CSCC09F
Programming on the Web

Node.js

server-side frameworks, loading modules,
routes, static files

What is Node.js?

Ryan Dahl

- Node.js is a software package comprised of a JavaScript engine together with a set of asynchronous input/output (I/O) libraries
- Node's JavaScript engine is Google's V8, which also powers the Chrome browser
  - Compiles JS to native code (e.g. IA-32, x86-64, ARM), making for really fast execution
  - Runtime (dynamic) code optimization based on execution profile, e.g. inline caching of data access and function calls
  - Very efficient memory management system

Lars Bak

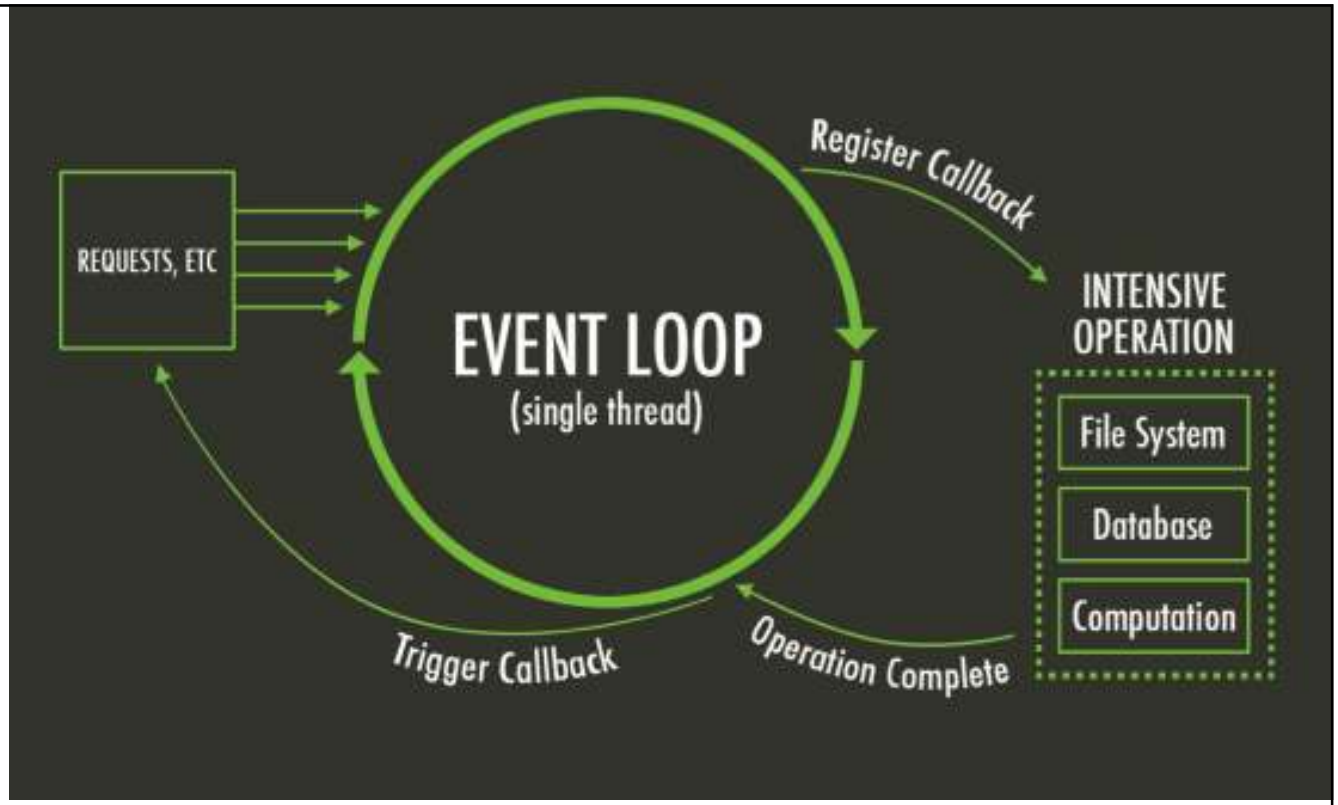
What is Node.js?

