CAIT Node.js Briefing

Mark Volkmann
mark@ociweb.com
Object Computing, Inc.
June 4, 2013

Those Vendors

- High-dollar software, hardware and consulting **vendors** won't tell you that most problems do not require an expensive, complicated **enterprise solution**
- But it's true!

- **Do the simplest thing** that will work
- It is SO much easier to understand, explain and maintain

Overview ...

- "Node's goal is to provide an easy way **to build scalable network programs**."
  - http://nodejs.org/#about
- A **full programming environment**, not just for building "servers"
- "The **official name** of Node is "Node".
  The unofficial name is "Node.js" to disambiguate it from other nodes."
  - https://github.com/joyent/node/wiki/FAQ
- Runs on top of **Chrome V8** JavaScript engine
- Implemented in **C++ and JavaScript**
- Supported on **Linux, Mac OS X and Windows**
- Created by Ryan Dahl at Joyent

passed control of the project to Isaac Schlueter on 1/30/12

a cartoon from substack

… Overview

- **Event-based** rather than thread-based
  - runs in a single thread
  - can use multiple processes
  - inspired by
    - **Reactor pattern** - http://en.wikipedia.org/wiki/Reactor_pattern
    - **Python Twisted** - http://twistedmatrix.com/
    - **Ruby EventMachine** - http://rubyeventmachine.com/
    - **Nginx** - http://wiki.nginx.org/Main

from Wikipedia,
"The **reactor design pattern** is an event handling pattern for handling service requests delivered concurrently to a service handler by one or more inputs.
The service handler then demultiplexes the incoming requests and dispatches them synchronously to the associated request handlers."

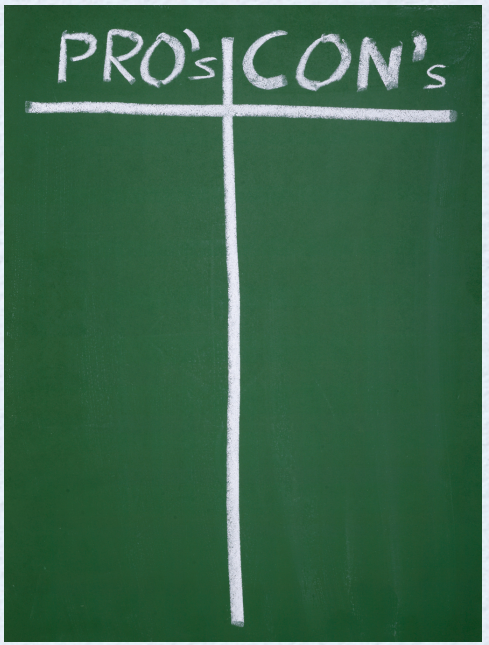
- Assumes most time consuming operations involve **I/O**
  - invoked asynchronously; non-blocking
  - a callback function is invoked when they complete

Chrome V8

- From Google
- Used by Chrome browser and Node.js
- Implemented in C++
- Currently supports ECMAScript 5
- Node adopts the JavaScript syntax supported by V8
  - so will support ES6 when V8 supports it

Should You Use It?

- **Reasons To Use**
  - application can benefit from asynchronous, non-blocking I/O
  - application is not compute-intensive
  - V8 engine is fast enough
  - prefer callback or actor models of concurrency
    - over thread-based approach with synchronized access to mutable state
  - same language on client and server
  - like dynamically typed languages
  - large number of JavaScript developers
- **Some issues being addressed**
  - finding packages - there are a large number of them and finding the best ones isn't easy enough
  - debugging - stack traces from asynchronously executed code are incomplete
  - event loop - sometimes difficult to determine why a program isn't exiting
    - typically due to open connections



Multiple Threads & Processes

- Node uses multiple threads internally
  - to simulate non-blocking file I/O
- You can't create new threads
  - unless you use "Threads A GoGo"
    - https://github.com/xk/node-threads-a-gogo
    - "provides an asynchronous, evented and/or continuation passing style API for moving blocking/longish CPU-bound tasks out of Node's event loop to **JavaScript threads that run in parallel** in the background and that **use all the available CPU cores automatically**; all **from within a single Node process**"
- Can use multiple, cooperating processes
  - see "Child Processes" core module
    - processes created with `fork` function can emit and listen for messages
  - see "Clusters" core module
    - "easily create a network of processes that all share server ports"

from Issac Schlueter on 11/7/12,

"Node uses threads for file system IO,
and for some slow CPU-intensive operations,
and for system calls that are not available asynchronously,
and for spawning child processes
(since you can't actually do that without a fork call).

