

Web Application Development

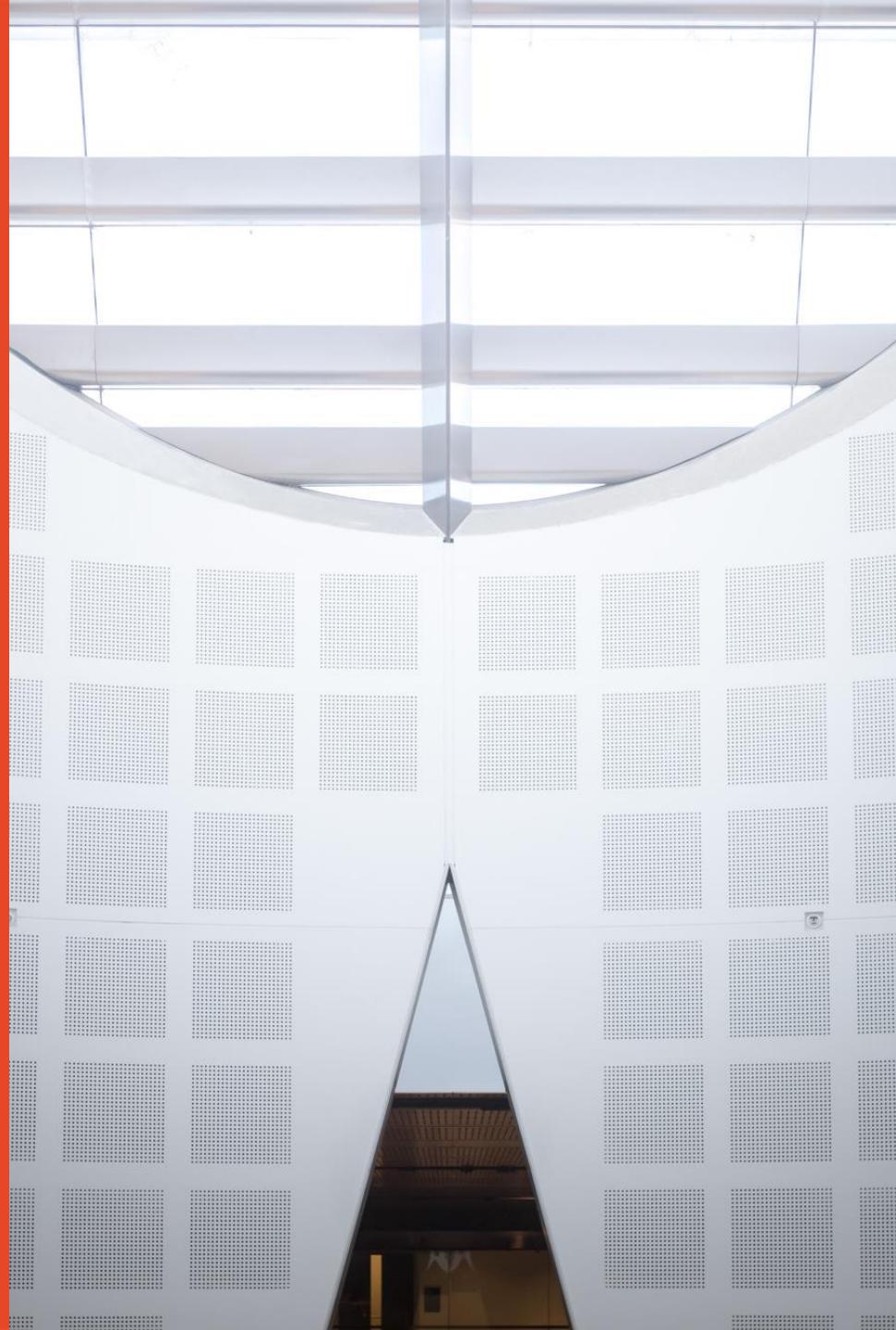
COMP4347

COMP5347

Server-side Development
Node.js

Week 5
Semester 1, 2025

Dr. Mehdi Nobakht
School of Computer Science



Outline

- **How web server works**
 - **Forms**
 - **HTTP GET, POST**
- **Node.js Execution**
- **Node.js Application Structure**
- **Express.js Basics**
 - **Routing**
 - **Middleware**
 - **Template Engines**

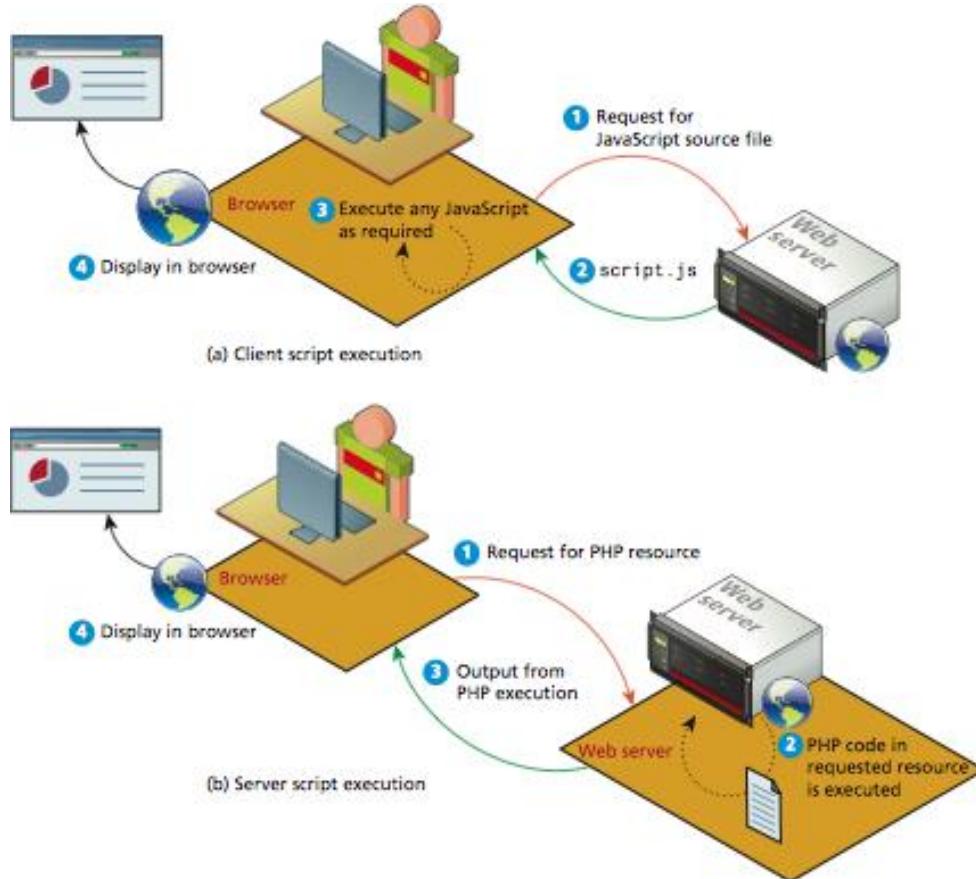
Server-side Development

- Server-side development involves the use of programming technology to create script that dynamically generate content

Execution of Client-side script (a) and server-side script (b)

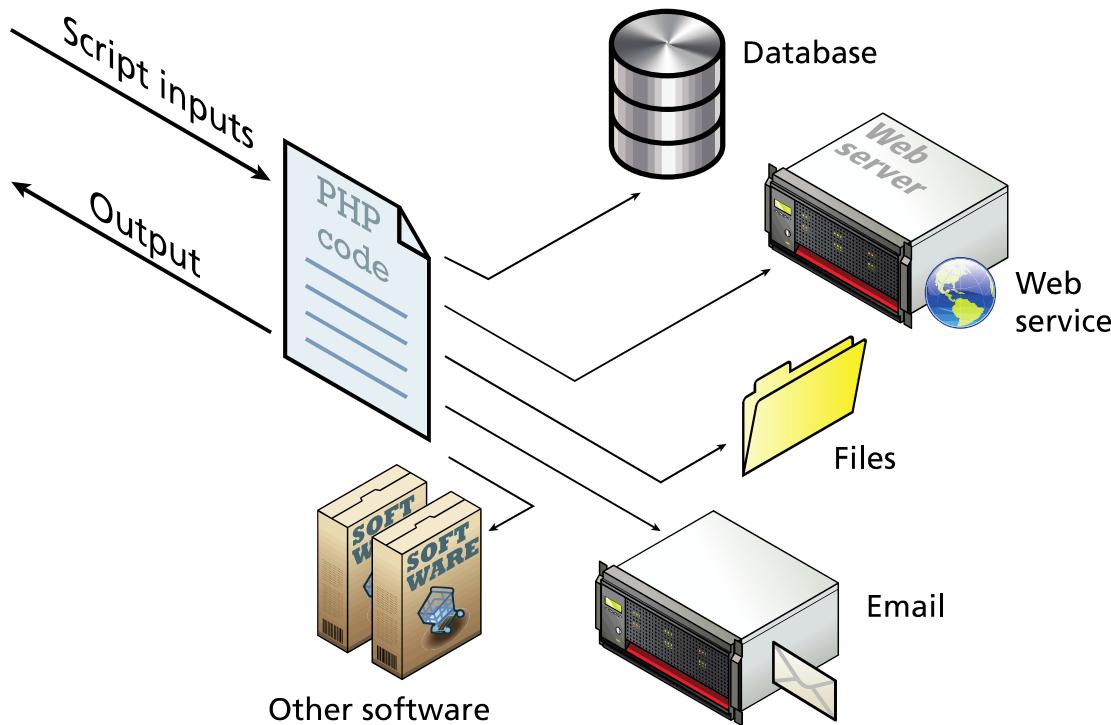
- Server-side development technologies:

- Java Server Pages
- Active Server Pages (.NET)
- PHP
- Node.js
- Ruby on Rails
- Python



