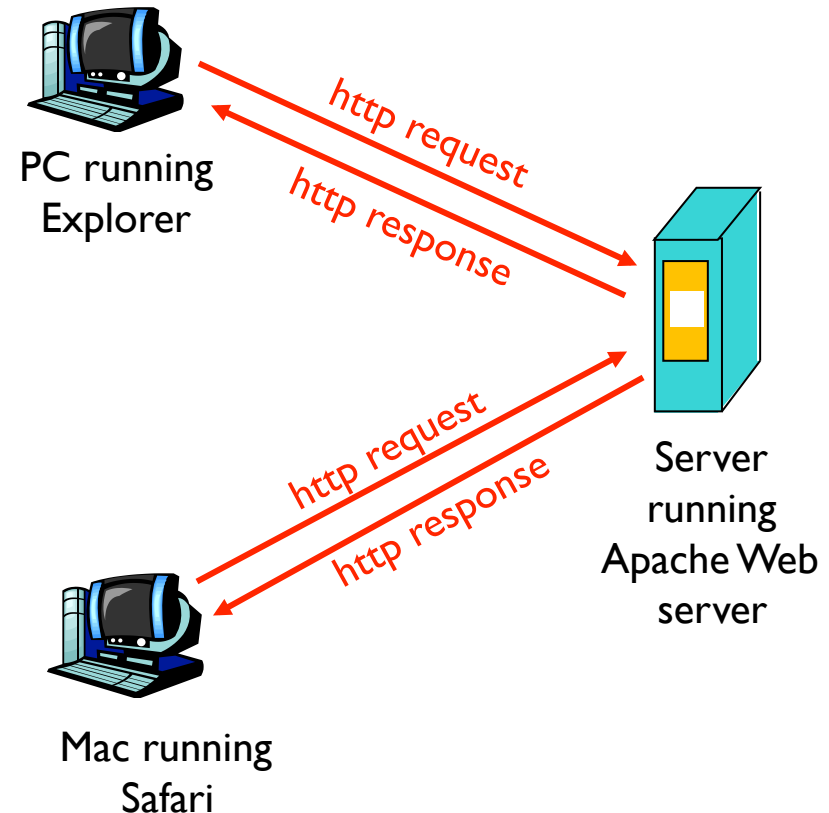


# The Web: the HTTP protocol

- Client-server model:

- **client**: browser that requests, receives, “renders” Web objects
- **server**: Web server sends objects in response to requests



