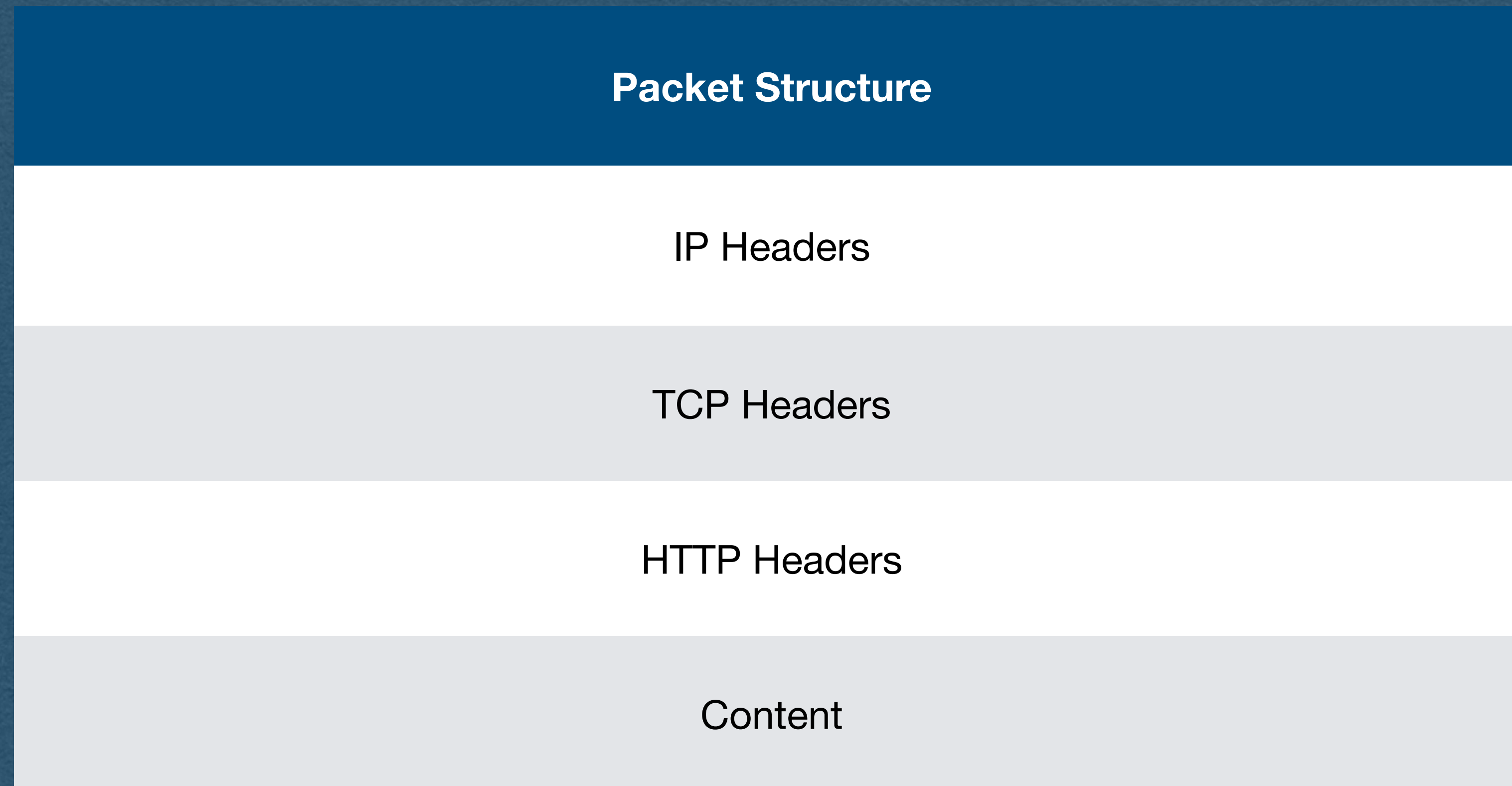


HTTP Overview

Network Stack (A simplified view)

