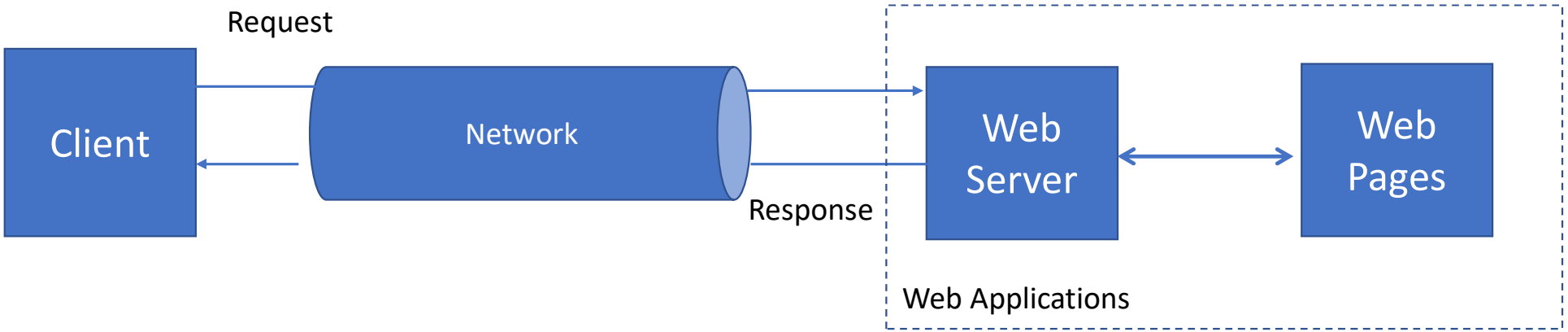
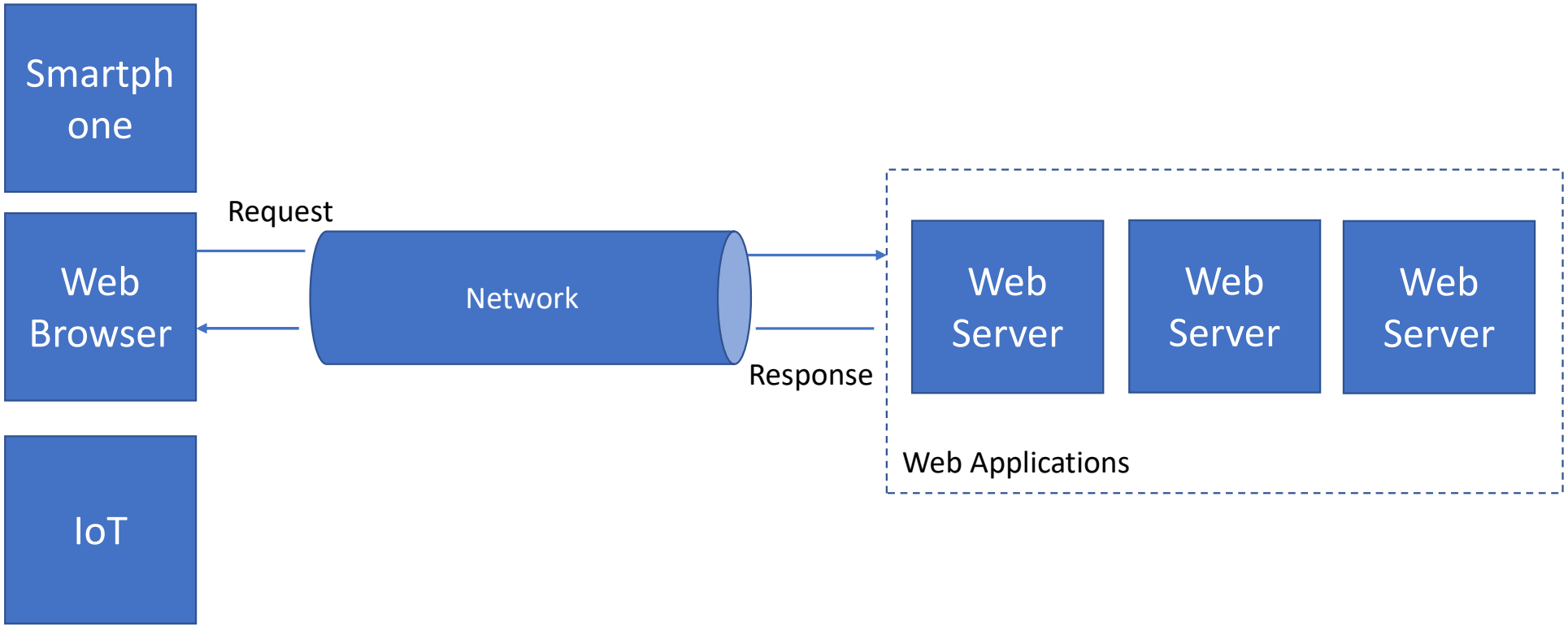

