**Node**

* Javascript was made to work inside the browser
* Over time, javascript has begun to be used to write scripts, and it can be used in the server side
* Javascript doesn’t define a standard library
* Common JS API fills gaps by defining API’s for common application needs
  + It defines a module format
  + Node follows the common JS module specification

Node modules

* Each file in node becomes its own node module
* The module variable (javascript object) gives access to the current module definition in a file
* The module.exports variable determines the export from the current module
  + Whatever is assigned to the module.exports property, it becomes the exported value from the current module
  + Whatever is exported from a module becomes available in the 2nd application
* The require function is used to import a module
* 3 types
  + File modules
  + Core modules
    - Part of node
    - Ex: path, fs, os, util
  + External node modules
    - Third party modules
    - Can be installed using NPM
    - The installed modules will be in node\_modules folder in your node application

Using node modules

* When a node module needs to be used within another node file, you have to use the require function
  + File based modules:
    - require(“./modulename”)
    - specify the relative path to the file
  + core and external modules
    - require(“./modulename”)
    - looks for external modules in the folder ./node\_modules and goes higher and higher in the folder hierarchy

Understanding node modules

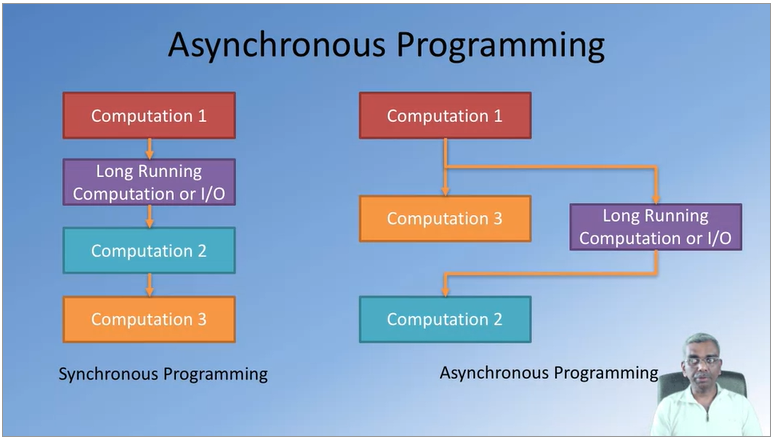
* npm init
  + this command initializes a node application inside the present location of the terminal
* using the npm command

**callback and error handling**

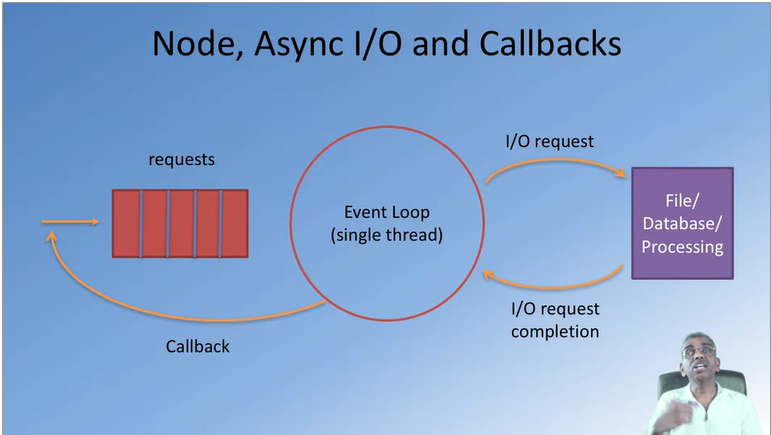
two salient features of Javascript

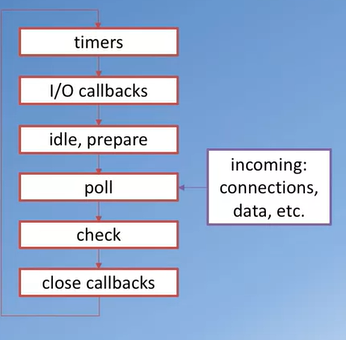
* First-class functions
  + A function can be treated the same way as any other variable
* Closures
  + A function defined inside another function has access to all the variables declared in the outer function
  + The inner function will continue to have access to the variables from the outer scope even after the outer function has returned

