Poll Everywhere

pollev.com/cse333

Which concept gave you the most difficulty in the context of Homework 3?

A. Understanding the index file layout
B. C++ Classes & Inheritance
C. C++ STL
D. Query Processor Algorithm
E. Debugging/GDB
F. Style considerations
G. Prefer not to say

Hypertext Transfer Protocol

CSE 333 Summer 2023

Instructor: Timmy Yang

Teaching Assistants:

Jennifer Xu Leanna Nguyen Pedro Amarante

Sara Deutscher Tanmay Shah

Relevant Course Information (1/2)

- Exercise 10 due Monday (8/7)
  - Client-side programming
- Exercise 11 due Thursday (8/10)
  - Server-side programming
  - Can use ex10 client solution to send messages to ex11 server

- Homework 4 due Wednesday (8/16)
  - Files going out late today (evening, 9-10pm at the latest)
  - Partnership form released, closes Thursday (8/10) @ 11:59pm
  - Can still use 2 late days for hw4 (hard deadline of 8/18)
  - Part of section next week will cover tools for debugging hw4

Relevant Course Information (2/2)

- Quiz 3
  - Open Monday (8/7) @ 2pm to Wednesday (8/9) @ 11:59pm
  - Will include questions about:
    - Exercise 7, 8 and 9
    - Homework 3
- Quiz 4
  - Open Wednesday (8/16) @ 2pm to Friday (8/18) @ 11:59pm
  - Will include questions about:
    - Exercise 10, 11, and 12
    - Homework 4
    - Course wrap-up

Homework 4 Summary

- Build a Multithreaded Web Server (333gle)
  - You will host the querying service that you built in your previous homework on a web server

files on disk proj-docs/ in repo

.idx files from HW3

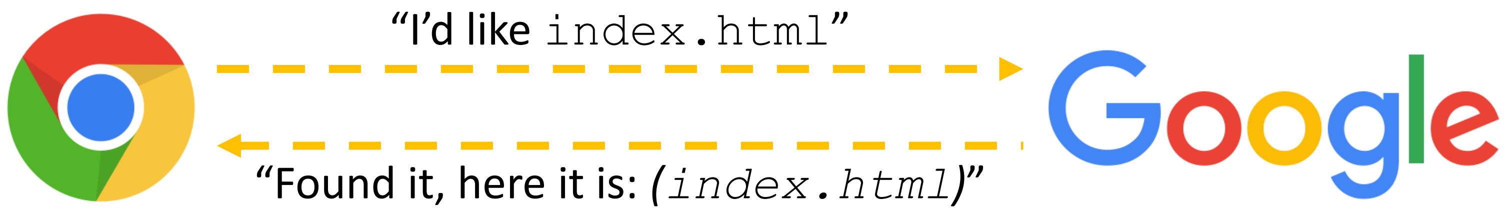
- Running your server
  - `./http333d <port> <static files> <unit indices>`
  - Static files are the files on disk corresponding to our index files
  - You (and others) can access it on any browser now!

- Implementation
  - Using network protocols to communicate between client/server
  - Handling some additional security flaws
  - ☆ Note: Multithreading is already implemented for you

Client and Server Communication

- Lecture 19 (Client-side and Server-side Networking) has already shown how to to this in C/C++
  - `sendreceive.cc` and `server_accept_rw_close.cc`

- This is what actually happens on the web!
  - Clients establish a stable TCP connection the server
  - Lots of bytes are interchanged/processed between each other



Case Study of Protocols: HTTP

- A **protocol** defines a set of rules governing the **format** and **exchange** of messages in a computing system
  - Syntax: The formatting or grammar of the system
  - Semantics: What messages are being exchanged
  - Allows everyone be on the same page of communication

- **H**yper**t**ext **T**ransfer **P**rotocol: Request/Response Protocol
  - HTTP defines how we should send information between a client and a server
  - A **request** will send a message to the server (about anything)
  - A **response** will process and respond to that message
  - And it's human readable! $\longrightarrow$ HTTP 1.1

Requests: Client Sending Messages

- A client wants to talk to a server about something
  - Initiates a conversation (establish or using existing connection)
  - Generally, this is for retrieving a resource, using **Uniform Resource Identifier (URI)** ↳ URL - Uniform Resource Locator

- Standard Syntax:

/index.html 1.1

```
[METHOD] [request-uri] HTTP/[version]\r\n
[headerfield1]: [fieldvalue1]\r\n
[headerfield2]: [fieldvalue2]\r\n
[...]
[headerfieldN]: [fieldvalueN]\r\n
\r\n
[request body, if any]
```

