Computer Networks 414/614

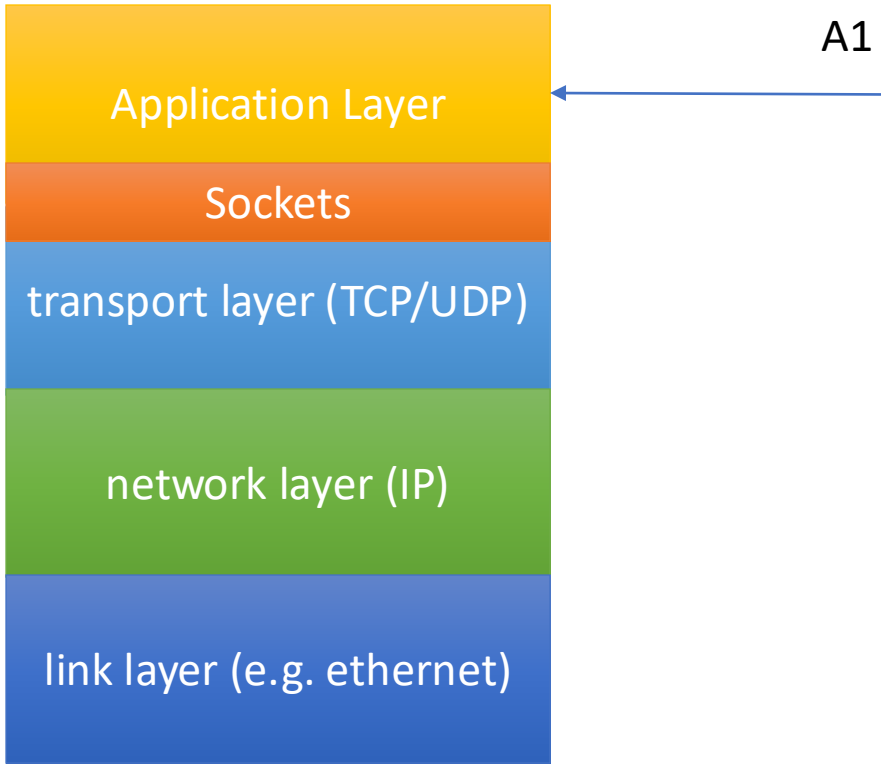
HTTP & HTTP Proxies

Assignment 1 Tutorial

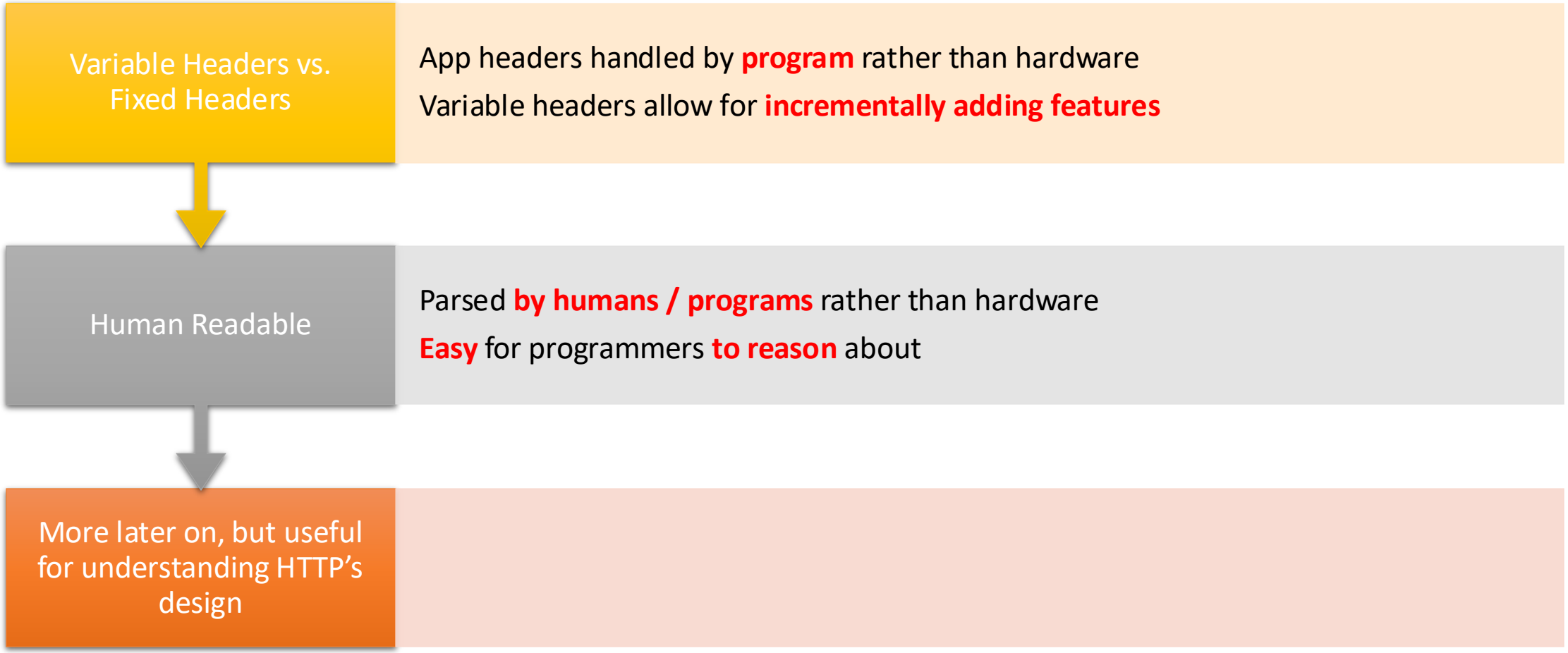
Spring 2026

Credit: Sougol Gheissi

Assignments Structure



Application Layer Protocols



Hypertexts Transport Protocol
HTTP Basics (Overview)