- An Nodeconf participant report of Ryan Dahl's presentation noted:
  - "His presentation was littered with the detritus of past failed attempts to come up with Web servers that used evented I/O to go fast. … it was inspirational and humbling to hear just how hard Ryan had to bang his head against the wall to come up with something as simple and refined as Node."
- What does Node.js do? enables you to run JavaScript from a command-line interface, outside of your browser. (imagine that!!!) … luckily, it does quite a bit more

What is Node.js?

- "Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js 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." (extract from the Node.js Web site)
  - high degree of concurrency without threads or processes – instead uses event-loop with stack
  - alleviates overhead of "context-switching" (as in CSCC69)
  - as in client-side JavaScript, use callbacks on I/O and other time-consuming tasks to avoid blocking on those requests

Node.js Event Loop



- I/O libraries enable Node.js to interface with files or network devices in an asynchronous manner, so Node.js can use a single-threaded event-queue to implement a fast, lightweight, event-based Web server.

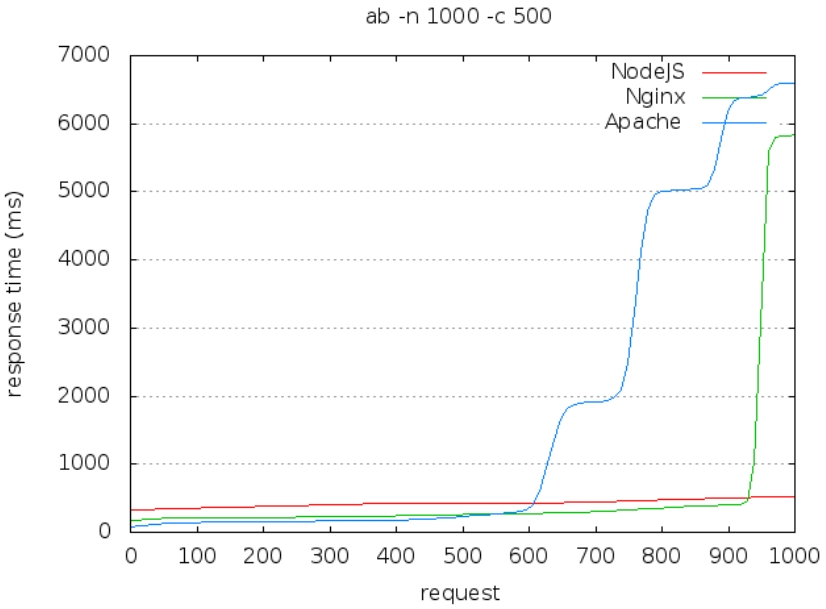
Relative Latency for I/O Types

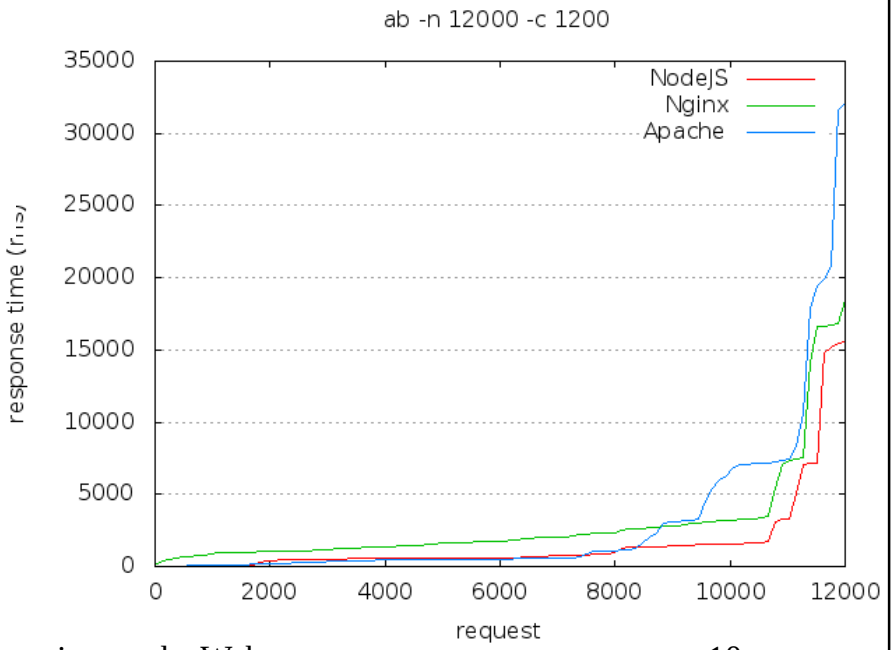
Type	*CPU cycles*	*Relative scale*
L1 cache	3	next room ~5m
L2 cache	14	across the street ~20m
RAM	250	next block ~400m
disk	40 000 000	earth circumference
network	370 000 000	distance to the Moon

Why Node.js?

