Introduction to Node
Agenda

- Server-side programming
- Introduction to Node.js
Recall the HTTP request / reply pattern

<table>
<thead>
<tr>
<th>Request</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;method&gt; &lt;resource identifier&gt; &lt;HTTP Version&gt; &lt;crlf&gt;</code></td>
<td>GET /course/95-702/ HTTP/1.1 Host: <a href="http://www.andrew.cmu.edu">www.andrew.cmu.edu</a> User-Agent: Joe typing Accept: text/html This line intentionally left blank</td>
</tr>
<tr>
<td><code>[&lt;Header&gt;: &lt;value&gt;] &lt;crlf&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>...</code></td>
<td></td>
</tr>
<tr>
<td><code>[&lt;Header&gt;: &lt;value&gt;] &lt;crlf&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>a blank line</code></td>
<td></td>
</tr>
<tr>
<td><code>[entity body]</code></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reply</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;HTTP Version&gt; &lt;Status&gt; &lt;crlf&gt;</code></td>
<td>HTTP/1.1 200 OK</td>
</tr>
<tr>
<td><code>[&lt;Header&gt;: &lt;value&gt;] &lt;crlf&gt;</code></td>
<td>Date: Mon, 24 Jan 2011 15:43:08 GMT</td>
</tr>
<tr>
<td><code>...</code></td>
<td>Server: Apache/1.3.39 (Unix) mod_throttle/3.1.2 ...</td>
</tr>
<tr>
<td><code>[&lt;Header&gt;: &lt;value&gt;] &lt;crlf&gt;</code></td>
<td>Set-Cookie: webstats-cmu=cmu128.2.87.50.8400; ...</td>
</tr>
<tr>
<td><code>a blank line</code></td>
<td>Last-Modified: Sun, 23 Jan 2011 21:46:30 GMT</td>
</tr>
<tr>
<td><code>[response body]</code></td>
<td>ETag: &quot;558425-2336-4d3ca1b6&quot;</td>
</tr>
<tr>
<td></td>
<td>Accept-Ranges: bytes</td>
</tr>
<tr>
<td></td>
<td>Content-Length: 9014</td>
</tr>
<tr>
<td></td>
<td>Content-Type: text/html</td>
</tr>
<tr>
<td></td>
<td>This line intentionally left blank</td>
</tr>
<tr>
<td></td>
<td><code>&lt;HTML&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;HEAD&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;META http-equiv=&quot;Content-Type&quot; content=&quot;text/html; charset=UTF-8&quot;&gt;</code></td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Server-Side Programming

• How does an HTTP static page request work?
• How does PHP work?
• How does Java Enterprise Edition (JEE) work?
• How does Rails work?
Node.js

- Node is yet a different approach
- It is a single, live, ongoing process
  - It is not just run when a request comes
- You need not run behind a web server
  - Though some do
  - Instead, it gives very simple means of incorporating web server functionality into your program
- It gives developers full flexibility to do whatever processing you want to do on the server side
  - With great flexibility
  - And lots of powerful tools to make magic easy.
What can you do with this flexibility?

- Return a static web page
- Query a database and return value
- Do complex computation (not one of node's strengths)
- Connect to 3rd party APIs
- Have ongoing communication across web sockets
- Streaming audio / video / data
- Peer to peer applications with multiple users
- Communicate with devices connected to the server:
  - e.g. weather sensors, light sensors, temperature sensors
  - e.g. web cams, servos, robots, quad helicopters, home automation
Why Node in 67-328

• Allows us to cover the interesting topics concerning server-side programming
• Allows us to re-use and further develop JavaScript skills
  – Universally useful for client-side work
• Exposes you to interesting cutting-edge tools in web application development.
• Nice addition to your resume: Node is hot.
  – e.g. LinkedIn, IBM, Yahoo, Microsoft, PayPal, Groupon, Walmart Labs (who knew they had labs?)
Node

• A bit of history
  – Late 90's and early 00's, companies fought *browser wars*.
  – Companies such as Microsoft and Netscape fought for browser market share by creating non-standard functionality.
  – In the end, standardization won out, and companies stopped fighting on non-standard functionality.

• So how to differentiate? Speed!
  – Browsers started competing on who could be faster.
  – Faster page rendering (displaying)
  – Faster JavaScript execution

• One outcome of this competition is Google's V8 JavaScript Engine.
V8

• http://code.google.com/p/v8/
• Open source JavaScript interpreter
  – (V8 is written in C++, which you don't have to worry about)
• V8 is very very fast
• V8 has a great garbage collection algorithm
  – It doesn't periodically stall, and does not run out of memory.
  – It does run smoothly continuously.
Ryan Dahl

• Ryan Dahl had the insight that he could use V8 to make a completely event-driven, non-blocking, server-side environment.

