NODEJS

Notes by Prof Nath

University of Colorado
Boulder

COMMUNICATION ACROSS THE STACK

Table
User interface

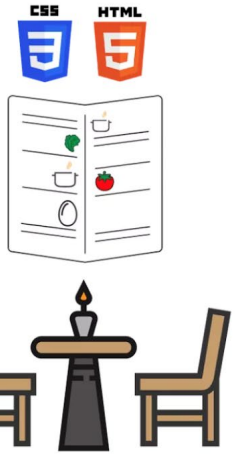
=

http://mysite.com
User interface

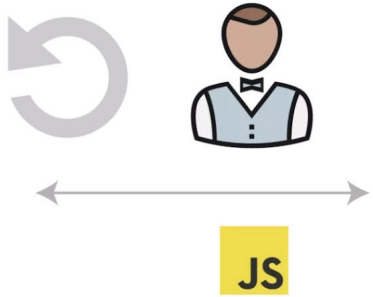
Menu
Website Content

=

HTML + CSS
Code for Static Content

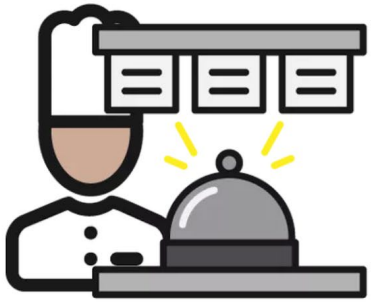


