HTTP:
Hypertext Transfer Protocol

Lecture 12

HTTP

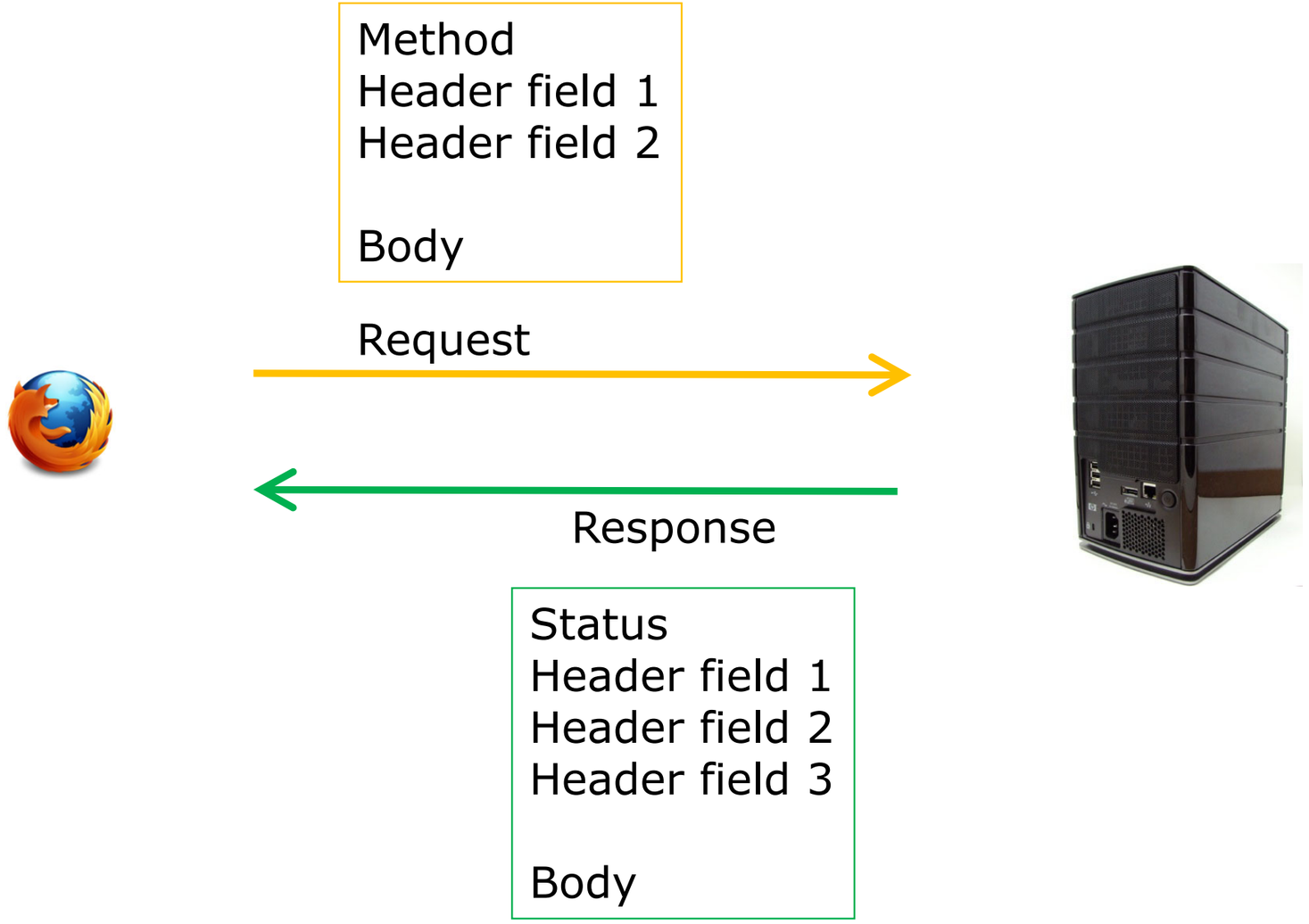
- Hypertext Transfer Protocol
- History
  - Early 90's: developed at CERN, Tim Berners-Lee
  - 1996: version 1.0
  - 1999: version 1.1 (still ubiquitous today!)
  - 2015: version 2
    - Performance improvements: binary, server push…
    - Backwards compatible
  - 2022: version 3
    - Performance improvements, same semantics

  w3techs.com/technologies/overview/site_element
- Simple request/response (client/server)
  - Client sends request to (web) server
  - (Web) server responds
  - Protocol itself is stateless

Anatomy of the Protocol

- Client sends a request:
  - Method: protocol identification
  - Header fields: meta information about request
  - (blank line)
  - Body (optional): payload
- Server sends a response
  - Status
  - Header fields: meta information about response
  - (blank line)
  - Body: payload
- ASCII-based protocol
  - First 3 lines are ASCII-only
  - Newline (CRLF) as separator: method/status, headers, blank