HTTP layered over bidirectional byte stream

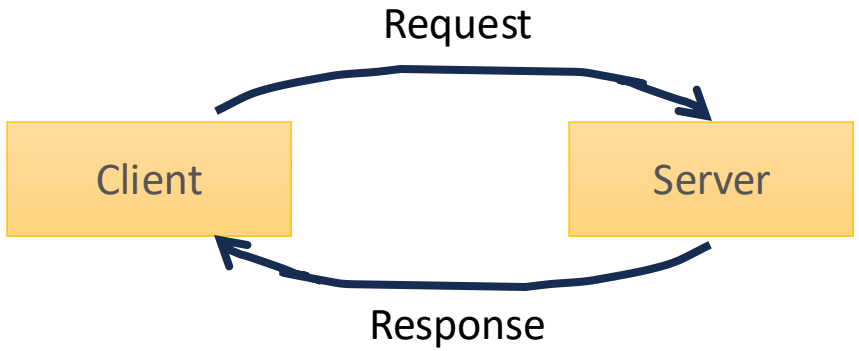
- Almost always TCP

Interaction

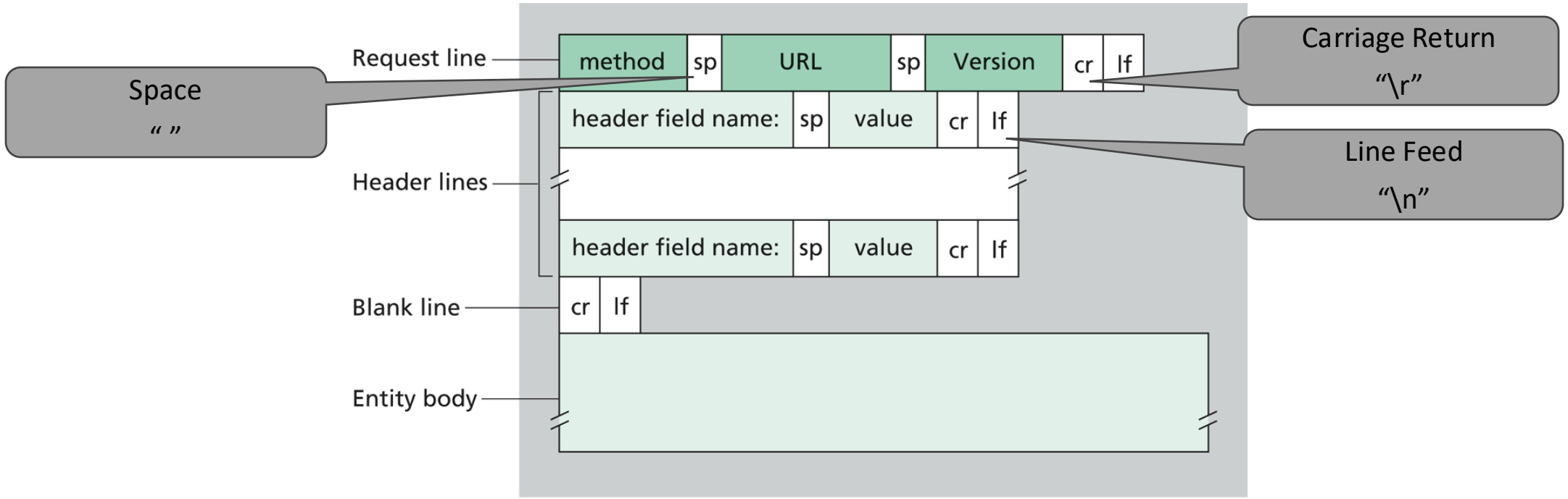
- Client looks up host (DNS)
- Client sends request to server
- Server responds with data or error
- Requests/responses are encoded in text

Stateless

- Server maintains no info about past client requests
- Why?