Choosing a Meal
User Experience



JS

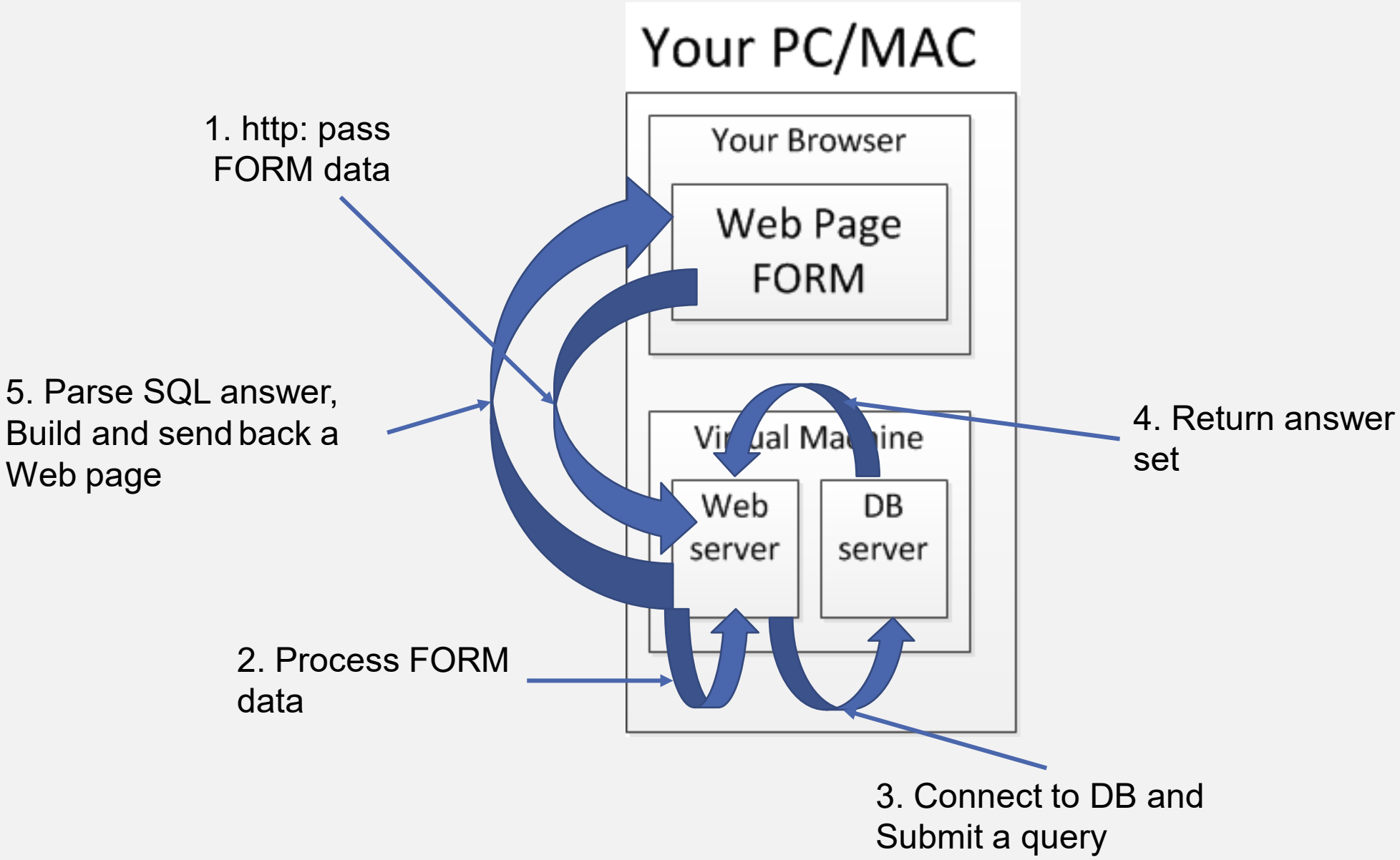
Interactivity/User Requests
JavaScript



Kitchen
Back-end

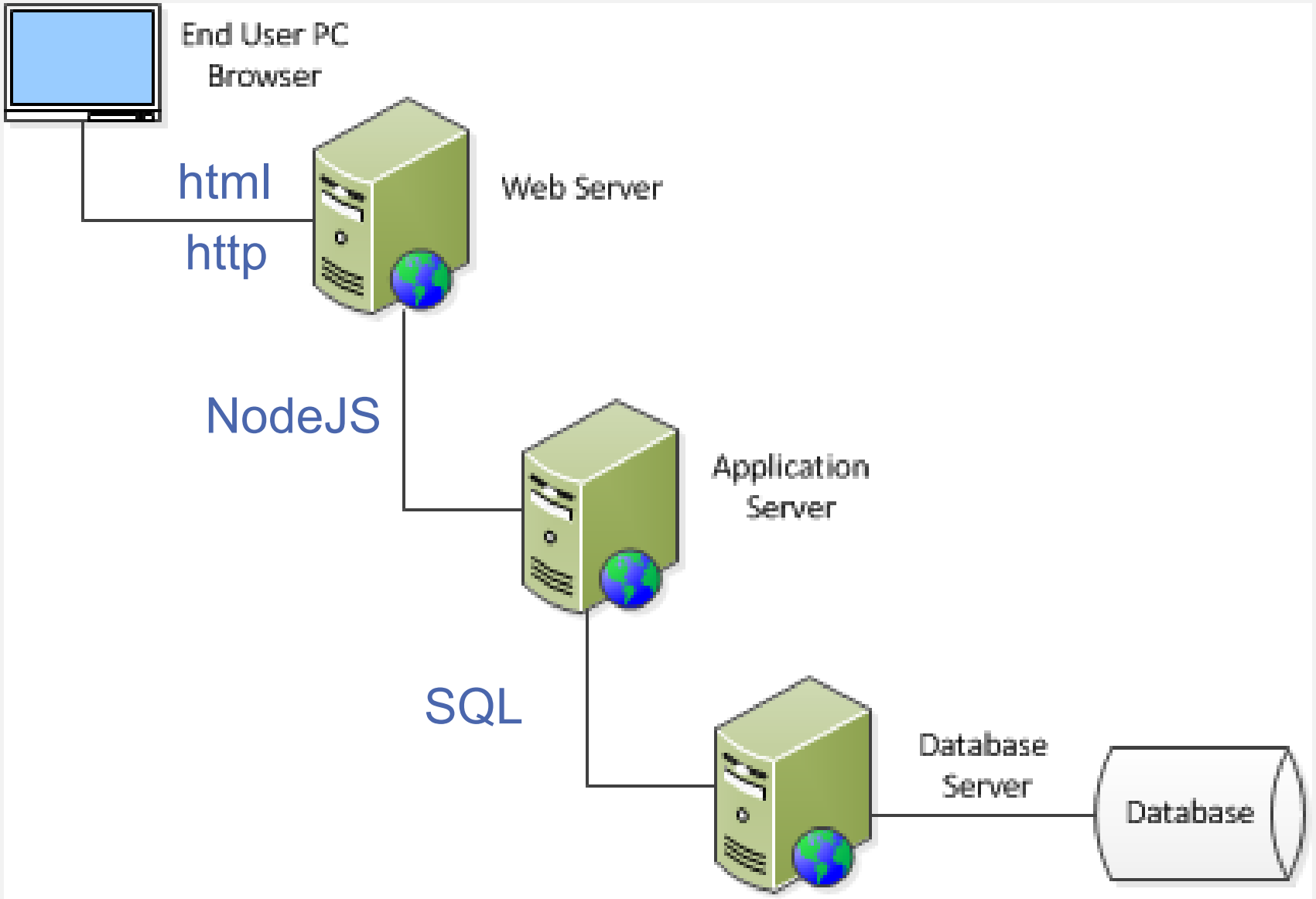
=

Server
Back-end

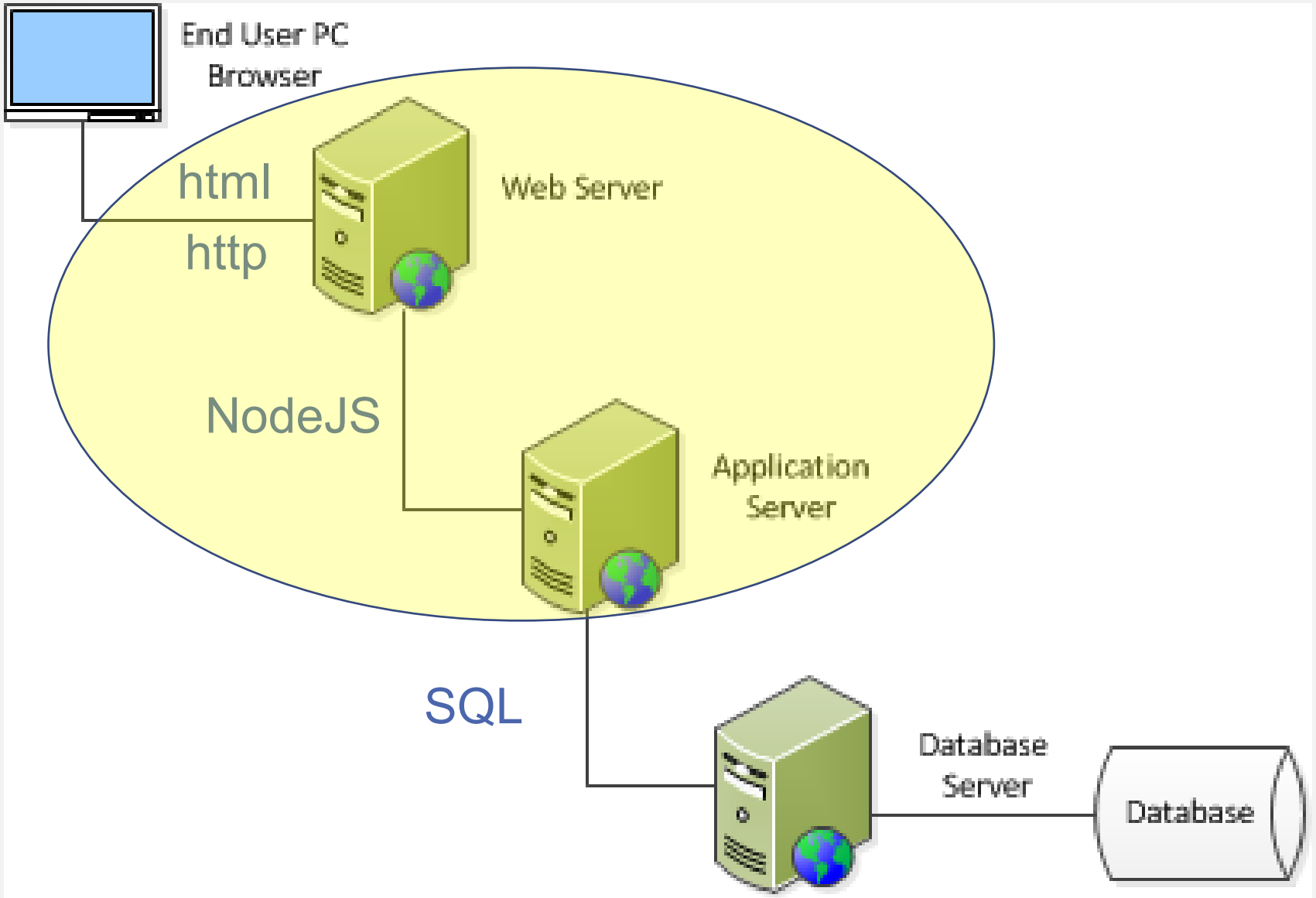

Your PC/MAC
Your Browser
Web Page
FORM
Virtual Machine
Web
server
DB
server
1. http: pass
FORM data
2. Process FORM
data
3. Connect to DB and
Submit a query
4. Return answer
set
5. Parse SQL answer,
Build and send back a
Web page

COMPONENTS OF STACK WE'RE LEARNING

- Three components:
  - **HTML** – Takes a marked-up file and renders it in the bowser on the client's PC
  - **NodeJS** – JS is the server-side scripting language. JS runs on a local web server running under NodeJS
  - **SQL** – Communicates with the database server. A DBMS runs on a database server.



PostgreSQL DBMS



PostgreSQL DBMS

NODE JS BASICS

NodeJS is a multi-purpose server-side processing engine.

