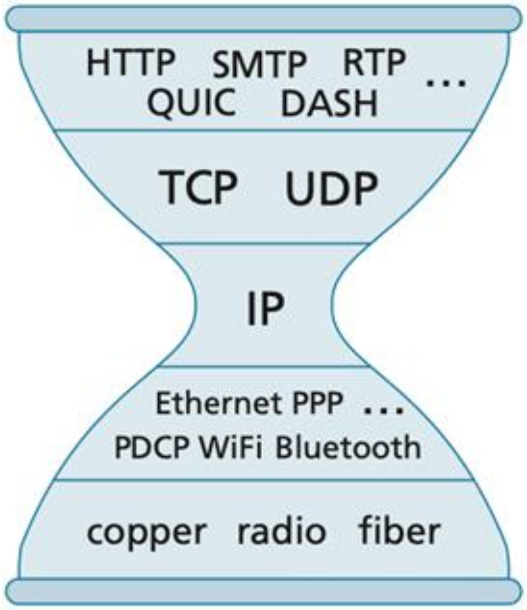
Networking and HTTP

COS 316 Precept 4

Layers of the network



Application layer

Transport layer

Network layer

Link layer

Physical layer

The Physical Layer

Deliver raw unstructured data (bits) over a physical medium

Functionality example: representation of bits

0 and 1 bits ⇔ voltage levels in circuits, pulse in optical medium, frequency and amplitudes in microwaves

Hardware examples:

Network interface card (NIC): the physical port on your computer

Cables: ethernet cables, fiber optical cables

The Link Layer

Device-to-device delivery

Devices that are directly connected

(e.g. laptop to wifi access point, router to router, machine to switch, etc.)

- **Organize bits into meaningful chunks**

The Physical Layer sends a constant stream of bits. The Link Layer breaks this stream into manageable data units called **Frames**.

- **Ensure the chunk get to the right device**

Medium access control (MAC) address: unique per device, permanent, assigned to NIC

- **Medium access control**

Reduce collision on shared medium

The Network Layer

End-to-end delivery over networks

e.g. my laptop to my mom’s phone, my laptop to a server in Virginia, etc.

· **IP Address**

Hierarchical, changeable, assigned by software (e.g. 192.168.10.1)

· **Routing protocols**

Figure out the path from source to destination

Routers decide which direction to send a packet based on its routing table.

· **Connectionless communication**

Best-effort

The Transport Layer

Process-to-process delivery

e.g. zoom on my laptop to zoom on Wyatt’s laptop

	TCP	UDP
Connection	connection-oriented	connectionless
Reliability	ordered, reliable	no guarantee
Weight	heavyweight	lightweight
Example apps	email, web, file transfer	video conference, online gaming