It does *not* use threads for async network IO,
because it's unnecessary, and it certainly does not
spawn a thread for each request to an HTTP server,
or for each outbound HTTP request it makes."

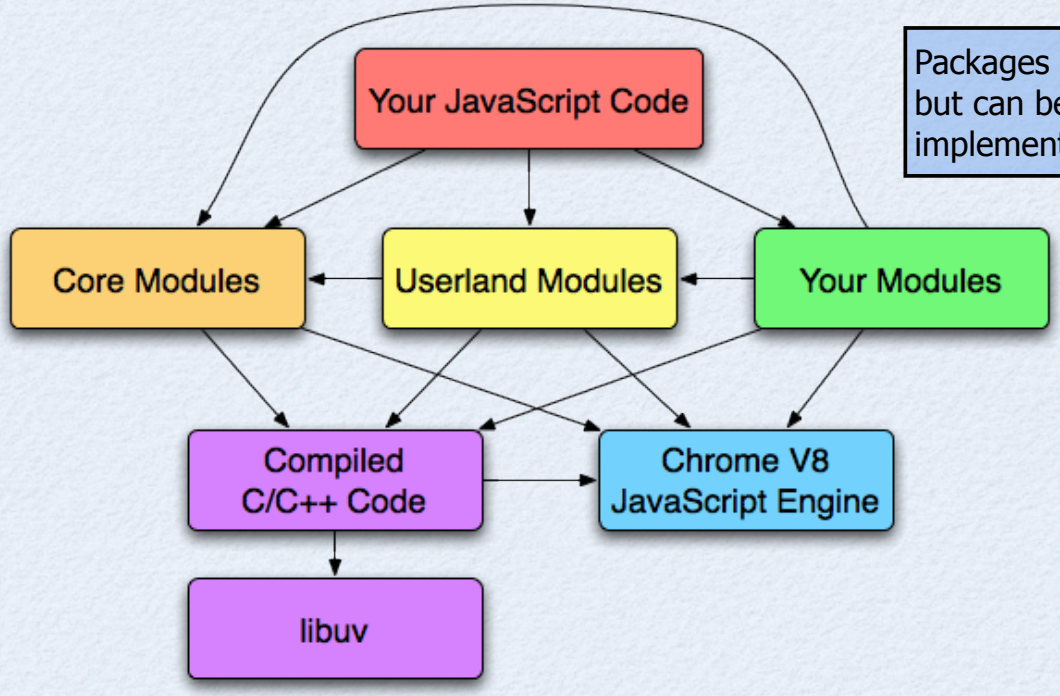
Where To Look For Functionality

1. JavaScript
   - core classes: **`Arguments`**, **`Array`**, **`Boolean`**, **`Date`**, **`Error`**, **`Function`**, **`Global`**, **`JSON`**, **`Math`**, **`Number`**, **`Object`**, **`RegExp`**, **`String`**

see JavaScript reference at https://developer.mozilla.org/en-US/docs/JavaScript/Reference

2. Core Modules
   - included with Node
   - http://nodejs.org/docs/latest/api/
   - view source at https://github.com/joyent/node
     - JavaScript is in **`lib`** directory
     - C++ code is in **`src`** directory
3. Userland Modules (third party)
   - typically installed using NPM tool
   - https://npmjs.org/
   - 31,535 NPM packages on 6/3/13
4. Write yourself

The Stack



Packages have JavaScript APIs, but can be partially implemented in C++.

libuv is a Node-specific abstraction over Windows Input/Output Completion Port (IOCP) and Unix libev (https://github.com/joyent/libuv)

Why JavaScript?

- First-class functions
- Closures
- Flexible objects
  - can add attributes and methods at any time
  - nice syntax for literal objects and arrays
- Only language supported by web browsers
- Can use same programming language on client and server
- Callbacks for asynchronous operations
  - callbacks vs. promises

JavaScript Classes

- Many people that have only taken a cursory look at JavaScript criticize it
- A common complaint is that prototypal inheritance is weird and complicated
- Let's look at that

cylinder.js

```
function Cylinder(height, diameter) {
  this.height = height;
  this.diameter = diameter;
}

Cylinder.prototype.getVolume = function () {
  var radius = this.diameter / 2;
  return this.height * Math.PI * radius * radius;
};

var cyl = new Cylinder(4, 2);
// Output volume of cylinder with two decimal places.
console.log('volume =', cyl.getVolume().toFixed(2));
```

- Not weird and not complicated!

