CSC 443: Web Programming

Haidar Harmanani
Department of Computer Science and Mathematics
Lebanese American University
Byblos, 1401 2010 Lebanon

Introduction

- NodeJS is a complete software platform for scalable server-side and networking applications
  - open-source under the MIT license
  - comes bundled with a JavaScript interpreter
  - runs on Linux, Windows, Mac OS & most other major operating systems

LAU Computer Science

2009

Created by Ryan Dahl
Version 1 in 2009 to revolutionize web applications
Inspired by Ruby Mongrel web server

2010

Joyent sponsors Node.js development

2011

First released version of Node.js available to the public
Initial version only available for Linux.

Microsoft partners with Joyent to provide Windows support

2012

Complete rewrite of central libraries

2014

Latest release v0.10.26
Still several improvements away from a stable v0.12
and a finalized v1.0

...

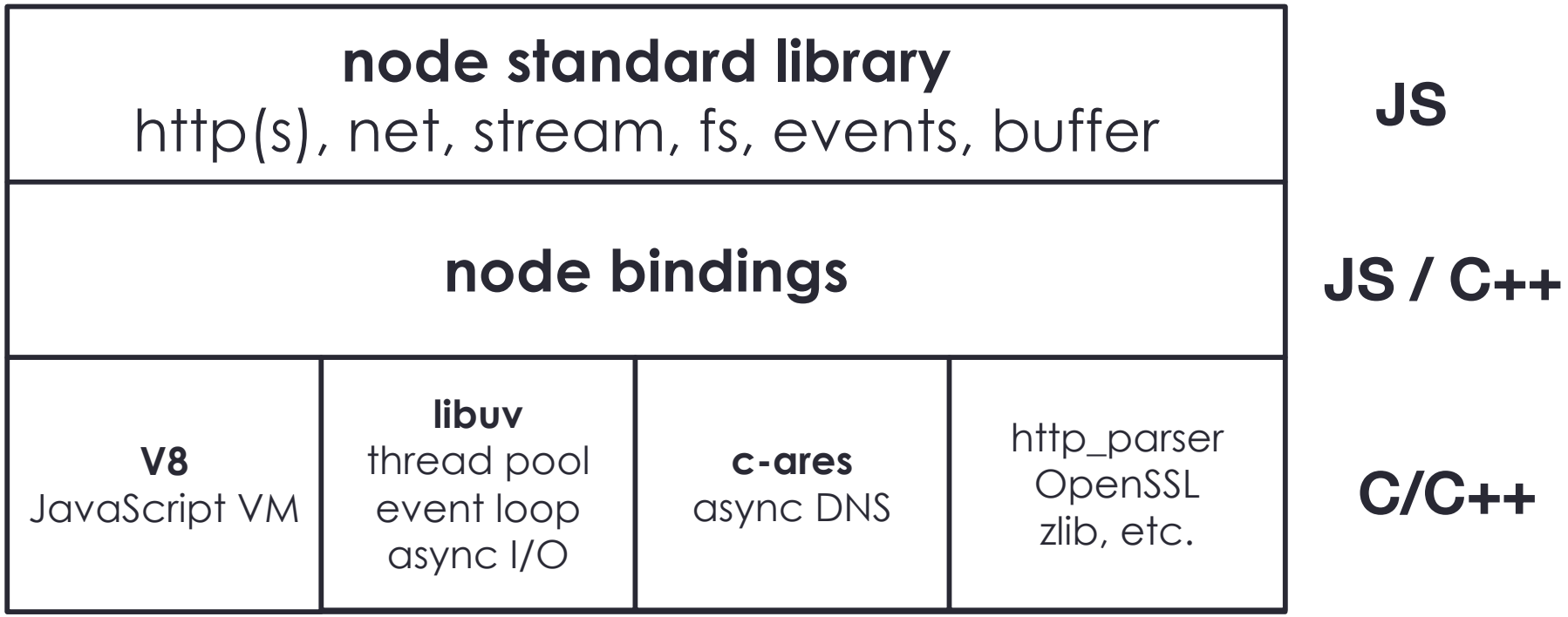
Current stable release v9.1.0

LAU
Computer Science

How does Node.js Work?

- Uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices
- Built on Chrome's V8 JavaScript runtime for easily building fast, scalable network applications
- Two major components:
  - Main core, written in C and C++
  - Modules, such as Libuv library and V8 runtime engine, also written in C++

What is Node.JS made of?