2x CRLF splits headers and req. body

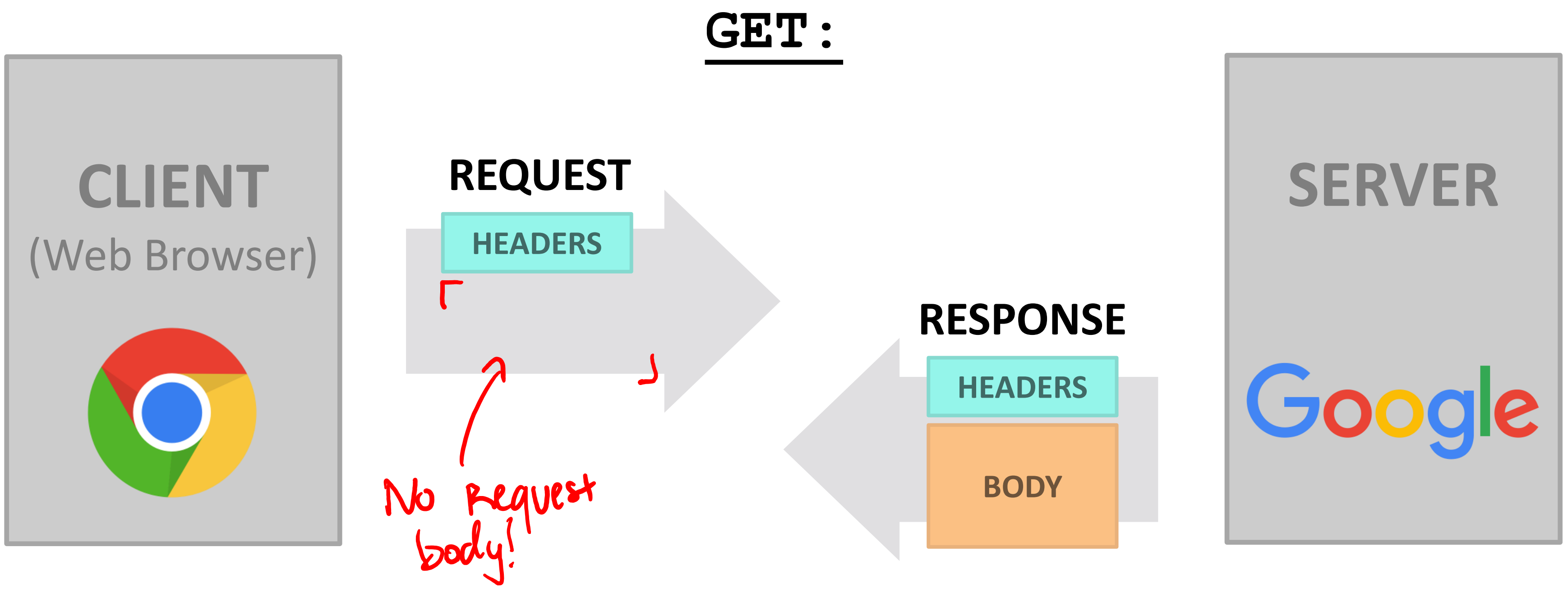
carriage return

new line or "line feed)

CRLF

HTTP Methods

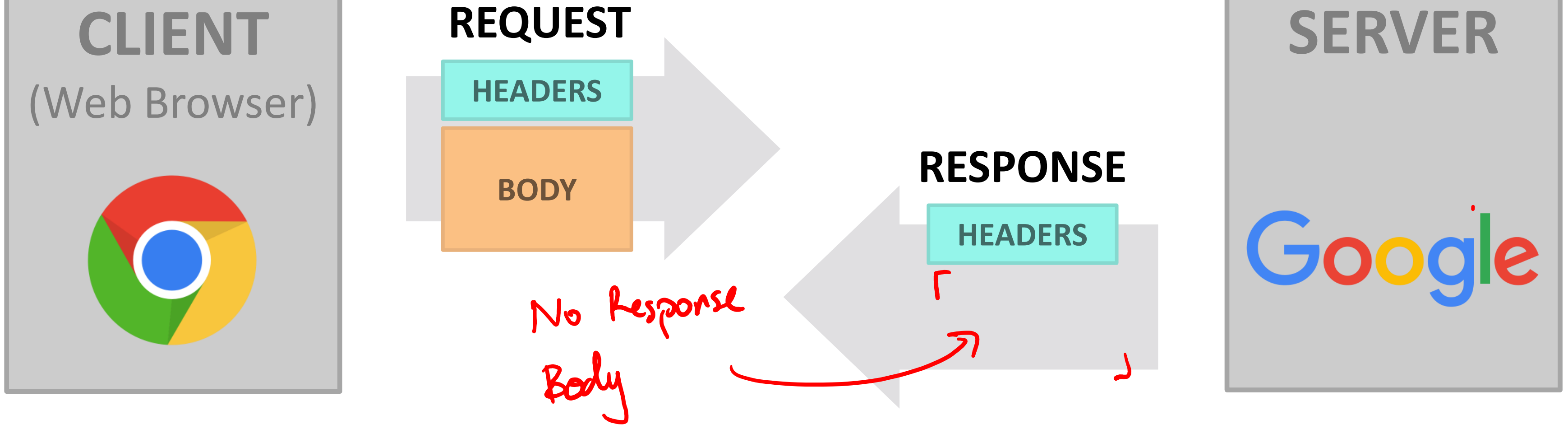
- There are three commonly-used HTTP methods:
  - `GET`: "Please send me the named resource"