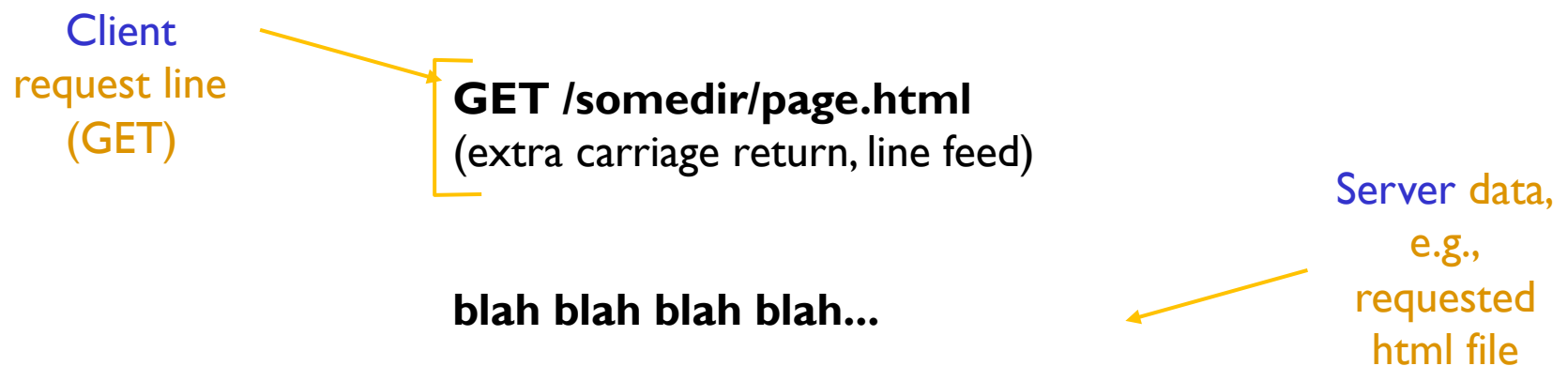
# A Little Historical Context

- The `WEB' is composed of a set of linked HTML documents.
- The WWW is a network
- The HyperText Transfer is an application level protocol for transferring 'web' documents over the internet.



# Growth of a Protocol (HTTP)

- Started with HTTP/0.9 (1990)
  - raw data transfer over the Internet (simple)
  - client issues **GET Request**
  - server replies with bytes comprising the HTML



# Growth of a Protocol (HTTP/1.0)

- Next HTTP/1.0 (1996)
  - sophisticated requests {GET, HEAD and POST}
  - server response {headers + data}
  - better client-server coordination/interaction.
  - coordinates with other protocols {SMTP, NNTP, ...}
  - MIME (Multipurpose Internet Mail Extensions)

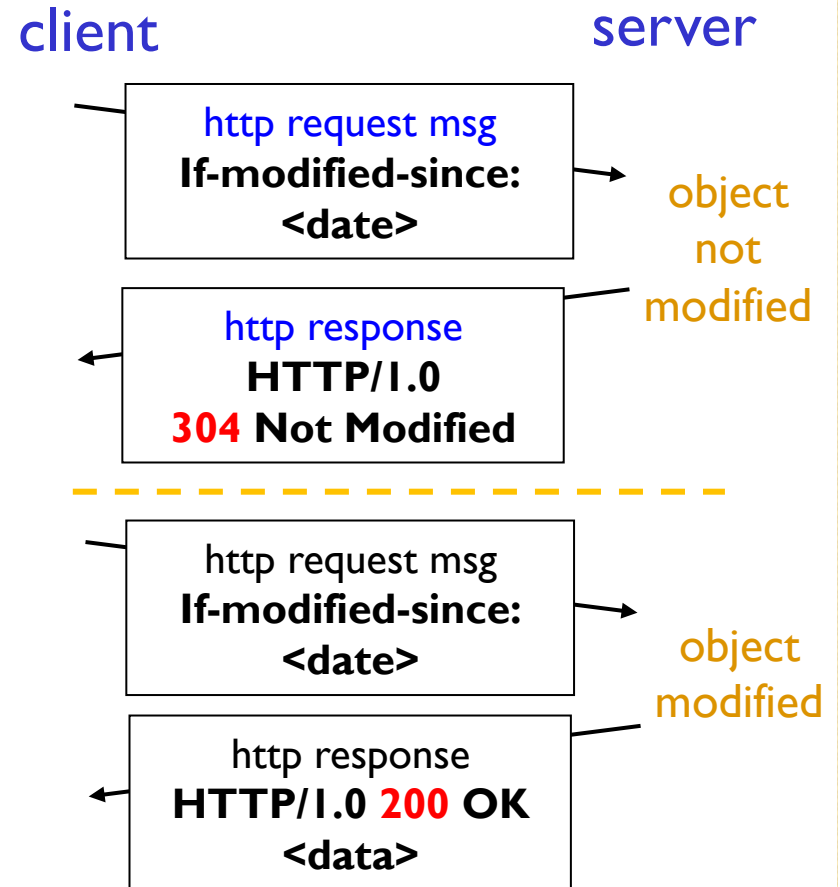


# Main HTTP/1.0 Commands

- GET <Request-URI>
  - request the page
- HEAD <Request-URI>
  - request the header for the page
- POST <Request-URI>
  - send data to the server

# Conditional GET

- **Goal:** don't send object if client has up-to-date cached version
- client: specify date of cached copy in http request
  - **If-modified-since:** <date>



# HTTP 1.1

- A successor to HTTP 1.0
  - The **HTTP/1.1** standard as defined in **RFC 2068** was officially released in January **1997**
  - Improvements and updates to the HTTP/1.1 standard were released under **RFC 2616** in June **1999**.
  - **Single connection** for multiple requests



# Quick exercise

- What are the possible benefits of using a **persistent connection**?
  - A. Reduce the delay for sending request and response messages.
  - B. Increase the communication throughput between browser and Web server.
  - C. Reduce resource consumption at the browser.
  - D. Simplify the implementation of HTTP.