HTTP Revisited
Chandreyee Chowdhury

Web App Architecture-web 1.0



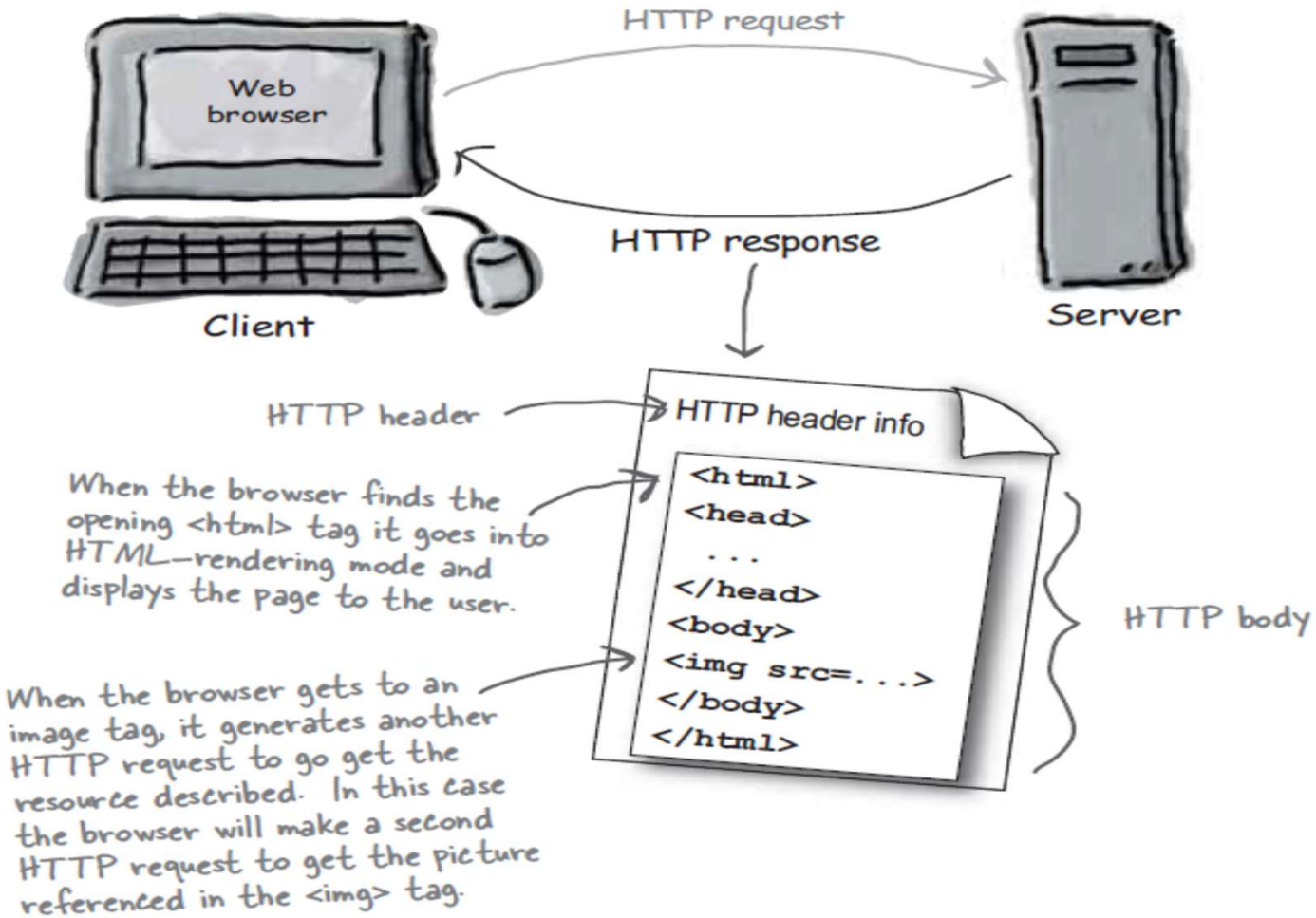
- Data and presentation are closely coupled as web pages are static