HTTP Methods

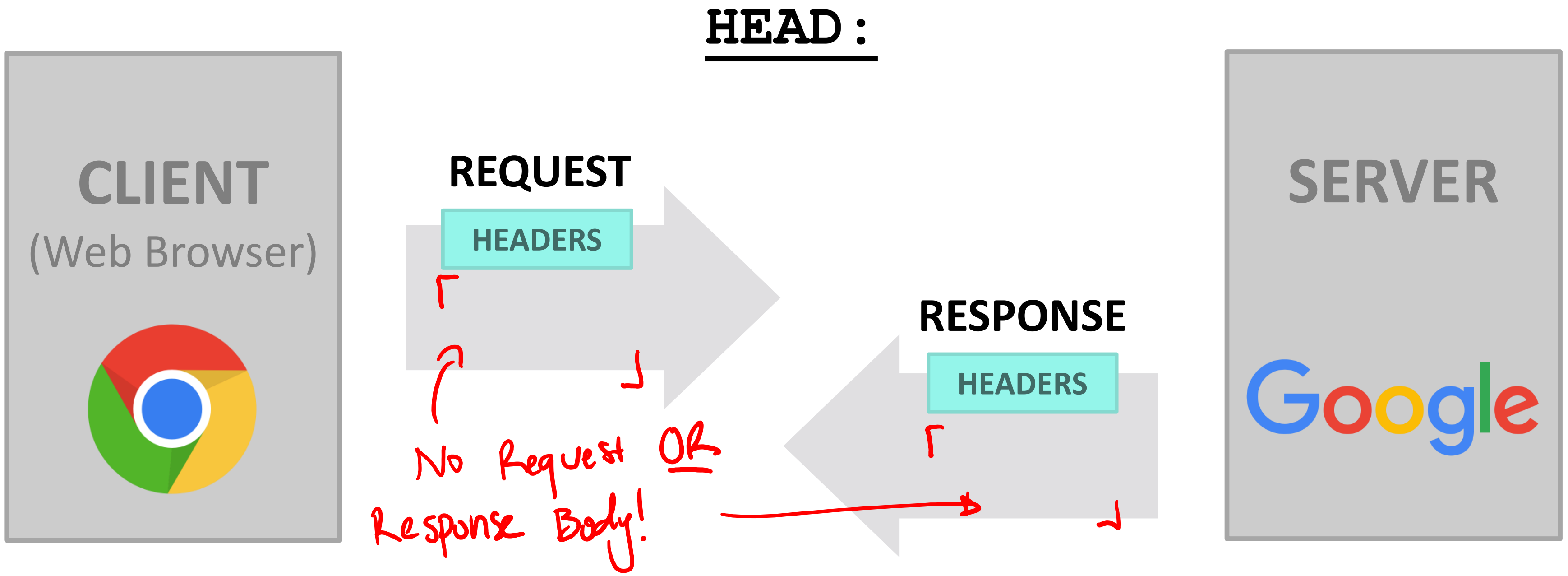
- There are three commonly-used HTTP methods:
  - `GET`: "Please send me the named resource"
  - **`POST`**: "I'd like to submit data to you" (*e.g.* file upload)

`POST:`



HTTP Methods

- There are three commonly-used HTTP methods:
  - `GET`: "Please send me the named resource"
  - `POST`: "I'd like to submit data to you" (*e.g.* file upload)
  - **`HEAD`**: "Send me the headers for the named resource"
    - Doesn't send resource; often to check if cached copy is still valid



HTTP Methods

- There are three commonly-used HTTP methods:
  - `GET`: "Please send me the named resource"
  - `POST`: "I'd like to submit data to you" (*e.g.* file upload)
  - `HEAD`: "Send me the headers for the named resource"
    - Doesn't send resource; often to check if cached copy is still valid

