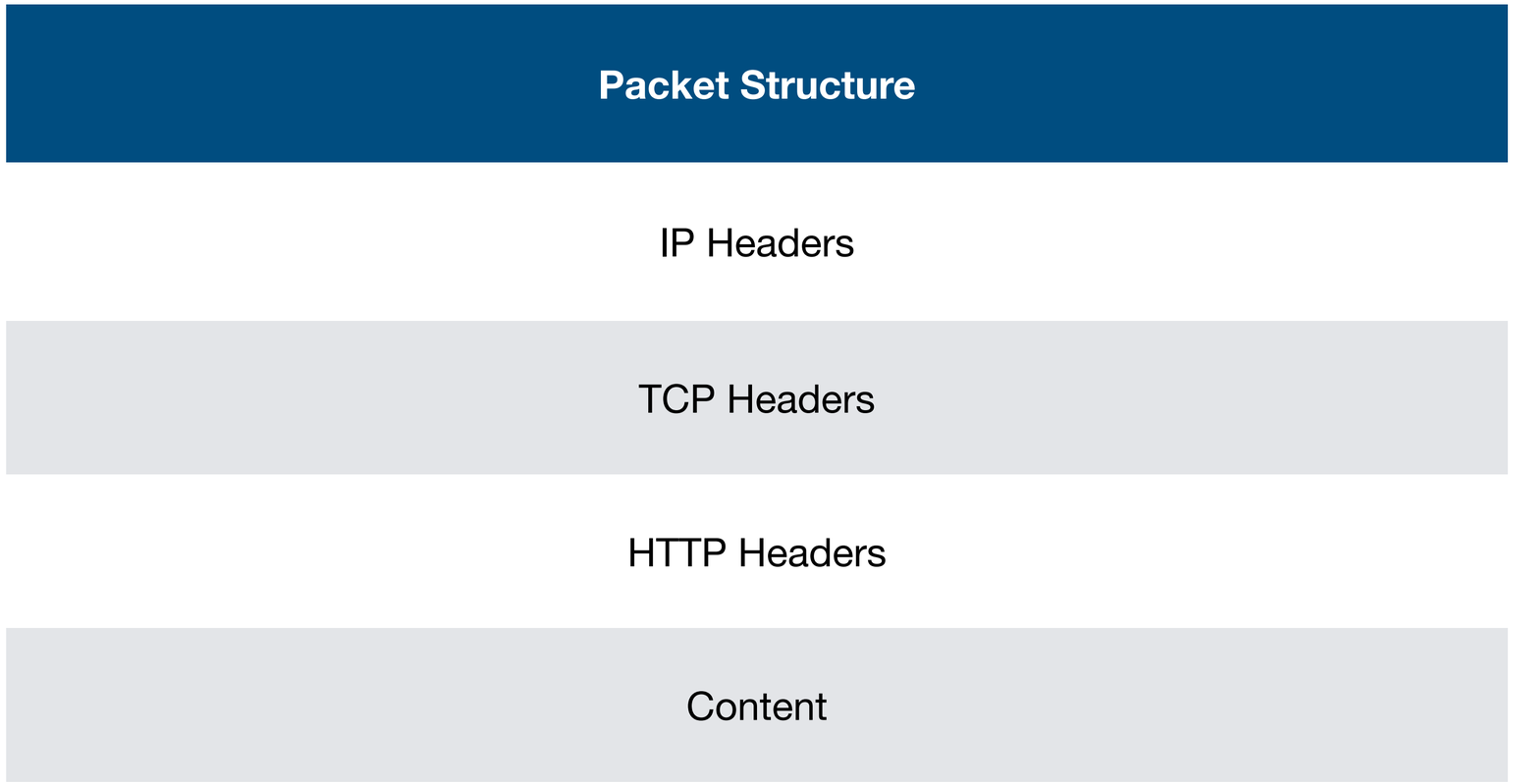
HTTP Overview

Roadmap

- The physical Internet
  - The Internet is a network of networks
  - Physically connected by cables and routers
- Internet Protocol (IP)
  - How routers move data through the Internet
  - Best effort basis
- Transport Control Protocol (TCP)
  - Transport information reliably through an unreliable network
  - Used by the client and server