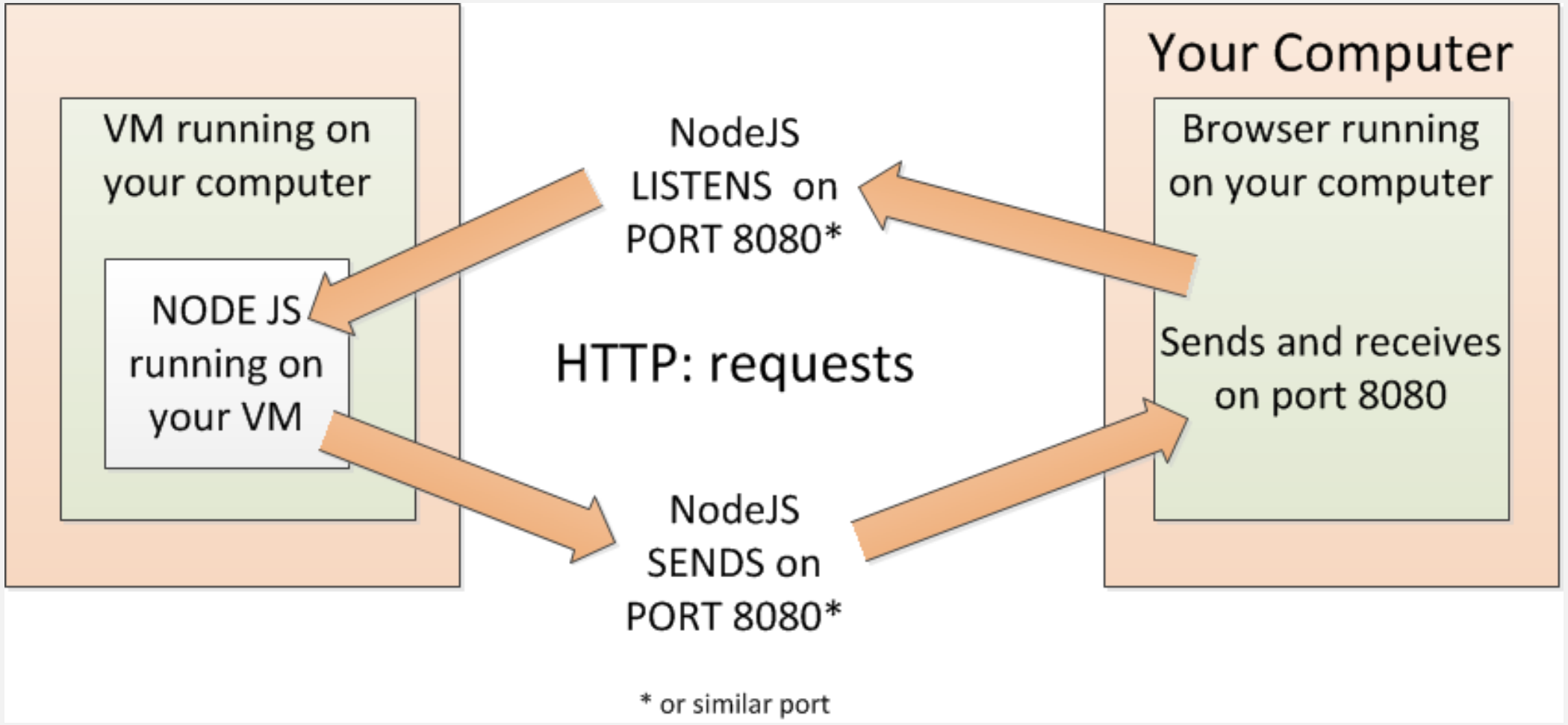
- It is Open-Source (GPL – Gnu Public License)
- It is FREE
- It runs anywhere (Windows, Linux, Unix, Mac OS X)
- It uses the Java Script programming language – the "default" language for most web-based applications.
- It looks good on your resume.

NODE JS BASICS

NodeJS can process HTTP requests from your browser:

- Node.js code can generate dynamic page content – it *creates* the HTML on the fly
- Node.js code can create, open, read, write, delete, and close files on the server
- Node.js code can collect and process form data from an HTML page
- Node.js code can read, add, change, delete data in your database

NODE.JS ARCHITECTURE



HTML FORMS

- How are they used?
  - Use the browser's window as a data entry screen
  - Collect information from the user
  - Pass it to the web server via http
  - Invoke a server-side script
  - Passes ***form data*** as input to the script
  - Script on server parses out the form data

HTML FORMS

- `<form>` tag has several attributes – two are required
- ACTION
  - `<form action="http://URL">` name of a program on the web server
    - URL specifies the location of the executable file on the web server
  - `<form action="mailto:mailrecipient">` sends an email
- METHOD
  - `<form method="POST" >` or `<form method="GET">`
    - **POST** when you have large amount of data being sent, encryption available, a two-step process
    - **GET** for small amounts, no security – all in one step

      `<form enctype=`
      - multipart/form-data (default)
      - text/plain (used only for mailto)

NODEJS CONCEPTS

- NodeJS programs handle HTTP: calls from the client
  - Two types of calls:
    - req – the HTTP: request coming from the client (properties for the request query string, parameters, body, HTTP headers)
    - res – the HTTP: response being sent back to the client

ROUTING

- Routing determines the way in which the NodeJS application responds to a client request to an endpoint.
- Defined by blocks/sections of code within your NodeJS program (will illustrate with code demos)
- Each section handles a URL and a method (GET or POST)
- uses the Express "app" object