LAU Computer Science

What is so good about Node.js?

- An event driven programming model
- Highly scalable
- Uses asynchronous, event-driven I/O (input/output), rather than threads or separate processes
- Ideal for web applications that are frequently accessed but computationally simple

LAU Computer Science

Recall: How does Apache Work?

- A traditional web server such as Apache creates a thread each time a web resource is requested
  - Typically quick response time
  - Cleans up after the request
- The approach may tie up a lot of resources, especially when dealing with popular web applications
  - May have serious performance issues.

Major Features

- Single threaded
- Event Loop
- Non-blocking I/O

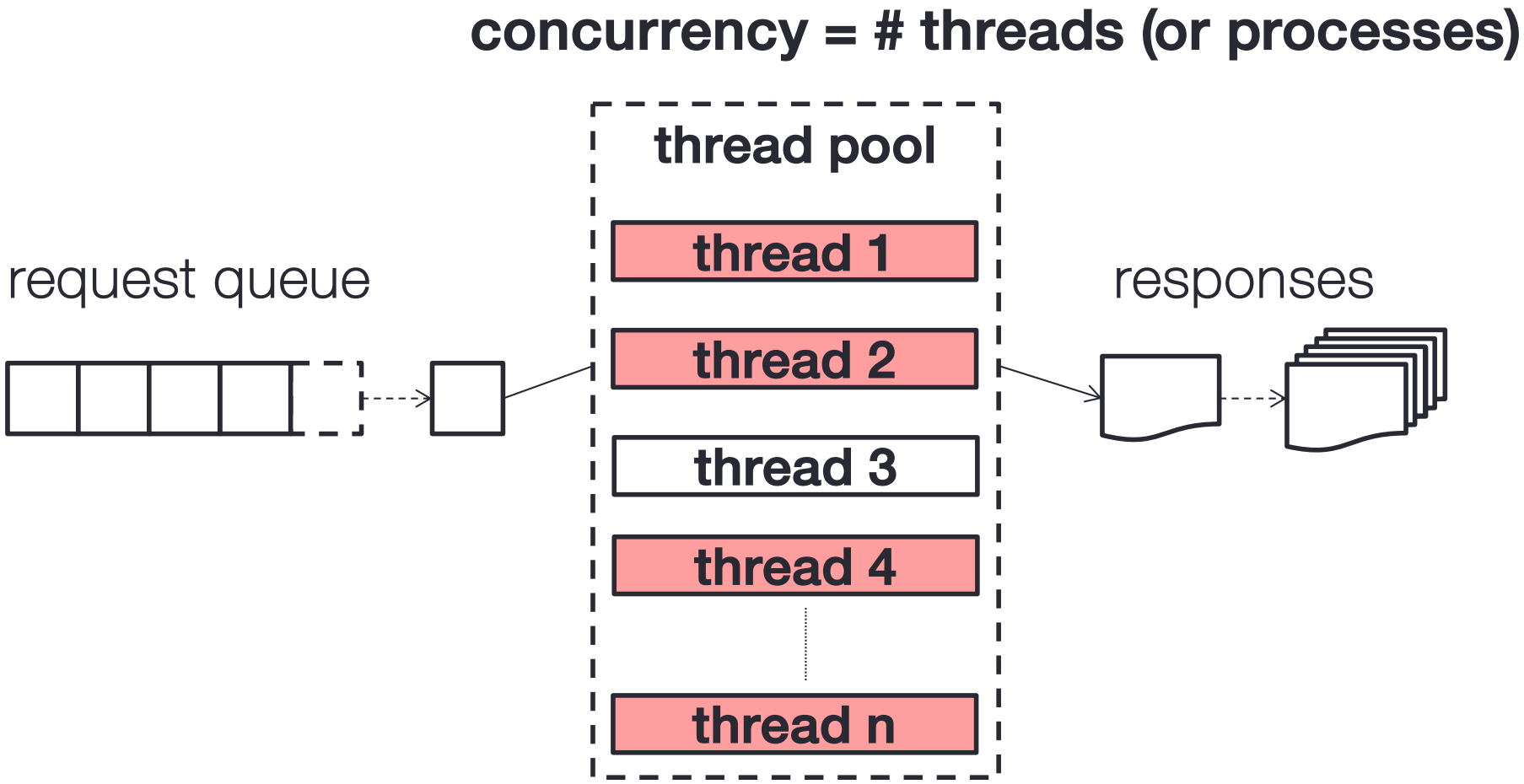
LAU Computer Science

Asynchronous Approach for Web Servers

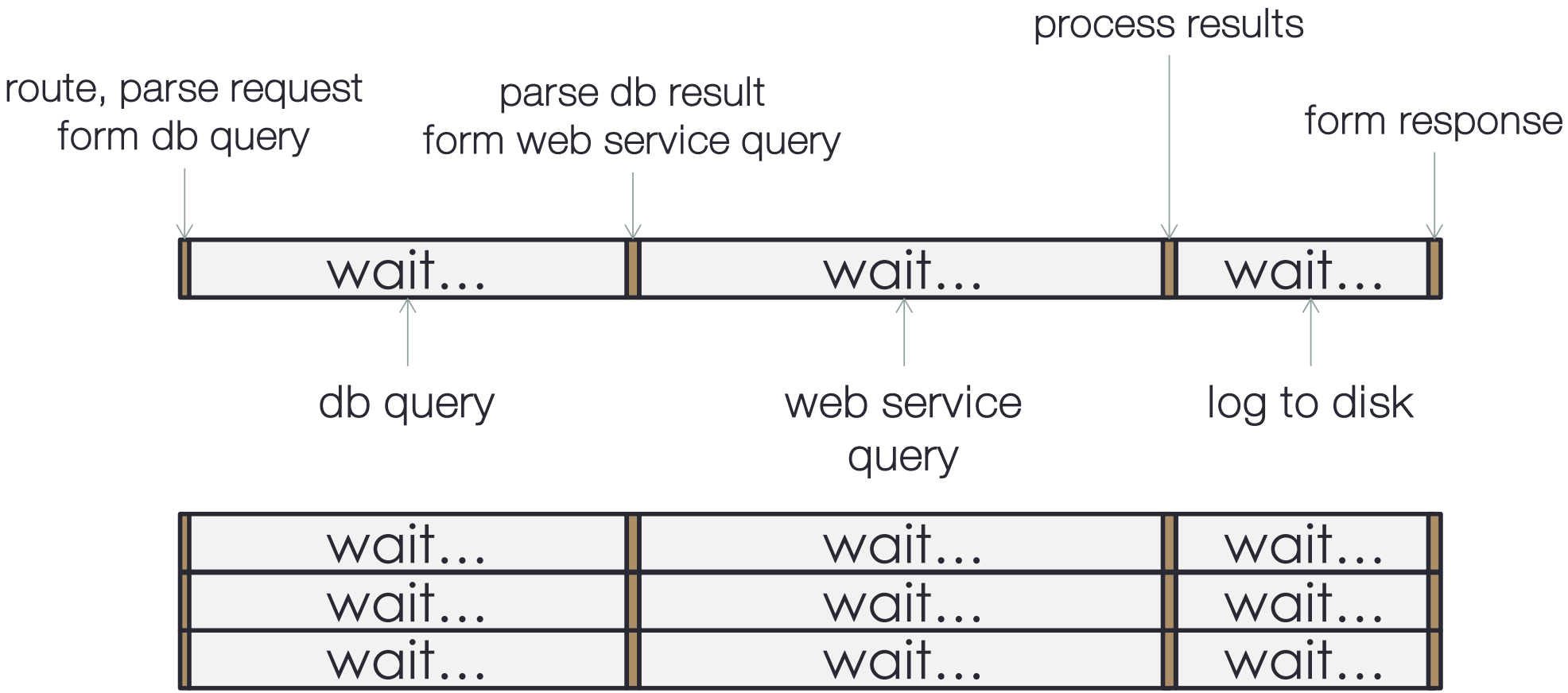
- Node doesn't create a new thread or process for every request
  - Server listens for specific events
  - When the event happens, server responds accordingly.
- Node doesn't block any other request while waiting for the event to complete
- Events are handled—first come, first served—in a relatively uncomplicated event loop

