HTTP

HyperText Transfer Protocol

HTTP

- Created by Tim Berners-Lee at CERN
  - Defined 1989-1991
- Standardized and much expanded by the IETF
- Rides on top of TCP protocol
  - TCP provides: reliable, bi-directional, in-order byte stream

- Goal: transfer objects between systems
  - Do not confuse with other WWW concepts:
    - HTTP is not page layout language (that is HTML)
    - HTTP is not object naming scheme (that is URLs)
- Text-based protocol
  - Human readable

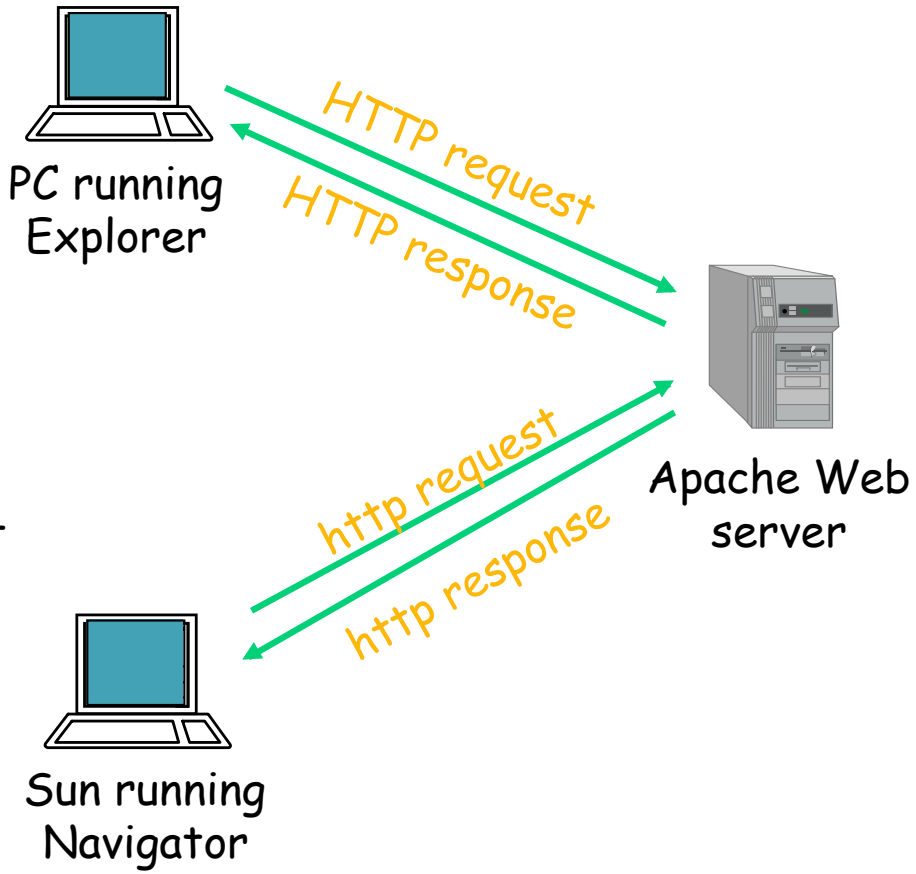
Client-Server Paradigm

Client:

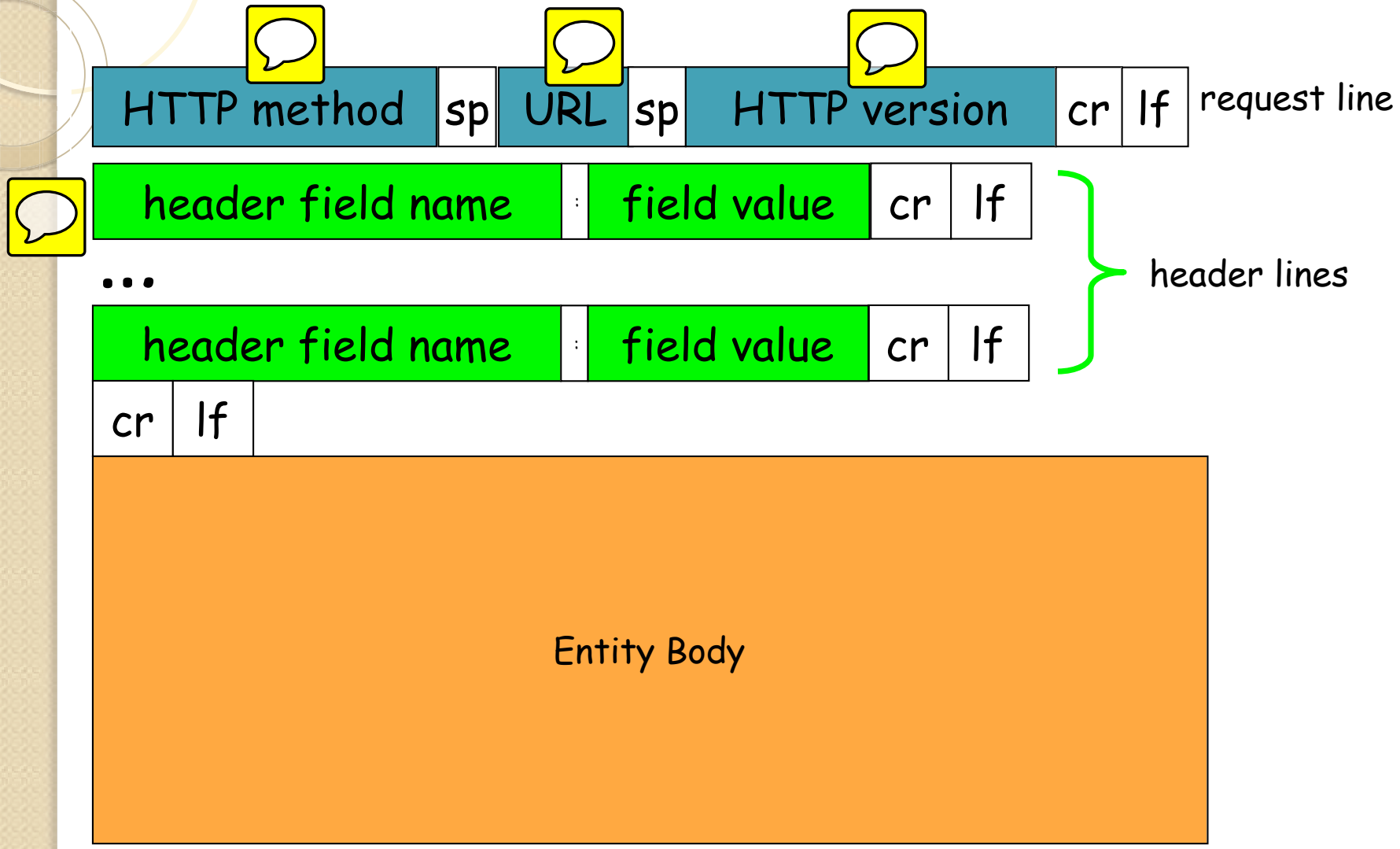
- initiates contact with server ("speaks first")
- typically requests service from server,
- for Web, client is implemented in browser; for e-mail, in mail reader