- OK, so now you may have some idea of the "what", but may still wonder WHY we are using Node.js in C09
- Node.js has a lot of nice properties, including simplified handling of concurrent activities using asynchronous callbacks rather than explicit threads, and scalability to handle very large numbers of client requests
- Node.js also has detractors, who point out there are languages other than JS better suited for expressing complex business logic and implementing computation-intensive tasks
- For us it is a chance to continue to hone JS programming skills (the de facto Web programming language) and to practice a new kind of programming: "event-driven"

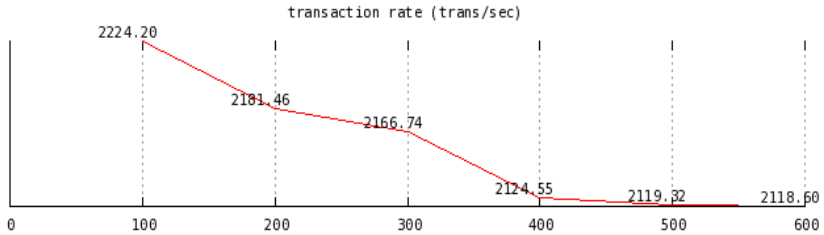
Node.js Performance

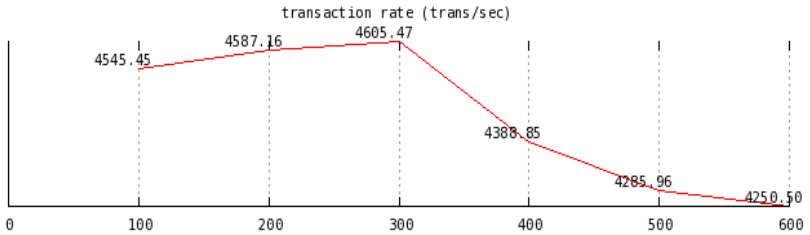


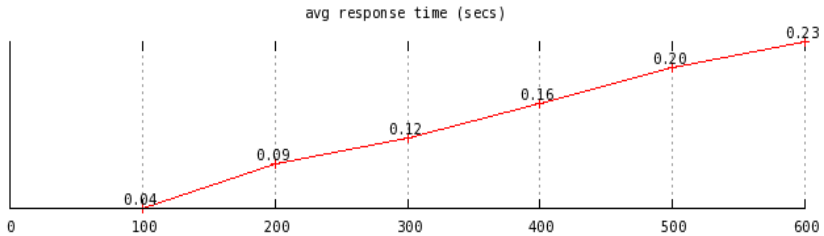


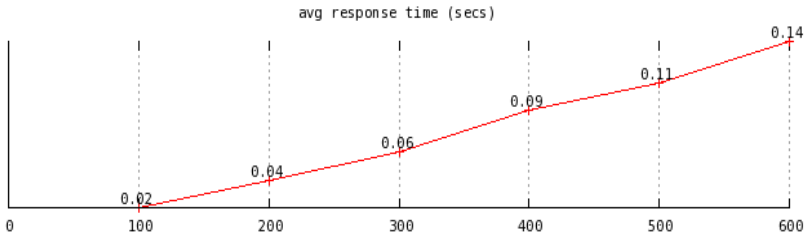
50 Node.js CSCC09 Programming on the Web 10

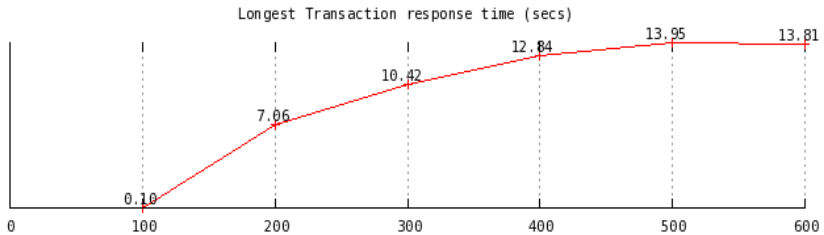
Node.js Performance (njinx vs Node.js)