Server-side Development

- Server-side script: software running on a server and uses HTTP request-response loop for interaction with the clients
- A server-side script can access resources made available by the server



Common Server Types

- Web Servers
 - A computer that runs a Web server software (e.g., Apache, MS IIS) and service HTTP requests
- Application Servers
 - A computer that hosts and execute Web applications developed in a certain technology; PHP, Ruby on Rails
- Database Server
 - A computer that is devoted for running a DBMS such as MySQL, SQL server
- Web servers must choose an **application stack** (OS, Web server software, database and scripting language for dynamic requests)
 - LAMP: Linux, Apache, MySQL and PHP
 - MEAN: MongoDB, Express.js, Angular.js, Node.js
- Server software should be installed and configured on the Server
 - N-tier architecture (physical/logical)

How server works - URLs

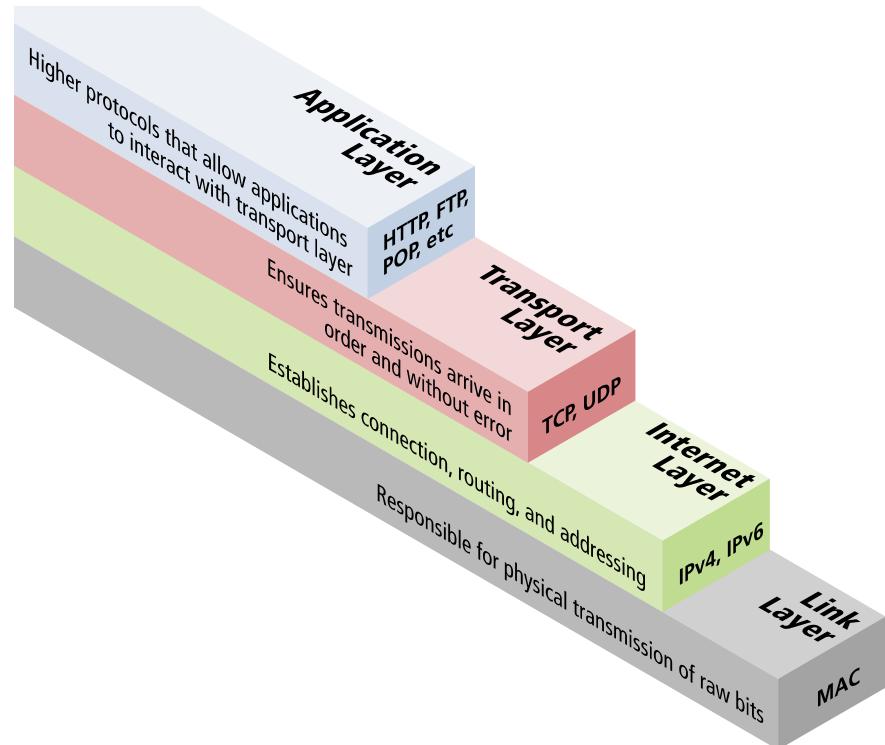
- Most basic web server acts like a file system while file requests are sent using HTTP protocol
 - Path: similar to a computer file system. The root of a web server correspond to a folder somewhere on the server.
 - Linux servers the path is /var/www/html

The diagram illustrates the structure of a URL by breaking it down into five components: Protocol, Domain, Path, Query String, and Fragment. The URL shown is `http://www.funwebdev.com/index.php?page=17#article`. Each component is underlined and labeled below the URL:

- Protocol*: `http`
- Domain*: `www.funwebdev.com`
- Path*: `/index.php`
- Query String*: `?page=17`
- Fragment*: `#article`

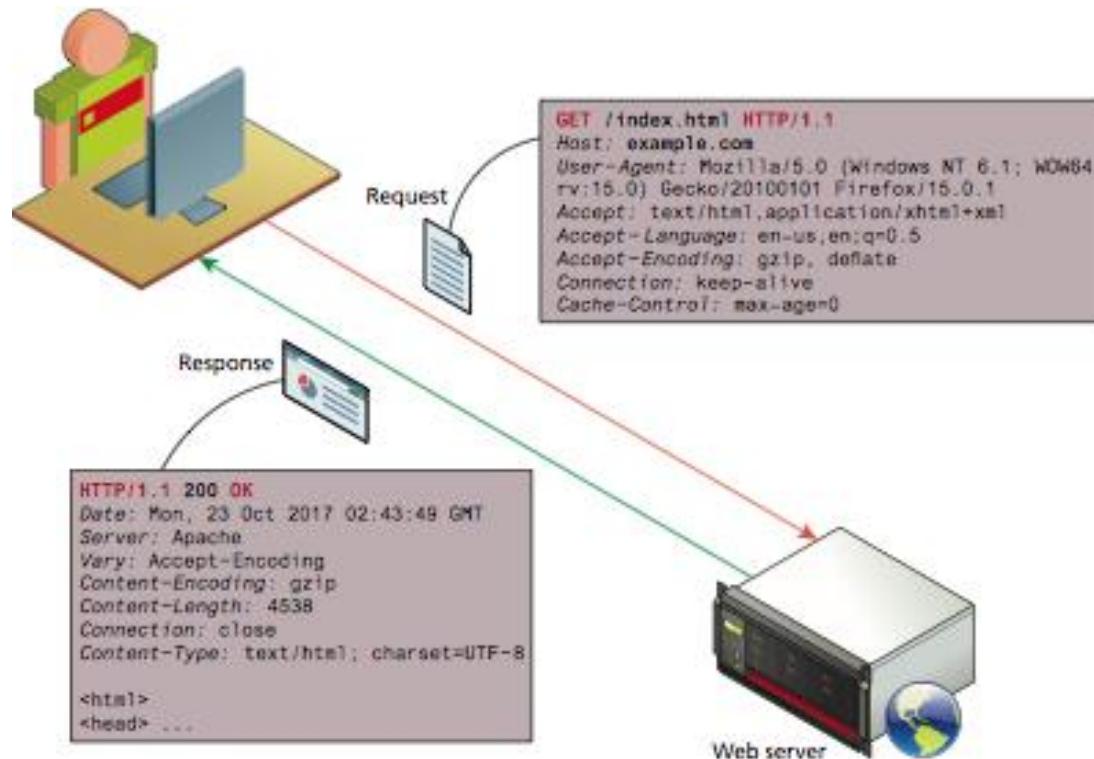
Internet Protocols – HTTP

- Transport layer protocol (e.g. TCP) provides logical communication between processes running on the *hosts identified by ip:port*
 - In particular, TCP ensures that transmissions arrive, in order, and without error
- Application layer protocol (e.g. HTTP) defines the *syntax and semantics of the message send between processes*



HTTP Headers

- **Request headers** include data about the client machine.
- **Response headers** have information about the server answering the request and the data being sent



Request/Response message

The screenshot shows the Network tab of a browser's developer tools. A blue box highlights the first item in the list: "www.apache.org". A yellow box labeled "HTTP request line" points to the URL "http://www.apache.org/" in the Headers tab. Another yellow box labeled "HTTP response status" points to the status code "200 OK". The Headers tab is active, displaying the following information:

Request URL: http://www.apache.org/
Request Method: GET
Status Code: 200 OK

Request Headers view source

```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-GB,en-US;q=0.8,en;q=0.6
Host: www.apache.org
If-Modified-Since: Tue, 28 Feb 2012 20:10:30 GMT
Proxy-Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 6.1) AppleWebKit/535.11 (KHTML, like Gecko) Chrome/17.0.963.56 Safari/535.11
```

Response Headers view source

```
Accept-Ranges: bytes
Age: 0
Cache-Control: max-age=3600
Connection: keep-alive
Content-Encoding: gzip
Content-Length: 8662
Content-Type: text/html; charset=utf-8
Date: Thu, 01 Mar 2012 03:55:00 GMT
ETag: "fb32ed-82f1-4ba24f8d2dbc0-gzip"
Expires: Thu, 01 Mar 2012 04:55:00 GMT
Last-Modified: Thu, 01 Mar 2012 02:11:03 GMT
Server: Apache/2.3.15-dev (Unix) mod_ssl/2.3.15-dev OpenSSL/1.0.0c
Vary: Accept-Encoding
Via: proxy-web-prd-2.ucc.usyd.edu.au, 1.0 www-cache.it.usyd.edu.au (squid/3.1.8)
X-Cache: MISS from www-cache.it.usyd.edu.au
```

Request Methods

HTTP/1.0

- GET
- POST
- HEAD
 - asks server to leave requested object out of response

HTTP/1.1

- GET, POST, HEAD
- PUT
 - uploads file in entity body to path specified in URL field
- DELETE
 - deletes file specified in the URL field

HTTP Response Status Code

- 3 digit response code
 - 1XX – informational
 - 2XX – success
 - 200 OK
 - 3XX – redirection
 - 301 Moved Permanently
 - 302 Found/Moved temporary
 - 303 Moved Temporarily
 - 304 Not Modified
 - 4XX – client error
 - 404 Not Found
 - 5XX – server error
 - 505 HTTP Version Not Supported

HTTP Request Processing

- Address Resolution
 - Simple example
 - A file stored under the absolute path
`/usr/usyd/yourlogin/lib/html/week2.html`
 - Is accessible through this URL:
<http://www.it.usyd.edu.au/~yourlogin/week2.html>
 - Apache administrator has set the mapping between URL path info and local file system path info
 - `/~yourlogin/week2.htm` is mapped to
`/usr/usyd/yourlogin/lib/html/week2.html` in local file system

Static and Dynamic Content

- Static content stored in local file system
 - Static content page: HTML page, plain text, image files, etc
 - Use the predefined mapping to locate the file
 - Construct HTTP response with header information
 - As-is page: static file containing complete HTTP responses
 - No response construction is required
 - Indicate is-as file (e.g., extension)
- Dynamic contents request explicit server side programmatic action to generate response
 - PHP, Perl scripts
 - Application server