Node Versions

- Stable versions have even minor release numbers
  - ex. 0.10.9
- Unstable versions have odd minor release numbers
  - ex. 0.11.2
  - where work toward next stable version takes place

Primary Node Resources

- http://nodejs.org
  - click "INSTALL" button to download platform-specific installer for latest stable version
  - see API docs
- Node modules at http://npmjs.org
  - look at "express" module
- Let's install express
  - `npm install express`

Demos

- Running Node programs
- REPL
- Serving static files - HTML, CSS, JavaScript, images, ...
- Implementing and calling REST services
- Saving data in a NoSQL database
- Pushing updates to browser clients using WebSockets
- Using multiple processors on web server

These slides and the code for the last four demos is available at
https://github.com/mvolkmann/nodeExpressMongoWebSocketsCluster

Running Node Programs

- Pass JavaScript file path to **`node`** command
  - **`node cylinder.js`**
- Can pass command-line arguments into program
  - access with **`process.argv`**
  - it's an array containing **`'node'`**, absolute file path to JavaScript file, and command-line arguments
  - so **`process.argv[2]`** holds first command-line argument

REPL

- Tool for evaluating JavaScript statements
  - outputs the value of each
- Useful for verifying understanding
- To start, enter **`node`**
- To load definitions in a JavaScript file enter **`.load file-path`**

Demo notes:
cd to `training/JavaScript/labs/prototypal`

```
$ node
> .load cylinder.js
... outputs each statement in file and its value ...
> c = new Cylinder(10, 4)
{ height: 10, diameter: 4 }
> c.height
10
> c.getVolume()
125.66370614359172
> .exit
```

- For help, enter **`.help`**
- To exit, enter **`.exit`**

Static File Web Server

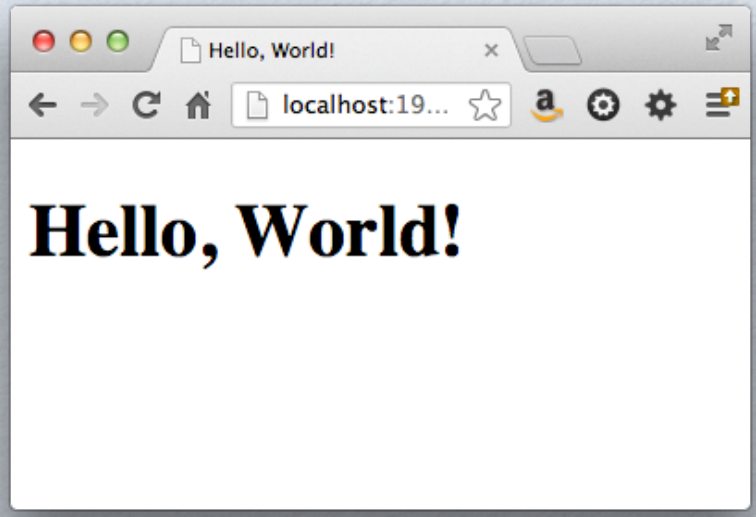
- Many options
  - can use core `http` module, Express, Strata, ...
  - we will use Express
- To install Express
  - `mkdir node_modules`
  - `npm install express`
- Example

static.js

```
var express = require('express');
var app = express();
app.use(express.static(__dirname));
app.listen(1919);
```

- To run server, enter `node static.js`
- Browse files in current directory with `http://localhost:1919/file-name`
  - can omit file-name for `index.html`

Demo notes:
cd to `express` directory under `nodejs-labs`
enter `node static.js`
browse `http://localhost:1919`
and `http://localhost:1919/google.gif`



Let's build an addressbook app!

REST Web Server ...

server1.js

```
var express = require('express');
var app = express();
var book = {}; // just storing data in memory

app.use(express.static(__dirname + '/public')); // serve static files
app.use(express.bodyParser()); // automatically convert JSON requests to objects

function del(req, res) {
  var id = req.params.id;
  if (book[id]) {
    delete book[id];
    res.send(200);
  } else {
    res.send(404);
  }
}

function get(req, res) {
  var id = req.params.id;
  var person = book[id];
  if (person) {
    res.set('Content-Type', 'application/json');
    res.send(200, JSON.stringify(person));
  } else {
    res.send(404);
  }
}
```

… REST Web Server

server1.js

```
function list(req, res) {
  res.set('Content-Type', 'application/json');
  res.send(200, JSON.stringify(Object.keys(book)));
}

function put(req, res) {
  var id = req.params.id;
  var person = req.body;
  book[id] = person;
  res.send(200);
}

app['delete']('/addressbook/:id', del);
app.get('/addressbook/list', list);
app.get('/addressbook/:id', get);
app.put('/addressbook/:id', put);

var PORT = 3000;
app.listen(PORT);
console.log('Express server listening on port', PORT);
```