Server:

- provides requested service to client
- e.g., Web server sends requested Web page, mail server delivers e-mail

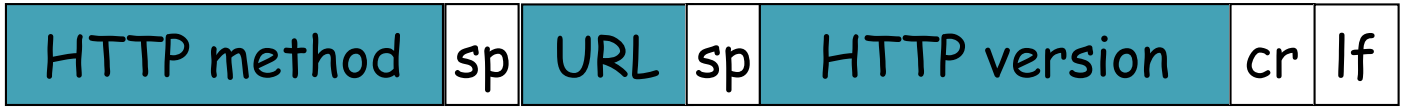


HTTP request message



HTTP request format

- Request line



  - HTTP method
    - GET – return content of specified document
    - HEAD – return headers only of GET response
    - POST – execute specified doc with enclosed data
  - URL (only domain portion)
    - /host-identifier/path
    - e.g. /www.toronto.edu/headlines/
  - HTTP version
    - e.g. HTTP/1.0

HTTP request format

- Header fields

- Examples:
  - Accept: text/html
  - Accept: image/jpg
  - Accept-language: en; en-gr; fr
  - If-modified-since: 17 May 2001
  - Content-Length: 2540

HTTP request example

```
GET /somedir/page.html HTTP/1.0
User-agent: Mozilla/4.0
Accept: text/html, image/gif,image/jpeg
Accept-language:fr
```

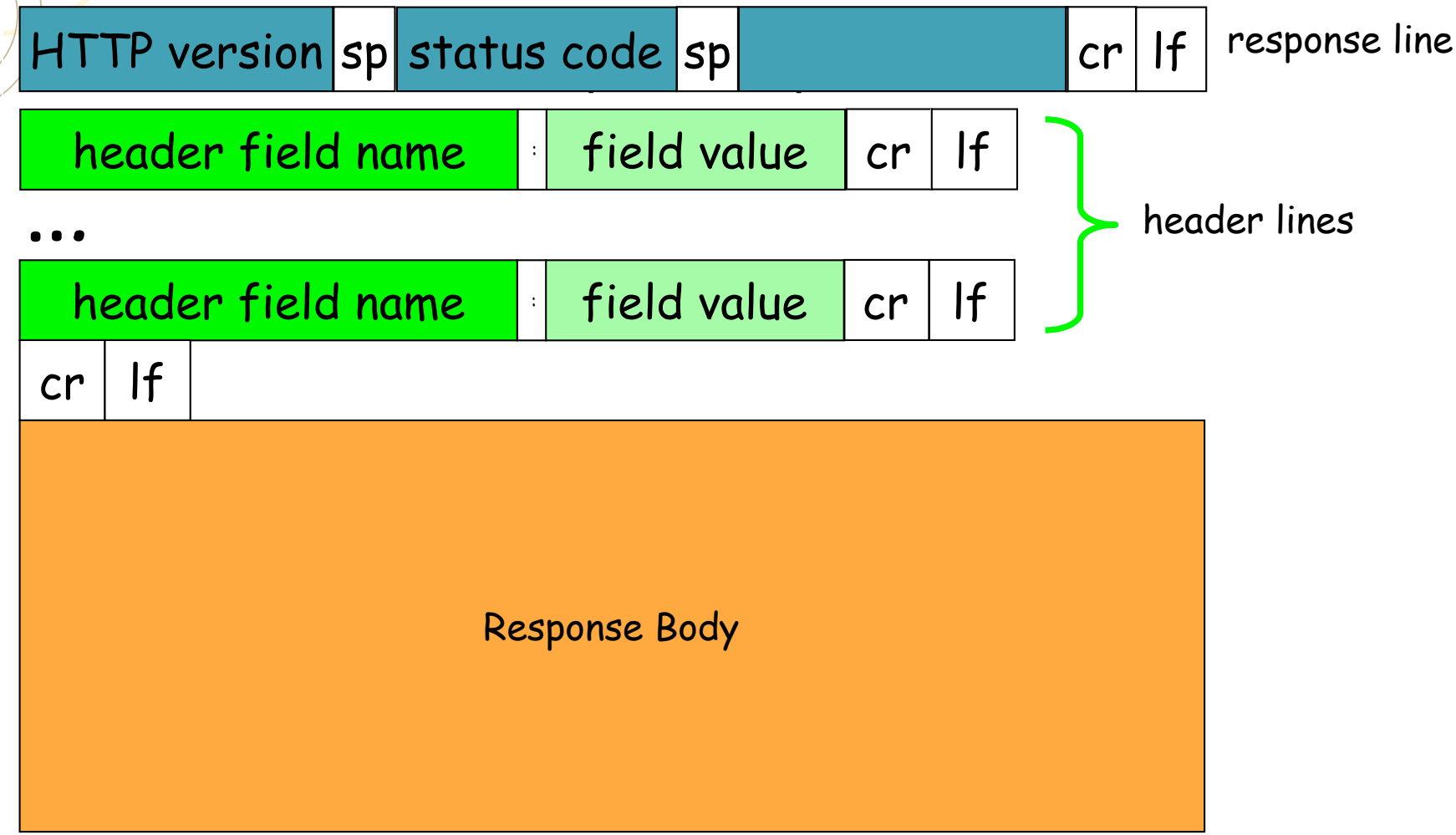
(extra carriage return, line feed)