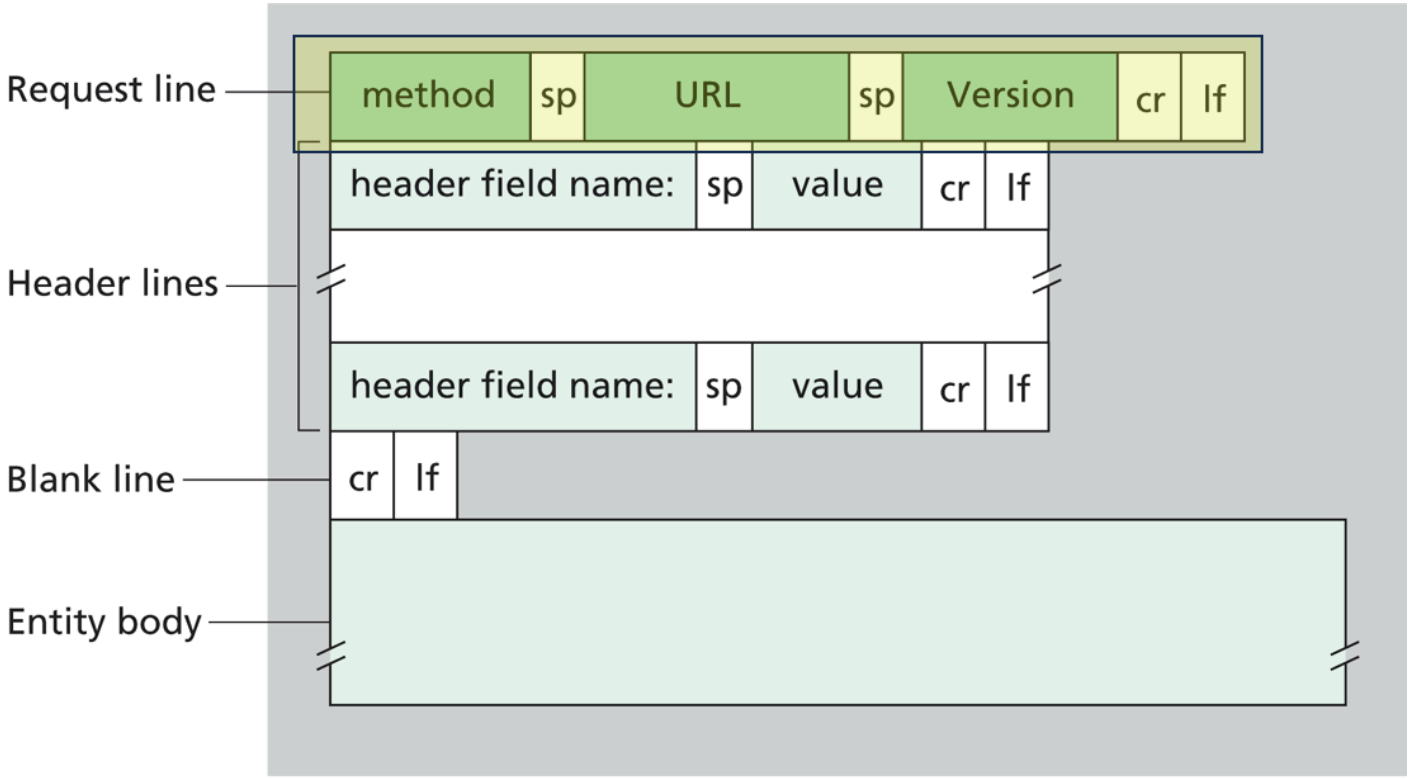
HTTP Request Format



HTTP Request

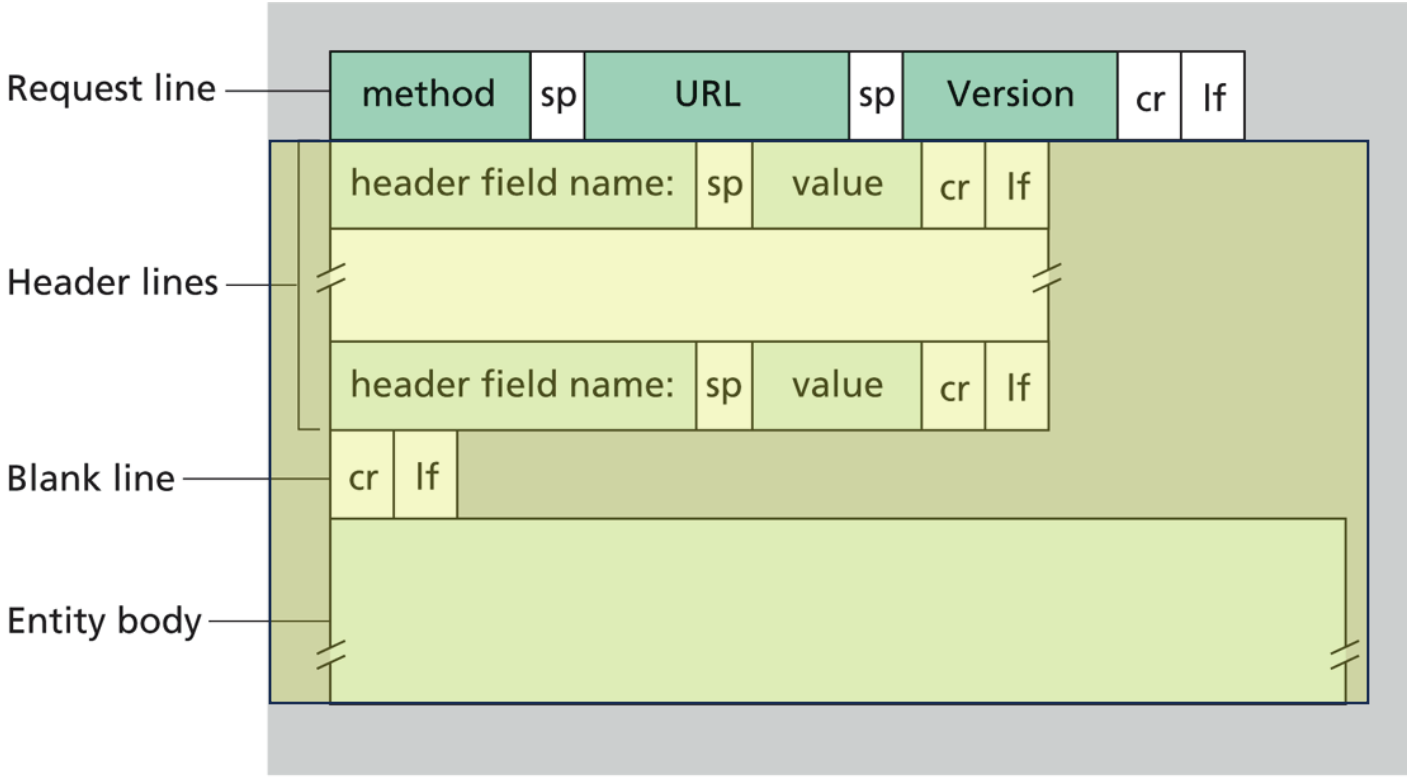
- Request line
  - Method
    - GET – return URI
    - HEAD – return headers only of GET response
    - POST – send data to the server (forms, etc.)
  - URL (relative)