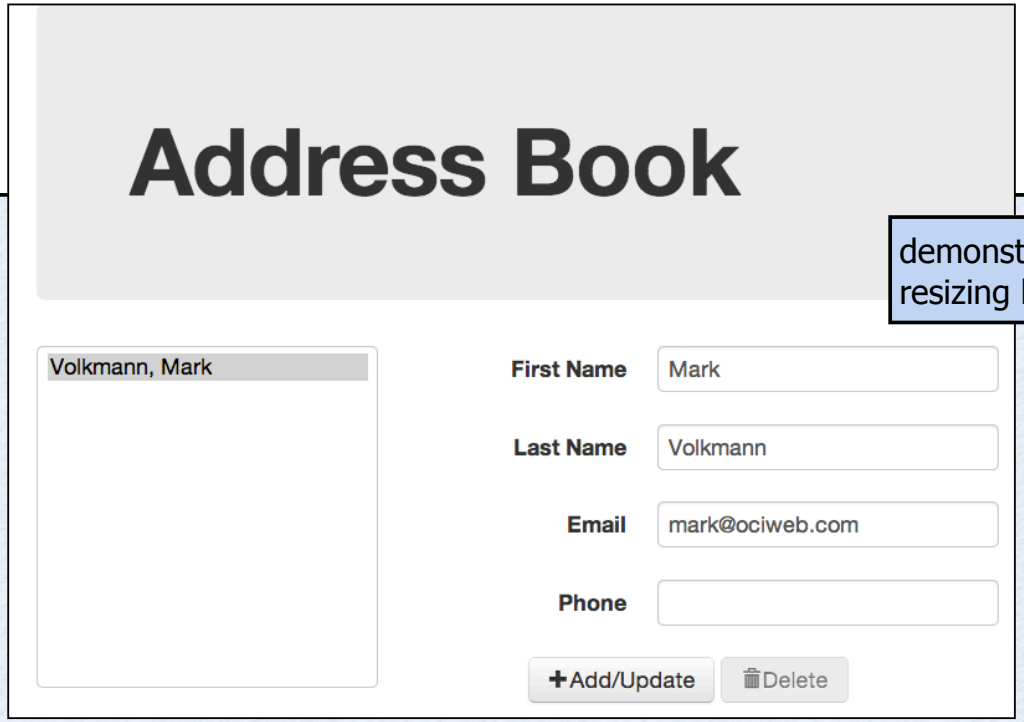
referred to as "routes"

Can test with curl command and browser without creating any HTML.

HTML

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" href="lib/bootstrap/css/bootstrap.min.css">
    <link rel="stylesheet" href="lib/bootstrap/css/bootstrap-responsive.min.css">
    <link rel="stylesheet" href="addressbook.css">
    <script src="lib/jquery-2.0.1.min.js"></script>
    <script src="lib/bootstrap/js/bootstrap.min.js"></script>
    <script src="addressbook.js"></script>
  </head>
  <body>
    ...
  </body>
</html>
```



demonstrate responsiveness by resizing browser window

Browser JavaScript ...

addressbook.js

```
(function () {
  var emailInput, firstNameInput, lastNameInput, phoneInput;
  var deleteBtn, nameList;
  var URL_PREFIX = 'http://localhost:3000/addressbook/';

  function Person(firstName, lastName, email, phone) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.email = email;
    this.phone = phone;
  }

  function add() {
    var id = getId();

    var doneCb = function () {
      insertId(id);
      nameList.val(getKey());
    };

    $.ajax(URL_PREFIX + id, {
      type: 'PUT',
      contentType: 'application/json',
      data: JSON.stringify(makePerson())
    }).done(doneCb).error(failCb);
  }

  function addId(id) {
    var pieces = id.split('-');
    var key = pieces.join(', ');
    nameList.append($('<option>', {id: id}).text(key));
  }
```

... Browser JavaScript ...

addressbook.js

```
function clear() {
  firstNameInput.val('');
  lastNameInput.val('');
  emailInput.val('');
  phoneInput.val('');
}

function del() {
  var doneCb = function () {
    $('#' + id).remove();
    clear();
    deleteBtn[0].disabled = true;
  };

  var id = getId();
  $.ajax(URL_PREFIX + id, {type: 'DELETE'}).done(doneCb).error(failCb);
}

function failCb(err) {
  alert(err.toString());
  console.log('error:', err);
}

function getId() {
  return lastNameInput.val() + '-' + firstNameInput.val();
}

function getKey() {
  return lastNameInput.val() + ', ' + firstNameInput.val();
}
```

... Browser JavaScript ...

addressbook.js