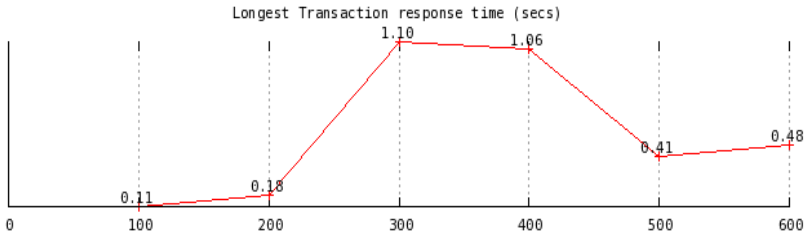


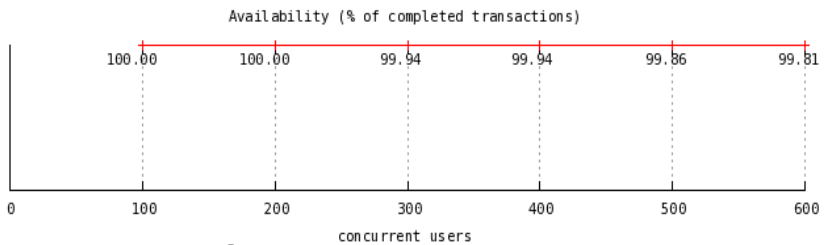


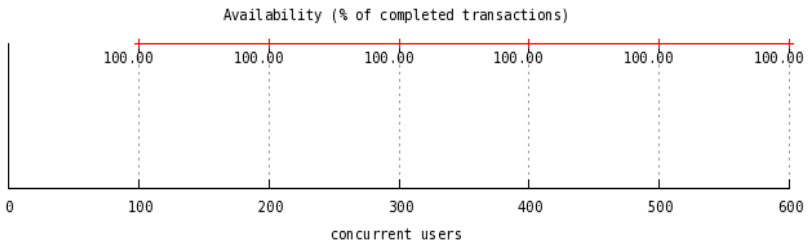












Getting Started

- Node.js is installed on the mathlab server
- You can run it from a command prompt by typing **`node`**
- This will drop you into a Read-Eval-Print-Loop (REPL), that works just Chrome's JavaScript console, so you can run any of your JavaScript commands here, woohoo!
- Node.js can also be used to execute JavaScript files

  Try creating a file called **`hello.js`** containing

```
console.log("Hello World!"); // console is stdout
```

  Now run: **`node hello.js`**

  Node.js will open the file, run your program and exit

on Your Own Machine

- To get started, download the Node.js package for your platform on the Node.js website (http://www.nodejs.org/). You can also get Node.js through your favorite package manager on Mac OS X or Linux distro
- After you've installed Node.js, run it by opening a command prompt and typing "node", as described for mathlab (previous slide)

The Node.js Environment

- Browsers provide an environment for running JavaScript, including access to the DOM
- Loading JavaScript happens through HTML <script> tags
- Node.js similarly provides an execution environment for JavaScript, includes access to files/databases/network (same-origin rule does not apply), and loading code is more sophisticated, based on the concept of a "module"
- When you run JS code in the Node.js REPL, some global Node variables are already defined, e.g.
  - **`global`** object, aka **`root`** same role as global-scope on client

```
x = 1;
global.x  // or root.x
```

Loading Modules

- We are going to be using a number of pre-existing libraries to extend Node.js base functionality
- JavaScript libraries export a single global object, that exposes all their functionality as properties or objects of that object
- For instance, jQuery exports a single object named jQuery AKA ”$”
- The “require” command provides a mechanism for importing external libraries, for example:
- `var http = require("http");`
- **`require`** parses its target script (here “http”) and returns that script’s **`exports`** object

Authoring Modules

- Here's a trivial **example.js** module:

```
console.log("evaluating example.js");
var privateFunction = function () {
  console.log("invisible");
};
exports.message = "hi";
exports.say = function () {
  console.log(exports.message);
}
```

What happens if instead we use just "exports = …" ?