The Application Layer

HTTP

SMTP

FTP://

mpeg-DASH

QUIC

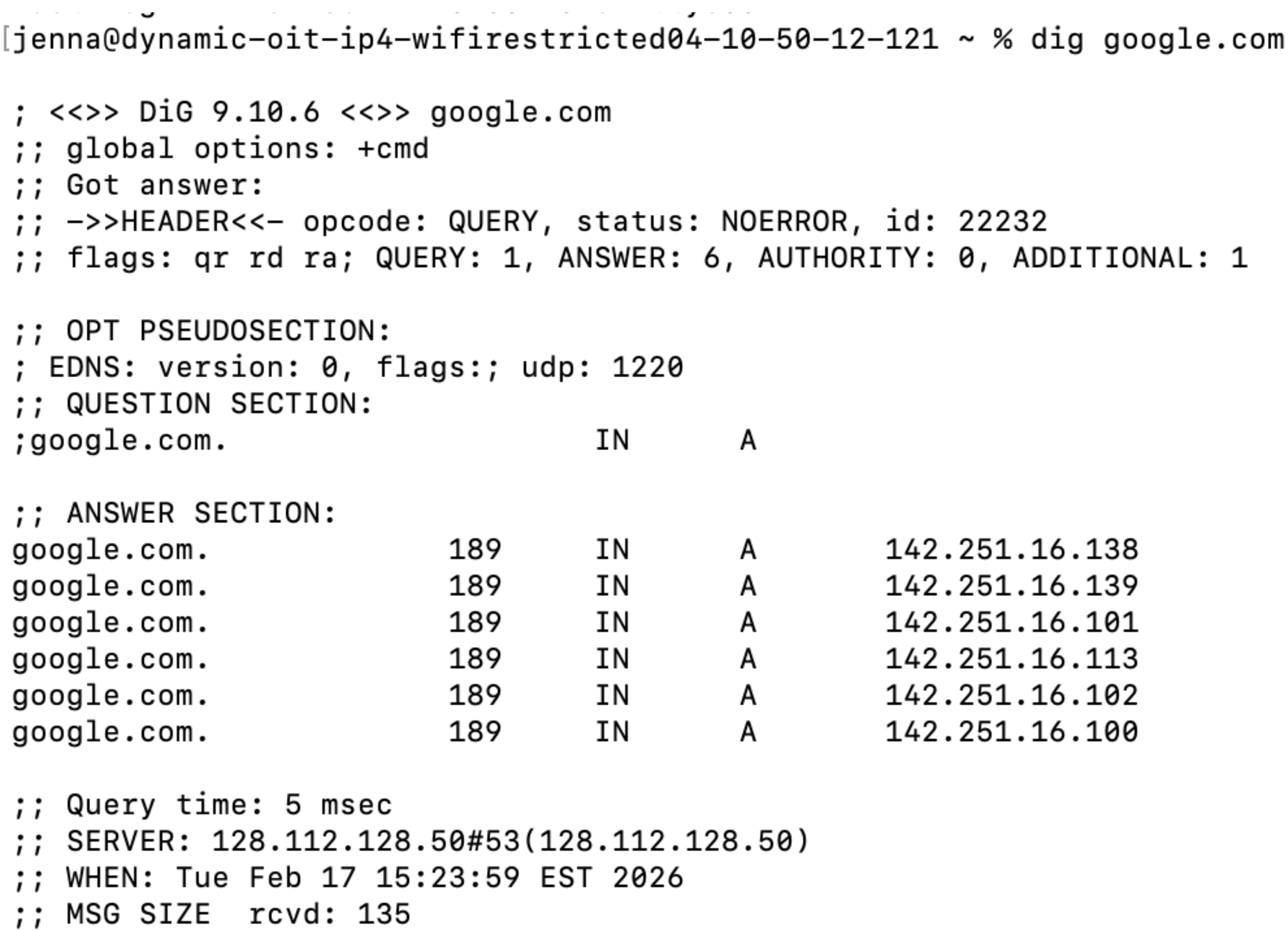
Example: DNS

Translate human readable website address to IP adress

e.g. https://example.com -> 1.1.1.1

· Use UDP as its transport protocol

speed is critical

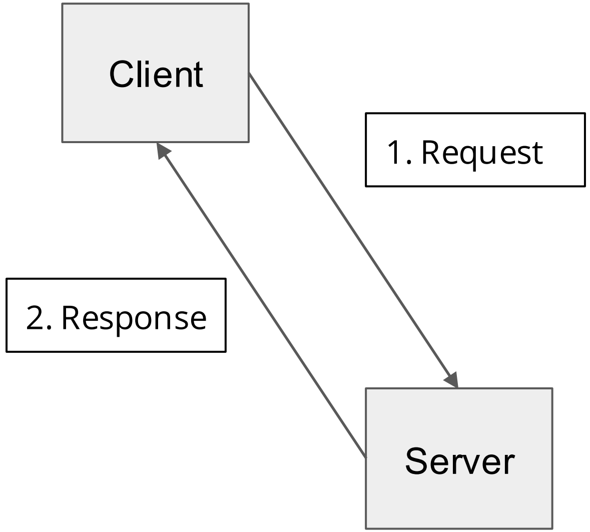


```
[jenna@dynamic-oit-ip4-wifirestricted04-10-50-12-121 ~ % dig google.com

; <<>> DiG 9.10.6 <<>> google.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 22232
;; flags: qr rd ra; QUERY: 1, ANSWER: 6, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1220
;; QUESTION SECTION:
;google.com.                    IN      A

;; ANSWER SECTION:
google.com.             189     IN      A       142.251.16.138
google.com.             189     IN      A       142.251.16.139
google.com.             189     IN      A       142.251.16.101
google.com.             189     IN      A       142.251.16.113
google.com.             189     IN      A       142.251.16.102
google.com.             189     IN      A       142.251.16.100

;; Query time: 5 msec
;; SERVER: 128.112.128.50#53(128.112.128.50)
;; WHEN: Tue Feb 17 15:23:59 EST 2026
;; MSG SIZE  rcvd: 135
```