```
function insertId(id) {
  var pieces = id.split('-');
  var key = pieces.join(', ');

  var option = $('<option>', {id: id}).text(key);

  var added = false;
  nameList.children().each(function (index, op) {
    if (added) return;
    if (id === op.id) {
      added = true; // already exists
    } else if (id < op.id) {
      option.insertBefore(op);
      added = true;
    }
  });

  if (!added) nameList.append(option);
}

function load() {
  var doneCb = function (ids) {
    ids.sort().forEach(addId);
  };

  $.getJSON(URL_PREFIX + 'list').done(doneCb).fail(failCb);
}

function makePerson() {
  return new Person(
    firstNameInput.val(),
    lastNameInput.val(),
    emailInput.val(),
    phoneInput.val());
}
```

... Browser JavaScript

addressbook.js

```
  function select(event) {
    var option = $(event.target);
    // If the select element was selected instead of one of its options ...
    if (option.prop('tagName') !== 'OPTION') return;

    var id = option.attr('id');
    var key = option.text();

    var doneCb = function (person) {
      firstNameInput.val(person.firstName);
      lastNameInput.val(person.lastName);
      emailInput.val(person.email);
      phoneInput.val(person.phone);
      deleteBtn[0].disabled = false;
    };

    $.getJSON(URL_PREFIX + id).done(doneCb).fail(failCb);
  }

  $(function () {
    firstNameInput = $('#firstName');
    lastNameInput = $('#lastName');
    emailInput = $('#email');
    phoneInput = $('#phone');
    nameList = $('#nameList');
    deleteBtn = $('#delete');

    load();

    $('#add').click(add);
    deleteBtn.click(del);
    nameList.click(select);
  });
}());
```

Where Are We Now?

- Works great, but there are two big problems
- 1) All the data is lost when the server is shut down.
- 2) If there is more than one client,
  they only see changes of others after a refresh

- Let's fix the first problem

MongoDB

- A popular NoSQL database
- To install on Mac OS X
  - `brew install mongodb`
- To start daemon process
  - `mongod`
- To start a MongoDB shell for interactively creating, retrieving, updating deleting data
  - `mongo`
- To install a Node.js module for MongoDB
  - `npm install mongodb`

Web Server With MongoDB ...

server2.js

```
function getDatabase() {
  var MongoClient = require('mongodb').MongoClient;
  MongoClient.connect('mongodb://localhost:27017/demoDb', function (err, db) {
    if (err) {
      console.error('failed to connect to database:', err);
    } else {
      getCollection(db);
    }
  });
}

function getCollection(db) {
  db.collection('addressbook', function (err, collection) {
    if (err) {
      console.error('failed to get collection:', err);
    } else {
      setupServer(collection);
    }
  });
}

function setupServer(collection) {
  var express = require('express');
  var app = express();
  app.use(express.static(__dirname + '/public')); // serve static files
  app.use(express.bodyParser()); // convert JSON requests to objects
```

... Web Server With MongoDB ...

server2.js

```
function getMongoQuery(req) {
  var id = req.params.id;
  var pieces = id.split('-');
  return {lastName: pieces[0], firstName: pieces[1]};
}

function del(req, res) {
  collection.remove(getMongoQuery(req), function (err) {
    res.send(err ? 500 : 200, err);
  });
}

function get(req, res) {
  var cursor = collection.findOne(getMongoQuery(req), function (err, person) {

    if (err) {
      res.send(500, err);
    } else if (person) {
      res.set('Content-Type', 'application/json');
      res.send(200 ,JSON.stringify(person));
    } else {
      res.send(404);
    }
  });
}
```

... Web Server With MongoDB ...

server2.js

```
function list(req, res) {
  collection.find().toArray(function (err, persons) {
    if (err) {
      res.send(500, err);
    } else {
      var ids = persons.map(function (person) {
        return person.lastName + '-' + person.firstName;
      });
      res.set('Content-Type', 'application/json');
      res.send(200, JSON.stringify(ids));
    }
  });
}

function put(req, res) {
  var person = req.body;
  var options = {upsert: true}; // insert if not present
  collection.update(getMongoQuery(req), person, options, function (err) {
    res.send(err ? 500 : 200, err);
  });
}
```

... Web Server With MongoDB

server2.js