  — With insights from:
    • Twisted in Python
    • EventMachine in Ruby

• And do so in JavaScript
Event-Driven & Non-Blocking

• These are considered two of the hallmarks of Node.
• You will understand them better as you start experiencing programming in Node.
• Event-Driven:
  – Like in the browser, all processing happens in response to an event.
    • Browser: click event, touch event, mouse movement event
    • Node:
      – network data arrived
      – database returned values
      – read from disk completed
      – custom event arrived over a websocket from a client
Blocking vs Non-blocking code

Synchronous, or blocking example

```javascript
var fs = require("fs");
var path = process.argv[2] || ".";
try {
  var files = fs.readdirSync(path);
  files.forEach(function(file) {
    console.log(file);
  });
} catch (e) { /* deal with error */ }
console.log("I am here.");
```

Asynchronous, or non-blocking example

```javascript
var fs = require("fs");
var path = process.argv[2] || ".";
fs.readdir(path, 
  function(err, files) {
    if (err) {/* deal with error*/ }
    files.forEach(function(file) {
      console.log(file);
    });
  });
console.log("I am here.");
```

Notice differences in:
- How are values returned
- When is code to process files executed
- How are errors handled

• When would "I am here" be logged?
What is the value of non-blocking?

• It can be more efficient.
  – Benchmark data supports this

• It can manage more user requests per second than the old scheme of using threads.
  – Buzzword: it can better handle *web scale*
Shell / Command line

• You will be spending a lot of time using a shell / command line terminal, so choose a good one.

• MacOS / Linux
  – The *bash* shell comes installed
  – *Terminal* is the default terminal application
  – I use *iTerm2*: open source with more functionality
    • [http://iterm2.com](http://iterm2.com)
  – *cdto* is useful for the frequent situation to open a terminal window in a selected finder folder
    • [https://github.com/jbtule/cdto](https://github.com/jbtule/cdto)
I've not used Windows recently – anyone have updates?

• Console window: *Command Prompt* is default
• Shell: cmd.exe or powershell.exe
• Alternatives:
  – Git for Windows includes *Git bash*
    • Right-click on folder to access bash in that folder
    • http://msysgit.github.io
  – Babun
    • Linux-like console with commands
    • Also includes git, curl, and lots more
    • http://babun.github.io
  – Console2 is a console window enhancement
    • http://sourceforge.net/projects/console/
    • (Not updated in 2 years?!?)
  – Others?
Code editor

• Also make sure you have a great code editor
• I've used TextWrangler
  – Free, most of the basics I need, configurable
  – Recently moved to paid version: BBEdit
• Alternatives
  – Brackets: http://brackets.io
    • Built in JavaScript
  – Atom: https://atom.io
    • Built by github ("hackable to the core") on node.js
  – Mac: Sublime Text
  – Windows: TextPad, Notepad++, Visual Studio Express
• Others?
Installing Node

• Download Node.js from http://nodejs.org
  – Available for MacOS, Windows, & others

• Two executables will be installed:
  – node
    • Executable program
    • JavaScript interpreter
    • Additional (primarily I/O) libraries
  – npm – a package manager
    • The way to add in packages (similar to RubyGems)
    • Where most of the extensibility of Node comes from
• Running node from a command line enters into a read-eval-print loop (REPL) interface
• Similar to the browser console, but on your laptop (i.e. server)

```plaintext
$ node
> for (i=0;i<10;i++) {
  ... console.log(i+" mississippi");
  ... }
0 mississippi
1 mississippi
2 mississippi
3 mississippi
4 mississippi
5 mississippi
6 mississippi
7 mississippi
8 mississippi
9 mississippi
undefined
>
```

Use control-d (end of file) to quit
Creating/running a node program

- Create a file with extension .js
- Provide the file as the first argument to node:

  $ cat argv.js
  process.argv.forEach(function (val, index) {
    console.log(index + ': ' + val);
  });
  $ node argv.js
  0: node
  1: /private/tmp/node/argv.js

*cat* means to concatenate files and print to standard output (often the console). I'm using it here to show the program contents.
Running a node program

• You can add additional arguments also

$ node argv.js Baker Porter Hunt Wean
0: node
1: /private/tmp/node/argv.js
2: Baker
3: Porter
4: Hunt
5: Wean
Shortcut, node argv == node argv.js

- You can omit the .js when giving the program file:

```
$ ls
argv.js
$ node argv
0: node
1: /private/tmp/node/argv
```
Demo

• cat.js
Getting Started with Node.js Lab

• For credit, show the TAs once you completed the tasks.
• Finish today, else show in office hours in first 10 minutes of class.
• Lab rules (different than homework):
  – Can collaborate, ask other students, look at each others' code.
  – Must get running on your own laptop.