## Quick exercise

- Why a persistent HTTP connection is more complicated to implement?
  - A. HTTP server cannot destroy the TCP connection once a response was sent back.
  - B. Security vulnerability needs to be addressed.
  - C. The pipelining feature requires more sophisticated control
  - D. None of the above.



# Newly introduced commands

- **PUT**: requests to store an enclosed entity.
- **DELETE**: deletes a resource.
- **OPTION**: returns supported methods.
- **TRACE**: echoes back the request.
- **CONNECT** and **PATCH**

# The HTTP protocol

- client initiates **TCP connection** (creates socket) to server, port **80**
- server accepts TCP connection from client
- http messages exchanged between browser (http client) and Web server (http server)
- TCP connection closed

- HTTP is “**stateless**”

**aside**  
Protocols that maintain “state” are **complex!**

past history (state) must be maintained

if server/client crashes, their views of “state” may be inconsistent, must be reconciled

# Client: request

- http request message:
  - ASCII (human-readable format)

The diagram illustrates the structure of an HTTP request message. It shows a sequence of lines: a request line, followed by header lines, and an ellipsis indicating more header lines. Annotations with arrows point to these parts: 'request line (GET, POST, HEAD commands)' points to the first line; 'header lines' points to the subsequent lines; and 'Carriage return, line feed indicates end of message' points to the end of the message block. Below the ellipsis, a note specifies '(extra carriage return, line feed)'.

request line  
(GET, POST,  
HEAD commands)