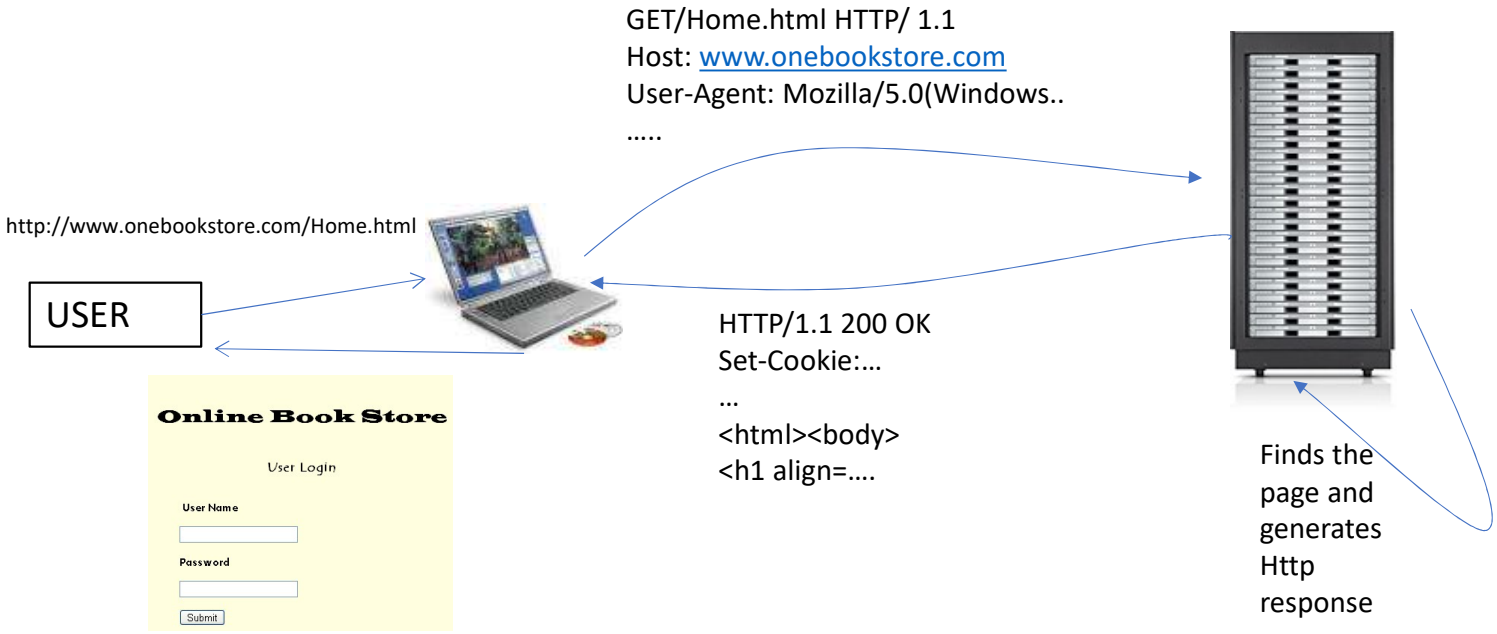

Smartph
one
Web
Browser
IoT
Request
Network
Response
Web
Server
Web
Server
Web
Server
Web Applications

Why HTTP

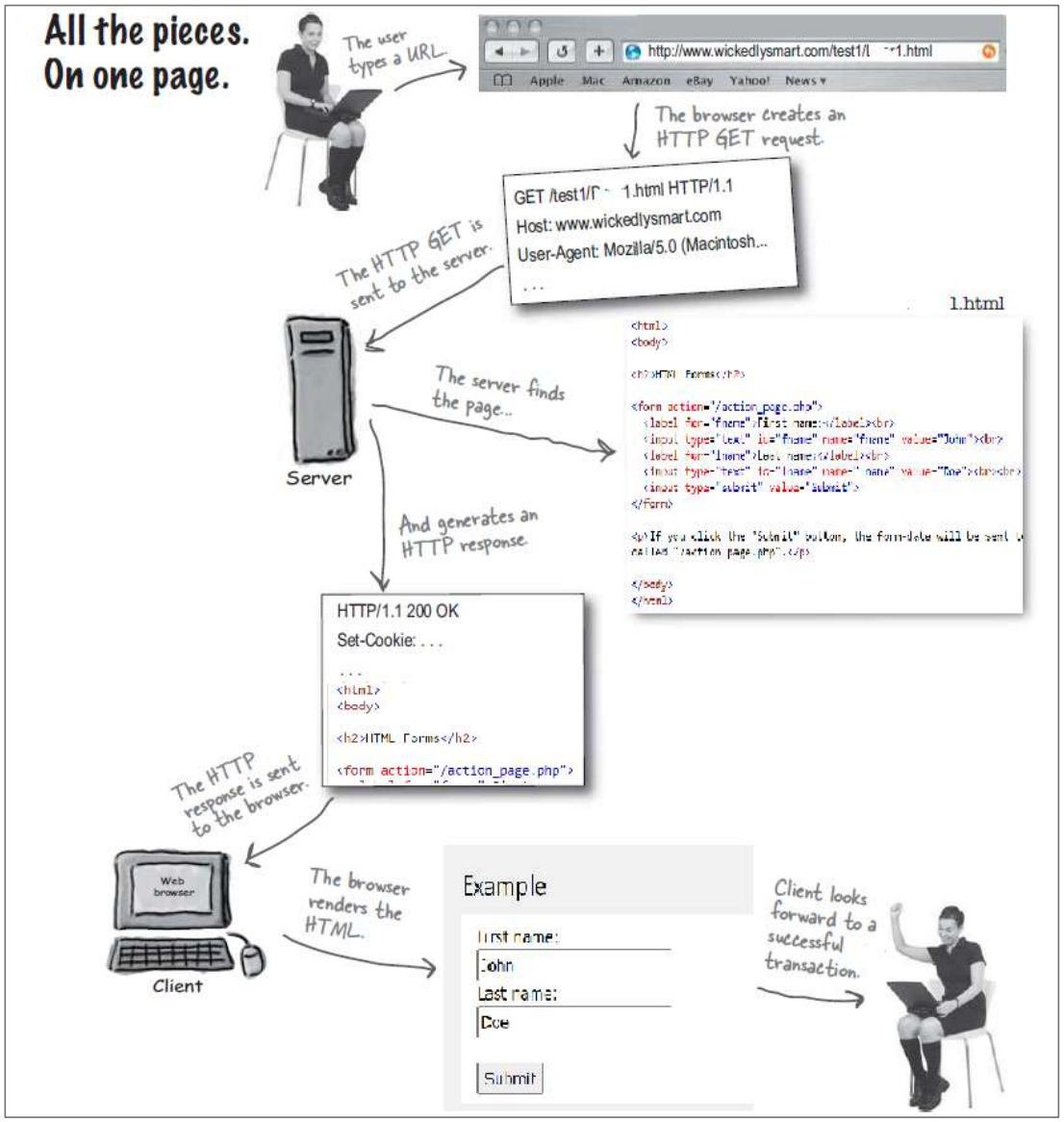
- It provides a uniform interface to access the resources and services from a web server or cloud
- Reuse infrastructure for ubiquity
  - Web is ubiquitous
- Reusing
  - Application frameworks and libraries
  - Load balancing infrastructure for different applications including interacting with cloud
  - Session handling
- Distribute requests throughout the servers