Questions?

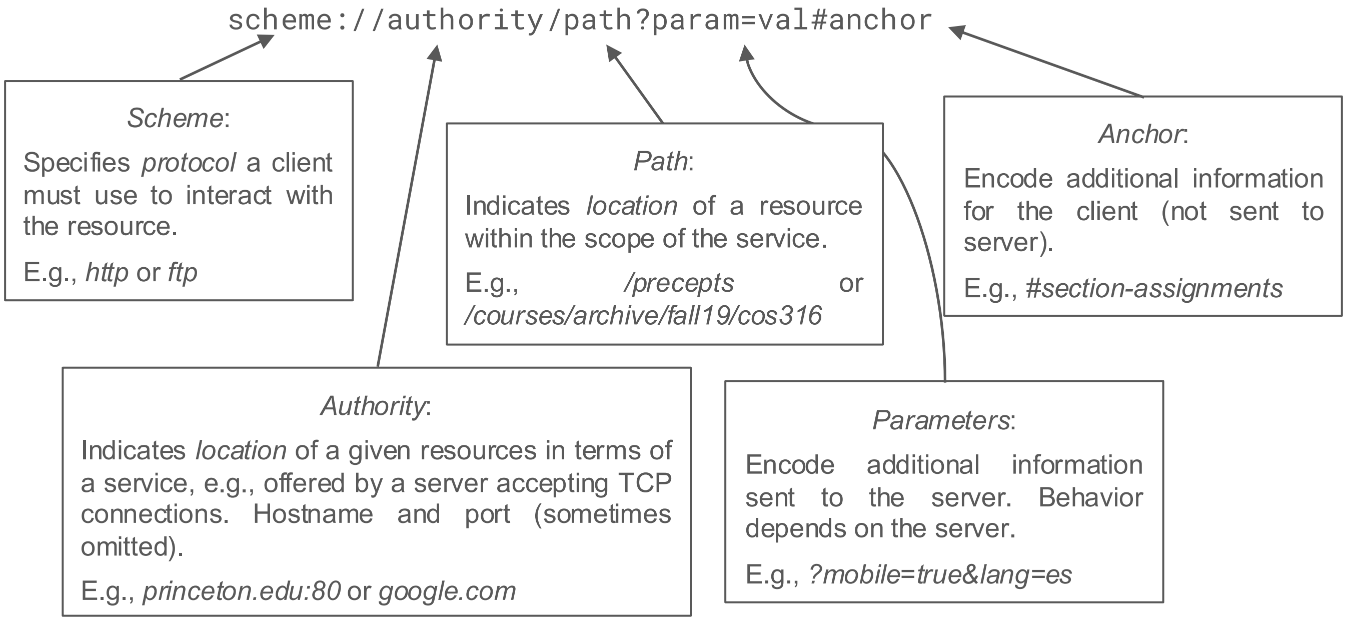
Overview of HTTP

- **H**yper**T**ext **T**ransfer **P**rotocol
  - Used to distribute *hypertext* over the Internet (i.e., *HT*ML web pages)
  - Relies on a bidirectional stream protocol underneath → ***TCP!***
- Unit of operation: **request+response pairs**
  - Establish a connection from client to server
  - Client: send *HTTP request* to server
  - Server: send HTTP *response* to client
- Stateless protocol
  - No mandatory state maintained beyond a request+response operation
  - Server & client can cooperate to maintain application state, e.g., through *cookies*
- Standardized through a series of *RFC*s → overview of applicable standards