```
  app['delete']('/addressbook/:id', del);
  app.get('/addressbook/list', list);
  app.get('/addressbook/:id', get);
  app.put('/addressbook/:id', put);

  var PORT = 3000;
  app.listen(PORT);
  console.log('Express server listening on port', PORT);
}

getDatabase();
```

Where Are We Now?

- Data is persisted across server restarts now
- Let's fix the problem with sharing changes between clients

WebSockets

- Wikipedia definition
  - "a **web technology** providing **for bi-directional, full-duplex communications channels over a single TCP connection**"
  - "The WebSocket **API** is being standardized by the W3C, and the WebSocket **protocol** has been standardized by the IETF"
- See "The WebSocket API - W3C Editor's Draft" 23 April 2013
  - http://dev.w3.org/html5/websockets/
- Supports long-lived connections between client and server
- Many server and client libraries
  - client libraries are for non-web clients; modern browsers have built-in support
- We will use the Node.js module "ws" at http://einaros.github.io/ws/
  - to install, `npm install ws`

WebServer With WebSockets ...

- Add **broadcast** and **setupWebSocket** functions on next slide
- Add these lines in **setupServer** after configuring app to setup use of WebSockets

server3.js

```
var wsArray = [];
setupWebSocket(app);
```

- Add calls to **broadcast** in **del** and **put** functions after response is sent so all clients are informed about these actions

server3.js

```
broadcast('delete', req.params.id);

broadcast('put', req.params.id);
```

… WebServer With WebSockets

server3.js

```
function setupWebSocket(app) {
  var WebSocketServer = require('ws').Server;
  var http = require('http');
  var server = http.createServer(app);
  var wss = new WebSocketServer({server: server});
  wss.on('connection', function (ws) {
    wsArray.push(ws);
  });
  server.listen(8080);
}


function broadcast(event, id) {
  var obj = {event: event, id: id};
  var msg = JSON.stringify(obj);

  wsArray.forEach(function (ws, index) {
    ws.send(msg, function (err) {
      if (err) wsArray[index] = null; // stop sending to this ws
    });
  });

  // Remove nulls from array.
  wsArray = wsArray.filter(function (ws) { return ws; });
}
```

happens when a client disconnects

Browser JavaScript

- In **`addressbook.js`**, add the **`setupWebSocket`** function and call it before the call to **`load`** in the ready function

addressbook.js

```
function setupWebSocket() {
  var ws = new WebSocket('ws://localhost:8080');
  ws.onmessage = function (event) {
    var obj = JSON.parse(event.data);
    if (obj.event === 'put') {
      insertId(obj.id);
    } else if (obj.event === 'delete') {
      $('#' + obj.id).remove();
    } else {
      console.error('received unrecognized message "' + event.data + '"');
    }
  };
}
```

Where Are We Now?

- All clients are updated when any client performs a put or delete now
- But every client request is being processed by a single thread on the server
- If we have a large number of clients, it would be nice to take advantage of multiple processors in the server machine

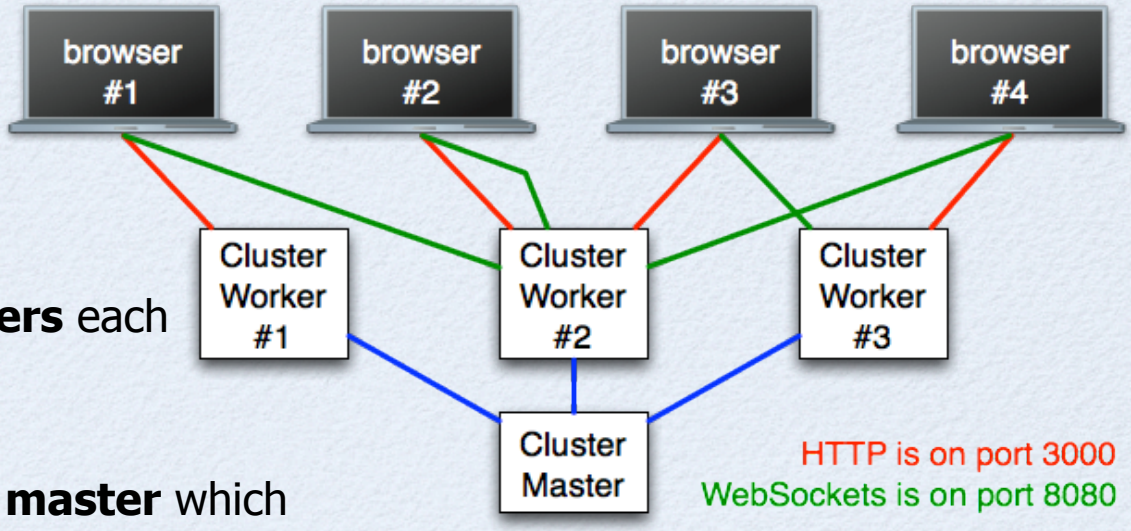
Node.js Cluster Module

- "easily create a network of processes that all share server ports"
  - works with any TCP-based server, including HTTP and HTTPS
- Builds on "Child Processes" module
- Initial process is called "master"
  - only process that listens on selected port
  - uses inter-process communication (IPC) pipes to communicate with workers
- Forked processes are called "workers"
  - typically want to fork a number of workers not greater than number of processors
  - get number of processors with `os.cpus().length`
  - no guarantees about order of selection of workers to handle requests
    - distributes connections across workers, but doesn't distribute requests
    - once a client gets a connection, all their requests will go to the same worker

"The Jewel Box (also known as NGC 4755, the Kappa Crucis **Cluster** and Caldwell 94) is an open cluster in the constellation of Crux." ... Wikipedia

HTTP & WebSocket Connections

- Browser **clients** connect to
  - an **HTTP server** managed by one of the cluster workers
  - a **WebSocket** managed by one of the cluster workers
- The cluster **master** and **workers** each run in a separate process
- Cluster **workers** can send messages to their cluster **master** which has access to a collection of all the cluster **workers** and can send messages to them
- Each cluster **worker** holds a collection of **WebSocket** connections and can send messages to them



Web Server With Cluster ...

- Add this to start of server code

server4.js