request line (GET, POST, HEAD commands)

header lines

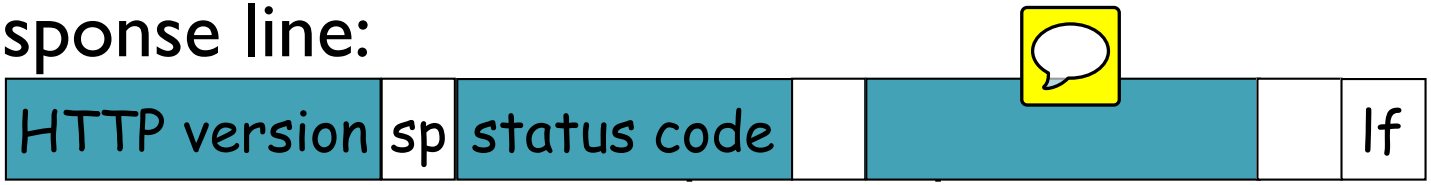
Carriage return, line feed indicates end of message

HTTP response message



HTTP response format

- Response line:



  - HTTP version
  - 3 digit response code
    - 1XX – informational
    - 2XX – success
    - 3XX – redirection
    - 4XX – client error
    - 5XX – server error
  - Brief text explanation of status code (e.g. OK)

HTTP 1.0: Problems

- Each request opens new connection
- Opening connection takes several packets (why?)
- Starting up is slow (why?)