- Enter HTTP



HTTP - Documentation

- HTTP/1.1 is defined by RFC2616 of the IETF
 - <https://tools.ietf.org/html/rfc2616>
 - This is THE document for all your questions about HTTP
 - Today we'll discuss topics in sections 4, 5, and 6
- RFC
 - Request For Comments
 - Submit an RFC for public discussion or to publish information
- IETF
 - Internet Engineering Task Force
 - Adopts some RFC's as Internet standards

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

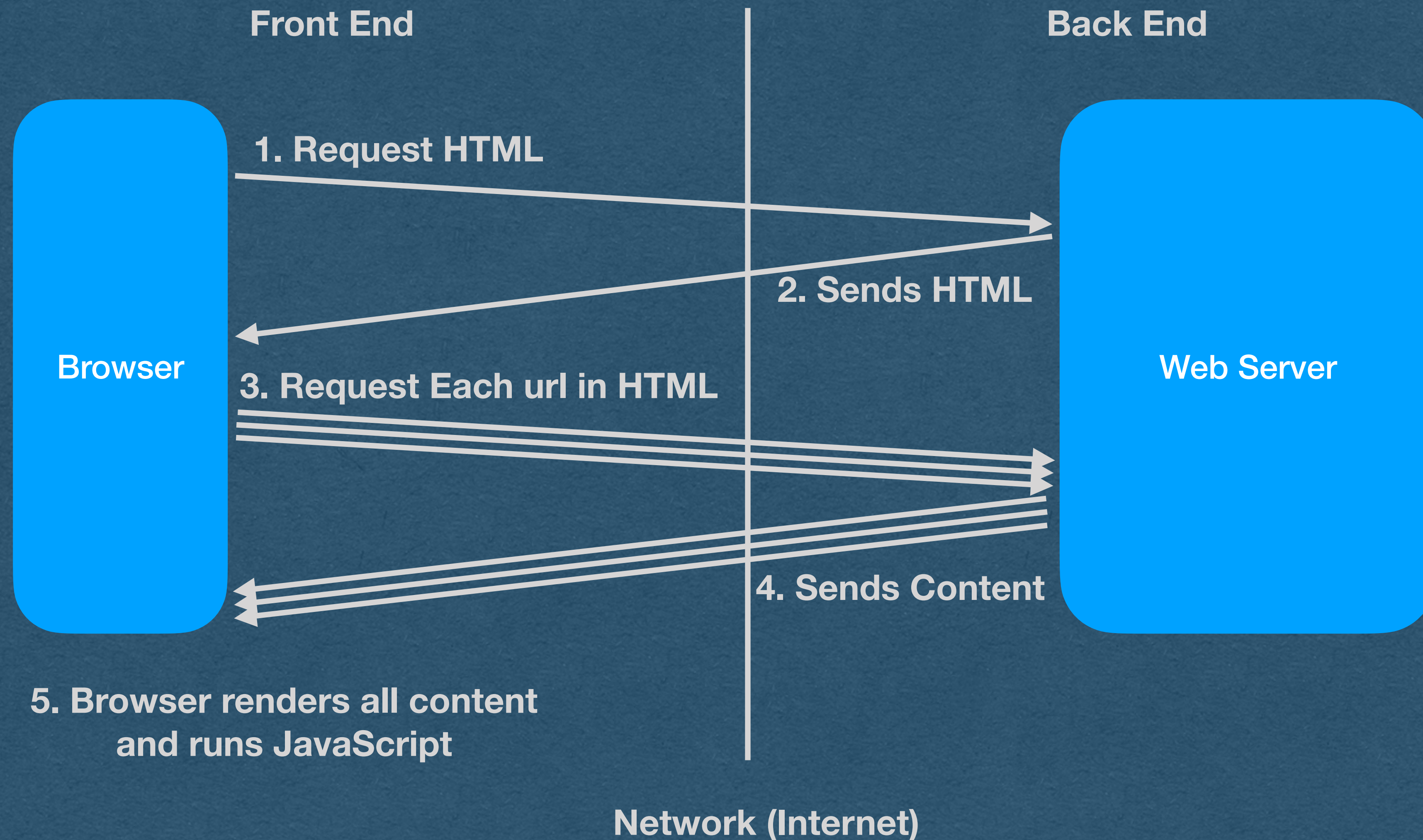
HTTP

- 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)
 - The front end will send HTTP requests to the back end
 - Back End - The web server and all software that does not run on the user's machine
 - The back end will will send HTTP responses to the front end

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
 - PATCH: Partial update of existing data
 - DELETE: Delete information from a service
 - HEAD: Request only the headers of a response
- To start, we'll focus on GET and POST only