LAU Computer Science

Thread Pool Model

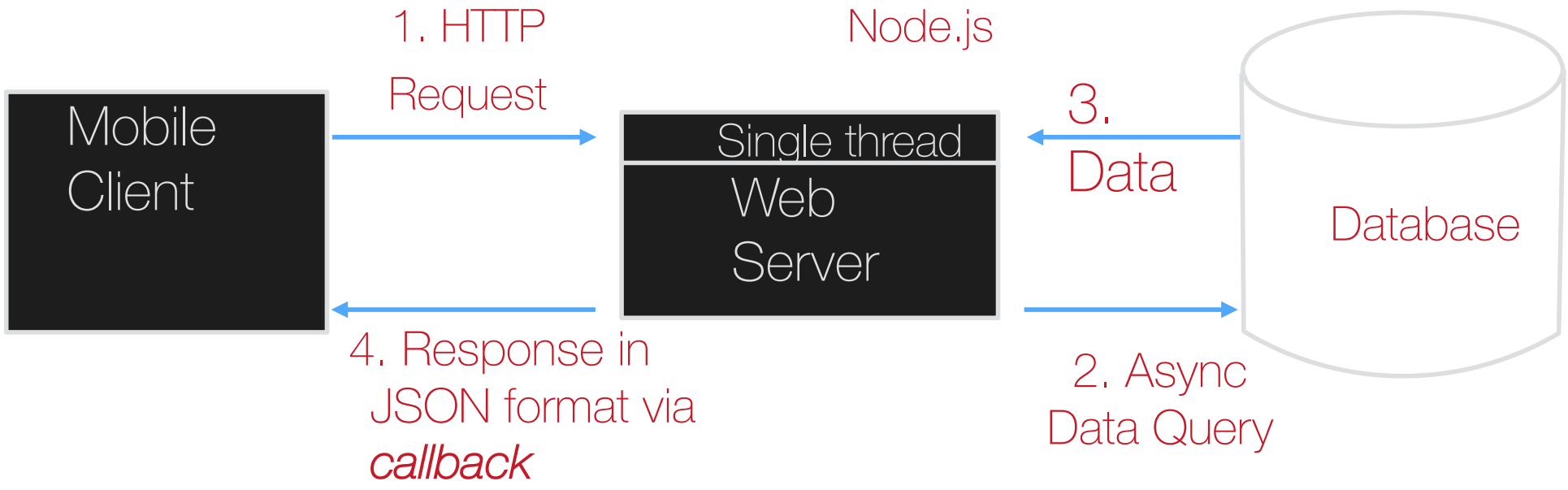


Efficiency of a Worker Thread



LAU
Computer Science

A simple example: accessing data from a database



* Here Node.js, acknowledges the request right away before writing any data to the database.

LAU
Computer Science

How to start?

- Node applications are created with JavaScript
  - The same as the client-side applications with one major difference: developers have to set up a development environment
- Grab a package installer for your favorite OS
  - The brave hearts can grab the source and compile it!
- Node is available for all major OSes
  - Linux
  - Windows (WebMatrix)
  - Mac OS

helloworld.js in Node.js

```
// load http module
var http = require('http');

// create http server
http.createServer(function (req, res) {
    // content header
    res.writeHead(200, {'content-type': 'text/plain'});

   // write message and signal communication is complete
   res.end("Hello, World!\n");
}).listen(8124);

console.log('Server running on 8124');
```

LAU Computer Science

`helloworld.js` in Node.js

- To run the application, from the command line in Linux, the Terminal window in Mac OS, or the Command window in Windows, type:

```
> node helloworld.js
```

- The following is printed to the screen:

```
>Server running at 8124
```

- Access the site using any browser
  - Type **`localhost:8124`** (assuming you are running the application locally) or the the URL of the remote site with the 8124 port (if it's running remotely).
  - A web page with the words "Hello, World!" is displayed

LAU Computer Science

The Anatomy of a Node Application…

- Most Node functionality is provided through external applications and libraries called modules
- Load the HTTP module and assign it to a local variable
  - `var http = require('http');`
- The `HTTP` module provides basic HTTP functionality, enabling network access of the application.

LAU Computer Science

The Anatomy of a Node Application…

- Create a new server with **`createServer`**, and an anonymous function is passed as the parameter to the function call.
  - **`http.createServer(function (req, res) { ...`**
- The `requestListener` function has two parameters:
  - A server request (http.ServerRequest)
  - A server response (http.ServerResponse).
- Within the anonymous function, we have the following line:
- The line `res.writeHead(200, {'content-Type': 'text/plain'})` within the anonymous function uses the method writeHead to send a response header with the response status code (200), as well as provides the content-type of the response
  - The method `writeHead` belonsg to the `http.ServerResponse` object
- The second, optional parameter to `writeHead` is a `reasonPhrase`, which is a textual description of the status code.