HTTP and HTML

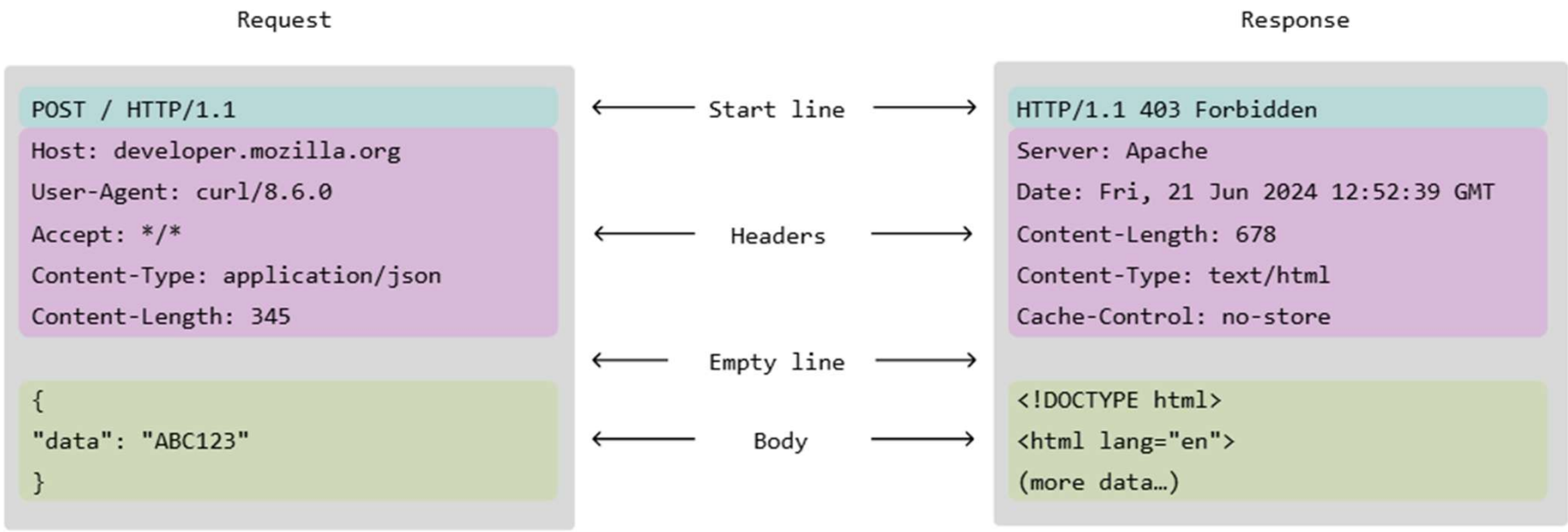



GET/Home.html HTTP/ 1.1
Host: www.onebookstore.com
User-Agent: Mozilla/5.0(Windows..
.....
http://www.onebookstore.com/Home.html
USER
HTTP/1.1 200 OK
Set-Cookie:…
…
<html><body>
<h1 align=….
Finds the page and generates Http response
Online Book Store
User Login
User Name
Password
Submit

HTTP Request Response



HTTP Message Anatomy

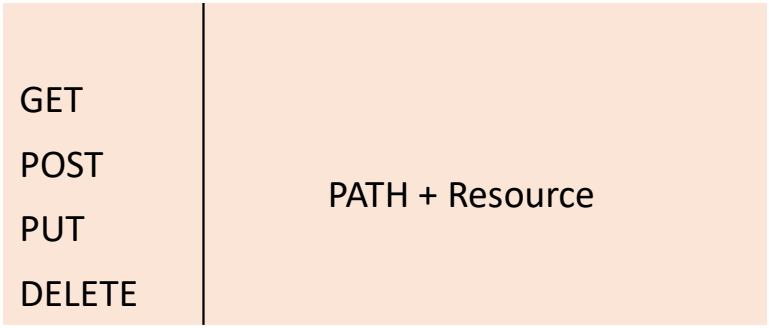


HTTP Request

- ***Every request has a method and a resource path***
- ***HTTP GET***
  - The total amount of characters in a GET is really limited (depending on the server)
  - The data you send with the GET is appended to the URL up in the browser bar, so whatever you send is exposed
  - Because of this, the user can bookmark a form submission if you use GET
- HTTP POST
  - The data is included in the request body
  - More data can be sent
  - General purpose sending of data