```
var cluster = require('cluster');
if (cluster.isMaster) return startWorkers();

function startWorkers() {
  var handleMsg = function (worker, msg) {
    Object.keys(cluster.workers).forEach(function (id) {
      var otherWorker = cluster.workers[id];
      // Don't send to sender.
      if (otherWorker.process.pid !== msg.senderPid) {
        otherWorker.process.send(msg);
      }
    });
  };

  var addWorker = function () {
    var worker = cluster.fork();
    worker.on('message', function (msg) {
      handleMsg(worker, msg);
    });
  };

  // If a worker exits, start a new one.
  cluster.on('exit', addWorker);

  // Fork worker processes.
  var cpuCount = require('os').cpus().length;
  for (var i = 1; i < cpuCount; i++) {
    addWorker();
  }
}
```

doesn't help clients that were using the exited worker

… Web Server With Cluster …

- Add this in **`setupServer`** after call to **`setupWebSocket`**

server4.js

```
// Listen for messages from cluster master.
process.on('message', function (msg) {
  if (msg.senderPid !== process.pid) sendToClients(msg);
});
```

- Change **`broadcast`** function

server4.js

```
function broadcast(event, id) {
  var msg = {event: event, id: id, senderPid: process.pid};

  // Send to cluster master.
  process.send(msg);

  sendToClients(msg);
}
```

on next slide

… Web Server With Cluster

- Add **`sendToClients`** function

server4.js

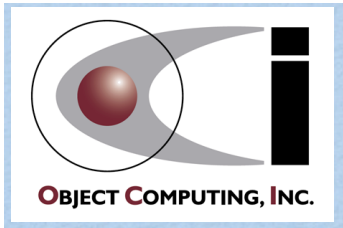
```
function sendToClients(msg) {
  var s = JSON.stringify(msg);
  wsArray.forEach(function (ws, index) {
    ws.send(s, function (err) {
      if (err) wsArray[index] = null; // stop sending to this ws
    });
  });

