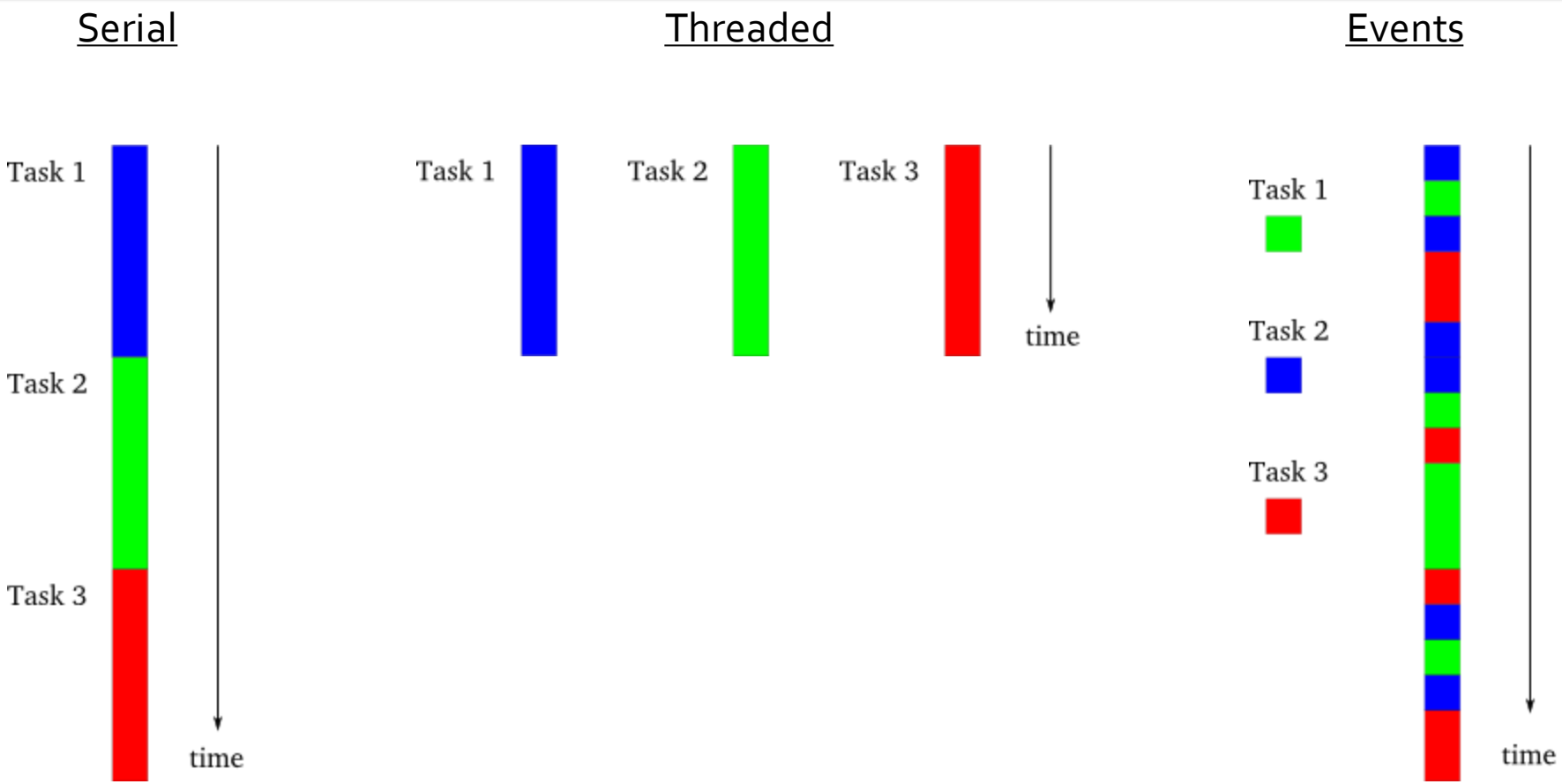
Threads vs. Events

Event Driven Programming

- Based on an *event loop* and *events.*
- Used extensively in GUI programming.
- Used extensively in the browser.
- Single threaded, 1 event at a time.

Golden rule: never block; short events.