Asynchronous programming



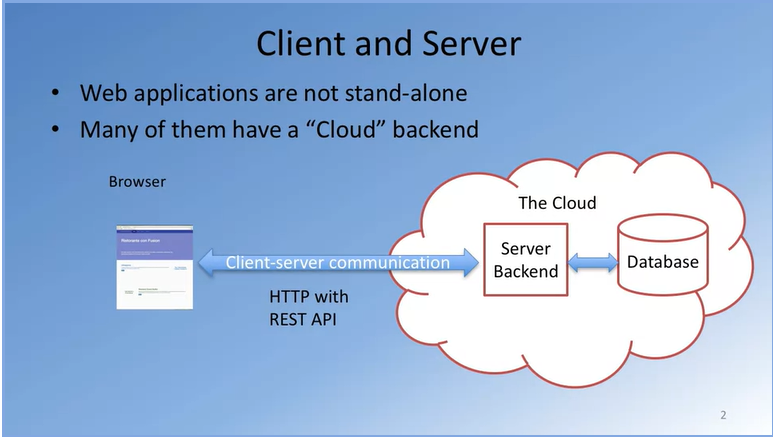
* In synchronous programming, computation 2 is dependent on computation 2
* In asynchronous programming, computation 3 is not dependent on computation 2
  + - Ensures that computation 3 will be done quicker
* Node, asynchronous I/O and callbacks



* + Node is based on a single thread event loop
  + The single thread event loops picks up requests and executes it one after another
  + When an I/O request is completed, the event loop will put the request inside a callback and send it to the request queue. The callback will then be handled by the event loop
  + The event loop is a continuously running loop which basically picks up requests from the request queue and services them one at a time
* Event loop phases
  + Timers: this phase executes callbacks scheduled by settimeout() and setInterval()
  + I/O callbacks: executes almost all callbacks w the exception of close callbacks, the ones scheduled by timers, and setImmediate()
  + Idle, prepare: only used internally
  + Poll: retrieve new I/O events; node will block here when appripriate
  + Check: setImmediate() callbacks are invoked here
  + Close callbacks: e.g., socket.on(‘close’, …). Handles any socket closures

**Networking essentials**

Client and server



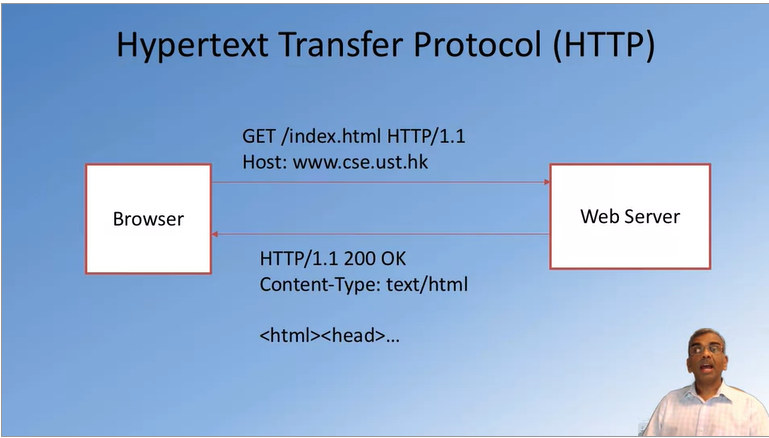
* Web applications are not stand-alone
* Many of them have a “cloud” backend