LAU Computer Science

The Anatomy of a Node Application…

- The application next writes the “Hello, World!” message:
- `res.end("Hello, World!\n");`
- The `http.ServerResponse.end` method signals that the communication is finished
  - All headers and the response body have been sent.
  - The method must be used with every `http.ServerResponse` object
- The end method has two optional parameters:
  - A chunk of data, which can be either a string or a buffer.
  - If the chunk of data is a string, the second parameter specifies the encoding.
- The second parameter is required only if the encoding of the string is anything other than utf8, which is the default.
- An alternative approach would have been as follows:
  - `res.write("Hello, World!\n");`
- and then:
  - `res.end();`

LAU Computer Science

The Anatomy of a Node Application…

- The anonymous function and the createServer function are both finished on the next line in the code:
  - `}).listen(8124);`
- The `http.Server.listen` method chained at the end of the `createServer` method listens for incoming connections on a given port—in this case, port 8124.
- Optional parameters are a hostname and a callback function.
- If a hostname isn't provided, the server accepts connections to web addresses

LAU Computer Science

The Anatomy of a Node Application…

- The `listen` method is asynchronous
  - The application doesn't block program execution, waiting for the connection to be established
  - Whatever code following the `listen` call is processed, and the `listen` callback function is invoked when the listening event is fired—when the port connection is established.
- The last line of code uses the console object from the browser world in order to provide a way to output text to the command line (or development environment), rather than to the client
  - `console.log('Server running on 8124/');`

LAU Computer Science

Programming Models

- Processing in traditional programming blocking models cannot continue until an operation finishes
  - Derives from time sharing systems
  - Needed to isolate users from one another
  - Model does not scale especially with the emergence of networks and the Internet
- Multi-threading is an alternative to the above programming model.
  - A thread is a of lightweight process that shares memory with every other thread within the same process
    - Threads are created as an ad hoc extension in order to accommodate several concurrent threads of execution.
  - When one thread is waiting for an I/O operation, another thread can take over the CPU.
  - When the I/O operation finishes, that thread can wake up
  - Threads can be interrupted and resumed later
  - Some systems allow threads to execute in parallel in different CPU cores.

Asynchronous Functions and the Node Event Loop

- One of the fundamental design issues behind `Node` is that an application is executed on a single thread (or process), and all events are handled asynchronously.
- A typical web server, such as Apache has two different approaches to how it handles incoming requests
  - The first assigns each request to a separate process until the request is satisfied
  - The second spawns a separate thread for each request.
- The first approach is known as the *prefork multiprocessing* model while the second is known as *worker multiprocessing model*
- Both approaches respond to requests in parallel
  - If five people access a web application at the exact same time, and the server is set up accordingly, the web server handles all five requests simultaneously

LAU Computer Science

Prefork Multiprocessing Model

- The prefork multiprocessing model, or prefork MPM can create as many child processes as specified in an Apache configuration file.
- The advantage to creating a separate process is that applications accessed via the request, such as a PHP application, don’t have to be thread-safe.
- The disadvantage is that each process is memory intensive and doesn’t scale very well.

LAU Computer Science

Worker Multiprocessing Model

- The worker multiprocessing model implements a hybrid process-thread approach
  - Each incoming request is handled via a new thread.
- It's more efficient from a memory perspective
- Requires that all applications be thread-safe.
- Although PHP is thread-safe, there's no guarantee that the many different libraries used with it are also thread-safe.

LAU Computer Science

Event-Driven Programming

- A Node application is created on a single thread of execution.
  - It waits for an application to make a request
  - When Node gets a request, no other request can be processed until it's finished processing the code for the current one.
- It does not sound very efficient, does it?
  - It wouldn't be except for one thing: Node operates asynchronously, via an event loop and callback functions.
  - An event loop is nothing more than functionality that basically polls for specific events and invokes event handlers at the proper time.
  - In `Node`, a callback function is this event handler.

