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.

Node gives Azure users the first end-to-end JavaScript experience for the development of a whole new class of real-time applications.
[illegible]
Principal Program Manager, Interoperability Strategy

Node's evented I/O model freed us from worrying about locking and concurrency issues that are common with multithreaded async I/O.
[illegible]
Principal Member, Technical Staff

On the server side, our entire mobile software stack is completely built in Node. One reason was scale. The second is Node showed us huge performance gains.
[illegible]
Director of Engineering, Mobile

Node.js is the execution core Manhattan. Allowing develope to build one code base using language - that is the nirvana developers.
[illegible]
Sr. Product Manger, Cocktail

Web applications

With NodeJS

INTRODUCTION

What is Node

- Node is a new platform for developing web application **servers**, and general purpose programming.
- Node is designed for extreme scalability in networked applications through
  - **single** execution **thread**
  - **event-driven** architecture
  - **asynchronous** I/O

How much faster?

Time (seconds)		
Loop size	PHP	node.js
100,000	0.077	0.002
1,000,000	0.759	0.016
10,000,000	7.605	0.157
100,000,000	75.159	1.567

From: http://www.matt-knight.co.uk/2011/node-js-vs-php-performance-maths/

Like no other

- The Node platform ≠ programming languages for web applications (PHP/Python/Ruby/Java/ …)
- The Node server ≠ the containers which deliver the HTTP protocol to web clients (Apache/Tomcat/Glassfish/ …).
- Normally: the server model uses blocking I/O and threads for concurrency.

  Blocking I/O causes threads to wait on I/O while the application server handles requests.

Node: main ideas

- A single execution thread
- Event loop is constantly running and dispatching the next immediately runnable task
- I/O calls do not block the thread. Instead, programmer sets up request handling functions that fire when some things become available.
- We already familiar with this model: The event loop and event handler model is used for JavaScript execution in a web browser.

JavaScript

- Before it was used for adding basic interactivity to your web pages – on the client side – “frontend”.
- With the introduction of jQuery, Prototype, Ajax – more powerful tasks for JavaScript. But this was all still frontend stuff.
- To write “backend”- you learned PHP, Ruby, Java…

- Node.js: JavaScript on the server?

NodeJS is a new context for JavaScript

- JavaScript is a "complete" language: you can use it in many contexts and achieve everything with it you can achieve with any other "complete" language.

- Usual context to run JS code: browser
- New context to run JS code: NodeJS – allows to run JS code outside the browser

- To execute any code it needs to be interpreted. That is exactly what browser software did and what NodeJS does. It uses the same Google's V8 Virtual Machine, the same runtime environment for JavaScript that Google Chrome uses.

- Node.js is really two things: **a runtime environment and a library**.

Node runs JavaScript, but isn’t JavaScript

- Node is a program for ***running*** JavaScript, but isn’t JavaScript itself.

- JavaScript is a poor language for writing server-side tools: bad for dealing with operating system-level sockets and network connectivity.
- But Node isn’t written in JavaScript; it’s written in C!

- JavaScript is sending instructions to a C program that can be carried out in the dungeons of your OS.

- Node is a program that you **feed JavaScript instructions**. You can actually write a server without worrying about how it is implemented in C.

Node in the enterprise world

- eBay released ql.io, an event-driven aggregation tool for API consumption.
- LinkedIn has said publicly that they achieved massive performance gains with Node.
- Voxer is using Node and Riak to build its walkie talkie-style mobile app (and have open sourced their in-house Riak client)
- Yahoo has used Node to develop their Mojito platform.
- WalMart Labs has been using lots of Node in mobile development.
- Promising game development done using tools like Node, socket.io, and Redis. Example: http://ajaxian.com/archives/aves-game-engine
- Even Oracle announced that they planned to develop Nashorn, a JVM-based JavaScript engine.

Goal:

UNDERSTAND HOW TO USE JAVASCRIPT IN NODE CONTEXT

Based on:
https://github.com/ManuelKiessling/NodeBeginnerBook/tree/master/code/application

Case study: customizable web page

Specifications:

- When requesting http://*domain*/start, the user should see a welcome page which displays a file upload form.
- By choosing an image file to upload and submitting the form, this image should then be uploaded to http://*domain*/upload, where it is displayed once the upload is finished.

To do list

- Create an HTTP server
- Design request router
- Design request handlers

Task 1

HTTP SERVER

Implementing server and app at the same time

- The typical setup would be an Apache HTTP server with mod_php5 installed.
  This means that serving requests doesn't happen within PHP itself.
- With Node.js, we not only implement our application, we also implement the whole HTTP server. Our web application and its web server are basically the same.

NodeJS programming: separation of concerns

- To keep the different parts of the code separated we put them into *modules*.
- This allows us to have a clean main file, which we execute with Node.js, and clean modules that can be used by the main file and reused by any other application.