    E.g., /index.html
  - HTTP version



HTTP Request (cont.)

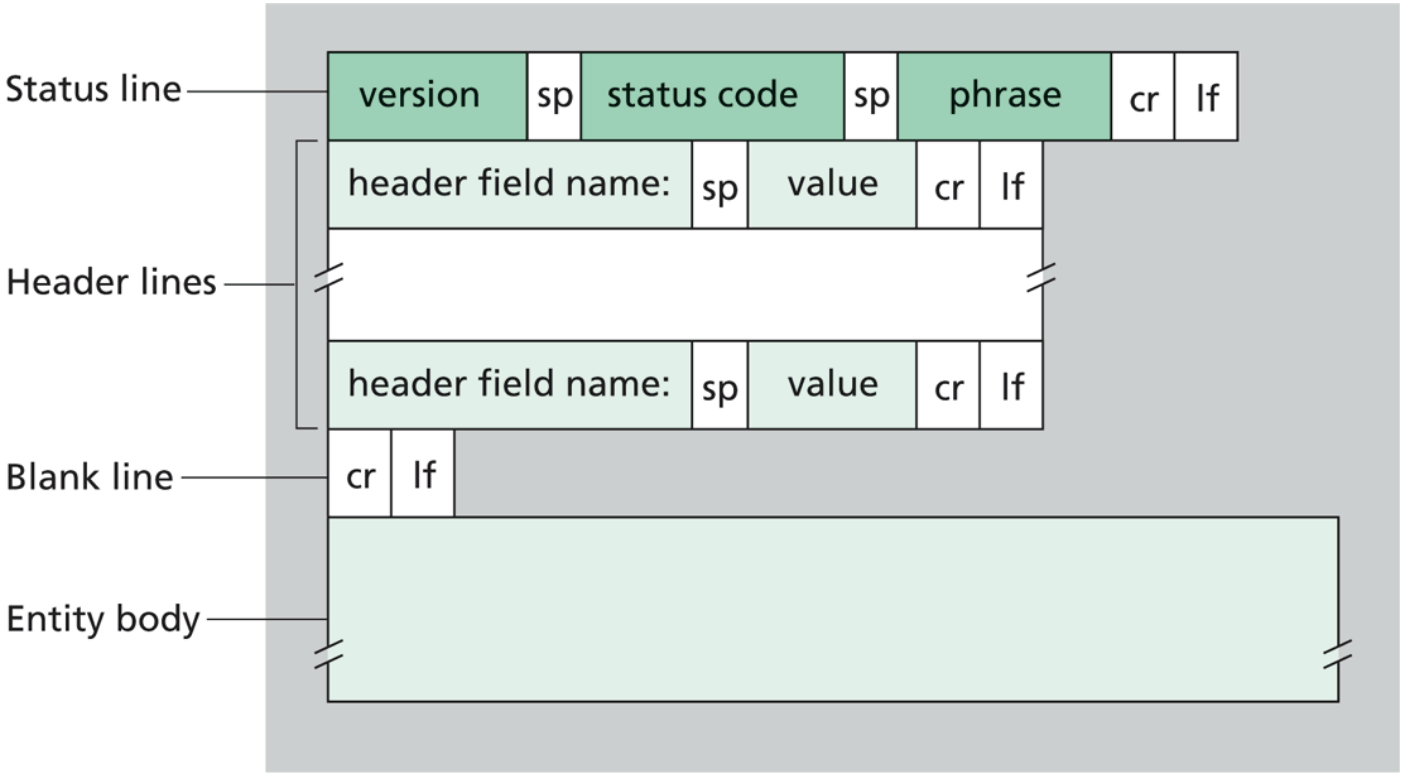
- Request headers
  - Variable length, human-readable
  - Uses:
    - Authorization – authentication info
    - Acceptable document types/encodings
    - From – user email
    - If-Modified-Since
    - Referrer – what caused this page to be requested
    - User-Agent – client software
- Blank-line
- Body