LAU Computer Science

The Node Event Loop…

- Let us explain this issue further…
- When a `Node` application makes some request of resources (such as a database request or file access)
  - Node initiates the request, but doesn't wait until the request receives a response.
  - It attaches instead a callback function to the request.
  - When whatever has been requested is ready (or finished), an event is emitted to that effect, triggering the associated callback function to do something with either the results of the requested

The Node Event Loop…

- If five people access a `Node` application at the exact same time, and the application needs to access a resource from a file, `Node` attaches a callback function to a response event for each request.
- As the resource becomes available for each, the callback function is called, and each person's request is satisfied in turn
- In the meantime, the `Node` application can be handling other requests, either for the same applications or a different application

LAU Computer Science

Simple Socket Server

```
var net = require('net')

var server =
net.createServer(function(socket) {
    socket.write('hello\n')
    socket.end()
})

server.listen(9898)
```

LAU Computer Science

Events –Listeners and Emitters

```
var server = net.createServer(function(socket) {

    socket.on('data', function(data) {
        console.log(data.toString())
    })

    socket.on('end', function() {
        console.log('client disconnected')
    })

}).listen(9898)
```

LAU Computer Science

Making HTTP Requests

```
var http = require('http')

var req = http.request({
    host: 'jssaturday.com',
    path: '/sofia'
}, function(res) {
    console.log(res.statusCode)
    res.on('data', function(data) {
        console.log(data.toString())
    })
})

req.end()
```

LAU Computer Science

Simple HTTP Forwarding Proxy

- How difficult would it be to write a local forwarding proxy?

Simple HTTP Forwarding Proxy

```
var http = require('http')

http.createServer(function(req, res) {
    req.pipe(http.request({
        host: req.headers.host,
        path: req.url,
        headers: req.headers
    }, function(xRes) {
        res.writeHead(xRes.statusCode,
xRes.headers)
        xRes.pipe(res)
    }))
}).listen(8080)
```

LAU Computer Science

Reading a File Asynchronously

```
// load http module
var http = require('http');
var fs = require('fs');

// create http server
http.createServer(function (req, res) {
   // open and read in helloworld.js
   fs.readFile('helloworld.js', 'utf8', function(err, data) {
      res.writeHead(200, {'Content-Type': 'text/plain'});
      if (err)
         res.write('Could not find or open file for reading\n');
      else
         // if no error, write JS file to client
         res.write(data);
      res.end();
     });
}).listen(8124, function() { console.log('bound to port 8124');});
```

LAU Computer Science

Quick Notes on the Previous Application

- A new module, File System (`fs`), that wraps standard POSIX file functionality, including opening up and accessing the contents from a file is used
  - The method used is `readFile`
- Callback functions are attached to the `readFile` and to the `listen` methods

LAU Computer Science

Challenges

- Debugging
  - why is my stack trace so short
  - exception handling
- Non-linear code
  - Nesting
  - Requires shift of programming paradigm
- Blocks on CPU
  - Beware of CPU intensive tasks
  - Run multiple nodes or child processes

Benefits

- Async I/O made easy
- Single-thread simplifies synchronization
- One language to rule them all
- Very active community
- Multi-platform

THE REAL WORLD

LAU
Computer Science

Modules

base64.js

```
var encoding = 'base64' // locals are private

exports.toBase64 = function(s) {
    return new Buffer(s).toString(encoding)
}
```

app.js

```
var b64 = require('./base64')
var a = b64.toBase64('JSSaturday')
```

LAU Computer Science

Node Package Management

- NPM
  - install and publish packages
  - upgrade, search, versioning
- npmjs.org
  - browse popular packages
  - publish your own

LAU Computer Science

Node.JS Resources

- nodejs.org
- which version to use?
  - Event X: stable (0.8.x, 0.10.x)
  - Odd X: unstable (0.9.x, 0.11.x)
- Documentation: nodejs.org/api
- Playing with the command line REPL
- Build from source: github.com/joyent/node

LAU Computer Science

ExpressJS: Web app Framework

- Node.JS is powerful
  - full control over HTTP server
  - but most of the time you'll use a web framework
- Web app frameworks likes ExpressJS provide
  - Request Routing
  - Body and Parameter Parsing
  - Session and Cookie Management
  - Templates
  - Static File Serving, Logger and many more

LAU Computer Science

ExpressJS – hit counter

```
var express = require('express')
var app = express();

app.use(express.cookieParser());
app.use(express.cookieSession({secret: "dG9wc2VjcmV0"}));

app.use(function(req, res) {
    var sess = req.session
    sess.hits = sess.hits || 0
    sess.hits++

    res.json({ visits: sess.hits })
});

app.listen(80)
```

LAU Computer Science