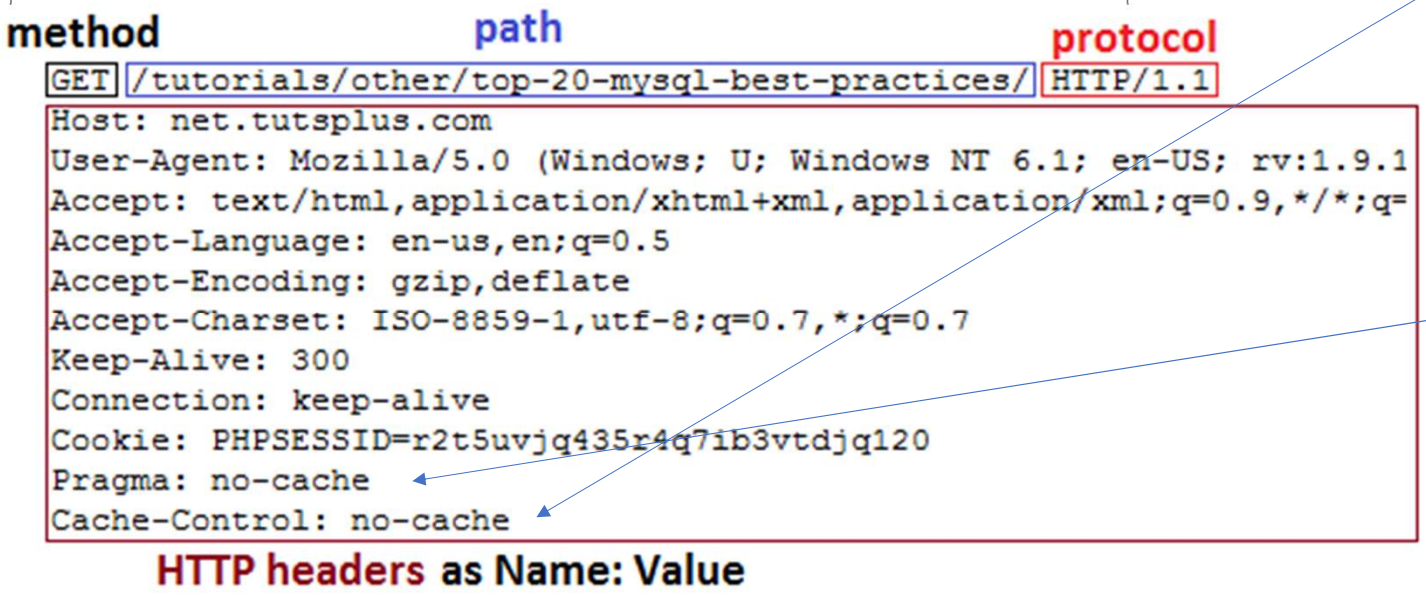
HTTP Methods

- Put
  - Asking the server to store some Data
- Delete
  - Remove some information from the server



Request line

- GET/com/Kolkata/Home.html HTTP/ 1.1
- Method
- <path+resource>

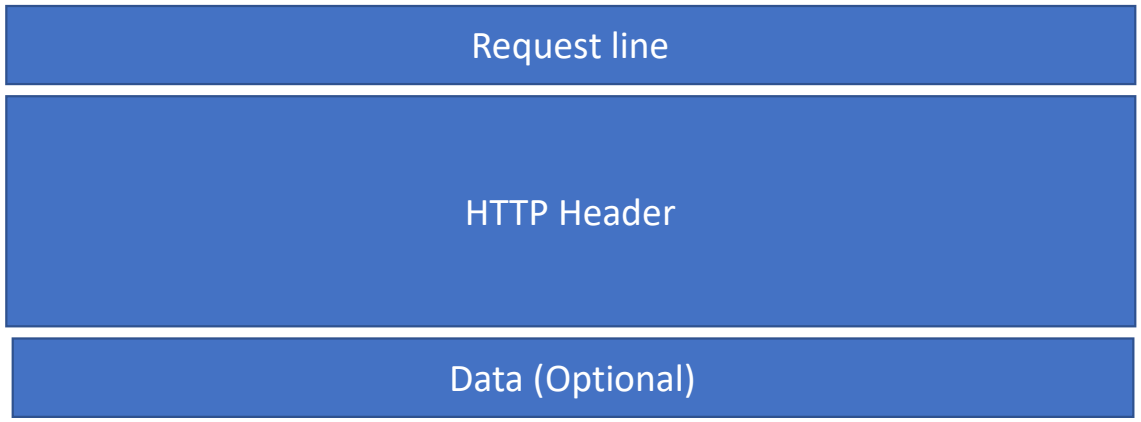


The response may be stored by *any* cache, even if the response is normally non-cacheable. However, the stored response MUST *always* go through validation with the origin server first before using it

The server MUST NOT use a cached copy when responding to such a request.