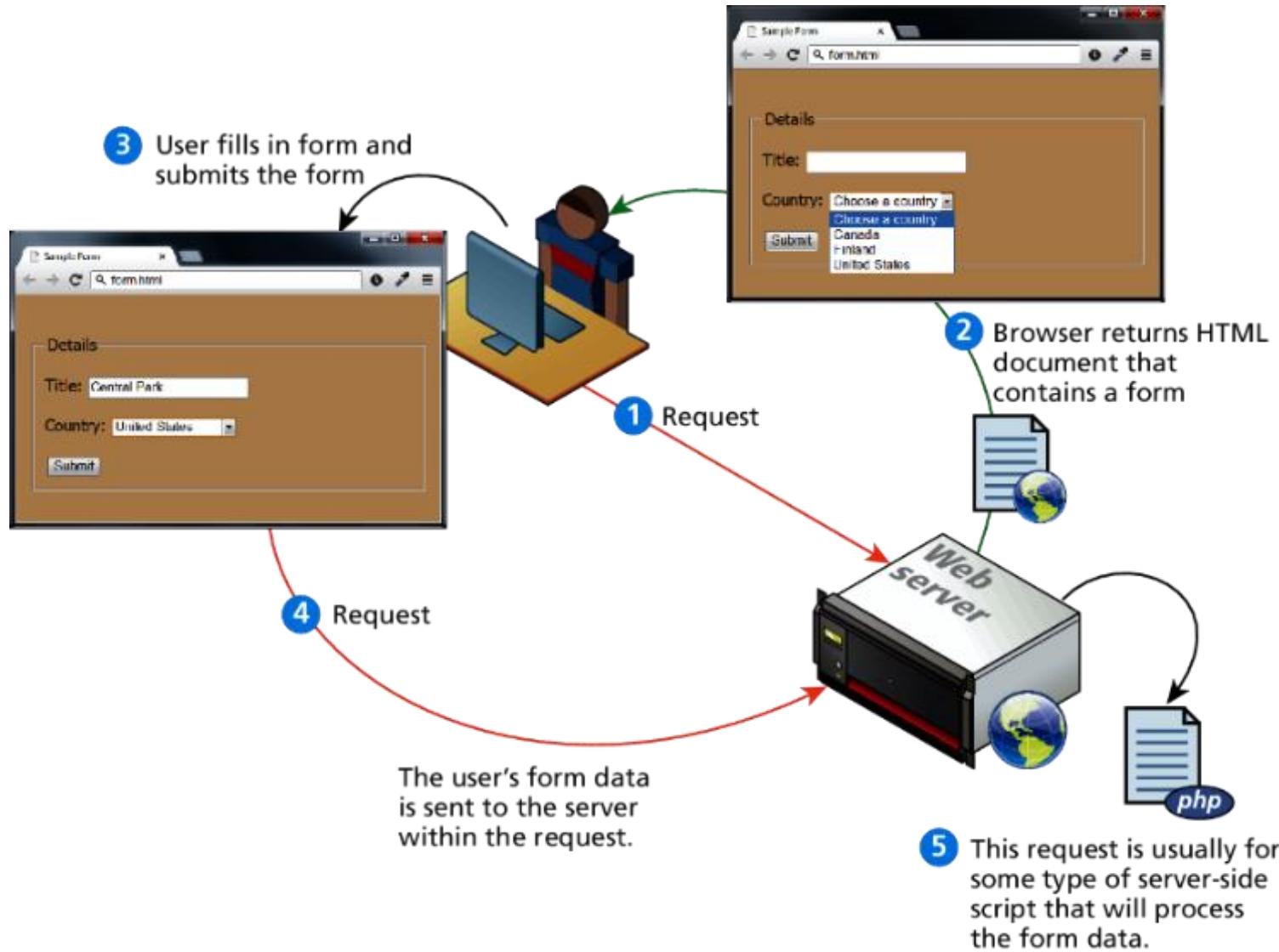
Server-Side Program Forms

- As a “script”
 - CGI scripts, or Java Servlets, or JavaScript
 - Just like a regular program
 - Parse request using regular expression
 - Write (HTML) response using stream operations provided by the language
- As a Server Page (template processing)
 - PHP, ASP, JSP
 - Program is structured around the returning text page
 - Script code are inserted into the HTML code to execute at certain points
- Combination based on MVC

Server-Side Program

- Basic processing steps
 - Parse HTTP request to obtain information carried in the request
 - Processing the HTTP request information
 - Generate response
- “script” style program regardless of the language chosen
- They are hidden in “server page” style program
- Supporting/external services for
 - networking
 - Common processing
- Supported by server (e.g., application server) or language framework

Sending data to server



HTML Forms – Query Strings

How the browser sends the data to the server

- Through HTTP requests
- The browser packages user's data into a query string
- Query string: a series of name=value pairs separated by &
 - HTML form element's name attribute
 - User input data

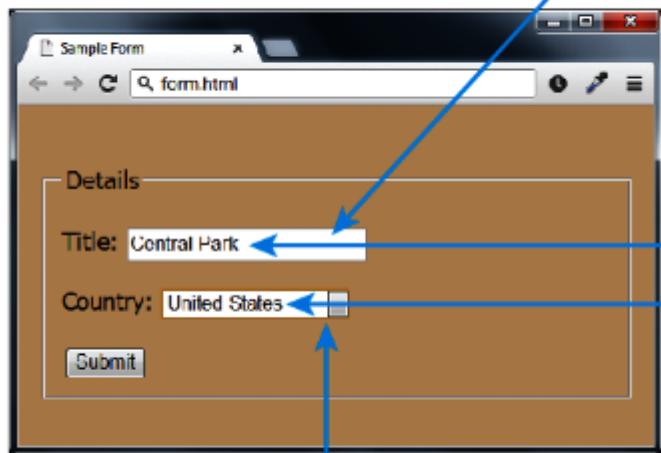
<form> Element

- There are two essential features of any form, namely the **action** and the **method** attributes
 - The **action** attribute specifies the URL of the server-side resource that will process the form data
 - The **method** attribute specifies HTTP request method
 - GET
 - POST
- GET and POST methods send form data to server in different ways

GET Method

- GET method attached the form data as query string to URL
- General format of query string
 - `http://example.com/over/there?title=test&where=here`

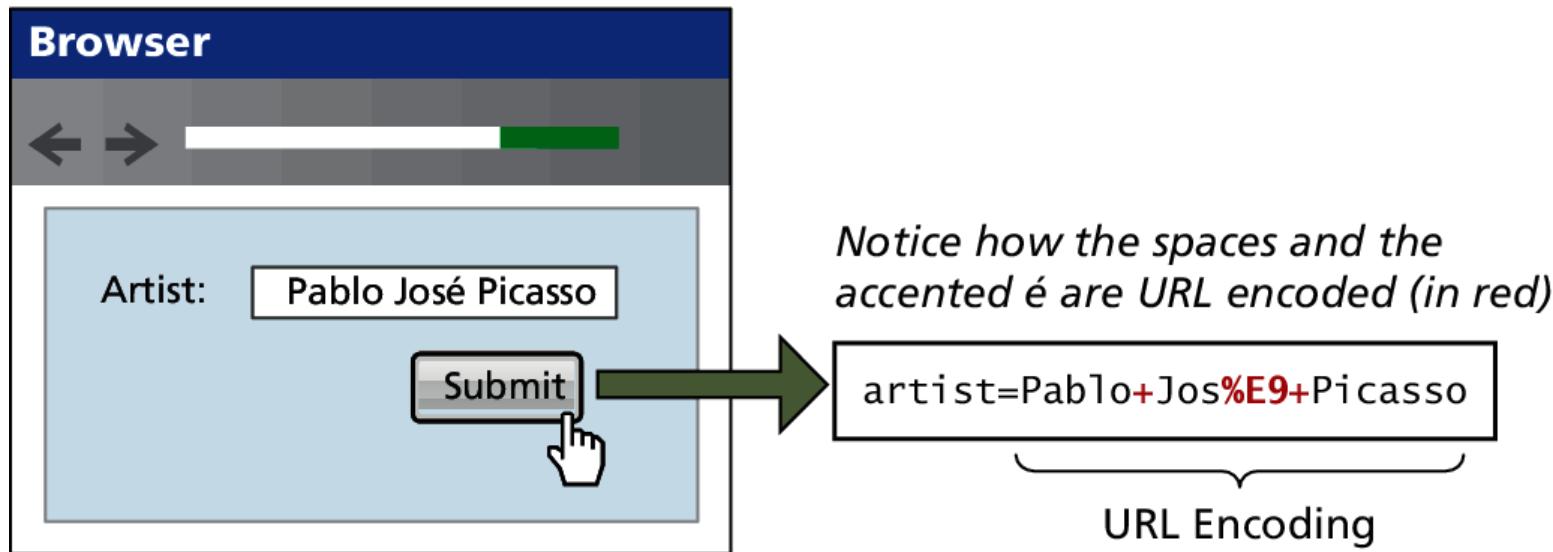
```
<input type="text" name="title" />
```



```
<select name="where">
```

`title=Central+Park&where=United+States`

URL Encoding



Which Value to Send

Select:

Second

- First
- Second
- Third

```
<select name="choices">
    <option>First</option>
    <option>Second</option>
    <option>Third</option>
</select>
```

?choices=Second

```
<select name="choices">
    <option value="1">First</option>
    <option value="2">Second</option>
    <option value="3">Third</option>
</select>
```

?choices=2

Radio Buttons and Checkboxes

Continent:

- North America
- South America
- Asia

```
<input type="radio" name="where" value="1">North America<br/>
<input type="radio" name="where" value="2" checked>South America<br/>
<input type="radio" name="where" value="3">Asia
```

I accept the software license

```
<label>I accept the software license</label>
<input type="checkbox" name="accept" >
```

Where would you like to visit?