Network Stack (A simplified view)

- Enter HTTP



HyperText Transfer Protocol (HTTP)

- HTTP is an application layer protocol
  - Protocols used by our applications
  - Protocols that are not concerned with the transmission of data
- [Almost] Always uses TCP for reliable communication
  - Always in this course

- Today:
  - A quick overview of many HTTP topics
  - We'll spend several lectures exploring these topics in more detail

HTTP

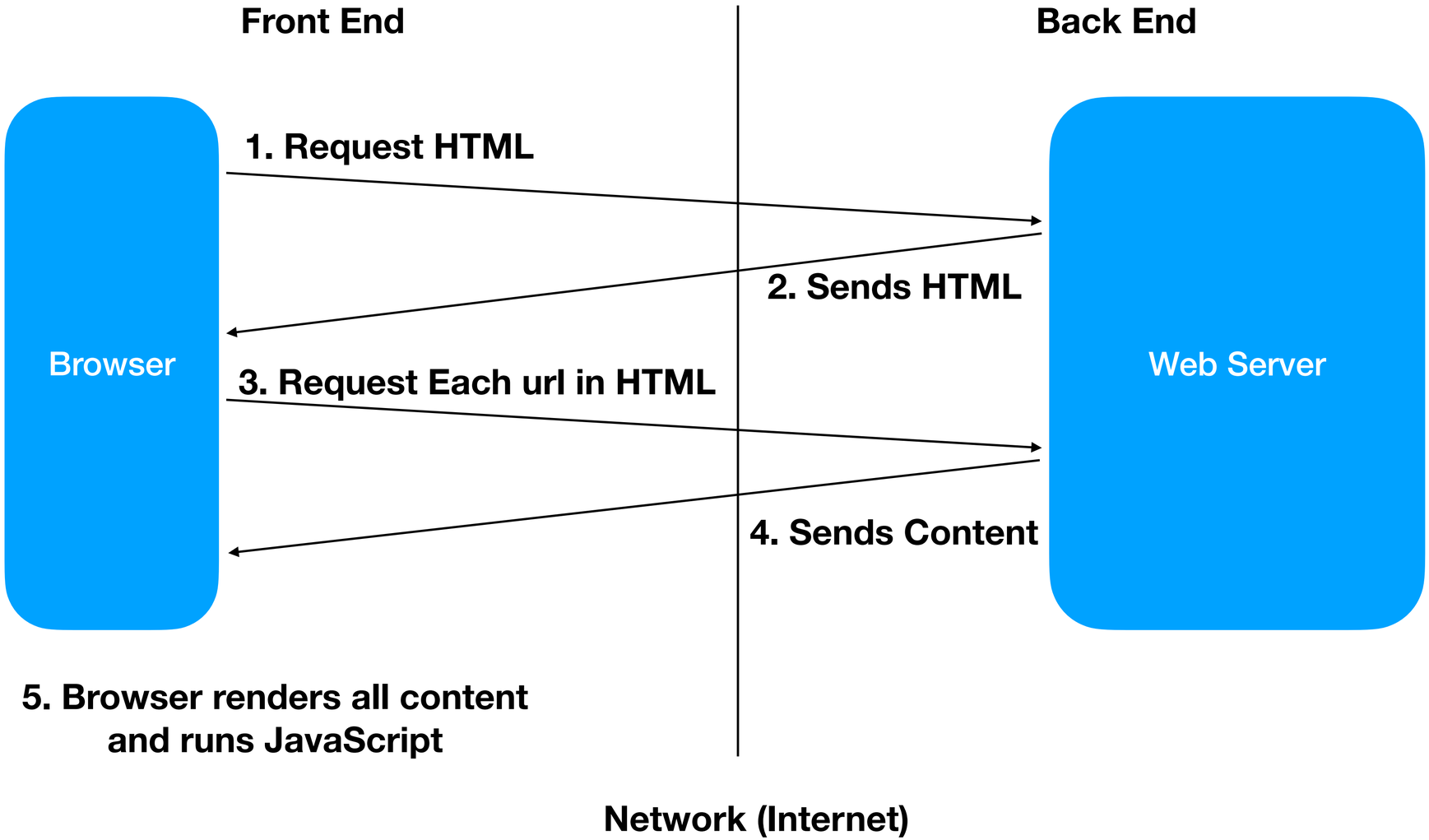
- HTTP is a protocol used to access content from a web server
- Protocol: An agreed upon set of rules
  - HTTP: Defines the format of messages sent to/from a web server
- HTTP is a Request - Response protocol
  - Client makes request to server
  - Server returns a response
  - Ex. Request The latest tweets from a user. Twitter server returns the tweets in its response
- Response may require more requests
  - Ex. Get HTML which requires CSS/JS/Images