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

HTTP Headers



- Headers are meta information but body of a HTTP message contains pure data
  - These are the extra information that the client is giving the server to help it complete that Request.
- If message body is sent without the header then the server may process the request
  - May not send the response in the expected format
- When body of a message is missing when it was required, the relevant information would not be processed

Uniform Resource Locator

https://wishnet.in/home/login.html

- The way resources are identified is called a URL
- `http://<host name>:<port number>/path/resource?key1=value1&key2=value2`
- Using the query parameters we can pass extra information about a specific aspect of a resource to the server
- URL encoding encodes any character that is not allowed in the query param spec
- For dynamically constructed URLs with data, it is better to encode all URLs as data may not follow the spec
- It is good to provide the correct file extension in the encoded URLs but not a requirement

Data Types

Image/jpg

Image/png

Text/plain

Text/html

- The way data is stored in the server and the way it is sent in the body may differ
  - MIME type allows this adaptation
- A media type (also known as a Multipurpose Internet Mail Extensions or MIME type) indicates the nature and format of a document, file, or assortment of bytes. MIME types are defined and standardized in IETF's RFC 6838
- There should be some way of interpreting the type of data sent in body
  - Image data
- The type represents the general category into which the data type falls, such as video or text.
  - The subtype identifies the exact kind of data of the specified type
- All of these different MIME types are identifiers for a well known format for the data in the body of either a request or a response.
  - Based on the MIME type the data will be processed
- MIME types are changed between client and server

MIME Type

- **Discrete** and **multipart**. Discrete types are types which represent a single file or medium, such as a single text or music file, or a single video.
- A multipart type represents a document that's comprised of multiple component parts, each of which may have its own individual MIME type;
- A multipart type may encapsulate multiple files being sent together in one transaction.
- Two multipart types are *message* and *multipart*.
- A *message* that encapsulates other messages.
- This can be used, for instance, to represent an email that includes a forwarded message as part of its data, or to allow sending very large messages in chunks as if it were multiple messages.

Content-Type: multipart/form-data

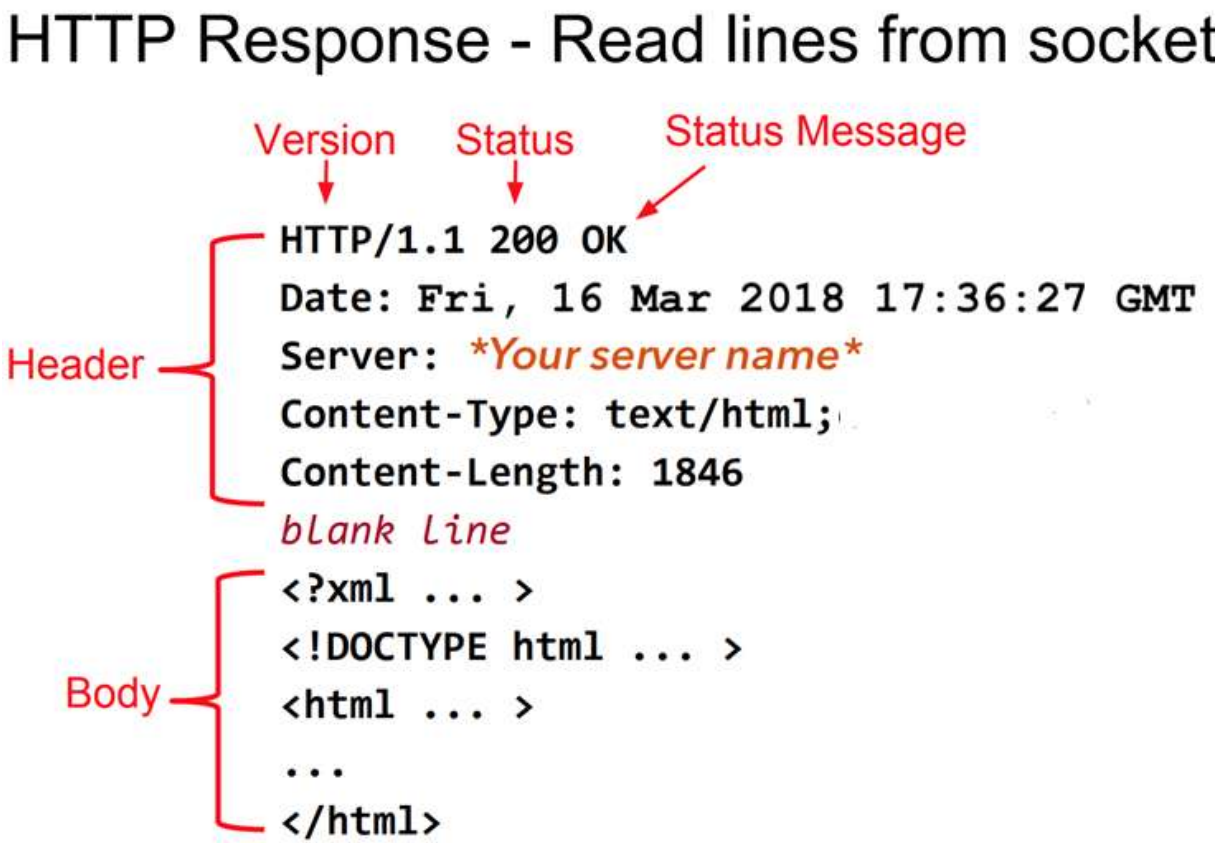
- These are written as content types
- Multipart types indicate a category of document broken into pieces, often with different MIME types

```
<form action="/SelectCoffeeType.html" method="post" enctype="multipart/form-data">
   <input type="text" name="description" value="some text">
   <input type="file" name="myFile">
   <button type="submit">Submit</button>
</form>
```

```
POST /foo HTTP/1.1
Content-Length: 68137
Content-Type: multipart/form-data; boundary=---------------------------9747672998524989295316105 75

-----------------------------9747672998524989295316105 75
Content-Disposition: form-data; name="description"

some text
-----------------------------9747672998524989295316105 75
Content-Disposition: form-data; name="myFile"; filename="foo.txt"
Content-Type: text/plain

(content of the uploaded file foo.txt)
-----------------------------9747672998524989295316105 75--
```