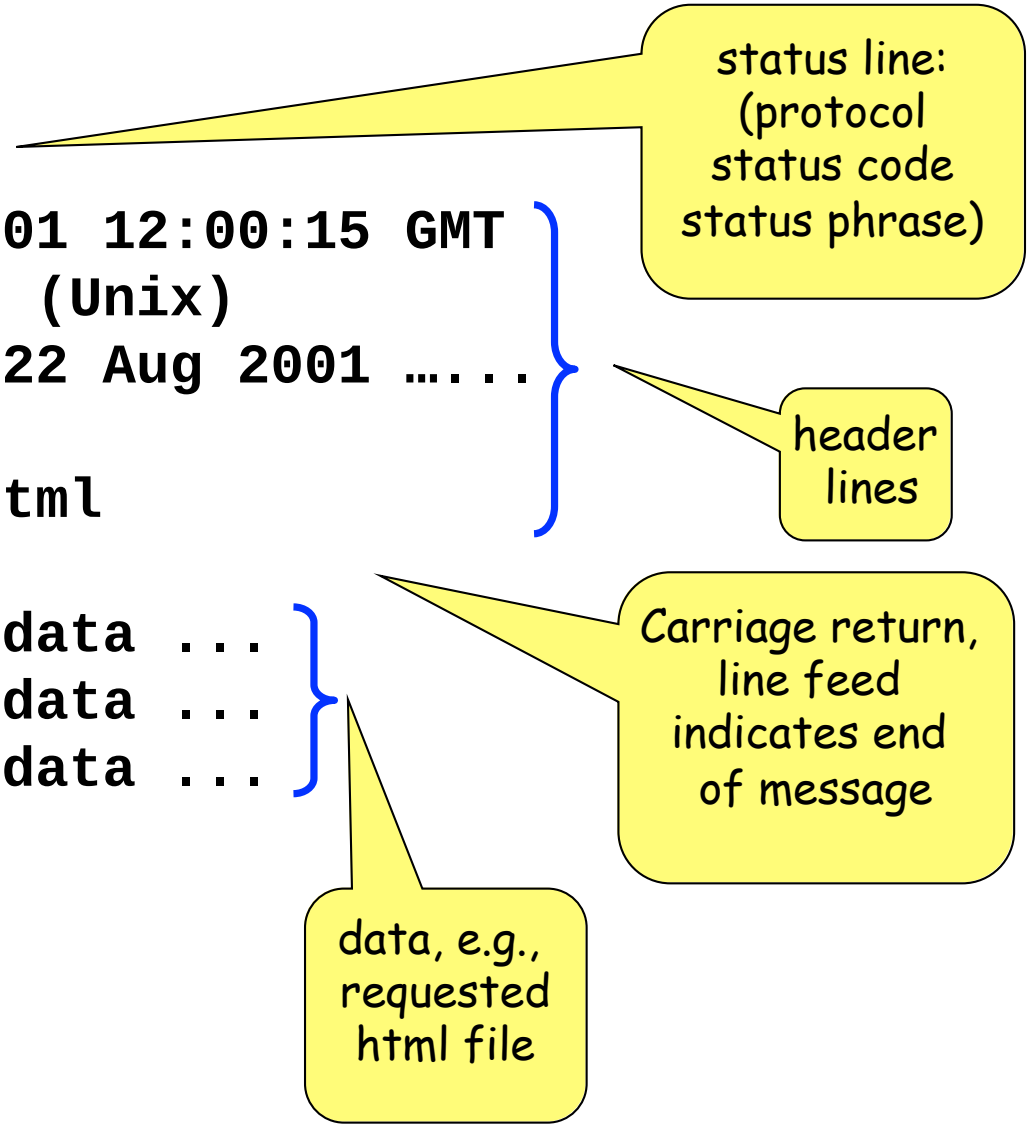
HTTP response format

- Header fields

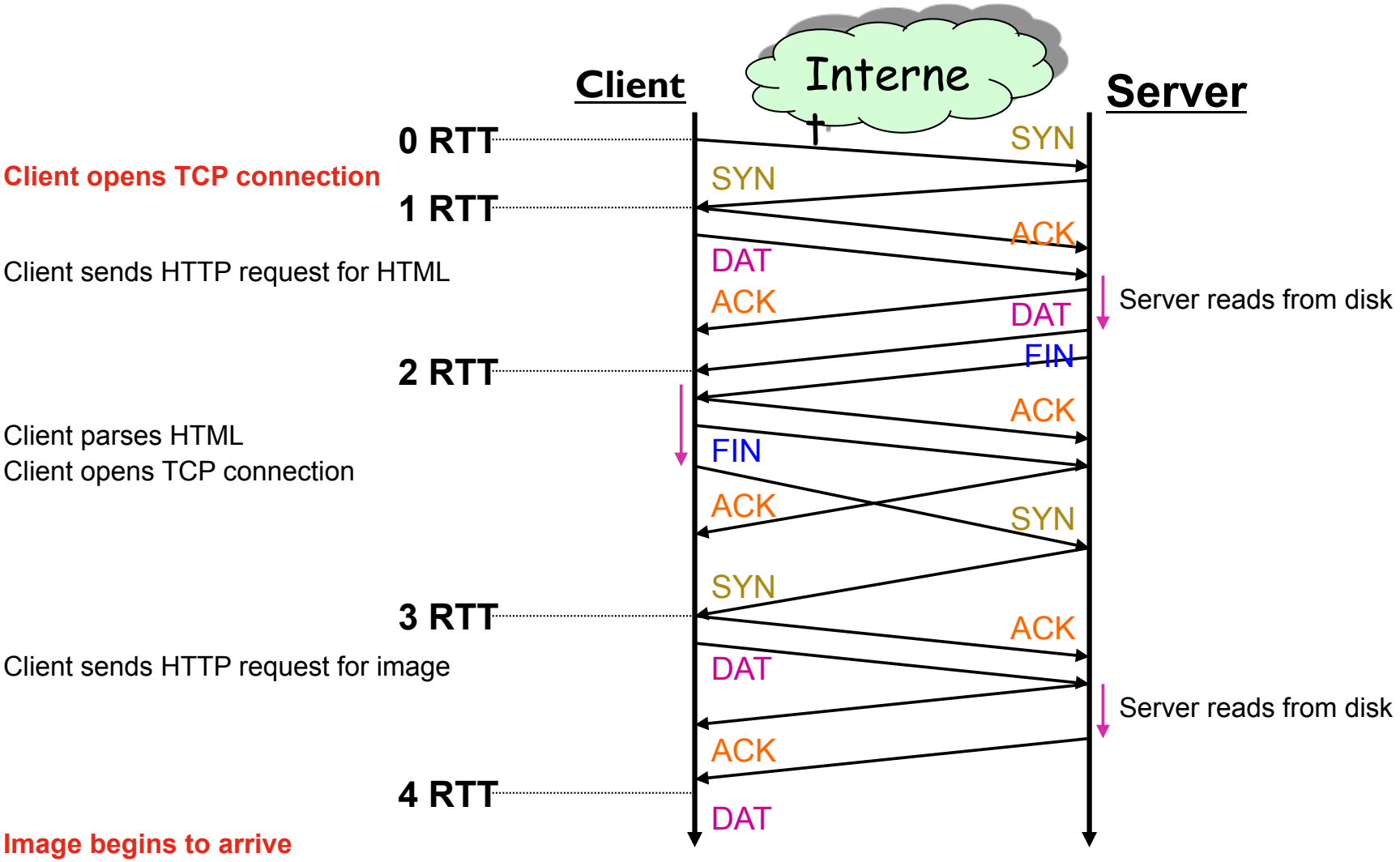
- Examples:
  - Content-Type: text/html
  - Content-Length: 4028
  - Language: en;
  - Last-modified: 17 May 2004