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

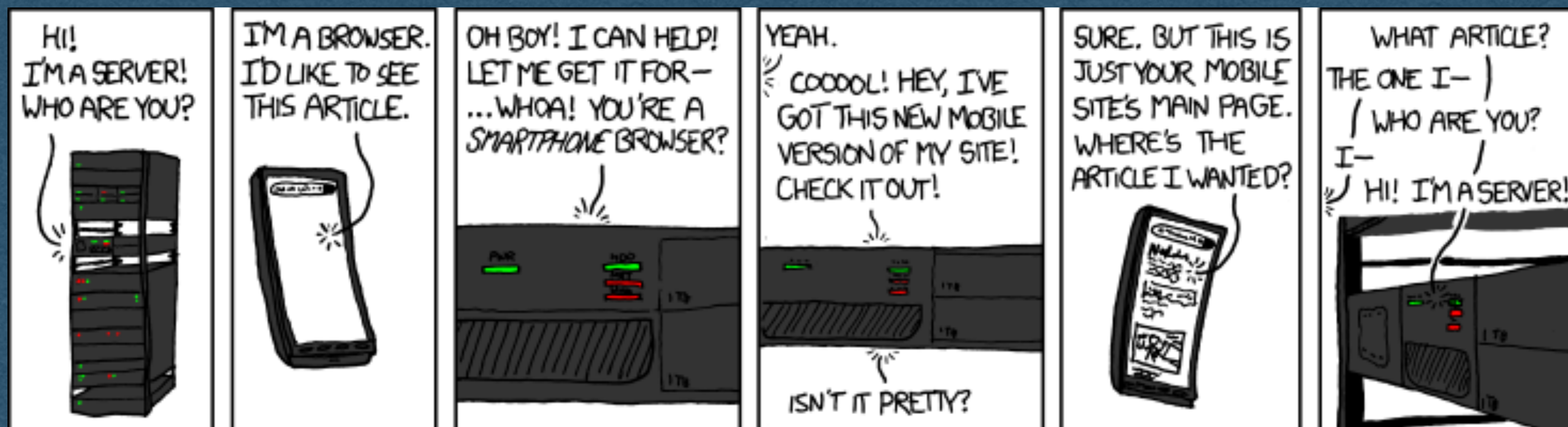
HTTP Request

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

- 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
- When handling an HTTP request, do not have to care who sent it



<https://xkcd.com/869/>

New Lines

- A new line character in an HTTP request/response must be:
 - `"\r\n"`
 - Carriage return (From the days of typewriters)
 - New line
 - In the documentation this is referred to as a CRLF
 - CRLF == Carriage Return Line Feed
- Be aware of this while parsing
- Use `"\r\n"` for new lines when preparing your responses

HTTP GET Request

GET Request

- We'll use this simple request as an example

```
GET / HTTP/1.1  
Host: cse312.com  
Connection: keep-alive  
Accept-Language: en-US,en
```

GET Request

- More accurately, it will be this

```
GET / HTTP/1.1\r\nHost: cse312.com\r\nConnection: keep-alive\r\nAccept-Language: en-US,en\r\n\r\n
```