- Canada
- France
- Germany

```
<label>Where would you like to visit? </label><br/>
<input type="checkbox" name="visit" value="canada">Canada<br/>
<input type="checkbox" name="visit" value="france">France<br/>
<input type="checkbox" name="visit" value="germany">Germany
```

```
?accept=on&visit=canada&visit=germany
```

POST Method

- POST method sends the form data as part of request body

Title
test

Description

Continent
North America ▾

Country
Canada ▾

City
Calgary

Copyright?
 All rights reserved
 Creative Commons

Creative Commons Types
 Attribution
 Noncommercial
 No Derivative Works
 Share Alike

I accept the software license

Rate this photo:
4

Color Collection:


Date Taken:
01/01/2017

Time Taken:
01:00 AM

POST request body

x Headers Preview Response Timing

▼ General

Request URL: file:///C:/Users/yzho8449/course/comp5347/2017/labs/code/week5-lecture/week3.html

▼ Request Headers

⚠ Provisional headers are shown

Content-Type: application/x-www-form-urlencoded

Origin: null

Upgrade-Insecure-Requests: 1

User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.87 Safari/537.36

▼ Form Data view source view URL encoded

title: test

description:

continent: North America

country: Canada

city: Calgary

copyright: 2

accept: on

x Headers Preview Response Timing

▼ General

Request URL: file:///C:/Users/yzho8449/course/comp5347/2017/labs/code/week5-lecture/week3.html

▼ Request Headers

⚠ Provisional headers are shown

Content-Type: application/x-www-form-urlencoded

Origin: null

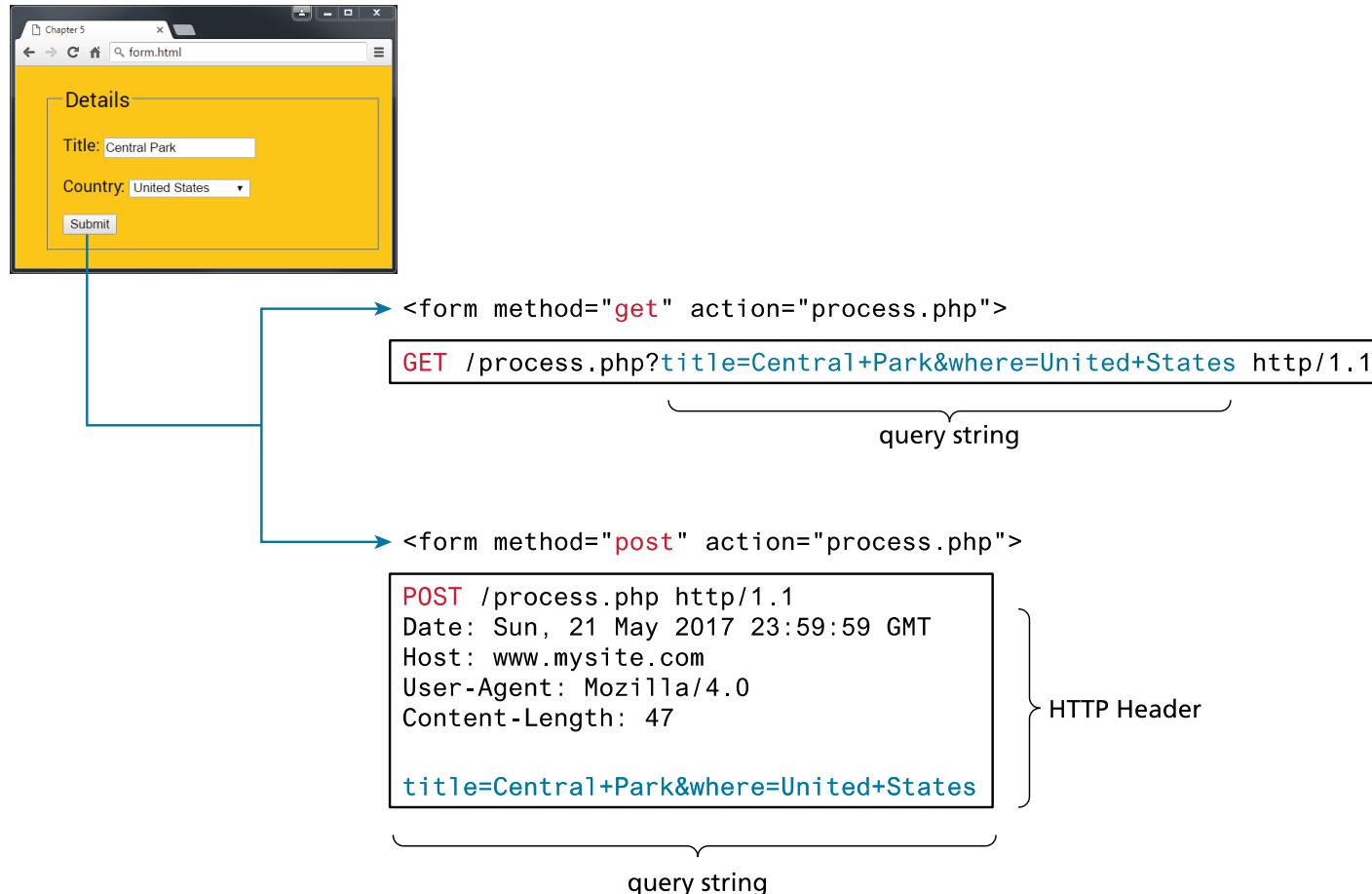
Upgrade-Insecure-Requests: 1

User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/56.0.2924.87 Safari/537.36

▼ Form Data view parsed

title=test&description=&continent=North+America&country=Canada&city=Calgary©right=2&accept=on&rate=4&color=%23800040&date=2017-01-01&time=01%3A00

GET vs POST Method



GET vs. POST

- GET
 - Explicit in the address bar
 - Browser's history and cache
 - Bookmarking
 - Limit number of characters
 - IE 2083 characters, Apache web server 4000 characters
- POST
 - Contain binary data
 - Hidden from user
 - Not cached, bookmarked or in history

GET vs. POST

- Implication
 - GET to **query** something (**without changing** any server data)
 - POST for sending data and **change something** on the server

Outline

- How web server works
 - Forms
 - HTTP GET, POST
- Node.js Execution
- Node.js Application Structure
- Express.js Basics
 - Routing
 - Middleware
 - Template Engines

Node.js Motivation

- Node.js design goal is *performance* (makes it popular)
- Node.js' server architecture is different
- Any web server is expected to handle a large number of user requests concurrently
 - In early days, each request is handled by a process
 - Most *modern* servers handle request at *thread* level
 - Each request is handled by a separate *thread*
 - Server usually maintains a thread pool
 - Thread is allocated fraction system resources
 - Max. concurrent thread number
- Node.js server runs on a single thread

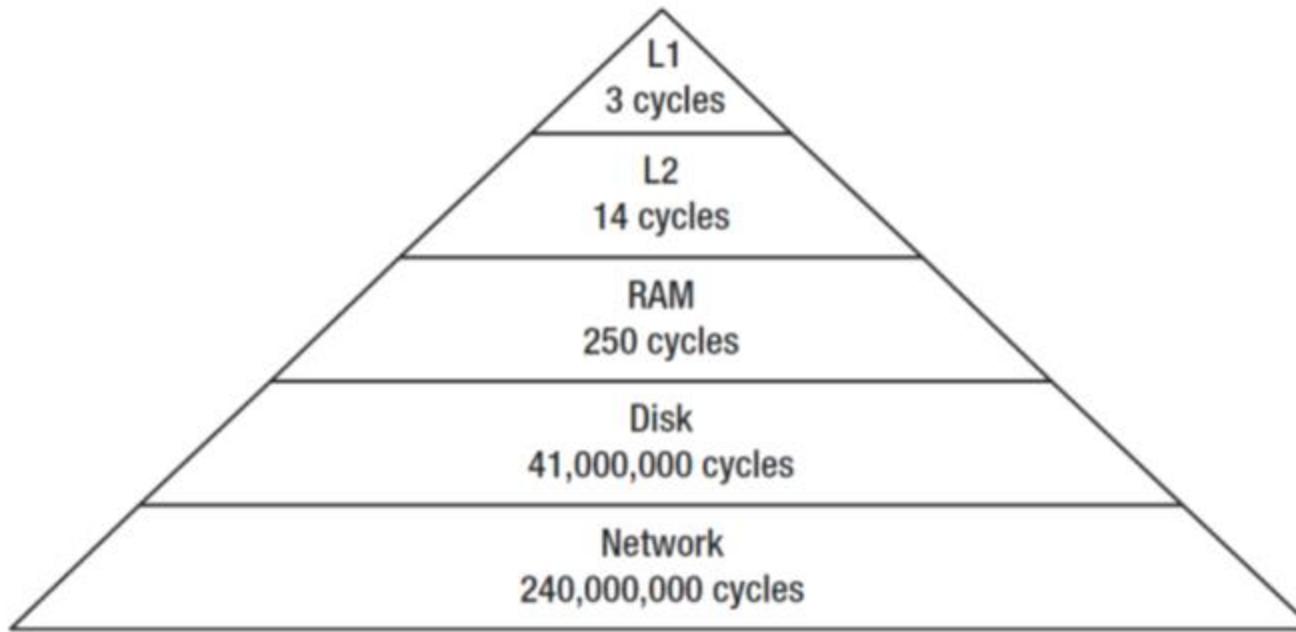
JavaScript

- JavaScript is synchronous, blocking and single-threaded language
 - One operation can be in progress at a time
 - Expensive database call?
 - Slow execution
- JavaScript code can be written to allow asynchronous behaviour
 - Asynchronous callbacks to invoke a callback function
 - Send request, wait for response while the rest of the code continue running
 - Results sent back for processing through event loop
 - *Event loop*: a process that constantly checking if there's anything required to be called