HTTP Request Example

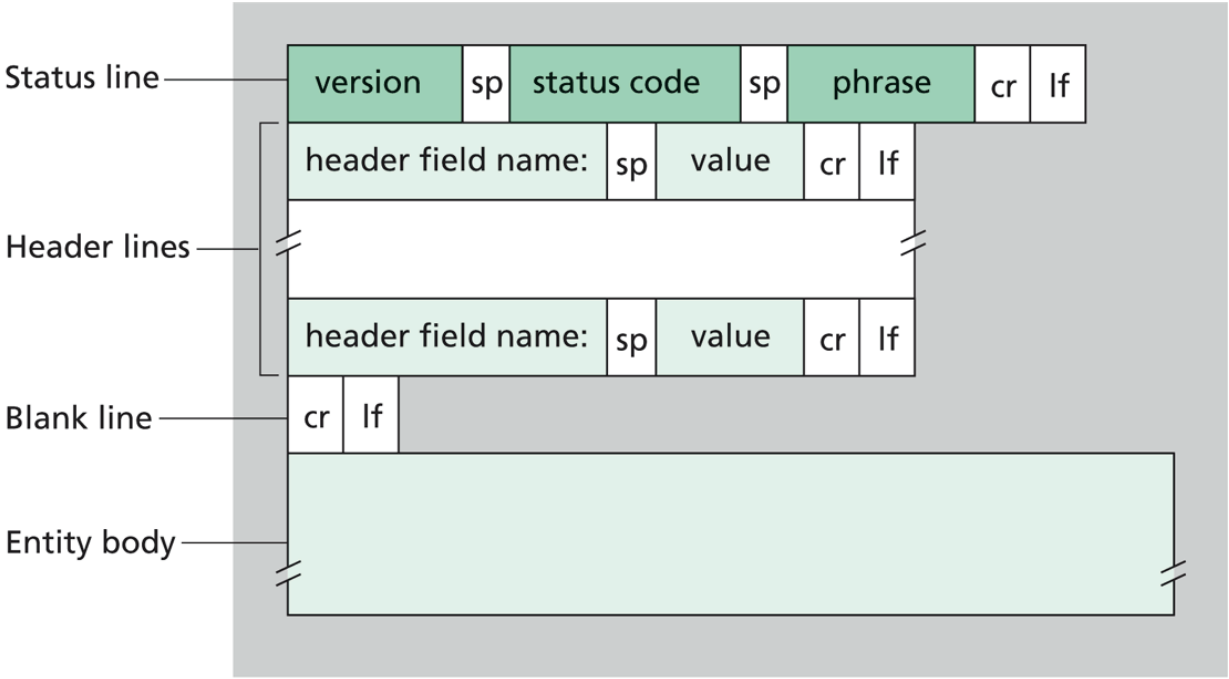
```
GET /index.html HTTP/1.1
Host: www.example.com
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible;
MSIE 5.5; Windows NT 5.0)
Connection: Keep-Alive
```