The networking alphabet soup

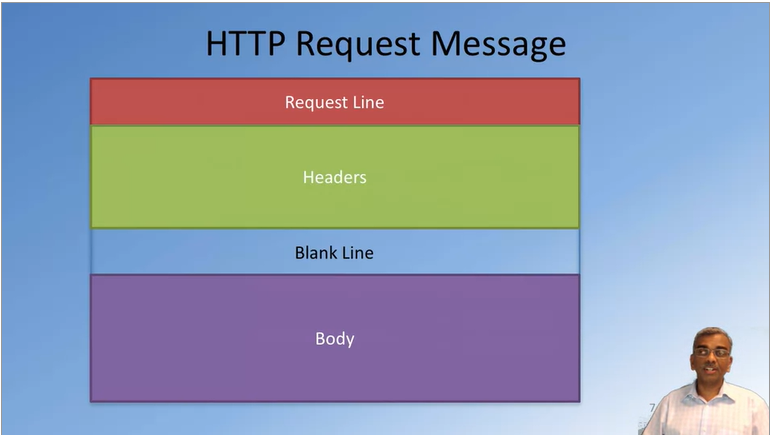
* HTTP protocol used in communicating between the client and the server
* URL is a string of directors separated by colons
* JSON one way of encoding data that is being shipped from the server side to the client side or vice versa
* XML another way of encoding data as it is in shipment between the client and server side
* SOAP is a protocol that allows communication between distributed entities within the network
* REST representational state transfer
* GET
* PUT
* POST
* DELETE

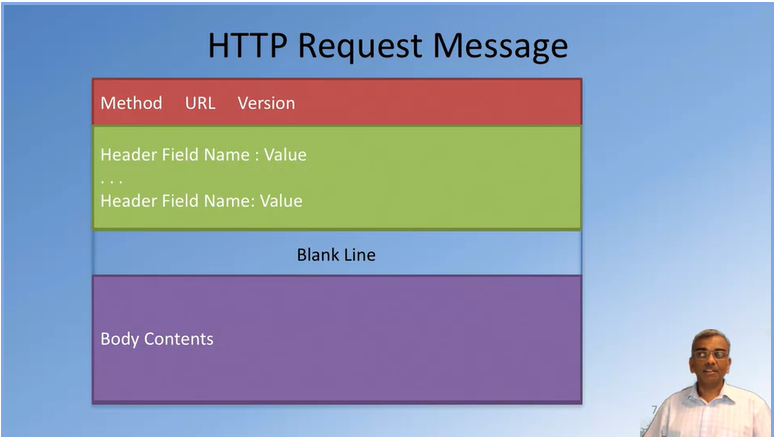
Client server communication

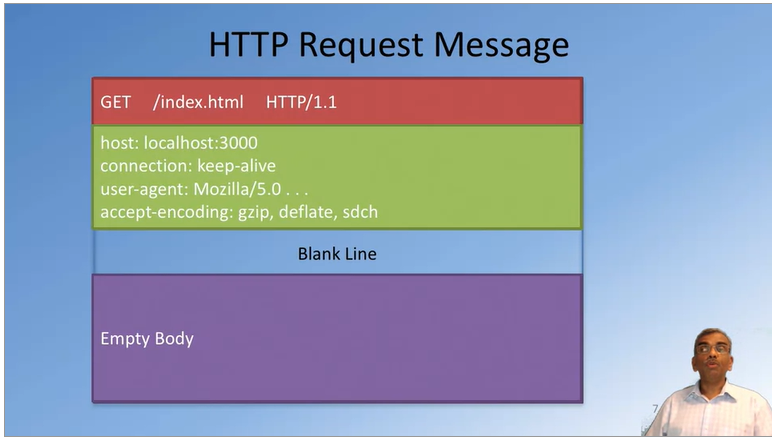
* Network operations cause an unexpected amount of delays
* You need to write applications recognizing the asynchronous nature of communication
* HTTP
  + A client-server communication protocol
  + Allows retrieving inter-linked text documents (hypertext)
  + HTTP verbs
    - HEAD
    - GET
    - POST
    - PUT
    - DELETE
    - TRACE
    - OPTIONS
    - CONNECT