HTTP Response

- We cant be present at the sever to check what has happened
- 1XX
- 2XX (201-created)
- 3XX
- 4XX
- 5XX

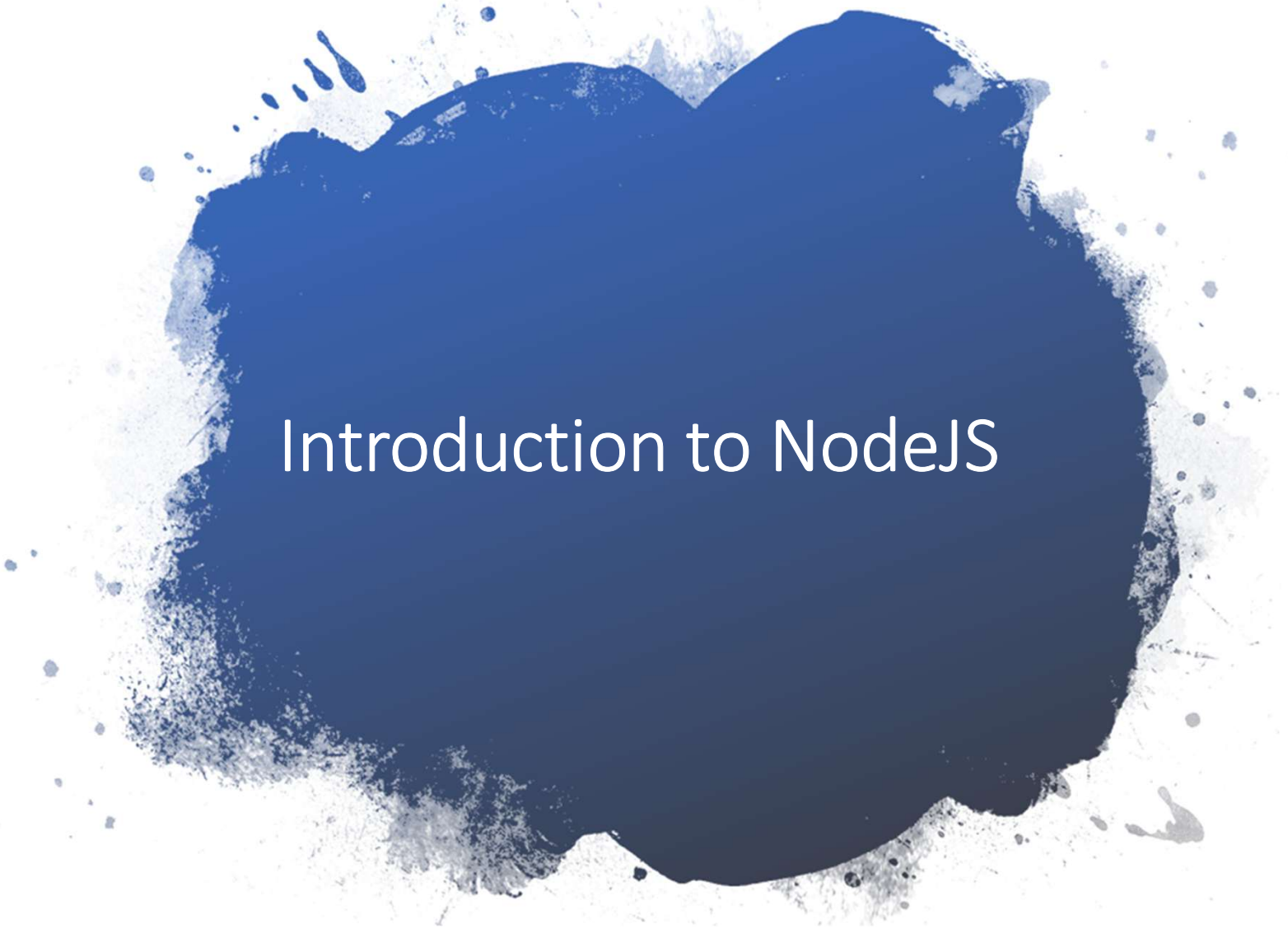


Response Codes

- 1XX- informal continuing process
- 2XX- successful
  - 200 means the client can assume that the server has successfully handled the request
- 3XX-redirection
  - Resend the request as the requested resource may have been moved
- 4XX- client error
  - Requested resource not found
  - Problem in request formatting
- 5XX-server error
  - The response body may contain the detailing of the error
- Depending on the response code and the MIME type, the body of the response is processed

Cookies

- They are very small limited pieces of data, that the server sends back to the client
- Goes through response header
- Size of the cookie has to be small
- Date/lifetime
- Encryption while sending the cookie
  - client only sends this cookie back to the server if a secure link, or an HTTPS communication protocol is being used