Event Driven Programming



From Dave Peticolas (http://krondo.com/blog/?p=1209)

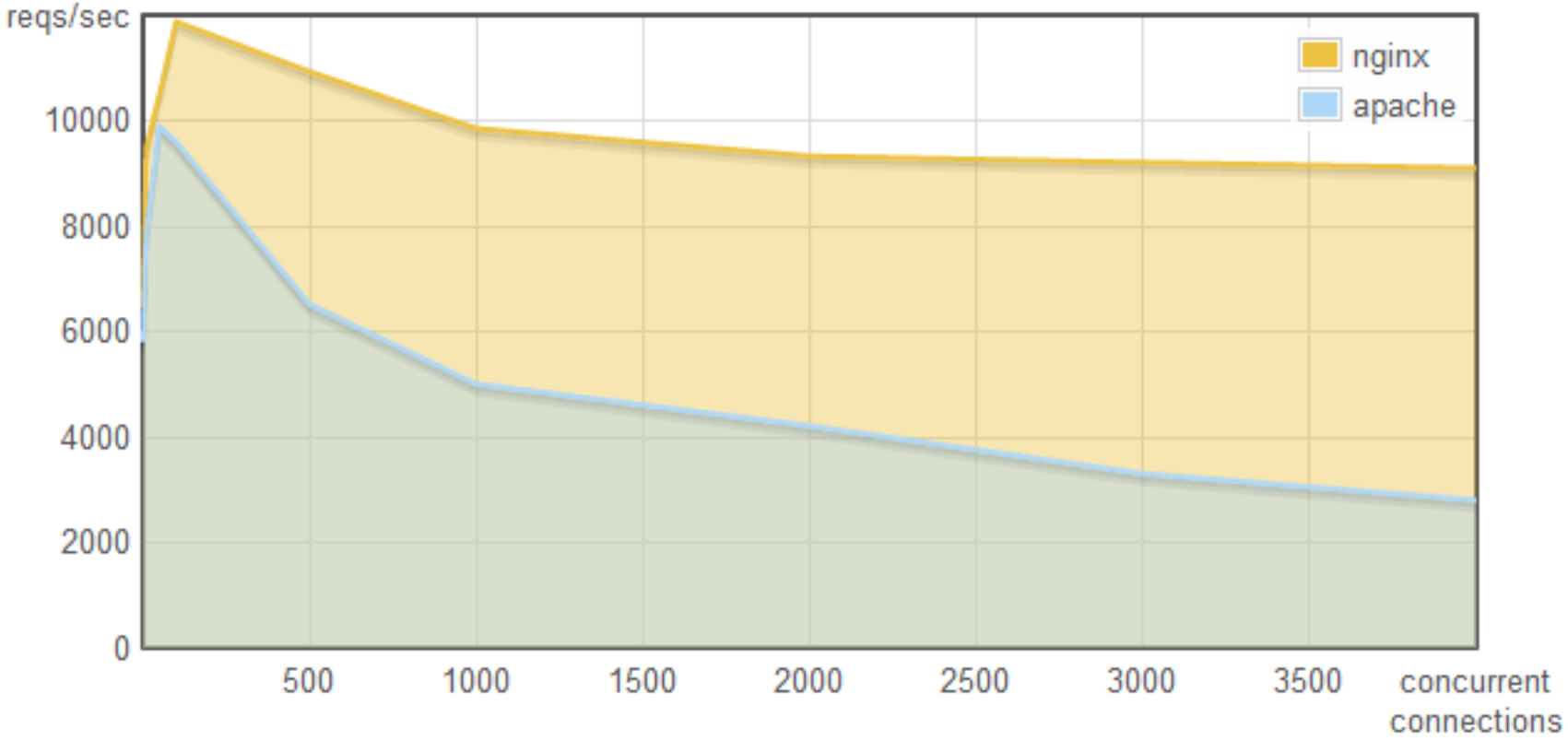
Threads vs. Events

- Threads
  - Cons
    - Use more resources
    - Require locking and resource protection
    - Error prone; non-determinism
    - See: The Problem with Threads by Edward Lee, Berkeley (2006)
  - Pros
    - Synchronous control flow within a thread
    - "Standard" way of achieving concurrency
    - Maps easily to multiple cores