Web Server

- Software that "speaks" HTTP
- Listens for HTTP requests and responds with HTTP responses
- We want to host our web pages/apps on the Internet using HTTP

- Terminology:
  - Front End - The part a web app that runs in the browser (HTML/CSS/JS)
  - Back End - The web server and all software that does not run on the user's machine

Loading a Web Site



HTTP Request

- Each HTTP request will contain the request type:
  - GET: Request information from a server
  - POST: Send information to a server
  - PUT: Add information to a service
  - DELETE: Delete information from a service
  - HEAD: Request only the headers of a response
- To start, we'll focus on GET and POST

HTTP Request

- HTTP GET Request
  - Used when requesting content from a server
  - [Typically] Only contains a URL and HTTP *headers*
  - When you click a link, your browser makes a get request
  - Requesting HTML/CSS/Javascript/Images/etc are GET requests
- HTTP POST Request
  - Used when sending data to a website
  - Contains a URL and a body [And HTTP headers]
  - When you submit a form, your browser [typically] makes a POST request
  - The contents of the form are sent in the *body* of the request

HTTP Request

Protocol://host:port/path?query_string#fragment

- Each request is made for a specific URL (Uniform Resource Location)
  - A URL uniquely identifies a resource and has the following parts
- Protocol - The protocol being used (ex. file, HTTP, HTTPS, FTP)
- Host - The IP address or domain name of the server
  - Used to route the request to the appropriate machine
- Port - The TCP port number of the host server
  - Defaults to 80/443 for HTTP/HTTPS respectively
- Path - Specifies the specific resource being requested from the server

HTTP Request

Protocol://host:port/path?query_string#fragment

- Query String - [Optional] Contains key-value pairs set by the client
- https://www.google.com/search?q=web+development
  - HTTPS request to Google search for the phrase "web development"
- https://duckduckgo.com/?q=web+development&ia=images
  - An HTTPS request to Duck Duck Go image search for the phrase "web development"

- Fragment - [Optional] Specifies a single value commonly used for navigation
- https://en.wikipedia.org/wiki/Uniform_Resource_Identifier
  - HTTPS Request for the URI Wikipedia page
- https://en.wikipedia.org/wiki/Uniform_Resource_Identifier#Definition
  - HTTPS Request for the URI Wikipedia page that will scroll to the definition of URI

HTTP Request

- Head of request
  - A Request-Line
    - <Request_Method> <Path> <HTTP_Version>
  - Header Fields
    - Contains additional information about the request in key-value pairs
    - <Header_Name>: <Header_Value>
- Body of request [Optional]
  - Content to be sent to the server

```
GET / HTTP/1.1
Host: cse312.com
```

```
POST /path HTTP/1.1
Host: cse312.com
Content-Length: 48
Content-Type: application/json

{"data": "Some data in the body of the request"}
```