HTTP response example

```
HTTP/1.0 200 OK
Date: Thu, 25 Aug 2001 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Aug 2001 …...
Content-Length: 6821
Content-Type: text/html

data data data data data ...
data data data data data ...
data data data data data ...
```



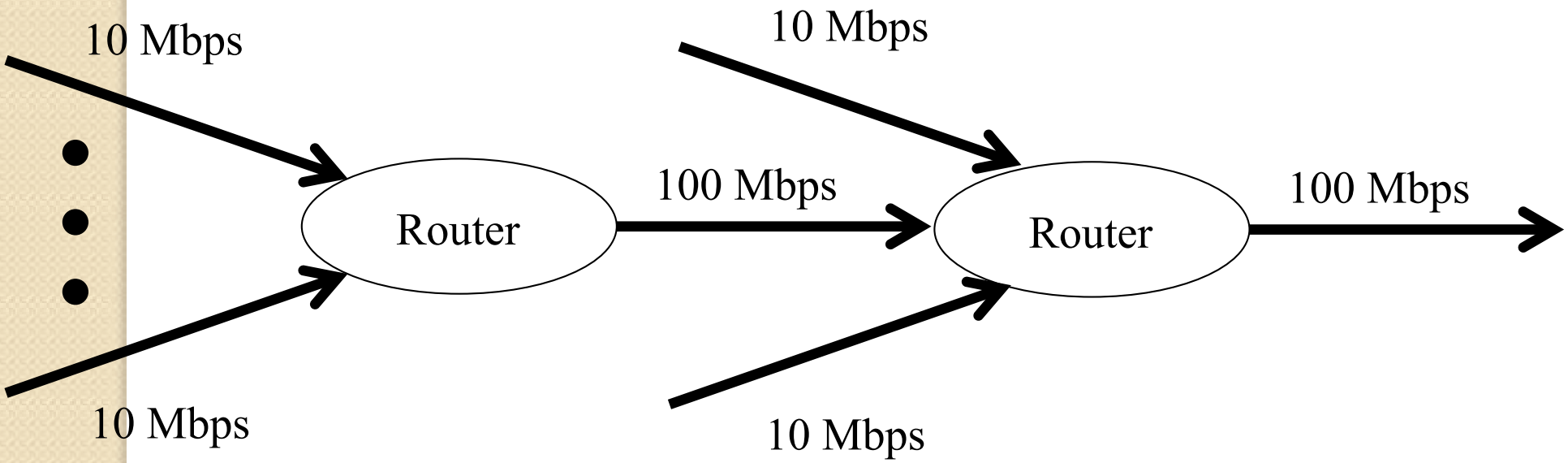
Web Page with Single Image



TCP

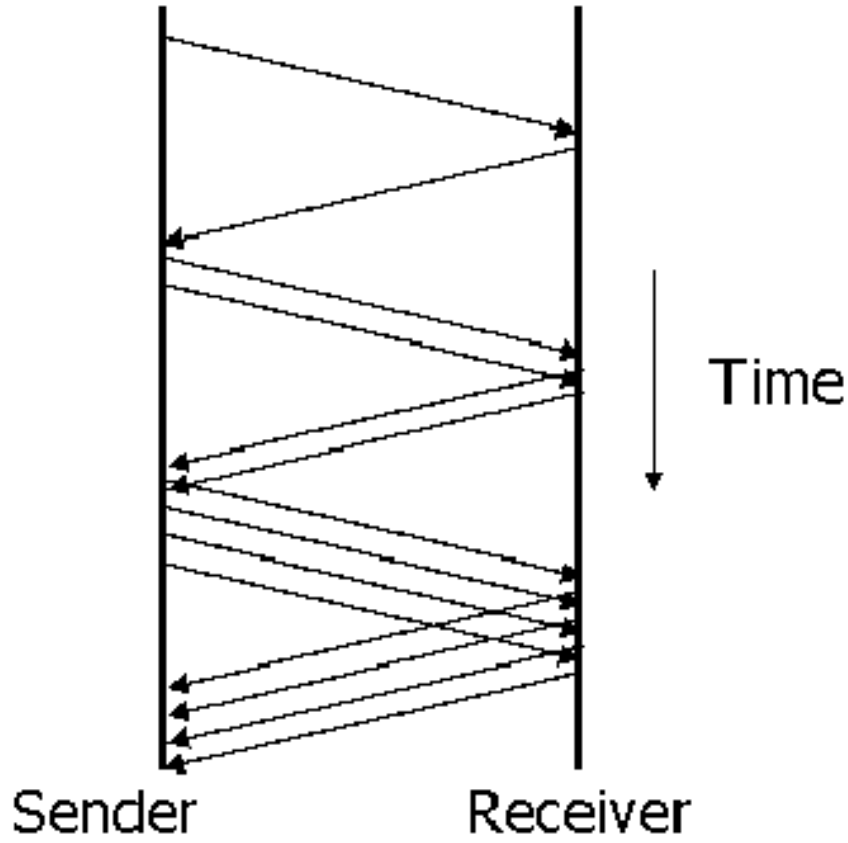
- HTTP rides on top of TCP transport service
- TCP provides: reliable, bi-directional, in-order byte stream
- Reliable
  - Prevent packet loss due to congestion
    - Overflow network queues
  - Send at rate at which network can forward packets
  - How to determine sending rate?
    - Dynamic
    - Depends on overall network condition

Network Congestion



TCP Slow Start

- Determine sending rate by proving network
- Increase sending rate until a packet is dropped
- Double the number of unacknowledged packets (window size) for every new acknowledgement
- After a drop,
  - Reset window size to 1 packet
  - Cut maximum window size in half
  - Grow window with additive increase