HTTP Response Header



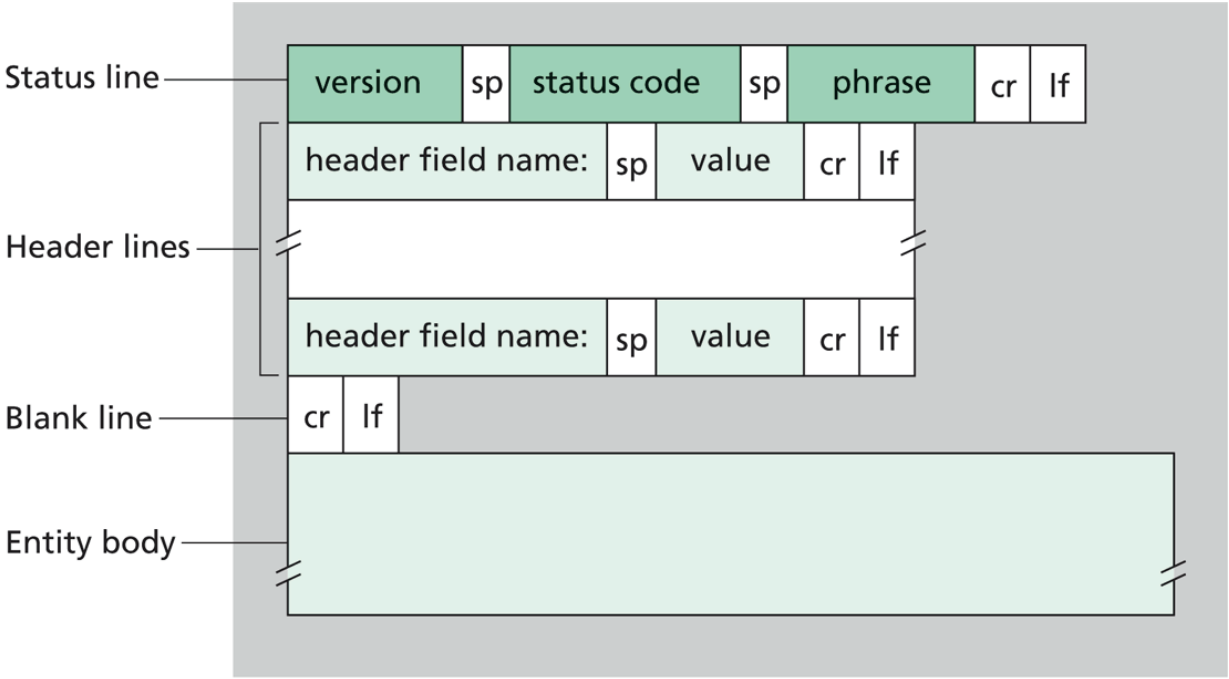
HTTP Response

- Status-line
  - HTTP version
  - 3 digit response code
    - 1XX – informational
    - 2XX – success
      - 200 OK
    - 3XX – redirection
      - 301 Moved Permanently
      - 303 Moved Temporarily
      - 304 Not Modified
    - 4XX – client error
      - 404 Not Found
    - 5XX – server error
      - 505 HTTP Version Not Supported
  - Reason phrase



HTTP Response (cont.)

- Headers
  - Variable length, human-readable
  - Uses:
    - Location – for redirection
    - Server – server software
    - WWW-Authenticate – request for authentication
    - Allow – list of methods supported (get, head, etc)
    - Content-Encoding – E.g x-gzip
    - Content-Length
    - Content-Type
    - Expires (caching)
    - Last-Modified (caching)
- Blank-line
- Body



HTTP Response Example

```
HTTP/1.1 200 OK
Date: Tue, 27 Mar 2001 03:49:38 GMT
Server: Apache/1.3.14 (Unix) (Red-Hat/Linux) mod_ssl/2.7.1
OpenSSL/0.9.5a DAV/1.0.2 PHP/4.0.1pl2 mod_perl/1.24
Last-Modified: Mon, 29 Jan 2001 17:54:18 GMT
Content-Length: 4333
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html
```

How to Mark End of Message?

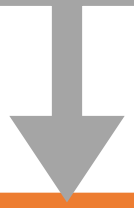
- Content-Length
  - Must know size of transfer in advance
- Close connection
  - Only server can do this
- Implied length
  - E.g., 304 never have body content
- Transfer-Encoding: chunked (HTTP/1.1)
  - After headers, each chunk is content length in hex, CRLF, then body. Final chunk is length 0.

Proxies

Proxies

End host that acts a broker between client and server

Speaks to server on client's behalf



Why?