Protocol: Request, Response



Request: Method

- Structure of first line: ***verb path version***
  - Verb: **GET, HEAD, POST, PUT, DELETE,…**
  - Path: part of URL (path and query)

    scheme://FQDN:port/path?query#fragment
  - Version: **HTTP/1.1, HTTP/2, HTTP/3**
- Example:
  - For URL

    http://www.osu.edu/academics#content
  - First line of HTTP request is

    GET /academics HTTP/1.1

Request Methods

- GET, HEAD
  - Request: should be *safe* (no side effects)
  - Request has header only (no body)
- PUT
  - Update (or create): should be *idempotent*
- DELETE
  - Delete: should be *idempotent*
- POST
  - Create (or update): changes server state
  - Beware re-sending!
- HTTP does not enforce these semantics

Request: Header Fields

- Each field is on its own line:

  name*: *value

- Examples

```
Host: www.osu.edu
Accept: text/*,image/apng
Accept-Language: en-US,en;q=0.9
If-Modified-Since: Sat, 11 May 2024 19:43:31 GMT
Content-Length: 349
User-Agent: Mozilla/5.0 (X11; Linux x86_64)
  Chrome128.0.0.0 Safari/537.36
```

- Names are not case-sensitive ("Accept" or "accept")
- Blank line indicates end of headers

Some Common Header Fields

- Host
  - The only required field
  - Q: Why is host field even needed?
- Accept, Accept-Language, Accept-Encoding
  - List of browser preferences for response
  - MIME types, language locales, transfer encodings
  - Priority based on order and q-value weight (0-1)
- If-Modified-Since
  - Send payload only if changed since (GMT) date
- Content-Length
  - Required if request has a body
  - Number of bytes in body
- User-Agent
  - Application making the request (browser, but not necessarily)
- Referer (misspelled in spec)
  - Previous web page, ie "source" of this request

Example HTTP Request


```
GET / HTTP/1.1
Host: www.osu.edu
User-Agent: Mozilla/5.0 (X11; Ubuntu;…etc
```

Request

"Nobody knows you're a dog"

```
$ telnet
```

```
GET / HTTP/1.1
Host: www.osu.edu
User-Agent: Mozilla/5.0 (X11; Ubuntu;…etc
```

Request

```
$ curl -A "Mozilla/5.0" http://www.osu.edu
```


```
require 'mechanize'
agent = Mechanize.new
page = agent.get 'http://www.osu.edu'
```

Demo: HTTP Request with telnet

- Example URL: http://www.osu.edu/academics
- At console

```
$ telnet www.osu.edu 80 # opens connection
GET /academics HTTP/1.1
Host: www.osu.edu
<blank line>
```

HTTP Response Anatomy

- Recall, four parts
  1. Status (one line)
  2. Header fields (separated by newlines)
  3. Blank line
  4. Body (*i.e.*, payload)
- Parts 1-2 collectively are the header
- Status line syntax:

  http-version status-code text

  - Examples

```
HTTP/1.1 200 OK
HTTP/1.1 301 Moved Permanently
HTTP/1.1 404 Not Found
```

Taxonomy of Status Codes

Code	Meaning
1xx	Informational
2xx	Success
3xx	Redirection
4xx	Client Error
5xx	Server Error

Some Common Status Codes

- 200 Success/OK
  - All is good!
  - Response body is the requested document
- 301 (302) Permanent (Temporary) Redirect
  - Requested resource is found somewhere else
  - 301 means agent can go directly to the new location in the future
- 304 Not Modified
  - Document hasn't changed since date/time in If-Modified-Since field of request
  - No response body
- 404 Not Found
  - Server could not satisfy the request
  - It is the client's fault (design-by-contract?)
- 500 Internal Server Error
  - Server could not satisfy the request
  - It is the server's fault (design-by-contract?)

Response: Header Fields

- Each field on its own line, syntax:

  name*: *value

- Examples

```
Date: Tue, 16 Sep 2025 17:31:18 GMT
Server: Apache/2.4.6 (Red Hat)
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip
Content-Length: 333
```

- Blank line indicates end of headers

Demo: Terminal (telnet)

- Redirects entail another telnet request

```
telnet www.osu.edu 80
GET /academics HTTP/1.1
Host: www.osu.edu

HTTP/1.1 301 Moved Permanently
Location: https://www.osu.edu/academics
```

