

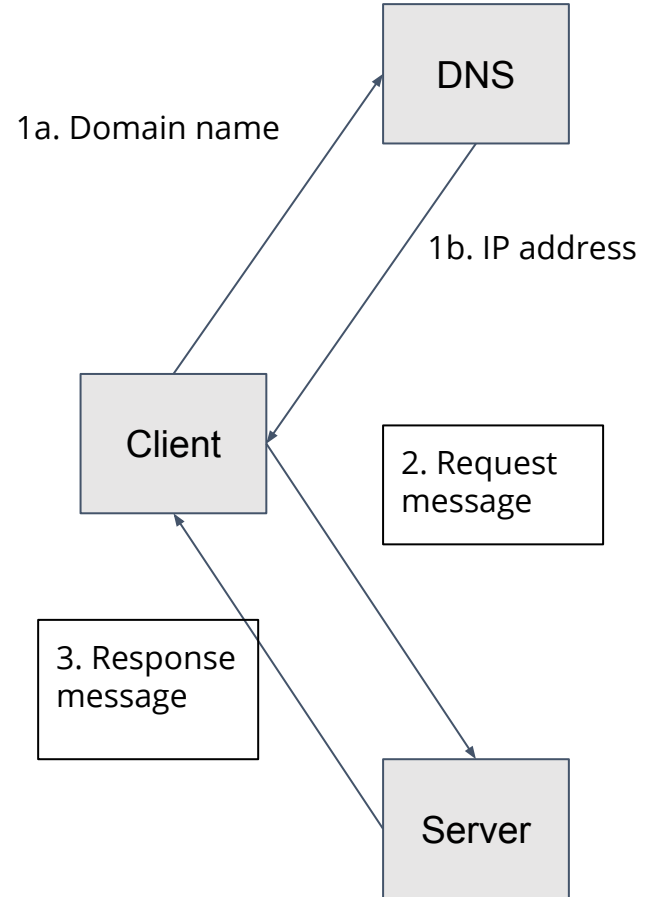
COS 316

Precept:

What is HTTP?

HTTP Overview

- **H**yper-**T**ext **T**ransfer **P**rotocol over bidirectional byte stream (e.g., TCP)
- Interaction
 1. Client looks up IP of server (DNS)
 2. Client sends request to server
 3. Server responds with data or error
- Requests/responses are encoded in text
- Stateless
 - HTTP maintains no info about past client requests
 - HTTP *cookies* allow server to identify client and associate requests into a client session



HTTP 1.1 Standard

- <https://httpwg.org/specs/rfc7230.html>

HTTP Example

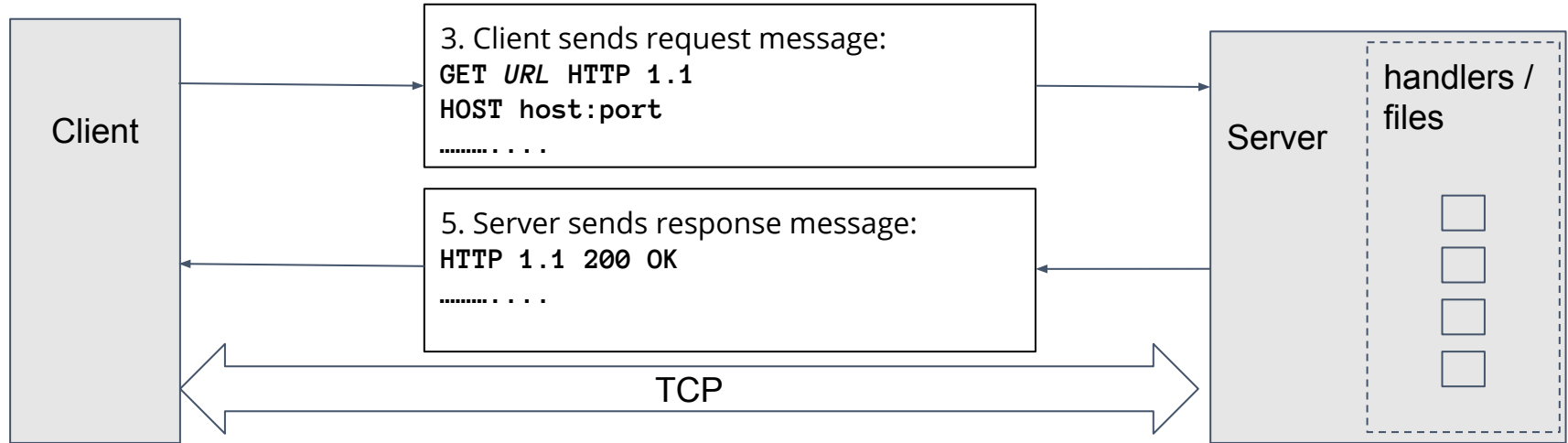
1. Client issues URL:

`http://www.xyz.org:80/path/file`

2. Domain resolved to IP address:

`http://10.11.16.10:80/path/file`

4. Server routes request to the appropriate handler/file



6. Client processes response

URLs

- Uniform Resource Locator
 - uniquely identifies a resource
- Syntax
 - protocol://host:port/path
- Protocol:
 - Application-level protocol used by the client and server, e.g., HTTP, FTP, and telnet
- Host:
 - DNS domain name (e.g., www.xys.org) or IP address (e.g., 192.128.1.2) of the server
- Port:
 - Port number server is listening for incoming requests from the clients

Examples

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

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

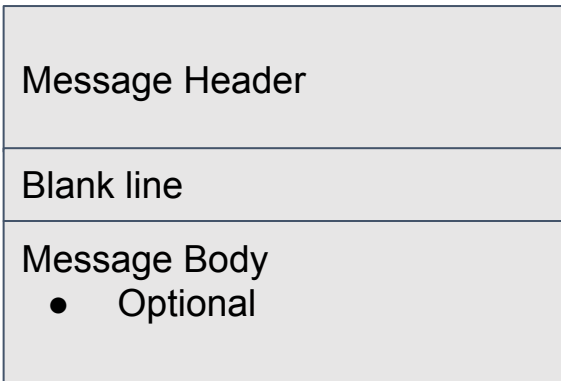
http://www.ietf.org/rfc/rfc959.txt

mailto:ak18@cs.princeton.edu

ftp://tug.ctan.org/pub

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

HTTP Request and Response Messages



HTTP Request Message

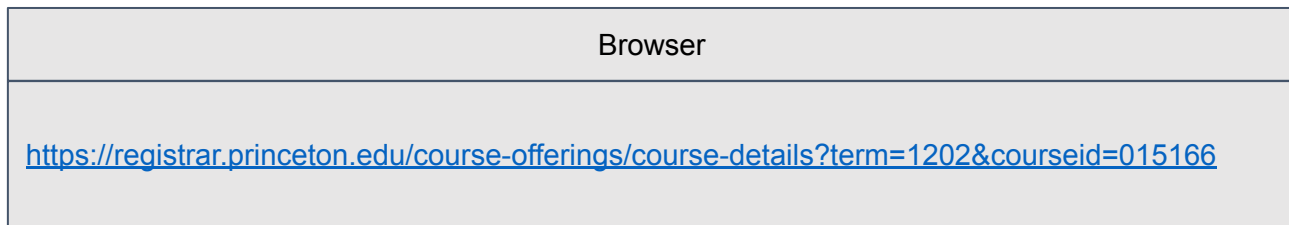
Request Message Header: <ul style="list-style-type: none">• Request Line• Request Headers
Blank line
Request Message Body <ul style="list-style-type: none">• Optional

- Request Line
 - request-method-name request-URI HTTP-version
 - request-method-name:
 - GET, HEADER, 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

- Common methods
 - GET
 - get web resource from server
 - HEAD
 - return headers only of GET response
 - POST
 - send data to the server (forms, etc.)
- Case Sensitive