Introduction to NodeJS

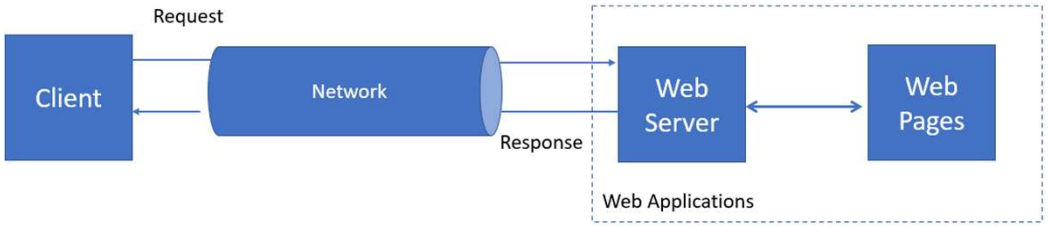
Javascript

- The first incarnations of JavaScript lived in browsers
- 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
- Node.js allows you to run JavaScript code in the backend, outside a browser.
- It was built on top of the Google Chrome V8 engine

```
console.log("Hello World");
```

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

Modular programming with Node.js



- Node.js is an open-source and cross-platform JavaScript runtime environment.
- With Node.js, we not only implement our application, we also implement the whole HTTP server
- In fact, our web application and its web server are basically the same
- It allows you to have a clean main file, which you execute with Node.js
- Clean modules that can be used by the main file and among each other
- A Node.js app runs in a single process, without creating a new thread for every request.
- Node.js provides a set of asynchronous I/O primitives in its standard library that prevent JavaScript code from blocking.
- In addition, libraries in Node.js are generally written using non-blocking paradigms.
- Accordingly, blocking behavior is the exception rather than the norm in Node.js.

Writing HTTP Server

- The first line *require*s the *http* module that ships with Node.js and makes it accessible through the variable *http*
  - `var http = require('http');`
- We then call one of the functions the http module offers: *createServer*
- This function returns an object
- 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 server = http.createServer(<function as argument>);`
  - `server.listen(8888);`
- Because in JavaScript, functions can be passed around like any other value.

Functional Programming in Javascript

```
function say(word) {
  console.log(word);
}

function execute(someFunction, value) {
  someFunction(value);
}

execute(say, "Hello");
```

```
function execute(someFunction, value) {
  someFunction(value);
}

execute(function(word){ console.log(word) }, "Hello");
```

- We define the function we want to pass to *execute* right there at the place where *execute* expects its first parameter.
- This way, we don't even need to give the function a name, which is why this is called an *anonymous function*.

Callbacks

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

- The JavaScript interpreter of Node.js first has to read the complete result set from the database
- Then it can execute the *console.log()* function
- According to the execution model of Node.js - 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 execution.

```
1  database.query("SELECT * FROM hugetable", function(rows) {
2      var result = rows;
3  });
4  console.log("Hello World");
```

- It introduces the concept of event-driven, asynchronous callbacks, by utilizing an event loop.

Callbacks

```
1  database.query("SELECT * FROM hugetable", function(rows) {
2      var result = rows;
3  });
4  console.log("Hello World");
```

- Node.js can handle the database request asynchronously.
- Provided that *database.query()* is part of an asynchronous library, this is what Node.js does: just as before, it takes the query
- Then sends it to the database.
- Here, instead of expecting *database.query()* to directly return a result to us, we pass it a second parameter, an anonymous function.
- But instead of waiting for it to be finished, it makes a mental note that says
  - "When at some point in the future the database server is done and sends the result of the query, then I have to execute the anonymous function that was passed to *database.query()*."
- After printing to the console log, it goes to an event loop
- Node.js continuously cycles through this loop again and again whenever there is nothing else to do, waiting for events.

Callbacks

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

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

 console.log("Server has started.");
```

Event loop in HTTP server

- This also explains why our HTTP server needs a function it can call upon incoming requests
- if Node.js would start the server and then just pause, waiting for the next request, continuing only when it arrives, that would be highly inefficient
- Multi client support

❑It's important to note that this asynchronous, single-threaded, event-driven execution model isn't an infinitely scalable performance option

- Node.js is just one single process, and it can run on only one single CPU core.
- Node.js supports cluster module that enables creation of child processes to be executed on separate cores of a multi-core machine

Scaling for Multi-core CPUs

- For scaling throughput on a webservice, you should run multiple Node.js servers on one box, one per core and split request traffic between them.
- This provides excellent CPU-affinity and will scale throughput nearly linearly with core count.

```
if (cluster.isMaster) {
// Fork workers.
     for (var i = 0; i < numCPUs; i++) {
             cluster.fork();
        } }
else {
         http.Server(function(req, res) { ... }).listen(8000); }
```

https://stackoverflow.com/questions/2387724/node-js-on-multi-core-machines

Creating Modules with Node.j

In C++ or C#, when we're talking about objects, we're referring to instances of classes or structs. Objects have different properties and methods, depending on which templates (that is, classes) they are instantiated from. That's not the case with JavaScript objects.
In JavaScript, objects are just collections of name/value pairs - think of a JavaScript object as a dictionary with string keys.

- Somewhere within Node.js lives a module called "http", and we can make use of it in our own code
- By requiring it and assigning the result of the require to a local variable
- This makes our local variable an object that carries all the public methods the *http* module provides.
- It's common practice to choose the name of the module for the name of the local variable

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