- Other methods exist, but are much less common:
  - `PUT`, `DELETE`, `TRACE`, `OPTIONS`, `CONNECT`, `PATCH`, . . .
    - For instance: `TRACE` – "show any proxies or caches in between me and the server"
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

Client Headers

- The client can provide one or more request "headers"
  - These provide information to the server or modify how the server should process the request

- You'll encounter many in practice
  - `Host:` the DNS name of the server
  - `User-Agent:` an identifying string naming the browser
  - `Accept:` the content types the client prefers or can accept
  - `Cookie:` an HTTP cookie previously set by the server
  - https://www.rfc-editor.org/rfc/rfc2616.html#section-5.3

A Real Request

```
GET / HTTP/1.1
Host: attu.cs.washington.edu:3333
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/66.0.3359.181 Safari/537.36
Cookie:
SESS0c8e598bbe17200b27e1d0a18f9a42bb=5c18d7ed6d369d56b69a1
c0aa441d
...
```

- Demo: use `nc` to see a real HTTP request

Response: Server Responding

- A server parses and sends a response to a user
  - Indicate how the server processed the request (accepted or not)
  - Send requested resource back to the client

- General form:

1.1   ex. 404   ex. "Not Found"

```
HTTP/[version] [status code] [reason]\r\n
[headerfield1]: [fieldvalue1]\r\n
[headerfield2]: [fieldvalue2]\r\n
[...]
[headerfieldN]: [fieldvalueN]\r\n
\r\n
[response body, if any]
```

404
400 403

Status Codes and Reason

- *Code*: numeric outcome of the request – easy for computers to interpret
  - A 3-digit integer with the 1st digit indicating a response category
    - `1xx`: Informational message
    - `2xx`: Success
    - `3xx`: Redirect to a different URL
    - `4xx`: Error in the client's request
    - `5xx`: Error experienced by the server

400 "Bad Request"
403 "Forbidden"
404 "Not Found"

- *Reason*: human-readable explanation
  - *e.g.* "OK" or "Moved Temporarily"

☆ https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

Common Statuses

- `HTTP/1.1 200 OK`
  - The request succeeded and the requested object is sent
- `HTTP/1.1 404 Not Found`
  - The requested object was not found
- `HTTP/1.1 301 Moved Permanently`
  - The object exists, but its name has changed
    - The new URL is given as the "`Location:`" header value
- `HTTP/1.1 500 Server Error`
  - The server had some kind of unexpected error

Server Headers

- The server can provide zero or more response "headers"
  - These provide information to the client or modify how the client should process the response

- You'll encounter many in practice
  - `Server`: a string identifying the server software
  - `Content-Type`: the type of the requested object
  - `Content-Length`: size of requested object
  - `Last-Modified`: a date indicating the last time the request object was modified
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types

A Real Response

```
HTTP/1.1 200 OK
Date: Mon, 21 May 2018 07:58:46 GMT
Server: Apache/2.2.32 (Unix) mod_ssl/2.2.32
OpenSSL/1.0.1e-fips mod_pubcookie/3.3.4a mod_uwa/3.2.1
Phusion_Passenger/3.0.11
Last-Modified: Mon, 21 May 2018 07:58:05 GMT
Content-Length: 82
Content-Type: text/html
...
<html><body>
<font color="chartreuse" size="18pt">Awesome!!</font>
</body></html>
```

- Demo: Use `nc -C` to see real HTTP responses*
  - `-C` argument allows us to send carriage returns over `nc`

→ Mac OS

*may be `nc -c` on your system, use `man nc` to check options

Poll Everywhere

pollev.com/cse333

Which statement is FALSE about the HTTP/1.1 Protocol?

HyperText Transfer Protocol

A. HTTP is a standard communication protocol for the web

Request → Response Protocol

B. A client always sends a message first before the server

Only 1 URI per GET

C. An HTTP Request can only request one resource at a time

[✓ D.] An HTTP Response needs to have a response body

↳ POST Response

E. I'm not really sure…

HTTP/1.1 Protocol

- HTTP / 1.1 (1997) – The protocols accepted by **all current browsers and servers**
  - Built after HTTP/0.9 (1991) and HTTP/1.0 (1996)
  - Better performance, richer caching features, better support for multihomed servers, and much more
- "Chunked Transfer-Encoding" – Send responses in multiple pieces (`Transfer-Encoding: chunked`)
  - https://en.wikipedia.org/wiki/List_of_HTTP_header_fields#transfer-encoding-response-header
- Persistent Connections: TCP connections can handle multiple requests (`Connection: keep-alive`)