Server module: *server.js*

```
var http = require("http");

http.createServer(function(request, response) {
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("Hello World");
  response.end();
}).listen(8888);
```

Explanation step-by-step

- The first line *require*s the *http* module and makes it accessible through the variable *http*.
- We then call one of the functions the http module offers: *createServer*. This function returns an object, and this object has a method named *listen*, and takes a numeric value which indicates the port number our HTTP server is going to listen on.

```
var http = require("http");

var server = http.createServer();
server.listen(8888);
```

- That starts an HTTP server listening at port 8888 and doing nothing else (not even answering any incoming requests).

Parameter of *createServer*: function

```
function(request, response) {
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("Hello World");
  response.end();
}
```

- This function definition IS the first (and only) parameter we are giving to the *createServer()* call.
- Reminder: in JavaScript, functions can be passed around like any other value.

We could write it as a named function

```
var http = require("http");

function onRequest(request, response) {
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("Hello World");
  response.end();
}

http.createServer(onRequest).listen(8888);
```

Now it is clear: *server.js*

```
var http = require("http");

http.createServer(function(request, response) {
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("Hello World");
  response.end();
}).listen(8888);
```

Digression:

CALLBACKS VS SEQUENTIAL EXECUTION. DATABASE QUERIES

Normal vs. asynchronous execution

```
var result = database.query("SELECT
            * FROM hugetable");
console.log("Hello World");
```

```
database.query("SELECT * FROM
            hugetable", function(rows)
            {var result = rows; });
console.log("Hello World");
```

- Sequential: finish reading all the results, then execute the next line of code. The user would have to wait for the database query to finish before he can see a message.
- In the execution model of PHP, the web server starts its own PHP process for every HTTP request. If one of these requests results in the execution of a slow piece of code, it results in a slow page load <u>for this particular user</u>, but other users requesting other pages would not be affected.

Normal vs. asynchronous execution

```
var result = database.query("SELECT
            * FROM hugetable");
console.log("Hello World");
```

```
database.query("SELECT * FROM
            hugetable", function(rows)
            {var result = rows; });
console.log("Hello World");
```

- Asynchronous: continue to the next line of code, and constantly check whether the slow process finished. When finished, execute callback code.
- In Node there is only one single process. If there is a slow database query somewhere in this process, this affects the whole process - everything comes to a halt until the slow query has finished.
- To avoid this, Node introduces the concept of event-driven, asynchronous **callbacks**, by utilizing an event loop.

Normal vs. asynchronous execution

```
var result = database.query("SELECT
             * FROM hugetable");
console.log("Hello World");
```

```
database.query("SELECT * FROM
             hugetable", function(rows)
             {var result = rows; });
console.log("Hello World");
```

Here our code is ***synchronous***:

first do the database query, and only when this is done,

then write to the console.

Here, instead of expecting *database.query()* to directly return a result to us, we pass it a second parameter, an anonymous function

…

Asynchronous request handling

Node.js can handle the database request ***asynchronously***.

- It takes the query and sends it to the database.
- Instead of waiting for it to be finished, it makes a note that says "When in the future the database server is done and sends the result, then I have to execute the anonymous function that was passed to *database.query()*."
- Then, it immediately executes *console.log()*, and afterwards, it enters the **event loop**.
- Node.js continuously cycles through this loop again and again whenever there is nothing else to do, waiting for events. Events like, e.g., a slow database query finally delivering its results.

Asynchronous database drivers

To take advantage of asynchronous database queries, we need a database that can notify event loop about finishing its query

- SQLite
- Redis
- MongoDB
- …

MongoDB

- **MongoDB** (from "hu**mongo**us") is an open source document-oriented database system developed and supported by 10gen.
- It is part of the NoSQL family of database systems.
- Instead of storing data in tables as is done in a "classical" relational database, MongoDB stores structured data as JSON-like documents with dynamic schemas (MongoDB calls the format BSON).

Start mongodb

- Installation: http://www.mongodb.org/
- Installed in the lab

- Before issuing queries from Node application, need to start **database server**. Example: http://docs.mongodb.org/manual/tutorial/manage-mongodb-processes/

MongoDB example: connection

MongoDB is always
listening on port 27017

```
var databaseUrl = "localhost:27017/somedb";
var collections = ["users", "reports"];
```

NPM driver Library
to be installed

```
var db = require("mongojs").connect(databaseUrl, collections);
insertTestData();
performTestQuery ();
```

MongoDB example: **insertTestData**

```
db.users.save(
{email: "mgbarsky@server.com", password: "ggg", sex:
"female"},
 function(err, saved)
 {
        if(err){  console.log(err);  }
        if( !saved )
               {  console.log("User not saved");  }
        else
               {console.log("User 2 saved");
 }
);
```

Callback function – executed when save is complete

performTestQuery

```
db.users.find({sex: "female"},
function(err, users) {
        if( err || !users) console.log("No female users found");
        else users.forEach(
                function(femaleUser) {
                        console.log(femaleUser);
        } );
});
```