EXAMPLE

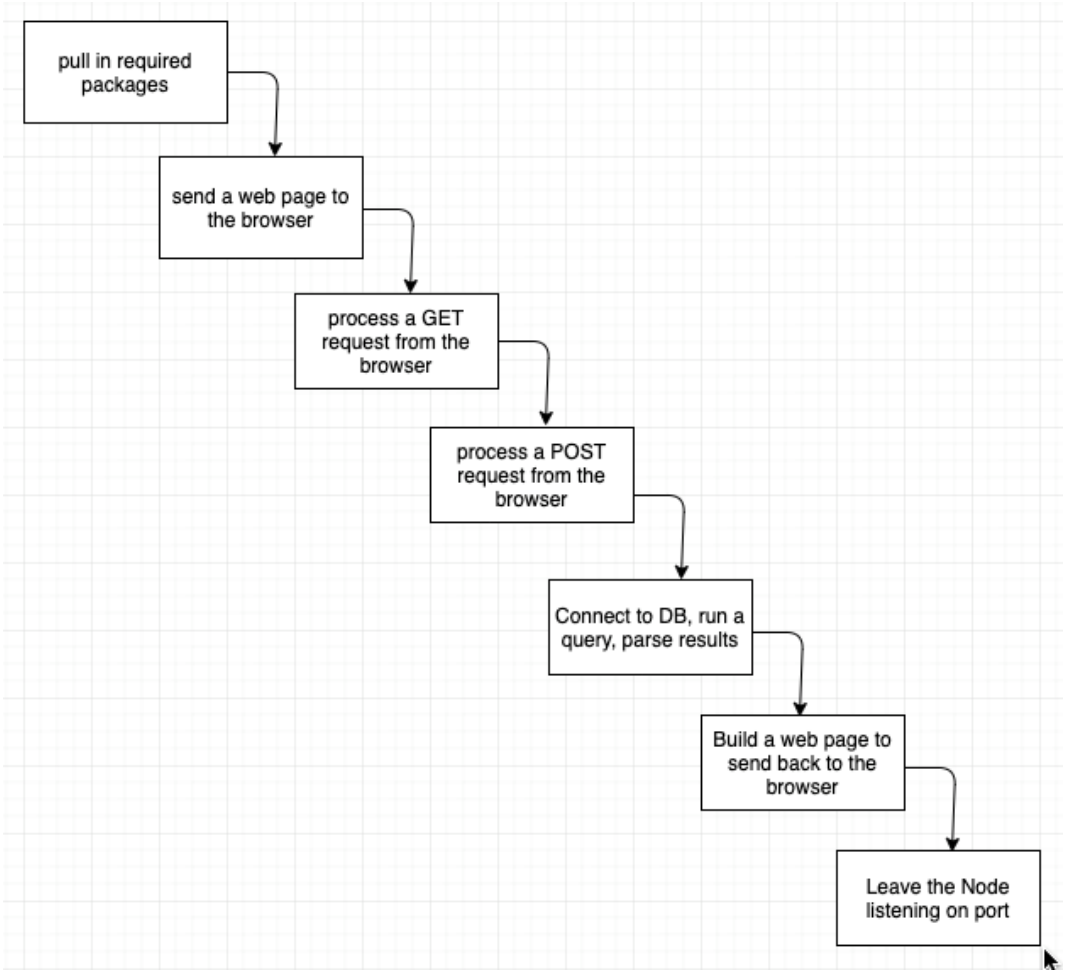
```
var express = require('express')
var app = express()


// respond with "hello world" when a GET request is made to the homepage

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

  res.send('hello world')

})
```


pull in required packages
send a web page to the browser
process a GET request from the browser
process a POST request from the browser
Connect to DB, run a query, parse results
Build a web page to send back to the browser
Leave the Node listening on port

ROUTING

- For example, a block of code may do
  - `app.get(/)` = process a get request for the "index" html file
  - `app.post(/)` = process a post request for the "index" html file
  - `app.get(/URL...)` = process a get request for the file "URL"
  - `app.post(/URL...)` = process a get request for the file "URL"