URLs

- Uniform Resource Locator
  - uniquely identifies a given resource on the web
- Syntax:



`scheme://authority/path?param=val#anchor`

Scheme:

Specifies *protocol* a client must use to interact with the resource.

E.g., *http* or *ftp*

Path:

Indicates *location* of a resource within the scope of the service.

E.g., */precepts* or */courses/archive/fall19/cos316*

Anchor:

Encode additional information for the client (not sent to server).

E.g., *#section-assignments*

Authority:

Indicates *location* of a given resources in terms of a service, e.g., offered by a server accepting TCP connections. Hostname and port (sometimes omitted).

E.g., *princeton.edu:80* or *google.com*

Parameters:

Encode additional information sent to the server. Behavior depends on the server.

E.g., *?mobile=true&lang=es*

Examples:

http://www.ietf.org/rfc/rfc959.txt

http://xyz.org:8081/route/subroute

http://www.ietf.org/rfc/rfc959.txt

mailto:ak18@cs.princeton.edu

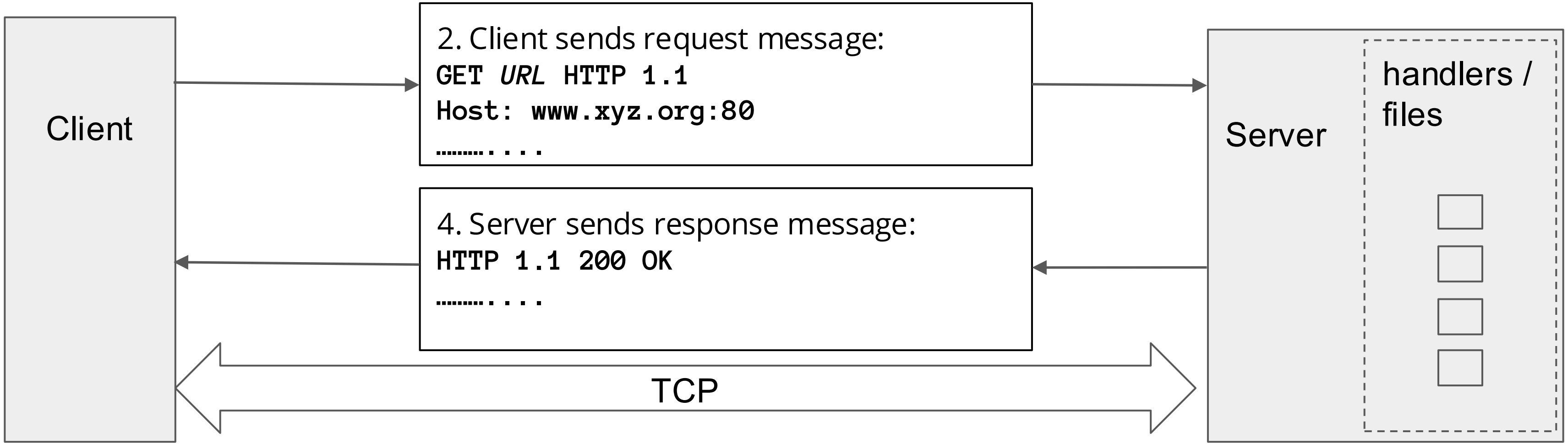
ftp://tug.ctan.org/pub

rtsp://192.168.0.164/axis-media/media.amp

HTTP Example

1. Client requests URL:
`http://www.xyz.org:80/path/file`