HTTP Request Message



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

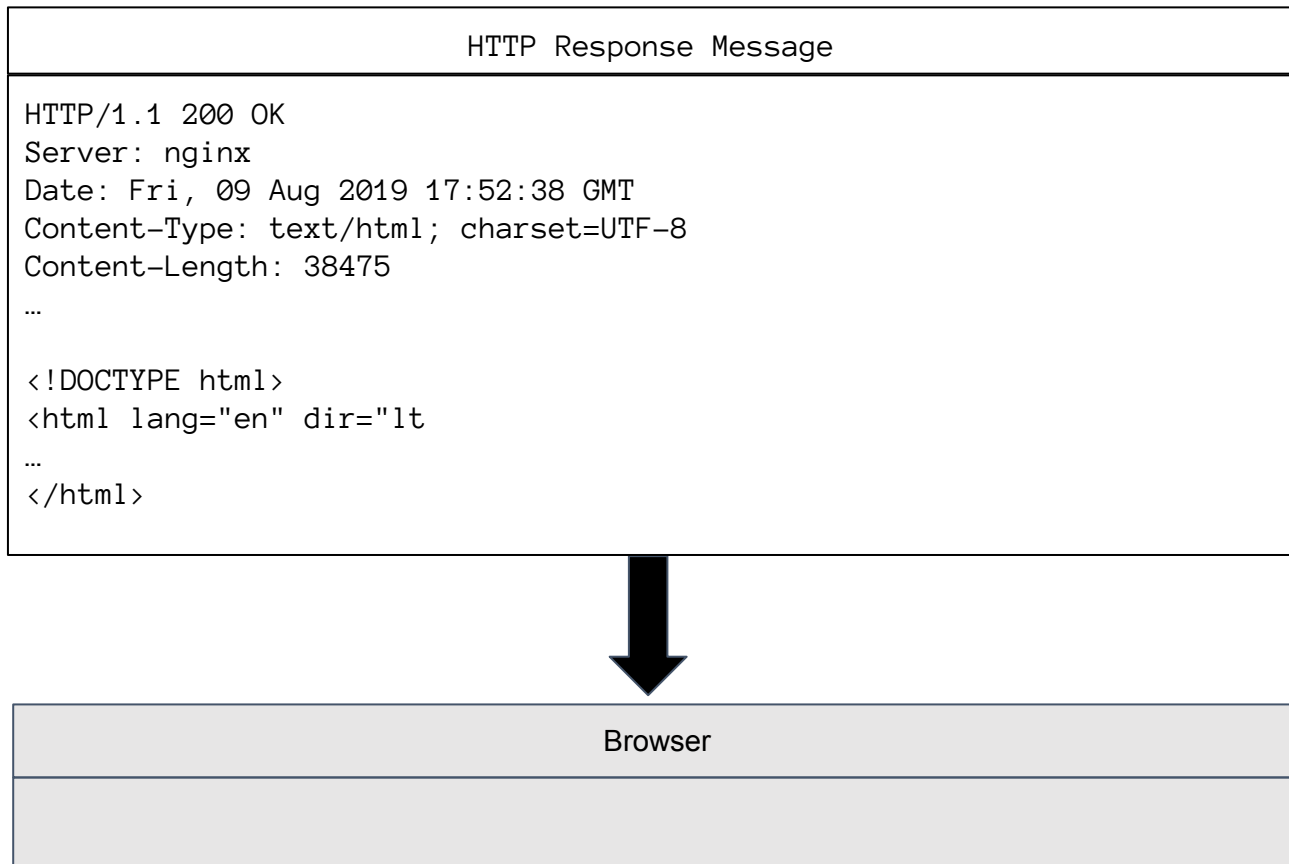
Blank line

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/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

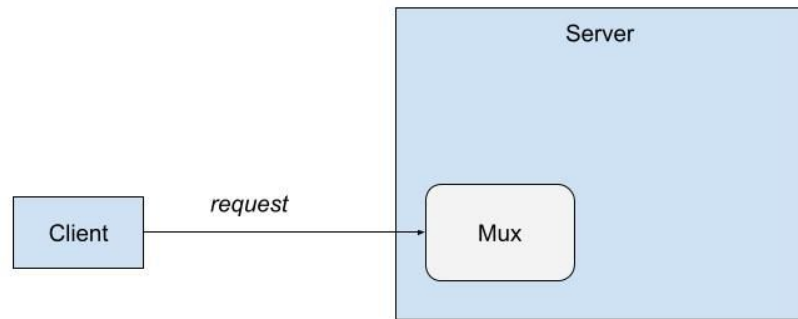
Exercises

- Browser inspection
- CURL

Building Simple HTTP Servers in Go

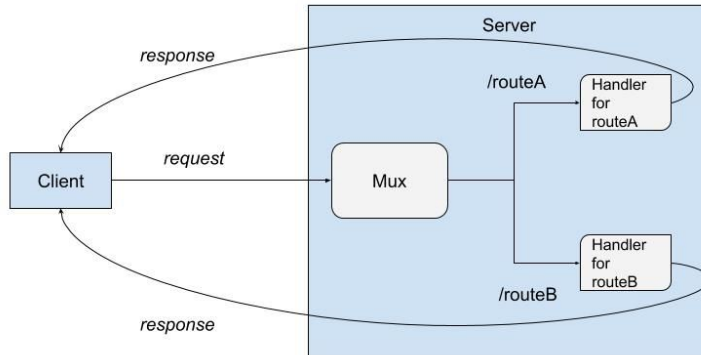
1. Write a simple web server which only listens
2. Extend the web server to serve content
3. What's in an `http.Request()`?
4. How do we build a custom Mux?

1. Write a simple web server which only listens



```
func ListenAndServe(addr string, handler Handler) error
```

2. Extend the web server to serve content



```
func HandleFunc(pattern string, handler func(ResponseWriter, *Request))
```