- Which we could import and use as shown:

```
var example = require('./example');  // note "./"
console.log(example);
// logs:    { message: 'hi', say: [Function] }
```

Loading Modules

- `var http = require("http");`
- This loads the **`http`** library and the single exported object available through the **`http`** variable
- By convention, the variable name should match the module name.
- Note you can leave off the ".js" file extension on imported modules (Node.js will automatically supply the suffix)
- Behind the scenes Node.js uses CommonJS to load modules
- If you write your own module for Node.js, your code should conform to the CommonJS specifications.

Installing Modules

Isaac Shleuter

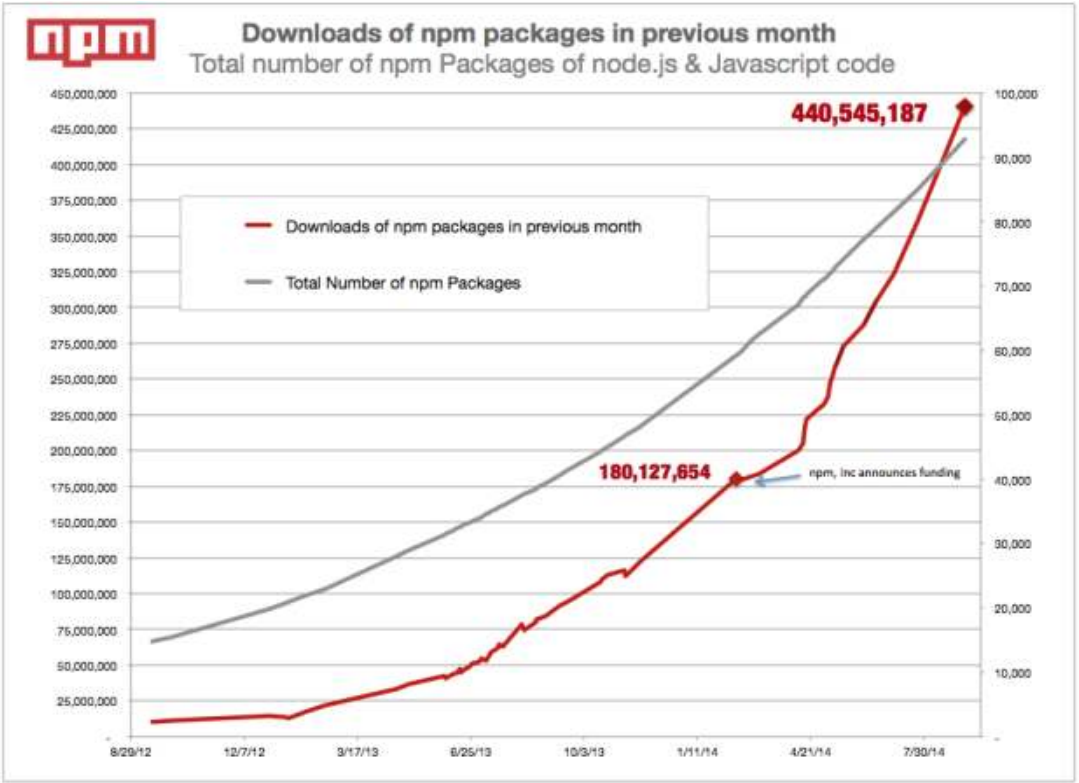
- Where do modules like "http" load from?
- Some, like http and fs, are bundled as part of the standard Node.js library installation
- Other user-contributed modules must be downloaded and installed (www.npmjs.org maintains an online registry)
- Recent versions of Node.js come with a package manager called npm (Node Package Manager), that enables you to easily download and install modules.
- To install a package, from the command line run:

```
npm install package_name
```

Where does **`package_name`** get installed?

Installing Modules

- Are there a lot of npm packages?
- http://www.modulecounts.com/
- Graph for August 2014
  - 440 million requests/month
  - closing in on 100k packages, 40k authors on public npm registry



Installing Modules

- An alternative way to install modules is by bundling them as part of your application by placing a file called **`package.json`** in the same folder as your application, and then running **`npm install`** in that folder.
- npm will extract the list of dependencies from package.json, and then install them locally in the application. For example, Express package.json begins:

```
{ "name": "express",
  "description": "Sinatra inspired web dev framework",
  "version": "3.4.8",
  "author": { "name":"TJ Holowaychuk", "email": ... },
  "contributors": [ ... ],
  "dependencies": { "connect": "2.12.0", ...
  "license": "MIT", ... }
```