- Further reading: https://expressjs.com/en/guide/routing.html

STARTING UP THE NODE

- Node is initiated from the command line
- Node runs in the background until you stop it (`<CTRL>+C`)

This code initiates the Node running in background: (`node StartServer.js` from `yourApp` folder)

StartServer.js

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

//create a server object:
http.createServer(function  (req, res) {
  res.writeHead(200,  {'Content-Type': 'text/html'}); // tells client it is HTML
  res.write('Hello 3308 World!'); //write a response to the client
  res.write(req.url);
  res.end(); //end the response

}).listen(8080); //the server object listens on port 8080

console.log('Server running at http://127.0.0.1:8080');
```

STARTING UP THE NODE

- The createServer function has two arguments:
  - "`req`" is the request coming in from the client
  - "`res`" is the result being sent to the client

StartServer.js

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

//create a server object:
http.createServer(function  (req, res) {
  res.writeHead(200,  {'Content-Type': 'text/html'}); // tells client it is HTML
  res.write('Hello 3308 World!'); //write a response to the client
  res.write(req.url);
  res.end(); //end the response

}).listen(8080); //the server object listens on port 8080

console.log('Server running at http://127.0.0.1:8080');
```

ACCESSING FILE SYSTEM

- The fs module allows node to work with the file system on your computer
- `fs.readFile()` method allows node to read a file
- Run this as `DemoHTML.js`
- It reads a file `DemoHTML.html`, passes it into "`data`", and writes "`data`" to the result sent back to the browser.
- The browser is displaying the URL: http://127.0.0.1:8080

DemoHTML.js

```
var http = require('http');
var fs = require('fs');

//create a server object:
http.createServer(function (req, res) {
  fs.readFile('DemoHTML.html', function(err, data) {
  res.writeHead(200, {'Content-Type': 'text/html'}); // tells client it is HTML
  res.write(data); //write a response to the client
  res.end(); //end the response
  });
 }).listen(8080); //the server object listens on port 8080
console.log('Server running at http://127.0.0.1:8080');
```

PARSING URL

- The url module allows node to parse a URL passed to it
- `url.parse()` method parses out host, pathname and variable values from a URL (which will eventually be passed from the client.)
- Run "`parseURL.js`" in a command prompt !

ParseURL.js

```
var url = require('url');
var adr = 'http://localhost:8080/default.htm?year=2019&month=march';
var q = url.parse(adr, true);

console.log(q.host); //returns 'localhost:8080'
console.log(q.pathname); //returns '/default.htm'
console.log(q.search); //returns '?year=2019&month=march'

var qdata = q.query; //returns an object: { year: 2019, month: 'march' }
 console.log(qdata.month); //returns 'march'
```

READING A FILE AFTER PARSING URL

- Now, we can combine the URL parser with the File Reader
  - For this example, we will use two HTML files:
    - Hello.html and Goodbye.html
  - We will pass from the browser the URL indicating which HTML file to write
- This code starts the node, retrieves a URL from the browser and opens one of two files specified. Run `URLFile.js`. Type URL `http://127.0.0.1:8080/Hello.html` into the browser

URLFile.js

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

http.createServer(function  (req, res) {
  var q = url.parse(req.url, true);
  var  filename = "." + q.pathname;
  fs.readFile(filename,  function(err, data) {
    if (err) {
      res.writeHead(404, {'Content-Type': 'text/html'});
      return res.end("404 Not Found");
    }
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(data);
    return res.end();
  });
}).listen(8080);
```

POSTGRES WITH NODEJS

- How to get NodeJS to talk to PostgreSQL (run `QueryDB_pg.js`)
  1. Build the configuration string
  2. Build and run the query (`db.any()` method)
  3. View the results (`rows[0]`)

QueryDB_pg.js

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

var pg = require('pg-promise')();
// DB connection String
const dbConfig = {
  host: 'localhost',
  port: 5432,
  database: 'football_db',
  user: 'postgres',
  password: '******'
};

var db = pg(dbConfig);

var query = 'SELECT * FROM football_games;';
db.any(query)
    .then(function (rows) {
        console.log(rows[0]);
    })
    .catch(function (err) {
        console.log("error message");
  });
```