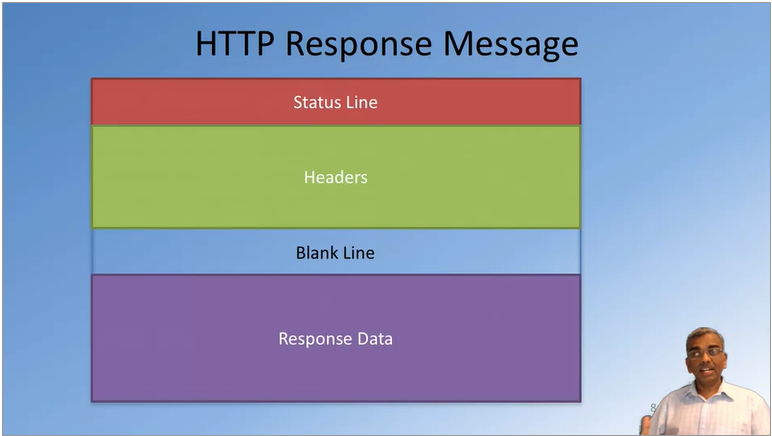
* + HTTP request message

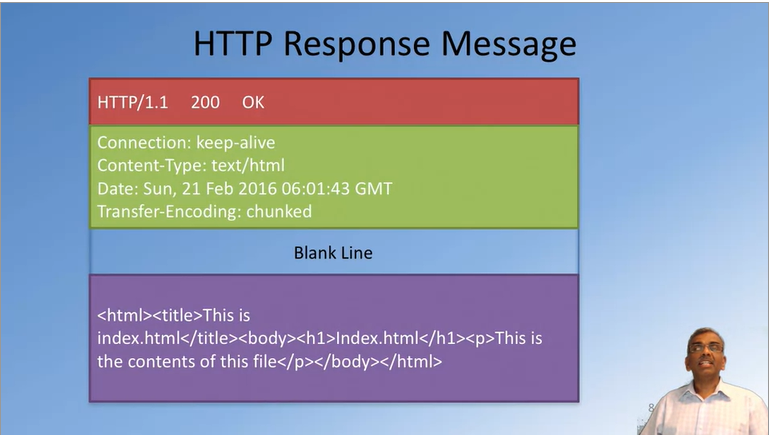






* + HTTP response message





* + - HTTP response codes
      * 200 – OK
      * 201 – Created
      * 301 – moved permanently
      * 304 – not modified
      * 400 – Bad request
      * 401 – unauthorized
      * 403 – forbidden
      * 404 – not found
      * 422 – unprocessable entry
      * 500 – internal server error
      * 505 – HTTP version not supported
    - Server may send back data in a specific format
      * eXtensible Markup Language (XML)
      * Javascript Object Notation (JSON)

**Node and the HTTP module**

Node HTTP module

* Core networking module supporting a high-performance foundation for a HTTP stack
* Using the module:
  + const http = require(‘http’);
    - imports the module “http”
* Creating a server
  + const server = http.createServer(function(req, res){ … });
    - once the module is supported, the http module creates a function which takes another function as a parameter which acts as a callback function for the created module, and has 2 parameters: request and response
* Starting the server
  + server.listen(port, …);
    - once the server is created, the server.listen takes in the port number and the host name of the server and that will start up the server
* incoming request message information is available through the first parameter: req
  + req.headers, req.body, etc.
* the response message is constructed on the second parameter: res
  + res.setHeader(“Content-Type”, “text/html”);
  + res.statusCode = 200;
  + res.write(“Hello World!”);
  + res.end(“<html><body><h1>Hello World</h1></body></html>”);

Node path module

* enables us to specify the path to a file and examine whether the file exists or examine more details about a file for example: the extensions
* using the path module
  + const path = require (“path”);
* some example path methods
  + path.resolve(“./public”+fileURL);
    - converts a relative path to an absolute path
  + path.extname(filePath);
    - checks the extension name of the file

Node fs module

* file system module, a core module in node enables us to read and write files that exist in the local file system in the computer
* use fs module in your application
  + const fs = require (“fs”);
* some example fs methods
  + fs.exists(filePath, function(exists) { … });
    - checks whether the file exists in the local file system or not
  + fs.createReadStream(filePath).pipe(res);
    - the file will be read from the file, which will be given by the file path, and the contents will be put into the body of the response message