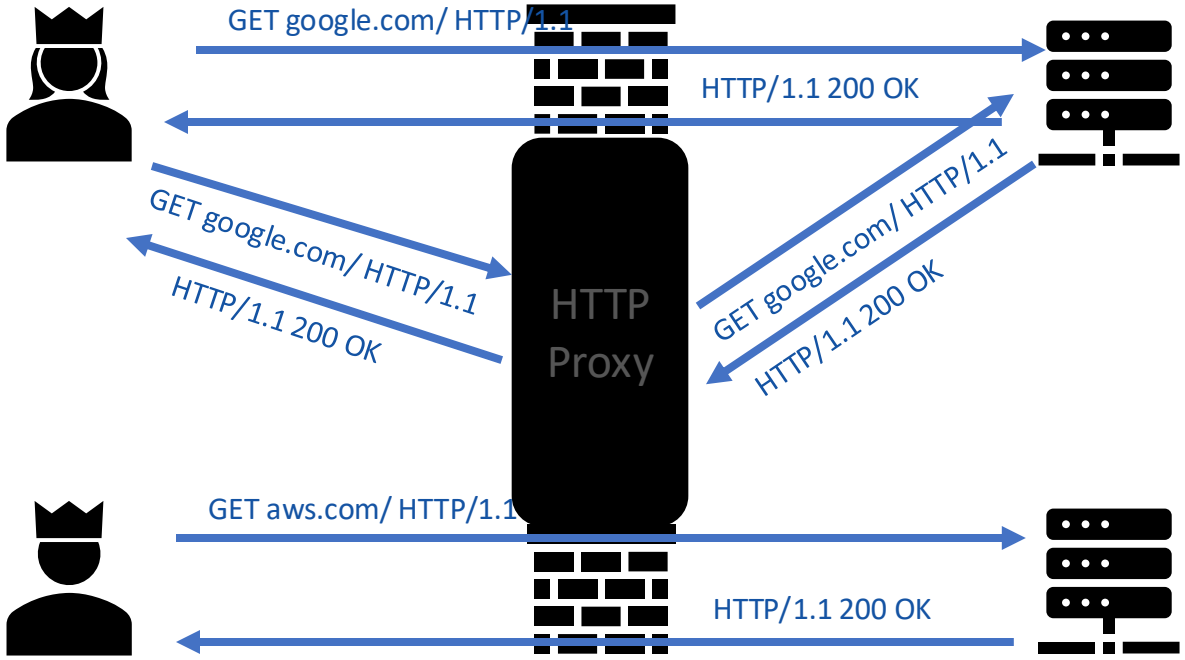
Privacy

Content filtering

Can use caching (not in this assignment)

Proxies (Cont.)

- Accept requests from multiple clients
- Takes request and reissues it to server
- Takes response and forwards to client



Assignment 1

- Non-caching, HTTP 1.0 proxy
  - Support only GET requests
- Multi-process
  - Use fork()
- Simple binary that takes a port number
  - ./proxy 12345 (proxy listens on port 12345)
- Work in Firefox & Chrome
  - Use settings to point browser to your proxy

Assignment 1 (Cont.)

- What you need from a client request: host, port, and URI path
  - `GET http://www.jhu.edu:80/ HTTP/1.0`
- What you send to a remote server:

```
GET / HTTP/1.0
Host: www.jhu.edu:80
Connection: close
```

- Check request line and header format
- Forward the response to the client

Assignment 1 (Cont.)

- Non-GET request?
  - return “Not Implemented” (code 501)
- Unparseable request?
  - return “Bad Request” (code 400)
- Use provided parsing library
- Postel’s law
  - Be liberal in what you accept, and conservative in what you send
  - convert HTTP 1.1 request to HTTP 1.0 (extra points 😀)
  - convert \r to \r\n
  - etc

Advice

- Networking is hard
  - Hard to know what's going on in network layers
  - **Start** out **simple**, **test often**
- Build in steps
  - **Incrementally** add pieces
  - Make sure they work
  - Will help reduce the effect of "incomplete" information

Getting Started

Modify Assn 0 to have server respond

- Simple echo of what client sent

Modify Assn 0 to handle concurrent clients

- Use fork()

Create “proxy” server

- Simply “repeats” client msg to a server, and “repeats” server msg back

Client sends HTTP requests, proxy parses