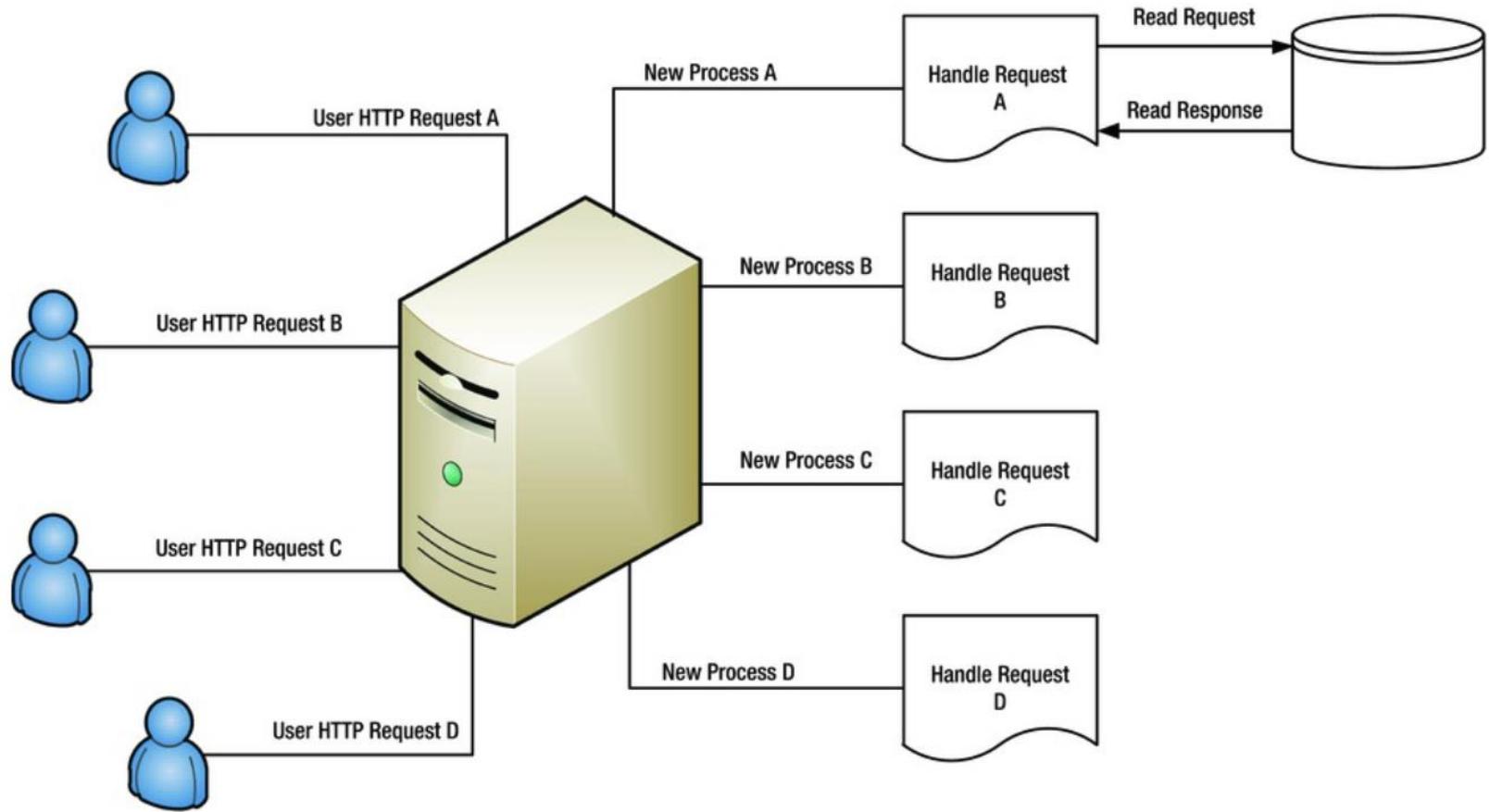
Single Threaded Execution

- Node.js server handles all requests in a single thread
- JavaScript is designed to run in a single thread, both in browser and on server side
 - *Event loop and callback for non-blocking, asynchronous execution*
 - Longer running processes: at least an event signaling the end of that process
 - Implement a callback function for that “end” event
 - E.g.
 - Disk IO done by OS’s file management component
 - Database call is executed mainly on database server
 - Downloading post request data (uses network resources)