HTTP 1.1: Persistent Connections

- Reuse connection for multiple requests

GET index.html

Connection: keep-alive

… multiple HTTP requests …

Get banner.gif

Connection: close

Connection length

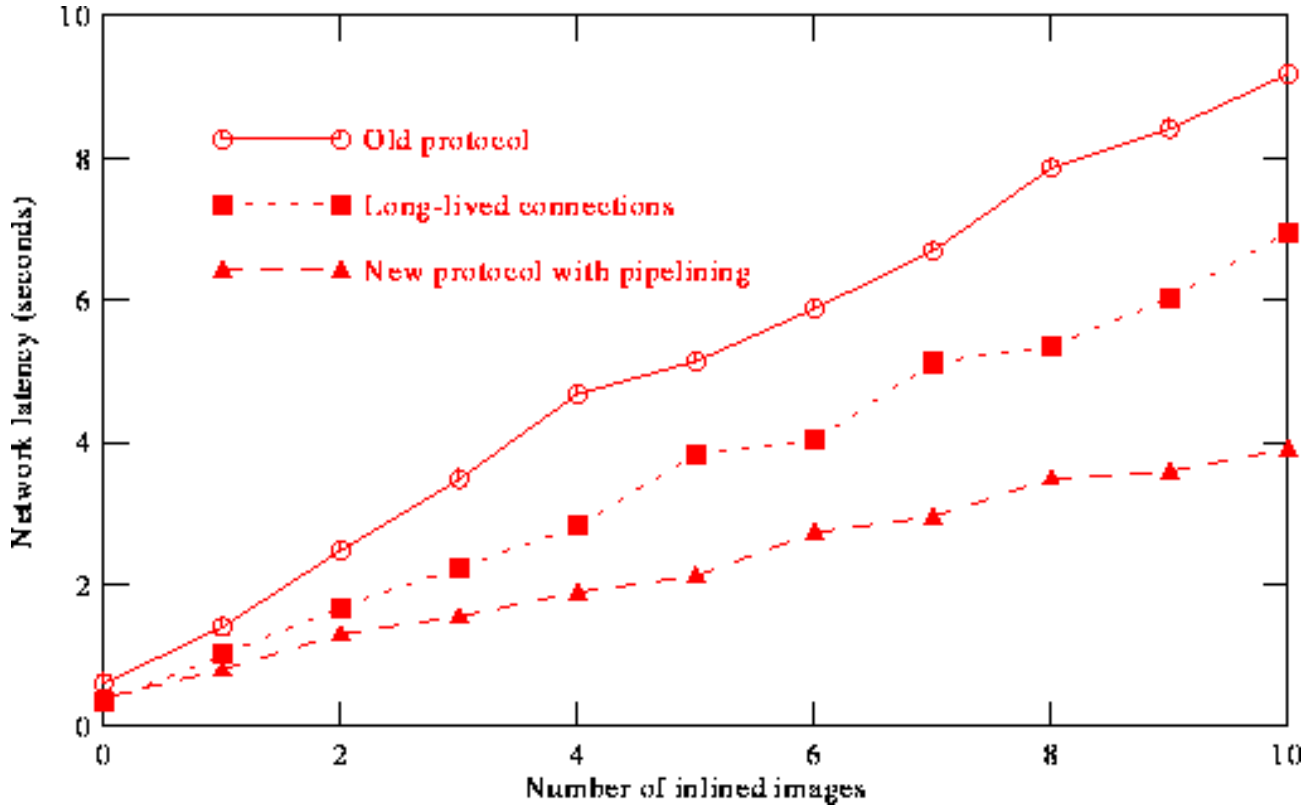
- When does the data end?
  - Without persistent connections, when connection closes.
  - With persistent connections, reply header includes content length.

HTTP1.0 vs. HTTP1.1

- Venkata N. Padmanabhan and Jeffrey C. Mogul, "*Improving HTTP Latency*.," in Proceedings of the The 2nd International WWW Conference, Chicago, IL, USA, Oct 1994
- Compared download latency for HTML documents with varying number of embedded images