Improvements: HTTP/2 and HTTP/3

- Human-readable text protocols can only go so far…

- HTTP/2 (2015) was a push to optimize HTTP/1.1
  - Built off Google Project SPDY which aimed to reduce latency
  - Compressed headers and message body
  - Many larger companies quickly transitioned
  - https://en.wikipedia.org/wiki/HTTP/2

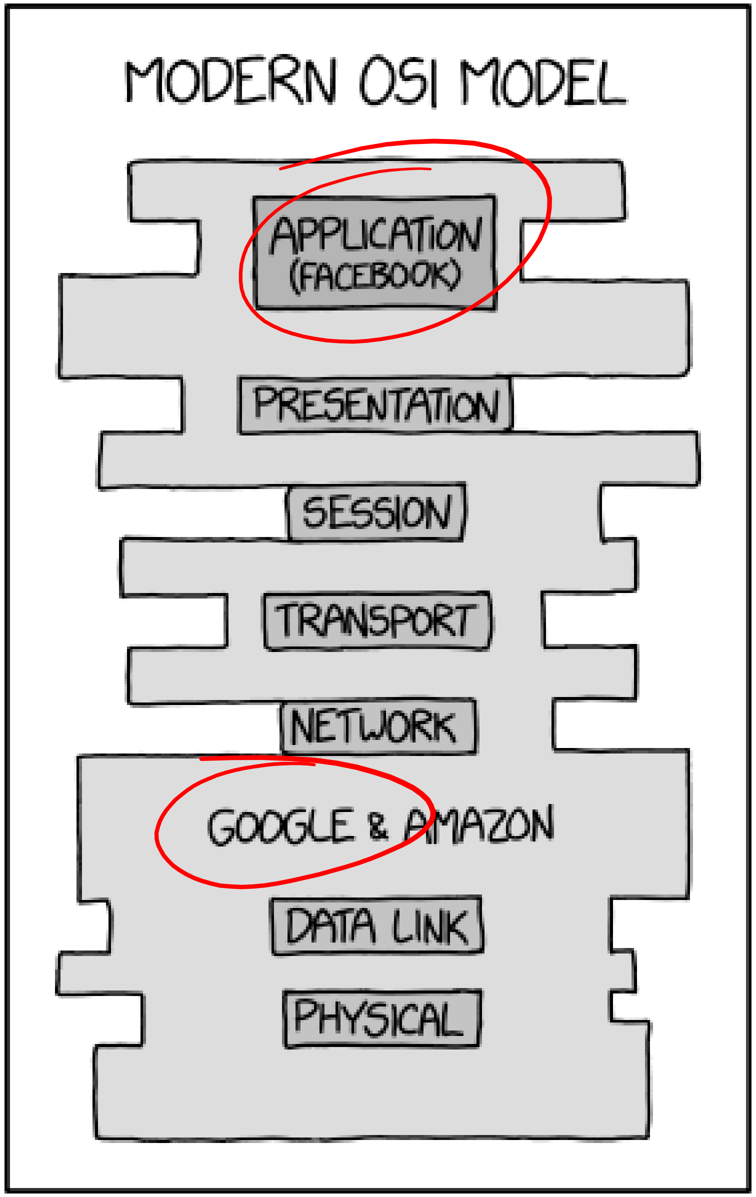
- HTTP/3 (2022) builds even more on HTTP/2
  - Mainly using UDP-based protocol called QUIC (holds a standard connection like TCP)
    spearheaded by Google!
  - https://en.wikipedia.org/wiki/HTTP/3

Slow to Change: HTTP Protocols

- HTTP/1.1 is still used today (1996 – Present)
  - https://almanac.httparchive.org/en/2022/http#fig-1
  - ~20% of requests still use HTTP/1.1
  - https://almanac.httparchive.org/en/2019/http#fig-3
  - Down from ~50% of requests 4 years ago

- Why is the transition taking so long?
  - Lack of knowledge about HTTP/2+
  - A good portion of web servers are still using HTTP/1.1
  - It takes engineering work to support a new HTTP protocol
  - HTTP/1.1 is human readable
  - Amongst more…

In Other Words…



Network infrastructure is heavily influenced by Tech Giants

https://xkcd.com/2105/

Extra Exercise #1

- Write a program that:
  - Creates a listening socket that accepts connections from clients
  - Reads a line of text from the client
  - Parses the line of text as a DNS name
  - Connects to that DNS name on port 80
  - Writes a valid HTTP request for "/"

```
GET / HTTP/1.1\r\n
Host: <DNS name>\r\n
Connection: close\r\n
\r\n
```

  - Reads the reply and returns it to the client