The IO Scaling Problem

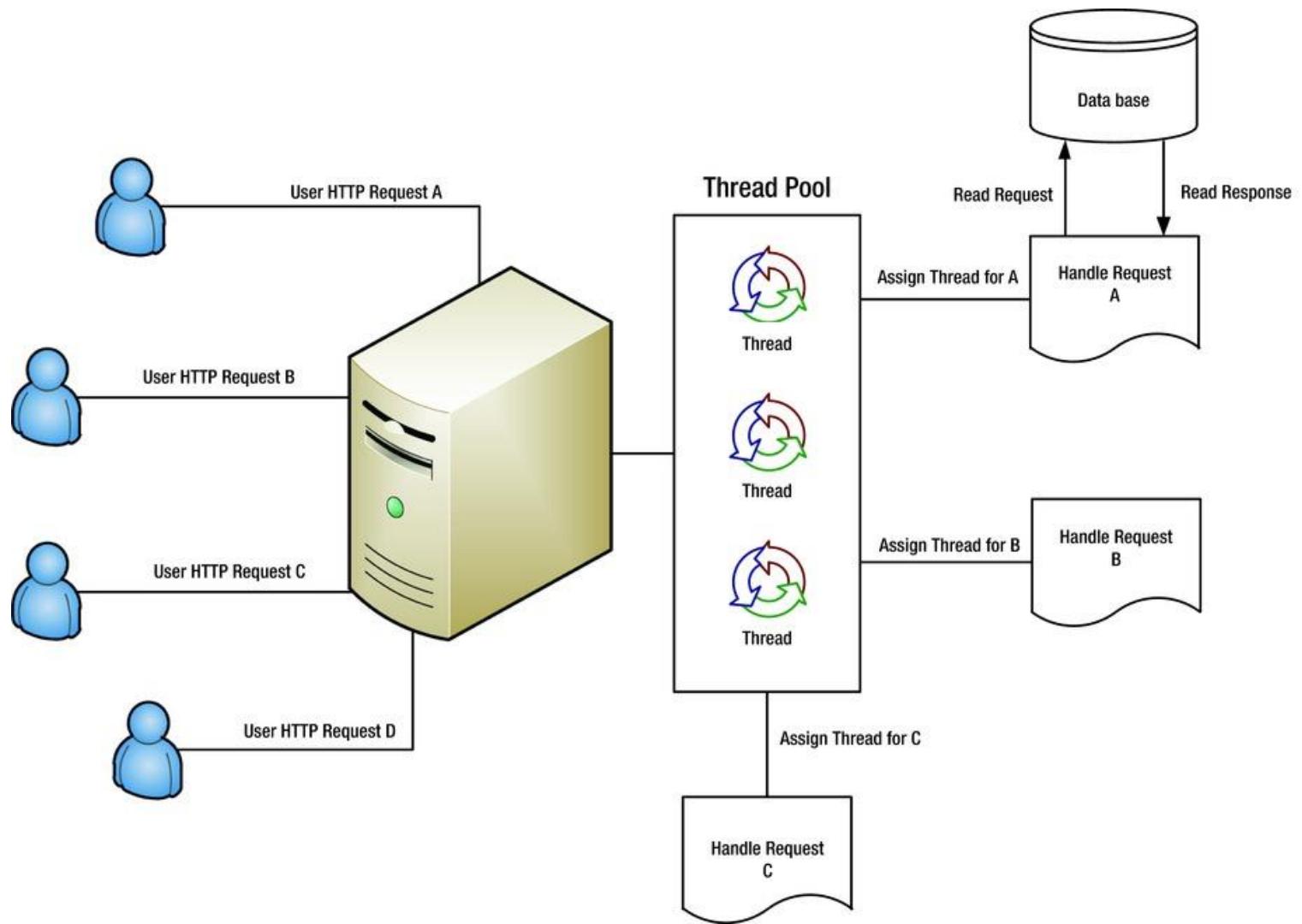


Traditional Server using Processes

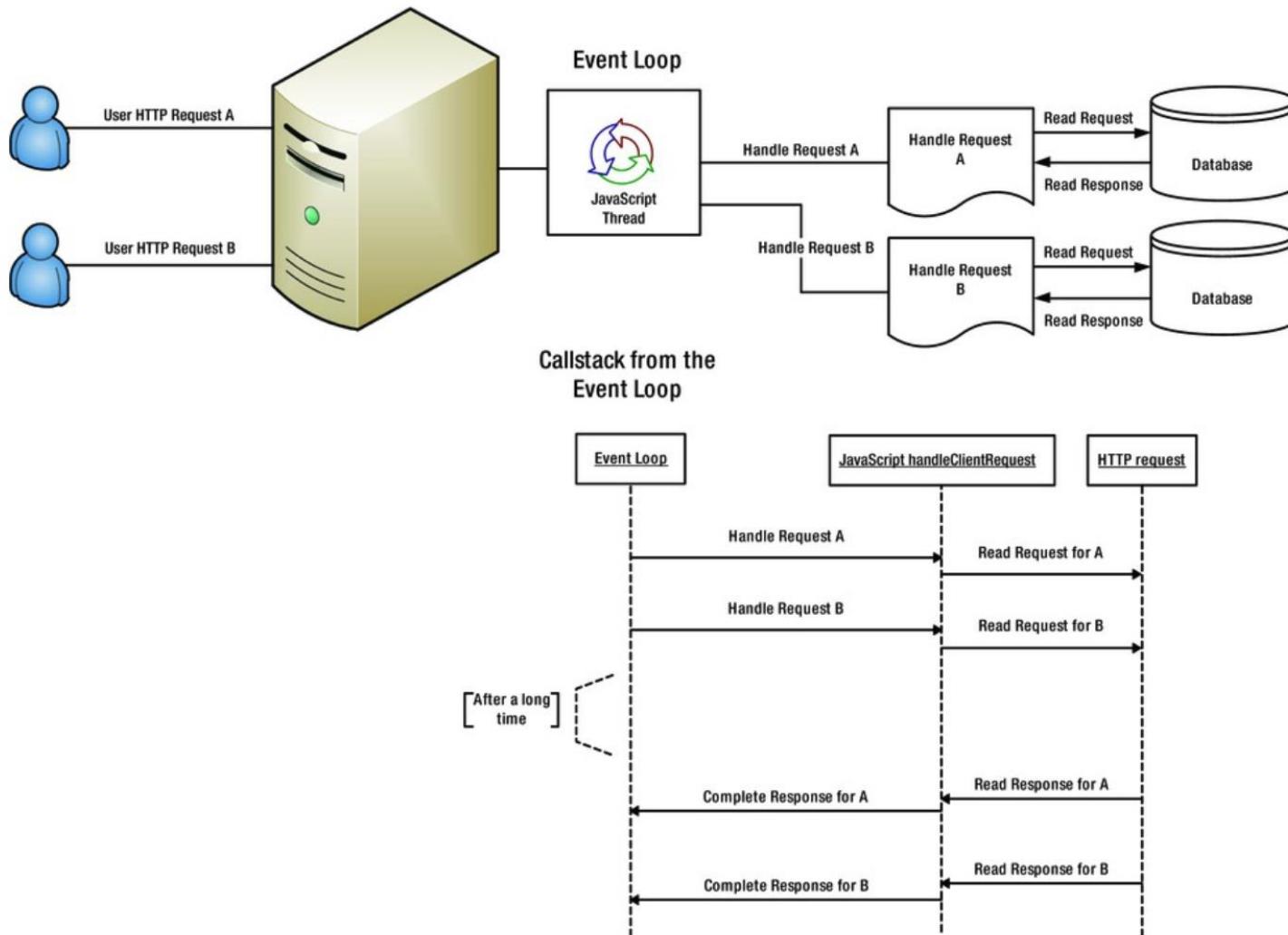


Basarat Ali Syed: Beginning Node.js, page 24

Servers with Multi-Threaded Execution

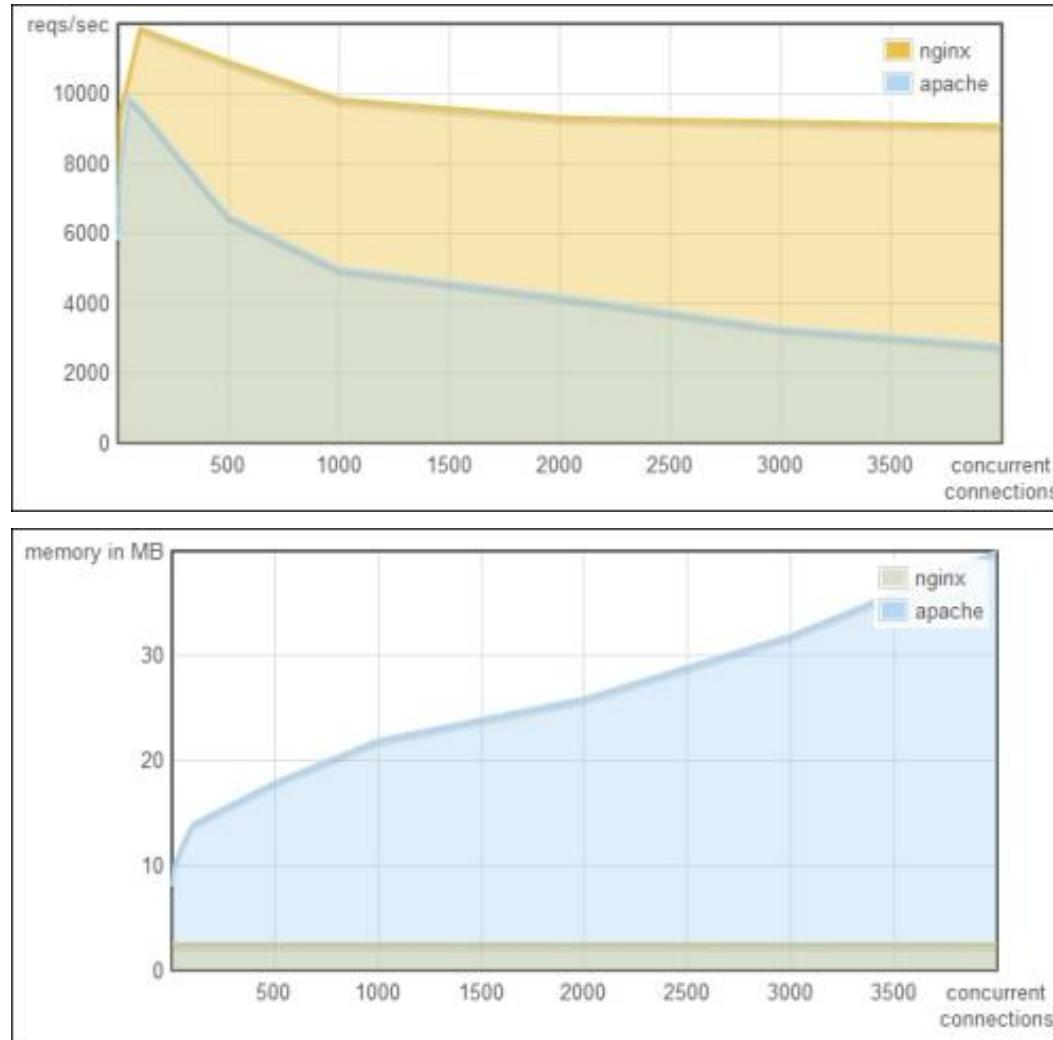


Asynchronous Request Handling

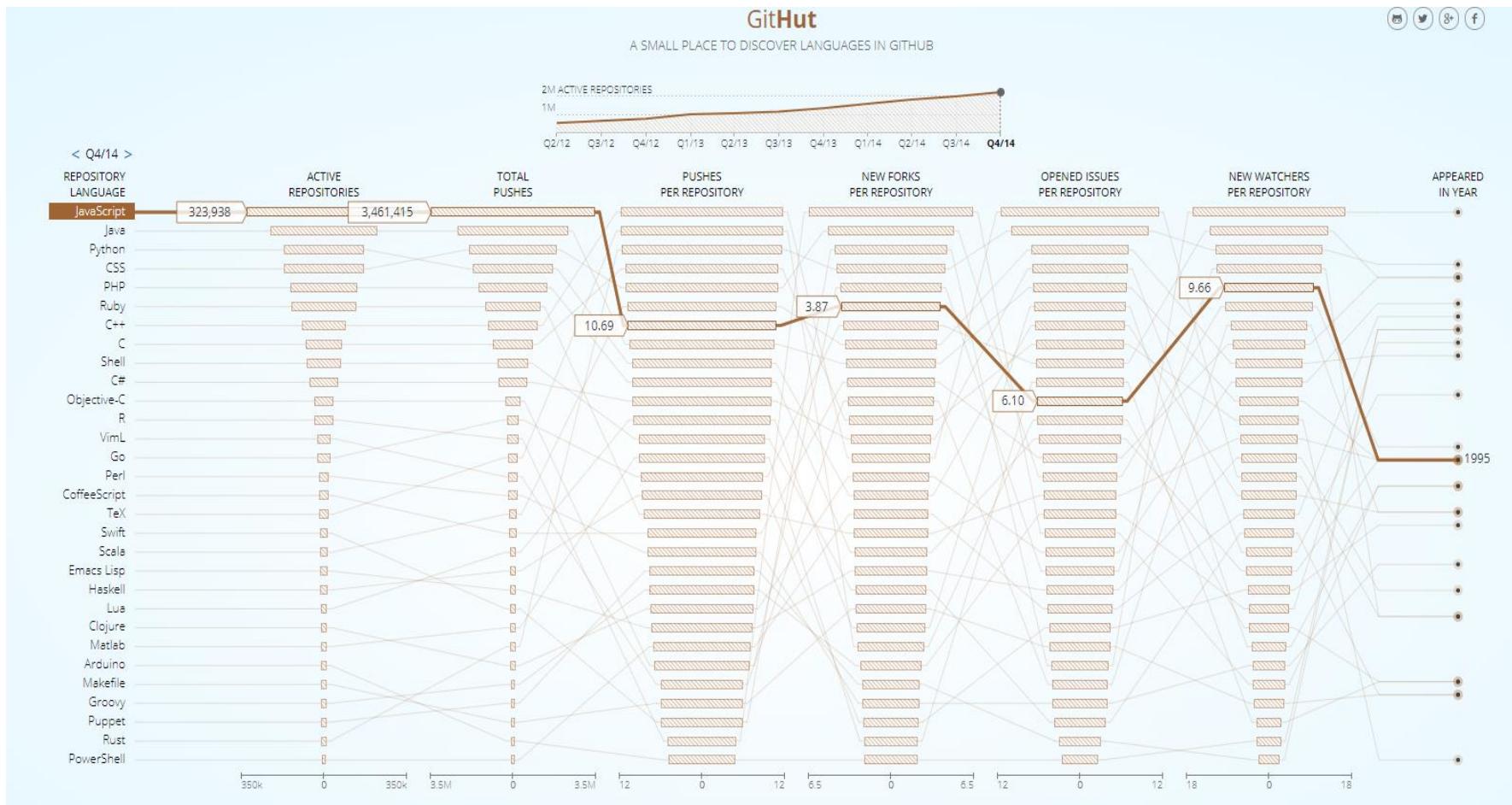


Basarat Ali Syed: Beginning Node.js, page 26

Single vs. Multi-threaded – Performance Comparison



Java Script – Popularity



<http://githut.info/>

Outline

- How web server works
 - Forms
 - HTTP GET, POST
- Node.js Execution
- **Node.js Application Structure**
- Express.js basics
 - Routing
 - Middleware
 - Template Engines