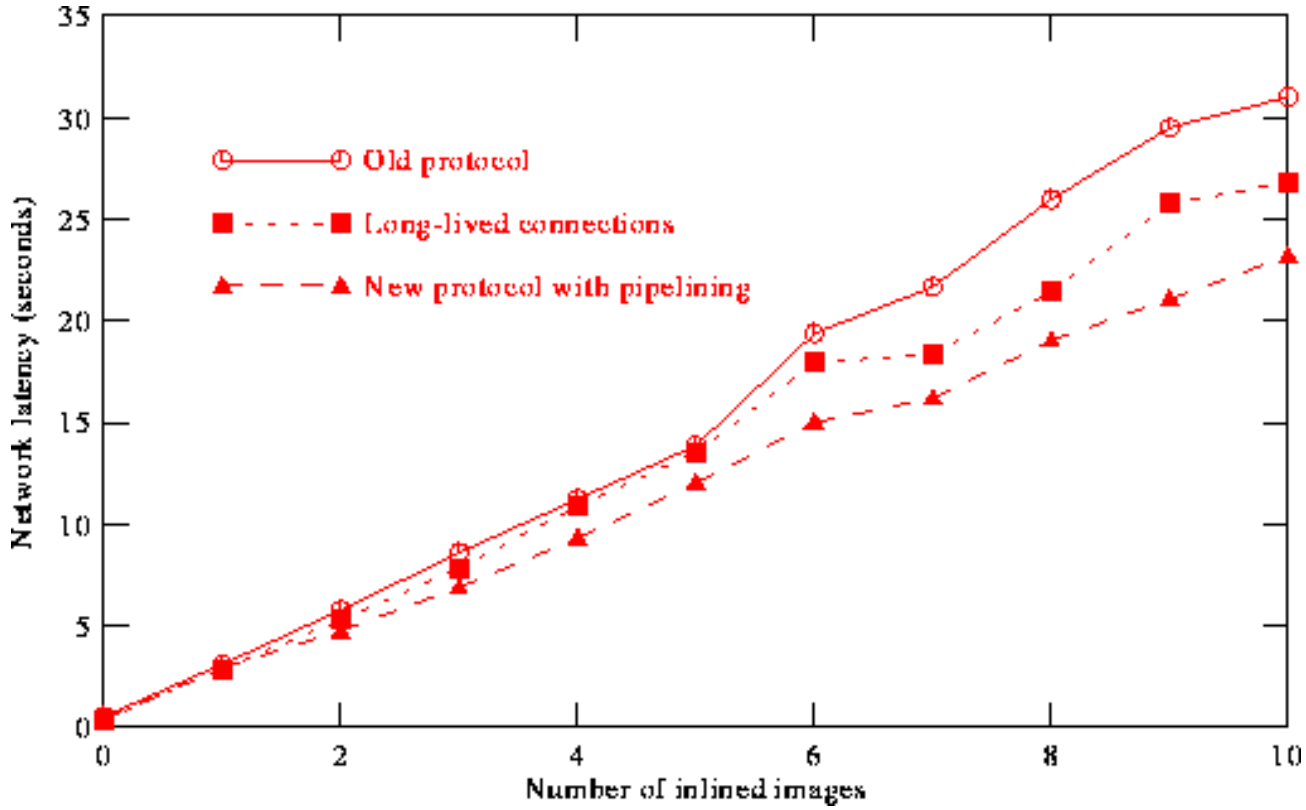
HTTP1.0 vs. HTTP1.1

- Image size 2544 bytes



HTTP1.0 vs. HTTP1.1

- Image size 45566 bytes



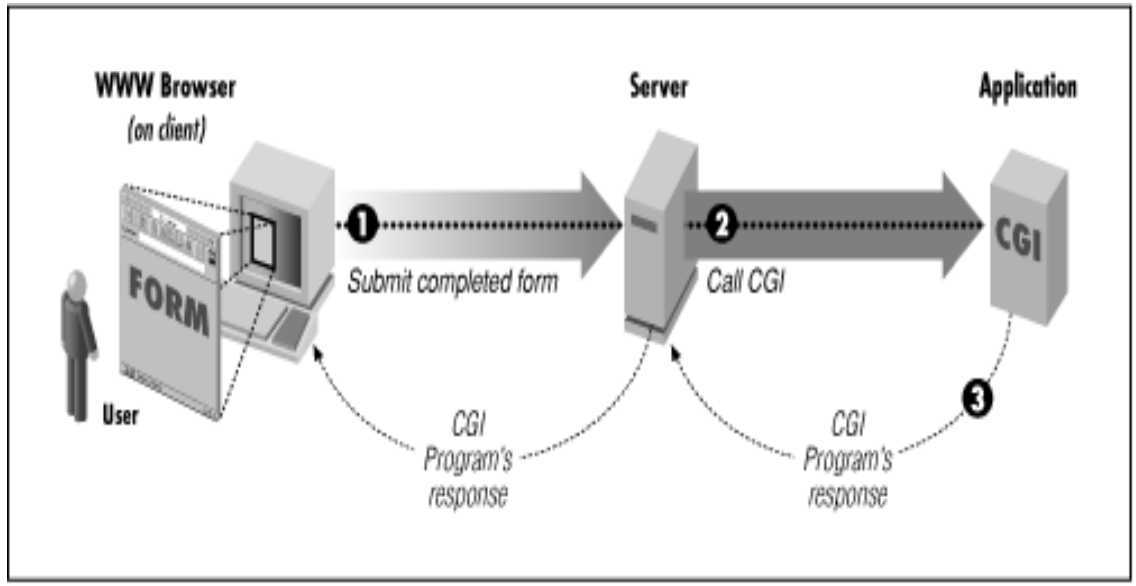
Persistent Connection Performance

- Benefits greatest for small objects.
- Serialized requests do not improve response time.
- Pipelining requests can result in large win.
- Server resource utilization reduced due to fewer connection establishments and fewer active connections.
- TCP behavior improved.
  - Longer connections help adaptation to available bandwidth.
  - Larger congestion window improves loss recovery.