  // Remove nulls from array.
  wsArray = wsArray.filter(function (ws) { return ws; });
}
```

happens when a client disconnects

Where Are We Now?

- The last feature was more complicated,
  but hopefully each piece of it is understandable
- How difficult would this be to implement
  in other programming languages?



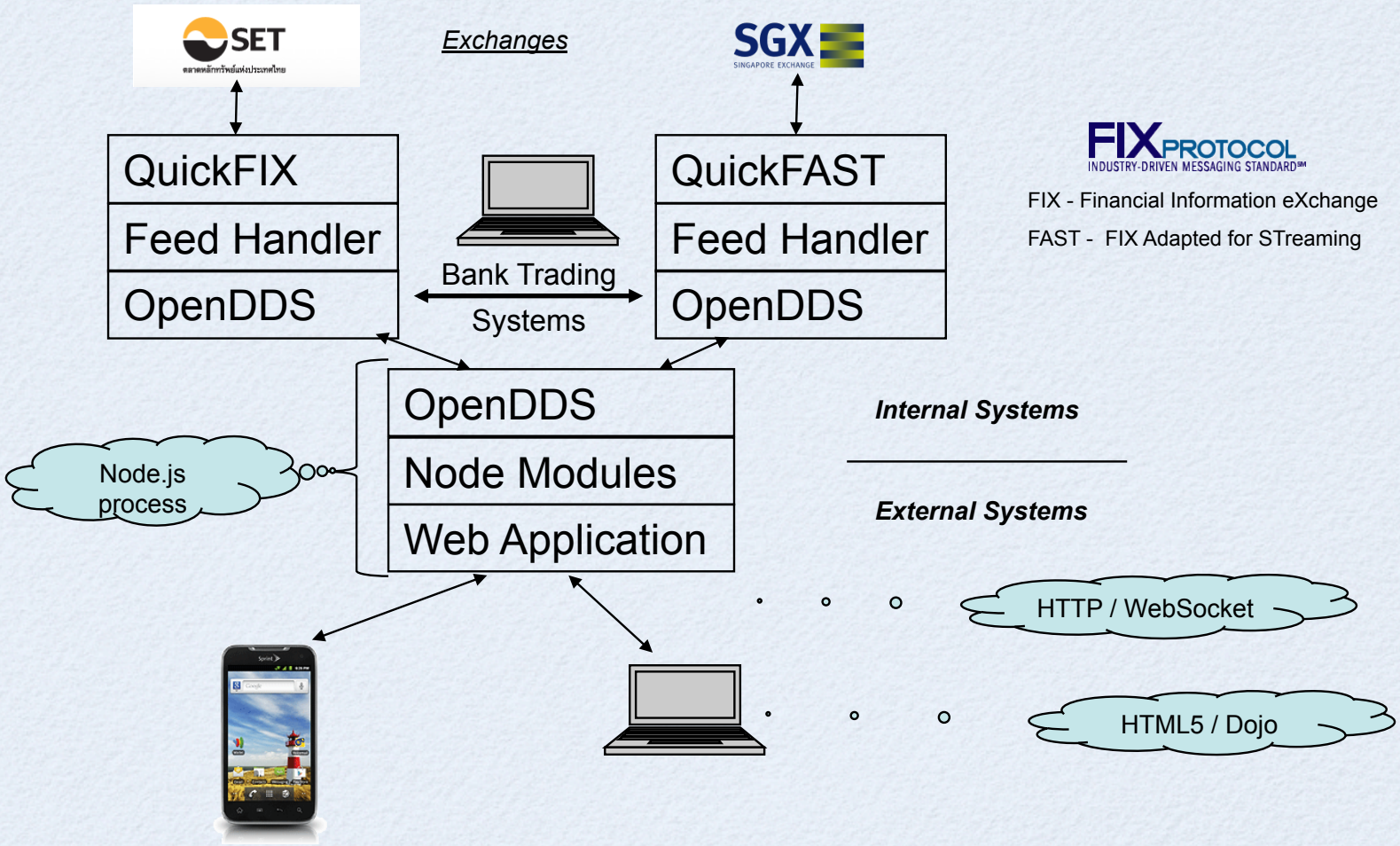
Case Study

- JavaScript/DDS Integration
  - DDS is an Object Management Group specification for a data distribution service for real-time systems, i.e. 3rd generation pub/sub

Extending DDS Global Data Space to Web

- Problem Statement
  - large Asian bank operating in several countries
  - expanding country-specific financial trading services to >10K users using desktop and mobile devices
  - hold down costs by moving to an all open source solution
- Solution Step #1
  - switch internal trading systems messaging to OpenDDS
  - implementation of OMG DDS 1.2 and DDS-RTPS 2.1 specifications
    - Data Centric Publish/Subscribe (DCPS) layer
  - open source, permissive license with public source repository
  - core libraries written in C++; includes Java API
  - configurable transports
    - TCP, RTPS, UDP-unicast, UDP-multicast, shared memory

Solution Architecture



Node.js Resources

- **Main site** - http://nodejs.org/
- **API doc** - http://nodejs.org/docs/latest/api/
- **NPM Registry Search** - https://npmjs.org/
- **How To Node** - http://howtonode.org/
- **node-toolbox** - http://toolbox.no.de/
- **NodeUp podcast** - http://nodeup.com/
- **Felix Geisendoerfer's guide** - http://nodeguide.com

- **JavaScript Reference** - https://developer.mozilla.org/en-US/docs/JavaScript/Reference
- **JSLint** - http://www.jslint.com/
- **JSHint** - http://www.jshint.com/

Closing Thought

- Take the road LESS COMPLICATED!

```
});
```

"Two roads diverged in a yellow wood,
And sorry I could not travel both
And be one traveler, long I stood
And looked down one as far as I could
To where it bent in the undergrowth;
Then took the other, as just as fair,
And having perhaps the better claim,
Because it was grassy and wanted wear;
Though as for that the passing there
Had worn them really about the same,
And both that morning equally lay
In leaves no step had trodden black.
Oh, I kept the first for another day!
Yet knowing how way leads on to way,
I doubted if I should ever come back.
I shall be telling this with a sigh
Somewhere ages and ages hence:
Two roads diverged in a wood, and I -
I took the one less traveled by,
And that has made all the difference."

Robert Frost, "The Road Not Taken", 1920