JavaScript – Global ‘things’

- Many APIs are globally defined
- Unlike (OO language), JS code has many variables and functions that do not belong to a class /object
 - These are ‘global’ variable or functions
 - Belong to a global object
- Browser’s global object window
 - variables, functions not belonging to some object/class belong to the `window` object and has global scope
 - `document` object’s fully qualified name is `window.document`
- In large application, with third party code or framework, there might be conflict in global namespaces

Module System

- Module system is a way to organize namespaces defined in various script code in large JavaScript applications
- Node.js uses file-based module system (CommonJS)
 - Each file is its own module
 - Each file has access to the current module definition using the *module variable*
 - The export of the current module is determined by the *module.exports variable*
 - To import a module, use the globally available *require* function

Node.js Basic Components

- **Node.js** has a few important globals
 - console
 - process
 - require
 - __filename and __dirname
- **Node.js** is shipped with a few core modules
 - **cli, fs, http**
 - List of modules in Node.js (<https://nodejs.org/api/modules.html>)
- Many other useful modules to be downloaded separately
 - Using NPM (Node Package Manager) - <https://www.npmjs.com/>

Simple Node.js Application

```
var http = require('http')
var server = http.createServer(function (request, response) {
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("Hello World");
  response.end();
});
server.listen(8888);
console.log("Web Server Running at http://localhost:8888 ... ");
```

Simple Node.js Application

```
var http = require('http')
var server = http.createServer(function (request, response) {
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("Hello World");
  response.end();
});
server.listen(8888);
console.log("Web Server Running at http://localhost:8888 ... ");
```

require is a global function to import module .

createServer is a higher order function taking another function as parameter. It returns a server object.

The anonymous function with parameter **request** and **response** describe the application logic of the webserver.

console is a global object

Handling Multiple Requests

```
var http = require('http')
var server = http.createServer(function(req, resp) {
  if (req.url == <some pattern>) {
    //write response
  }else if (req.url == <some other pattern>) {
    //write response
  }else if (req.url == <...>) {
    //write response
  }else{ // a not recognizable url
    // send back an error status code and
    // error message
  }
}) ;
server. listen(8888);
console.log("Web Server Running at http://localhost:8888 ... ");
```

All requests are handled by a single thread

Handling Query String

Import another module with url parsing utilities

```
var http = require('http'),
    url = require('url');

var server = http.createServer(function(req, res) {
  //targeting url like /sayHello?name=xxx
  if (req.url.indexOf('/sayHello') > -1) {
    var query = url.parse(req.url,true).query;

    res.end('<body><h2> Hello ' +
            + query.name +
            '</h2></body></html>');

  }else{ // a not recognizable url. Send back an error status
    // code and error message
  }});
server.listen(8888);
console.log("Web Server Running at http://localhost:8888 ... ");
```

Handling POST Request Data

```
var http = require('http'),  
    qs= require('querystring');  
  
var server = http.createServer(function(req, res) {  
    //targeting request like POST /sayHello with a body  
    //name:xxx  
    if (req.url.indexOf('/sayHello') > -1){  
        var body = '';  
  
        req.on('data', function(chunk){  
            body += chunk;  
        });  
  
        req.on('end', function(){  
            res.write('<body><h2> Hello '+  
                qs.parse(body).name +  
                '</h2></body></html>')  
        });  
    }else{ // a not recognizable url. Send back an error  
        // status code and error message  
    }});  
server.listen(8888);  
console.log("Web Server Running at  
http://localhost:8888 ... ");
```

Handling POST Request Data

Post request contains a body part. The body may contain simple form data or large file in a uploading request

The request body is accessed through **data** event, so we listen to it, and save the received data in a variable.

When there is no more data, an **end** event fires. We listen to it to and call a function to generate the response.

Both handlers, defined as anonymous functions are able to access the local variable **body** of the outer function. This is called closure.

```
var http = require('http'),  
    qs = require('querystring');  
  
var server = http.createServer(function(req, res) {  
  //targeting request like POST /sayHello with a body  
  //name:xxx  
  if (req.url.indexOf('/sayHello') > -1) {  
    var body = '';  
  
    req.on('data', function(chunk) {  
      body += chunk;  
    });  
  
    req.on('end', function() {  
      res.write('<body><h2> Hello ' +  
               qs.parse(body).name +  
               '</h2></body></html>')  
    });  
  } else{ // a not recognizable url. Send back an error  
    // status code and error message  
  });  
server.listen(8888);  
console.log("Web Server Running at  
http://localhost:8888 ... ");
```

Outline

- How web server works
 - Forms
 - HTTP GET, POST
- Node.js execution
- Node.js application structure
- Express.js basics
 - Routing
 - Middleware
 - Template Engines

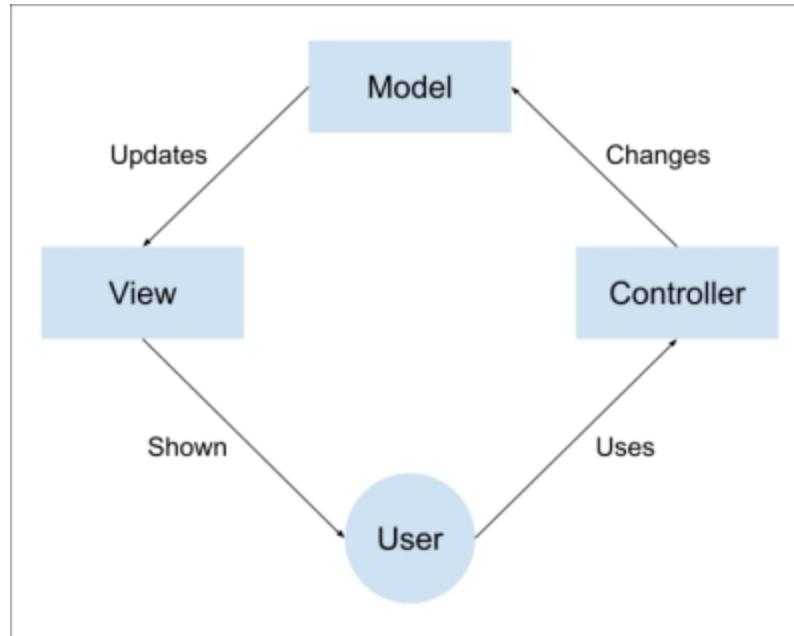
Web Application Structure

- The MVC pattern
 - What is the view technology (similar to early server page technology such as JSP)
 - *Template language* and template engines
 - A file that contains static and dynamic content
- Some framework to enable
 - URL / controller mapping
 - MVC wiring
 - Session management
 - Security management
 - ...
- Express is a popular framework for building MVC node.js application

Common MVC Architecture

Business or application logic implemented in a programming language

Usually a file with mixed static content (either in actual HTML or other mark down format) and processing logic and variables



Common processing such as network working, protocol handling usually provided by language framework, developers need to write code for customized processing

Express Framework

- Express is the framework that implements lots of common tasks when writing web app
 - Server setup
 - Manage the request response paradigm
 - Defines directory structure
 - Routing URL to code
 - Talk to template engine(s) to convert template files into proper response HTML
 - Remembering visitors for session support

Express.js – Popularity

Express

Home Getting started Guide API reference Advanced topics

Companies using Express in production

The screenshot shows a collection of company logos arranged in a grid, each representing a company that uses Express.js. The companies listed include:

- accenture
- DentiSphere
- EXOVE
- FOX SPORTS
- IBM
- icicle TECHNOLOGIES
- MuleSoft
- myntra
- nodeBB
- QuizUp
- ripjar
- RisingStack
- SPARKPOST
- UBER
- Yandex
- agrippa solutions
- CircleHD
- teachoo
- Taskade
- HASURA
- iLoveCoding
- kuali

<https://expressjs.com/en/resources/companies-using-express.html>

Express – Routing

- Routing: the mapping between HTTP request (method, url) to the piece of code handling the request
- Express routing take the following structure:
 - `app.METHOD(PATH,HANDLER)`
 - app is an instance of express
 - METHOD is an HTTP request method, in lowercase.
 - PATH is a path on the server.
 - HANDLER is the function executed when the route is matched.
- Example routing code

```
app.get('/', function (req, res) {  
  res.send('Hello World!')  
})
```

```
app.post('/', function (req, res) {  
  res.send('Got a POST request')  
})
```

<https://expressjs.com/en/starter/basic-routing.html>

Express – Question

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

var app = express()

app.get('/', function (req, res) {
    res.send('Hello World!')
})

app.listen(3000)
```

The `express()` function is a top-level function exported by the `express` module

The `app` object created by calling the `express()` and conventionally denotes the Express application

Q: What response the above application would return when the user types the following?
`http://localhost:3000>HelloWorld`

Express – Hello World Example

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

var app = express()

app.get('/', function (req, res) {
    res.send('Hello World!')
})

app.listen(3000)
```

The `express()` function is a top-level function exported by the `express` module

The `app` object created by calling the `express()` and conventionally denotes the Express application

Q: What response the above application would return when the user types the following?