Threads vs. Events

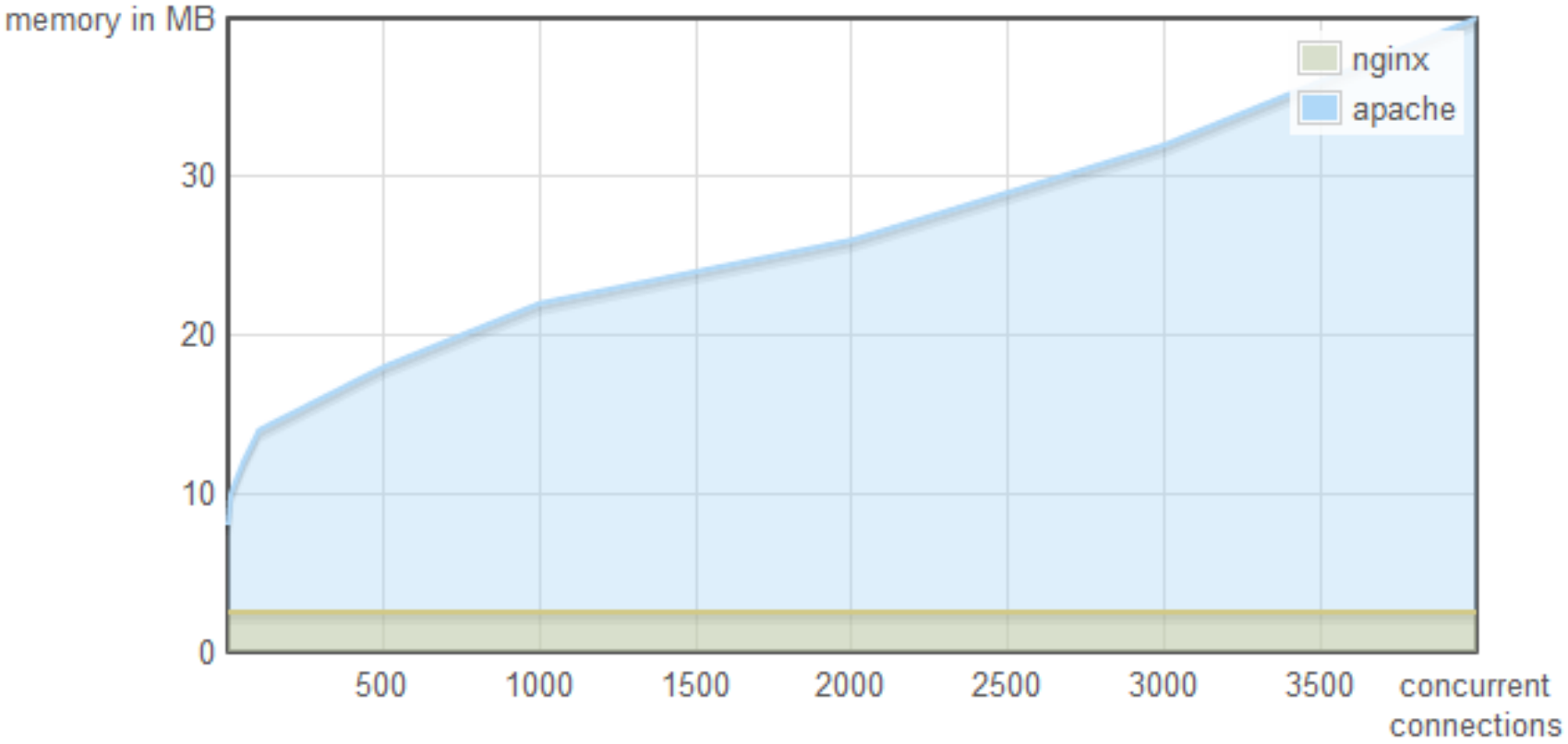
- Events
  - Cons
    - Control flow is not as straightforward
    - Hard to implement (especially in low-level languages)
    - See: Why Events Are a Bad Idea (for high-concurrency servers) by Behren et al, Berkeley (2003)
  - Pros
    - Resource efficient
    - Typically single threaded, no parallelism
    - See: Event-Driven Programming for Robust Software by Dabek et al, MIT (2002)

Apache vs. nginx



http://blog.webfaction.com/a-little-holiday-present

Apache vs. nginx



http://blog.webfaction.com/a-little-holiday-present

Blocking vs. Non-blocking I/O

Blocking vs. Non-blocking I/O

Blocking

```
console.log('Reading file 1');
var x = FS.readFileSync('file1.txt');
console.log('File 1 done.');

console.log('Reading file 2');
var y = FS.readFileSync('file2.txt');
console.log('File 2 done.');
```

```
Reading file 1
File 1 done.
Reading file 2
File 2 done.
```

Non-Blocking

```
console.log('Reading file 1.');
FS.readFile('file1.txt', function (err, data) {
    if (!err) {
        console.log('File 1 done');
    }
});

console.log('Reading file 2.');
FS.readFile('file2.txt', function (err, data) {
    if (!err) {
        console.log('File 2 done');
    }
});
```

```
Reading file 1.
Reading file 2.
File 2 done
File 1 done
```

Event Driven, Non-Blocking IO

- Goal: separation of CPU tasks and IO tasks.
- Never wait on an IO task inside of a CPU task.
- Encapsulate CPU tasks inside of events.
- Execute event listeners (fire an event) when data is ready to be processed by CPU.