PARSING QUERY RESULTS

- The db.any() function within JS returns a two-dimensional array
  - "rows" occurs once for every row in the table, indexed by numbers starting at zero
  - "fields" occurs once for every column in the table, indexed by column name
  - We can use a "for" loop to see all the rows (run `QueryDB_2_pg.js`)

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

var pg = require('pg-promise')();
// DB connection String
const dbConfig = {
  host: 'localhost',
  port: 5432,
  database: 'football_db',
  user: 'postgres',
  password: '********'
};

var db = pg(dbConfig);


  // Run a query
  var query = 'SELECT * FROM football_players;';
  db.any(query)
    .then(function(rows){
        for (i = 0; i < rows.length; i++)
        {
          console.log(rows[i]);
        }
    })
    .catch(function(err){
      console.log("error running query: ", err);
    })
```

INTEGRATING UI AND DB USING NODEJS

Demonstration of Integration using a NodeJS program `HandleForm.js` with Express framework routing

- Program Steps:
  1. Does a res.send to send the HTML form page to the browser
  2. Does a app.get to receive the HTTP GET from the form, passing back a PlayerId when SUBMIT is pressed in the browser
  3. Builds a database connection string
  4. Connects to the database
  5. Runs a query getting the row for that PlayerId
  6. Parse out the results
  7. Build a web page and send it back to the browser
  8. Leave the Node web server running, listening on port 8080

HTML FORM

```
<html>
   <body>
      <form action = "http://127.0.0.1:8080/process_get" method = "GET">
         <div align=center>
          <h1>PlayerId:</h1> <input type = "text" name = "playerId">  <br><br>
          <input type = "submit" value = "Submit">
          </div>
      </form>
   </body>
</html>
```

NODEJS CODE – SENDING FORM TO HTML

```
//including express framework and creating express object ‘app’
var express = require('express'),
    app = express();

// send the form page to the browser
app.use(express.static('public'));
app.get('/index.html', function (req, res) {
   res.sendFile( __dirname + "/" + "index.html" );
})
```

NODEJS CODE – PARSING URL TO READ USER INPUT

```
// process the GET request sent by the form
app.get('/process_get', function (req, res) {

// Prepare output in JSON format
   response = {
      playerId:req.query.playerId,
   };
   console.log(response);

//read input value from the request
   PlayerId = req.query.playerId;
```

NODEJS CODE – SETTING UP DB CONNECTION

```
// Build the DB connection String
var pg = require('pg-promise')();

// DB connection String
const dbConfig = {
  host: 'localhost',
  port: 5432,
  database: 'football_db',
  user: 'postgres',
  password: '********'
};

//creating a db object
var db = pg(dbConfig);
```

NODEJS CODE – QUERY EXECUTION AND RESPONSE FORMULATION

```
//initializing variable with user input

var key = PlayerId;


//building query

var query = 'SELECT name as "name", major as "major" FROM football_players
where id = ' + key + ';';


//running query on database

 db.any(query)

    .then(function(rows){

      console.log('result = ', rows[0]);

      Name = rows[0].name;

      Major = rows[0].major;

     res.send('</br></br><h2 align=center>Player:</h2><h1
align=center>'+ PlayerId +' - ' + Name + ' ' + Major +
' ' + '</h1>');

    })

    .catch(function(err){

      console.log('error running query', err);

    })
```

NODEJS CODE – CONTINUE LISTENING TO PORT

```
// Leave the NodeJS web server listening on port 8080
  var server = app.listen(8080, function () {
     var port = server.address().port
     console.log("Example app listening at port", port)
  })
```