3. Server routes request to the appropriate handler/file



2. Client sends request message:

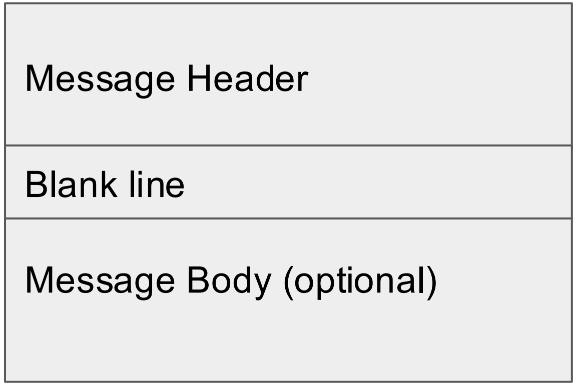
```
GET URL HTTP 1.1
Host: www.xyz.org:80
………. . . .
```

4. Server sends response message:

```
HTTP 1.1 200 OK
………. . . .
```

5. Client processes response

HTTP Request and Response Messages

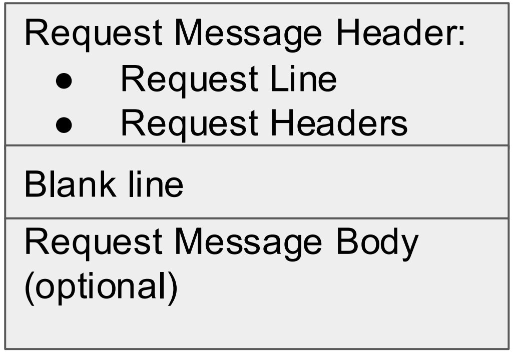


Message Header

Blank line

Message Body (optional)

HTTP Request Message



Request Message Header:

- Request Line
- Request Headers

Blank line

Request Message Body (optional)

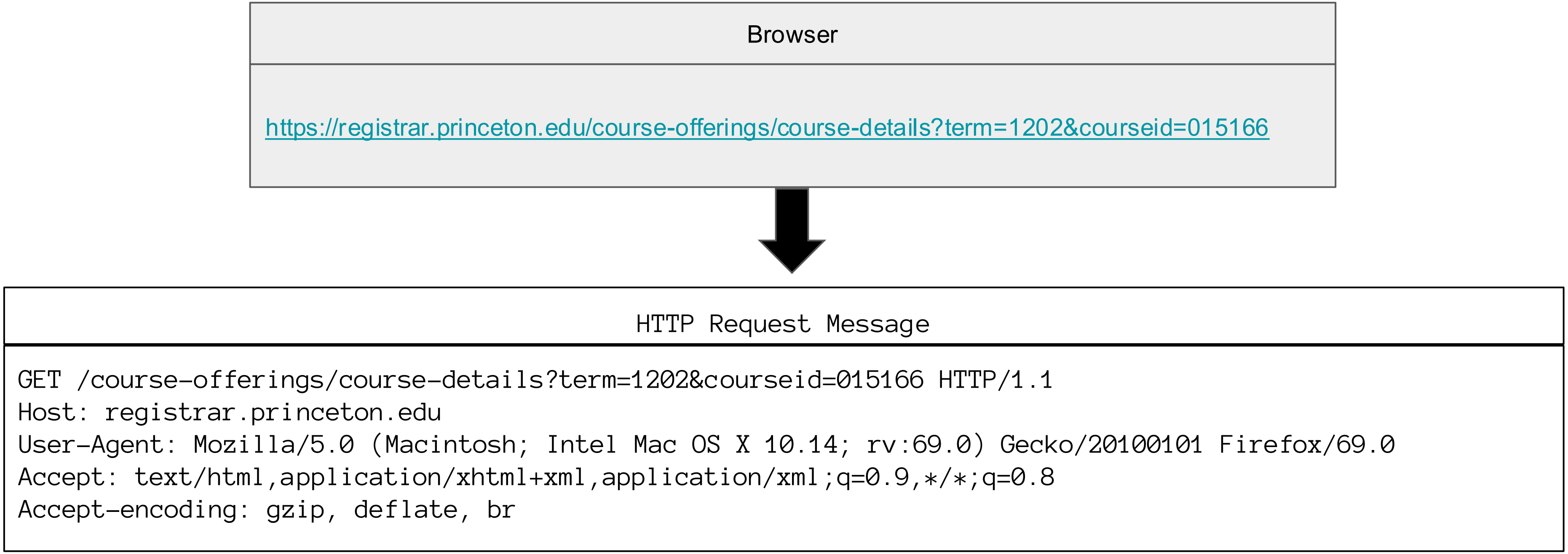
- Request Line
  - **[request-method-name] [request-URI] [HTTP-version]**
  - request-method-name: *HTTP verb*
    - GET, HEAD, POST, etc.
  - request-URI:
    - Name of resource (route) requested
  - HTTP-version:
    - HTTP/1.0, HTTP/1.1 or HTTP/2.0
- Request Header
  - Consists of name:value pairs
  - Multiple values, separated by commas
  - request-header-name: request-header-value1, request-header-value2, ...
- Examples

```
Host: www.xyz.com
Connection: Keep-Alive
Accept: image/gif, image/jpeg, */*
Accept-Language: us-en, fr, cn
```