- In the slides, we'll render "\r\n" as a new line
- Note that there is a blank line at the end of the request

The Request Line

- The first line of the request is always the request line
- The request line has 3 values separated by spaces
 - The request type (GET/POST/PUT/DELETE/etc)
 - The path of the request (ex. "/") - Everything after the port in the requested url
 - The HTTP Version
 - We'll always use HTTP/1.1 in this course

GET / HTTP/1.1

Host: cse312.com

Connection: keep-alive

Accept-Language: en-US,en

The Request Line

- Parse the request line by looking for the 2 space characters
 - Separate the values and check the strings
- Typically: When the root path "/" is requested, serve the HTML of your home page
 - By convention, web servers look for index.html to serve at the root path
- If the url contains a different path, it will appear in the request line

GET / HTTP/1.1

Host: cse312.com

Connection: keep-alive

Accept-Language: en-US,en

<http://cse312.com/>

GET /static_files/slides/1_2_HTTP.pdf HTTP/1.1

Host: cse312.com

Connection: keep-alive

Accept-Language: en-US,en

http://cse312.com/static_files/slides/1_2_HTTP.pdf

Headers

- Following the request line are any number of headers
- HTTP Headers
 - Key-Value pairs
 - Key and value separated by a colon ":"
- Each header will be on a new line (look for \r\n)
- To parse, look for the colon ":" and read the key and value
 - There is an optional space after the colon which should be removed if present <-- don't forget to do this! Leaving in the space will cause bugs that are very difficult to detect

GET / HTTP/1.1

Host: cse312.com

Connection: keep-alive

Accept-Language: en-US,en

HTTP Response

Response

- Your web server will listen for HTTP requests over the TCP sockets and respond with HTTP responses
- Send a response back to the client to serve them the requested content
- This response will display "hello" in their browser

```
HTTP/1.1 200 OK  
Content-Type: text/plain  
Content-Length: 5
```

```
hello
```

Response

- More accurately, this is the response

```
HTTP/1.1 200 OK\r\nContent-Type: text/plain\r\nContent-Length: 5\r\n\r\nhello
```

Status Line

- The first line of the response must be the status line
- Status line contains 3 values separated by spaces
 - The HTTP version
 - The status code
 - The status message (Reason phrase in docs)

HTTP/1.1 200 OK

Content-Type: text/plain

Content-Length: 5

hello

Response Codes

- Tells the browser the nature of the response
 - 200-level codes: Everything went well
 - 300-level codes: Redirect the request
 - 400-level codes: Error caused by the client
 - 500-level codes: Error caused by the server
- Include a human readable message

```
HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 5

hello
```

Response Headers

- The headers in the response follow the same format as request headers
- Should have at least two headers
 - Content-Type - Tells the browser how to parse this content
 - Content-Length - How many **bytes** should be read from the body of the response

HTTP/1.1 200 OK

Content-Type: text/plain

Content-Length: 5

hello

Blank Line

- There is a blank line (`\r\n\r\n`) separating the headers and the body of the response
- The blank line is very important
 - It tells the browser that you're done with headers and the next bytes will be the body of the response

```
HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 5
hello
```

Body

- After the blank line are the bytes of the body
- The body contains the content that is being served
- The number of bytes in the body exactly match the value of the Content-Length header
- The browser will read exactly this many bytes
- Note: There is no required format for the body. The format depends on the type of content being served

```
HTTP/1.1 200 OK  
Content-Type: text/plain  
Content-Length: 5
```

```
hello
```

404 Not Found

- If a path is requested that your server does not handle
 - Respond with a 404 Not Found
 - Note: Spaces are allowed in your reason message
- The response format is the same as a 200 response
 - Include content type and length
 - Include a body that will be displayed to the client

HTTP/1.1 404 Not Found

Content-Type: text/plain

Content-Length: 36

The requested content does not exist