`http://localhost:3000>HelloWorld`

A: the express server will responds with a “404 Not Found” as the server script (above) only respond to the root URL (/) or route

<https://expressjs.com/en/4x/api.html>

Express – app object

- The `app` object has methods for:
 - Routing HTTP requests; e.g., `app.METHOD`
 - Configuring middleware; `app.route`
 - Rendering HTML views; `app.render`
 - Registering a template engine
- The `app` object also has settings (properties) that affect how the application behaves
 - `app.mountpath` property contains one or more path patterns on which a sub-app was mounted

<https://expressjs.com/en/4x/api.html#app>

Middleware

- Middleware is a software (function) sits between the application code and some low-level API
- It has access to the request and response object
- It is used to implement common tasks on the request or response objects
 - In Java web dev. middleware is like filter where you can stack a lot of them before/after servlet
- The overall process would be
 - Client sends request
 - Request arrives at server
 - Handled by middleware 1, passed to *next function*
 - Handled by middleware 2, passed to *next function*
 - route method, passed to *next function*
 - Middleware 3, stop here

<https://expressjs.com/en/guide/writing-middleware.html>

Middleware stack

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

app.get('/', function(req, res, next) {
  next();
})

app.listen(3000);
```

HTTP method for which the middleware function applies

Path (route) for which the middleware function applies.

The middleware function.

Callback argument to the middleware function

HTTP response argument to the middleware function

HTTP request argument to the middleware function

Route method is also a middleware, most of the time we call it *route method* or *handler*. It can use the `next()` function to pass the control to a middleware running after the response is sent out

Middleware

Middle functions can do the following:

- Execute any code
- Make changes to request or response object
- End the request-response cycle
- Call next in the stack, middleware or route method

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

var myLogger = function (req, res, next) { // A middleware function
    console.log('LOGGED')
    next() // Call next() so the request-response cycle does not stop here
}
app.use(myLogger) // It will be called before the route method

app.get('/', function (req, res) {

    res.send('Hello World!')
})
app.listen(3000)
```

Middleware Control Flow

- `app.use()` mounts a middleware function(s) at the specified path (the default path is “`/`”)
 - `app.use([path,] callback [,callback...])`
- In the previous example, we used `app.use()` to call the middleware `MyLogger` before the route method/handler
- The order of middleware loading is important: middleware functions that are loaded first are also executed first
- Middleware functions are executed sequentially (in the order they are declared)
- A middleware function that does not invoke a `next()` function will end the request-response cycle
- If the current middleware function does not end the request-response cycle, it must call the `next()` function, otherwise the request will be left hanging

Middleware – Exercise

What would be the output of the following code when the following request is sent?

`http://localhost:3000/`

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

var myLogger = function (req, res, next) {
    console.log('LOGGED')
    next()
}
app.get('/', function (req, res) {

    res.send('Hello World!')
})
app.use(myLogger)
app.listen(3000)
```

Middleware – Exercise

What would be the output of the following code when the following request is sent?

`http://localhost:3000/`

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

var myLogger = function (req, res, next) {
    console.log('LOGGED')
    next()
}

app.get('/', function (req, res) {
    res.send('Hello World!')
})

app.use(myLogger)  The order of middleware loading is important
app.listen(3000)
```

This middleware function will not be called as it was loaded after the route method/handler. So, "LOGGED" will not be printed to the console

This route handler terminates the request-response cycle as it does not pass control to the next function

Useful Middlewares

- Middlewares are commonly used for tasks like
 - Parsing POST body
 - Parsing cookies
 - Many others
- Many third party middlewares for common processing
 - E.g. body-parser module consists of a few middleware functions for parsing form data

<https://expressjs.com/en/guide/using-middleware.html>

Template Engine

- Template engine represents the view technology
 - Defines a template format to blend static content with processing logic
 - Translate the template file into proper HTML file
 - Running processing logic
 - Replacing variable with values
- Many template engines that can work with Express
 - EJS
 - PUG (previously called JADE)
 - Mustache
- As expected, the oldest/simplest template engine (EJS) support the format of embedding JavaScript code in HTML

EJS – Embedded JavaScript Template

- JavaScript code can be embedded nearly everywhere in an HTML file
 - The convention is very similar to early JSP
 - The control flow should be embedded with <% %>
 - Escaped output (expression) should be embedded with <%= %>
 - And others
- Example EJS

```
<% if (user) { %>
  <h2><%= user.name %></h2>
<% } %>
```

<https://www.npmjs.com/package/ejs>

Simple Example – The Form

```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Greetings Input</title>
</head>
<body>

<form method="GET" action = "greeting">
Please type in your name: <input type = "text" name="name" ></input>
<input type = "submit" value = "submit"></input>
</form>

</body>
</html>
```

Greetingform.ejs

A screenshot of a web browser window. The address bar shows 'localhost:3000'. The page content is a form with the text 'Please type in your name:' followed by a text input field containing 'Joe'. To the right of the input field is a 'submit' button. A red arrow points to the 'submit' button.

Simple Example – Express App.

```
var express = require('express')
var path = require('path')

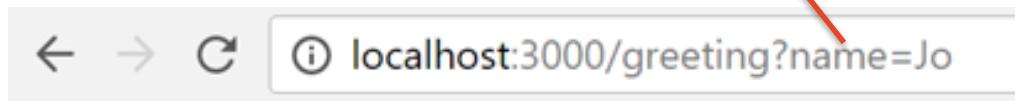
var app = express()
app.set('views', path.join(__dirname, 'views'));
app.get('/', function(req,res){
    res.render('greetingform.ejs')
});
app.get('/greeting', function(req,res){
    name=req.query.name
    res.render('greeting.ejs', {name:name})
});

app.listen(3000, function () {
  console.log('greeting app listening on port 3000!')
})
```

By using use '`__dirname`', we make sure that the path is always relative to the current file instead of the current working directory (CWD).

The CWD may be different from the file directory if we run our application from another directory such as from one level up using '`node static/greeting.js`' instead of the same directory—that is, '`node 1 basic.js`'. Relative path names such as '`./public`' resolve relative to the CWD.

Using '`__dirname`', making it Independent of the CWD.



Simple Example – the response

```
<!DOCTYPE html>
<html>
  <head>
    <title>Customized Greeting!</title>
  </head>
  <body>
    Welcome <%= name %>
  </body>
</html>
```

greeting.ejs



Welcome Jo

Pug Template Engine

- Influenced by *Haml* and implemented with JavaScript for Node.js and browsers
- Pug defines simple rules for writing static HTML content together with processing logic
 - HTML tags
 - Attributes
 - Code and Control Structure
 - Interpolation

<https://www.npmjs.com/package/pug>

Pug – HTML Tags

- By default, text at the start of a line (or after only white space) represents an html tag
- Indented tags are nested, creating the tree like structure of html.

```
ul
```

```
  li Item A  
  li Item B  
  li Item C
```

```
<ul>
```

```
  <li>Item A</li>  
  <li>Item B</li>  
  <li>Item C</li>
```

```
</ul>
```

Pug – Attributes

- Attributes
 - Tag attributes look similar to html (with optional comma), but their values are just regular JavaScript.

```
a(href='google.com') Google  
|  
a(class='button' href='google.com') Google  
|  
a(class='button', href='google.com') Google
```

Using Pug

```
<a href="google.com">Google</a>  
<a class="button" href="google.com">Google</a>  
<a class="button" href="google.com">Google</a>
```

Using HTML

Pug - Simple Control

- Any JavaScript code can be included with a leading '-' character
- There are also first class conditional and iteration syntax

```
- for (var x = 0; x < 3; x++)  
  li item
```

```
<li>item</li>  
<li>item</li>  
<li>item</li>
```

```
- var user = {description:'foo bar baz'}  
- var authorised = false  
div#user  
  if user.description  
    h2.green Description  
    p.description= user.description  
  else if authorised  
    h2.blue Description  
    p.description.  
      User has no description,  
      why not add one...  
  else  
    h2.red Description  
    p.description User has no description
```

```
<div id="user">  
  <h2  
  class="green">Description</h2>  
  <p class="description">foo  
  bar baz</p>  
</div>
```

Pug – Interpolation

- JavaScript variables and expression can be included in various ways

```
- var title = "On Dogs: Man's Best Friend";
- var author = "enlore";
- var theGreat = "<span>escape!</span>";
- var unescaped = "The tag <em> names </em> stays";

h1= title
p Written with love by #{author}
p This will be safe: #{theGreat}
P This is the unescaped example: !{unescaped}
```

```
<h1>On Dogs: Man's Best Friend</h1>
<p>Written with love by enlore</p>
<p>This will be safe: &lt;span&gt;escape!&lt;/span&gt;</p>
<p>This is the unescpaed example: The tag <em> names </em> stays</p>
```

Pug – Interpolation

- Inline tags
 - Inline tags, as opposite to block styled tags, do not start a new line.
 - PUG provides an easy way of writing them to avoid unnecessary indent

p

This is a very long and boring paragraph that spans multiple lines. Suddenly there is a `#[strong` strongly worded phrase] that cannot be `#[em` ignored].

p

And here's an example of an interpolated tag with an attribute:`#[q(lang="es")` ¡Hola Mundo!]

```
<p>This is a very long and boring paragraph that spans multiple lines.  
Suddenly there is a <strong>strongly worded phrase</strong> that cannot  
be <em>ignored</em>.</p>
```

```
<p>And here's an example of an interpolated tag with an attribute: <q  
lang="es">¡Hola Mundo!</q></p>
```

Resources

- Randy Connolly, Ricardo Hoar, *Fundamentals of Web Development, Global Edition*, Pearson
- Rauch, Guillermo 2012, *Smashing Node.js: JavaScript Everywhere*,
 - e-book, accessible from USYD library, Chapter 7 HTTP
- Basarat Ali Syed 2014, *Beginning Node.js*
 - E-book, accessible from USYD library, Chapter 2 and 3
- Haviv, Amos Q, *MEAN Web Development*
 - E-book, accessible from USYD library, Chapter 2 and 3

W5 Tutorial: Node/Express Application

**Week 6: MVC and
Session Management
NoSQL Database**



THE UNIVERSITY OF
SYDNEY