HTTP Request Methods (*verbs*)

- Common methods
  - GET
    - retrieve a resource from the server
  - HEAD
    - return only the headers of GET response
  - POST
    - create a resource on the server (client sends resource in the request body)
- Case Sensitive

HTTP Request Message



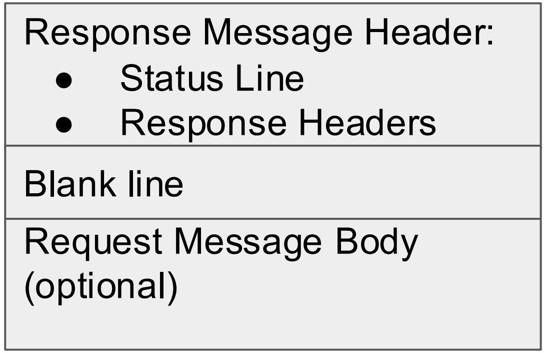
Browser

https://registrar.princeton.edu/course-offerings/course-details?term=1202&courseid=015166

HTTP Request Message

```
GET /course-offerings/course-details?term=1202&courseid=015166 HTTP/1.1
Host: registrar.princeton.edu
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.14; rv:69.0) Gecko/20100101 Firefox/69.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-encoding: gzip, deflate, br
```

HTTP Response Message



Response Message Header:

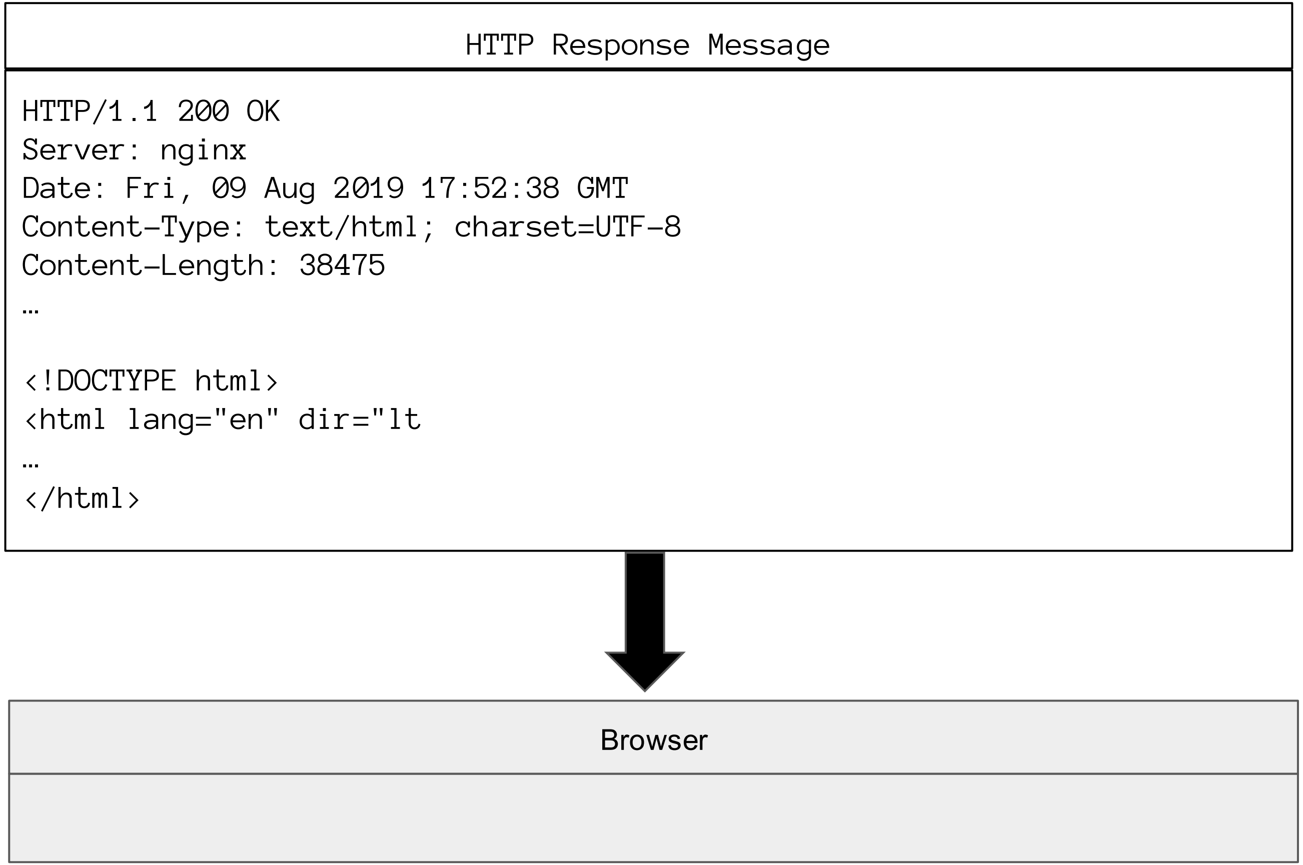
- Status Line
- Response Headers

Blank line

Request Message Body (optional)

- Status Line
  - **[HTTP-version] [status-code] [reason-phrase]**
    - HTTP-version: HTTP version used in this session e.g., HTTP/1.0,HTTP/1.1,HTTP2.0
    - status-code: 3-digit response code
    - reason-phrase: short explanation for status code
    - Common status-code and reason-phrases are
      - "200 OK"
      - "404 Not Found"
    - Examples
      - HTTP/1.1 200 OK
      - HTTP/1.0 404 Not Found
- Response Headers
  - Multiple values, separated by commas
      - response-header-name: response-header-value1, response-header-value2, …
  - Examples
    - Content-Type: text/html
    - Content-Length: 35
    - Keep-Alive: timeout=15, max=10
- Response Message Body
  - Data requested, e.g., HTML+CSS+JavaScript

HTTP Response Message



HTTP Response Message

```
HTTP/1.1 200 OK
Server: nginx
Date: Fri, 09 Aug 2019 17:52:38 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 38475
…

<!DOCTYPE html>
<html lang="en" dir="lt
…
</html>
```

HTTP/2

- Features
  - is binary, instead of textual
  - is fully *multiplexed*, instead of ordered and blocking
  - can therefore use one connection for parallelism
  - uses header compression to reduce overhead
  - allows servers to “push” responses proactively into client caches
- IETF Standard
  - https://httpwg.org/specs/rfc7540.html
- More on HTTP later in semester