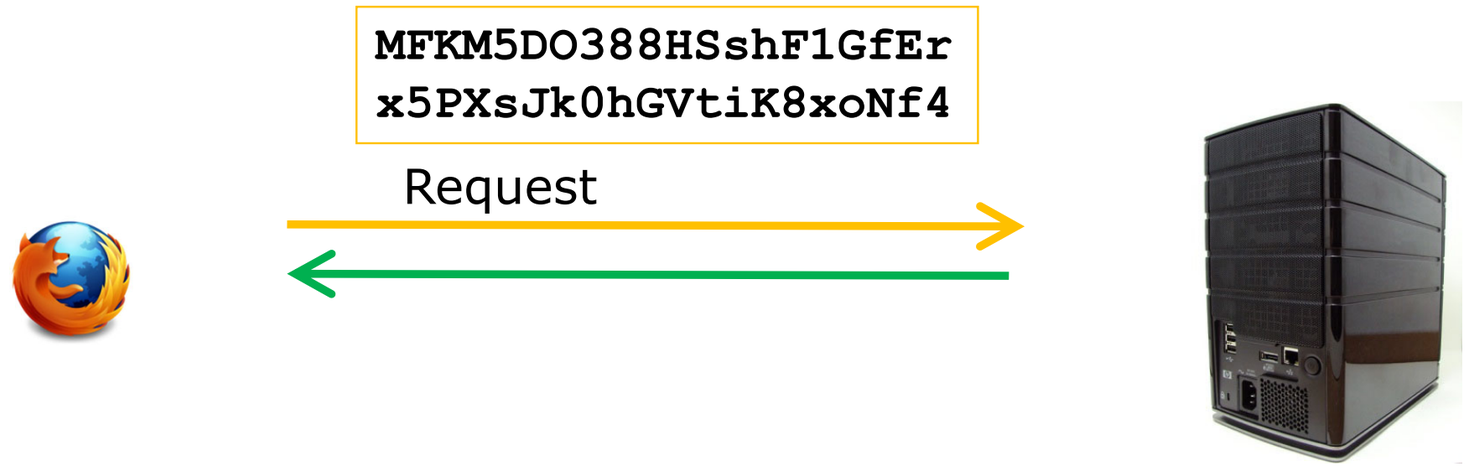
- Plain-text http is increasingly rare

```
telnet www.osu.edu 443
GET /academics HTTP/1.1
Host: www.osu.edu

HTTP/1.1 400 Bad Request
```

HTTP Traffic Transparency

- Everything is visible to an eavesdropper
  - HTTP headers are plain text
  - HTTP payload may be binary
- To protect communication, use encryption
  - SSL, TLS: protocols to create secure channel
  - Initial handshake between client and server
  - Subsequent communication is encrypted
- HTTP over secure channel = HTTPS
  - Default port: 443



Demo: HTTPS with openssl

- Use openssl instead of telnet
  - Negotiates initial handshake with server
  - Handles encryption/decryption of traffic
- Example URL **https://www.osu.edu/**
- At console

```
$ openssl s_client -connect www.osu.edu:443
```

  - Note connection to port 443 (standard for https)
- Syntax of subsequent request is the same
- Send the following HTTP request:

```
GET /academics HTTP/1.1
Host: www.osu.edu
<blank line>
```

Demo: Terminal (curl)

- Better command-line tool: cURL

```
$ curl -v www.osu.edu/academics # -v for verbose
```

  - Handles redirection (-L), chunking, https, headers, …

```
$ curl -Li www.osu.edu/academics # follows redirect
```

  - Explicitly set request headers (-H)

```
$ curl https://www.osu.edu \
  -A "Mozilla/5.0" \ # default user agent: curl/VER
  -H "accept: text/html,text/plain" \
  -H "accept-language: en-US,en;q=0.9"
```

- Very useful, Swiss Army knife, for network requests

```
$ curl -V # see list of Protocols
```

Demo: Chrome Developer Tools

- Powerful inspection tool for web development
  - Kabob > More Tools > Developer Tools (Ctrl+Shift+I)
  - See the Network tab
- One GET can cause many HTTP requests by browser

  `http://www.osu.edu/academics#content`
- For each request, inspect:
  - Request method, headers
  - Response status code, and headers
  - Response body (and preview)
- To reproduce a request:
  - Right click, Copy > Copy as cURL

Demo: Using Ruby

- Mechanize: A Ruby gem that acts like a browser

```
require 'mechanize'
```

- Create an agent to send requests

```
agent = Mechanize.new do |a|
  a.user_agent_alias = 'Mac Safari'
end
```

- Use agent to issue a request

```
page = agent.get 'http://news.osu.edu'
```

- Follow links, submit forms, etc

```
h = page.link_with(text: /Top/).click
f = page.forms[0]
f.field_with(name: 'q').value = 'CSE'
s = f.submit
```

HTTP is Stateless

- Every request looks the same
- But maintaining state between requests is useful:
  - First user logs in, then able to GET account info
  - Shopping cart "remembers" contents
- Solution: Keep a shared secret
  - Server's first response contains a unique session identifier (a long random value)
  - Subsequent requests from this client include this secret value
  - Server recognizes the secret value, request must have come from original client

HTTP Session (4)

Request

Response
Secret: `38afes7a8`

Request
id: `38afes7a8`

Response

Request
id: `38afes7a8`

Response

Store secret

`38afes7a8`

Check id

Check id

HTTP Cookies

- Popular mechanism for session manag’nt
- Set in *response* header field

  `Set-Cookie: session=38afes7a8`

  - Any name/value is ok
  - Options: expiry, require https
- Client then includes cookie(s) in any subsequent request to that domain
- Sent in *request* header field:

  `Cookie: session=38afes7a8`
- Cookies also used for
  - Tracking/analytics: What path did they take?
  - Personalization

Summary

- HTTP: request/response
- Anatomy of request
  - Methods: GET, PUT, DELETE, POST
  - Headers
  - Body: arguments of POST
- Anatomy of response
  - Status Codes: 200, 301, 404, etc
  - Headers
  - Body: payload
- Tools
  - Curl, Developer Tools, Mechanize