**GET /somedir/page.html HTTP/1.1**

**User-agent: Mozilla/4.0**

**Accept: text/html,image/gif,image/jpeg**

**If-Modified-Since: Mon, 22 Jun 2009...**

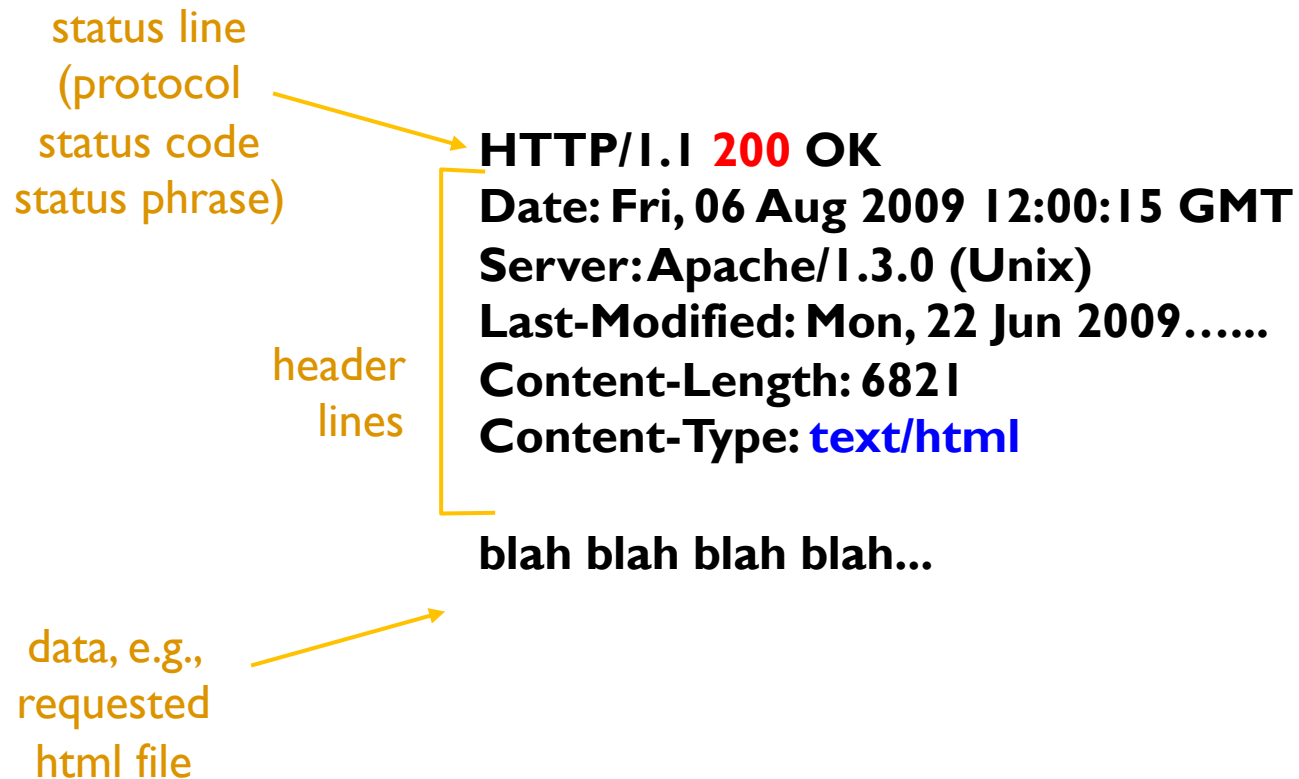
header lines

...

(extra carriage return, line feed)

Carriage return,  
line feed  
indicates end  
of message

# Server: response





# HTTP header fields

- **Header fields** are components of the **header section** of request and response messages in the HTTP.
- Types of header fields
  - General header
  - Client request header
  - Server response header
  - Entity header



# Quick exercise

- Which header field below is for general use?
  - A. Date header
  - B. Accept header
  - C. Location header
  - D. Content-Length header

# HTTP response status codes

In first line in server->client response message.

A few sample codes:

## **200 OK**

- request succeeded, requested object later in this message

## **301 Moved Permanently**

- requested object moved, new location specified later in this message (**Location:**)

## **400 Bad Request**

- request message not understood by server

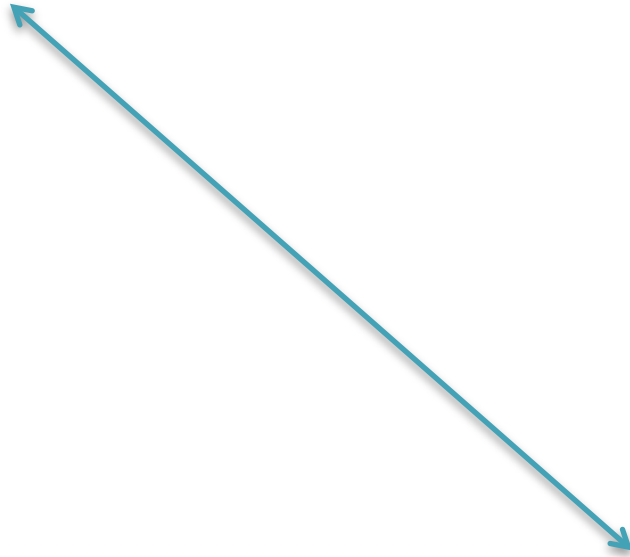
## **404 Not Found**

- requested document not found on this server

## **500 Internal Server Error**



# Meaning of the first digit



1xx	Success
2xx	Server error
3xx	Redirection
4xx	Informational
5xx	Client error

# Trying out http (client side)

## 1. Telnet to your favorite Web server

```
telnet www.victoria.ac.nz 80
```

```
GET / HTTP/1.1
```

## 2. Use the CURL command

```
curl --data "birthyear=1905" http://www.example.com/when.cgi
```

## 3. Look at response message sent by http server!