Event Considerations

```
var x = 0;
console.log('Reading file 1.');
FS.readFile('file1.txt', function (err, data) {
    if (!err) {
        console.log('File 1 done');
    }
    while (true) {
        x++;
    }
});

console.log('Reading file 2.');
FS.readFile('file2.txt', function (err, data) {
    if (!err) {
        console.log('File 2 done');
    }
    while (true) {
        x++;
    }
});
```

Other NIO Frameworks

- Netty
  - Java
- EventMachine
  - Ruby
- Twisted
  - Python

Outline

- Background
  - Threads vs. events
  - Blocking vs. non-blocking I/O
- **Node.js**
  - What is it?
  - Why JavaScript?
  - Architecture
  - API
  - Ecosystem

Node.js

- "a purely **evented**, **non-blocking** [I/O] **infrastructure** to script highly **concurrent** programs" – Ryan Dahl

In Practical Terms

- Node.js is…
  - a JavaScript interpreter with:
    - A module system
    - I/O and helper libraries, exposed as modules
  - 1 binary file, statically linked (~8.5MB)

Usage: node <script.js> <args>

Why JavaScript?

- Already designed around events
  - BOM and DOM already have events and timers.
  - Closures make callbacks easy.
- No pre-existing I/O libraries, "untainted".
- Google V8
  - Compiles to machine code.
  - Designed for speed.

Node.js Strengths

- Large numbers of concurrent connections
  - Think WebSockets, Comet, long-polling
- Good at acting as an aggregator of backend services
- Rapid development
- Full web stack in JavaScript

Node.js Weaknesses

- Not good for CPU intensive tasks.
- New, constantly changing.
- No killer web framework built on top.
- Not battle tested on a top website.

API Overview

- CommonJS module system
  - client.js

```
var mod = require('module_name');
mod.foo();
var y = mod.SomeValue
```

  - module_name.js

```
exports.foo = function () {
    console.log('foo called');
};
var x = 3; // This is local to this module
exports.SomeVariable = 5;
```

Some Built-In Node.js Modules

- fs – File system
- net – TCP & UNIX domain sockets
- dgram – UDP sockets.
- dns – DNS tools (resolving)
- http & https – HTTP clients and servers
- tls – secure sockets and servers
- child_process – spawn, like popen()

Common Abstractions: EventEmitter

- Examples:
  - server.on('connection', function (socket) {….});
  - socket.on('data', function (data) {…});
  - Similar to events in the browser:
    - element.addEventListener('click', function (event) {…});
- EventEmitter
  - addListener(event, listener);
  - removeListener(listener);
  - on(event, listener);
  - once(event, listener);
  - emit(event, arg1, arg2, …);

Demo: HTTP server

Dealing with Binary Data

- Buffer class
  - Represents raw memory allocated outside of V8.
  - Specified in bytes; length is immutable
  - String encodings:
    - ascii
    - utf-8
    - base64
    - others..

```
str = "node.js";
buf = new Buffer(str.length);

for (var i = 0; i < str.length ; i++) {
    buf[i] = str.charCodeAt(i);
}

console.log(buf);
```

```
brianmcd@thinklinux:~/node-presentation$ node buffer.js
<Buffer 6e 6f 64 65 2e 6a 73>
```

Architecture

- libev
  - Provides the event loop and events.
  - Provides file descriptor watchers for sockets and pipes.
- libeio
  - Provides asynchronous wrappers for file operations and blocking libraries.
  - Uses a thread pool to execute blocking operations.

Architecture

- V8
  - Provides the JavaScript implementation.
- Node
  - Provides module system, underlying I/O operations, and the JavaScript library.
  - Glues everything together.

Ecosystem

- npm – node package manager
  - Currently hosting 1655 packages.
  - Easy: npm install packagename@version
- Popular packages
  - Connect and Express
  - socket.io
  - JSDOM
  - database wrappers

Current Node.js Users

- Voxer
  - Real-time iPhone communication app
- Plurk
  - Switched from Netty to Node.js
    - 10x less memory usage
    - Slightly more CPU usage
- Yahoo!
  - Unspecified use for Yahoo! Mail

My Research

- Goal: create a server-centric web framework where the DOM is rendered on the server, and synced with the client.
- Benefits: persistence, collaboration, easier development
- Currently using Node for:
  - A custom HTTP server (http module + Connect)
  - Persistent connections (socket.io)
  - Rendering the DOM (JSDOM)

Chat Server Example

Common Abstractions: Streams

- Readable
  - event: 'data'
  - setEncoding(), pause(), resume(), pipe()
- Writeable
  - write(), end()
- Examples
  - Sockets, HTTP request object, stdin/stdout
  - FS.createReadStream()/createWriteStream()