CGI

Common Gateway Interface

CGI Architecture



Objectives

- Standardizes execution of server-side applications/handlers 
- Specifies: 

Web Server-Handler Interaction

- Handler Input 
  - Meta-variables 
  - Stdin (message body) 
- Handler Output
  - Stdout

Example: env.sh

```
#!/bin/bash

echo Content-type: text/plain
echo ""

echo "SERVER_PORT is set to $SERVER_PORT"
echo "The USER_AGENT is $HTTP_USER_AGENT"

echo ------------------

echo "Values for all environment variables:"

/usr/bin/printenv

echo ------------------
```

Client-Handler Interaction

- Clients request the execution of a handler program by means of a:
  - A request method (e.g., GET, POST)
  - Universal Resource Identifier (URI)
    - Identifies handler
  - Extra arguments

URI Syntax

<protocol>://<host><port>/<path-info><script>"?"<query-string>

http://finance.yahoo.com/add?op1=10&op2=20

<protocol>	http
<host>	finance.yahoo.com
<port>	80
<path-info>	null
<script>	add
<query-string>	op1=10, op2=20

Query String Encoding

- RFC 2396

- Why encode? 
  - Can think of a CGI script as a function, send arguments by specifying name/value pairs.
  - Way to pack and unpack multiple arguments into a single string

Encoding Rules

- All arguments
  - Single string of ampersand (&) separated name=value pairs

    name_1=value_1&name_2=value_2&...

- Spaces
  - Replaced by a plus (+) sign

- Other characters (ie, =, &, +)
  - Replaced by a percent sign (%) followed by the two-digit hexadecimal ASCII equivalent

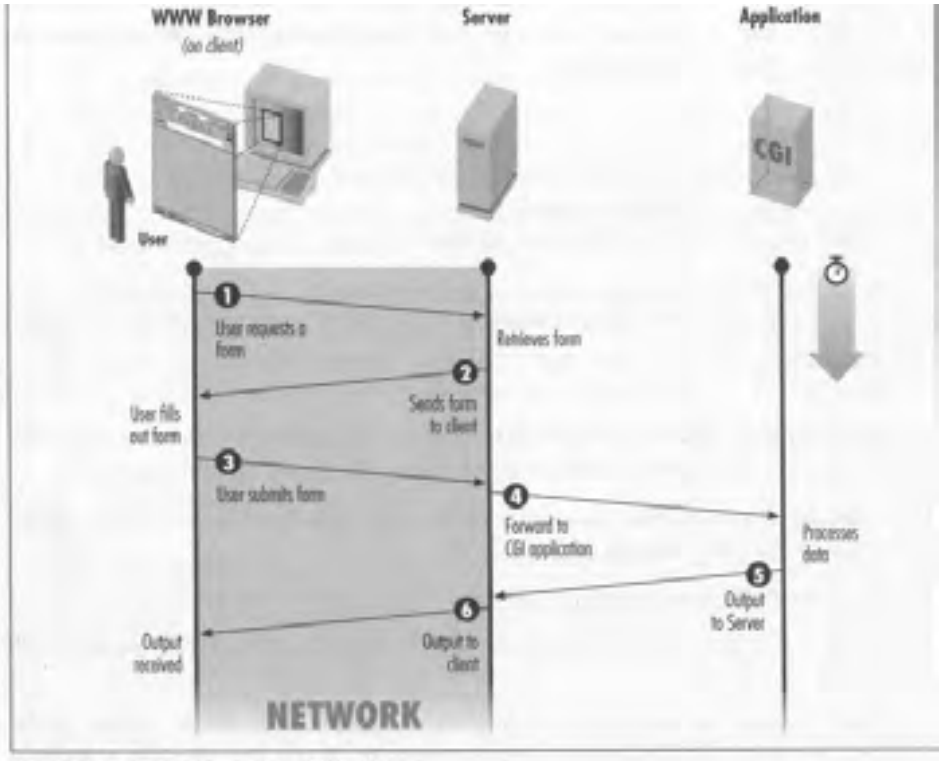
Method

- GET
  - Arguments appear in the URL after the ?
  - Can be expressed as a URL
  - Limited amount of information can be passed this way
  - URL may have a length restriction on the server
  - Arguments appear on server log
- POST
  - Arguments are sent in the HTTP message body
  - Cannot be expressed as URL
  - Arbitrarily long form data can be communicated (some browsers may have limits (i.e. 7KB)).
  - Arguments usually does not appear in server logs.

Example: add.sh

```
#!/bin/bash

echo "Content-type: text/plain"
echo ""


IFS="&"
set -- $QUERY_STRING

array=($@)

declare -A params

for i in "${array[@]}";
  do IFS="=" ;
  set -- $i;
  params[$1]=$2;
done

mysum = 0
let mysum+=${params["op1"]}
let mysum+=${params["op2"]}

echo ${params["op1"]} + ${params["op2"]} = $mysum
```

Forms and CGI



- Specify request method in "method" attribute
- Automatically encodes all **named** field values

Example: form.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>form</title>
    <style>
      input { display: block; }
      form { margin: 10px; }
    </style>

  </head>
  <body>
    <h1>GET vs POST</h1>
    <form action="cgi-bin/stdin.sh" method="get">
      Param 1 <input type="text" name="param1" />
      Param 2 <input type="text" name="param2" />
      <input type="submit" value="GET"/>
    </form>

    <form action="cgi-bin/stdin.sh" method="post">
      Param 1 <input type="text" name="param1" />
      Param 2 <input type="text" name="param2" />
      <input type="submit" value="POST"/>
    </form>

  </body>
</html>
```

Example: stdin.sh

```
#!/bin/bash

echo Content-type: text/plain
echo ""


read arguments

echo "STDIN = " $arguments
echo "QUERY_STRING = " $QUERY_STRING
```

Apache CGI

- Apache directories
  - cgi-bin cgi files
  - htdocs .html, .gif, .jpeg, .css, .js
  - conf configuration files

- httpd.conf
  - Main Apache configuration file
  - CGI are disabled by default
  - To enable CGI:
    - Add "ExecCGI" to the "Options" directive

    Options Indexes FollowSymLinks ExecCGI