HTTP Request

- Head and body of request are separated by a blank line
- If the request has a body,
  - The Content-Length header will be set to the size of the body
    - Lets the server know when the request ends
  - The Content-Type will be set
    - Lets the server know how to parse the data

```
GET / HTTP/1.1
Host: cse312.com
```

```
POST /path HTTP/1.1
Host: cse312.com
Content-Length: 48
Content-Type: application/json

{"data": "Some data in the body of the request"}
```

HTTP Request

- The first goal of your server:
  - Parse these requests

```
GET / HTTP/1.1
Host: cse312.com
```

```
POST /path HTTP/1.1
Host: cse312.com
Content-Length: 48
Content-Type: application/json

{"data": "Some data in the body of the request"}
```

HTTP Response

- When the server receives an HTTP request
  - Process the request and send a response to the client
- Response includes a response code indicating the status of the response
- Whether the request was GET or POST, the response will have a body containing the requested content
- Response also contains HTTP headers including meta information about the response and the server
  - Must include the content length and content type

HTTP Response

- Similar format as a request
- Response line
  - <HTTP_Version> <Response_Code> <Response_Text>
- Specify the content length and content type in headers
- Response itself goes in the body

HTTP/1.1 200 OK
Content-Length: 152
Content-Type: text/html

<html> ... </html>

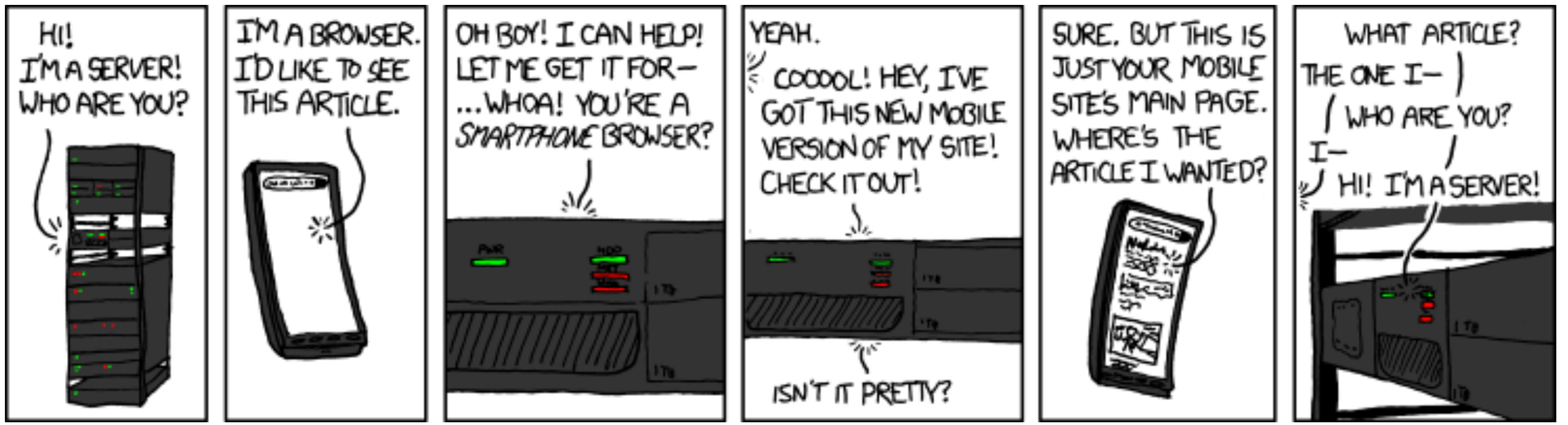
HTTP - Response

Common Response codes include:

- 200 OK
  - Request was handled as expected
- 301 Moved Permanently
  - Redirect to the new location
- 304 Not Modified
  - File in local cache can be used
- 403 Forbidden
  - You don’t have access to the requested page
- 404 Not Found
  - Requested data could not be found
- 500
  - Internal server error

HTTP

- HTTP is a stateless protocol
- Each request is handled in isolation even if a client just made another request
- If state is desired (ex. Login), the state must be sent with each request
- Cookies
- Tokens
- When handling an HTTP request, do not have to care who sent it



https://xkcd.com/869/

Examples in the browser